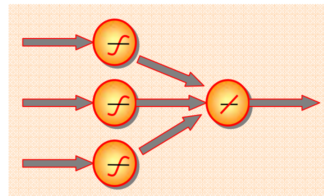
TECHSOURCE
www.techsource.com.my

Statistical & Data Analysis Using Neural Network

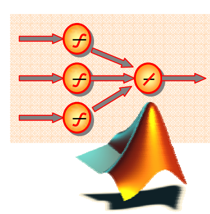


TechSource Systems Sdn. Bhd.

TECHSOURCE
www.techsource.com.my

Course Outline:

1. **Neural Network Concepts**
   a) Introduction
   b) Simple neuron model
   c) MATLAB representation of neural network
2. **Types of Neural Network**
   a) Perceptrons
   b) Linear networks
   c) Backpropagation networks
   d) Self-organizing maps
3. **Case Study: Predicting Time Series**



Neural Network Concepts

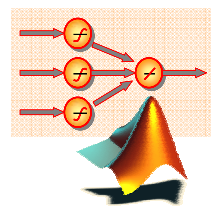
www.techsource.com.my

Section Outline:

1. Introduction
- Definition of neural network
- Biological perspective of neural network
- Neural network applications

2. Simple neuron model
- Components of simple neuron

3. MATLAB representation of neural network
- Single neuron model
- Neural network with single-layer of neurons
- Neural network with multiple-layer of neurons

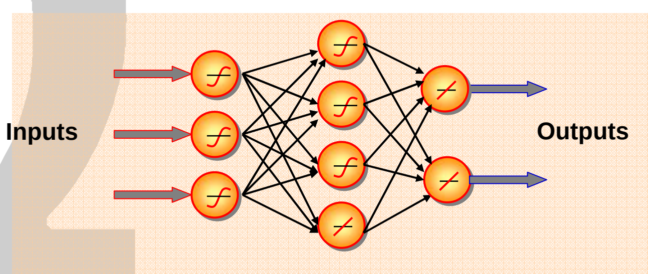


Neural Network Concepts

www.techsource.com.my

Definition of Neural Network

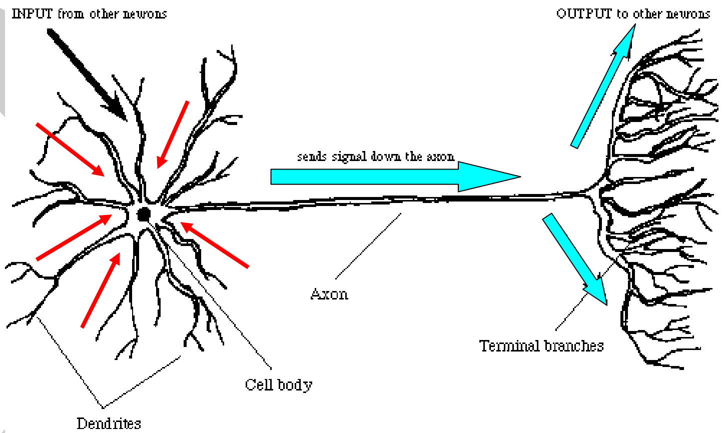
"A neural network is an interconnected assembly of simple processing elements, *units* or *nodes*, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the inter-unit connection strengths, or *weights*, obtained by a process of adaptation to, or *learning* from, a set of training patterns."



Neural Network Concepts

TECHSOURCE
www.techsource.com.my

The Biological Perspective of Neural Networks



Neural Network Concepts

TECHSOURCE
www.techsource.com.my

Neural Network Applications:

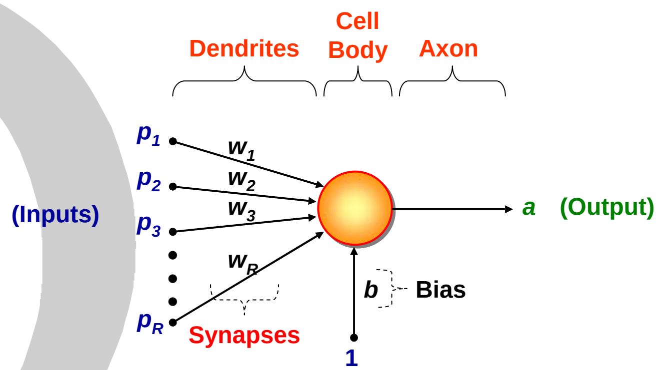
- Aerospace
- Automotive
- Banking
- Credit Card Activity Checking
- Defense
- Electronics
- Entertainment
- Financial
- Industrial
- Insurance
- Manufacturing
- Medical
- Oil & Gas
- Robotics
- Speech
- Securities
- Telecommunications
- Transportation

©2005 TECHSOURCE Systems Sdn. Bhd.

Neural Network Concepts

TECHSOURCE www.techsource.com.my

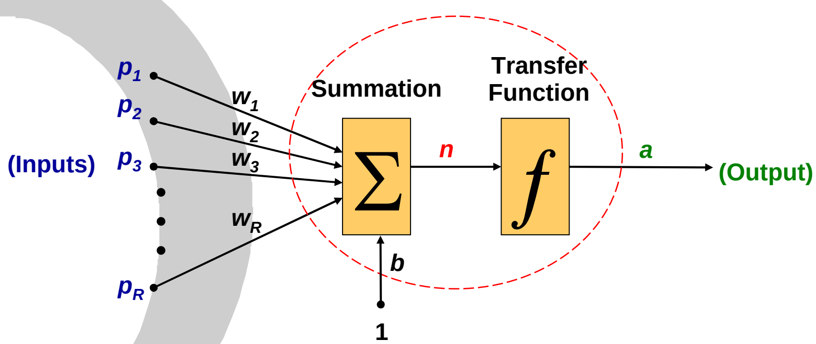
Components of Simple Neuron



Neural Network Concepts

TECHSOURCE www.techsource.com.my

Components of Simple Neuron

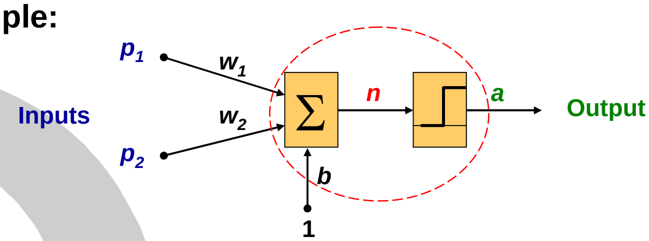


$$a = f(w_1p_1 + w_2p_2 + w_3p_3 + \ldots + w_Rp_R + b)$$

Neural Network Concepts

TECHSOURCE www.techsource.com.my

Example:



If p_1 = 2.5; p_2 = 3; w_1 = 0.5; w_2 = -0.7; b = 0.3. Let's assume the transfer function of the neuron is *hardlimit*, where,

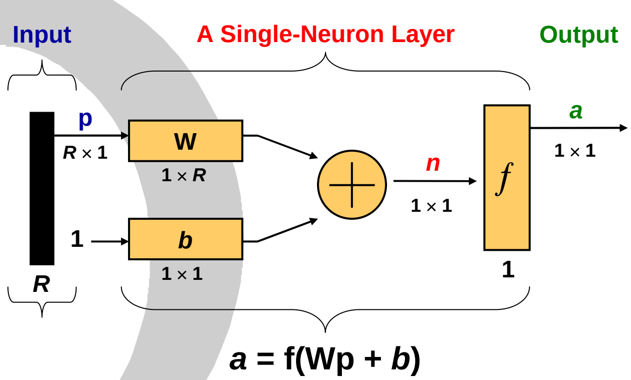
$$a = \text{hardlim}(n) = \begin{cases} 0 & \text{for } n < 0 \\ 1 & \text{for } n \geq 0 \end{cases}$$

$$\begin{aligned} \therefore\ a = \text{hardlim}(n) &= \text{hardlim}(\ w_1p_1 + w_2p_2 + b) \\ &= \text{hardlim}(\ 0.5\times 2.5 + (-0.7)\times 3 + 0.3\ ) \\ &= \text{hardlim}(\ -0.55) \\ &= 0 \end{aligned}$$

Neural Network Concepts

TECHSOURCE www.techsource.com.my

MATLAB® Representation of the Simple Neuron Model



$$a = f(\mathbf{W}\mathbf{p} + b)$$

R **= number of input elements**

Input Vector $\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \bullet \\ \bullet \\ \bullet \\ p_R \end{bmatrix}_{R\times 1}$

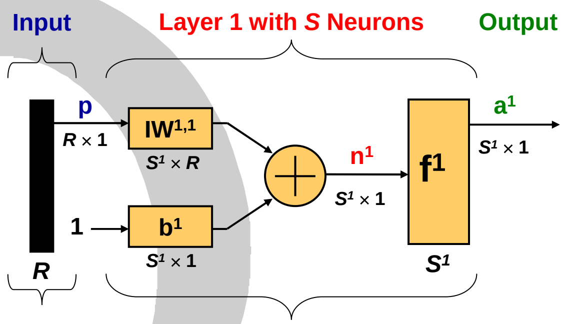
Weight Vector

$\mathbf{W} = [w_1\ w_2 \bullet\bullet\bullet\ w_R]_{1\times R}$

Neural Network Concepts

TECHSOURCE www.techsource.com.my

Single Layer of Neurons



$$a^1 = f^1(IW^{1,1}p + b^1)$$

R = number of input elements

Where,

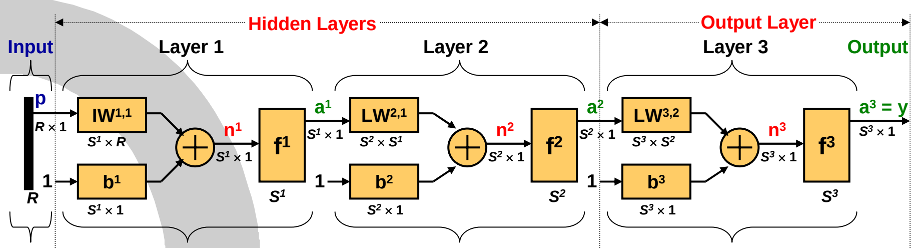
- S^1: Number of neurons in Layer 1
- $IW^{1,1}$: Input Weight matrix for connection from Input to Layer 1

$$IW^{1,1} = \begin{bmatrix} iw^{1,1}_{1,1} & iw^{1,1}_{1,2} & \dots & iw^{1,1}_{1,R} \\ iw^{1,1}_{2,1} & iw^{1,1}_{2,2} & \dots & iw^{1,1}_{2,R} \\ \vdots & \vdots & & \vdots \\ iw^{1,1}_{S,1} & iw^{1,1}_{S,2} & \dots & iw^{1,1}_{S,R} \end{bmatrix}_{S \times R}$$

Neural Network Concepts

TECHSOURCE www.techsource.com.my

Multiple Layer of Neurons



$a^1 = f^1(IW^{1,1}p + b^1)$ $\quad a^2 = f^2(LW^{2,1}a^1 + b^2)$ $\quad a^3 = f^3(LW^{3,2}a^2 + b^3)$

$$\therefore a^3 = f^3(LW^{3,2} f^2(LW^{2,1} f^1(IW^{1,1}p + b^1) + b^2) + b^3) = y$$

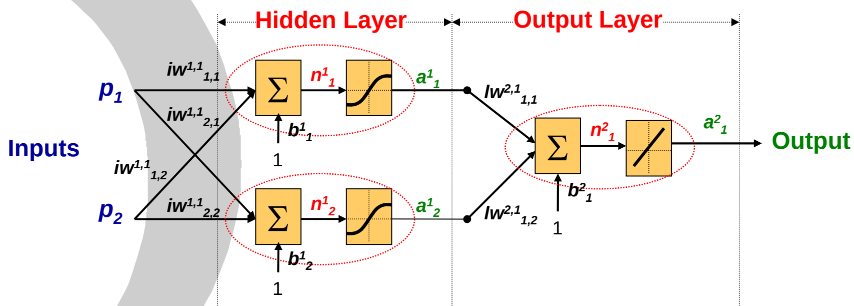
- S^1, S^2, S^3: Number of neurons in Layer 1, Layer 2, Layer 3 respectively
- $IW^{1,1}$: Input Weight matrix for connection from Input to Layer 1
- $LW^{2,1}$: Layer Weight matrix for connection from Layer 1 to Layer 2
- $LW^{3,2}$: Layer Weight matrix for connection from Layer 2 to Layer 3

Neural Network Concepts

TECHSOURCE www.techsource.com.my

Example:

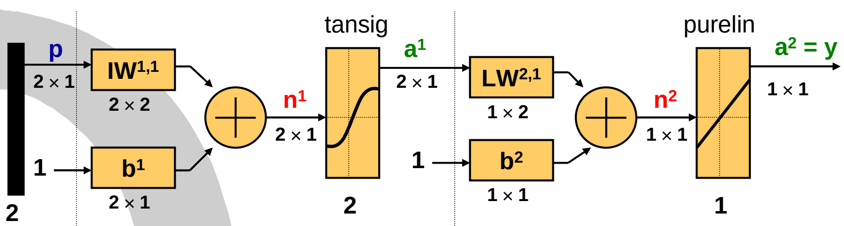
Neural network with 2 layers. 1st layer (hidden layer) consists of 2 neurons with *tangent-sigmoid* (tansig) transfer functions; 2nd layer (output layer) consists of 1 neuron with *linear* (purelin) transfer function.



Neural Network Concepts

TECHSOURCE www.techsource.com.my

In MATLAB® abbreviated notation, the neural network is represented by the diagram below.

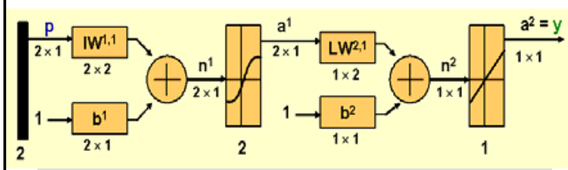


$$\therefore\ a^2 = \text{purelin}(LW^{2,1}\,\text{tansig}(IW^{1,1}p + b^1) + b^2) = y$$

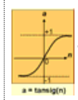
$$p = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \quad IW^{1,1} = \begin{bmatrix} iw^{1,1}_{1,1} & iw^{1,1}_{1,2} \\ iw^{1,1}_{2,1} & iw^{1,1}_{2,2} \end{bmatrix} \quad b^1 = \begin{bmatrix} b^1_1 \\ b^1_2 \end{bmatrix} \quad LW^{2,1} = \begin{bmatrix} lw^{2,1}_{1,1} & lw^{2,1}_{1,2} \end{bmatrix} \quad b^2 = \begin{bmatrix} b^2_1 \end{bmatrix}$$

$$n^1 = \begin{bmatrix} n^1_1 \\ n^1_2 \end{bmatrix} \quad a^1 = \begin{bmatrix} a^1_1 \\ a^1_2 \end{bmatrix} \quad n^2 = \begin{bmatrix} n^2_1 \end{bmatrix} \quad a^2 = \begin{bmatrix} a^2_1 \end{bmatrix} = y$$

Neural Network Concepts



$\therefore$ $a^2 = \text{purelin}(LW^{2,1}\,\text{tansig}(IW^{1,1}p + b^1) + b^2) = y$



tansig(n) = 2 / ((1 + e^{-2n}) -1)

purelin(n) = n

For,

$p = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ $IW^{1,1} = \begin{bmatrix} 0.3 & -0.7 \\ -0.2 & 0.5 \end{bmatrix}$ $b^1 = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$ $LW^{2,1} = \begin{bmatrix} 0.1 & -0.2 \end{bmatrix}$ $b^2 = \begin{bmatrix} 0.3 \end{bmatrix}$

$$
\begin{aligned}
\therefore y = a^2 &= \text{purelin}(LW^{2,1}\,\text{tansig}(IW^{1,1}p + b^1) + b^2) \\
&= \text{purelin}([0.1\ -0.2] \times \text{tansig}([0.3\ -0.7\ ;\ -0.2\ 0.5] \times [1\ ;\ 2] + [0.1\ ;\ 0.2]) + 0.3) \\
&= \text{purelin}([0.1\ -0.2] \times \text{tansig}([-1\ ;\ 1]) + 0.3) \\
&= \text{purelin}(0.0715) \\
&= 0.0715
\end{aligned}
$$

Neural Network Concepts

Section Summary:

1. Introduction

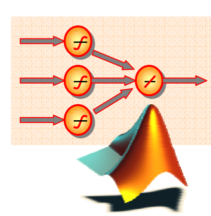
- Definition of neural network
- Biological perspective of neural network
- Neural network applications

2. Simple neuron model

- Components of simple neuron

3. MATLAB representation of neural network

- Single neuron model
- Neural network with single-layer of neurons
- Neural network with multiple-layer of neurons



Types of Neural Network

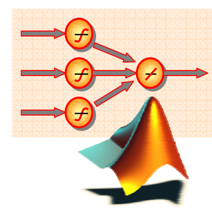
Section Outline:

1. Perceptrons
- Introduction
- The perceptron architecture
- Training of perceptrons
- Application examples

2. Linear Networks
- Introduction
- Architecture of linear networks
- The Widrow-Hoff learning algorithm
- Application examples

3. Backpropagation Networks
- Introduction
- Architecture of backprogation network
- The backpropagation algorithm
- Training algorithms
- Pre- and post-processing
- Application examples

4. Self-Organizing Maps
- Introduction
- Competitive learning
- Self-organizing maps
- Application examples



Types of Neural Network

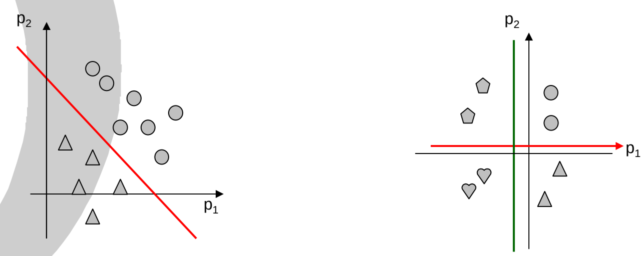
Perceptrons

- Invented in 1957 by Frank Rosenblatt at Cornell Aeronautical Laboratory.
- The perceptron consists of a single-layer of neurons whose weights and biases could be *trained* to produce a correct target vector when presented with corresponding input vector.
- The output from a single perceptron neuron can only be in one of the two states. If the weighted sum of its inputs exceeds a certain threshold, the neuron will *fire* by outputting 1; otherwise the neuron will output either 0 or -1, depending on the transfer function used.
- The perceptron can only solve ***linearly separable*** problems.

Types of Neural Network

Linearly Separable Problems

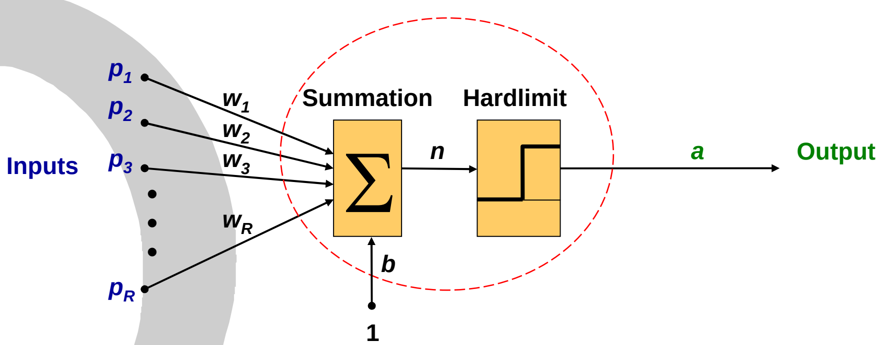
- **If a *straight* line can be drawn to separate the input vectors into two categories, the input vectors are linearly separable, as illustrated in the diagram below. If need to identify four categories, we need to use two perceptron neurons.**



(a) Two categories of input vectors **(b) Four categories of input vectors**

Types of Neural Network

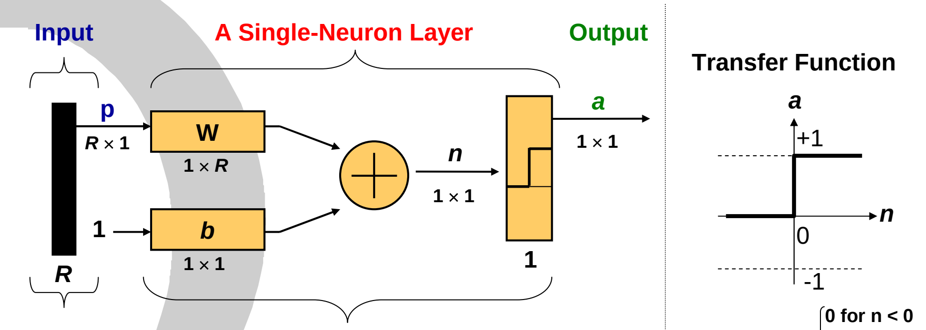
The Perceptron Neuron



$$a = hardlim(w_1p_1 + w_2p_2 + w_3p_3 + \ldots + w_Rp_R + b)$$

Types of Neural Network

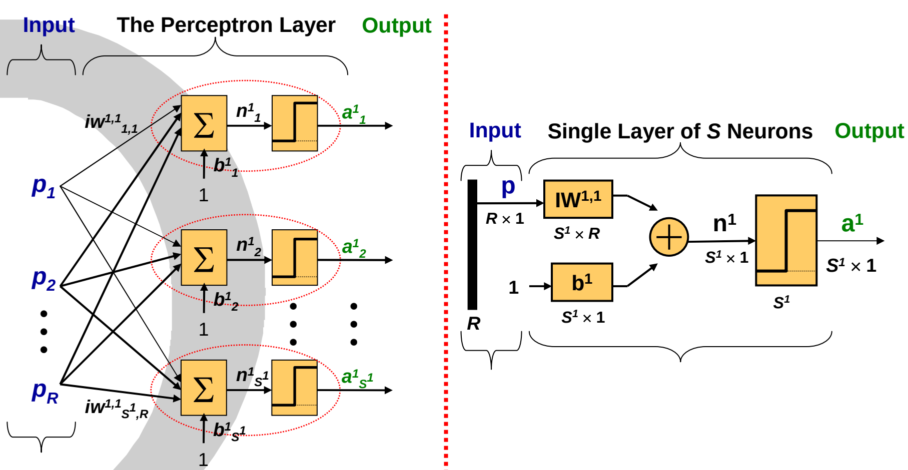
MATLAB® Representation of the Perceptron Neuron



$$a = \text{hardlim}(Wp + b)$$

Types of Neural Network

The Perceptron Architecture

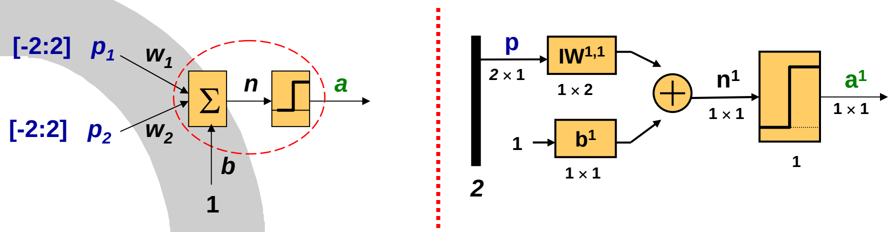


$$a^1 = \text{hardlim}(IW^{1,1}p + b^1)$$

Types of Neural Network

TECHSOURCE www.techsource.com.my

Creating a Perceptron: *Command-Line Approach*



The example below illustrates how to create a two-input, single-ouput perceptron. The input values range from -2 to 2.

```
% Creating a perceptron
>> net = newp([-2 2; -2 2], 1);

% Checking properties and values of Input Weights
>> net.inputWeights{1,1}   % properties
>> net.IW{1,1}   % values
```


Types of Neural Network

TECHSOURCE www.techsource.com.my

```
% Checking properties and values of bias
>> net.biases{1}  % properties
>> net.b{1}  % values

% Note that initial weights and biases are initialized to zeros using "initzero"
>> net.inputWeights{1,1}.initFcn
>> net.biases{1}.initFcn
```

% To compute the output of perceptron from input vectors [p_1; p_2], use the "sim" command

```
>> p = [ [2; 2] [1; -2] [-2; 2] [-1; 1] ]
>> a = sim(net, p)
>> a =
        1    1  1  1
```

Types of Neural Network

TECHSOURCE www.techsource.com.my

The Perceptron Learning Rule

- **Perceptrons are trained on examples of desired behavior, which can be summarized by a set of input-output pairs**

$$\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}$$

- **The objective of training is to reduce the error e, which is the difference $t - a$ between the perceptron output a, and the target vector t.**
- **This is done by adjusting the *weights* (W) and *biases* (b) of the perceptron network according to following equations**

$$W^{new} = W^{old} + \Delta W = W^{old} + ep^T$$

$$b^{new} = b^{old} + \Delta b = b^{old} + e$$

Where $e = t - a$

Types of Neural Network

TECHSOURCE www.techsource.com.my

Training of Perceptron

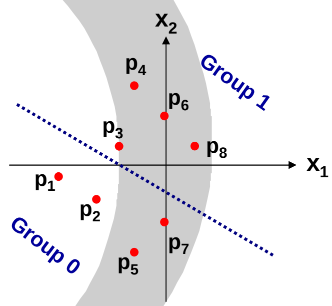
- **If the Perceptron Learning Rule is used repeatedly to adjust the weights and biases according to the error e, the perceptron wil eventually find weight and bias values that solve the problem, given that the perceptron *can* solve it.**
- **Each traverse through all the training vectors is called an *epoch*.**
- **The process that carries out such a loop of calculation is called *training*.**

Types of Neural Network

TECHSOURCE www.techsource.com.my 10

Example:

We can train a Perceptron network to classify two groups of data, as illustrated below



Data	x_1	x_2	Group
p_1	-3	-0.5	0
p_2	-2	-1.2	0
p_3	-1.5	0.7	1
p_4	-1	3	1
p_5	-1	-3.5	0
p_6	0	2	1
p_7	0	-2.5	0
p_8	1	0.7	1

Types of Neural Network

TECHSOURCE www.techsource.com.my 11

Procedures:

```
% Load the data points into Workspace
>> load data

% Assign training inputs and targets
>> p = points; % inputs
>> t = group;  % targets

% Construct a two-input, single-output perceptron
>> net = newp(minmax(p), 1);

% Train the perceptron network with training inputs (p) and targets (t)
>> net = train(net, p, t)

% Simulate the perceptron network with same inputs again
>> a = sim(net, p)
>> a =
      0  0  1  1  0  1  0  1   % correct classification
>> t =
      0  0  1  1  0  1  0  1
```

Types of Neural Network

TECHSOURCE www.techsource.com.my 12

% Let's be more adventurous by querying the perceptron with inputs it never *seen* before

>> t_1 = [-2; -3];

>> t_2 = [0.5; 4];

>> a_t_1 = sim(net, t_1)

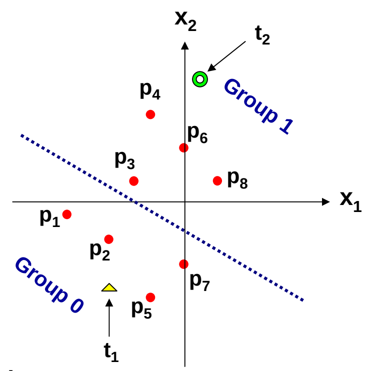
>> a_t_1 =

0

>> a_t_2 = sim(net, t_2)

>> a_t_2 =

1



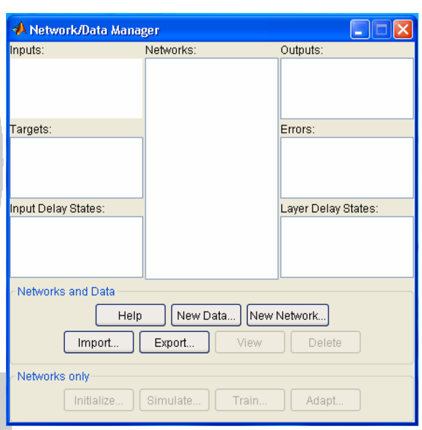
∴ The perceptron classifies t_1 and t_2 correctly.

Types of Neural Network

TECHSOURCE www.techsource.com.my 13

Using "nntool" GUI

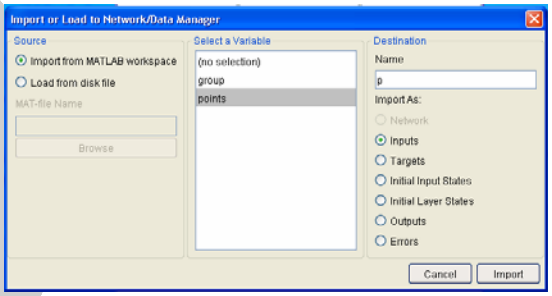
- The "nntool" GUI can be used to create and train different types of neural network available under MATLAB® Neural Network Toolbox
- The GUI can be invoked by typing at the command window,
  >> nntool



Types of Neural Network

TECHSOURCE www.techsource.com.my 14

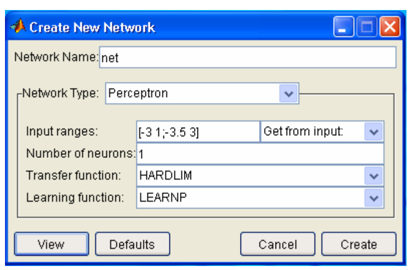
- **First, define the training inputs by clicking *"Import…"*, select *group* from the list of variables. Assign a *name* to the inputs and indicate that this variable should be imported as *inputs*.**
- **Define the targets similarly.**



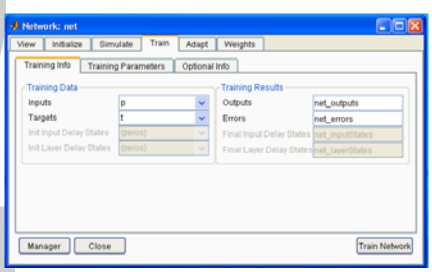
- **Create a new perceptron network by clicking *"New Network…"*, a new window appears where network architecture can be defined. Click *"Create"* to create the network.**

Types of Neural Network

TECHSOURCE www.techsource.com.my 15

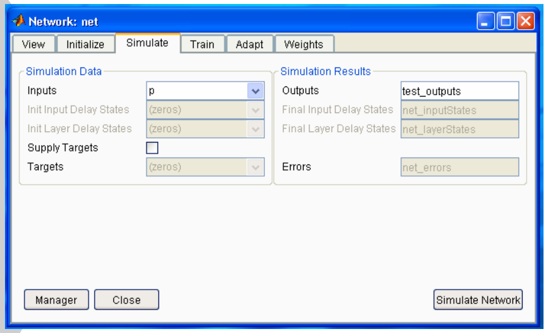


- **Next, select the *"Train…"* tab and set Inputs to *p* and Targets to *t*. Click on *"Train Network"* to start the training.**



Types of Neural Network

- **The network completed training in 4 epochs, which is 4 complete passes through all training inputs.**
- **Now, we can test the performance of the trained network by clicking *"Simulate..."*. Set the Inputs to *p*.**



Types of Neural Network

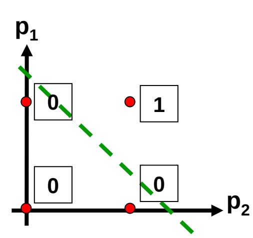
Exercise 1: Modeling Logical AND Function

The Boolean AND function has the following truth table:

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

The problem is linearly-separable, try to build a one-neuron perceptron network with following inputs and output:

p_1	p_2	a
0	0	0
0	1	0
1	0	0
1	1	1



Types of Neural Network

Solution:

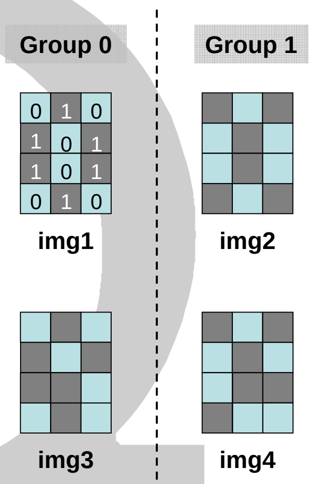
Command-line approach is demonstrated herein. The "nntool" GUI can be used alternatively.

```
% Define at the MATLAB® command window, the training inputs and targets
>> p = [0 0 1 1; 0 1 0 1]; % training inputs, p = [p1; p2]
>> t = [0 0 0 1]; % targets

% Create the perceptron
>> net = newp([0 1; 0 1], 1);

% Train the perceptron with p and t
>> net = train(net, p, t);

% To test the performance, simulate the perceptron with p
>> a = sim(net, p)
>> a =
        0  0  0  1
>>  t =
        0  0  0  1
```


Types of Neural Network

Exercise 2: Pattern Classification

Build a perceptron that can differentiate between two group of images:



Hint:

Use Boolean values 1's and 0's to represent the image.

Example for image_1 is shown.

∴ image_1 = [0 1 0
1 0 1
1 0 1
0 1 0]';

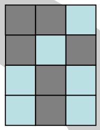
Load the training vectors into workspace:

```
>> load train_images
```

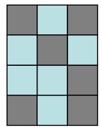
Types of Neural Network

TECHSOURCE www.techsource.com.my 20

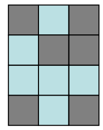
Try testing the trained perceptron on following images:



timg1



timg2



timg3

```
>> load test_images
```


Types of Neural Network

TECHSOURCE www.techsource.com.my 21

Solution:

Command-line approach is demonstrated herein. Tne "nntool" GUI can be used alternatively.

```
% Define at the MATLAB® command window, the training inputs and targets
>> load train_images
>> p = [img1 img2 img3 img4];
>> t = targets;

% Create the perceptron
>> net = newp(minmax(p), 1);

% Training the perceptron
>> net = train(net, p, t);

% Testing the performance of the trained perceptron
>> a = sim(net, p)

% Load the test images and ask the perceptron to classify it
>> load test_images
>> test1 = sim(net, timg1) % to do similarly for other test images
```

Types of Neural Network

www.techsource.com.my

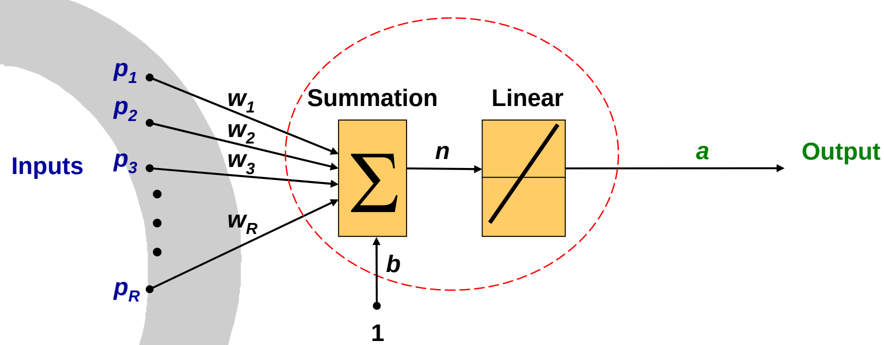
Linear Networks

- Linear networks are similar to perceptron, but their transfer function is linear rather than hard-limiting.
- Therefore, the output of a linear neuron is not limited to 0 or 1.
- Similar to perceptron, linear network can only solve ***linearly separable*** problems.
- Common applications of linear networks are linear classification and adaptive filtering.

Types of Neural Network

www.techsource.com.my

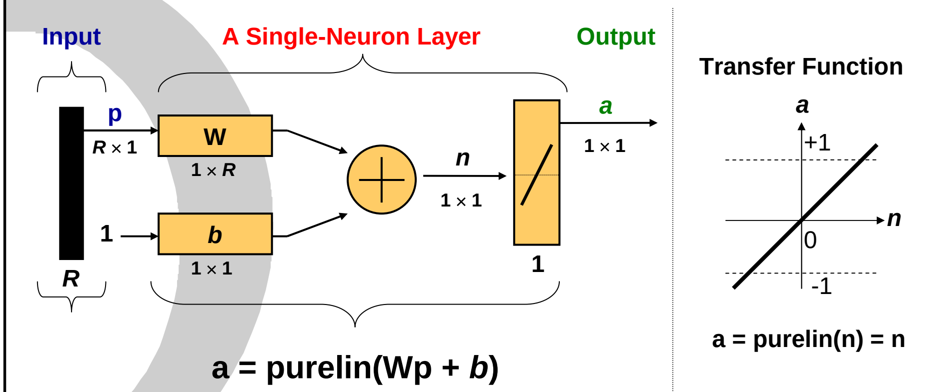
The Linear Neuron



$$a = purelin(w_1p_1 + w_2p_2 + w_3p_3 + \ldots + w_Rp_R + b)$$

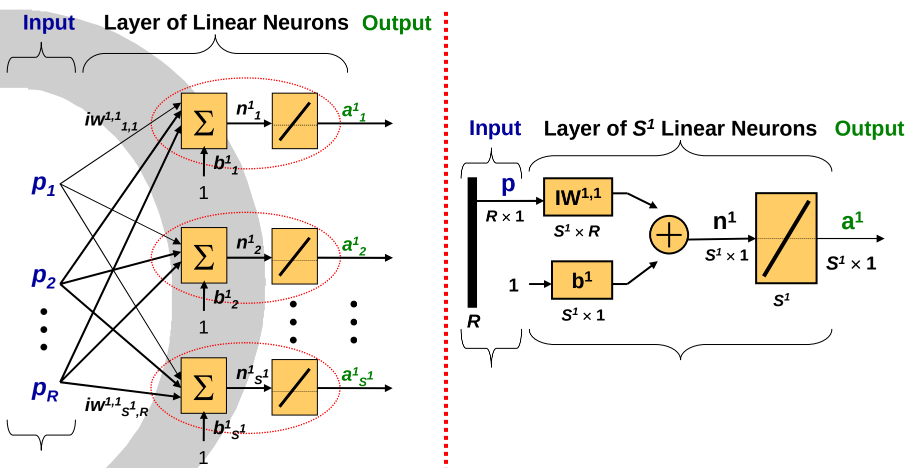
Types of Neural Network

MATLAB® Representation of the Linear Neuron



Types of Neural Network

Architecture of Linear Networks

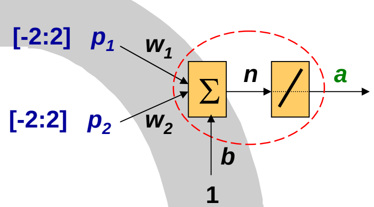


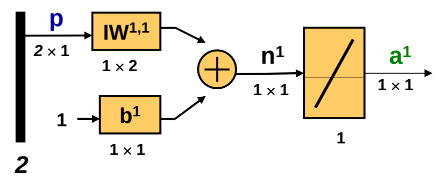
$$a^1 = \mathrm{purelin}(IW^{1,1}p + b^1)$$

Types of Neural Network

TECHSOURCE www.techsource.com.my 5

Creating a Linear Network: *Command-Line Approach*





The example below illustrates how to create a two-input, single-ouput linear network via command-line approach. The input values range from -2 to 2.

```
% Creating a linear network
>> net = newlin([-2 2; -2 2], 1);

% Checking properties and values of Input Weights
>> net.inputWeights{1,1}   % properties
>> net.IW{1,1}   % values
```


Types of Neural Network

TECHSOURCE www.techsource.com.my 6

```
% Checking properties and values of bias
>> net.biases{1}  % properties
>> net.b{1}  % values

% Note that initial weights and biases are initialized to zeros using "initzero"
>> net.inputWeights{1,1}.initFcn
>> net.biases{1}.initFcn

% To compute the output of linear network from input vectors [p1; p2], use
the "sim" command
>> p = [ [2; 2] [1; -2] [-2; 2] [-1; 1] ]
>> a = sim(net, p)
>> a =
        0   0   0   0
```

The Widrow-Hoff Learning Algorithm

- Similar to perceptron, the Least Mean Square (LMS) algorithm, alternatively known as the Widrow-Hoff algorithm, is an example of supervised training based on a set of training examples.

$$\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}$$

- The LMS algorithm adjusts the weights and biases of the linear networks to minimize the mean square error (MSE)

$$MSE = \frac{1}{Q}\sum_{k=1}^{Q} e(k)^2 = \frac{1}{Q}\sum_{k=1}^{Q}\left(t(k) - a(k)\right)^2$$

- The LMS algorithm adjusts the weights and biases according to following equations

$$W(k + 1) = W(k) + 2\alpha e(k)p^T(k)$$

$$b(k + 1) = b(k) + 2\alpha e(k)$$

Linear Classification (train)

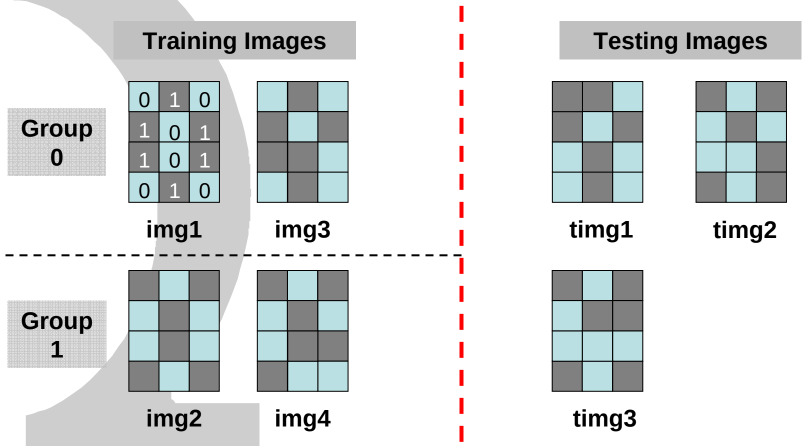
- Linear networks can be trained to perform linear classification with the function train.
- The train function applies each vector of a set of input vectors and calculates the network weight and bias increments due to each of the inputs according to the LMS (Widrow-Hoff) algorithm.
- The network is then adjusted with the sum of all these corrections.
- A pass through all input vectors is called an *epoch*.

Types of Neural Network

Example:

Let's re-visit Exercise 2: Pattern Classification of the Perceptrons.

We can build a Linear Network to perform not only pattern *classification* but also *association* tasks.



Types of Neural Network

Solution:

Command-line approach is demonstrated herein. Tne "nntool" GUI can be used alternatively.

```
% Define at the MATLAB® command window, the training inputs and targets
>> load train_images
>> p = [img1 img2 img3 img4];
>> t = targets;

% Create the linear network
>> net = newlin(minmax(p), 1);

% Train the linear network
>> net.trainParam.goal = 10e-5; % training stops if goal achieved
>> net.trainParam.epochs = 500; % training stops if epochs reached
>> net = train(net, p, t);

% Testing the performance of the trained linear network
>> a = sim(net, p)
>> a =
          -0.0136    0.9959    0.0137    1.0030
```

Types of Neural Network

TECHSOURCE www.techsource.com.my

% Comparing actual network output, a, with training targets, t:

```
>> a =
      -0.0136    0.9959    0.0137    1.0030
>> t =
            0         1         0         1
```

∴ **The actual network output, a, closely resembles that of target, t. It is because the output from Linear Network is not straightly 0 or 1, the output can be a range of values.**

% Now, test the Linear Network with 3 images not seen previously

```
>> load test_images
>> test1 = sim(net, timg1)
>> test1 =
            0.2271
>> test2 = sim(net, timg2)
>> test2 =
            0.9686
>> test3 = sim(net, timg3)
   test3 =
            0.8331
```


Types of Neural Network

TECHSOURCE www.techsource.com.my

How should we interpret the network outputs test1, test2 and test3? For that we need to define a Similarity Measure, S

$$S = |t - test|$$

Where t is the target-group (i.e. 0 or 1) and test is the network output when presented with test images.

The smaller the S is, the more similar is a test image to a particular group.

Similarity Measure, S		
test image	wrt. Group 0	wrt. Group 1
timg1	0.2271	0.7729
timg2	0.9686	0.0314
timg3	0.8331	0.1669

∴ timg1 belongs to Group 0 while timg2 and timg3 belong to Group 1.

These results are similar to what we obtained previously using Perceptron. By using Linear Network we have the added advantage of knowing how *similar* is it a test image is to the target group it belonged.

Types of Neural Network

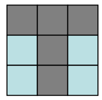
TECHSOURCE
www.techsource.com.my

Exercise 1: Simple Character Recognition

Create a Linear Network that will differentiate between a Letter 'U' and Letter 'T'. The Letter 'U' and 'T' are represented by a 3×3 matrices:

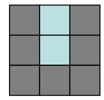
Group 0

T = [1 1 1
0 1 0
0 1 0]'



Group 1

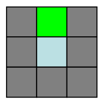
U = [1 0 1
1 0 1
1 1 1]'



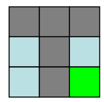
```
>> load train_letters
```

Test the trained Linear Network with following test images:

U_odd = [1 1 1
1 0 1
1 1 1]'



T_odd = [1 1 1
0 1 0
0 1 1]'



```
>> load test_letters
```


Types of Neural Network

TECHSOURCE
www.techsource.com.my

14

Solution:

Command-line approach is demonstrated herein. Tne "nntool" GUI can be used alternatively.

```
% Define at the MATLAB® command window, the training inputs and targets
>> load train_letters
>> p = [T U];
>> t = targets;

% Create the linear network
>> net = newlin(minmax(p), 1);

% Train the linear network
>> net.trainParam.goal = 10e-5; % training stops if goal achieved
>> net.trainParam.epochs = 500; % training stops if epochs reached
>> net = train(net, p, t);

% Testing the performance of the trained linear network
>> a = sim(net, p)
>> a =
        0.0262   0.9796
```

Types of Neural Network

TECHSOURCE
www.techsource.com.my
15

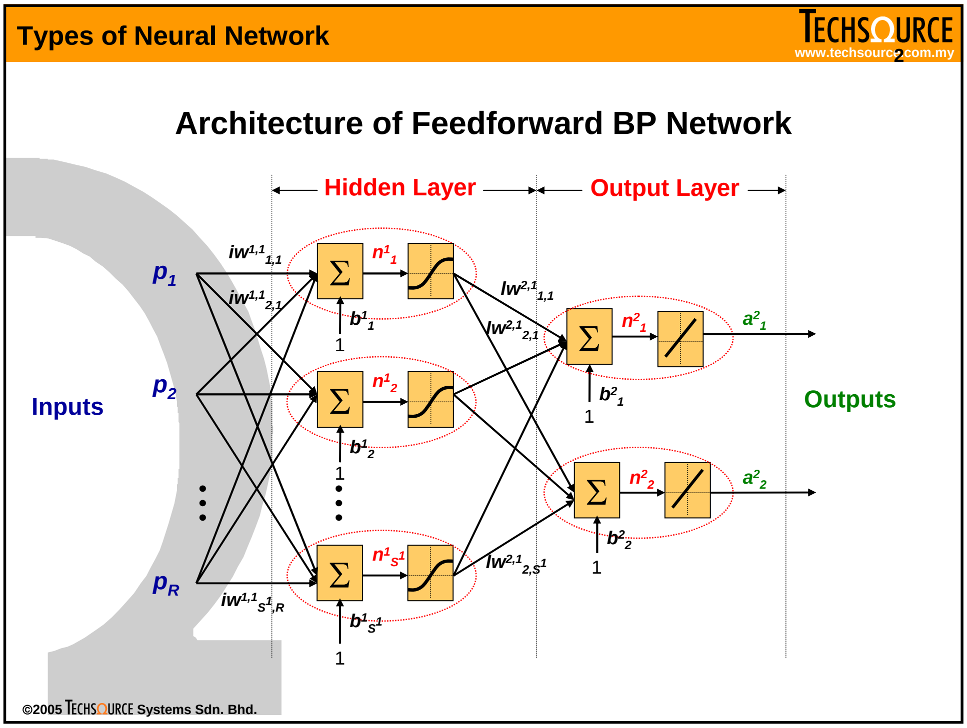
```
% Comparing actual network output, a, with training targets, t:
>> a =
       0.0262    0.9796
>> t =
            0          1

% Now, test the Linear Network with odd-shapes of T and U
>> load test_letters
>> test1 = sim(net, T_odd)
>> test1 =
          0.2066  % more similar to T
>> test2 = sim(net, U_odd)
>> test2 =
          0.8637  % more similar to U
```

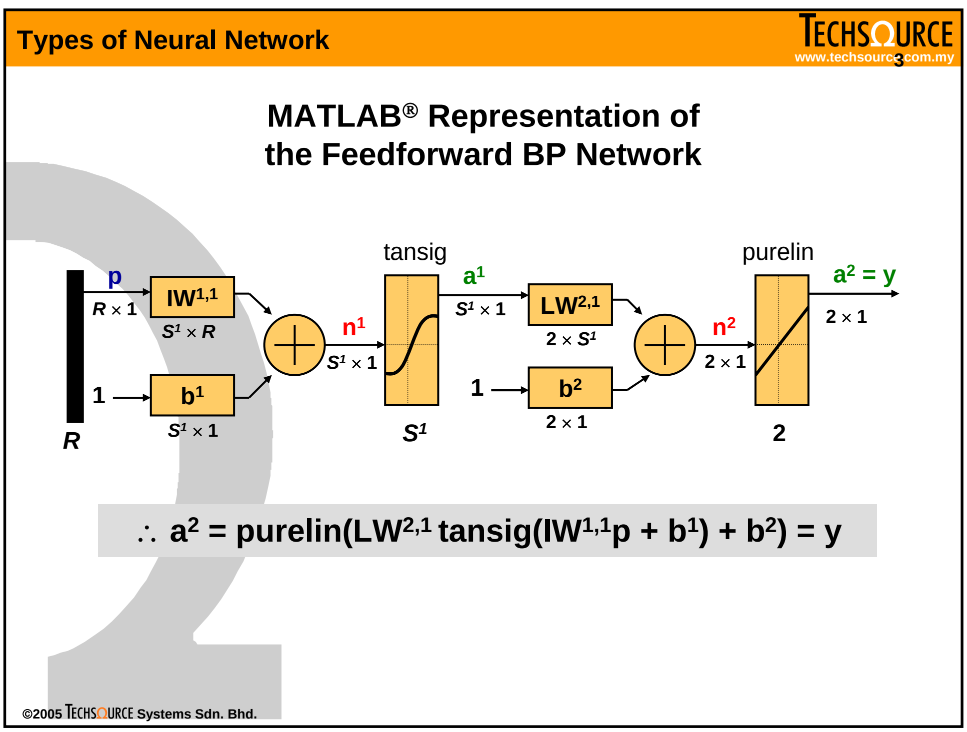

Types of Neural Network

TECHSOURCE
www.techsource.com.my
1

Backpropagation (BP) Networks

- Backpropagation network was created by generalizing the Widrow-Hoff learning rule to *multiple*-layer networks and *non-linear differentiable* transfer functions (TFs).
- Backpropagation network with biases, a sigmoid TF layer, and a linear TF output layer is capable of approximating *any* function.
- Weights and biases are updated using a variety of gradient descent algorithms. The gradient is determined by propagating the computation *backwards* from output layer to first hidden layer.
- If properly trained, the backpropagation network is able to generalize to produce reasonable outputs on inputs it has never "seen", as long as the new inputs are *similar* to the training inputs.

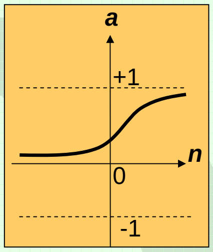

Types of Neural Network
TECHSOURCE
www.techsource.com.my
2
Architecture of Feedforward BP Network
Hidden Layer
Output Layer
Inputs
p1
p2
pR
iw1,1 1,1
iw1,1 2,1
iw1,1 S1,R
n1 1
n1 2
n1 S1
b1 1
b1 2
b1 S1
lw2,1 1,1
lw2,1 2,1
lw2,1 2,S1
n2 1
n2 2
b2 1
b2 2
a2 1
a2 2
Outputs
©2005 TECHSOURCE Systems Sdn. Bhd.


Types of Neural Network
TECHSOURCE
www.techsource.com.my
3
MATLAB® Representation of
the Feedforward BP Network
p
R × 1
IW1,1
S1 × R
1
b1
S1 × 1
R
n1
S1 × 1
tansig
S1
a1
S1 × 1
LW2,1
2 × S1
1
b2
2 × 1
n2
2 × 1
purelin
2
a2 = y
2 × 1
∴ a2 = purelin(LW2,1 tansig(IW1,1p + b1) + b2) = y
©2005 TECHSOURCE Systems Sdn. Bhd.

Types of Neural Network

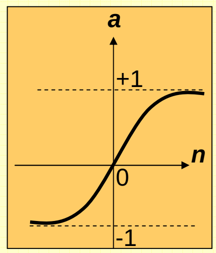
Transfer Functions for BP Networks

Log-Sigmoid



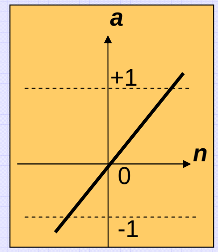
logsig(n) = 1 / (1 + exp(-n))

Tangent-Sigmoid



tansig(n) = 2/(1+exp(-2*n))-1

Linear



purelin(n) = n

Types of Neural Network

The Backpropagation Algorithm

- **The backpropagation algorithm is used to update the weights and biases of the neural networks. Full details of the algorithm is given in Appendix 1.**
- **The weights are updated according to following formulae:**

$$w_{j,i} \leftarrow w_{j,i} + \Delta w_{j,i} \quad \text{where,} \quad \Delta w_{j,i} = -\alpha \frac{\partial E}{\partial w_{j,i}} = -\alpha \delta_j x_{j,i}$$

For output neuron *k*,

$$\delta_k = -a_k \left(1 - a_k\right)\left(t_k - a_k\right)$$

For hidden neuron *h*,

$$\delta_h = a_h \left(1 - a_h\right) \sum_{k \in Downstream(h)} \delta_k w_{k,h}$$

Please see Appendix 1 for full derivation of the algorithm

Training of Backpropagation Networks

- There are generally four steps in the training process:
  1. Assemble the training data;
  2. Create the network object;
  3. Train the network;
  4. Simulate the network response to new inputs.
- The MATLAB® Neural Network Toolbox implements some of the most popular training algorithms, which encompass both original gradient-descent and faster training methods.

Batch Gradient Descent Training

- Batch Training: the weights and biases of the network are updated only after the entire training data has been applied to the network.
- Batch Gradient Descent (traingd):
  - Original but the slowest;
  - Weights and biases updated in the direction of the negative gradient (note: backprop. algorithm);
  - Selected by setting trainFcn to traingd:

```
net = newff(minmax(p), [3 1], {‘tansig’, ‘purelin’}, ‘traingd’);
```

Types of Neural Network TECHSOURCE www.techsource.com.my

Batch Gradient Descent with Momentum

- Batch Gradient Descent with Momentum (traingdm):
  - Faster convergence than traingd;
  - Momentum allows the network to respond not only the local gradient, but also to recent trends in the error surface;
  - Momentum allows the network to ignore small features in the error surface; without momentum a network may get stuck in a shallow local minimum.
  - Selected by setting trainFcn to traingdm:

```
net = newff(minmax(p), [3 1], {'tansig', 'purelin'}, 'traingdm');
```


Types of Neural Network TECHSOURCE www.techsource.com.my

Faster Training

- The MATLAB® Neural Network Toolbox also implements some of the faster training methods, in which the training can converge from ten to one hundred times faster than traingd and traingdm.
- These faster algorithms fall into two categories:
  1. Heuristic techniques: developed from the analysis of the performance of the standard gradient descent algorithm, e.g. traingda, traingdx and trainrp.
  2. Numerical optimization techniques: make use of the standard optimization techniques, e.g. conjugate gradient (traincgf, traincgb, traincgp, trainscg), quasi-Newton (trainbfg, trainoss), and Levenberg-Marquardt (trainlm).

Types of Neural Network

Comparison of Training Algorithms

Training Algorithms		Comments
traingd	Gradient Descent (GD)	Original but slowest
traingdm	GD with momentum	Faster than traingd
traingda	GD with adaptive α	Faster than traingd, but can use for batch mode only.
traingdx	GD with adaptive α and with momentum	
trainrp	Resilient Backpropagation	Fast convergence
traincgf	Fletcher-Reeves Update	Conjugate Gradient Algorithms with fast convergence
traincgp	Polak-Ribiére Update	
traincgb	Powell-Beale Restarts	
trainscg	Scaled Conjugate Gradient	
trainbfg	BFGS algorithm	Quasi-Newton Algorithms with fast convergence
trainoss	One Step Secant algorithm	
trainlm	Levenberg-Marquardt	Fastest training. Memory reduction features
trainbr	Bayesian regularization	Improve generalization capability

Types of Neural Network

Pre- and Post-Processing Features

Function	Description
premnmx	Normalize data to fall into range [-1 1].
postmnmx	Inverse of premnmx. Convert data back into original range of values.
tramnmx	Preprocess new inputs to networks that were trained with data normalized with premnmx.
prestd	Normalize data to have zero mean and unity standard deviation
poststd	Inverse of prestd. Convert data back into original range of values.
trastd	Preprocess new inputs to networks that were trained with data normalized with prestd.
prepca	Principal component analysis. Reduces dimension of input vector
trapca	Preprocess new inputs to networks that were trained with data transformed with prepca.
postreg	Linear regression between network outputs and targets. Use to determine adequacy of network fit.

Types of Neural Network

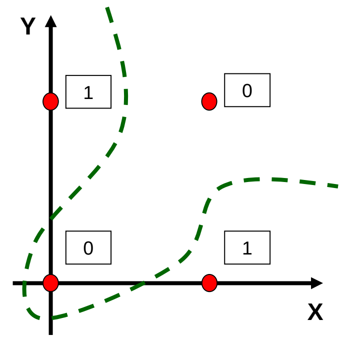
 12

Example: Modeling Logical XOR Function

The XOR Problem is highly non-linear, thereby cannot be solved using Perceptrons or Linear Networks. In this example, we will construct a simple backpropagation network to solve this problem.

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

p_1	p_2	a
0	0	0
0	1	0
1	0	0
1	1	1



Types of Neural Network

 13

Solution:

Command-line approach is demonstrated herein. Tne "nntool" GUI can be used alternatively.

```
% Define at the MATLAB® command window, the training inputs and targets
>> p = [0 0 1 1; 0 1 0 1];
>> t = [0 0 0 1];

% Create the backpropagation network
>> net = newff(minmax(p), [4 1], {'logsig', 'logsig'}, 'traingdx');

% Train the backpropagation network
>> net.trainParam.epochs = 500; % training stops if epochs reached
>> net.trainParam.show = 1; % plot the performance function at every epoch
>> net = train(net, p, t);

% Testing the performance of the trained backpropagation network
>> a = sim(net, p)
>> a =
          0.0002  0.0011  0.0001  0.9985
>>  t =
              0        0        0        1
```

Improving Generalization with Early Stopping

- The generalization capability can be improved with the *early stopping* feature available with the Neural Network toolbox.

- In this technique the available data is divided into three subsets:

  1. Training set
  2. Validation set
  3. Testing set

- The *early stopping* feature can be invoked when using the train command:

  [net, tr] = train(net, p, t, [ ], [ ], VV, TV)

  VV: Validation set structure; TV: Test set structure.

Example: Function Approximation with Early Stopping

```
% Define at the MATLAB® command window, the training inputs and targets
>> p = [-1: 0.05: 1];
>> t = sin(2*pi*p) + 0.1*randn(size(p));

% Construct Validation set
>> val.P = [-0.975: 0.05: 0.975]; % validation set must be in structure form
>> val.T = sin(2*pi*val.P) + 0.1*randn(size(val.P));

% Construct Test set (optional)
>> test.P = [-1.025: 0.05: 1.025]; % validation set must be in structure form
>> test.T = sin(2*pi*test.P) + 0.1*randn(size(test.P));

% Plot and compare three data sets
>> plot(p, t), hold on, plot(val.P, val.T,‘r:*’), hold on, plot(test.P, test.T, ‘k:^’);
>> legend(‘train’, ‘validate’, ‘test’);

% Create a 1-20-1 backpropagation network with ‘trainlm’ algorithm
>> net = newff(minmax(p), [20 1], {‘tansig’, ‘purelin’}, ‘trainlm’);
```

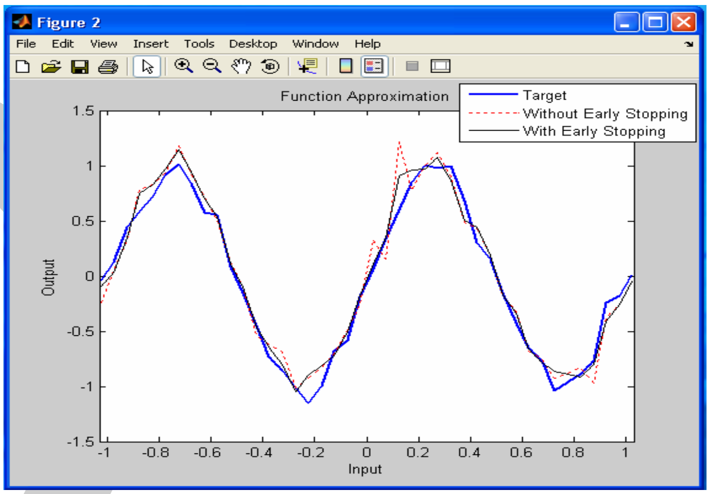
Types of Neural Network

TECHSOURCE www.techsource.com.my 16

```
>> net.trainParam.show = 1;
>> net.trainParam.epochs = 300;

% First, train the network without early stopping
>> net = init(net); % initialize the network
>> [net, tr] = train(net, p, t);
>> net1 = net; % network without early stopping

% Then, train the network with early stopping with both Validation & Test sets
>> net = init(net);
>> [net, tr] = train(net, p, t, [], [], val, test);
>> net2 = net; % network with early stopping

% Test the modeling performance of net1 & net2 on Test sets
>> a1 = sim(net1, test.P); % simulate the response of net1
>> a2 = sim(net2, test.P); % simulate the response of net2
>> figure, plot(test.P, test.T), xlim([-1.03 1.03]), hold on
>> plot(test.P, a1, 'r'), hold on, plot(test.P, a2, 'k');
>> legend('Target', 'Without Early Stopping', 'With Early Stopping');
```


Types of Neural Network

TECHSOURCE www.techsource.com.my 17

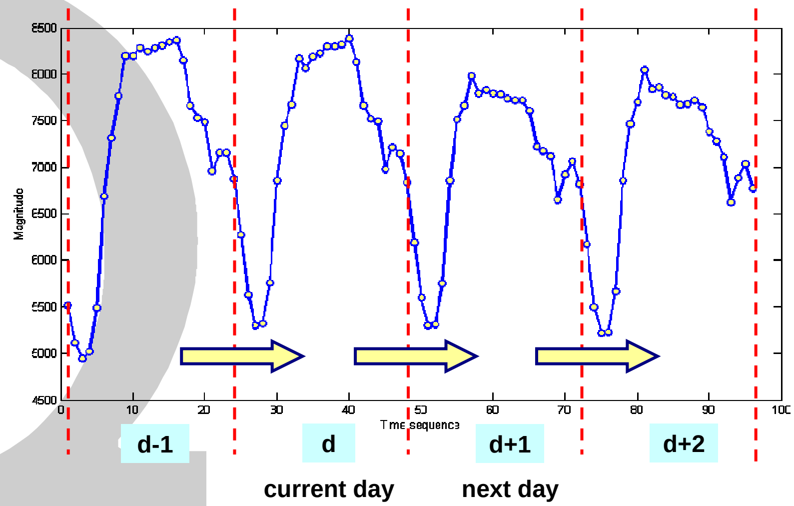


∴Network with *early stopping* can better fit the Test data set with less discrepancies, therefore the *early stopping* feature can be used to prevent *overfitting* of network towards the training data.

Types of Neural Network

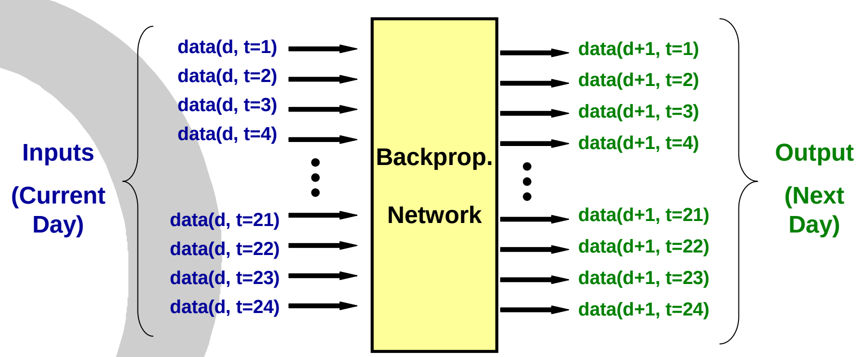
Exercise 1: Time-Series Prediction

Create a Neural Network that can predict the next-day 24-hour time-series based on current-day 24-hour time-series.



Types of Neural Network

The Neural Network structure is as follow:



Types of Neural Network

Network details:

- **Architecture: 24-48-24 network, with tansig TF and purelin TF in hidden and output layer respectively.**
- **Training: trainlm algorithm with 7 epochs and plot the performance function every 1 epoch.**

Data details:

- **Load timeseries.mat into MATLAB® workspace.**
- **Training Data (1st to 37th days): TrainIp (inputs), TrainTgt (targets)**
- **Testing Data (38th to 40th days): TestIp (query inputs), TestTgt (actual values)**

Hint: use pre- and post-processing functions premnmx, tramnmx, postmnmx to have more efficient training.

Types of Neural Network

Solution:

```
% Load the Time Series data into MATLAB® Workspace
>> load timeseries

% Prepare the data for the network training
>> [PN, minp, maxp, TN, mint, maxt] = premnmx(TrainIp, TrainTgt);

% Create the backpropagation network
>> net = newff(minmax(PN), [48 24], {'tansig', 'purelin'}, 'trainlm');
>> net.trainParam.epochs = 7;
>> net.trainParam.show = 1;

% Training the neural network
>> [net, tr] = train(net, PN, TN);

% Prepare the data for testing the network (predicting 38th to 40th days)
>> PN_Test = tramnmx(TestIp,minp,maxp);

% Testing the neural network
>> TN_Test = sim(net, PN_Test);
```

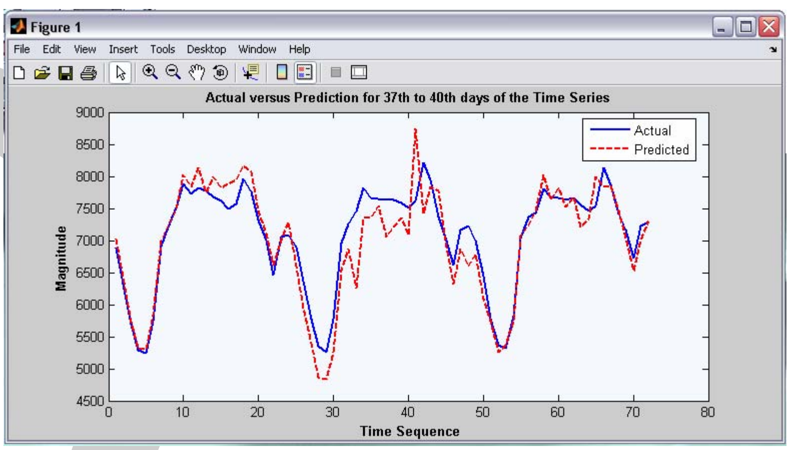
Types of Neural Network

22

```
% Convert the testing output into prediction values for comparison purpose
>> [queryInputs predictOutputs] =  postmnmx(PN_Test, minp, maxp, …
TN_Test, mint, maxt);

% Plot and compare the predicted and actual time series
>> predictedData = reshape(predictOutputs, 1, 72);
>> actualData = reshape(TestTgt, 1, 72);
>> plot(actualData, '-*'), hold on
>> plot(predictOutputs, 'r:');
```


Types of Neural Network

23



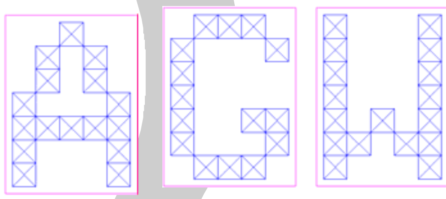
Homework: Try to subdivide the training data [TrainIp TrainTgt] into Training & Validation Sets to accertain whether the use of *early stopping* would improve the prediction accuracy.

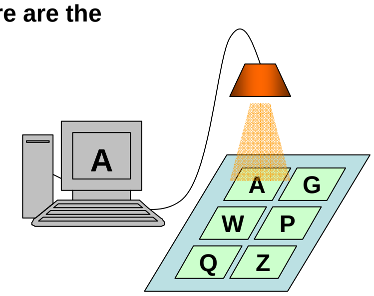
Types of Neural Network

TECHSOURCE www.techsource.com.my 24

Exercise 2: Character Recognition

Create a Neural Network that can recognize 26 letters of the alphabet. An imaging system that digitizes each letter centered in the system field of vision is available. The result is each letter represented as a 7 by 5 grid of boolean values. For example, here are the Letters A, G and W:

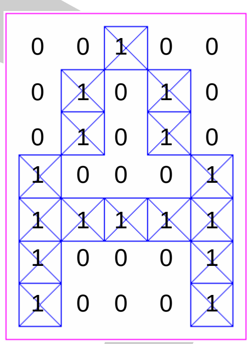




Types of Neural Network

TECHSOURCE www.techsource.com.my 25

- **For each alphabetical letter, create a 35-by-1 input vector containing boolean values of 1's and 0's. Example for Letter A:**



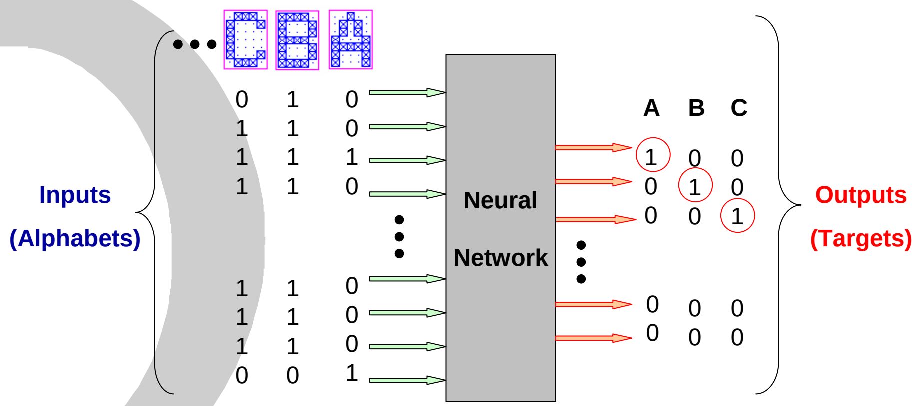
**Letter A = [0 0 1 0 0 ...
0 1 0 1 0 ...
0 1 0 1 0 ...
1 0 0 0 1 ...
1 1 1 1 1 ...
1 0 0 0 1 ...
1 0 0 0 1]';**

**Target A = [1 0 0 0 0 ...
0 0 0 0 0 ...
0 0 0 0 0 ...
0 0 0 0 0 ...
0 0 0 0 0 ...
0]';**

- **Corresponding Output is a 26-by-1 vector, indicating a 1 (i.e. TRUE) at the correct alphabetical sequence. Example for Target A:**

Types of Neural Network

The Neural Network structure is as follow:



Types of Neural Network

Network details:

- **Architecture: 35-10-26 network, with logsig TFs in hidden and output layers.**
- **Training: traingdx algorithm with 500 epochs and plot the performance function every 1 epoch. Performance goal is 0.001.**

Data details:

- **Load training inputs and targets into workspace by typing**

  [alphabets, targets] = prprob;

Types of Neural Network

TECHSOURCE www.techsource.com.my 28

Solution:

```
% Load the training data into MATLAB® Workspace
>> [alphabets, targets] = prprob;

% Create the backpropagation network
>> net = newff(minmax(alphabets), [10 26], {'logsig', 'logsig'}, 'traingdx');
>> net.trainParam.epochs = 500;
>> net.trainParam.show = 1;
>> net.trainParam.goal = 0.001;

% Training the neural network
>> [net, tr] = train(net, alphabets, targets);

% First, we create a normal 'J' to test the network performance
>> J = alphabets(:,10);
>> figure, plotchar(J);
>> output = sim(net, J);
>> output = compet(output) % change the largest values to 1, the rest 0s
>> answer = find(compet(output) == 1); % find the index (out of 26) of network output
>> figure, plotchar(alphabets(:, answer));
```


Types of Neural Network

TECHSOURCE www.techsource.com.my 29

```
% Next, we create a noisy 'J' to test the network can still identify it correctly…
>> noisyJ = alphabets(:,10)+randn(35,1)*0.2;
>> figure; plotchar(noisyJ);
>> output2 = sim(network1, noisyJ);
>> output2 = compet(output2);
>> answer2 = find(compet(output2) == 1);
>> figure; plotchar(alphabets(:,answer2));
```

Types of Neural Network

www.techsource.com.my

Self-Organizing Maps

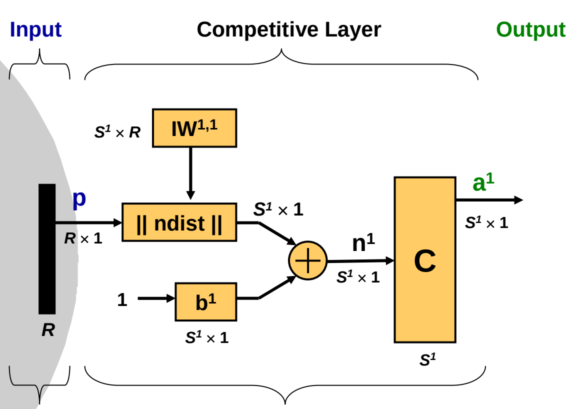
- **Self-organizing in networks is one of the most fascinating topics in the neural network field. Such networks can learn to detect *regularities* and *correlations* in their input and adapt their future responses to that input accordingly.**

- **The neurons of competitive networks learn to recognize groups of similar input vectors. Self-organizing maps learn to recognize groups of similar input vectors in such a way that neurons physically near each other in the neuron layer respond to similar input vectors.**

Types of Neural Network

www.techsource.com.my

Competitive Learning



Types of Neural Network

TECHSOURCE www.techsource.com.my 3

Learning Algorithms for Competitive Network

- **The weights of the winning neuron (represented by a row of the input weight matrix) are adjusted with the *Kohonen learning* rule (learnk).**
- **Supposed that the i^{th} neuron wins, the elements of the i^{th} row of the input weight matrix are adjusted according to following formula:**

$$ {}_i\mathrm{IW}^{1,1}(q) = {}_i\mathrm{IW}^{1,1}(q-1) + \alpha(\mathrm{p}(q) - {}_i\mathrm{IW}^{1,1}(q-1)) $$

- **One of the limitations of the competitive networks is that some neurons may *never* wins because their weights are *far* from any input vectors.**
- **The bias learning rule (learncon) is used to allow every neuron in the network learning from the input vectors.**

Types of Neural Network

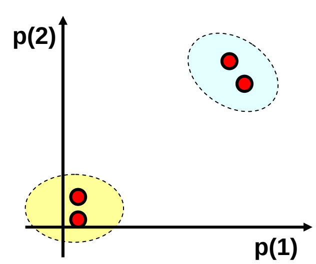
TECHSOURCE www.techsource.com.my 4

Example: Classification using Competitive Network

We can use a competitive network to classify input vectors without any learning targets.

Let's say if we create four two-element input vectors, with two very close to (0 0) and others close to (1 1).

p = [0.1 0.8 0.1 0.9
0.2 0.9 0.1 0.8];



Let's see whether the competitive network is able to identify the classification structure…

Types of Neural Network

TECHSOURCE www.techsource.com.my 5

```
% Define at the MATLAB® command window, the four two-element vectors
>> p = [0.1 0.8 0.1 0.9; 0.2 0.9 0.1 0.8];

% Create a two-neuron competitive network
>> net = newc([0 1; 0 1], 2);

% The weights are initialized to the center of input ranges with 'midpoint' fcn
>>net.IW{1,1}
ans =
        0.5    0.5
        0.5    0.5

% The biases are computed by 'initcon' fcn, which gives
>> net.b{1}
ans =
        5.4366
        5.4366

% Let's train the competitive network for 500 epochs
>> net.trainParam.epochs = 500;
>> net = train(net, p);
```


Types of Neural Network

TECHSOURCE www.techsource.com.my 6

```
% Simulate the network with input vectors again
>> a  = sim(net, p)
>> ac = vec2ind(a)
>> ac =
          2    1    2    1
```

∴The network is able to classfy the input vectors into two classess, those who close to (1,1), class 1 and those close to origin (0,0), class 2. If we look at the adjusted weights,

```
>> net.IW{1,1}
ans =
        0.8500    0.8500
        0.1000    0.1501
```

∴Note that the first-row weight vector (associated with 1st neuron) is near to input vectors close to (1,1), which the second-row weight vector (associated with 2nd neuron) is near to input vectors close to (0,0).

Types of Neural Network

Exercise 1: Classification of Input Vectors:

Graphical Example

First, generate the input vectors by using the built-in nngenc function:

```
>> X = [0 1; 0 1];   % Cluster centers to be in these bounds
>> clusters = 8;     % Number of clusters
>> points = 10;      % Number of points in each cluster
>> std_dev = 0.05;   % Standard deviation of each cluster
>> P = nngenc(X,clusters,points,std_dev);  % Number of clusters
```

Plot and show the generated clusters

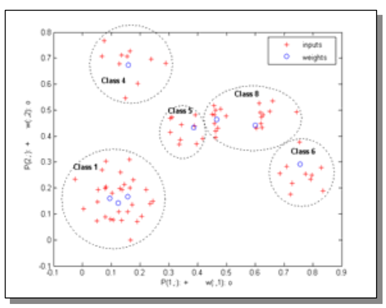
```
>> plot(P(1,:),P(2,:),'+r');
>> title('Input Vectors');
>> xlabel('p(1)');
>> ylabel('p(2)');
```

Try to build a competitive network with 8 neurons and train for 1000 epochs. Superimpose the trained network weights onto the same figure. Try to experiement with the number of neurons and conclude on the accuracy of the classification.

Types of Neural Network

Solution:

```
% Create and train the competitive network
>> net = newc([0 1; 0 1], 8, 0.1);  % Learning rate is set to 0.1
>> net.trainParam.epochs = 1000;
>> net = train(net, P);

% Plot and compare the input vectors and cluster centres determined by the
competitive network
>> w = net.IW{1,1};
>> figure, plot(P(1,:),P(2,:),'+r');
>> hold on, plot(w(:,1), w(:,2), 'ob');

% Simulate the trained network to new inputs
>> t1 = [0.1; 0.1], t2 = [0.35; 0.4], t3 = [0.8; 0.2];
>> a1 = sim(net, [t1 t2 t3]);
>> ac1 = vec2ind(a1);
ac1 =
        1     5     6
```



Homework: Try altering the number of neurons in the competitive layer and observe how it affects the cluster centres.

Types of Neural Network

TECHSOURCE www.techsource.com.my 9

Self-Organizing Maps

- **Similar to competitive neural networks, self-organizing maps (SOMs) can learn the *distribution* of the input vectors. The distinction between these two networks is that the SOM can also learn the *topology* of the input vectors.**
- **However, instead of updating the weight of the winning neuron i^*, all neurons within a certain neighborhood $N_{i^*}(d)$ of the winning neuron are also updated using the Kohonen learning learnsom, as follows:**

$$ {}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) $$

- **The neighborhood $N_{i^*}(d)$ contains the indices for all the neurons that lie within a radius d of the winning neuron i^*.**

$$ N_i(d) = \{j, d_{ij} \le d\} $$

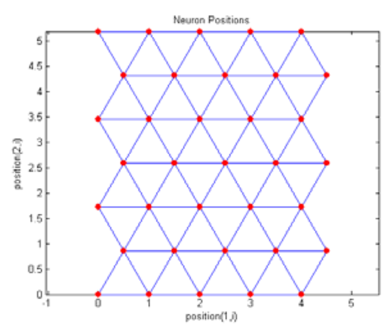
Types of Neural Network

TECHSOURCE www.techsource.com.my 10

Topologies & Distance Functions

- **Three different types of topology can be specified for the initial location of the neurons:**
  1. **Rectangular grid: gridtop**
  2. **Hexagonal grid: hextop**
  3. **Random grid: randtop**
- **For example, to create a 5-by-7 hexagonal grid,**

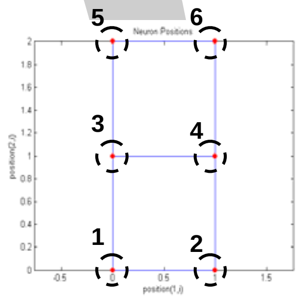
```
>> pos = hextop(5,7);
>> plotsom(pos);
```



Types of Neural Network

- Similarly, there are four different ways of calculating the distance from a particular neuron to its neighbors:
    1. Euclidean distance: dist
    2. Box distance: boxdist
    3. Link distance: linkdist
    4. Manhattan distance: mandist

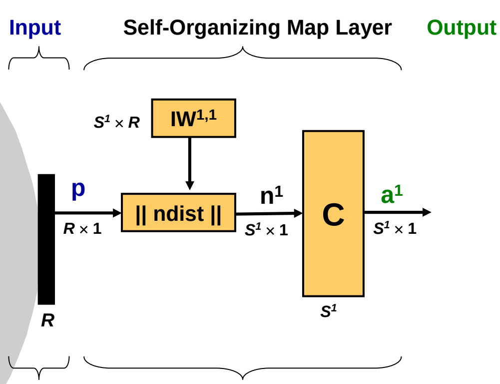
- For example, with the 2-by-3 rectangular grid shown below,




```
>> d = boxdist(pos)
d =
     0     1     1     1     2     2
     1     0     1     1     2     2
     1     1     0     1     1     1
     1     1     1     0     1     1
     2     2     1     1     0     1
     2     2     1     1     1     0
```


Types of Neural Network

The SOM Architecture



$n_i^1 = -\| {}_iIW^{1,1} - p \|$

$a^1 = compet(n^1)$

Types of Neural Network

TECHSOURCE www.techsource.com.my 13

Creating and Training the SOM

- Let's load an input vector into MATLAB® workspace

```
>> load somdata
>> plot(P(1,:), P(2,:), 'g.', 'markersize', 20), hold on
```

- Create a 2-by-3 SOM with following command, and superimpose the initial weights onto the input space

```
>> net = newsom([0 2; 0 1], [2 3]);
>> plotsom(net.iw{1,1}, net.layers{1}.distances), hold off
```

- The weights of the SOM are updated using the learnsom function, where the winning neuron's weights are updated proportional to α and the weights of neurons in its neighbourhood are altered proportional to ½ of α.

Types of Neural Network

TECHSOURCE www.techsource.com.my 14

- The training is divided into two phases:
  1. Ordering phase: The neighborhood distance starts as the maximum distance between two neurons, and decreases to the tuning neighborhood distance. The learning rate starts at the ordering-phase learning rate and decreases until it reaches the tuning-phase learning rate. This phase typically allows the SOM to learn the *topology* of the input space.
  2. Tuning Phase: The neighborhood distance stays at the tuning neighborhood distance (i.e., typically 1.0). The learning rate continues to decrease from the tuning phase learning rate, but very slowly. The small neighborhood and slowly decreasing learning rate allows the SOM to learn the *distribution* of the input space. The number of epochs for this phase should be much larger than the number of steps in the ordering phase.

Types of Neural Network

15

- **The learning parameters for both phases of training are,**

```
>> net.inputWeights{1,1}.learnParam
ans =
         order_lr: 0.9000
      order_steps: 1000
          tune_lr: 0.0200
          tune_nd: 1
```

- **Train the SOM for 1000 epochs with**

```
>> net.trainParam.epochs = 1000;
>> net = train(net, P);
```

- **Superimpose the trained network structure onto the input space**

```
>> plot(P(1,:), P(2,:), 'g.', 'markersize', 20), hold on
>> plotsom(net.iw{1,1}, net.layers{1}.distances), hold off
```

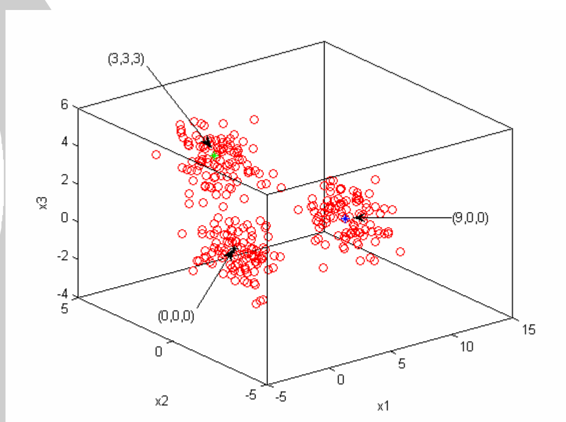
Try alter the size of the SOM and learning parameters and draw conclusion on how it affects the result.

Types of Neural Network

16

Exercise 2: Mapping of Input Space

In this exercise we will test whether the SOM can map out the *topology* and *distribution* of an input space containing three clusters illustrated in the figure below,



Types of Neural Network

TECHSOURCE www.techsource.com.my 17

Solution:

```
% Let's start by creating the data for the input space illustrated previously

>> d1 = randn(3,100); % cluster center at (0, 0, 0)
>> d2 = randn(3, 100) + 3; % cluster center at (3, 3, 3)
>> d3 = randn(3,100), d3(1,:) = d3(1,:) +  9; % cluster center at (9, 0, 0)
>> d = [d1 d2 d3];

% Plot and show the generated clusters

>> plot3(d(1,:), d(2,:), d(3,:), 'ko'), hold on, box on

% Try to build a 10-by-10 SOM and train it for 1000 epochs,

>> net = newsom(minmax(d), [10 10]);
>> net.trainParam.epochs = 1000;
>> net = train(net, d);

% Superimpose the trained SOM's weights onto the input space,
>> gcf, plotsom(net.IW{1,1}, net.layers{1}.distances), hold off

% Simulate SOM to get indices of neurons closest to input vectors
>> A = sim(net, d), A = vec2ind(A);
```


Types of Neural Network

TECHSOURCE www.techsource.com.my

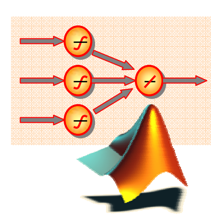
Section Summary:

1. Perceptrons
- Introduction
- The perceptron architecture
- Training of perceptrons
- Application examples

2. Linear Networks
- Introduction
- Architecture of linear networks
- The Widrow-Hoff learning algorithm
- Application examples

3. Backpropagation Networks
- Introduction
- Architecture of backprogation network
- The backpropagation algorithm
- Training algorithms
- Pre- and post-processing
- Application examples

4. Self-Organizing Maps
- Introduction
- Competitive learning
- Self-organizing maps
- Application examples

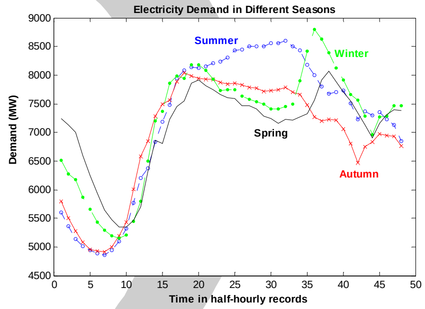


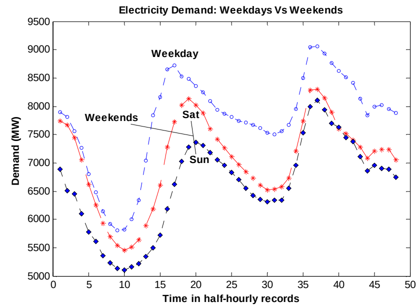
Case Study

www.techsource.com.my

Predicting the Future Time Series Data

The demand for electricity (in MW) varies according to seasonal changes and weekday-weekend work cycle. How do we develop a neural-network based Decision-Support System to forecast the next-day hourly demand?





Note: NSW electricity demand data (1996 – 1998) courtesy of NEMMCO, Australia

Case Study

www.techsource.com.my

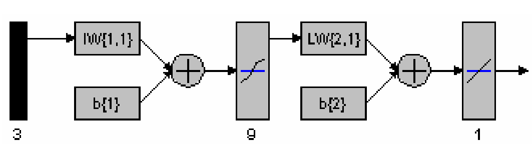
Solution:

Step 1: Formulating Inputs and Outputs of Neural Network

By analysing the time-series data, a 3-input and 1-output neural network is proposed to predict next-day hourly electricity demand,

Inputs:
p1 = L(d,t)
p2 = L(d,t) - L(d-1,t)
p3 = Lm(d+1,t) - Lm(d,t)

Output:
a1 = L(d+1, t)



Where,
L(d, t): Electricity demand for day, d, and hour, t
L(d+1, t): Electricity demand for next day, (d+1), and hour, t
L(d-1, t): Electricity demand for previous day, (d-1), and hour t
Lm(a, b) = ½ [L(a-k, b) + L(a-2k, b)]
k = 5 for Weekdays Model & k = 2 for Weekends Model

Case Study

Step 2: Pre-process Time Series Data to Appropriate Format

The time-series data is in MS Excel format and was date- and time-tagged. We need to preprocess the data according to following steps:

1. **Read from the MS Excel file [`histDataIp.m`]**
2. **Divide the data into weekdays and weekends [`divideDay.m`]**
3. **Remove any outliers from the data [`outlierRemove.m`]**

Step 3: Constructing Inputs and Output Data for Neural Network

Arrange the processed data in accordance to neural-network format:

1. **Construct Input-Output pair [`nextDayIp.m`]**
2. **Normalizing the training data for faster learning [`processIp.m`]**

Step 4: Training & Testing the Neural Network

Train the neural network using command-line or NNTOOL. When training is completed, proceed to test the robustness of the network against "unseen" data.

The End

Kindly return your Evaluation Form

Tel: 603 – 8076 1953 Fax: 603 – 8076 1954
Email: info@techsource.com.my Web: www.techsource.com.my
Tech-Support: techsupport@techsource.com.my