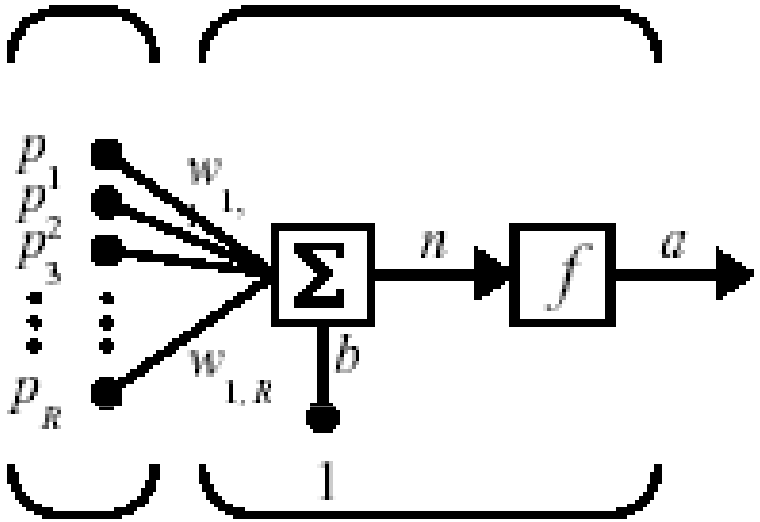
Artificial Neural Network in Matlab

Waslaluddin

Architecture (single neuron)



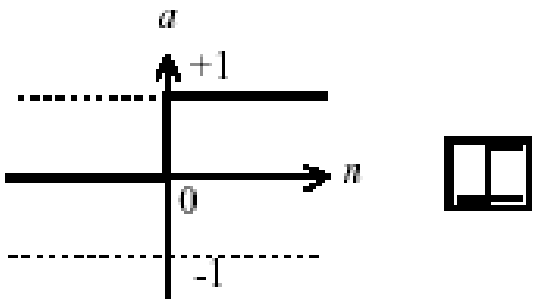
w is weight matrices, dimension 1xR

p is input vector, dimension Rx1

b is bias

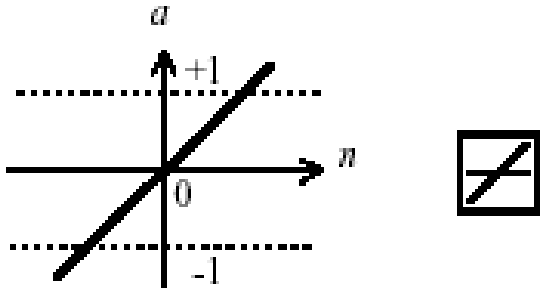
$$a = f(\mathbf{W}\mathbf{p} + b)$$

Transfer Function



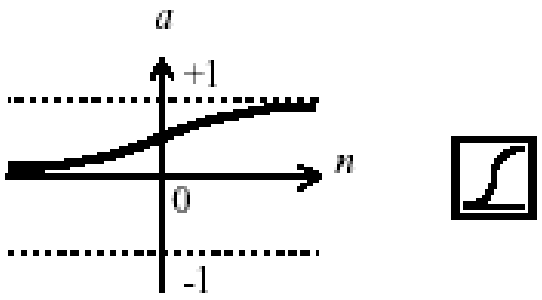
$a = hardlim(n)$

Hard-Limit Transfer Function



$a = purelin(n)$

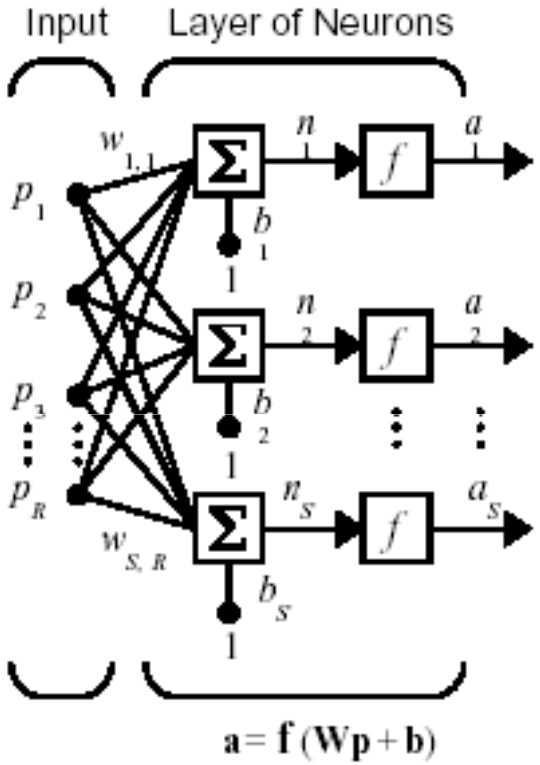
Linear Transfer Function



$a = logsig(n)$

Log-Sigmoid Transfer Function

Architecture with neurons



Where...

R = number of elements in input vector

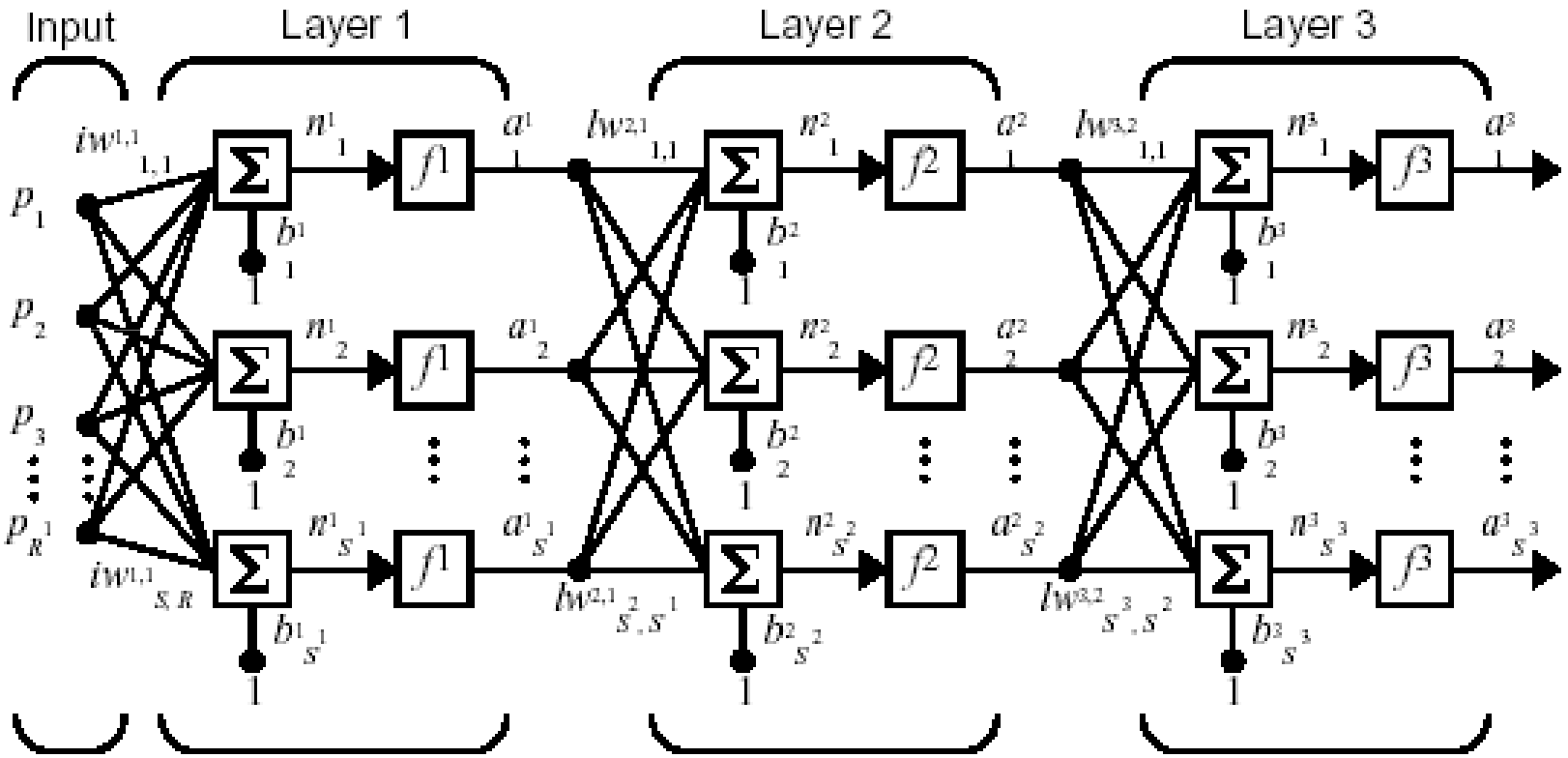
S = number of neurons in layer

w is weight matrices, dimension SxR

p is input vector, dimension Rxn

b is bias

Multiple layers



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

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

Perceptrons in Matlab

Make the perceptrons with `net = newp(PR,S,TF,LF)`

PR = Rx2 matrix of min and max values for R input elements

S = number of output vector

TF = Transfer function, default = 'hardlim', other option = 'hardlims'

LF = Learning function, default = 'learnp', other option = 'learnpn'

hardlim = hardlimit function

hardlims = symetric hardlimit function

learnp → $\Delta\mathbf{w} = (t-a)\mathbf{p}^T = e\mathbf{p}^T$
learnpn → normalized learnp

$W_{new} = W_{old} + \Delta W$ $\quad$ $b_{new} = b_{old} + e$ $\quad$ where e = t - a

Compute manually…

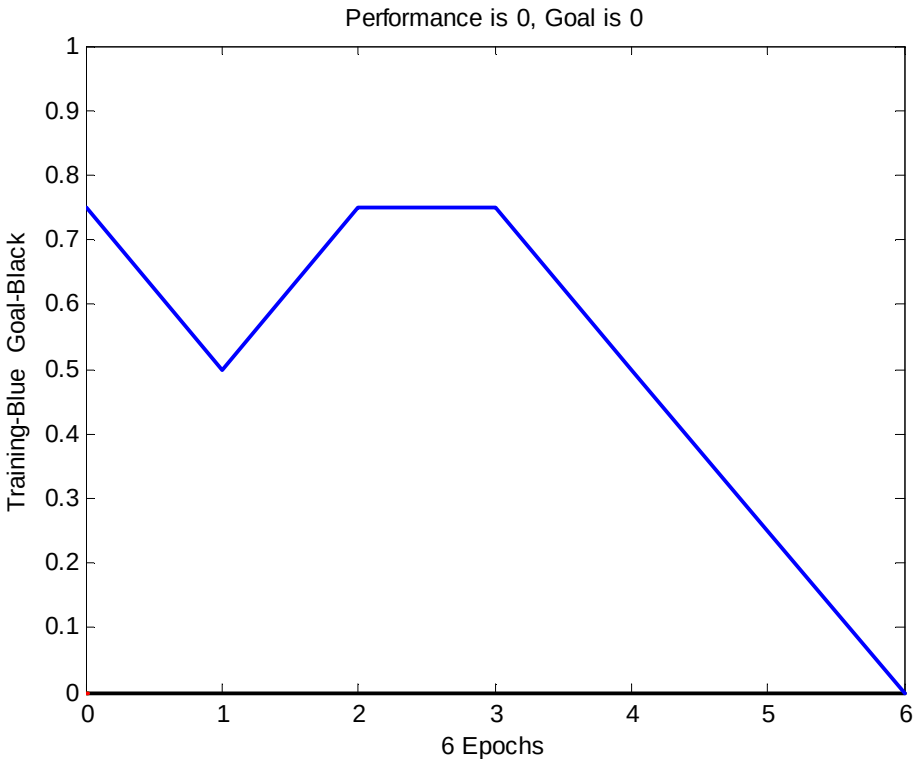
- This is an exercise how to run the artificial neural network
- From the next problem, we will compute the weights and biases manually

AND Gate in Perceptron

```
P = [0 0 1 1; 0 1 0 1];
T = [0 0 0 1];

net = newp([0 1; 0 1],1);
weight_init = net.IW{1,1}
bias_init = net.b{1}

net.trainParam.epochs = 20;
net = train(net,P,T);
weight_final = net.IW{1,1}
bias_final = net.b{1}
simulation = sim(net,P)
```



weight_init = [0 0], bias_init = 0

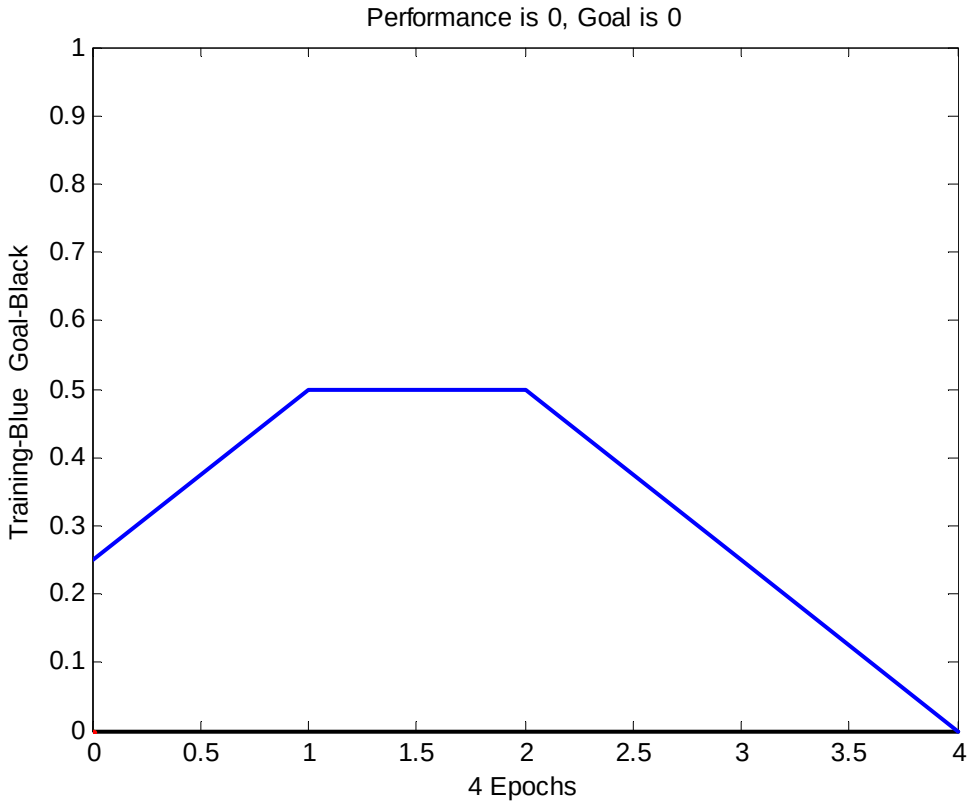
weight_final = [2 1], bias_final = -3

OR Gate in Perceptron

```
P = [0 0 1 1; 0 1 0 1];
T = [0 1 1 1];

net = newp([0 1; 0 1],1);
weight_init = net.IW{1,1}
bias_init = net.b{1}

net.trainParam.epochs = 20;
net = train(net,P,T);
weight_final = net.IW{1,1}
bias_final = net.b{1}
simulation = sim(net,P)
```



weight_init = [0 0], bias_init = 0

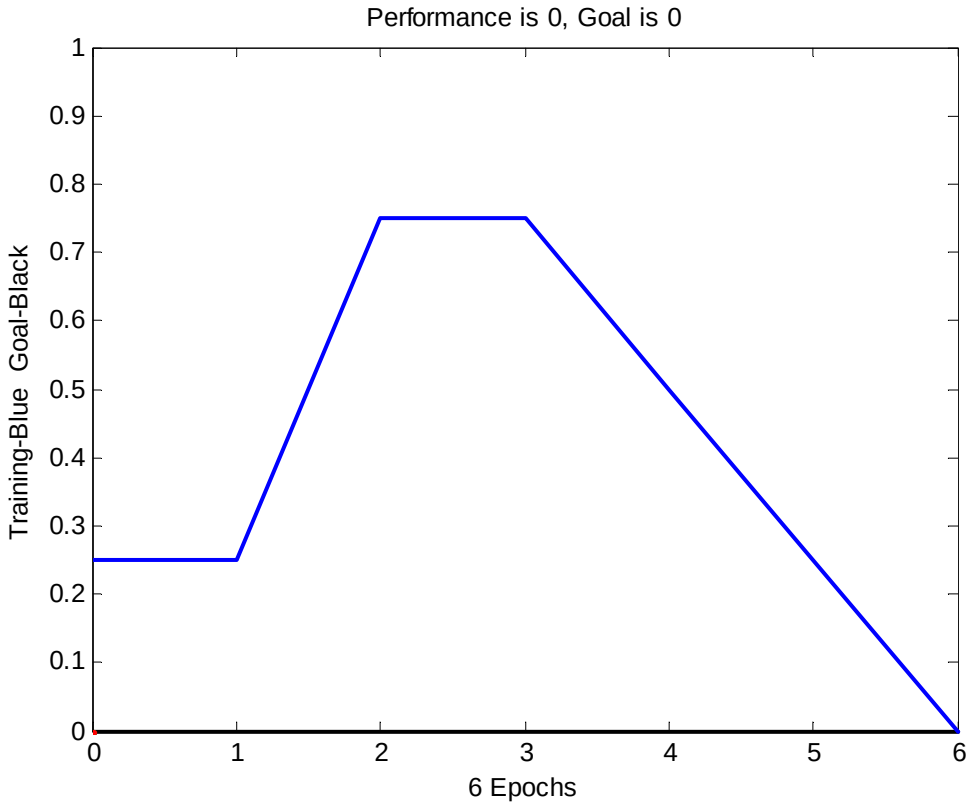
weight_final = [1 1], bias_final = -1

NAND Gate in Perceptron

```
P = [0 0 1 1; 0 1 0 1];
T = [1 1 1 0];

net = newp([0 1; 0 1],1);
weight_init = net.IW{1,1}
bias_init = net.b{1}

net.trainParam.epochs = 20;
net = train(net,P,T);
weight_final = net.IW{1,1}
bias_final = net.b{1}
simulation = sim(net,P)
```



weight_init = [0 0], bias_init = 0

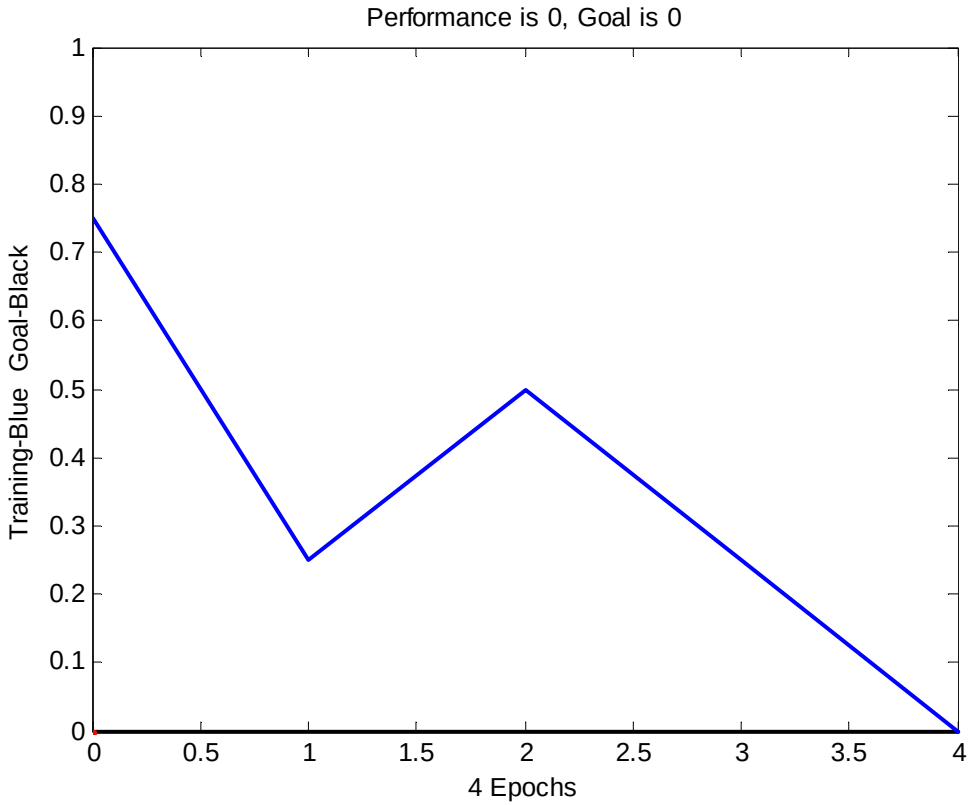
weight_final = [-2 -1], bias_final = 2

NOR Gate in Perceptron

```
P = [0 0 1 1; 0 1 0 1];
T = [1 0 0 0];

net = newp([0 1; 0 1],1);
weight_init = net.IW{1,1}
bias_init = net.b{1}

net.trainParam.epochs = 20;
net = train(net,P,T);
weight_final = net.IW{1,1}
bias_final = net.b{1}
simulation = sim(net,P)
```



weight_init = [0 0], bias_init = 0

weight_final = [-1 -1], bias_final = 0

Backpropagation in Matlab

Make the backpropagation with

```
net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
```

PR = Rx2 matrix of min and max values for R input elements

S = number of output vector

BTF = Transfer function (user can use any transfer functions)

BLF = Learning function

PF = performance

$$x_{k+1} = x_k - \alpha_k g_k$$

Linear Filter (with ANN) in Matlab

Make the Linear Filter with `newlin(PR,S,ID,LR)`

PR = Rx2 matrix of min and max values for R input elements

S = number of output vector

ID = delay

LR = Learning Rate

Transfer function for linear filter is only linear line (purelin)