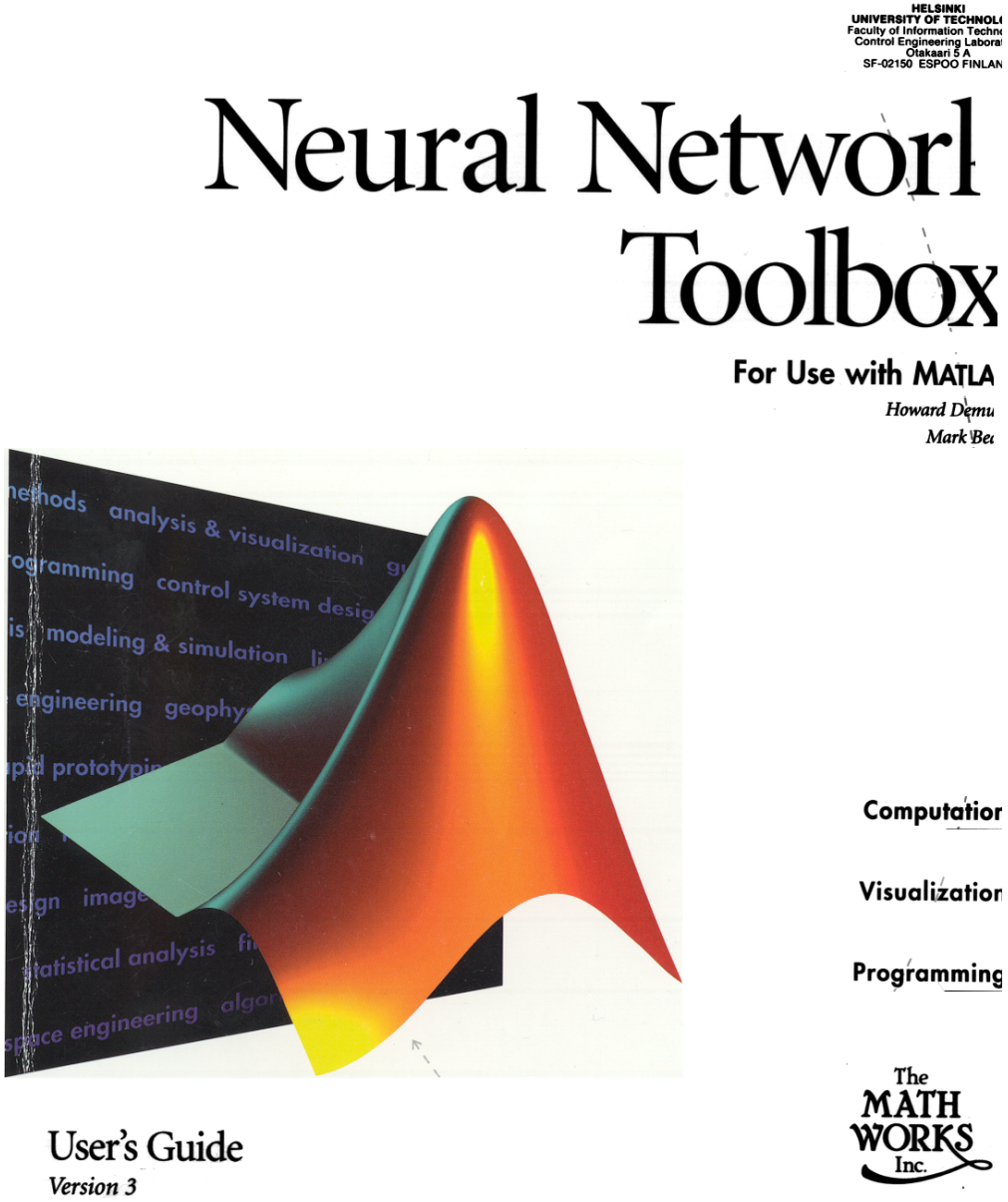
Neural Network Toolbox in MATLAB

2010

Neural Network Toolbox in MATLAB



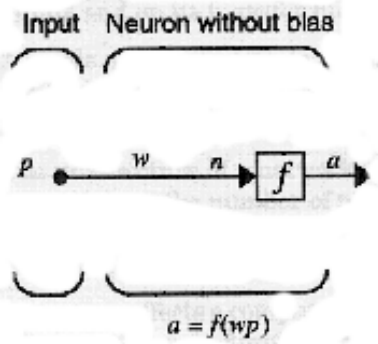
2

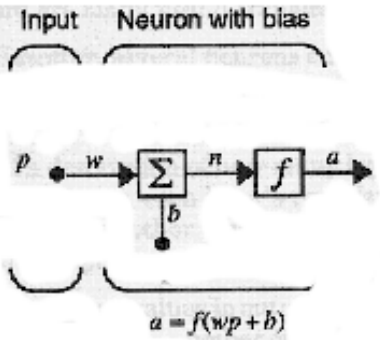
Neuron Model and Network Architectures

Neural Network Toolbox in MATLAB

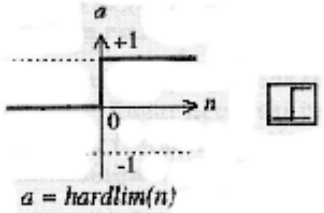
Figures and Equations

Simple Neuron



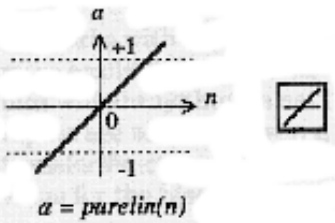


Hard Limit Transfer Function



Hard Limit Transfer Function

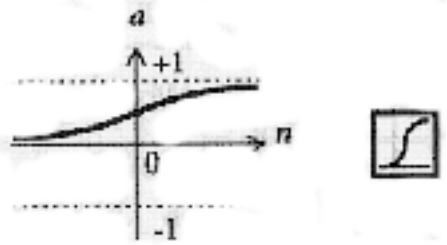
Purelin Transfer Function



Linear Transfer Function

Neural Network Toolbox in MATLAB

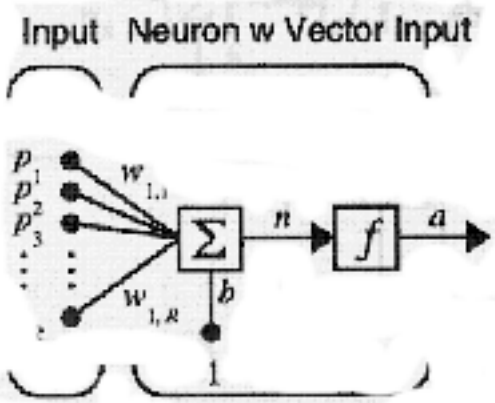
Log Sigmoid Transfer Function



$a = logsig(n)$

Log-Sigmoid Transfer Function

Neuron With Vector Input



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

Where...

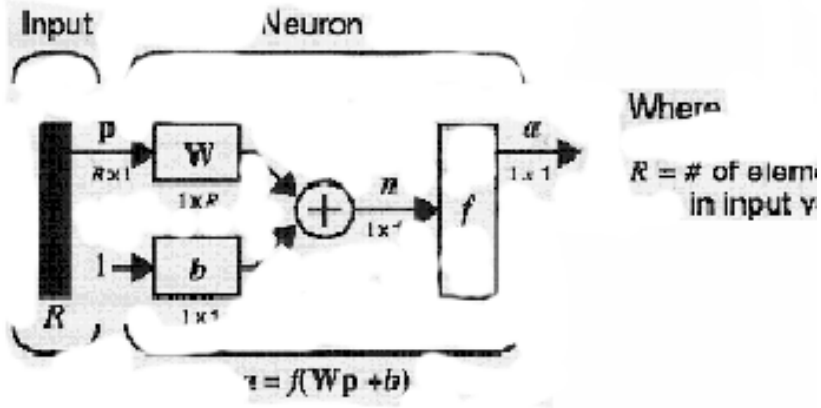
R = # Elements in input vector

Net Input

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b$$

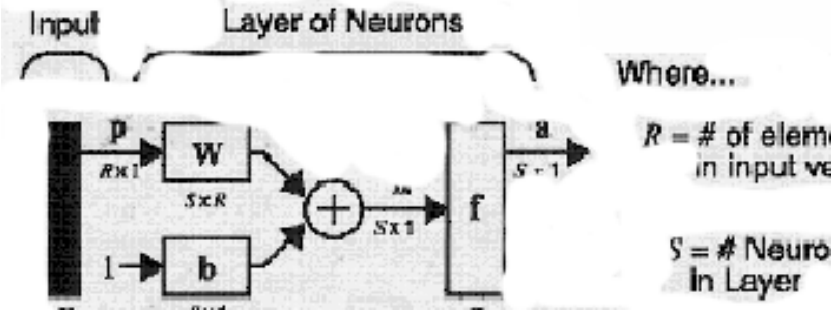
Neural Network Toolbox in MATLAB

Single Neuron Using Abbreviated Notation



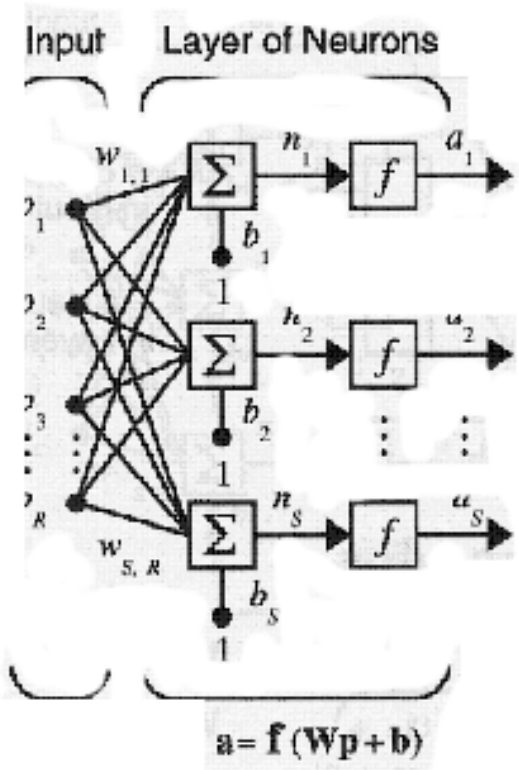
Icons for Transfer Functions

Layer of Neurons



Neural Network Toolbox in MATLAB

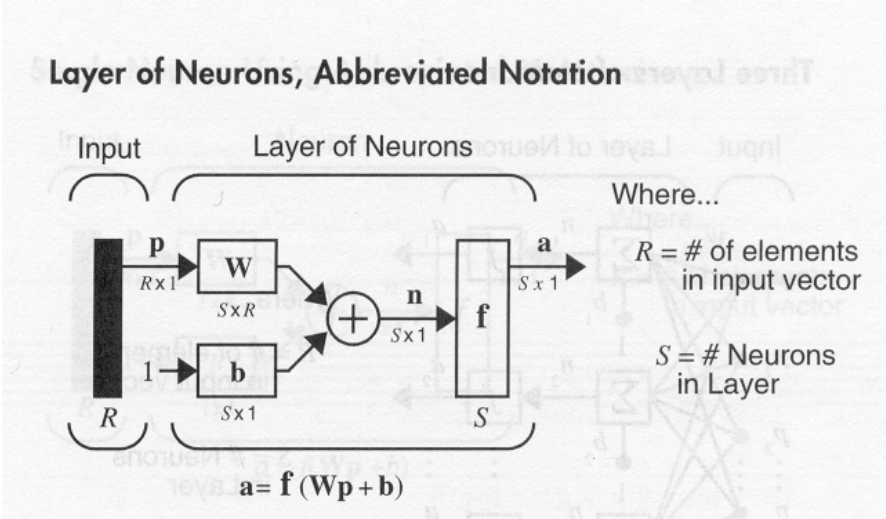
Three Layers of Neurons

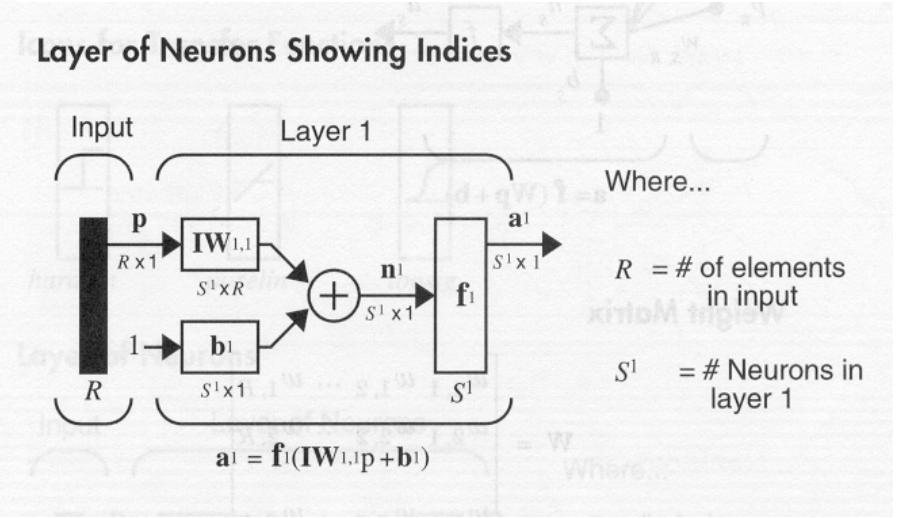


Weight Matrix

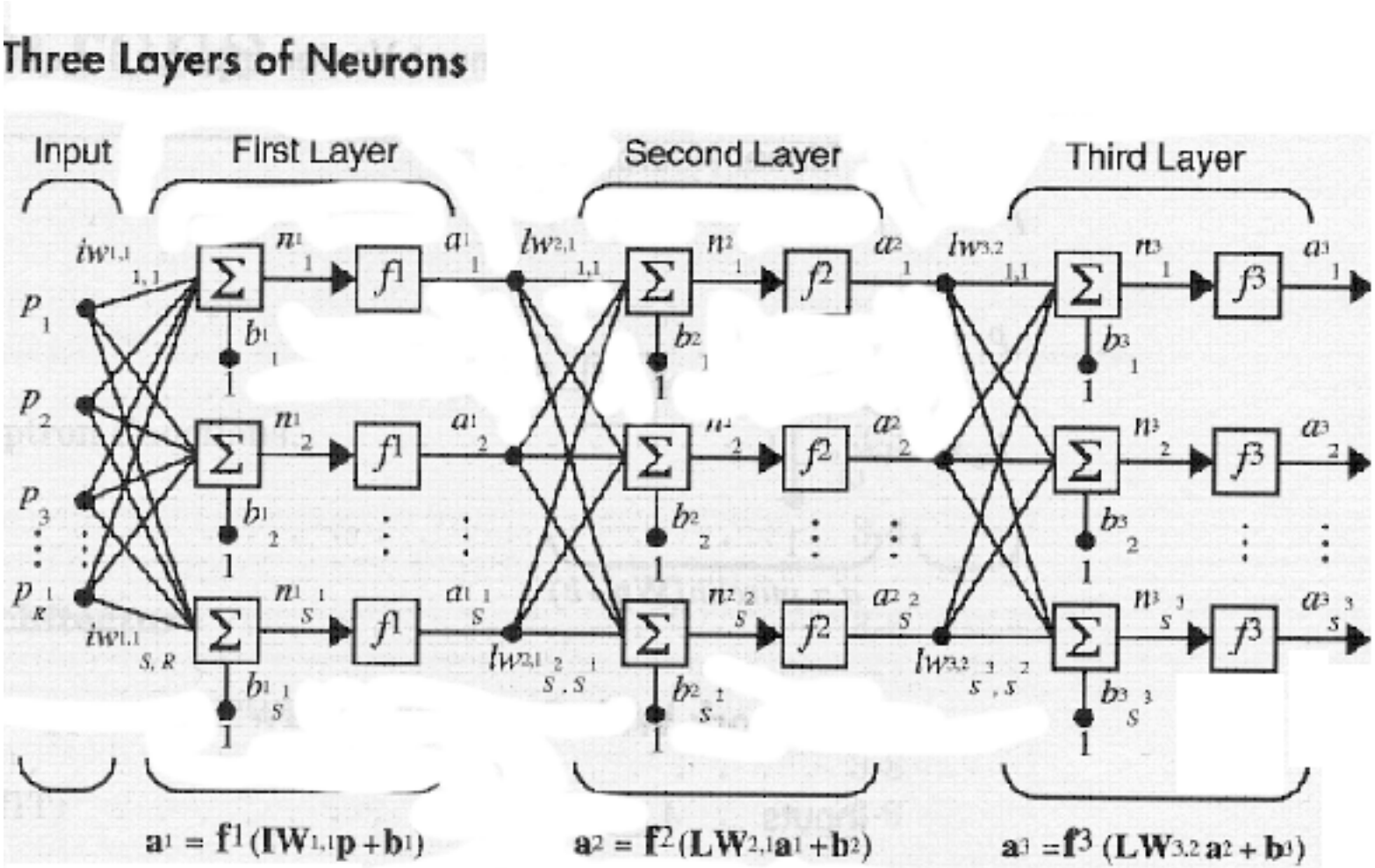
$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ & & & \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

Neural Network Toolbox in MATLAB

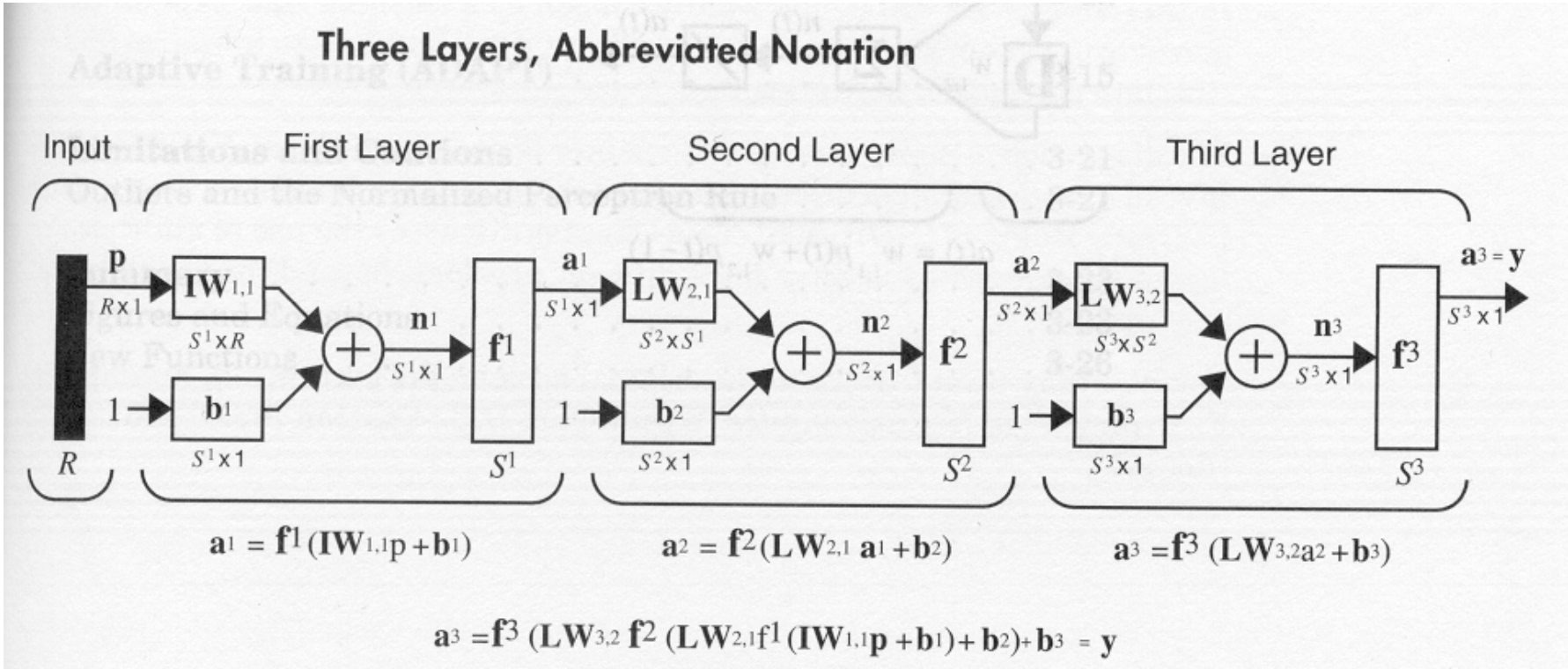




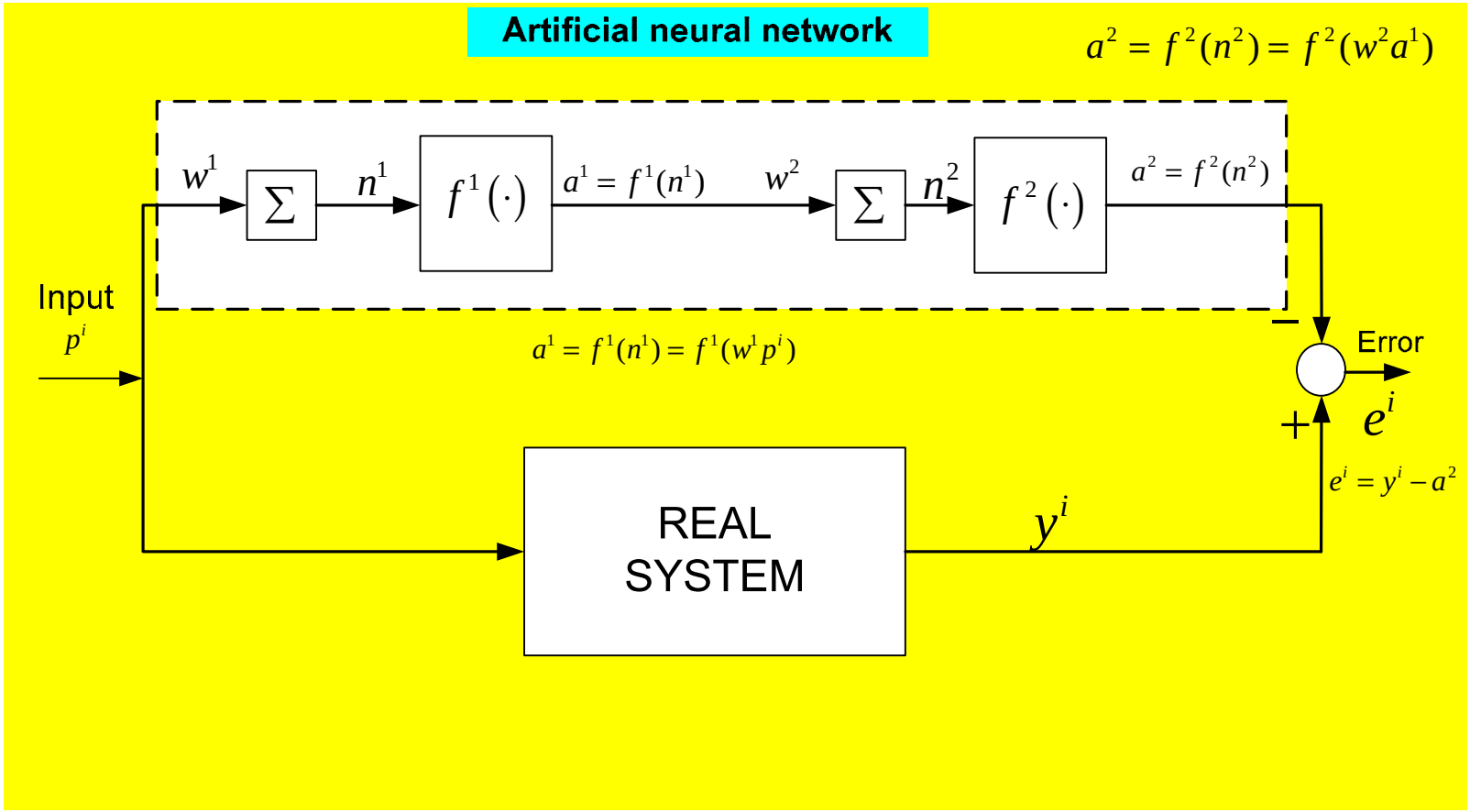
Neural Network Toolbox in MATLAB



Neural Network Toolbox in MATLAB



Backpropagation algorithm – Simple network



Choose activation functions f^1 and f^2

determine weights w^1 and w^2

$$a^2 = f^2(w^2 f^1(w^1 p^i))$$

Backpropagation algorithm – Simple network

Cost function

$$J = \frac{1}{2}(y^i - f^2(\underbrace{w_k^2 a^1}_{n^2}))^2$$

Apply gradient algorithm
to the last layer

$$w_{k+1}^2 = w_k^2 - \alpha_k \left(\frac{\partial J}{\partial w^2} \right)\Big|_{w_k^2} = w_k^2 - \alpha_k \left((y^i - f^2(w_k^2 a^1)) \left(-\frac{\partial f^2}{\partial n^2} \right) a^1 \right)\Big|_{w_k^2}$$

$$w_{k+1}^2 = w_k^2 - \alpha_k \left\{ \underbrace{(y^i - f^2(w_k^2 a^1)) \left(-\frac{\partial f^2}{\partial n^2} \right)}_{\delta_{oi}} \Big|_{w_k^2} \quad a^1 \right\}$$

$$= w_k^2 - \alpha_k \delta_{oi} a^1$$

Backpropagation algorithm – Simple network

Cost function

$$J = \frac{1}{2}(y^i - f^2(\underbrace{w_k^2 a^1}_{n^2}))^2 = \frac{1}{2}(y^i - f^2(w_k^2 f^1(n^1)))^2$$

$$= \frac{1}{2}(y^i - f^2(w_k^2 \underbrace{f^1(\overbrace{w^1 p^i}^{n^1})}_{a^1}))^2$$

Apply gradient algorithm to the next layer

$$w_{k+1}^1 = w_k^1 - \alpha_k \left(\frac{\partial J}{\partial w^1} \right) \Bigg|_{w_k^1}$$

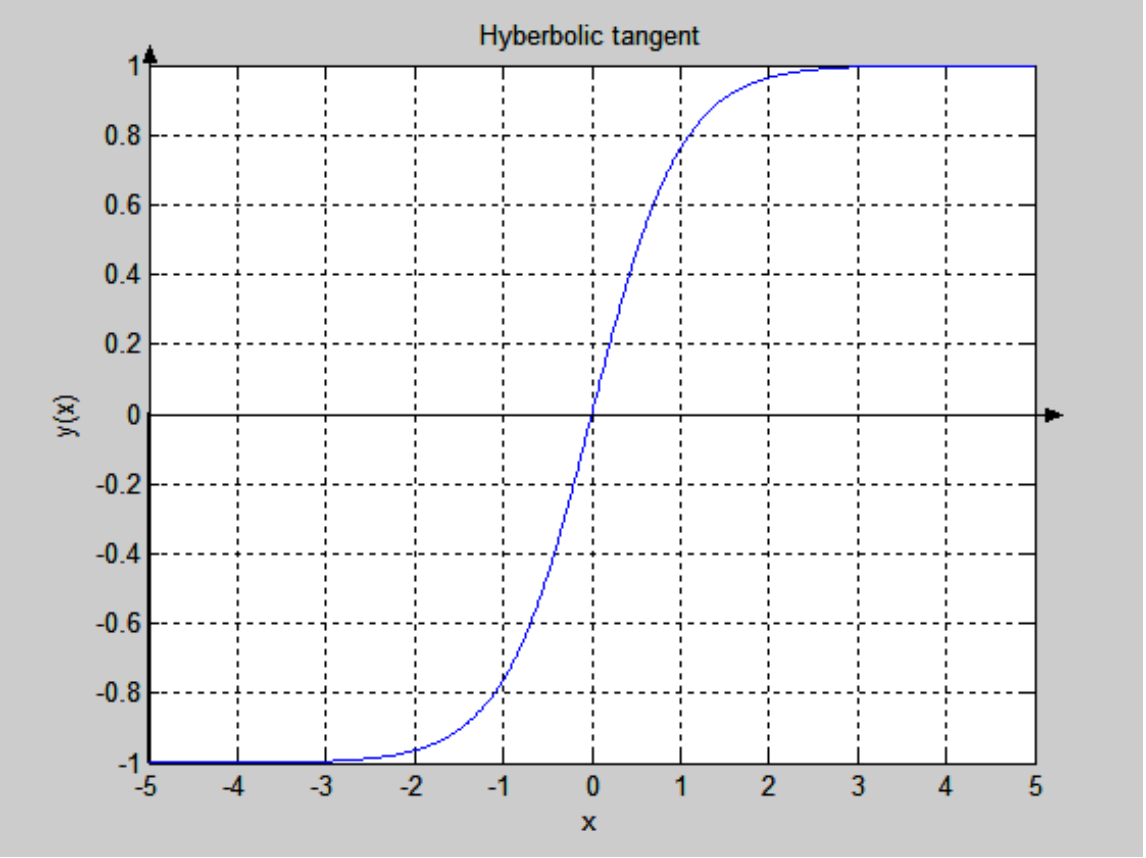
$$w_{k+1}^1 = w_k^1 - \alpha_k \underbrace{\left\{ (y^i - f^2(w_k^2 a^1)) \left[-\frac{\partial f^2}{\partial n^2} \right] \right\}_{w_k^2}}_{\delta_{oi}} w_k^2 \left[-\frac{\partial f^1}{\partial n^1} \right]_{w_k^1} p^i$$

$$= w_k^1 - \alpha \delta_{oi} w_k^2 p^i$$

Activation functions – squashing functions

Example
Hyperbolic tangent

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



$$\frac{df}{dx} = \frac{e^{x} + e^{-x}}{e^{x} + e^{-x}} - \frac{e^{x} - e^{-x}}{\left(e^{x} + e^{-x}\right)^{2}}\left(e^{x} - e^{-x}\right) = 1 - \tanh(x)^{2} = 1 - f(x)^{2}$$

Backpropagation algorithm – Simple network

Apply gradient algorithm to the last layer (i.e. w^2)

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \left(\frac{\partial J}{\partial \mathbf{w}} \right)\Bigg|_{\mathbf{w}_k}$$

Cost function

$$J = J(w^1, w^2) = \frac{1}{2}(e^i)^2 = \frac{1}{2}(y^i - a^2)^2$$

$$= \frac{1}{2}(y^i - f^2(n^2))^2 = \frac{1}{2}(y^i - f^2(w^2 a^1))^2 = \frac{1}{2}(y^i - f^2(w^2 a^1))^2$$

Unknown

Initially set the weights

$$\left(w_0^1, w_0^2\right)$$

Since input p_1 known, compute

$$a_0^1 = f^1\left(w_0^1 p_1\right)$$

Backpropagation algorithm - General

1. Set initial weights for all $\left(w_{ij}^{l}\right)_0$, usually small, random values
2. Use the first input set and compute all the values for the network, i.e. compute forward all the way to the output.
3. Update the weights starting from the last layer: if elements in weight matrix are $\left\{\mathbf{W}^{\mathbf{N}}\right\}_{ij} = \mathbf{w}_{ij}^{N}$

$$\left(\mathbf{w}_{ij}^{N}\right)_{k+1} = \left(\mathbf{w}_{ij}^{N}\right)_{k} + \alpha(y_i - (a_i^N)_k)\left[\frac{df^N}{dn^N}\right]_{n_i^N} (a_i^N)_{k-1}$$

$$= \left(\mathbf{w}_{ij}^{N}\right)_{k} + \alpha\delta_i^N (a_i^N)_{k-1}$$

where

$$\delta_i^N = \alpha(y_i - (a_i^N)_k)\left[\frac{df^N}{dn^N}\right]_{n_i^N}$$

Backpropagation algorithm

4. For hidden layers $l = 1, \cdots, N-1$

$$\left(\mathbf{w}_{ij}^{N}\right)_{k+1} = \left(\mathbf{w}_{ij}^{l}\right)_{k} + \alpha\delta_i^l (a_i^{l-1})_k$$

where

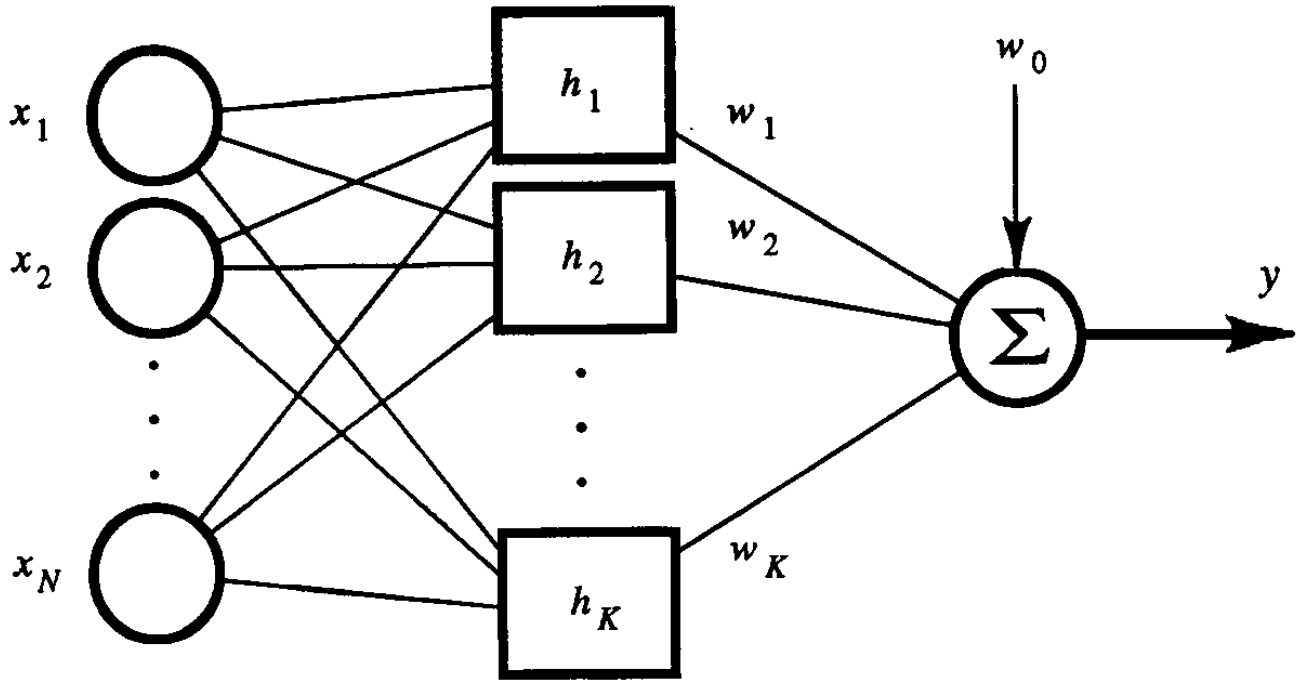
$$\delta_i^l = \left[\frac{df^l}{dn^l}\right]_{n_i^l} \sum_{i=1}^{n} \delta_i^{l+1}(w_{ij}^{l+1})$$

5. Check tolerance – if not satisfied, go to 2, else stop.

Radial Basis Function Networks (RBFN)

Radial Basis Function Networks (RBFN)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$



Other

Gaussian $$h_i(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{x}_{ci}\|^2 / \delta_i^2}$$

Output $$y(\mathbf{x}) = w_0 + \sum_{i=1}^{K} w_i h_i(\mathbf{x})$$

Radial Basis Functions

1. Gaussian

$$h_i(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{x}_{ci}\|^2 / \delta_i^2}$$

2. Thin-plate splin

$$h_i(\mathbf{x}) = \|\mathbf{x}-\mathbf{x}_{ci}\|^2 \log\|\mathbf{x}-\mathbf{x}_{ci}\|$$

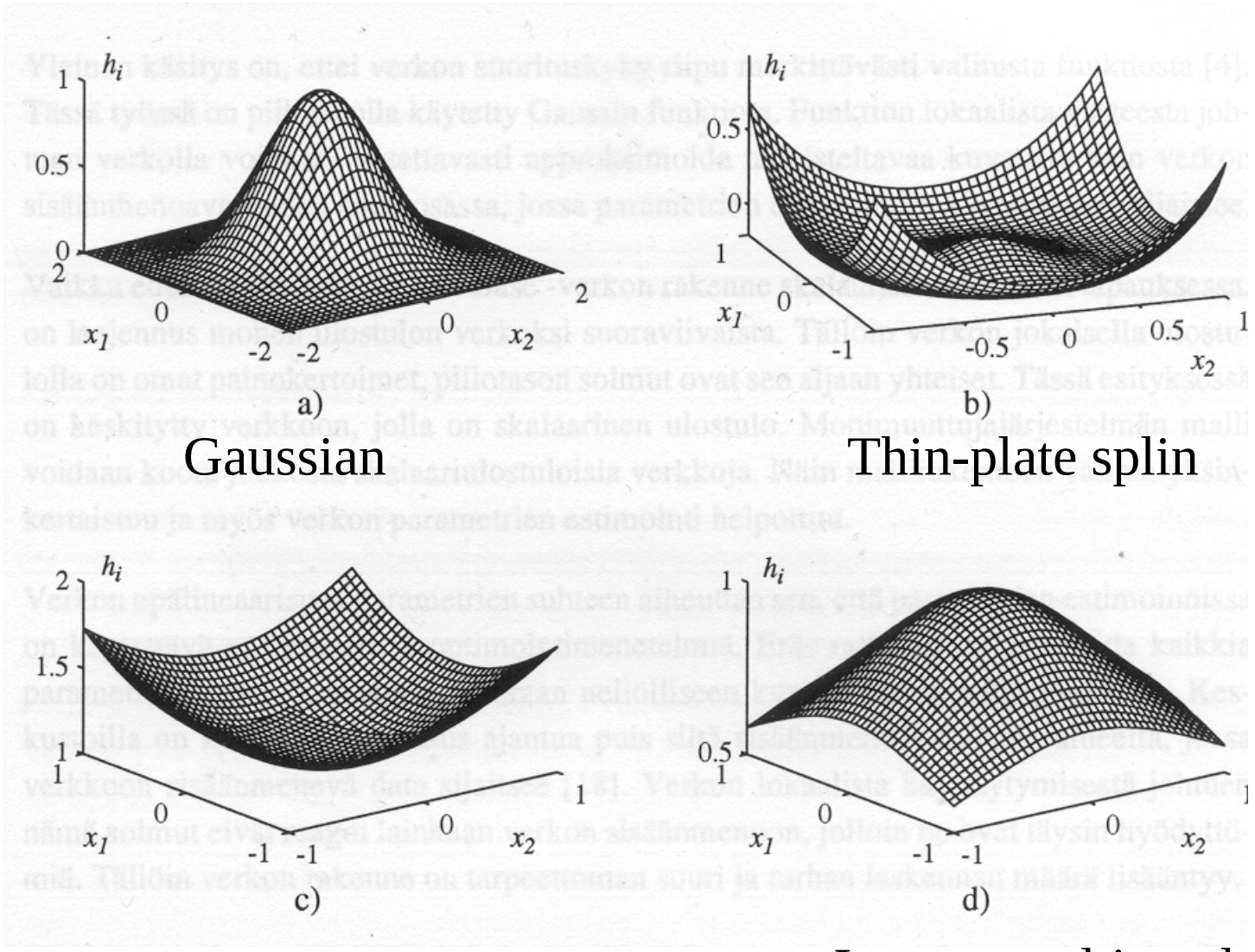
3. Multiquadratic

$$h_i(\mathbf{x}) = \sqrt{\|\mathbf{x}-\mathbf{x}_{ci}\|^2 + \beta^2}$$

4. Inverse multiquadratic

$$h_i(\mathbf{x}) = \frac{1}{\sqrt{\|\mathbf{x}-\mathbf{x}_{ci}\|^2 + \beta^2}}$$

Radial Basis Function Networks (RBFN)



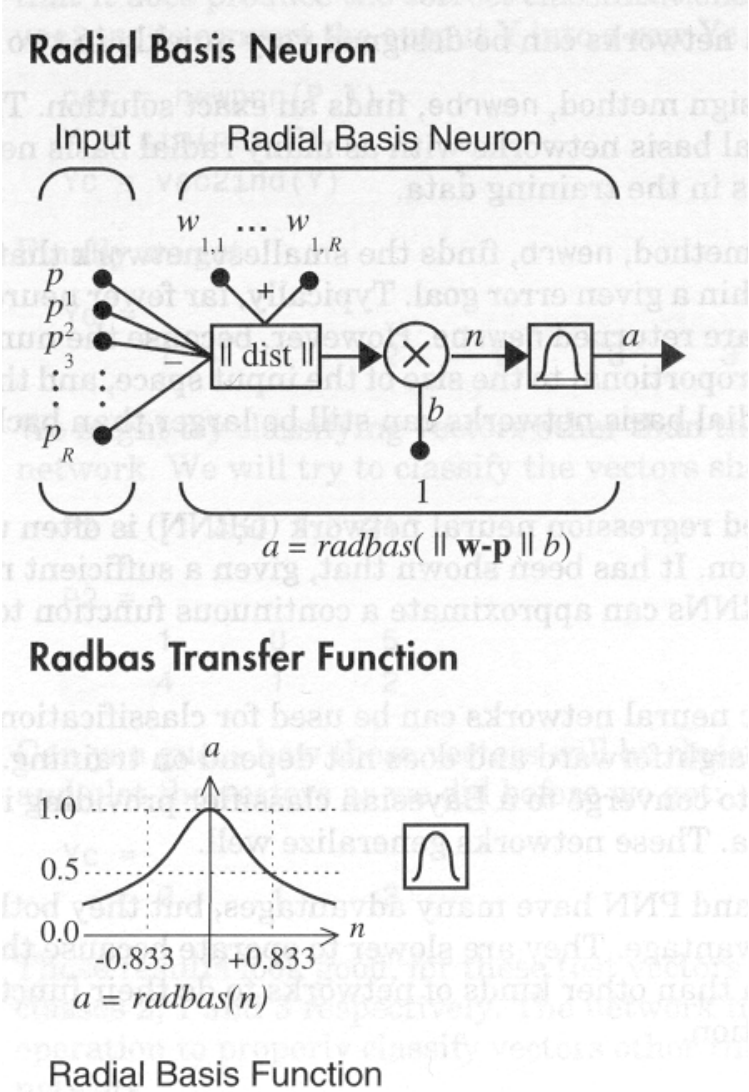
Gaussian

Thin-plate splin

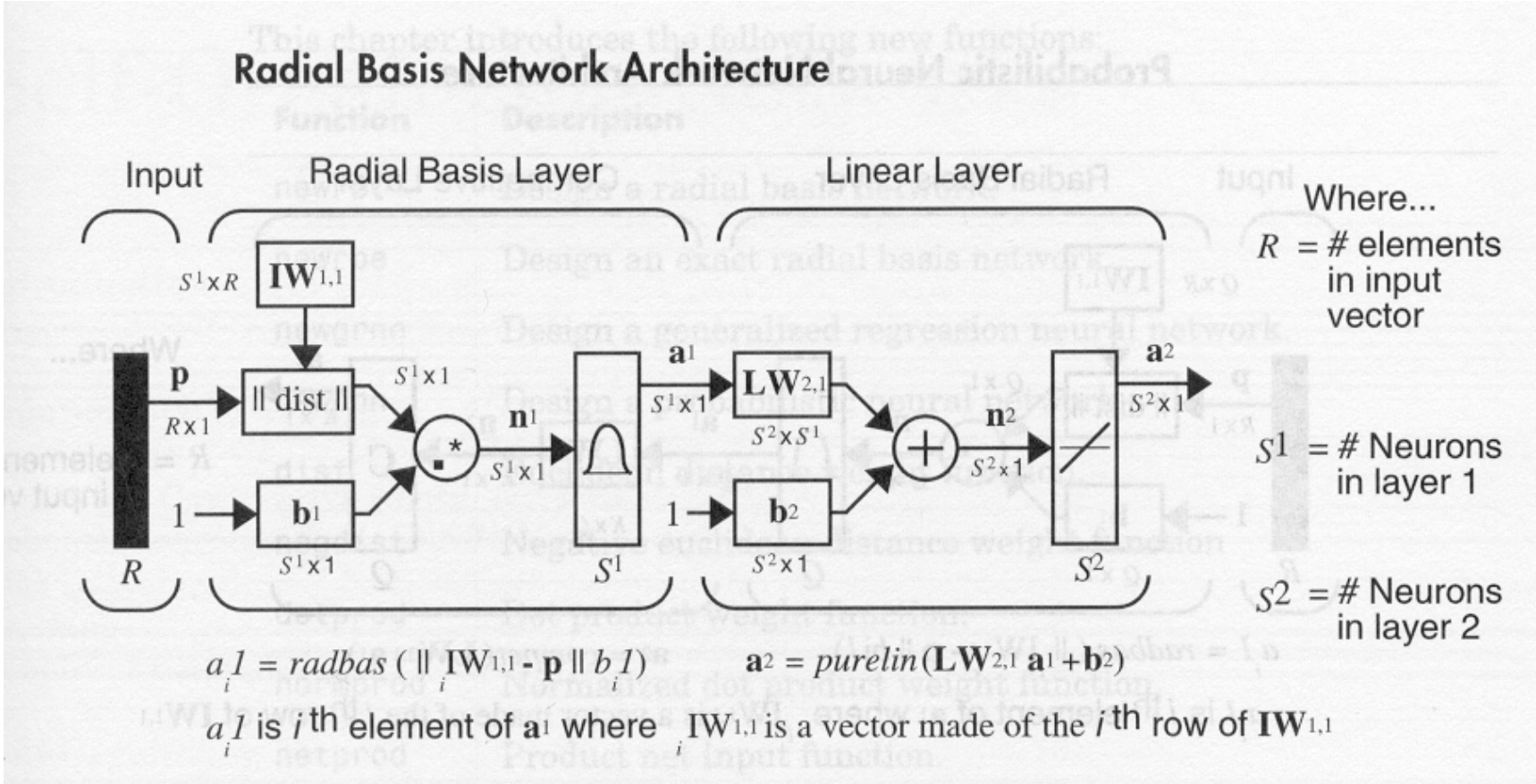
Multiquadratic

Inverse multiquadratic

Neural Network Toolbox in MATLAB



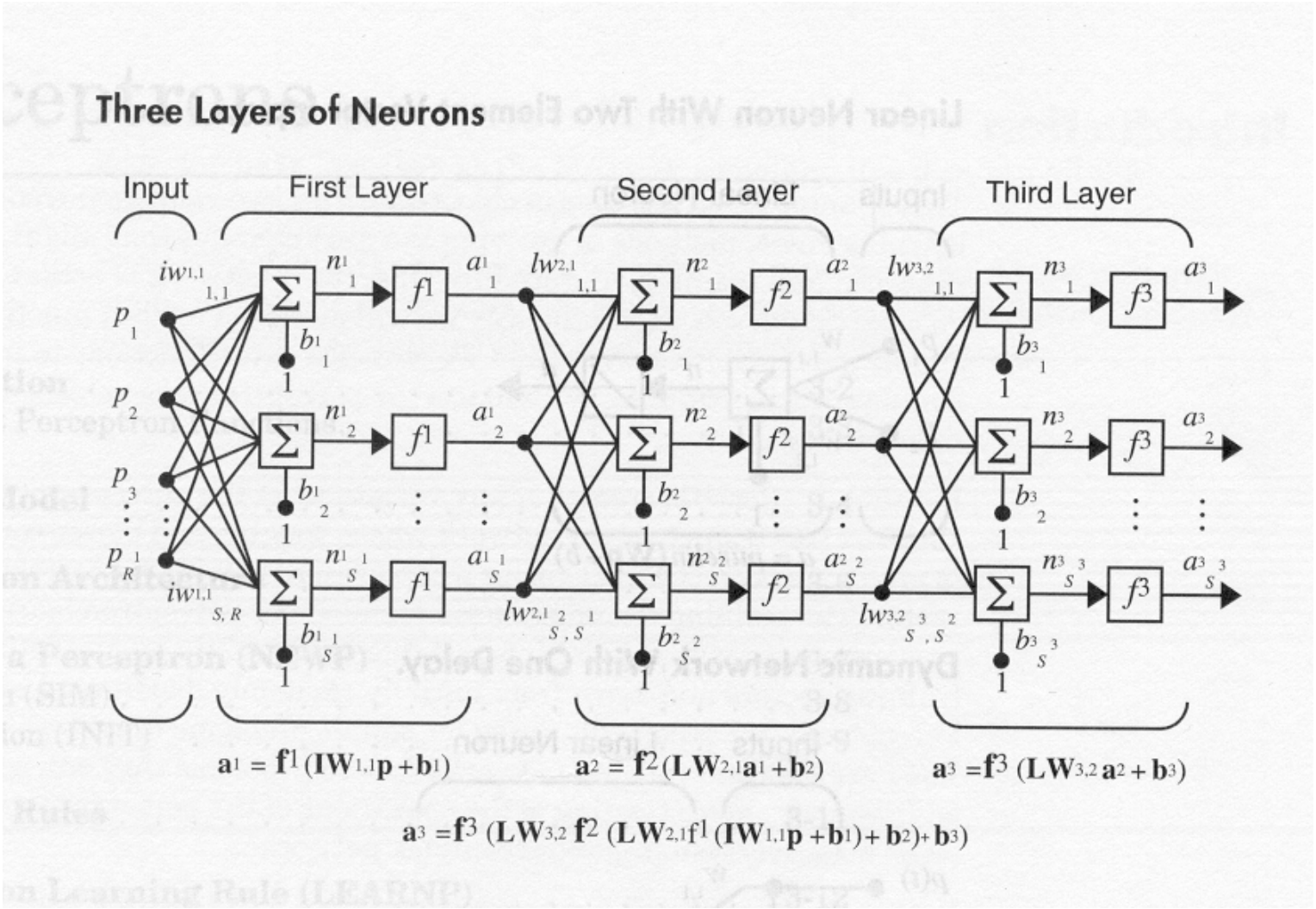
Neural Network Toolbox in MATLAB



LEARNING ALGORITHMS FOR RBFN

- *Random Selection (RS)* - Fixed centers assigned randomly among input samples (Lowe, 1989)
- *Orthogonalization of regressors (OLS)* (Chen S., et al, 1991) – One center at a time to maximize the increment of the output energy.
- *Supervised Selection (SS) of Centers* (Karayiannis, 1999): The centers and linear weights are updated using a back-propagation-like gradient-descent algorithm.
- *Input Clustering (IC)* (Moody, Darken, 1989): Centers are determined by a clustering algorithm (such as Vector Quantization (VQ), k-means) applied to input training sample vectors.
- *Input Output Clustering (IOC)* (Chen C.L. et.al., 1993, Uykan et.al., 1997): IOC method applies a clustering algorithm to the augmented vectors obtained by concatenating the output vectors to the input vectors in input-output space.

Neural Network Toolbox in MATLAB



Approximation results

Theorem (Weierstrass):

If f is a continuous complex function on $[a,b]$,
there exists a sequence of polynomials P_n such that
we have

$$\lim_{n\to\infty} P_n(x) = f(x)$$

uniformly on $[a,b]$.
If f is real, the P_n may be taken real.

WHAT IF NEURAL NETWORKS ARE USED INSTEAD OF POLYNOMIALS?

Approximation results

Theorem (Stone-Weierstrass): Let X be a compact space and A an algebra of continuous real-valued functions on X which separates the points of X and which contains the constant functions.
Then given any continuous real-valued function f on X and any $\varepsilon > 0$ there is a function g in A such that for all $x \in X$ we have

$$|g(x) - f(x)| < \varepsilon$$

In other words, A is a dense subset of $C(X)$

or closure of A $\overline{A} = C(X)$

NOTE: NEURAL NETWORKS ARE CONTINUOUS REAL-VALUED FUNCTIONS FORMING AN ALGEBRA!

Approximation results – Two-layer feedforward network

Theorem: Let g be a bounded, increasing real function, $\mathbf{K}$ a compact set in $\mathbf{R}^d$
and $f : \mathbf{K} \to \mathbf{R}^d$ a continuous function.
Then for every $\varepsilon > 0$ there exists $k \in \mathbf{N}$ and $w_i, w_{ij}, \theta_i \in \mathbf{R}$ such that

$$\max_{\mathbf{x} \in K} \left| f(\mathbf{x}) - y(\mathbf{x}) \right| < \varepsilon$$

where

$$y(\mathbf{x}) = \sum_{i=1}^{k} w_i g\left(\sum_{j=1}^{d} w_{ij} x_j - \theta_i \right).$$

Proofs: Cybenko,Carrol-Dickinson, Funahashi, all in 1989

Approximation results – Radial basis function network

Theorem: Let $g : \mathbf{R}^d \to \mathbf{R}$
be a radial symmetric, integrable, bounded, function, such that g is continuous almost everywhere and $\int_{\mathbf{R}^d} g(\mathbf{x})\, d\mathbf{x} \neq 0$.

Then the weighted sum $\sum_{i=1}^{k} w_i \phi\left(\frac{\|\mathbf{x}-\mathbf{x}_i\|}{\sigma}\right)$

is dense in $L^p\left(\mathbf{R}^n\right)$, where $\phi(\|\mathbf{x}\|) = g(\mathbf{x})$.

Park-Sandberg (1991)

MATLAB demo

- NN toolbox has nice demos about function approximations.

NEURAL NETWORKS

Basics using *MATLAB Neural Network Toolbox*

http://www.iau.dtu.dk/research/control/nnsysid.html

CLASSES OF NEURAL NETWORKS STUDIED

- Multilayer perceptron networks (MLPN) also called

  Feeforward or Backpropagation networks
- Radial basis function networks (RBFN)
- Self-Organizing Maps (SOMs) - later

- Other well-known e.g.
  - Hopfield networks
  - Recurrent networks

CLASSES OF NEURAL NETWORKS STUDIED

- Supervised networks (output target known)
  - MLPN, RBFN
- Unsupervised
  - SOM

PARAMETER ESTIMATION – ’TEACHING’ THE NETWORK

- Define the structure of the network
  - Choose the network structure
  - Choose activation functions
  - Initialize network parameters,:weights and biases
- Define parameters associated with the training algorithm like error goal, maximum number of epochs (iterations)
- Call the training algorithm.
- Simulate the output of the neural network with the measured input data.
- Compare with the validation data.

- MATLAB COMMANDS: *newff*, *train* and *sim*

PARAMETER ESTIMATION – 'TEACHING' THE NETWORK

- Define the structure of the network with newff (initial MLP)

$$net = newff(\underbrace{PR}_{\text{min,max values}}, [\underbrace{S1\ S2 ... SNl}_{\text{size of the } i\text{th layer}}], \{\underbrace{TF1\ TF2 ... TFNl}_{\text{activation function of } i\text{th layer}}\}, \underbrace{BTF}_{\text{training algorithm}})$$

- NEW
- ***net = newff(P,T,S,TF,BTF,BLF,PF,IPF,OPF,DDF)***
- *PR* = Rx2 matrix of min and max values for R input elements,
- *Si* = Size of the *i*th layer,
- *TFi* = Activation (or transfer function) of the *i*th layer, default = '*tansig*',
- *BTF* = Network training function, default = '*trainlm*'

PARAMETER ESTIMATION – ’TEACHING’ THE NETWORK

- NEWFF(P,T,S,TF,BTF,BLF,PF,IPF,OPF,DDF) takes,
- P - RxQ1 matrix of Q1 representative R-element input vectors.
- T - SNxQ2 matrix of Q2 representative SN-element target vectors.
- Si - Sizes of N-1 hidden layers, S1 to S(N-1), default = [].
- (Output layer size SN is determined from T.)
- TFi - Transfer function of ith layer. Default is 'tansig' for
- hidden layers, and 'purelin' for output layer.
- BTF - Backprop network training function, default = 'trainlm'.
- BLF - Backprop weight/bias learning function, default = 'learngdm'.
- PF - Performance function, default = 'mse'.
- IPF - Row cell array of input processing functions.
- Default is {'fixunknowns','remconstantrows','mapminmax'}.
- OPF - Row cell array of output processing functions.
- Default is {'remconstantrows','mapminmax'}.
- DDF - Data division function, default = 'dividerand';
- and returns an N layer feed-forward backprop network.
-
- The transfer functions TF{i} can be any differentiable transfer
- function such as TANSIG, LOGSIG, or PURELIN.
-
- The training function BTF can be any of the backprop training
- functions such as TRAINLM, TRAINBFG, TRAINRP, TRAINGD, etc.

PARAMETER ESTIMATION – 'TEACHING' THE NETWORK

- Simplest command to generate initial MLP

- ***net = newff(P,T)***

PARAMETER ESTIMATION – ’TEACHING’ THE NETWORK

- Call the training algorithm.

$$net1 = train(\underbrace{net}_{\substack{initial\\MLPN}}, \underbrace{P}_{\substack{input\\vector}}, \underbrace{T}_{\substack{output\\vector}})$$

- P = input vector,
- T = output vector,

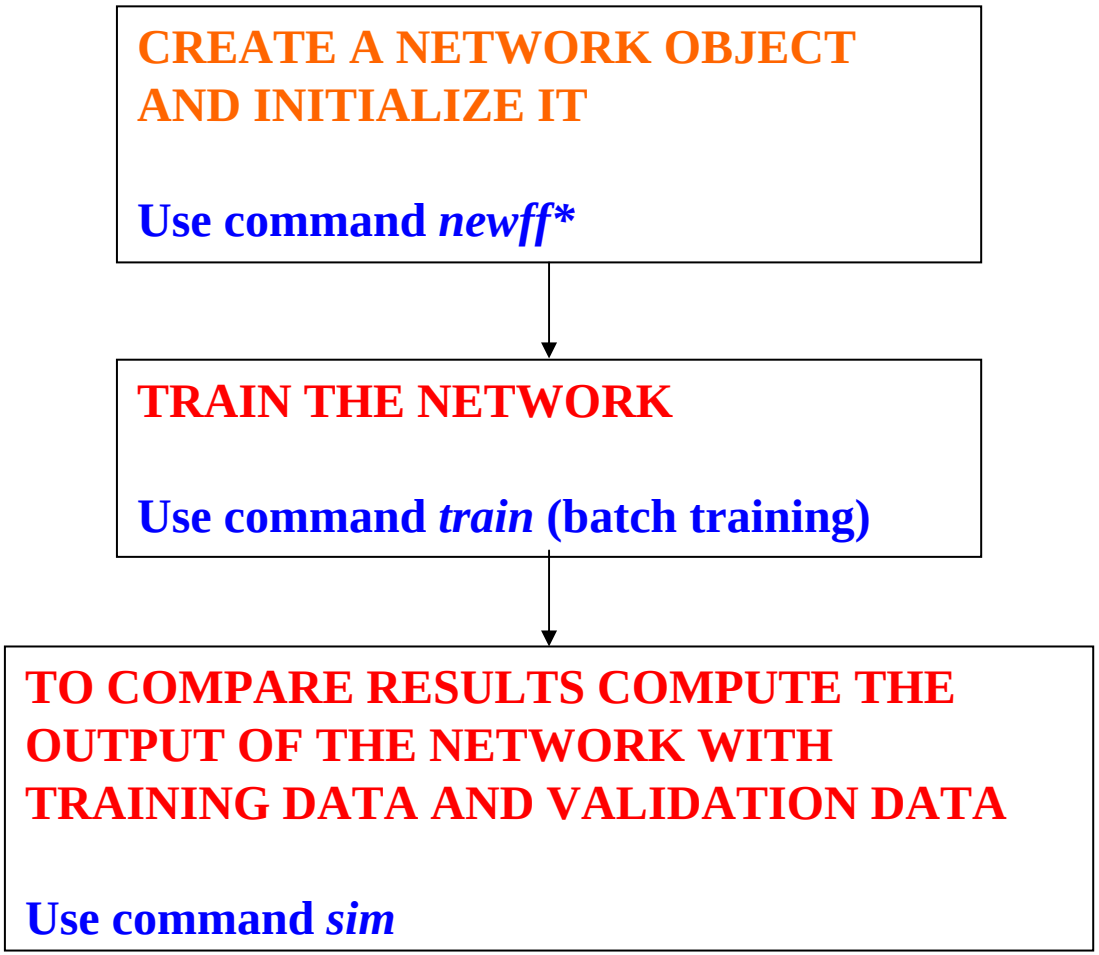
PARAMETER ESTIMATION – 'TEACHING' THE NETWORK

- Simulate the output of the network.

$$a = sim(\underbrace{net1}_{\substack{final\\MLPN}}, \underbrace{P}_{\substack{input\\vector}})$$

- Here the same input P as used in training is applied
- Compare with measured output T

BASIC FLOW DIAGRAM



***The last two letters in the command *newff* indicate the type of neural network in question: feedforward network.**

EXAMPLE 1

Consider *humps* function in MATLAB. It is given by

```
y = 1 ./ ((x-.3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6;
```

but in MATLAB can be called by *humps*.
Here we like to see if it is possible to find a neural network to fit the data generated by humps-function between [0,2].

a) Fit a multilayer perceptron network on the data.
Try different network sizes and different teaching algorithms.

b) Repeat the exercise with radial basis function networks.

EXAMPLE 1 – *feedforward NN*

1. Generate data

```
x = 0:.05:2; y=humps(x);  P=x; T=y;
```

2. Define the NN object and initialize

```
net=newff([0 2], [ 20  ,1], {'tansig','purelin'},'trainlm');
```

3. Train the network (find the best network parameters to fit the data)

```
net1 = train(net, P, T);
```

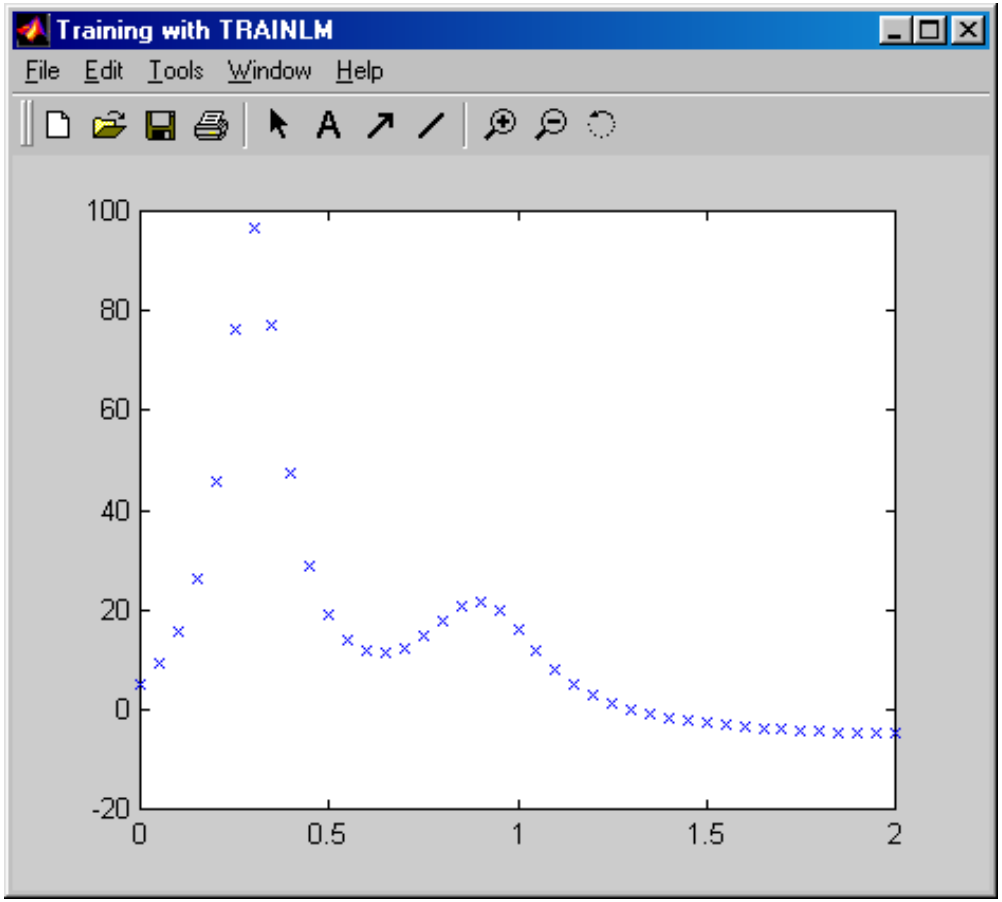
4. Simulate and and compare with data

```
a= sim(net1,P);              %Simulate result
plot(P,a-T,P,T,P,a);         %Plot the result and the error
```

EXAMPLE 1 - *feedforward NN*

1. Generate data

x = 0:.05:2; y=humps(x); P=x; T=y;
plot(P,T,'x')



EXAMPLE 1 - *feedforward NN*

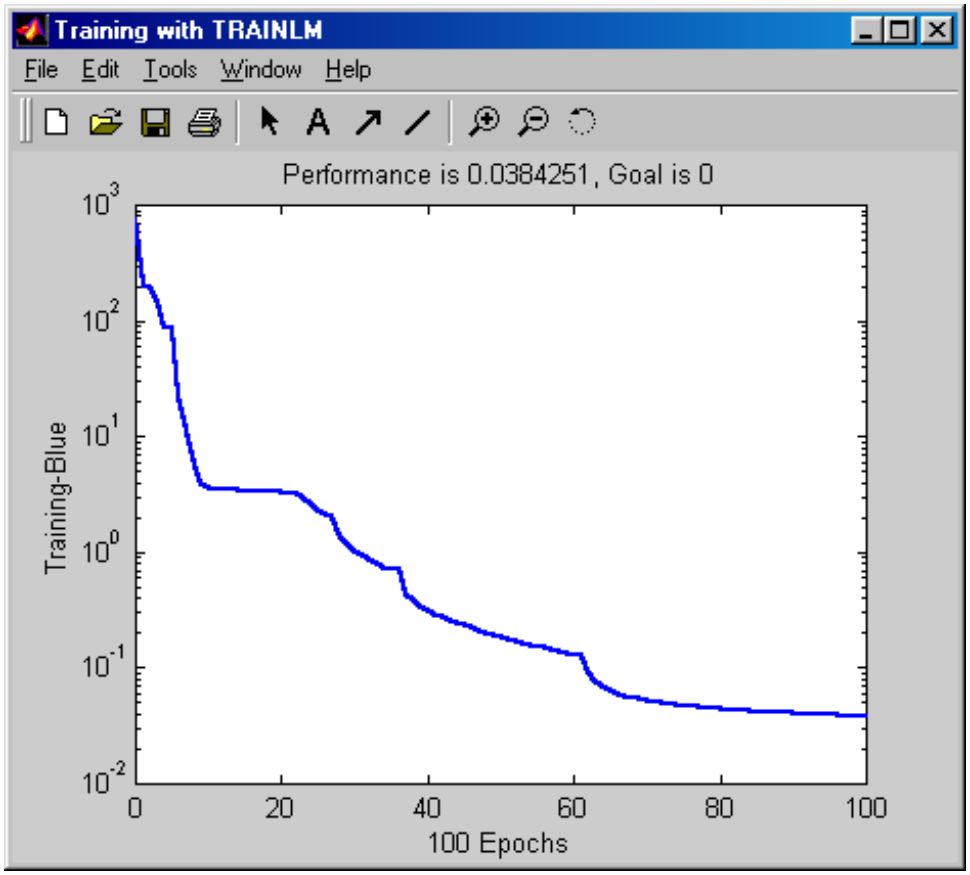
2. Define the NN object and initialize

net=newff([0 2], [10,1], {'tansig','purelin'},'trainlm');

EXAMPLE 1 - *feedforward NN*

3. Train the network (find the best network parameters to fit the data)

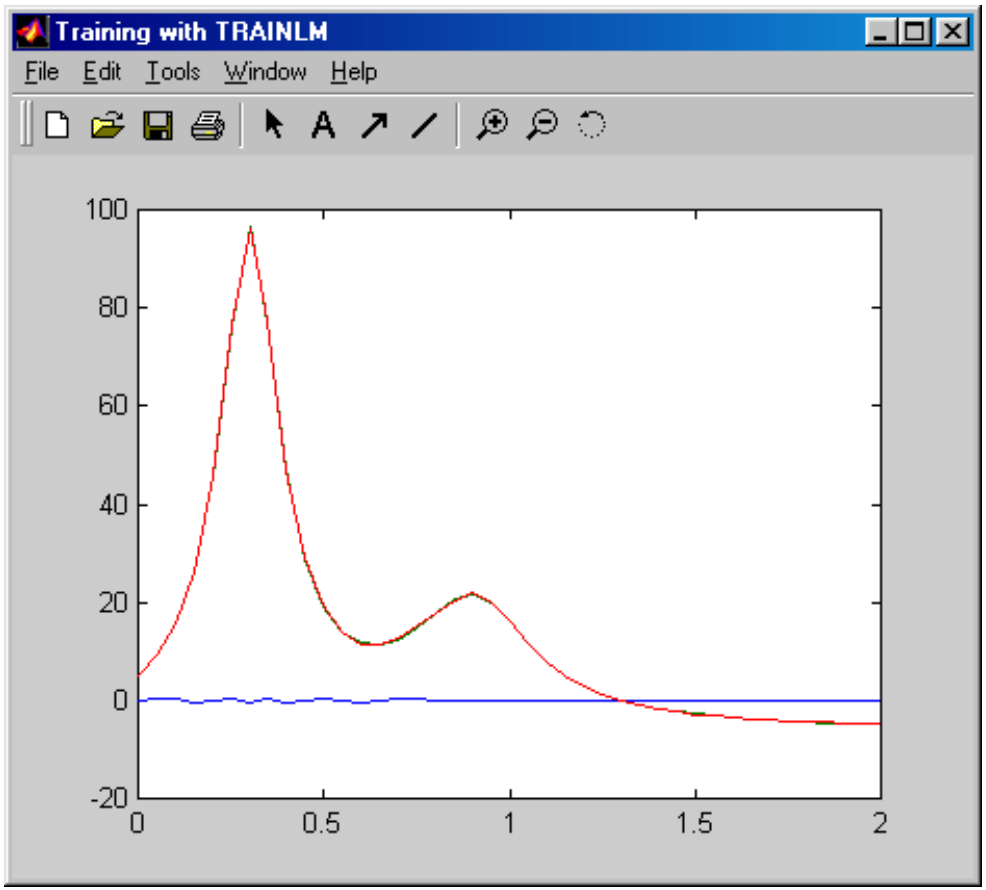
net1 = train(net, P, T);



EXAMPLE 1 - *feedforward NN*

4. Simulate and and compare with data

```
a= sim(net1,P);          %Simulate result
plot(P,a-T,P,T,P,a);     %Plot the result and the error
```



EXAMPLE 1 – *radial basis function NN*

1. Generate data

x = 0:.05:2; y=humps(x); P=x; T=y;

2. Define the NN object and 3. Train the network .

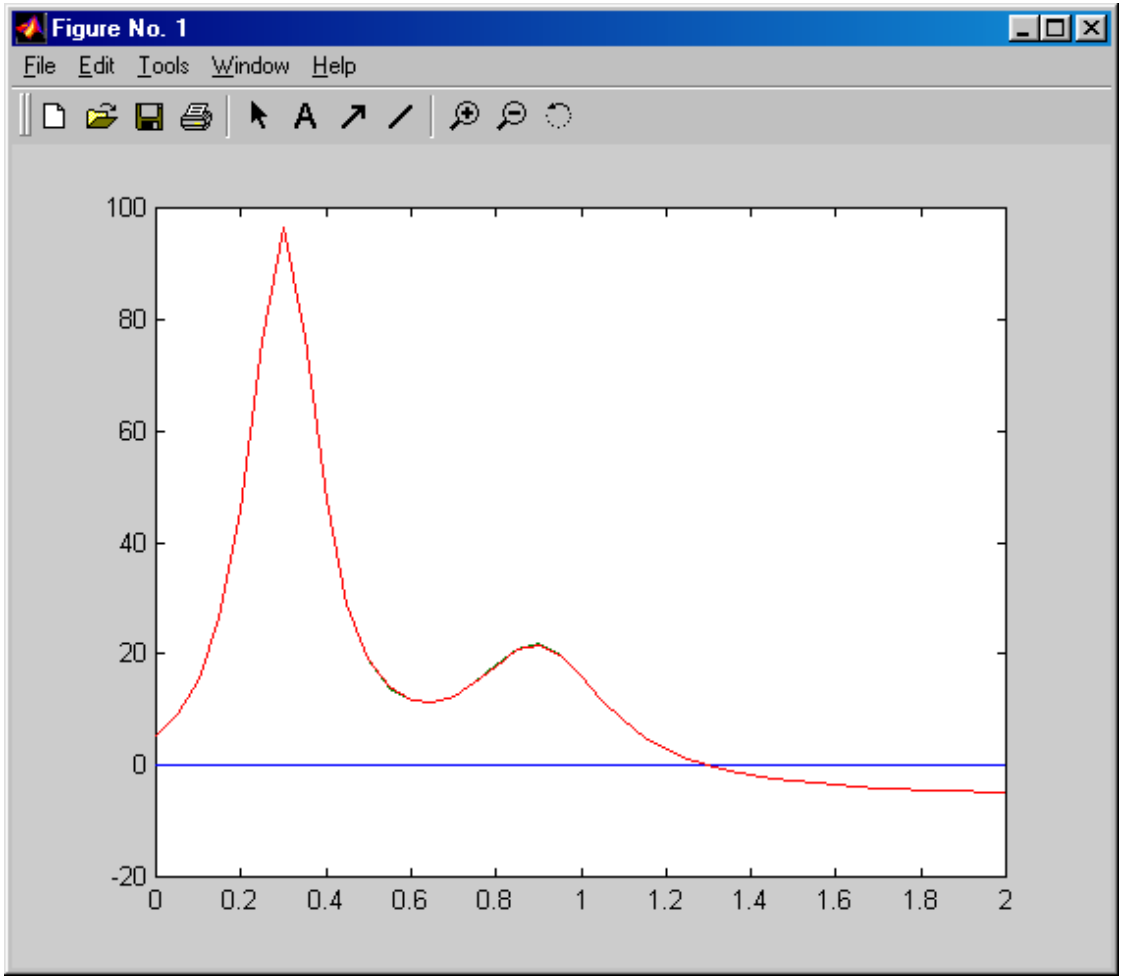
goal=0.02; spread= 0.1;

```
net1 = newrb(P,T,goal,spread);
```

4. Simulate and and compare with data

```
a= sim(net1,P);          %Simulate result
plot(P,a-T,P,T,P,a);     %Plot the result and the error
```

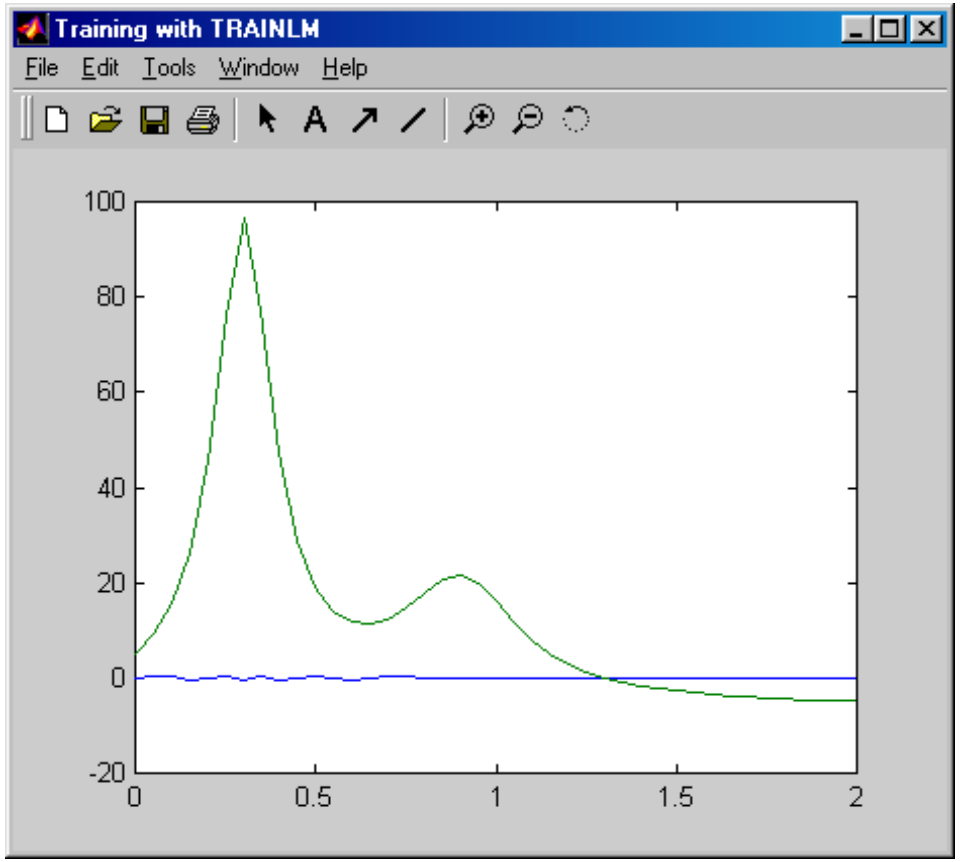
EXAMPLE 1 - – *radial basis function NN*



EXAMPLE 1

4. Simulate and and compare with data

a= sim(net1,P); *%Simulate result*
plot(P,a-T,P,T) ; *%Plot the result and the error*



EXAMPLE 1

3. Train the network (find the best network parameters to fit the data)

net1 = train(net, P, T);

4. Simulate and and compare with data

```
a= sim(net1,P);              %Simulate result
plot(P,a-T,P,T) ;            %Plot the result and the error
```