LAB 4 – NN TOOLBOX

CE889 – ARTIFICIAL NEURAL NETWORKS

Introduction to Neural Networks Toolbox in MATLAB

Head of Module : Professor Hani Hagras

Lab Assistant : Aysenur Bilgin (abilgin@essex.ac.uk)

Lab Assistant : Andrew Starkey(astark@essex.ac.uk

Introduction

- Neural Network Toolbox™ provides functions and apps for modelling complex nonlinear systems that are not easily modelled with a closed-form equation.
- NN Toolbox supports supervised learning with feed forward, radial basis, and dynamic networks. It also supports unsupervised learning with self-organizing maps and competitive layers.
- With the toolbox you can design, train, visualize, and simulate neural networks.
- For more information on Neural Networks, go to the Help menu in MATLAB.

Neural Network Design Process

- The work flow for the neural network design process has seven primary steps:

1. Collect data
2. Create the network
3. Configure the network
4. Initialize the weights and biases
5. Train the network
6. Validate the network
7. Use the network

Neural Network Design Process

- The Neural Network Toolbox software uses the network object to store all of the information that defines a neural network.

- After a neural network has been created, it needs to be configured and then trained.

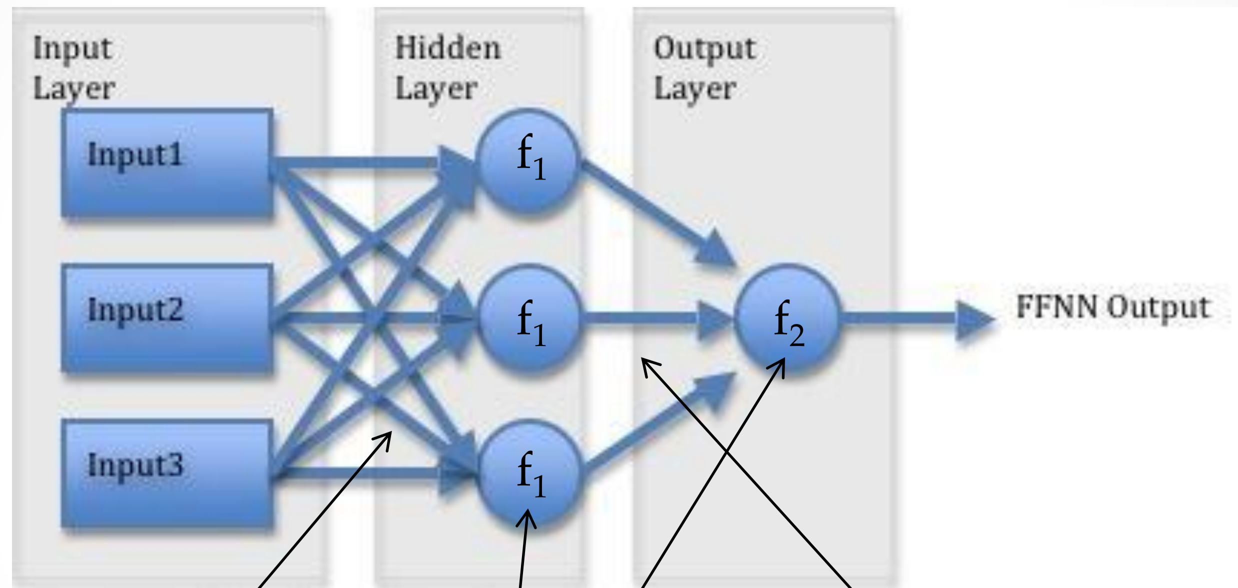
- Configuration involves arranging the network so that it is compatible with the problem you want to solve, as defined by sample data.

- After the network has been configured, the adjustable network parameters (called weights and biases) need to be tuned, so that the network performance is optimized. This tuning process is referred to as training the network.

- Configuration and training require that the network be provided with example data.

Feed Forward Neural Networks

- The architecture of a multi-layer feed forward neural network is as follows:



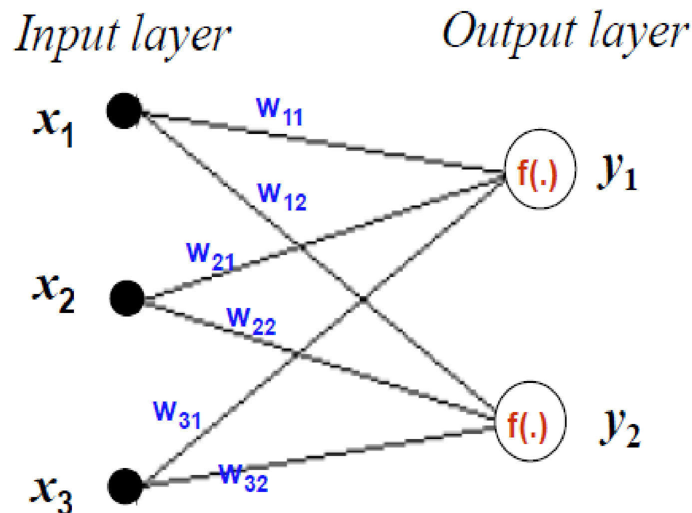
Hidden Layer weights

Output Layer weights

Activation (transfer) functions

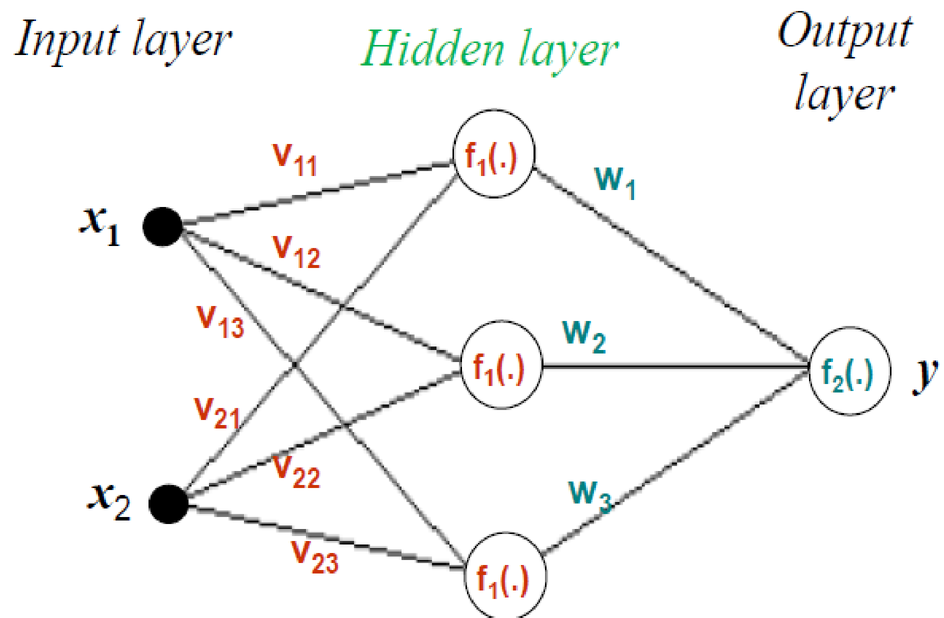
Feed Forward Neural Networks

One layers FFNN
(3 inputs & 2 outputs)



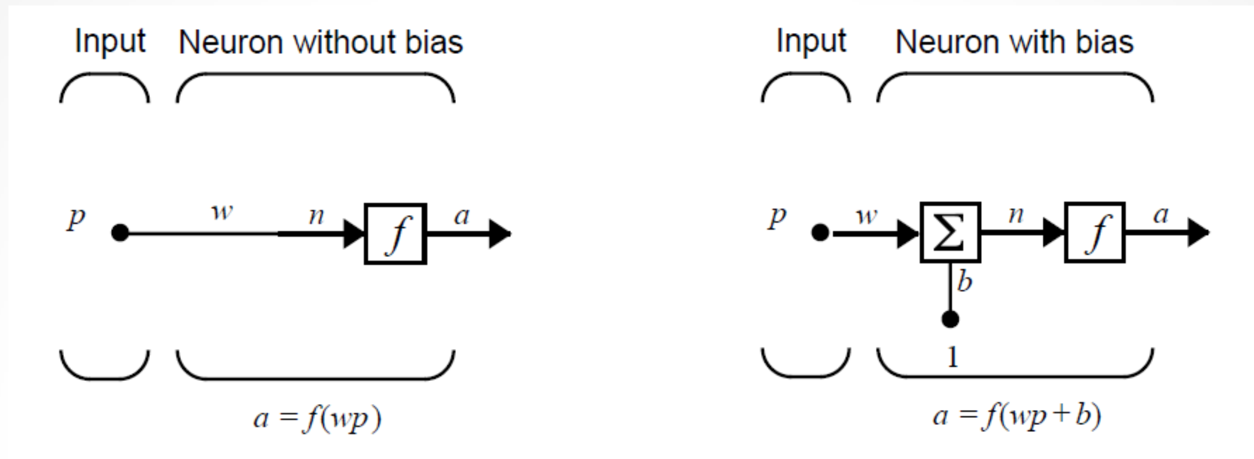
$$y_k = f\left(\sum_{i=1}^{3} w_{ik} x_i\right), \text{ for } k = 1{:}2$$

Two layers FFNN
(2 inputs & 1 output)



$$y = \sum_{k=1}^{3} f_2\left(w_k . f_1\left(\sum_{i=1}^{2} v_{ik} . x_i\right)\right)$$

Simple Neuron



- In the neuron model, *f* is a transfer function, typically a step function or a sigmoid function, which takes the argument *n* and produces the output *a*.

- Note that *w* and *b* are both adjustable scalar parameters of the neuron. The central idea of neural networks is that such parameters can be adjusted so that the network exhibits some desired or interesting behaviour.

- Thus, we can train the network to do a particular job by adjusting the weight or bias parameters, or perhaps the network itself will adjust these parameters to achieve some desired end.

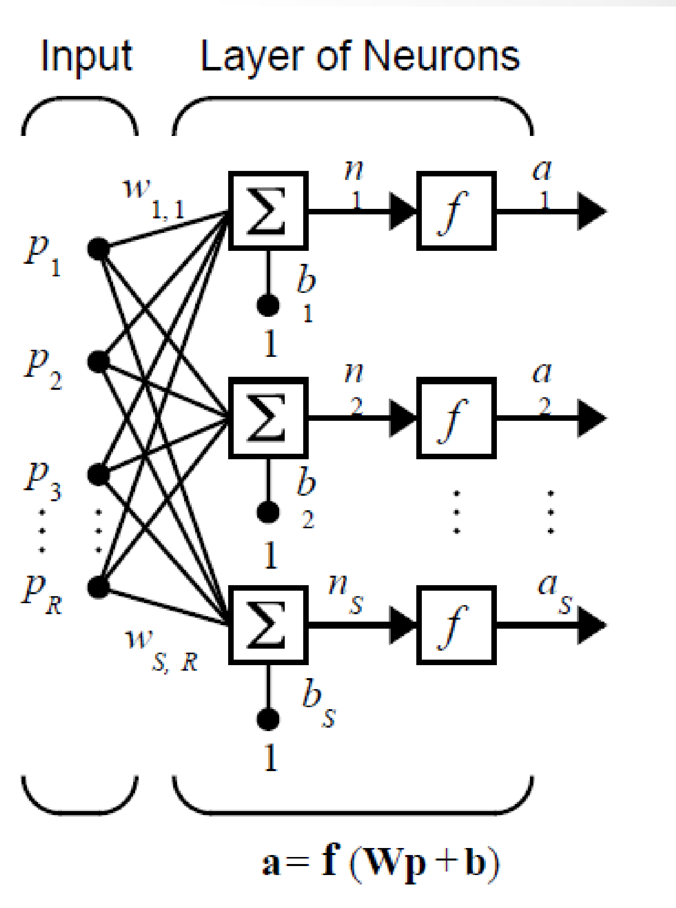
Network Architecture

- A one-layer network with *R* input elements and *S* neurons.

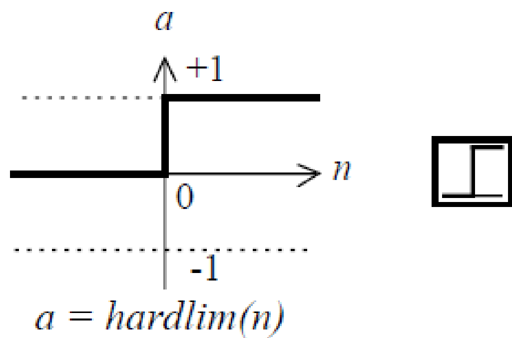
- In this network, each element of the input vector **p** is connected to each neuron input through the weight matrix **W**.

- The *i*th neuron has a summer that gathers its weighted inputs and bias to form its own scalar output *n(i)*.

- The various *n(i)* taken together form an *S*-element net input vector **n**. Finally, the neuron layer outputs form a column vector **a**.



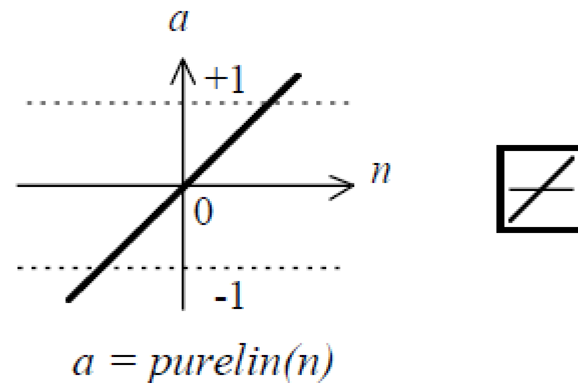
Transfer Functions



Hard-Limit Transfer Function

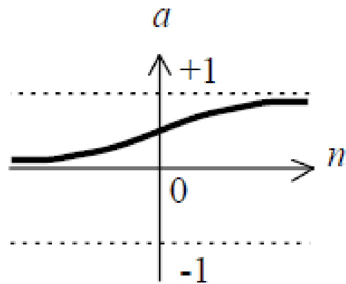
- The hard-limit transfer function limits the output of the neuron to
  - either 0, if the net input argument *n* is less than 0;
  - or 1, if *n* is greater than or equal to 0.

- Neurons of this type are used as linear approximators.



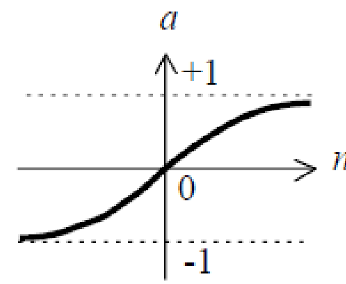
Linear Transfer Function

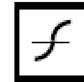
Transfer Functions



$a = logsig(n)$

Log-Sigmoid Transfer Function





$a = tansig(n)$

Tan-Sigmoid Transfer Function

- The log-sigmoid transfer function takes the input, which may have any value between plus and minus infinity, and squashes the output into the range 0 to 1.
- This transfer function is commonly used in backpropagation networks, in part because it is differentiable.
- The symbol in the square to the right of each transfer function graph shown above represents the associated transfer function. These icons will replace the general *f* in the boxes of network diagrams to show the particular transfer function being used.

Training Functions

- Once the network weights and biases have been initialized, the network is ready for training.
- The training process requires a set of examples of proper network behavior - network inputs p and target outputs t. During training, the weights and biases of the network are iteratively adjusted to minimize the network performance function.

Function	Algorithm
trainlm	Levenberg-Marquardt
trainbr	Bayesian Regularization
traingdx	Variable Learning Rate Gradient Descent
traingd	Gradient Descent
traingdm	Gradient Descent with Momentum

Training Functions

- **traingd**: The weights and biases are updated in the direction of the negative gradient of the performance function.

- **traingdm:** The weights and biases are updated according to gradient descent with momentum.
  - Momentum allows a network to respond not only to the local gradient, but also to recent trends in the error surface.
  - Acting like a low-pass filter, momentum allows the network to ignore small features in the error surface.
  - Without momentum a network may get stuck in a shallow local minimum.
  - With momentum a network can slide through such a minimum.

- The above algorithms can train any network as long as its weight, net input, and transfer functions have derivative functions. They are variations of backpropagation algorithm.

Learning Functions

Learning Functions	
learncon	Conscience bias learning function
learngd	Gradient descent weight/bias learning function
learngdm	Grad. descent w/momentum weight/bias learning function
learnp	Perceptron weight/bias learning function

- **learngdm:** calculates the weight change dW for a given neuron from the neuron's input P and error E, the weight (or bias) W, learning rate LR, and momentum constant MC, according to gradient descent with momentum: dW = mc*dWprev + (1-mc)*lr*gW. The previous weight change dWprev is stored and read from the learning state LS.

- **learngd:** calculates the weight change dW for a given neuron from the neuron's input P and error E, and the weight (or bias) learning rate LR, according to the gradient descent dw = lr*gW.

Performance Functions

Performance Functions	
mae	Mean absolute error performance function
mse	Mean squared error performance function
msereg	Mean squared error w/reg performance function (weight sum of two factors: the mean squared error and the mean squared weight and bias values)
sse	Sum squared error performance function

- The default performance function for feed forward networks is mean square error mse - the average squared error between the network outputs a and the target outputs t.

Neural Network GUI in MATLAB

- A simple and user-friendly graphical user interface has been added to the Neural networks toolbox.

- This interface allows you to:

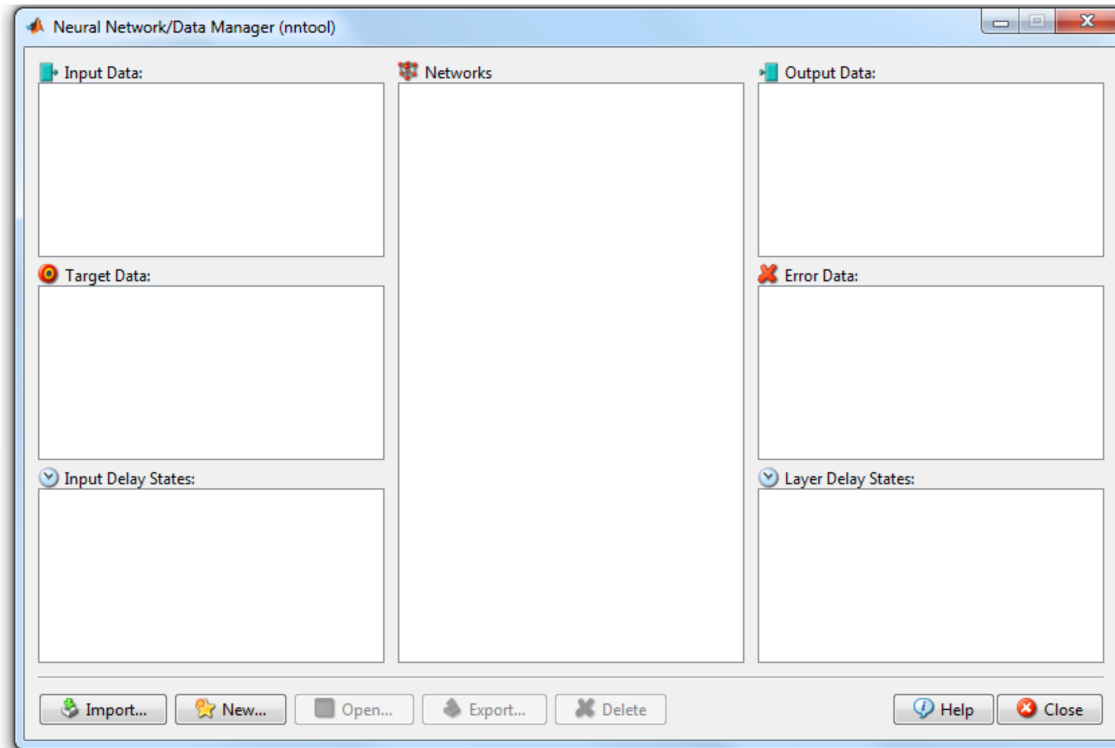
  - Create networks
  - Enter data into the GUI
  - Initialize, train, and simulate networks
  - Export the training results from the GUI to the command line workspace
  - Import data from the command line workspace to the GUI

- To open the **Neural Network/Data Manager** window, just type **nntool**

- This window has its own work area, separate from the more familiar command line workspace.

Neural Network GUI in MATLAB

- Once the GUI **Neural Network/Data Manager** is up and running, you can
  - create a network,
  - view it,
  - train it,
  - simulate it and export the final results to the workspace.
- Similarly, you can import data from the workspace for use in the GUI.



Example – Step 1: Data collection

- We create a network to perform the **sinusoidal function** in the following example:

- It has an input vector

```
>> p = 0:0.01:2.5;
```

- and a target vector

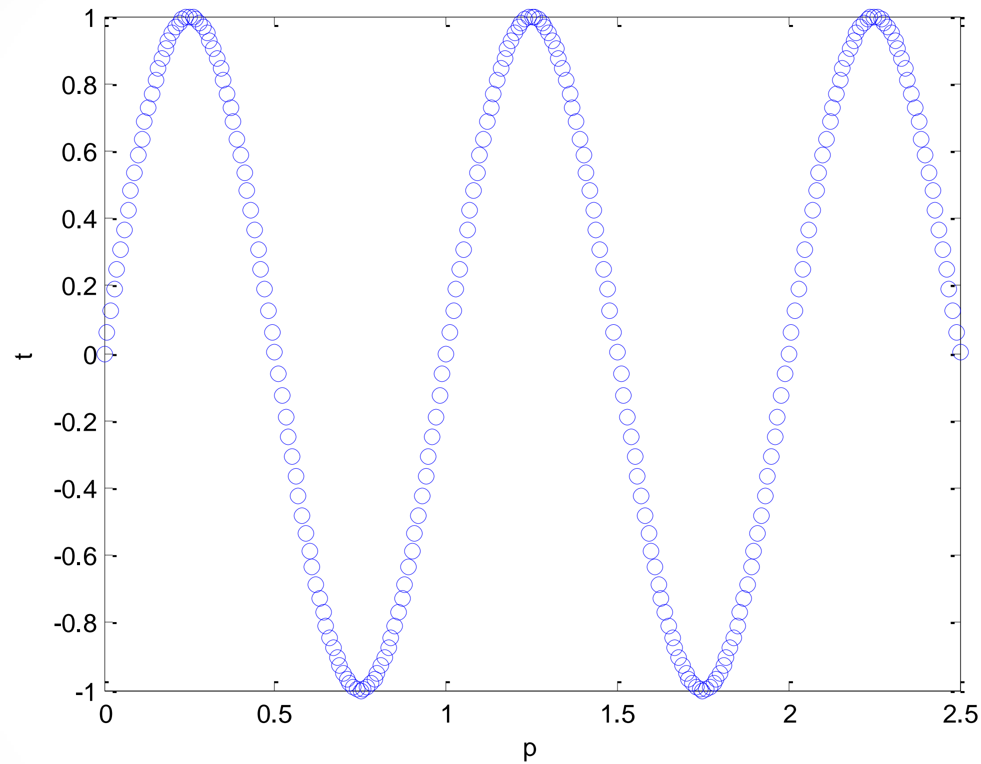
```
>> t = sin(2*pi*p);
```

- Now we can plot the relation between the two data using

```
>> figure;plot(p,t,'o')
>> xlabel('p')
>> ylabel('t')
```

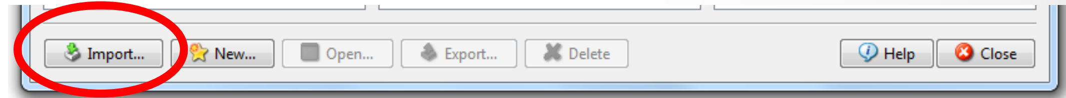
Example – Step 1: Data collection

- Training data looks as follows:

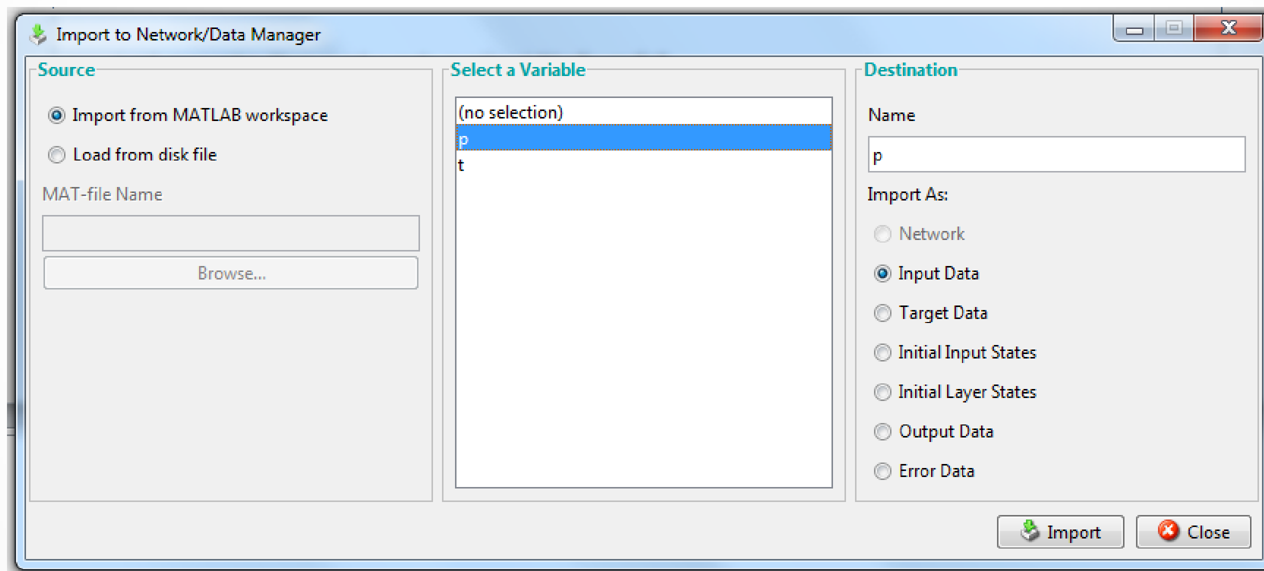


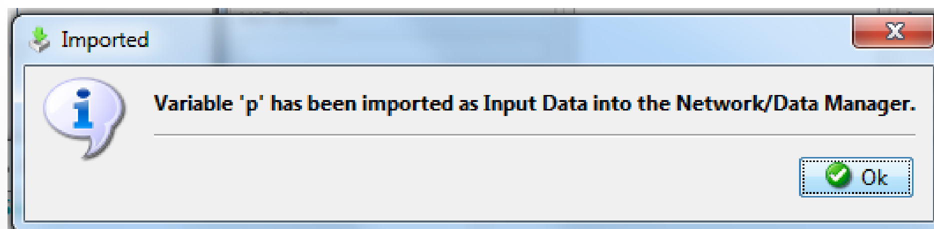
Example – Steps 2&3: Network Creation & Configuration

- Go to **Neural Network/Data Manager** window .
- Click on Import button.



- Select the data you wish to use as input data (**p** matrix in the example above) and choose **Input Data**, then click on **Import.**

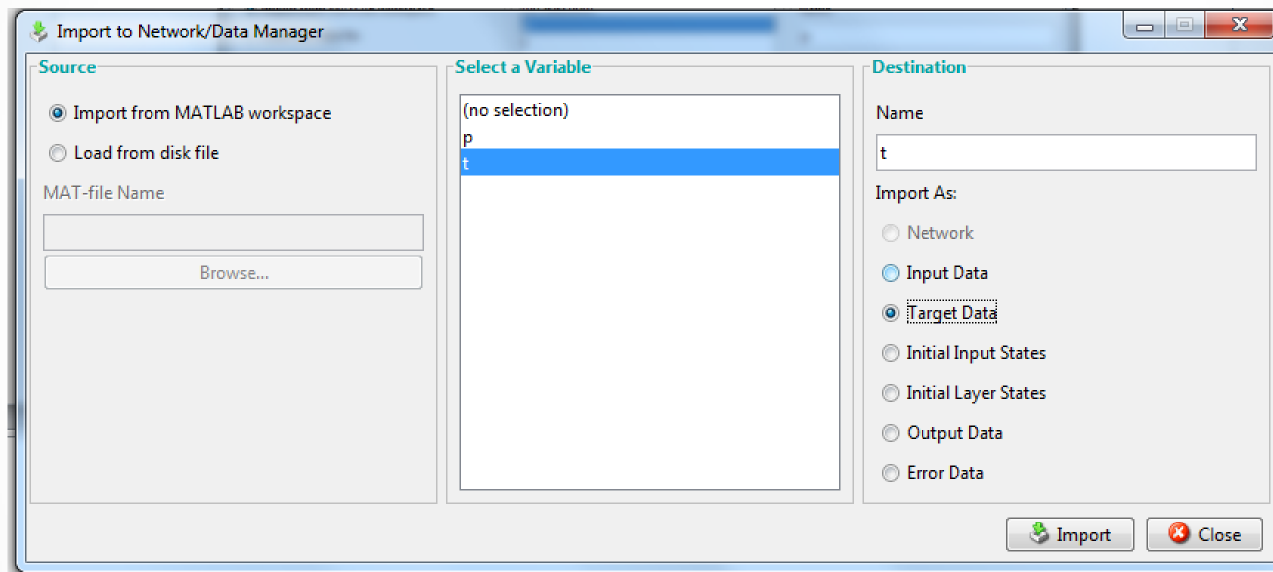


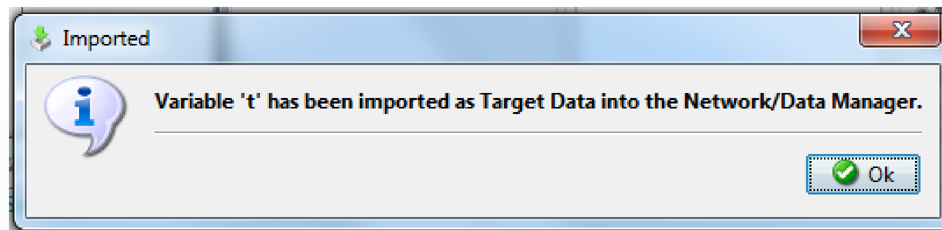


Example – Steps 2&3: Network Creation & Configuration

- Similarly, select the data you wish to use as target (**t** vector in the example above) and choose **Target Data**, then click on **Import.**

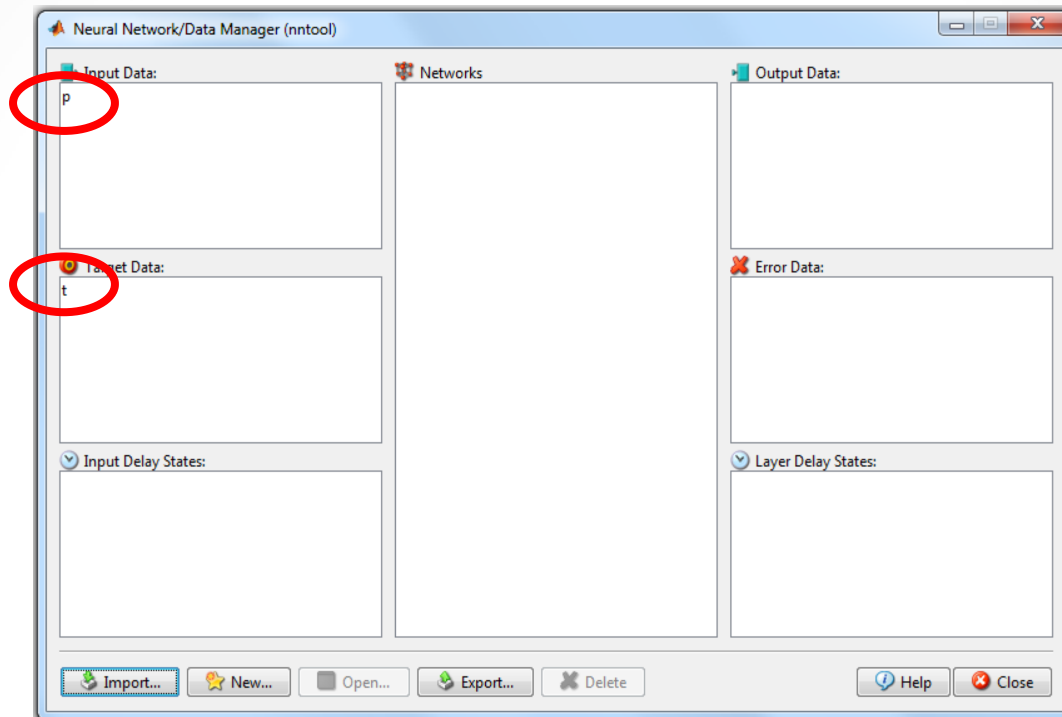
- To also include validation and testing data, import the desired data sets as **Inputs** or **Targets**.



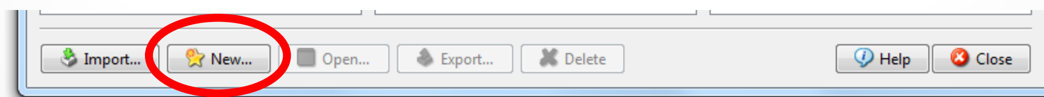


Example – Steps 2&3: Network Creation & Configuration

- In the **Neural Network/Data Manager** window, **p** shows as an input and **t** as target.

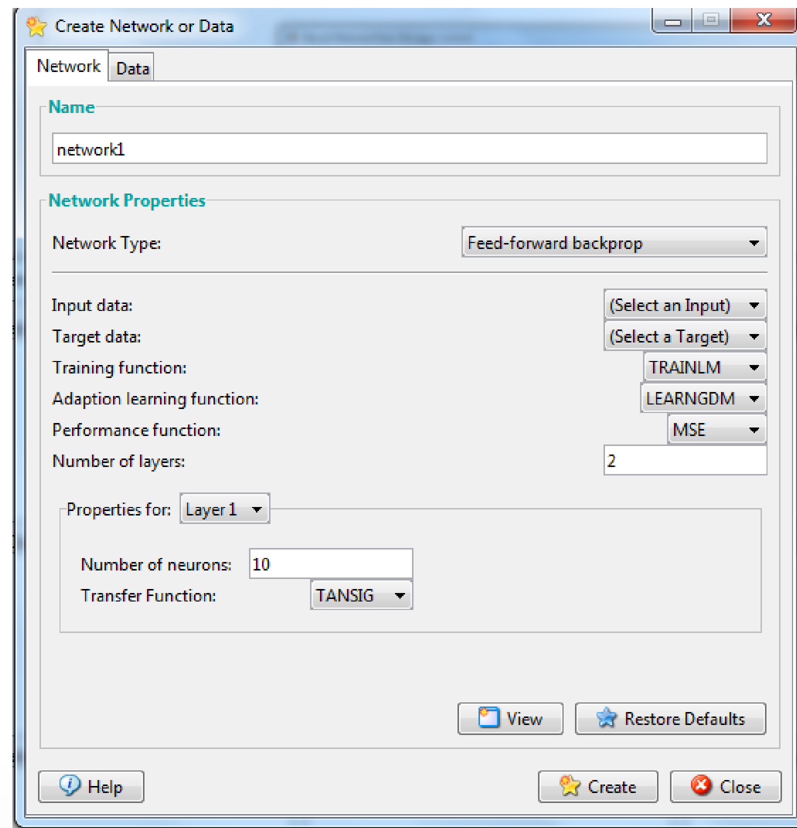


- To create a new network, click on **New**, and a **Create Network or Data** window appears.



Example – Steps 2&3: Network Creation & Configuration

- Enter **network1** under **Network Name**.
- Set the **Network Type** to **feed-forward backprop**, as that is the kind of network we want to create for this example.



Example – Steps 2&3: Network Creation & Configuration

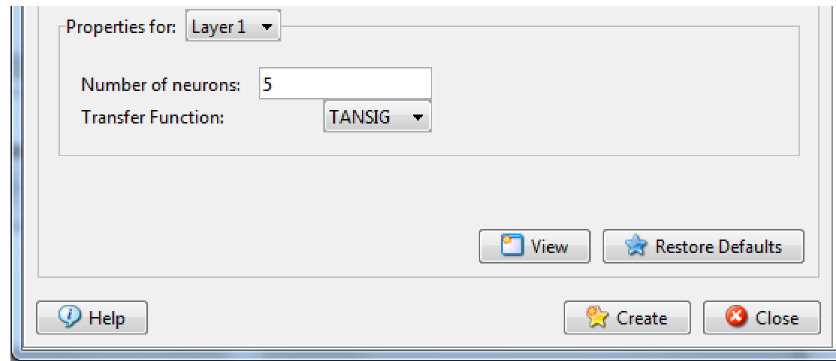
- To set the input range, click on **Input data** and select **p**.
- Similarly, click on **Target data** and select **t** .
- We want to use **trainlm** for training function, **learngdm** for learning function, and **MSE** for performance. So, set those values using the arrows for **Training function** and **Adaption learning function**, and **Performance function** respectively.

Network Type: Feed-forward backprop
Input data: p
Target data: t
Training function: TRAINLM
Adaption learning function: LEARNGDM
Performance function: MSE
Number of layers: 2

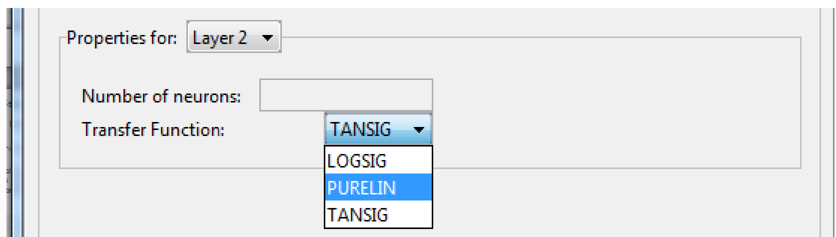
Example – Steps 2&3: Network Creation & Configuration

- For our example, we will construct a simple 2 layer FFNN with a **tansig** transfer function as the hidden layer neuron and a **purelin** transfer function as the output layer neuron.

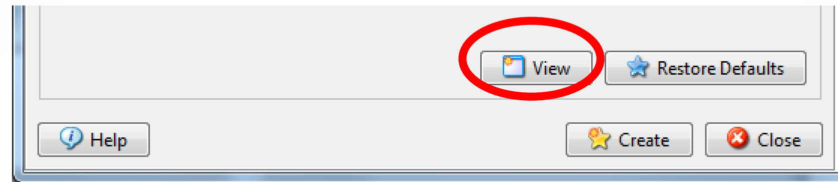
- For Layer 1, put 5 in the Number of neurons box. Layer 1 is already set to **tansig**, which is what we want.

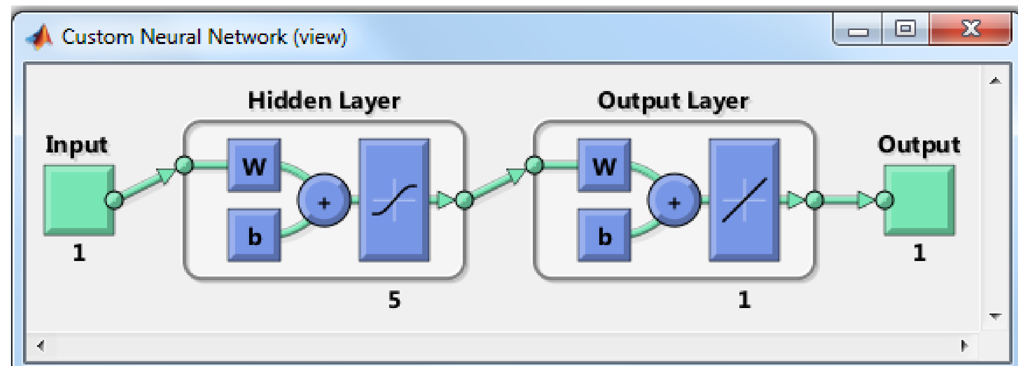


- Click the drop-down menu **Properties for**… and choose Layer 2. Next, change the transfer function to **purelin**.



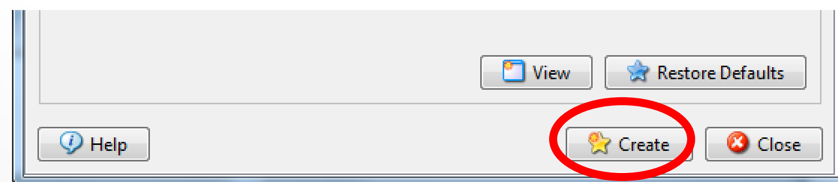
Example – Steps 2&3: Network Creation & Configuration

- Next, you might look at the network by clicking on **View**.
- 



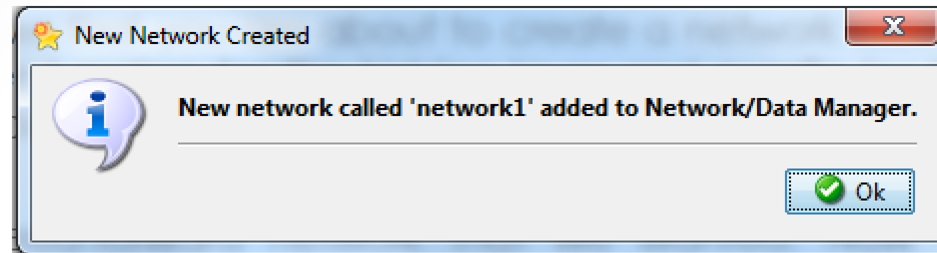
- The view shows that you are about to create a network with a single input, a **tansig** transfer function for the hidden layer and **purelin** for the output layer, and a single output.

- This is the feed-forward network that we wanted. Now, click **Create** to generate the network.

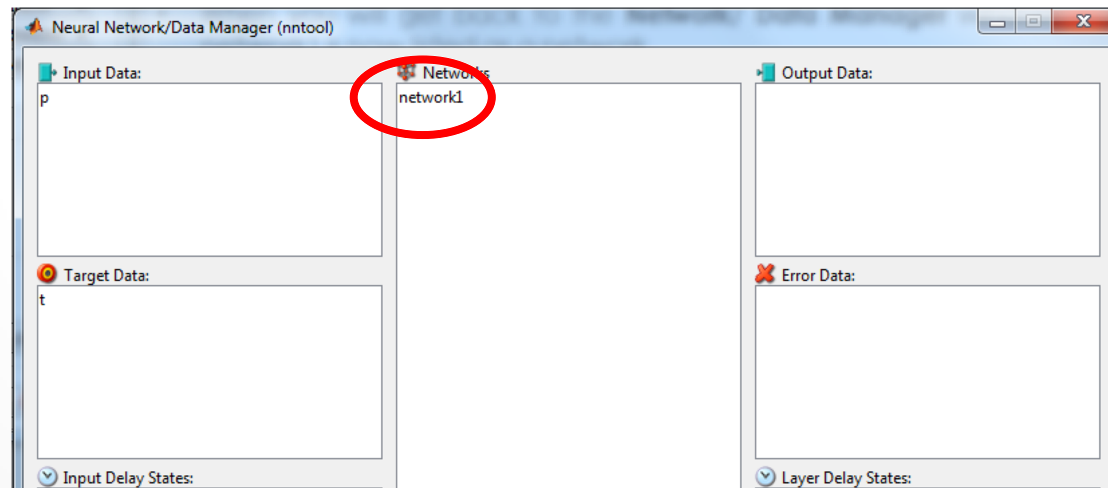


Example – Steps 2&3: Network Creation & Configuration

- You will be notified for creating a new network called ‘network1’. Click OK.



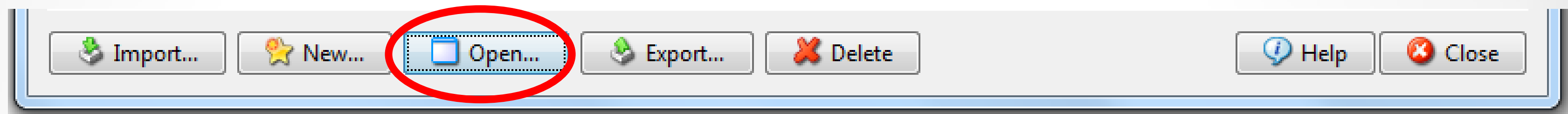
- When you get back to the **Neural Network/Data Manager** window, note that network1 is now listed as a network.



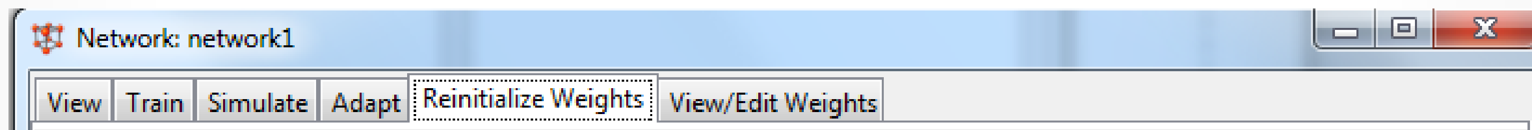
Example – Steps 4&5: Network Initialization & Training

- Before training a feed forward network, you must initialize the weights and biases.

- Click on network1 to highlight it. Then click on **Open**.

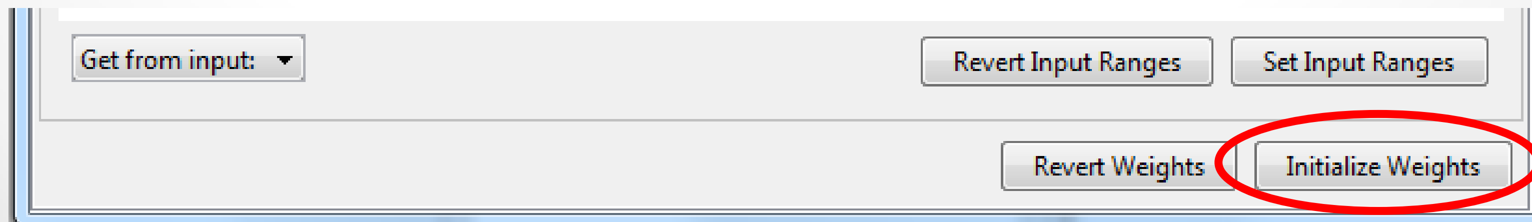


- This leads to a new window labelled **Network:network1**.



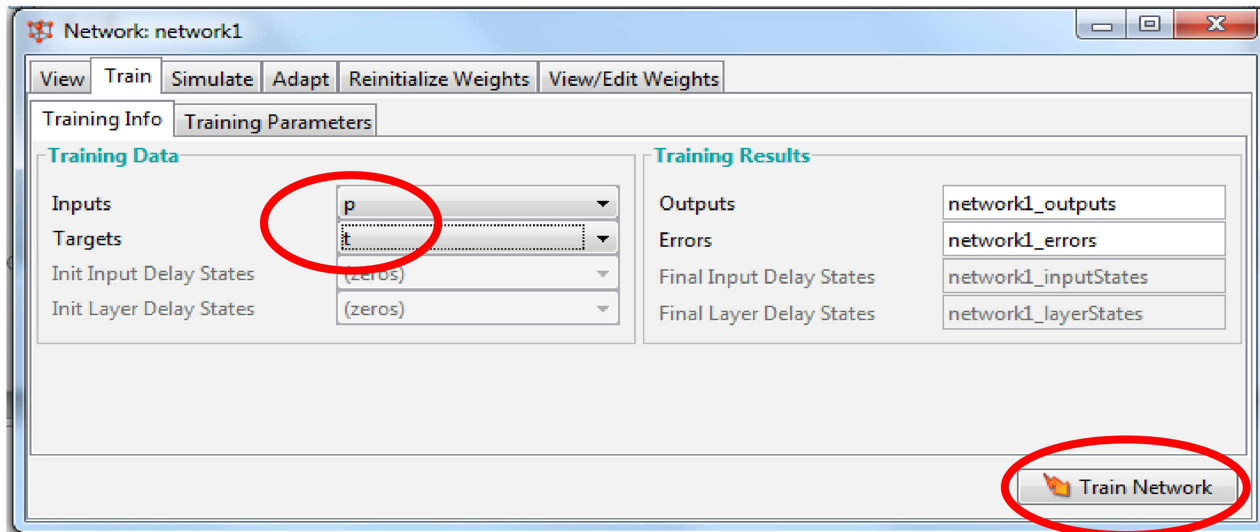
- At this point you can view the network again by clicking on the top tab **View**.

- You can also check the initialization by clicking on the top tab **Reinitialize weights** and clicking on **Initialize weights** button (especially for re-training on the same data and network).



Example – Steps 4&5: Network Initialization & Training

- For training, specify the inputs and output by clicking on the top tab **Train**, then left tab **Training Info.**
- Select **p** from the drop-down list of inputs and **t** from the drop-down list of targets. The **Network:network1** window should look like follows:



- By clicking on **Training Parameters** tab, you can adjust the number of epochs (the maximum number of times the complete data set may be used for training) and show (the time between status reports of the training function). The default is show = 25. Try setting this value to 10.
- Now click on the **Train Network** button.

Example – Steps 4&5: Network Initialization & Training

- The pop-up window labelled **Neural Network Training (nntraintool)** will appear . You can see here the training results such as; the number of **epochs**, **time**, and **performance** which are displayed during the training phase.

Neural Network Training (nntraintool)

Neural Network

Input 1 — Hidden Layer (W, b) 5 — Output Layer (W, b) 1 — Output 1

Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)

Progress

Epoch:	0	317 iterations	1000
Time:		0:00:09	
Performance:	2.04	1.33e-05	0.00
Gradient:	4.33	0.000388	1.00e-05
Mu:	0.00100	1.00e-05	1.00e+10
Validation Checks:	0	0	6

Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

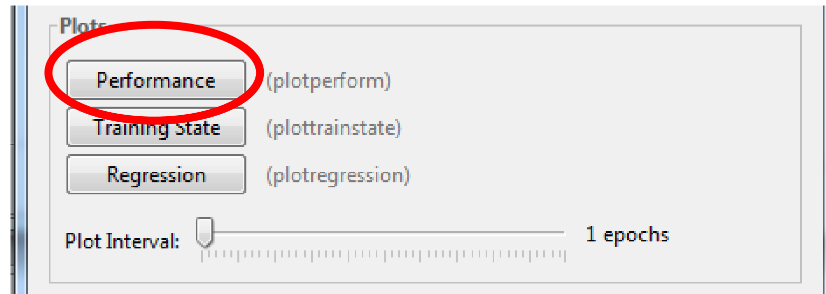
Plot Interval: 1 epochs

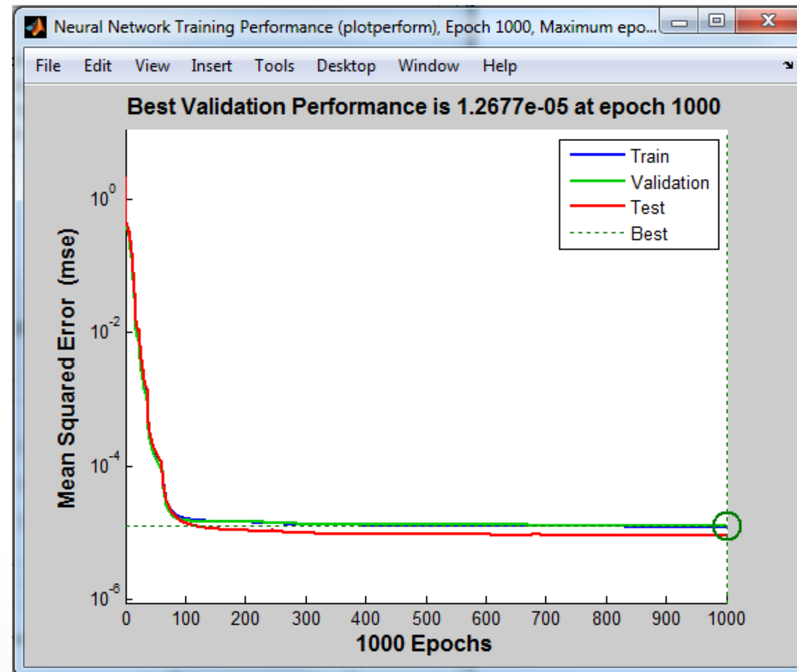
Training neural network...

Stop Training Cancel

Example – Steps 4&5: Network Initialization & Training

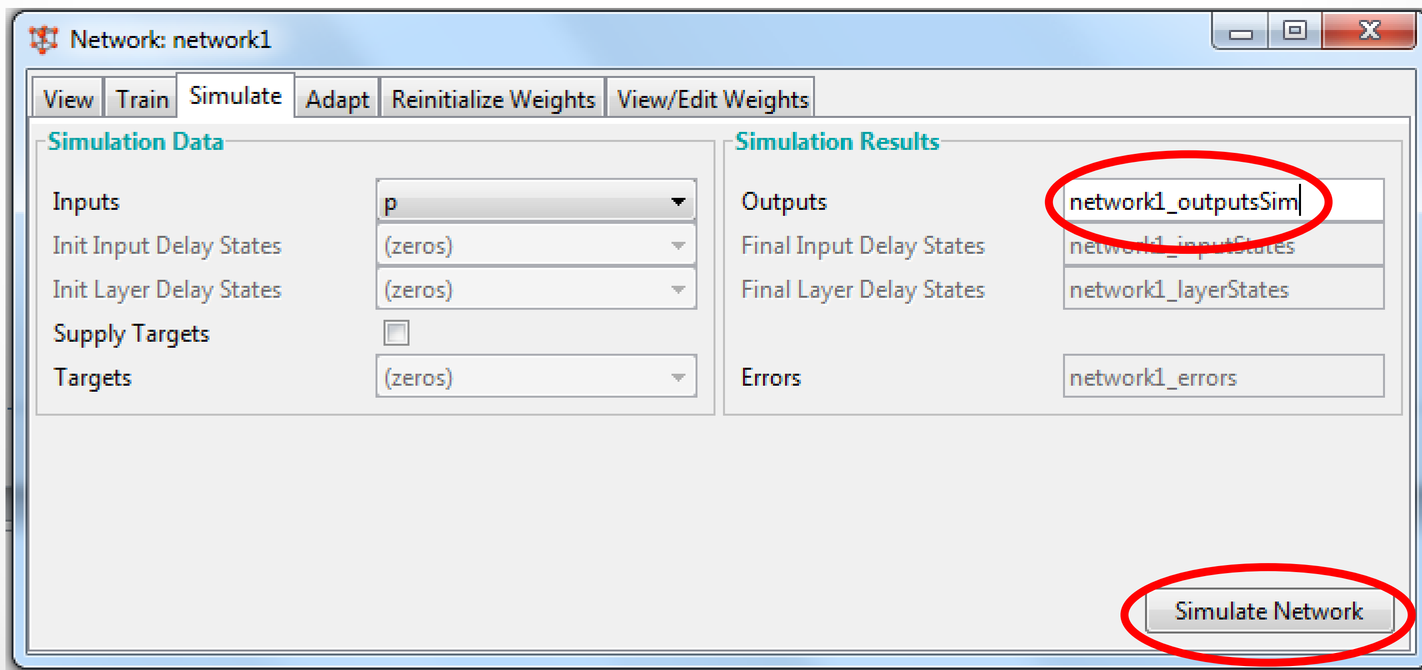
- You can see the training plot by clicking the **Performance** button.





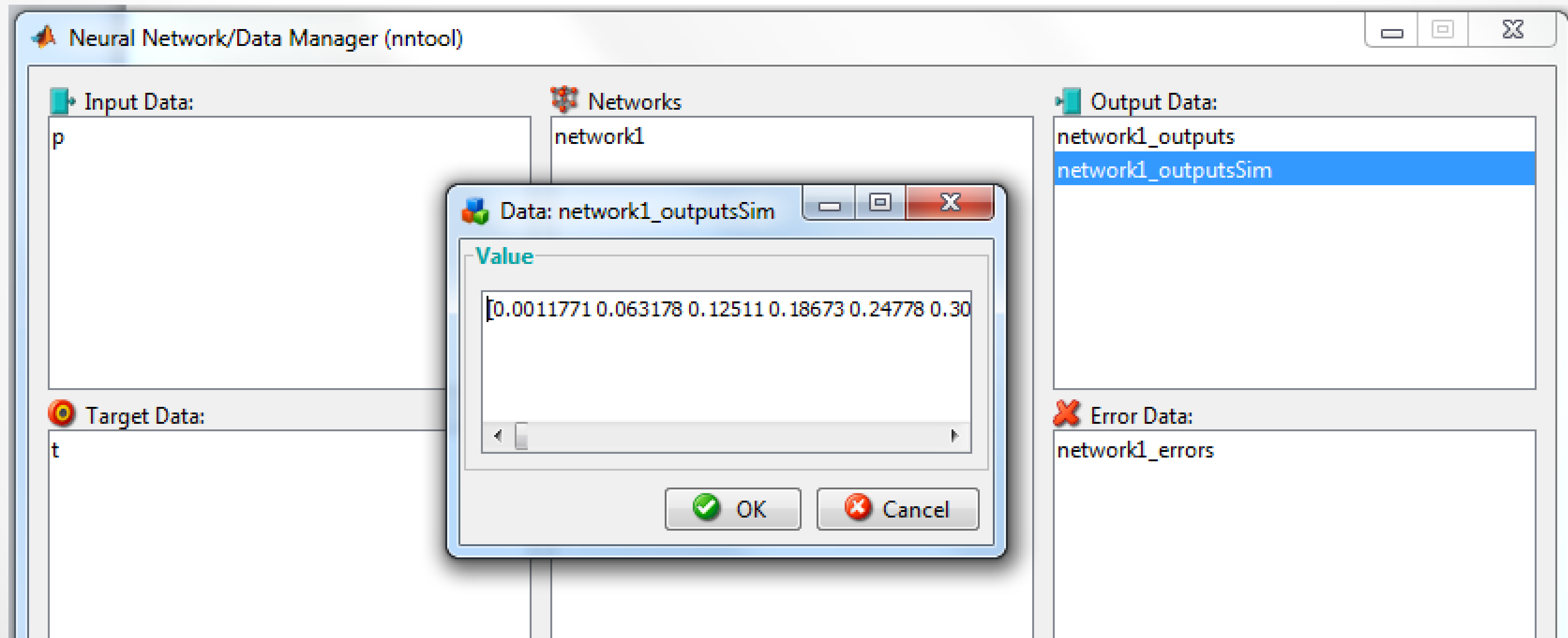
Example – Steps 6&7: Network Validation & Testing

- You can check that the trained network does indeed give approximately zero error by using the input **p** and simulating the network.
- To do this, go to the **Neural Network/Data Manager** window and select the network **network1**, then double click (or click **Open**).
- In the **Network:network1** window, click on the tab **Simulate**. Now use the **Inputs** drop-down menu to specify **p** as the input, and label the output as **network1_outputsSim** to distinguish it from the training output. Then click **Simulate Network**.



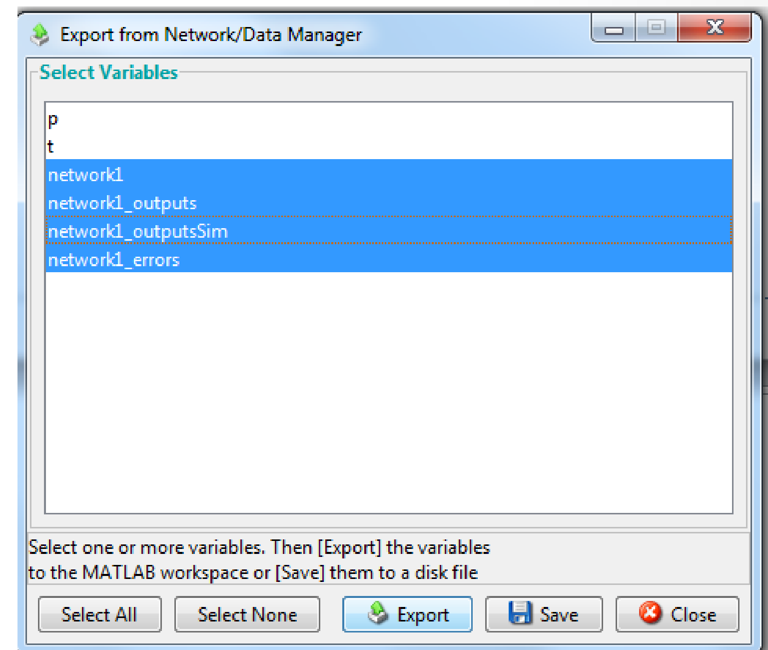
Example – Steps 6&7: Network Validation & Testing

- Look at the **Neural Network/Data Manager** window and you will see a new variable in the output: **network1_outputsSim**.
- Double-click on it and a small window **Data:Network1_outputsSim** appears with its values.



Example – Exporting the Network

- To export the network, network outputs and errors to the MATLAB command line workspace, go back to the **Neural Network/Data Manager**.
- Note that the output and error for the network1 are listed in the **Outputs and Error** lists on the right hand side.
- Next click on **Export.** This will give you an **Export** or **Save from Network/Data Manager** window.
- Click on
  - network1,
  - network1_outputs
  - network1_errors, and
  - network1_outputSim to highlight them.
- Then click on the **Export** button.



Example – Exporting the Network

- These four variables now should be in the command line workspace.
- To check this, go to the command line and type **who** to see all the defined variables. The result should be:

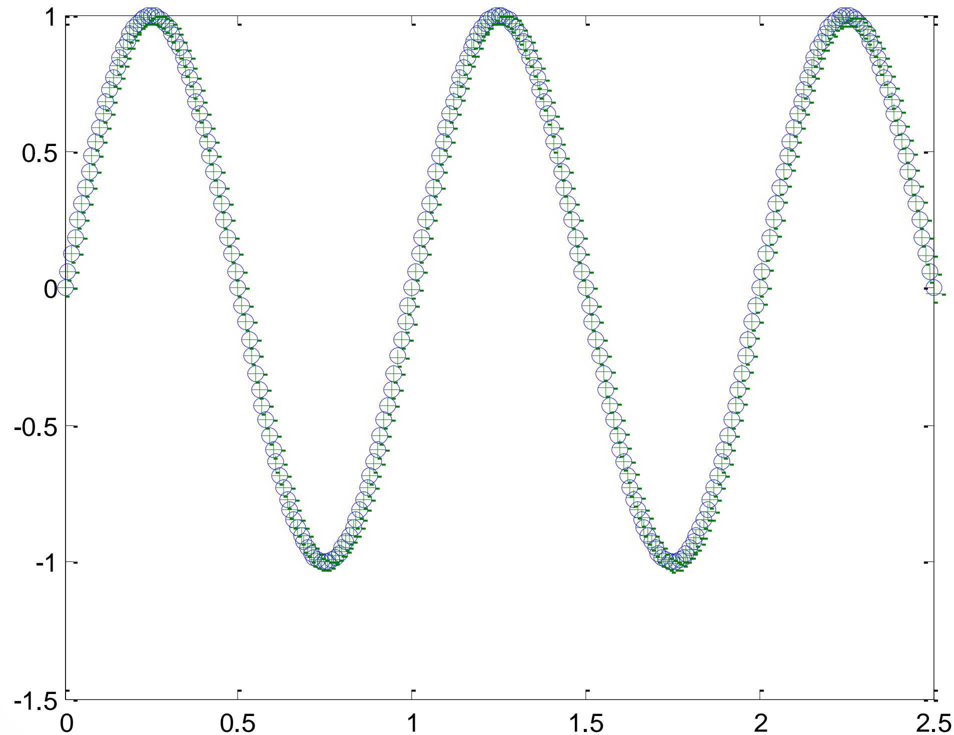
```
>> who

Your variables are:

network1          network1_outputs     p
network1_errors   network1_outputsSim  t
```

- Now we can plot the simulation output with the original target as:

```
>> plot(p,t,'o',p,network1_outputsSim,'+')
```

Example – Plotting the Network outputs

- This plot shows that the output of the trained network (symbol '+') agrees with the target (symbol 'o').



Example – Examining the Network

- Now that network1 is exported you can view the network description and examine the network weight matrix.

- For instance, the following gives the weight values of the hidden layer:

```
>> network1.iw{1,1}
```

- For instance, the following gives the bias values of the hidden layer:

```
>> network1.b{1}
```

- For instance, the following gives the weight values of the output layer:

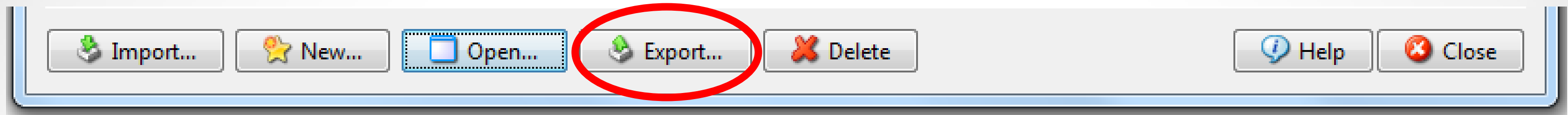
```
>> network1.LW{2}
```

- For instance, the following gives the bias values of the output layer:

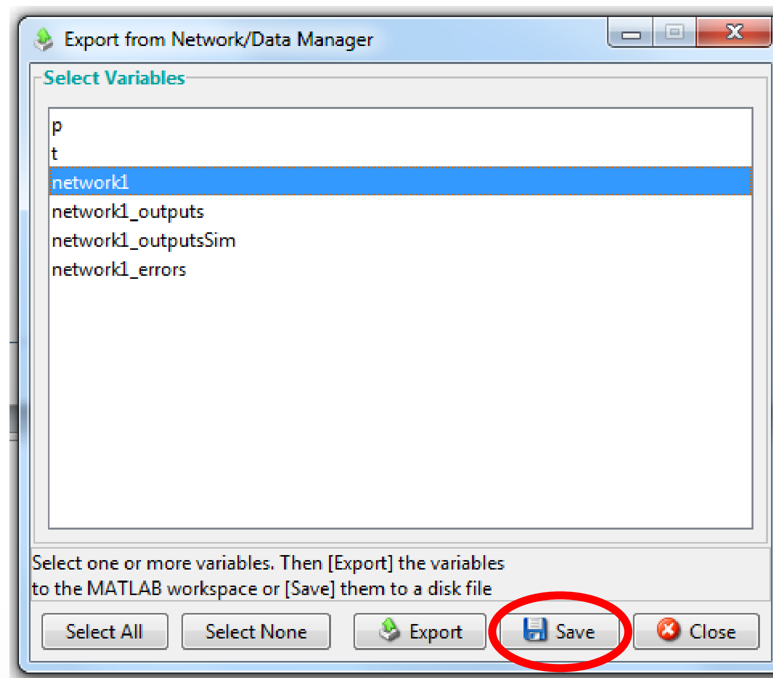
```
>> network1.b{2}
```

Example – Saving the Network

- Go to the **Neural Network/Data Manager** window and click on **Export**.

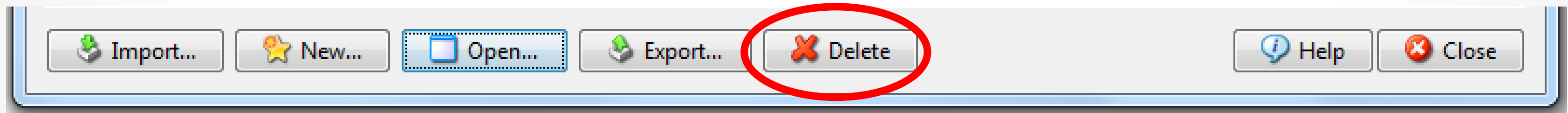


- Select **network1** in the variable list of the window and click on **Save**. This leads to the **Save to a MAT file** window. Save to a file named network1 or any name you like.



Example – Deleting the Network

- Go to the **Neural Network/Data Manager** window.
- You can clear the variables in the **Neural Network/Data Manager** window by highlighting a variable such as p and clicking the **Delete** button until all entries in the list boxes are gone.



Exercises

- Create and train different networks with different configuration:
  - Increase the number of the neurons in the hidden layer to 50
  - Change the transfer function of both layers to 'logsig'
  - Use 'traingdm' as training function instead of 'trainlm'
  - Reduce the number of neurons in the hidden layer to 2
- Use different names for each network above and also plot the simulation results and network errors for each case.

References

- Neural Network Toolbox™ User's Guide
  http://www.mathworks.com/help/pdf_doc/nnet/nnet_ug.pdf
- MathWorks Documentation Center
  http://www.mathworks.co.uk/help/nnet/index.html
- CE889 Lab Notes (Autumn 2011)