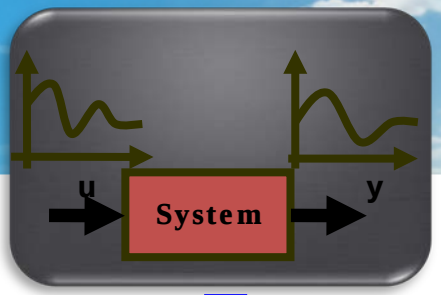
Introduction to MATLAB Neural Network Toolbox

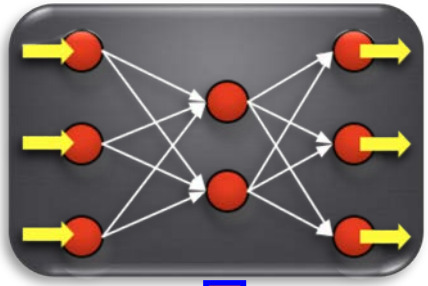
Static Neural Networks

Jeen-Shing Wang
Wei-Chun Chiang
Department of Electrical Engineering
National Cheng Kung University
2016/10/24

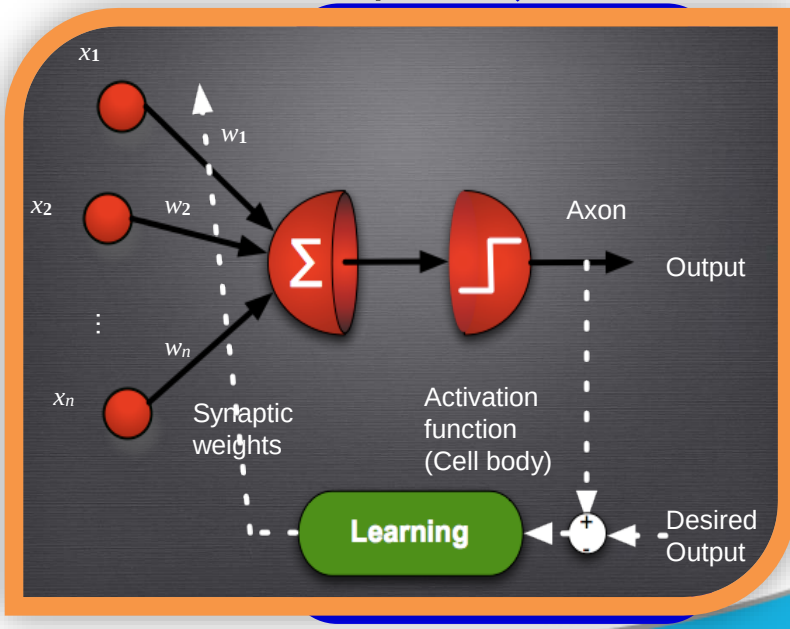
NCKU Computational Intelligence & Learning Systems LAB

Basic Concept



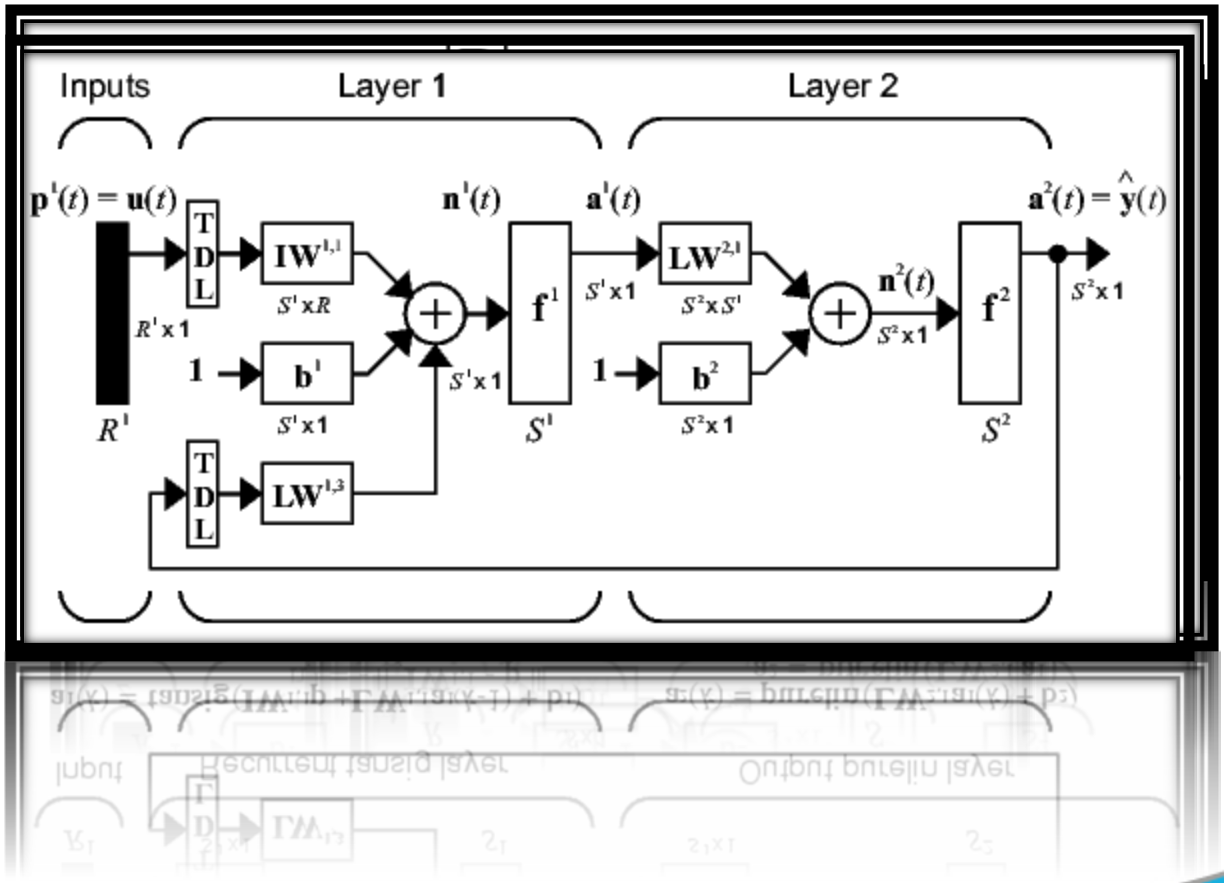


- Step 1: Data collection
  - ✓ Create training and testing data
- Step 2: Network creation
  - ✓ Static networks
  - ✓ Dynamic networks
- Step 3: Network parameter setting
  - ✓ No. of hidden layer, neurons, activation function, …
- Step 4: Learning algorithm selection
  - ✓ Backpropagation, …
  - ✓ Learning rate
- Step 5: Learning condition setting
  - ✓ Learning epochs, error, …
- Step 6: Network learning
- Step 7: Data testing

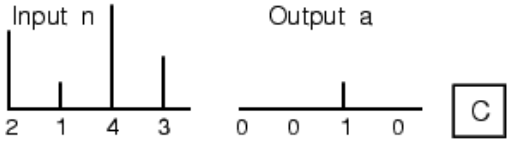


Neural Network Models

1. Perceptron
2. Linear Filters
3. Backpropagation
4. Radial Basis Networks
5. Competitive Networks
6. Learning Vector Quantization Network
7. Recurrent Networks
8. NARX Networks

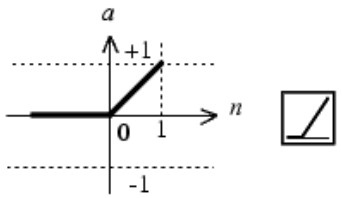


Transfer Function Graphs



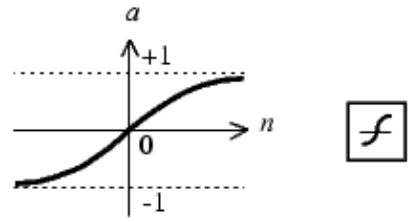
$a = compet(n)$

Compet Transfer Function



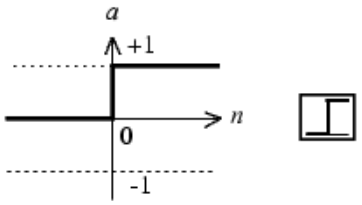
$a = poslin(n)$

Positive Linear Transfer Function



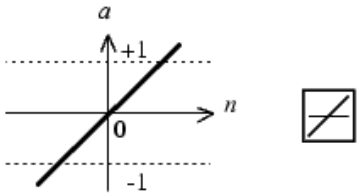
$a = tansig(n)$

Tan-Sigmoid Transfer Function



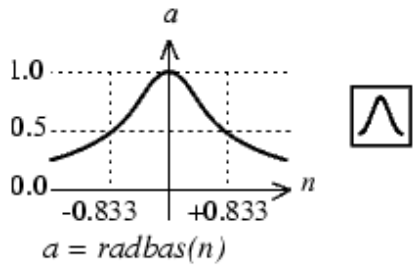
$a = hardlim(n)$

Hard-Limit Transfer Function



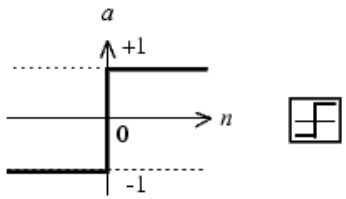
$a = purelin(n)$

Linear Transfer Function



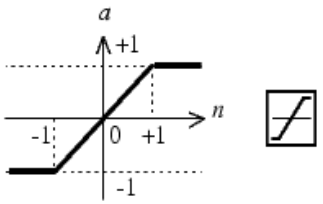
$a = radbas(n)$

Radial Basis Function



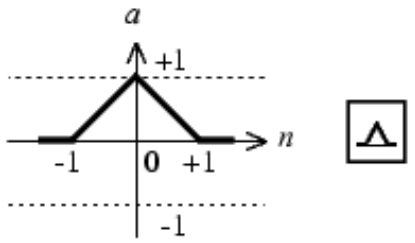
$a = hardlims(n)$

Symmetric Hard-Limit Transfer Function



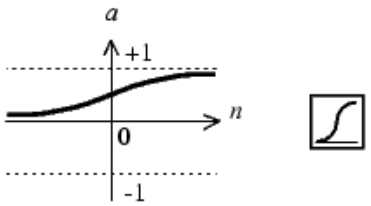
$a = satlins(n)$

Satlins Transfer Function



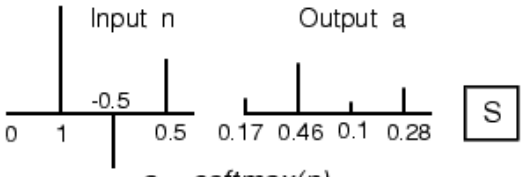
$a = tribas(n)$

Triangular Basis Function



$a = logsig(n)$

Log-Sigmoid Transfer Function



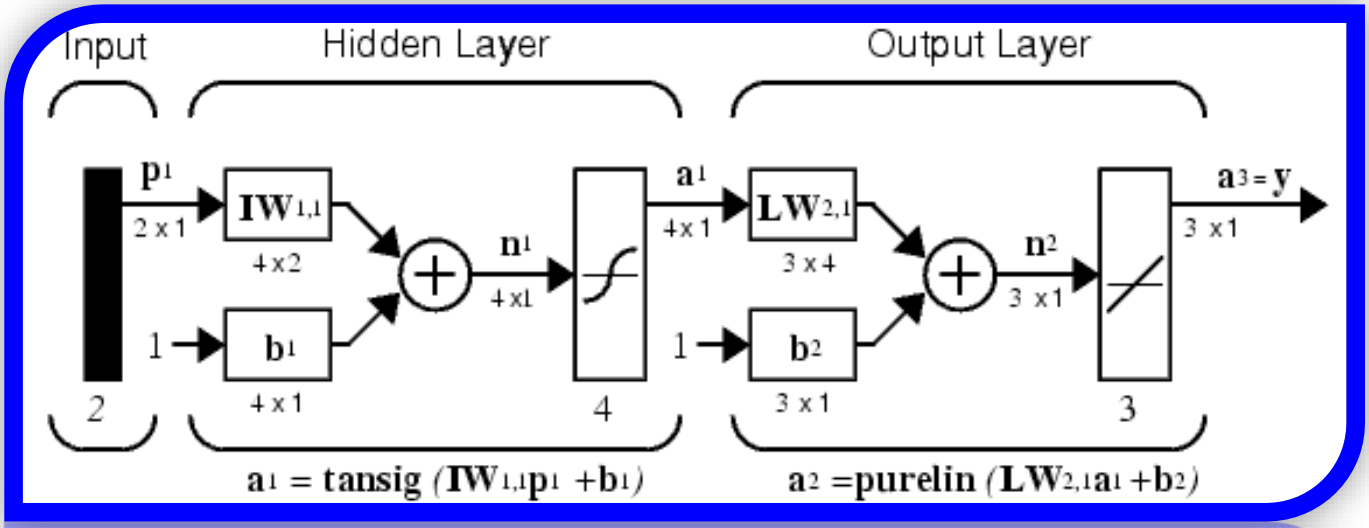
$a = softmax(n)$

Softmax Transfer Function

Learning Algorithms

Acronym	Algorithm	FA	PR	Scale	Memory
LM	trainlm - Levenberg-Marquardt	☼	☻	☻	☻
BFG	trainbfg - BFGS Quasi-Newton	☺	☻	☻	☺
OSS	trainoss - One-Step Secant	☺	☻	☻	☺
SCG	trainscg - Scaled Conjugate Gradient	☺	☺	☼	☺
CGB	traincgb - Conjugate Gradient with Powell/Beake Restarts	☺	☺	☼	☺
CGF	traincgf - Fletcher-Powell Conjugate Gradient	☺	☺	☼	☺
CGP	traincgp - Polak-Ribiere Conjugate Gradient	☺	☺	☼	☺
RP	trainrp - Resillent Backpropagation	☻	☼	☺	☼
GDX	traingdx - Variable Learning Rate Backpropagation	☻	☻	☺	☺

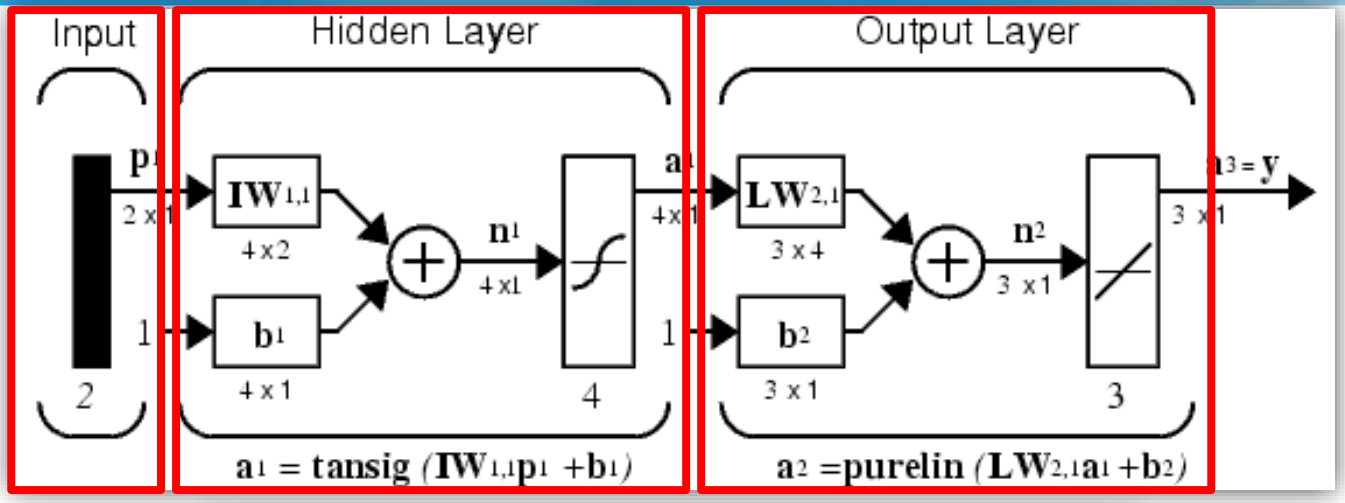
Feedforward Neural Networks

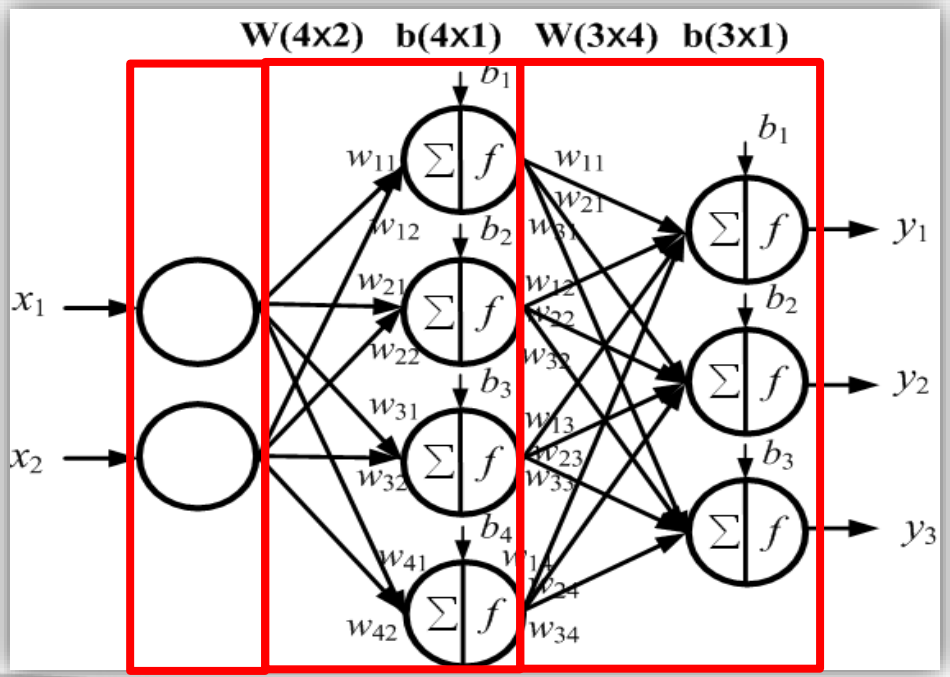


- Scalar: *a*, *b*, *c*;
- Vector: a, b, c;
- Matrix: A, B, C;

P ($r \times N$): Input data (r: No. of input elements, N: No. of input patterns)
Y ($m \times N$): Output data (m: No. of output elements)
n: n^{th} layer, including the hidden and output layers
$\mathbf{IW}^{k,l}$: Input weight matrix (l^{th} input set to k^{th} layer)
$\mathbf{LW}^{k,l}$: Layer weight matrix (l^{th} layer to k^{th} layer)

Feedforward Neural Networks





Feedforward Neural Networks

- **Syntax: newff**

net = newff(P, T, [S1 … S(n-1)], {TF1 … TFn}, BTF)

  - Si: Size of i^{th} layer, for N-1 layers.
    (Default = [].)
  - TFi: Transfer function of i^{th} layer.
    (Default = 'tansig' for hidden layers and 'purelin' for output layer.)
  - BTF: Backpropagation network training function
    (Default = 'trainlm')

Feedforward Neural Networks

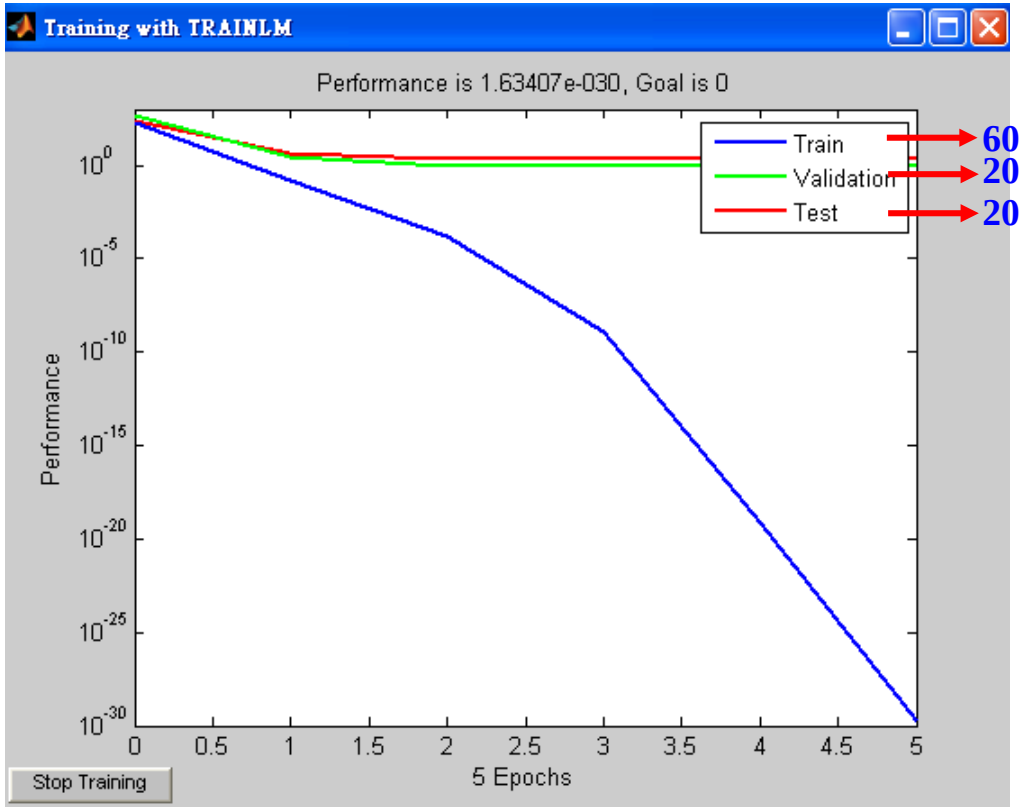
◆ **Example 1:** $t = \text{abs}(p)$

Training input data: **P** = [0 -1 2 -3 4 -5 6 -7 8 -9 10];

Training output data: **T** = [0 1 2 3 4 5 6 7 8 9 10];

60%

```
>>clc;  clear all; close all;
>>% Collect data
>>P = [0 -1 2 -3 4 -5 6 -7 8 -9 10];
>>T = [0 1 2 3 4 5 6 7 8 9 10];
>>% Create network
>>net = newff(P,T,10);
>>net
>>% Training
>>net = train(net, P, T);
>>% Testing
>>y = sim(net,P);
>>% Error
>>error = mse(y-T)
```



Feedforward Neural Networks

```
net =

    Neural Network object:

  architecture:

      numInputs: 1
      numLayers: 2
    biasConnect: [1; 1]
   inputConnect: [1; 0]
   layerConnect: [0 0; 1 0]
  outputConnect: [0 1]

     numOutputs: 1  (read-only)
 numInputDelays: 0  (read-only)
 numLayerDelays: 0  (read-only)

  subobject structures:

         inputs: {1x1 cell} of inputs
         layers: {2x1 cell} of layers
        outputs: {1x2 cell} containing 1 output
         biases: {2x1 cell} containing 2 biases
   inputWeights: {2x1 cell} containing 1 input weight
   layerWeights: {2x2 cell} containing 1 layer weight

  functions:

      adaptFcn: 'trains'
     divideFcn: 'dividerand'
   gradientFcn: 'calcjx'
       initFcn: 'initlay'
    performFcn: 'mse'
      trainFcn: 'trainlm'

  parameters:

     adaptParam: .passes
    divideParam: .trainRatio, .valRatio, .testRatio
 gradientParam: (none)
     initParam: (none)
  performParam: (none)
    trainParam: .epochs, .goal, .max_fail, .mem_reduc,
                .min_grad, .mu, .mu_dec, .mu_inc,
                .mu_max, .show, .time

  weight and bias values:

             IW: {2x1 cell} containing 1 input weight matrix
             LW: {2x2 cell} containing 1 layer weight matrix
              b: {2x1 cell} containing 2 bias vectors

  other:

       userdata: (user information)
```

Train Feedforward Neural Networks

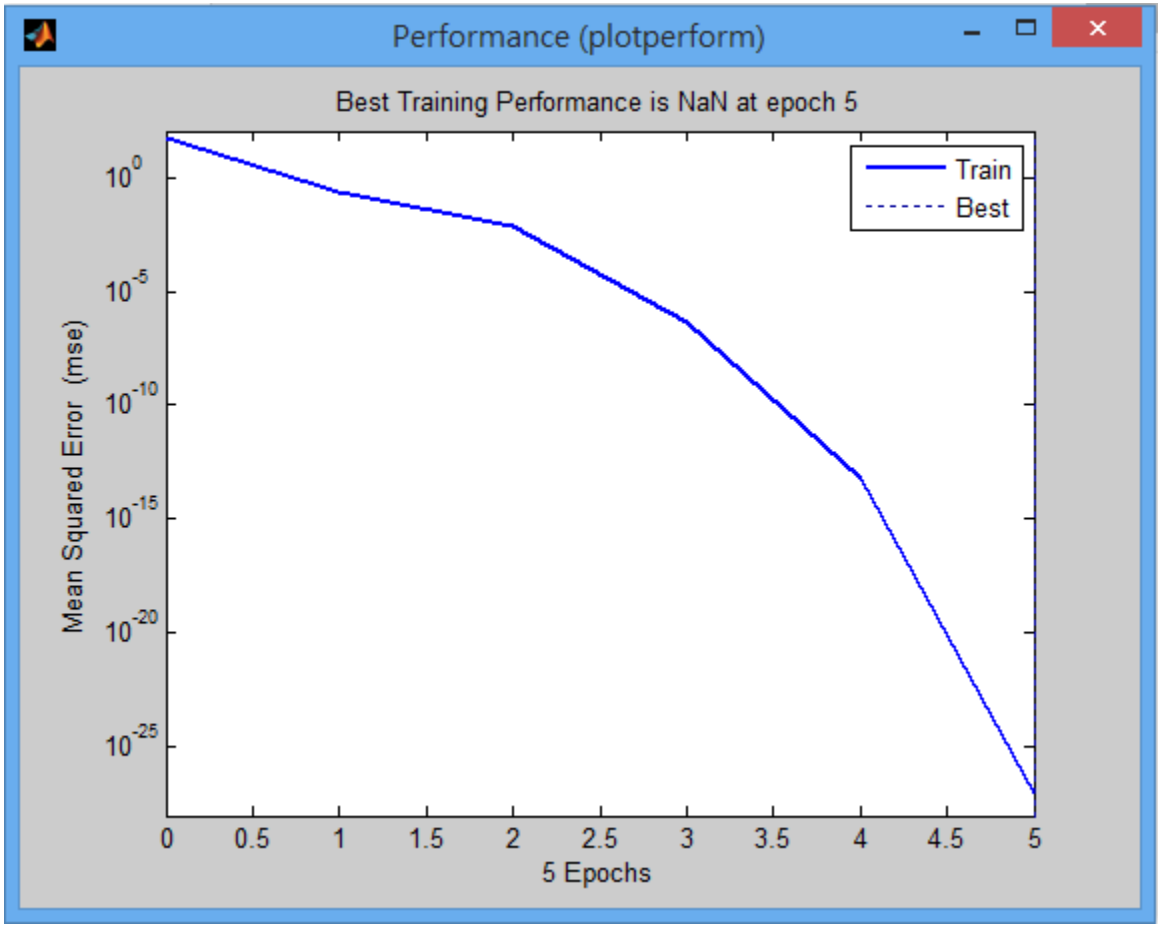
- **Set training parameter values: net.trainParam**
  - >>net.trainParam.epochs = 100;
  - >>net.trainParam.lr = 0.01;
  - >>net.trainParam.goal = 0;

- **Train the network**
  - >>net = train(net, **P**, **T**);
- **Simulate or test the network**
  - >>y = sim(net, $\mathbf{P}_t$);

Testing input data: $\mathbf{P}_t$ = [0 1 -2 3 -4 5 -6 7 -8 9 -10];
Testing output data: $\mathbf{T}_t$ = [0 1 2 3 4 5 6 7 8 9 10];

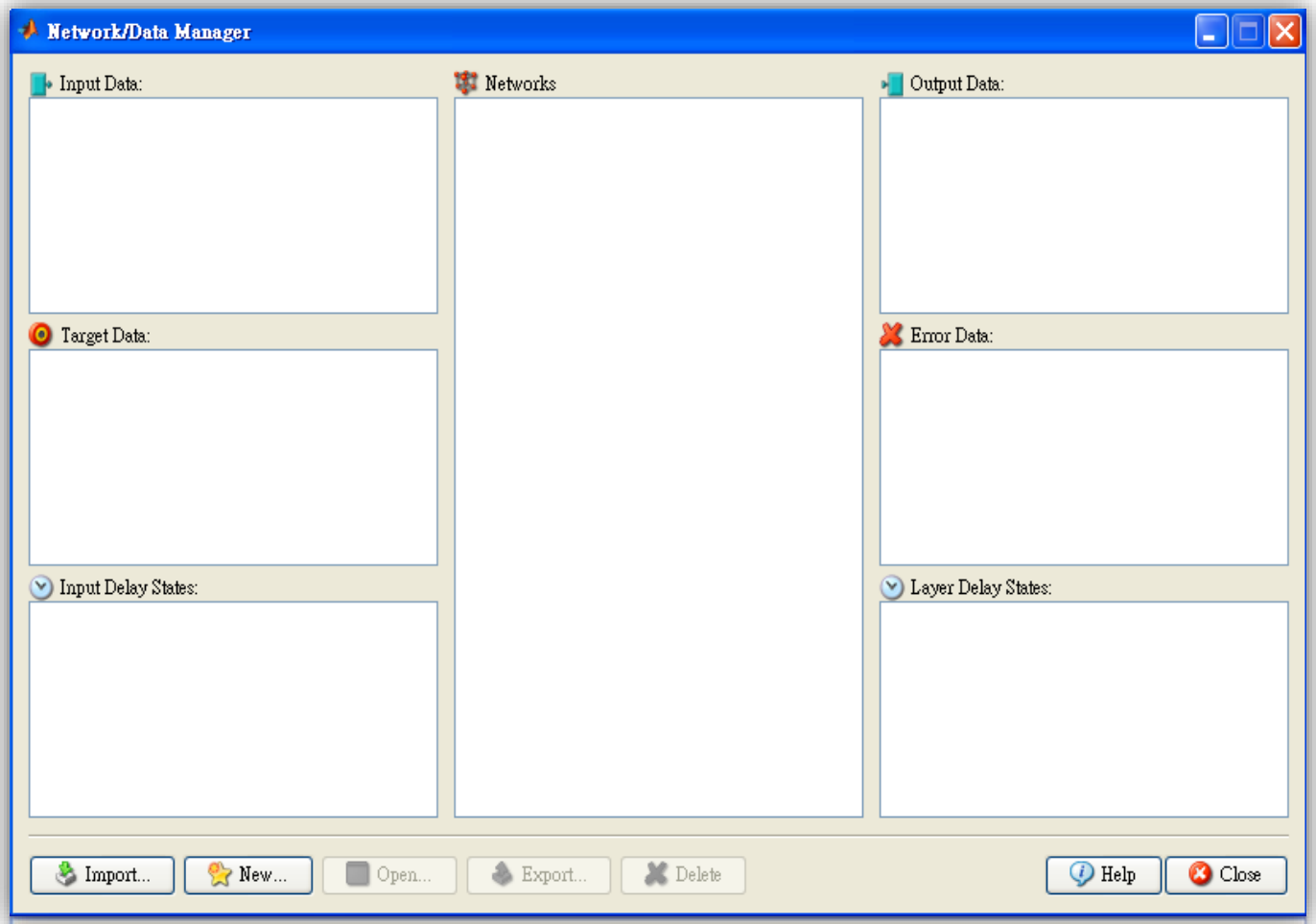
Train Feedforward Neural Networks

- >>net = train(net, **P**, **T**,[],[],[],[]);

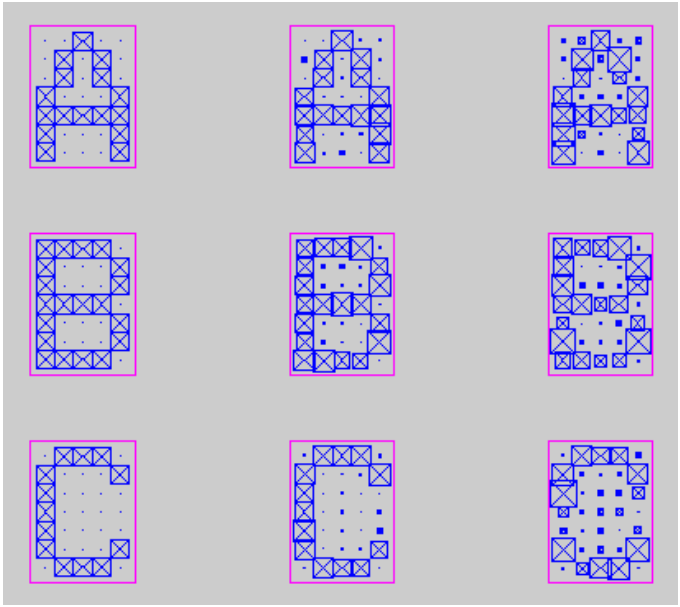


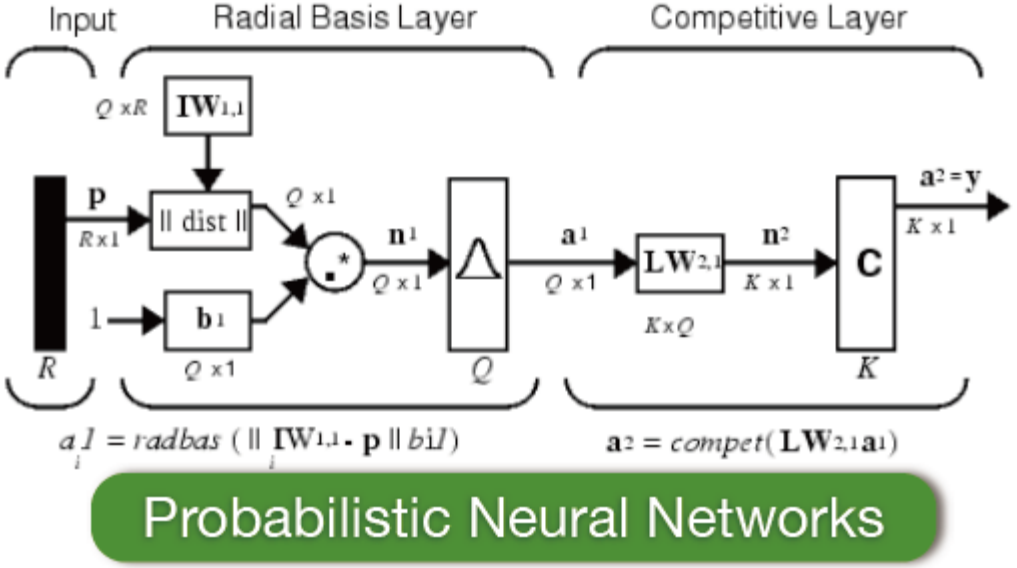
Graph User Interface (GUI)

- >>nntool



Pattern Recognition





Probabilistic Neural Networks

Function: ind2vec

```
>>P = [P1 P2 P3];
>>T = [1 3 2]; % class indices
>>Tc = ind2vec(T)
>>Tc = (1, 1)  1
       (3, 2)  1
       (2, 3)  1
```

```
>>Tc = ind2vec(T_for_PNN);
>>pnnnet = newpnn(P, Tc);
>>Yc = sim(pnnnet, test_letter);
>>test_index
>>Y = vec2ind(Yc)
```

The training/testing data are available on the course website (http://140.116.215.51)

Thanks for your attention !!

Any Question?