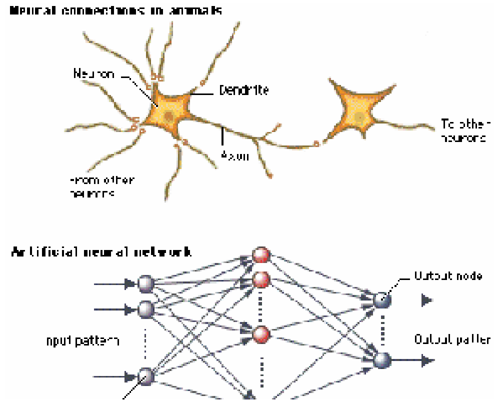
Programming in MATLAB

Chapter 2: Artificial Neural Network (Perceptron)

Gp.Capt.Thanapant Raicharoen, PhD



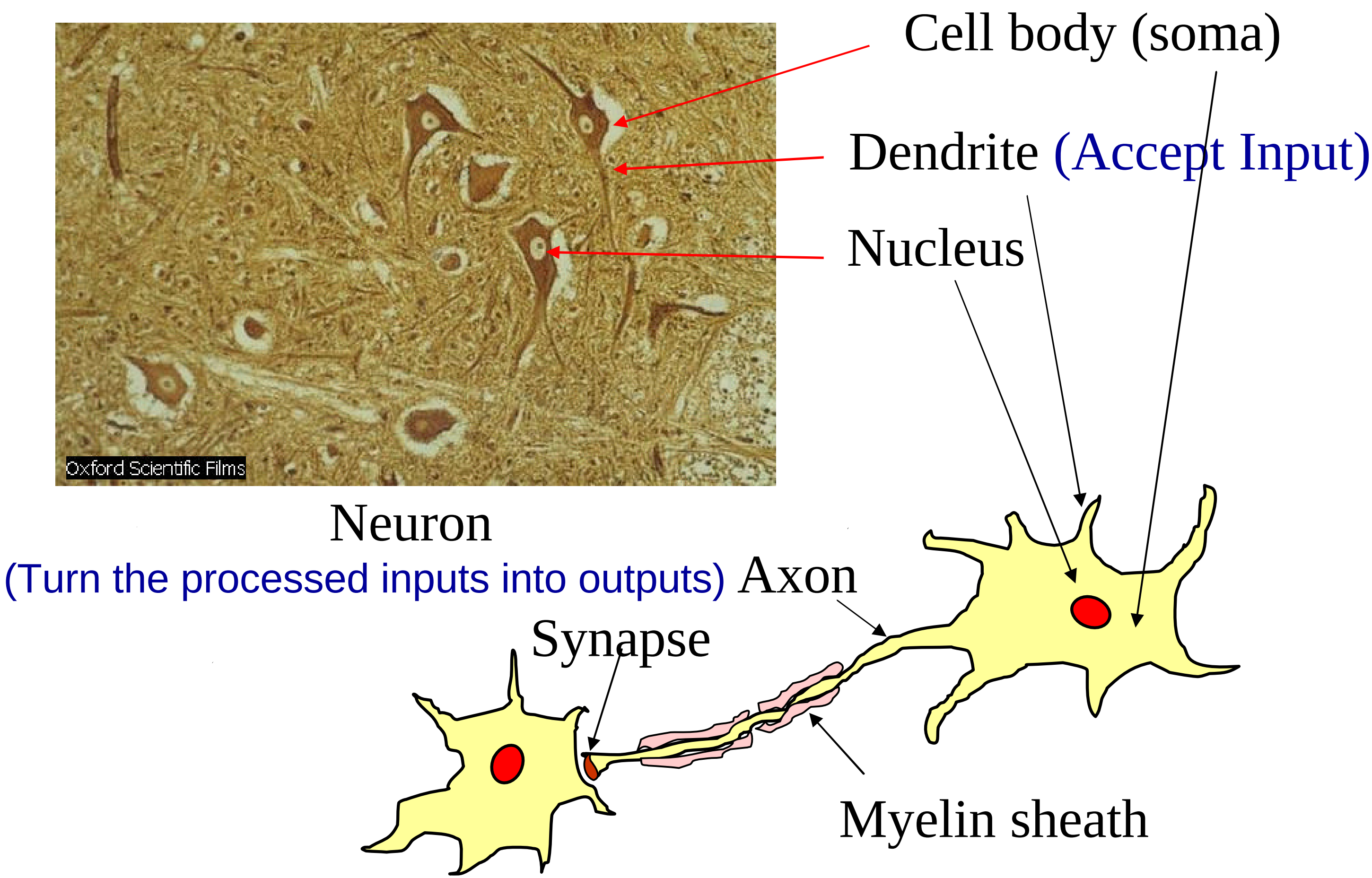
Outline

- Introduction to Artificial Neural Network
- Machine Learning and Learning Rule
- Perceptron Learning Rule
- Perceptron program in MATLAB
- Perceptron MATLAB Toolbox

Human vs. Computer

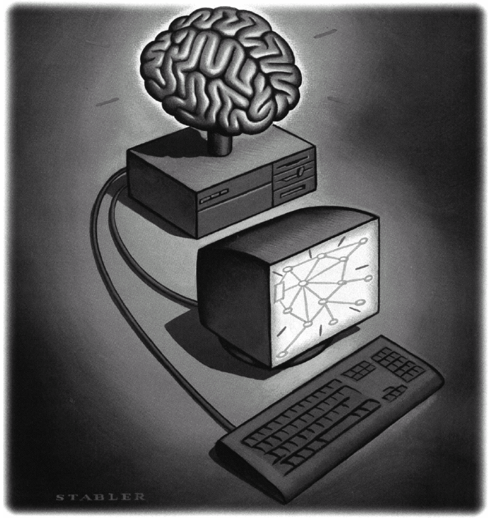
- Speed
  - Human: Neuron Cell Signal 1000 times/Second
  - Computer: CPU 1 GHz
- (mathematical) Processing
  - Human: not good
  - Computer: > 1000 MFLOP
- Problem Analysis
  - Human: good
  - Computer: not good
- (pattern) Recognition
  - Human: good
  - Computer: not good

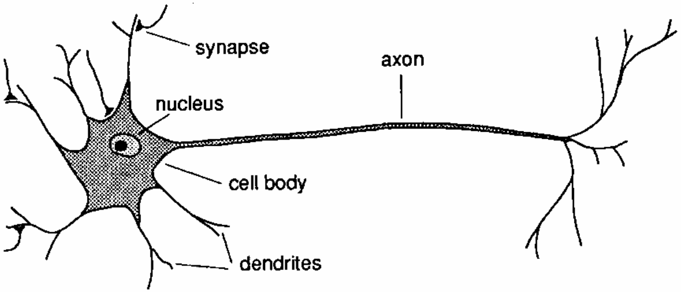
Human Brain


Oxford Scientific Films
Cell body (soma)
Dendrite (Accept Input)
Nucleus
Neuron
(Turn the processed inputs into outputs)
Axon
Synapse
Myelin sheath

(Artificial) Neural Networks

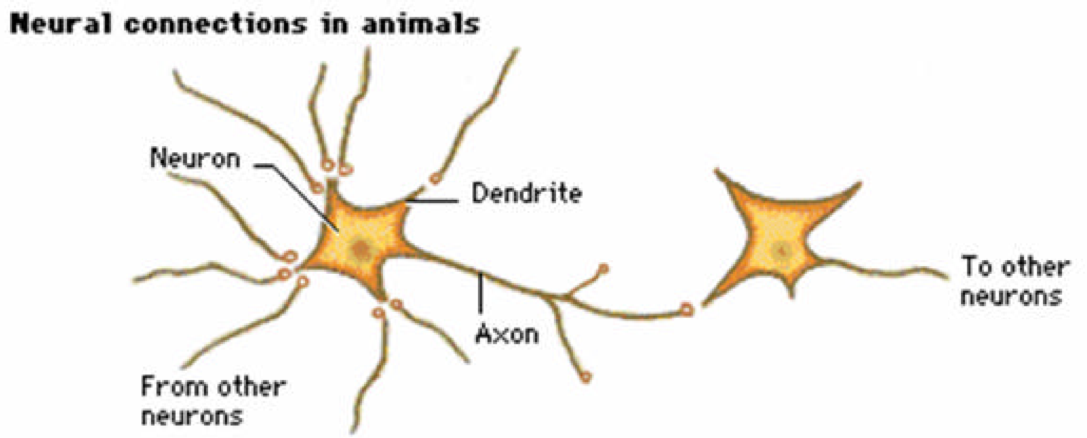
- **What is a Neural Network?**
  - Biologically motivated approach to machine learning
  - Similarity with biological network
- **Human brain**
  - interconnected network of 10^{11} neurones (100,000,000,000)
  - each connected to 10^4 others
  - surprisingly complex decisions, surprisingly quickly (10^{-1}s)
  - robust & fault tolerant, flexible
  - deals with fuzzy, noisy, or inconsistent information

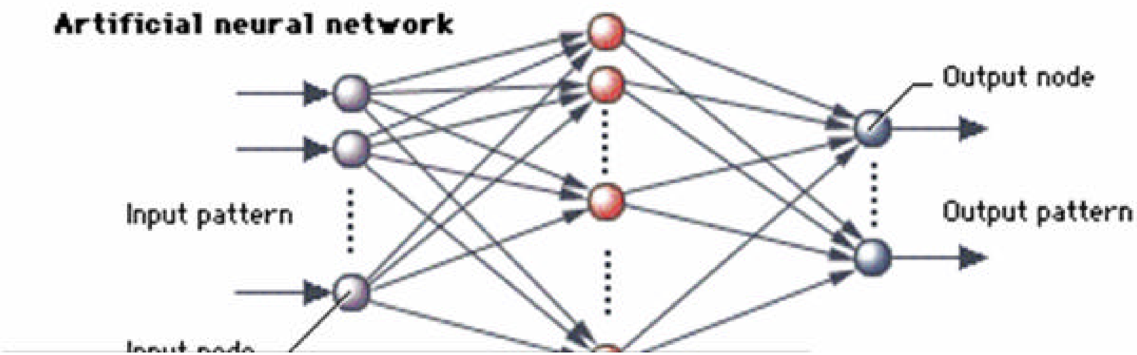




(Artificial) Neural Networks

A biological neurone showing its cell body and connections



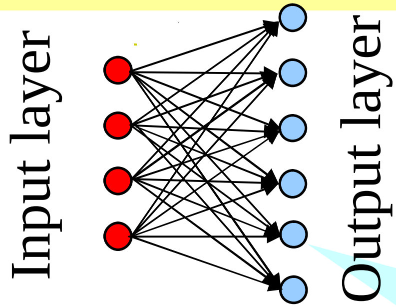


A non-linear model for a neurone

Biological and Artificial Neural Network

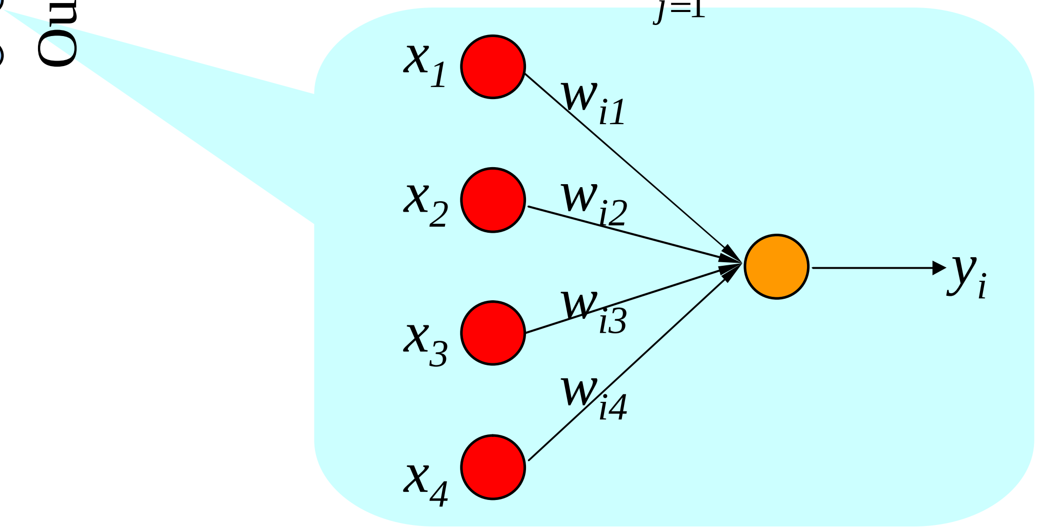
- **Biological NN**
  - Cell Body
  - Dendrite
  - Soma
  - Axon
- **Artificial NN**
  - Neurons
  - Weights or interconnections
  - Net Input
  - Output

A Single Layer Perceptron Network



For each node, the output is given by

$$y_i = g\left(\sum_{j=1}^{N} w_{ij} x_j - m_i\right)$$



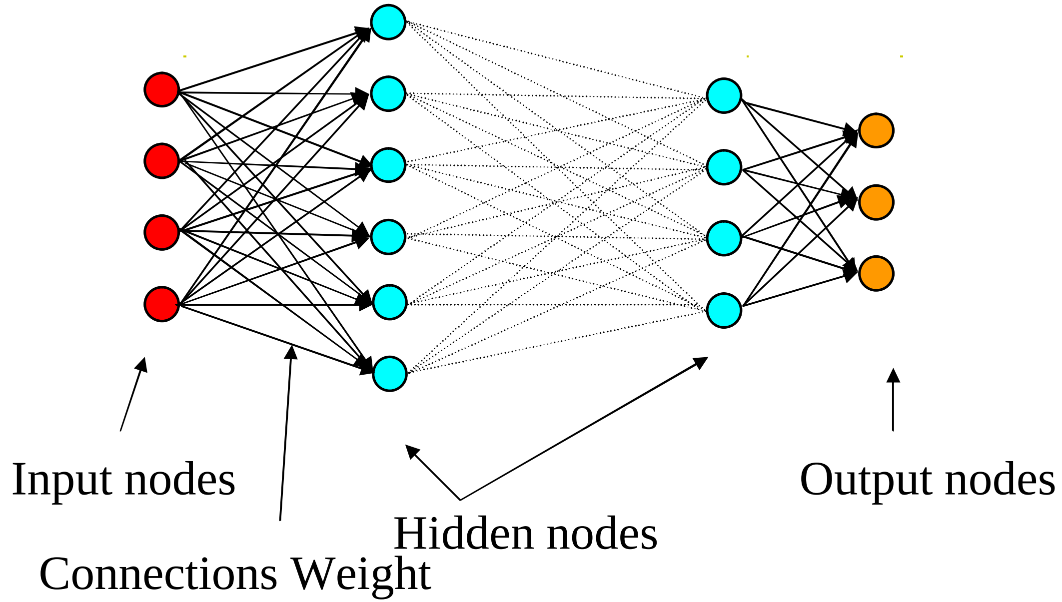
w_{ij} = Connection *weight* of branch *(i,j)*

x_j = *Input data* from node *j* in the input layer

m_i = *Bias* or *Threshold* value of node *i* in the output layer

g = *Activation* or *Transfer function*

(Artificial) Neural Network



Output of each node (mathematical equation)

$$y_i = f(w_i^1 x_1 + w_i^2 x_2 + w_i^3 x_3 + \cdots + w_i^m x_m)$$
$$= f(\sum_j w_i^j x_j)$$

X_i = input from the other nodes $\quad$ w_i^j = weight of each connection

Mathematic Foundation for Neural Network

- **Linear Algebra is the core of the mathematics required for understanding neural network**
  - Vectors and Vector Spaces
  - Inner Product, Norm and Orthogonality
  - Matrices, Linear Transformations
  - Eigenvalues and Eigenvectors
  - Etc.
- **Calculus**
  - Differentiation and Integration
- **Probability**
- **Set Theory**

Machine Learning

- **Machine learning is programming computers to optimize a performance criterion using example data or past experience.**
- **We need learning in cases where we cannot directly write a computer program to solve a given problem.**
- **Learning is used when:**
  - Human expertise does not exist (navigating on Mars),
  - Humans are unable to explain their expertise (speech recognition)
  - Solution changes in time (routing on a computer network)
  - Solution needs to be adapted to particular cases (user biometrics)

Three Types of Learning

- **Supervised Learning**
  - Learning (Training) with teacher (Supervisor)
  - There exist input (training) data and right output (response) target.
- **Unsupervised Learning**
  - Learning (Training) without teacher (Supervisor)
  - There exist input (training) data without right output (response) target.
- **Reinforcement Learning**
  - Learning (Training) assumed with teacher (Supervisor)
  - There exist input (training) data and output (response) target (but do not know right or not).

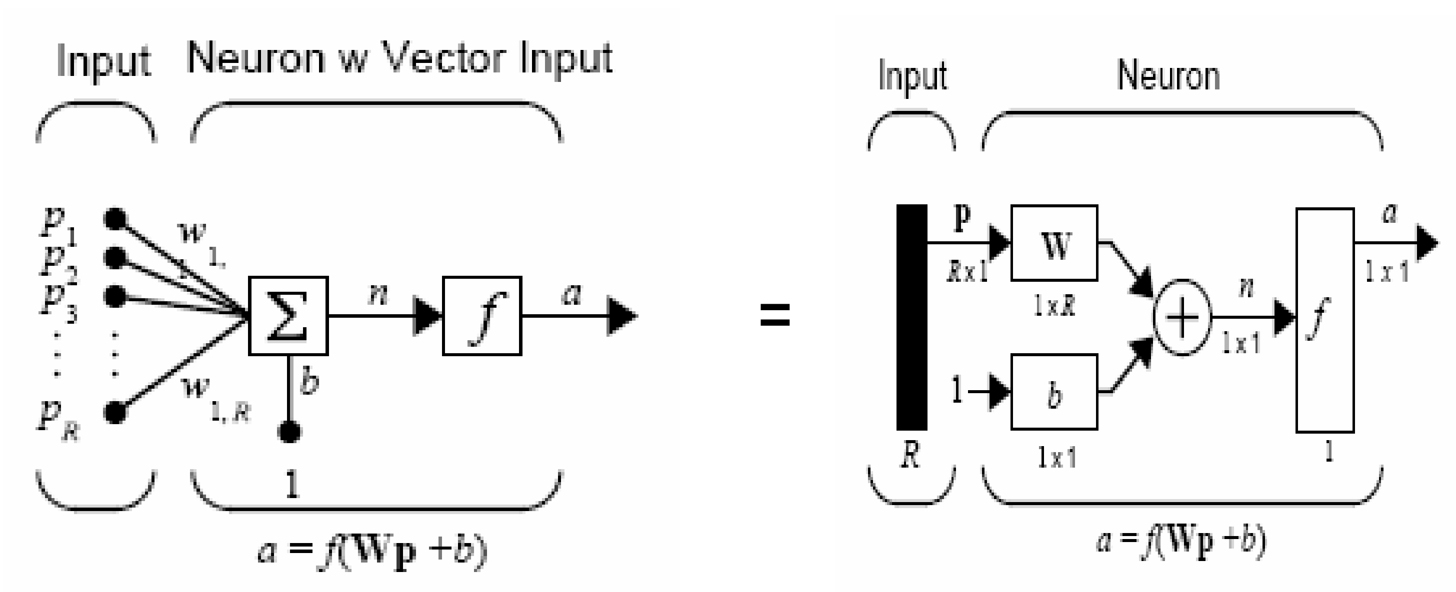
Neural Network Model

- **Supervised Learning**
  - Perceptron
  - Multi-Layer Perceptron (MLP)
    - Feed Forward Network
    - Back Propagation
  - Radius Bias Function (RBF)
  - Support Vector Machine (SVM)
  - k-Nearest Neighbour (k-NN)
  - etc.
- **Unsupervised Learning**
  - Clustering
  - Self Organisation Map (SOM)
  - Principal Component Analysis (PCA)
  - etc.
- **Reinforcement Learning**

Neural Network Applications

- Pattern Recognition
- Process identification & control
- Time series prediction
- Forecasting/Market Prediction: finance and banking
- Data Mining and Intelligent Data Analysis
- Medicine: analysis of electrocardiogram data, RNA & DNA sequencing, drug development without animal testing
- Control: process, robotics
- Manufacturing: quality control, fault diagnosis
- etc.

Neuron Model (One Neuron) by MATLAB



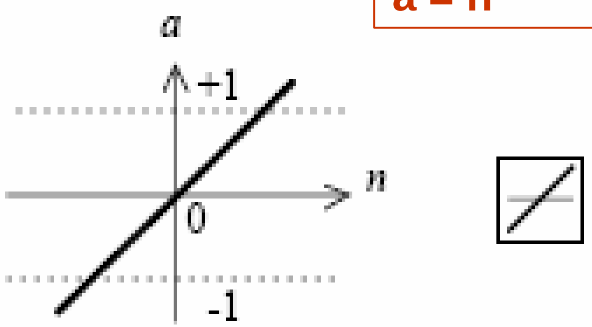
- $p_1, p_2, \ldots, p_R$ = inputs (vector)
- $w_1, w_2, \ldots, w_R$ = synaptic weights
- b = threshold (Bias)
- f = activation (transfer) function:
- a = output: $a = f(Wp+b) = f(n)$

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

```
n = W*p + b
```

activation (transfer) functions

a = n

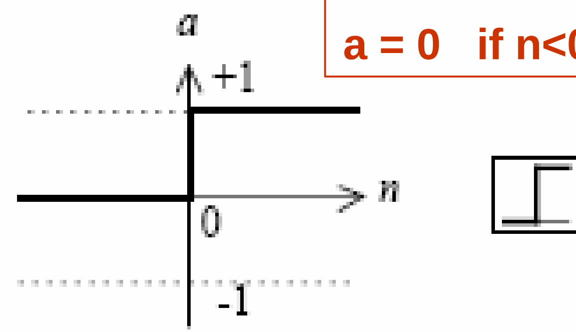


$a = purelin(n)$

Linear Transfer Function

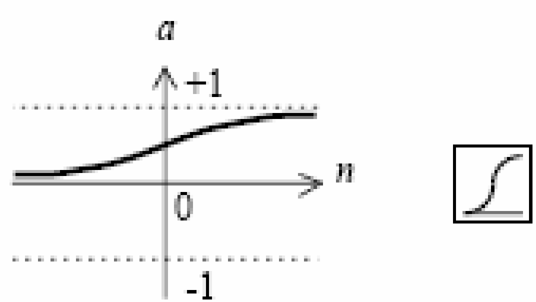
a = +1 if n >= 0

a = 0 if n<0



$a = hardlim(n)$

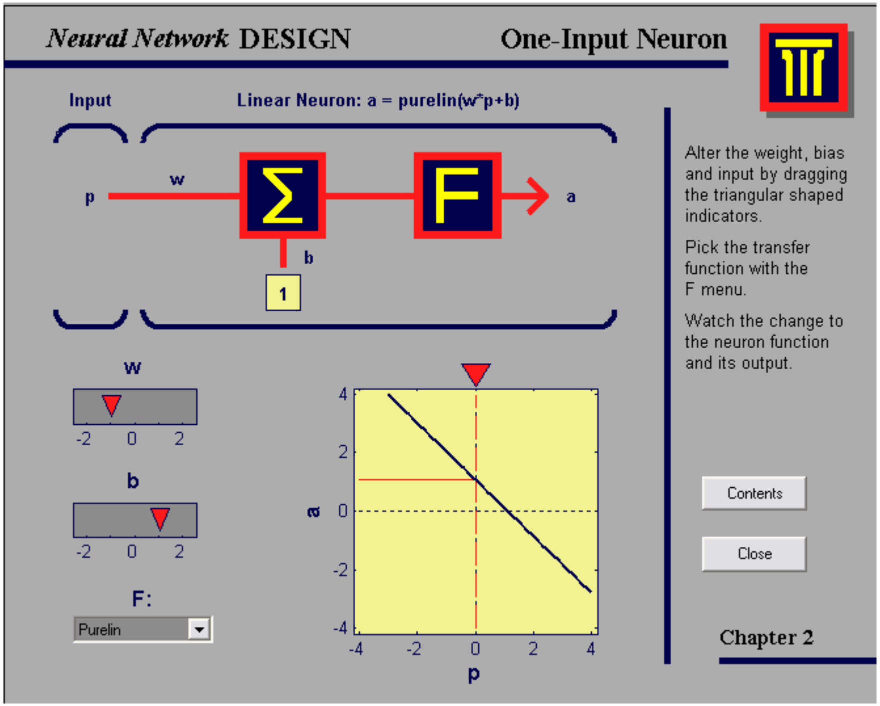
Hard-Limit Transfer Function



$a = logsig(n)$

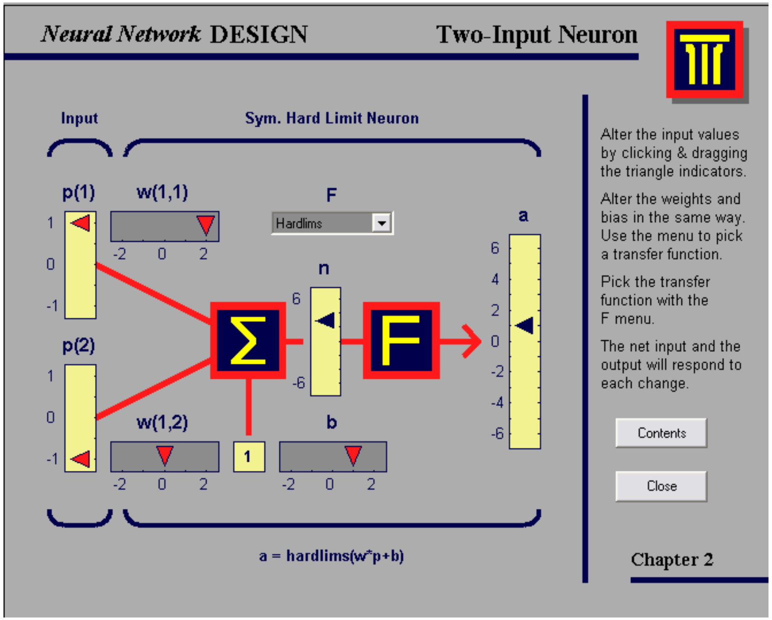
Log-Sigmoid Transfer Function

Neuron Model in MATLAB (Single/One-Input)



- p_1 – inputs (patterns) = 0
- w_1- synaptic weights = -1
- *b* - threshold (Bias) = 1
- *f* - activation (transfer) function: = pureline
- *a* - output: = 1

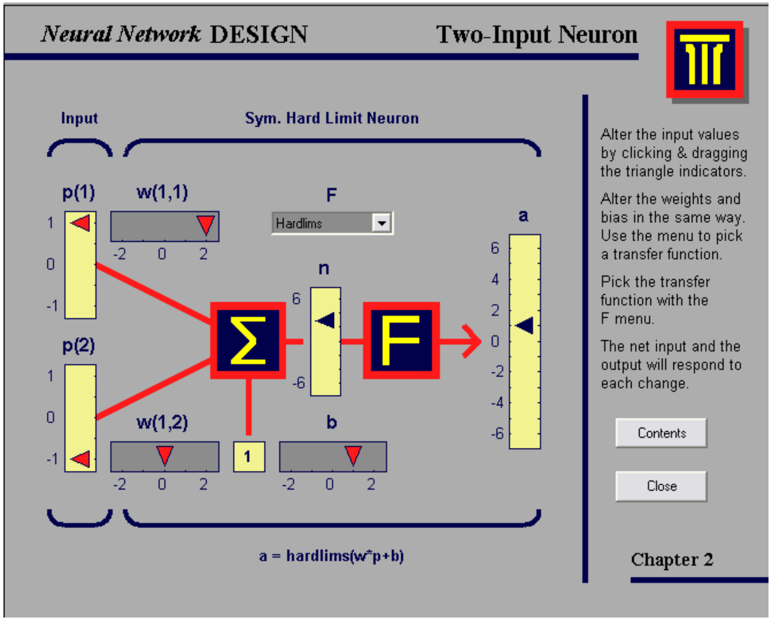
Neuron Model in MATLAB (Two-Input)



- p_1, p_2 - inputs (patterns) = (1,-1)
- w_1, w_2 - synaptic weights = (2,0)
- b - threshold (Bias) = 1
- f - activation (transfer) function:
  = Hard Limit
- a - output: = 1

```
% ex_One_Neural.m
p = [1; -1];      % R = 2
W = [2 0];
b = 1;
n = W*p + b;
a = hardlim(n);
```

Neuron Model in MATLAB (Two-Input)



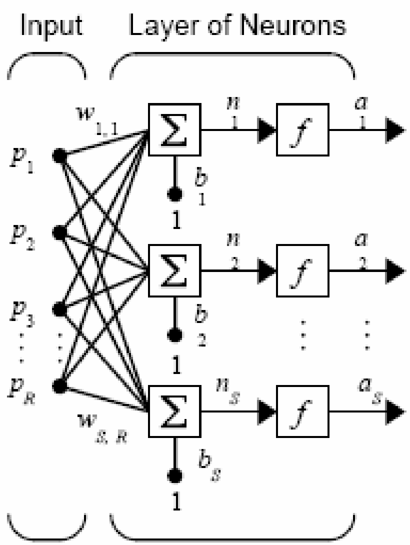
- p_1, p_2 - inputs (patterns) = (1,-1)
- w_1, w_2 - synaptic weights = (2,0)
- b - threshold (Bias) = 1
- f - activation (transfer) function:
  = Sigmoid
- a - output: = 0.9526

```
% ex_One_Neural.m
p = [1; -1];      % R = 2
W = [2 0];
b = 1;
n = W*p + b;
a2 = sigmoid(n);
```

Neuron Model (S Neuron)

A Layer of Neurons

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



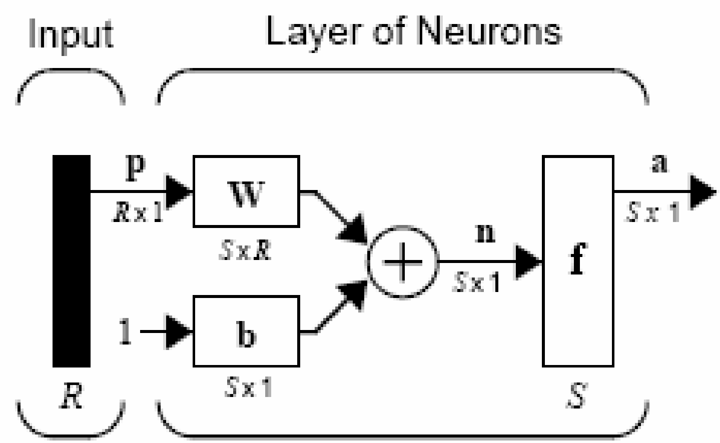
$$\mathbf{a} = \mathbf{f}\,(\mathbf{W}\mathbf{p}+\mathbf{b})$$

Where...

R = number of elements in input vector

S = number of neurons in layer

=



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

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

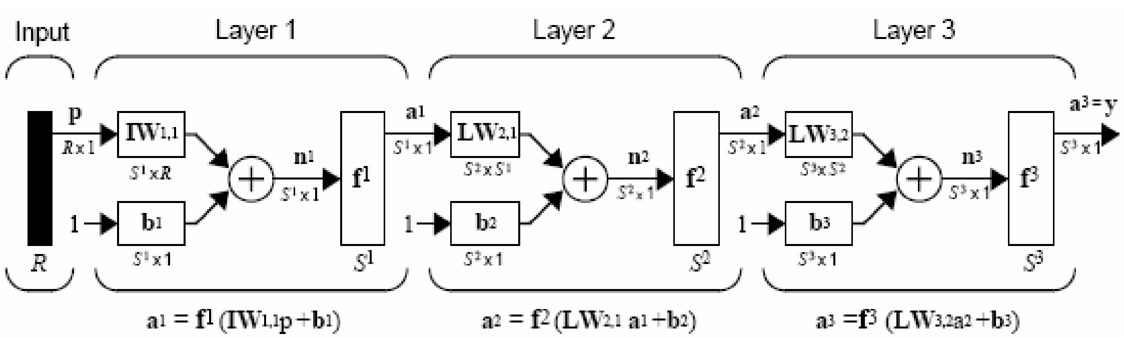
$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ & & & \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

```
% ex_S_Neural.m
p = [1; -1];
W = [2 0; 1 2; 1 3];
b = [1; 2; 3];
n = W*p + b;
a = hardlim(n);
a2 = sigmoid(n);
```

Multiple layer of Neuron

```
% ex_Multilayer_Neural2.m
% Input vector % R = 2 (Column)
p    = [1; -1];
% Weight & bias for Layer1, 2, 3
W1 = [2 0; 1 2; 1 3];         % Layer1: S=3 Nodes
 b1 = [1; 2; 3];
W2 = [2 0 1; 1 2 1];          % Layer2: S=2 Nodes
b2  = [0; 0];
W3 = [2 0];                   % Layer3: S=1 Node
b3  = [1];
a3 = sigmoid(W3*(sigmoid(W2*(sigmoid(W1*p+b1))'+b2))'+b3)
```

=



$a^3 = f^3(LW_{3,2}\, f^2(LW_{2,1} f^1(IW_{1,1}p + b^1) + b^2) + b^3 = y$

Programming in MATLAB Exercise

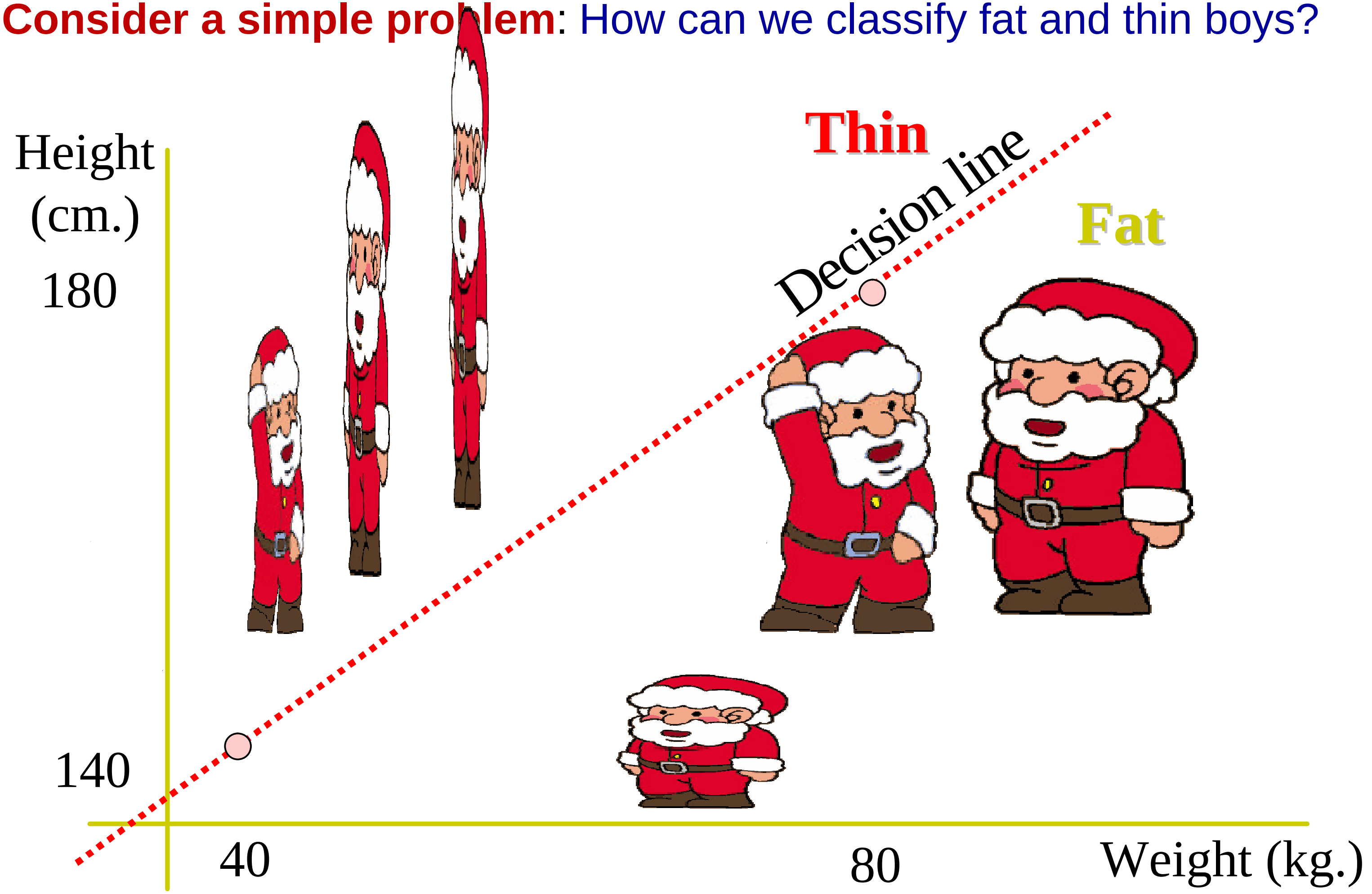
- **Exercise:**

1. **Write MATLAB to solve the question 8 in Exercise 1.**

2. **Write MATLAB to solve the question 3 in Exercise 2.**

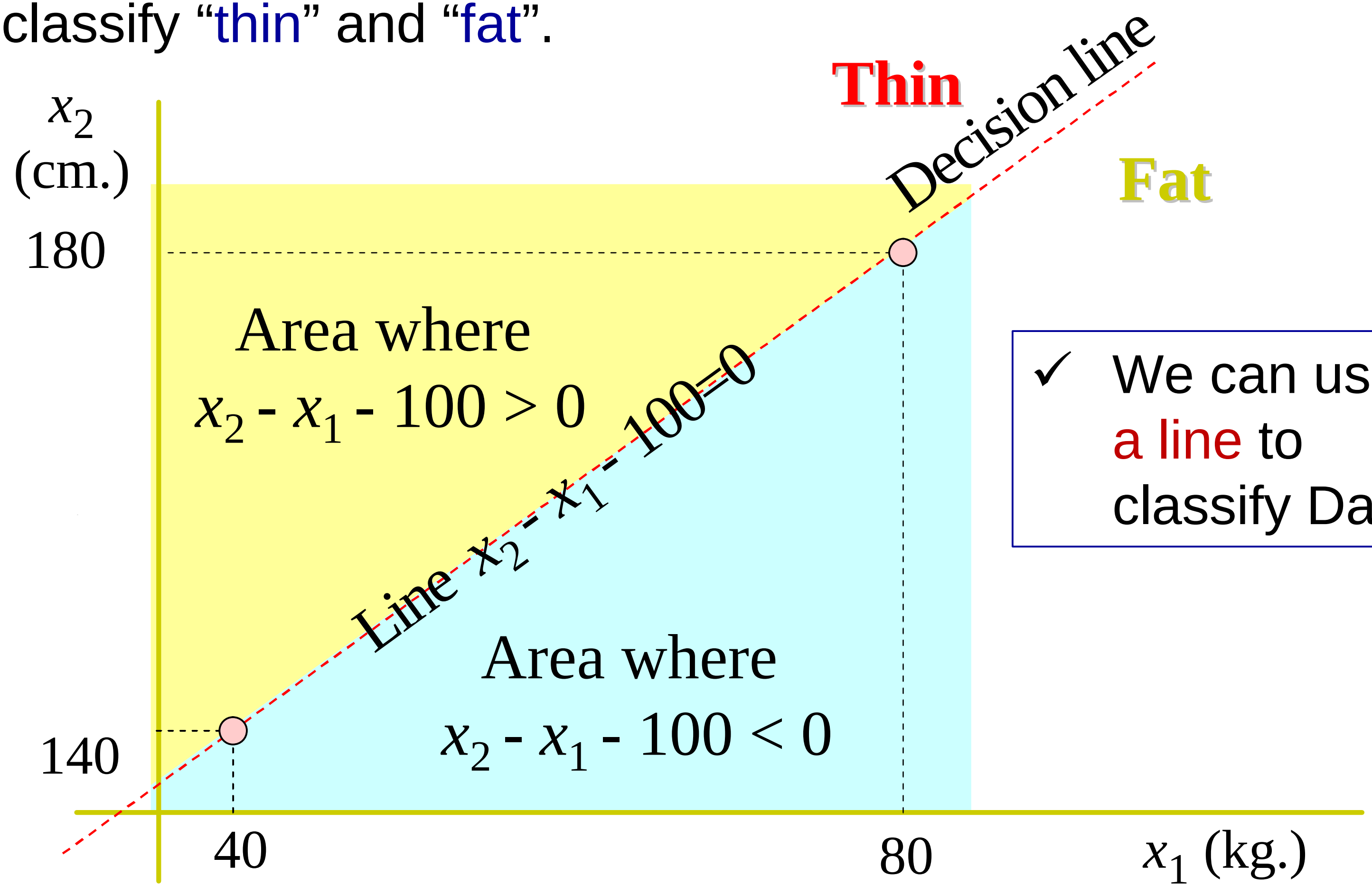
How can we use a mathematical function to classify ?

- **Consider a simple problem**: How can we classify fat and thin boys?



How can we use a mathematical function to classify ?

- We used 2 inputs, weight (x_1) and height (x_2), to classify "thin" and "fat".



Weight-height space

✓ We can use a line to classify Data.

How can we use a mathematical function to classify ?

- We can write the decision function for classifying "thin" and "fat" as follows:

$$y = \begin{cases} 1 \text{ (thin)} & \text{if } \; x_2 - x_1 - 100 \geq 0 \\ 0 \text{ (fat)} & \text{if } \; x_2 - x_1 - 100 < 0 \end{cases}$$

or

$$\begin{aligned} y &= g(w_1 x_1 + w_2 x_2 - m) \\ &= g(-x_1 + x_2 - 100) \end{aligned}$$

where

$$g(z) = \begin{cases} 1 & \text{if } \; z \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

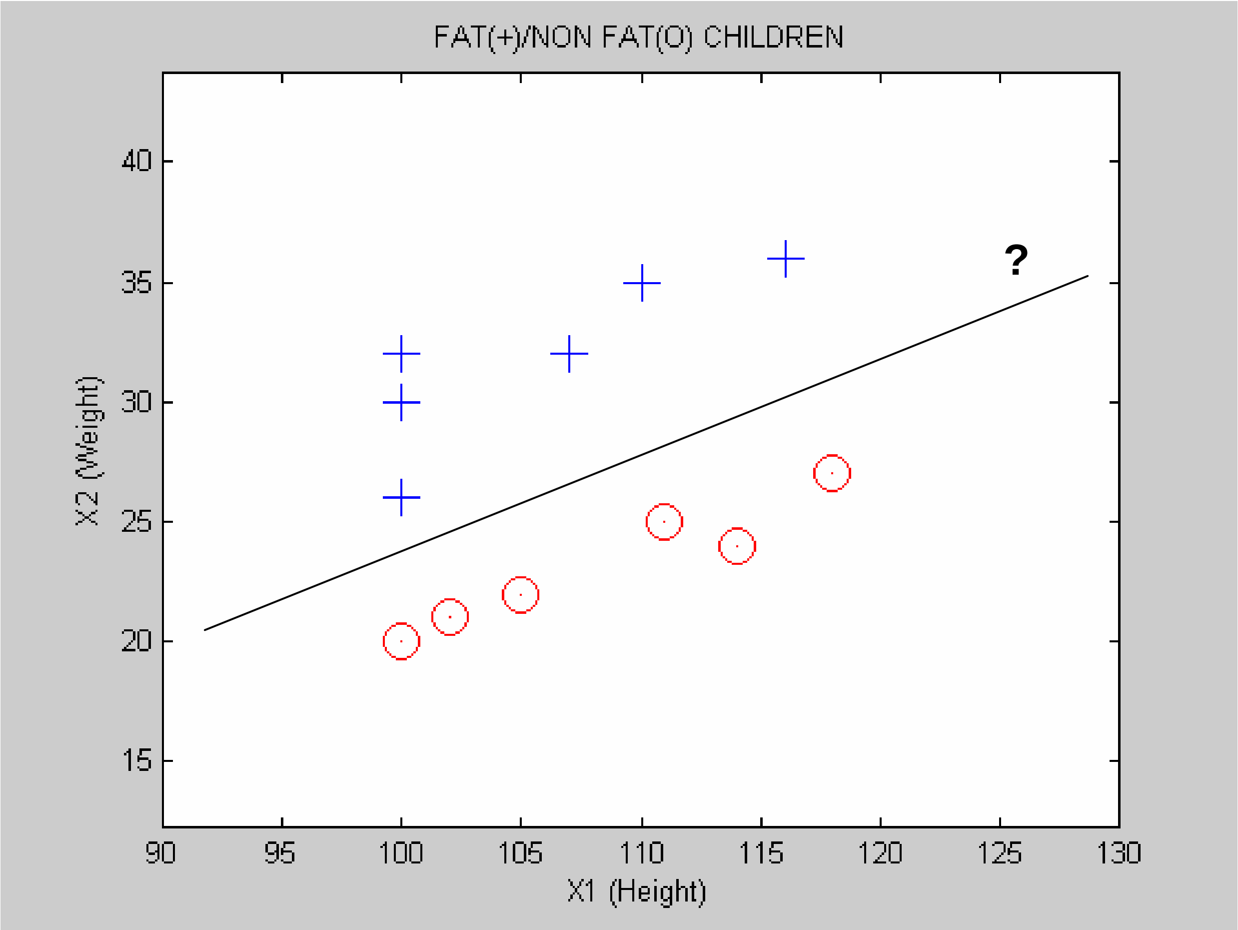
Advantage: Universal linear classifier

Problem: For a particular problem, **how can we choose suitable weights *w* and bias *μ* of the function ?**

Fat Children Training Samples

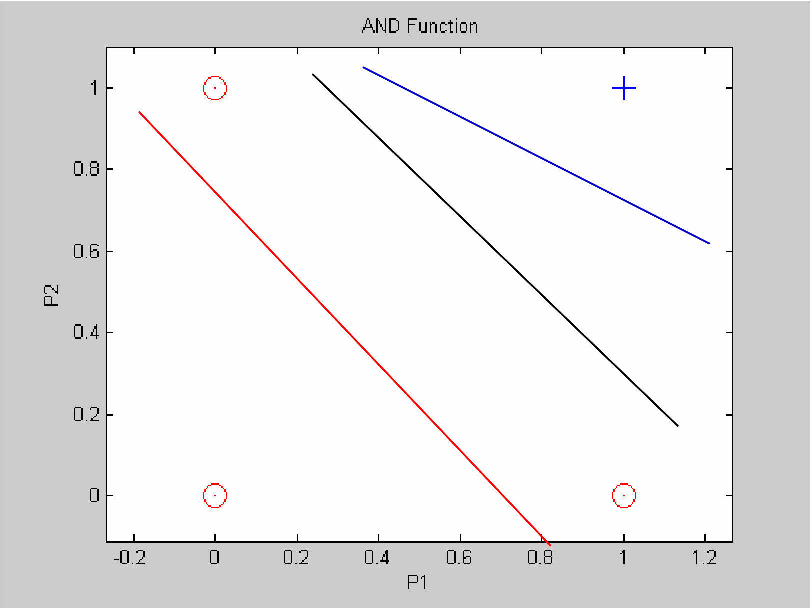
No.	Height (Cm.)	Weight (Kg.)	Fat/Not Fat (+1/-1)
1.	100	20	-1
2.	100	26	+1
3.	100	30	+1
4.	100	32	+1
5.	102	21	-1
6.	105	22	-1
7.	107	32	+1
8.	110	35	+1
9.	111	25	-1
10.	114	24	-1
11.	116	36	+1
12.	118	27	-1

Fat Children Training Samples



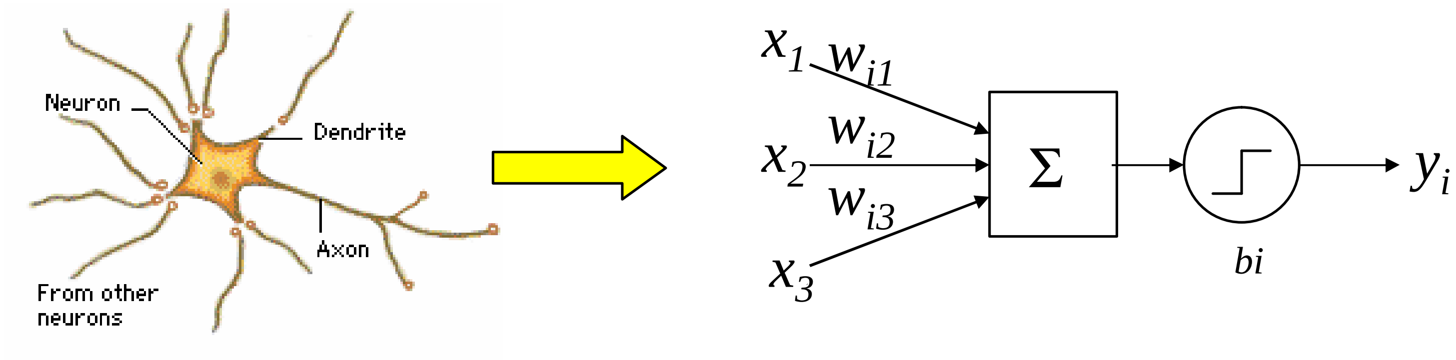
AND Function

No.	P1	P2	Output/Target
1.	0	0	0
2.	0	1	0
3.	1	0	0
4.	1	1	1



Supervised Learning: Perceptron Networks

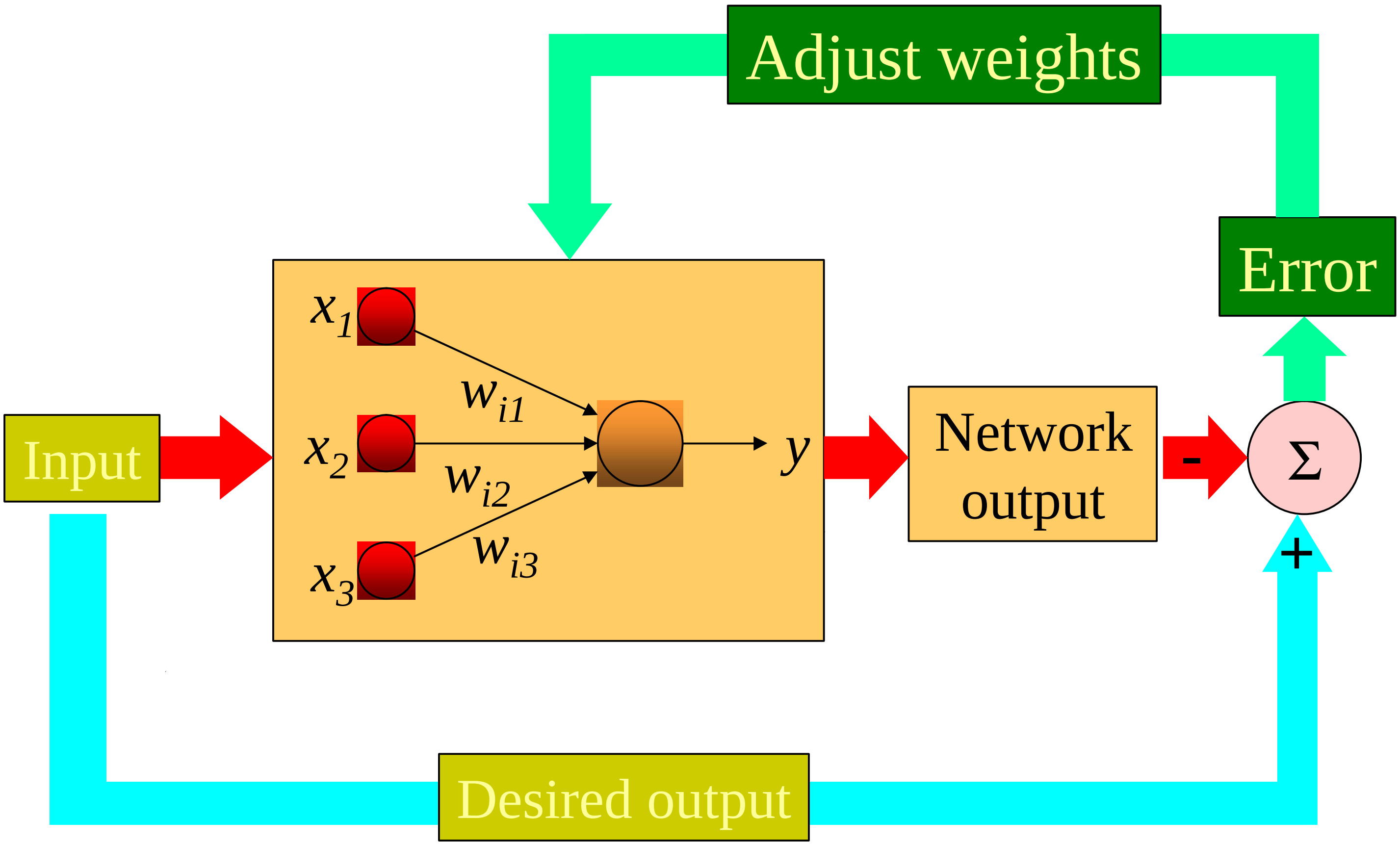
- Rosenblatt and his colleague proposed the Perceptron model
  in 1962 by inspiration of neural model from McCulloch-Pitts in 1943.



McCulloch-Pitts model

$$y_i = f(w_{i1}x_1 + w_{i2}x_2 + w_{i3}x_3 - b_i)$$
$$= f(\sum_j w_{ij}x_j - b_i)$$

Learning Algorithm: Training Perceptron Networks



Perceptron Learning Rule

The perceptron learning rule can be written more succinctly in terms of the error $\mathbf{e} = \mathbf{t} - \mathbf{a}$, and the change to be made to the weight vector $\Delta\mathbf{w}$:

CASE 1. If $\mathbf{e} = 0$, then make a change $\Delta\mathbf{w}$ equal to 0.

CASE 2. If $\mathbf{e} = 1$, then make a change $\Delta\mathbf{w}$ equal to $\mathbf{p}^T$.

CASE 3. If $\mathbf{e} = -1$, then make a change $\Delta\mathbf{w}$ equal to $-\mathbf{p}^T$.

All three cases can then be written with a single expression:

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

The Perceptron Learning Rule can be summarized as follows

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T \text{ and}$$

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

where $\mathbf{e} = \mathbf{t} - \mathbf{a}$.

Perceptron Learning Rule

The Perceptron Learning Rule can be summarized as follows

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T \text{ and}$$

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

where $\mathbf{e} = \mathbf{t} - \mathbf{a}$.

```
function [ W_new, b_new ] = perceptron(P,t,W,b)
% Perceptron.m (P,t,W,b);
% P = Input vector, t = target, W = Weight, b = bias
% Use Hard limit as Activation Function
a = hardlim(W*P'+b);
% e = error and Perceptron Rule as follow
e = t-a;
W_new = W + e.*P;
b_new = b + e;
end
```

Perceptron Training Algorithm

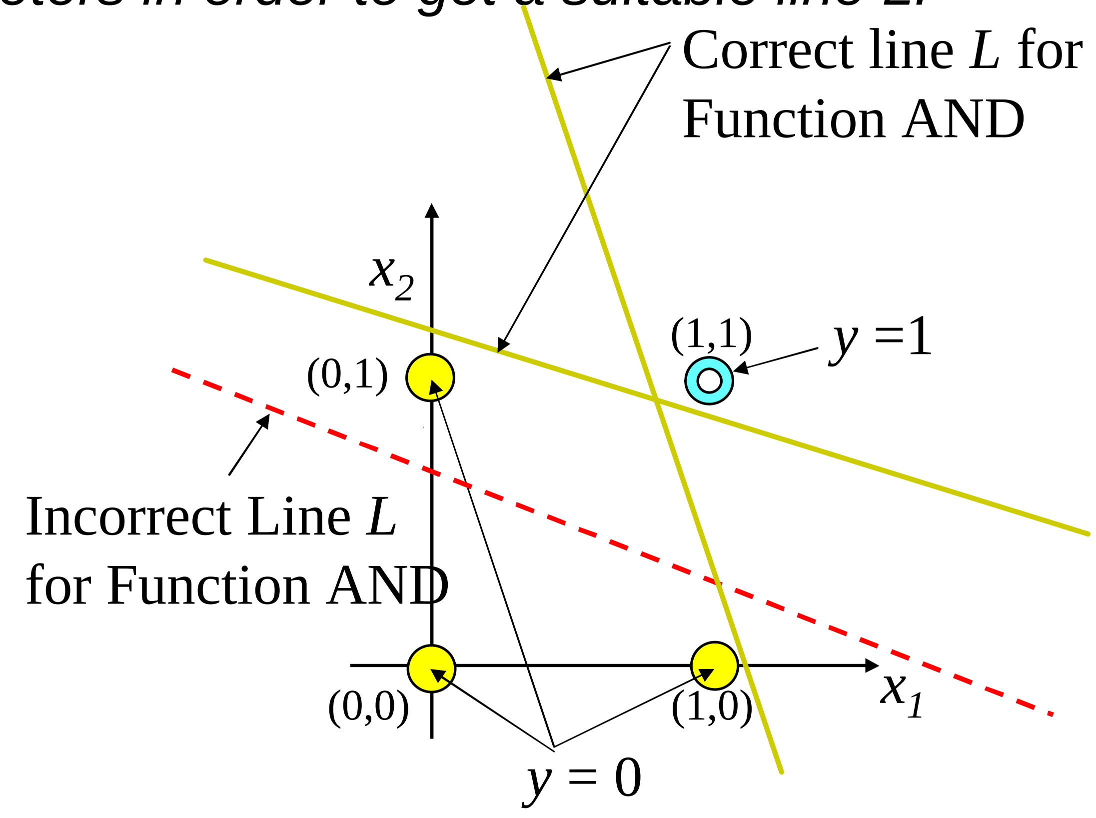
- **Step 1:** Initial weights and bias (can be zero or any).
- **Step 2:** For each training data (input and target) to be classified do **Steps 3-4**.
- **Step 3:** Calculate the response of output unit.
- **Step 4:** The weights and bias are updated by using this learning rule.
- **Step 5:** Test for stopping condition.
  may be the number of loops/iterations (**epoc**).

How a single layer perceptron works (cont.)

The Slope and Position of the line $L : w_1x_1+w_2x_2 - b = 0$ *depend on parameters* w_1, w_2, and b. *Therefore we have to adjust these parameters in order to get a suitable line L.*

Function AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



How a single layer perceptron works (cont.)

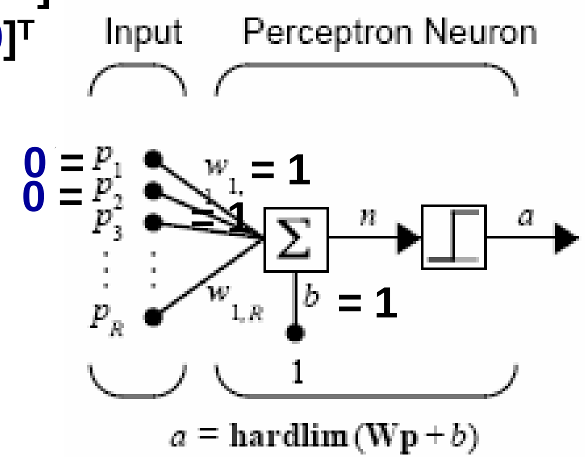
- Example: AND Function

```
% Run_AND_Perceptron.m
P = [0 0 1 1; 0 1 0 1];  % AND Function
T = [0 0 0 1];
plotpv(P,T,[-1, 2, -1, 2]); % plot data
% initial weight vector and bias
W = [1 1]; b = 1;
plotpc(W,b);        % plot line
epoc = 1;     % number of epoc
for j=1:epoc
    for i=1:size(P,2)
        p = P(:,i);
        t = T(i);
        [W b] = perceptron(p',t,W,b);
        plotpc(W,b);
    end
end
% test data test = [0 0]'
test = [0 0];
output = hardlim(W*test'+b)
```

AND Function

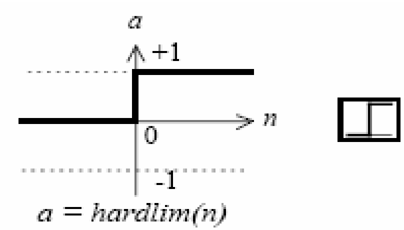
No.	P1	P2	Target	Output (n)	Actual (a)	W1	W2	b
						1	1	1
1.	0	0	0					
2.	0	1	0					
3.	1	0	0					
4.	1	1	1					

$P = [P1 \quad P2]^T$
$\quad = [0 \quad 0]^T$



Where...

R = number of elements in input vector

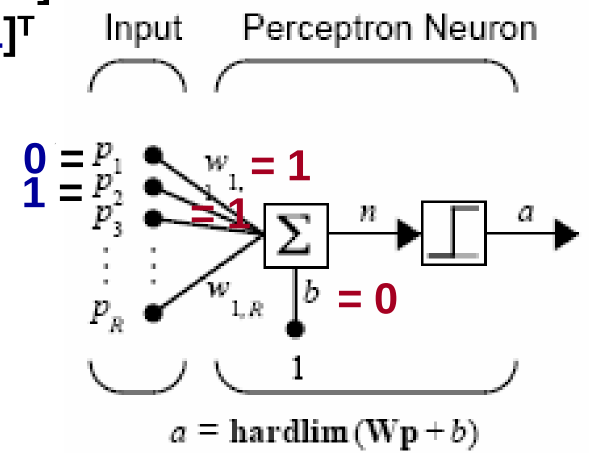


Hard-Limit Transfer Function

AND Function

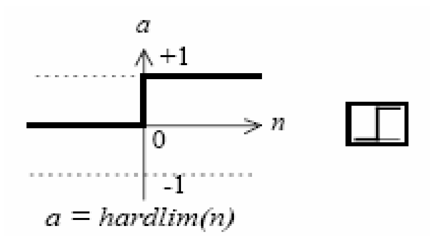
No.	P1	P2	Target	Output (n)	Actual (a)	W1	W2	b
						1	1	1
1.	0	0	0	1	1	1	1	0
2.	0	1	0					
3.	1	0	0					
4.	1	1	1					

$P = [P1 \quad P2]^T$
$\quad = [0 \quad 1]^T$



Where...

R = number of elements in input vector

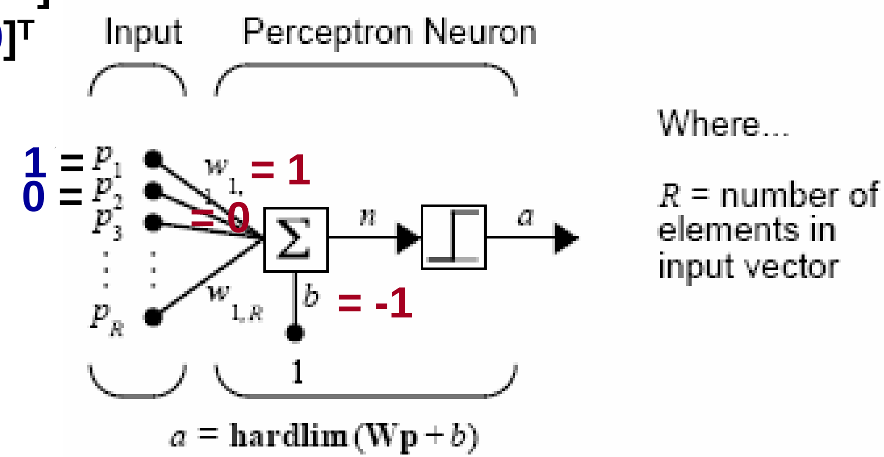


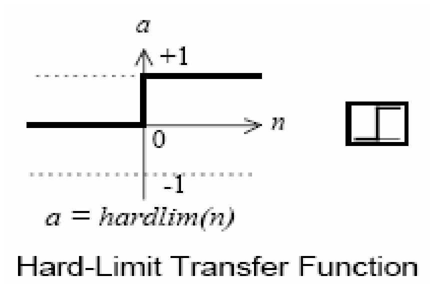
Hard-Limit Transfer Function

AND Function

No.	P1	P2	Target	Output (n)	Actual (a)	W1	W2	b
						1	1	1
1.	0	0	0	0	1	1	1	0
2.	0	1	0	1	1	1	0	-1
3.	1	0	0					
4.	1	1	1					

$P = [P1 \quad P2]^T$
$\quad = [1 \quad 0]^T$



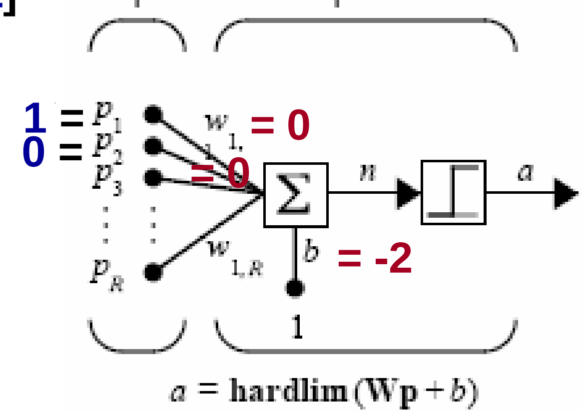


AND Function

No.	P1	P2	Target	Output (n)	Actual (a)	W1	W2	b
						1	1	1
1.	0	0	0	0	1	1	1	0
2.	0	1	0	1	1	1	0	-1
3.	1	0	0	0	1	0	0	-2
4.	1	1	1					

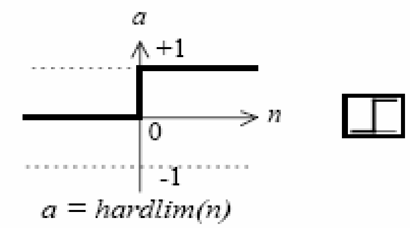
$P = [P1 \quad P2]^T$

$= [1 \quad 1]^T$



Where...

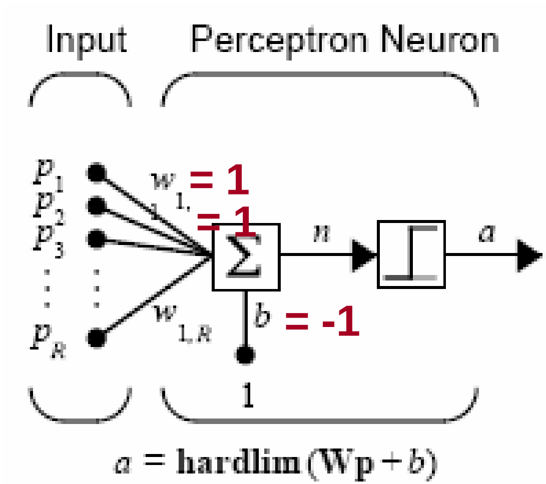
R = number of elements in input vector

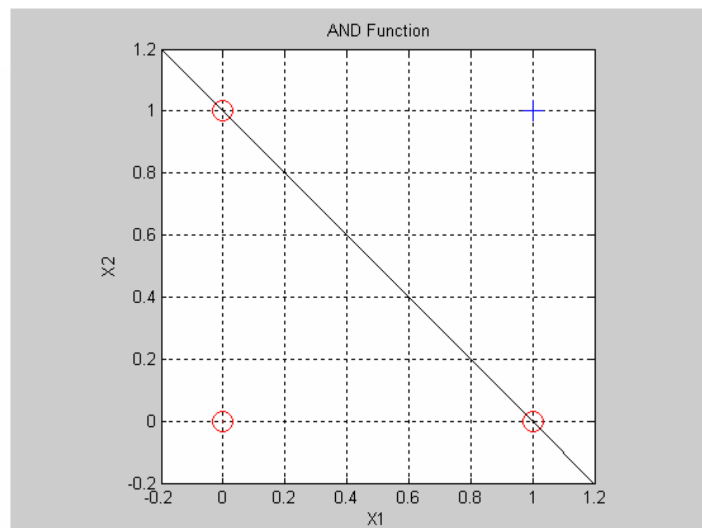


Hard-Limit Transfer Function

AND Function

No.	P1	P2	Target	Output (n)	Actual (a)	W1	W2	b
						1	1	1
1.	0	0	0	0	1	1	1	0
2.	0	1	0	1	1	1	0	-1
3.	1	0	0	0	1	0	0	-2
4.	1	1	1	-2	0	1	1	-1





AND Function

No.	P1	P2	Target	Actual Output	W1	W2	b
					1	1	1
1.	0	0	0	1	1	1	0
2.	0	1	0	1	1	0	-1
3.	1	0	0	1	0	0	-2
4.	1	1	1	0	1	1	-1

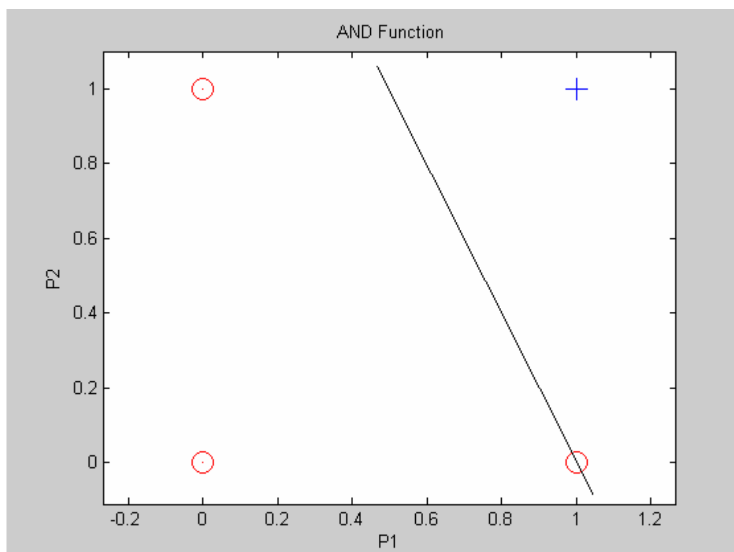
1. epoch

No.	P1	P2	Target	Actual Output	W1	W2	b
					1	1	-1
1.	0	0	0	1	1	1	-1
2.	0	1	0	1	1	0	-2
3.	1	0	0	1	1	0	-2
4.	1	1	1	0	2	1	-1

2. epoch

AND Function

No.	P1	P2	Target	Actual Output	W1	W2	b
					2	1	-1
1.	0	0	0	1	2	1	-1
2.	0	1	0	1	2	0	-2
3.	1	0	0	1	1	0	-3
4.	1	1	1	0	2	1	-2



Learning Algorithm: Training Perceptron Networks (cont.)

Function AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

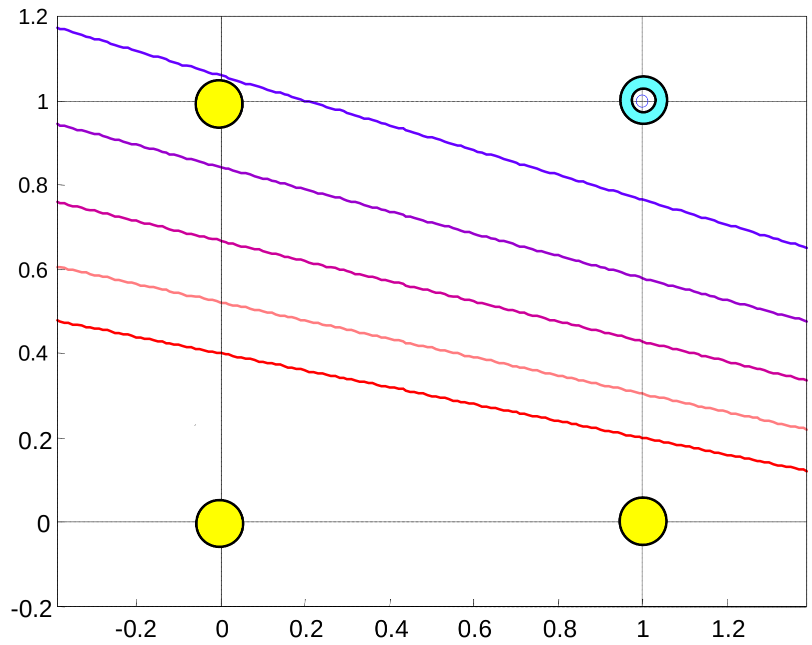
epoc: 16

epoc: 12

epoc: 8

epoc: 4

Initial



Initial Values

$w_1 = 0.5$

$w_2 = 2.5$

q (b) = 1.0

$a = 0.2$

Perceptron Learning Rule depends on initial values

$$a*(t-o)$$

$$\sum X_i * W_i$$

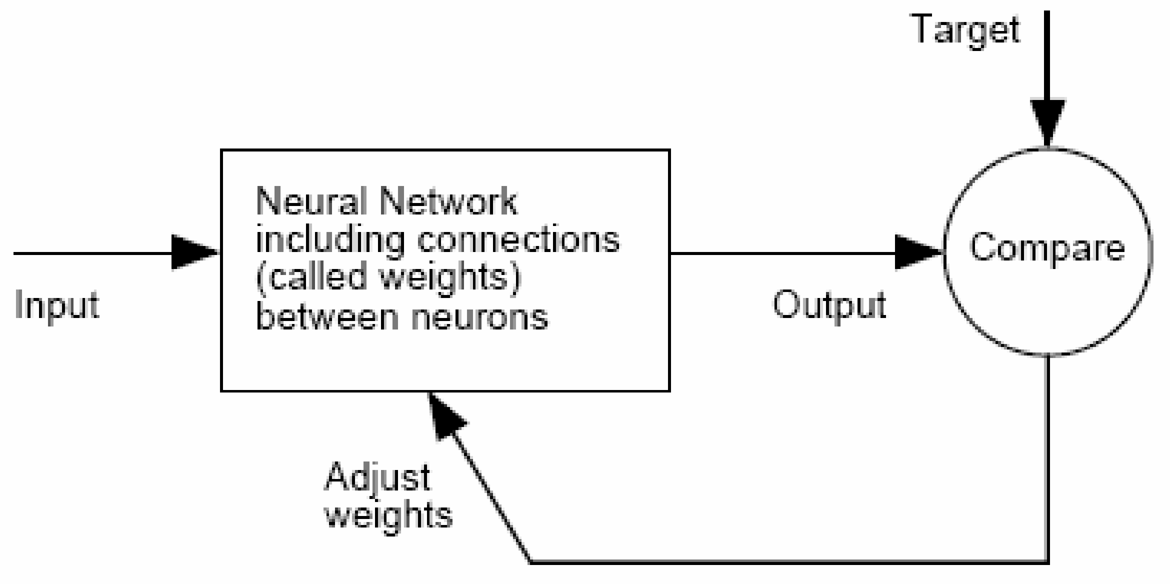
	Perceptron Learning Example: AND Gate									Bias: Let X0 = 1 , Alpha = 0.5			
	Input							Output		Alpha =	0.5		
	X0	X1	X2	1.0 * W0	X1* W1	X2* W2	Total	target	actual	Alpha * (t-o)	W0	W1	W2
											0.1	0.1	0.1
1	1	0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	0.10	0.10
	1	0	1	-0.40	0.00	0.10	-0.30	0	0	0.00	-0.40	0.10	0.10
	1	1	0	-0.40	0.10	0.00	-0.30	0	0	0.00	-0.40	0.10	0.10
	1	1	1	-0.40	0.10	0.10	-0.20	1	0	0.50	0.10	0.60	0.60
2	1	0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	0.60	0.60
	1	0	1	-0.40	0.00	0.60	0.20	0	1	-0.50	-0.90	0.60	0.10
	1	1	0	-0.90	0.60	0.00	-0.30	0	0	0.00	-0.90	0.60	0.10
	1	1	1	-0.90	0.60	0.10	-0.20	1	0	0.50	-0.40	1.10	0.60
3	1	0	0	-0.40	0.00	0.00	-0.40	0	0	0.00	-0.40	1.10	0.60
	1	0	1	-0.40	0.00	0.60	0.20	0	1	-0.50	-0.90	1.10	0.10
	1	1	0	-0.90	1.10	0.00	0.20	0	1	-0.50	-1.40	0.60	0.10
	1	1	1	-1.40	0.60	0.10	-0.70	1	0	0.50	-0.90	1.10	0.60

Perceptron Learning Rule depends on initial values

	Input							Output		Alpha =	0.5		
	X0	X1	X2	1.0 * W0	X1* W1	X2* W2	Total	target	actual	Alpha * (t-o)	W0	W1	W2
4	1	0	0	-0.90	0.00	0.00	-0.90	0	0	0.00	-0.90	1.10	0.60
	1	0	1	-0.90	0.00	0.60	-0.30	0	0	0.00	-0.90	1.10	0.60
	1	1	0	-0.90	1.10	0.00	0.20	0	1	-0.50	-1.40	0.60	0.60
	1	1	1	-1.40	0.60	0.60	-0.20	1	0	0.50	-0.90	1.10	1.10
5	1	0	0	-0.90	0.00	0.00	-0.90	0	0	0.00	-0.90	1.10	1.10
	1	0	1	-0.90	0.00	1.10	0.20	0	1	-0.50	-1.40	1.10	0.60
	1	1	0	-1.40	1.10	0.00	-0.30	0	0	0.00	-1.40	1.10	0.60
	1	1	1	-1.40	1.10	0.60	0.30	1	1	0.00	-1.40	1.10	0.60
6	1	0	0	-1.40	0.00	0.00	-1.40	0	0	0.00	-1.40	1.10	0.60
	1	0	1	-1.40	0.00	0.60	-0.80	0	0	0.00	-1.40	1.10	0.60
	1	1	0	-1.40	1.10	0.00	-0.30	0	0	0.00	-1.40	1.10	0.60
	1	1	1	-1.40	1.10	0.60	0.30	1	1	0.00	-1.40	1.10	0.60
7	1	0	0	-1.40	0.00	0.00	-1.40	0	0	0.00	-1.40	1.10	0.60
	1	0	1	-1.40	0.00	0.60	-0.80	0	0	0.00	-1.40	1.10	0.60
	1	1	0	-1.40	1.10	0.00	-0.30	0	0	0.00	-1.40	1.10	0.60
	1	1	1	-1.40	1.10	0.60	0.30	1	1	0.00	-1.40	1.10	0.60

$a*(t-o)$ $isn't\ change$

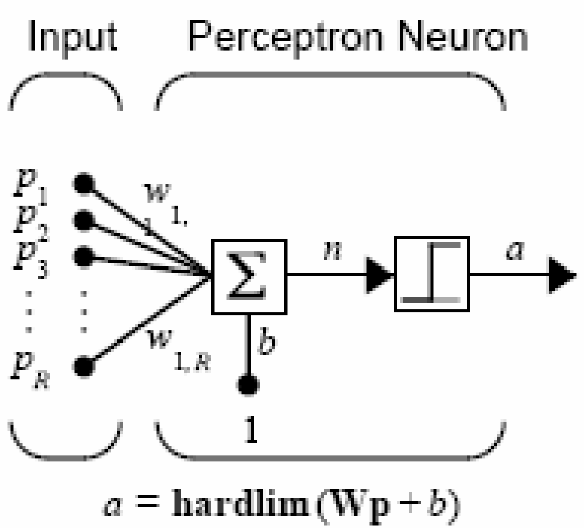
Supervised Learning



- ***Perceptron are learning by adjusting weights !***

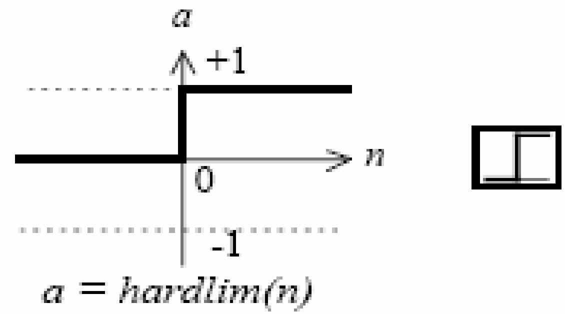
- ***Feeding Input and Output (target)***
  - ***One by one (Online Learning) Perceptron***
  - ***All in one (Batch Learning)***

Perceptron – One Neuron and one layer Model



Where...

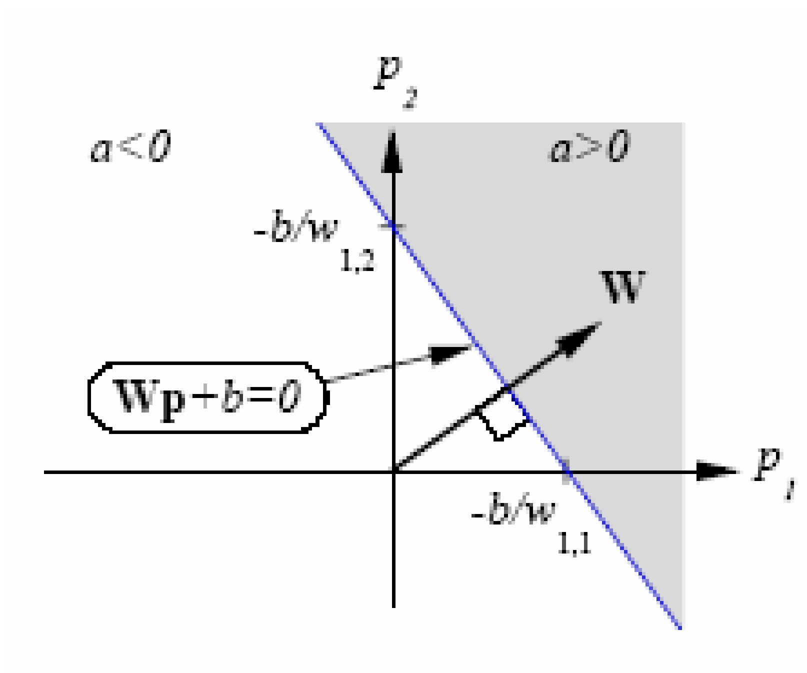
R = number of elements in input vector



Hard-Limit Transfer Function

- ***p_1, ..., p_R*** – **input patterns**
- ***w_1, ..., w_R*** **- synaptic weights**
- ***w_0 (b)*** **- threshold (Bias)**
- ***f(n)*** **- activation function: *f*(n) = hardlim(*n*)**
- ***a*** **- output: *a = f(n)***

Perceptron – As decision boundary



Two classification regions are formed by the *decision boundary* line L at $\mathbf{W}\mathbf{p} + b = 0$. This line is perpendicular to the weight matrix **W** and shifted according to the bias b. Input vectors above and to the left of the line L will result in a net input greater than 0; and therefore, cause the hard-limit neuron to output a 1. Input vectors below and to the right of the line L cause the neuron to output 0. The dividing line can be oriented and moved anywhere to classify the input space as desired by picking the weight and bias values.

<u>Linear equation</u>:

y = ax+b, where a is a slop.

ax – y + b **= 0**

is equal to

$\mathbf{w_1\,x_1 + w_2\,x_2 +}\ b = 0$

or

Wp + b = 0

Perceptron by using MATLAB toolbox

NEWP Create a perceptron.

Syntax

```
net = newp
net = newp(pr,s,tf,lf)
```

Description

Perceptrons are used to solve simple (i.e. linearly separable) classification problems.

- **PR** - Rx2 matrix of min and max values for R input elements.
- **S** - Number of neurons.
- **TF** - Transfer function, default = 'hardlim'.
- **LF** - Learning function, default = 'learnp'.

Returns a new perceptron.

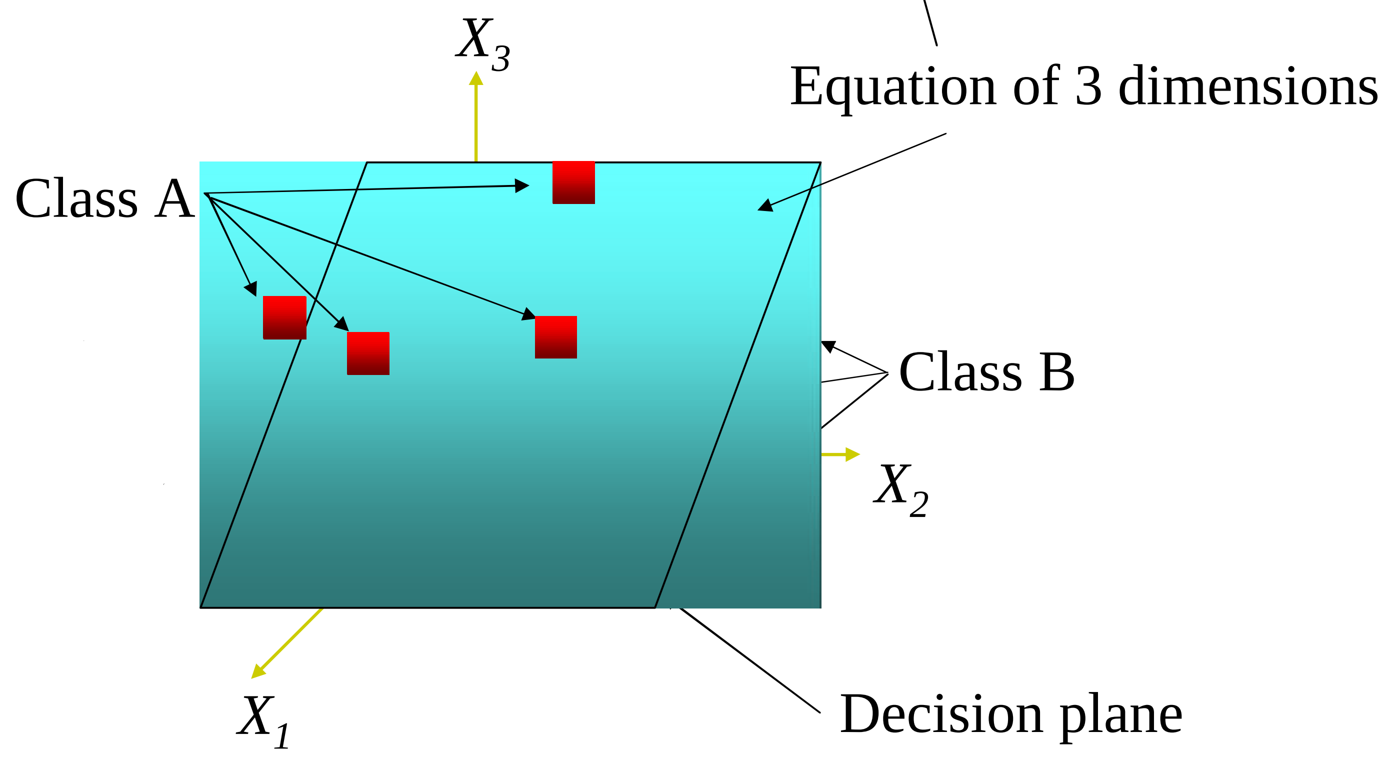
AND Function by using MATLAB toolbox

```
net = newp([0 1; 0 1],1);    %This code creates a perceptron layer with one  2-element
                             % input (ranges [0 1] and [0 1]) and one neuron.
                             % Initial weight [0 0] and bias 0

P = [0 0 1 1; 0 1 0 1];      % And Input patterns
T = [0 0 0 1];               % Target

net.IW{1} = [1 1]            % display weight vector
net.b{1} = 1                 % display bias

net.trainParam.epochs = 2;   % train (learn) for a maximum of  2 epochs
net = train(net,P,T);

a = sim(net,P);              % simulate the network's output again (test pattern)
a = [0 1 1 1]                % Result from test pattern
```

Higher Dimension Feature Space

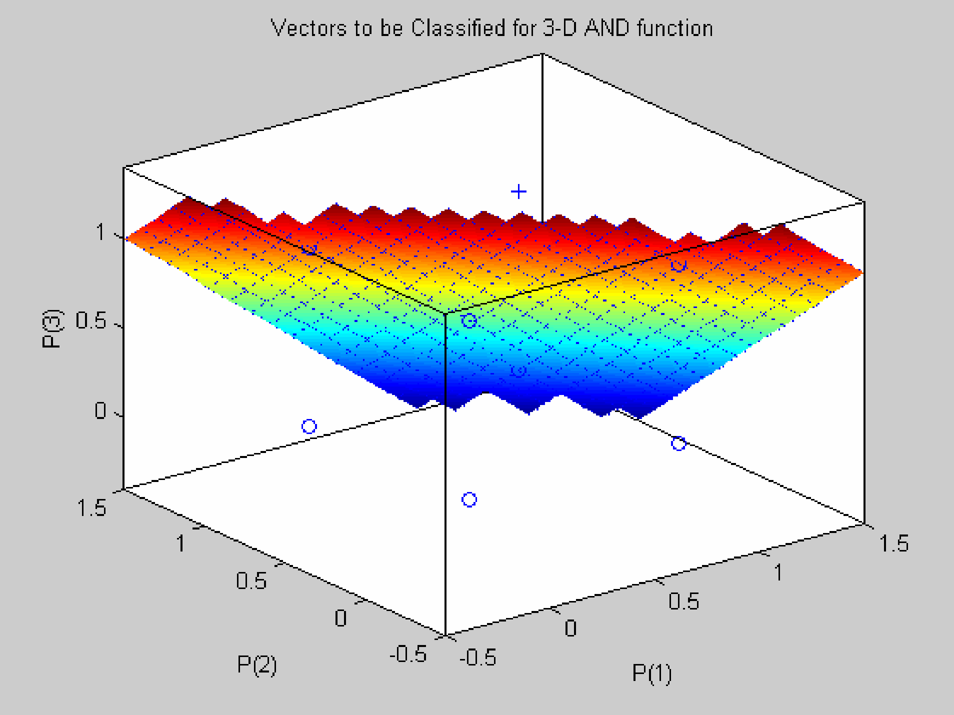
In case of 3 dimensions input data, decision function is hyper plane

$$w_1x_1 + w_2x_2 + w_3x_3 - q = 0$$



3-D AND Function by using MATLAB toolbox

```
% 3D perceptron
net = newp([0 1; 0 1; 0 1],1);
P = [0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1]';  % AND Function
T = [0; 0; 0; 0; 0; 0; 0; 1]';
% initial weight and bias
net.IW{1} = [1 1 1]; net.b{1} = 0;
% train or learning
net.trainParam.epochs = 1;
net = train(net,P,T);
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
```

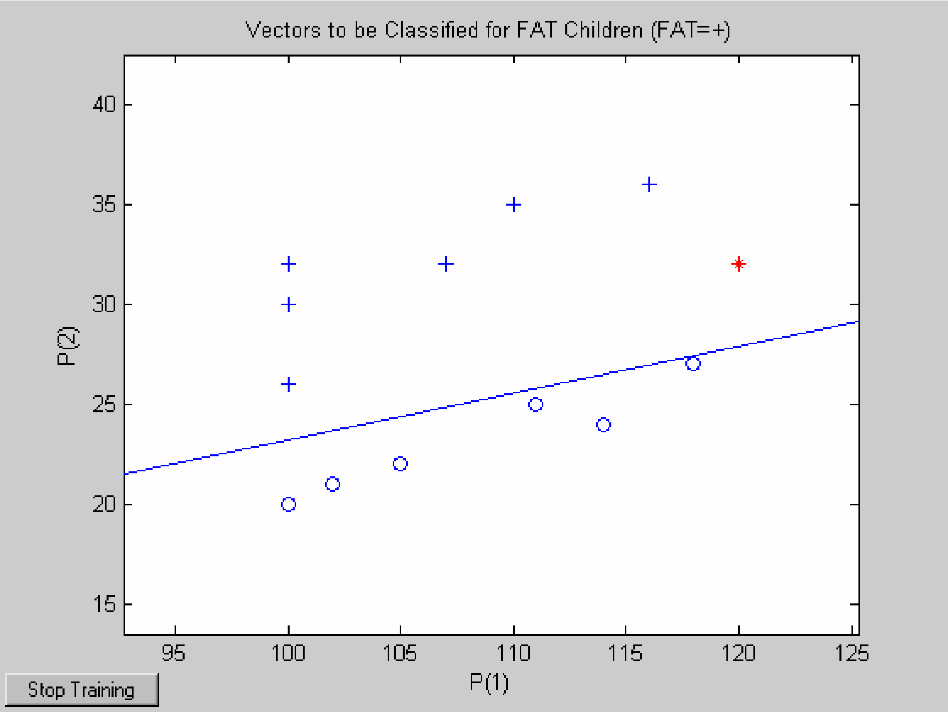


OR Function by using MATLAB

```
net = newp([0 1; 0 1],1);    %This code creates a perceptron layer with one  2-element
                             % input (ranges [0 1] and [0 1]) and one neuron.
                             % Initial weight [0 0] and bias 0

P = [0 0 1 1; 0 1 0 1];      % OR Input patterns
T = [0 1 1 1];               % Target

net.IW{1} = [1 1]            % display weight vector
net.b{1} = 1                 % display bias

net.trainParam.epochs = 2;   % train (learn) for a maximum of  2 epochs
net = train(net,P,T);

a = sim(net,P);              % simulate the network's output again (test pattern)
a = [? ? ? ?]                % Result from test pattern
```

FAT Children by using MATLAB toolbox

```
% FAT Children train by perceptron
net = newp([90 130; 10 50],1);
P = [100 20; 100 26; 100 30; 100 32; 102 21; 105 22; 107 32; 110 35; 111 25;
      114 24; 116 36; 118 27] ';          % Fat Children
T = [0; 1; 1; 1; 0; 0; 1; 1; 0; 0; 1; 0] ';    % 1 = fat, 0 = not fat
net.trainParam.epochs = 40;
net = train(net,P,T);
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
```




```
% FAT Children test a new data (*)
NP = [120 32]';
hold on;
plot(NP(1),NP(2),'*r');
Y = sim(net,NP)
```

Programming in MATLAB Exercise

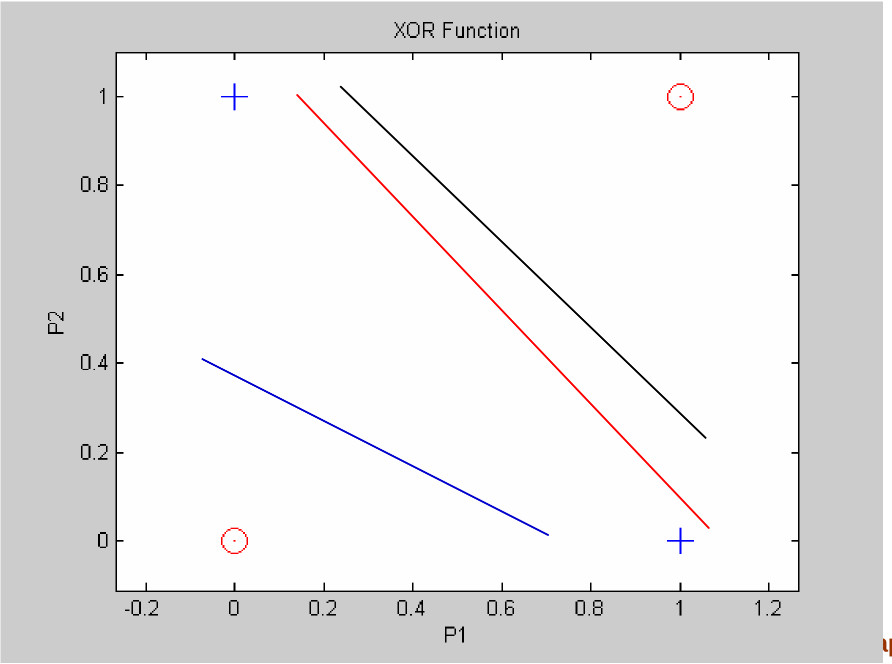
- **Exercise:**

  1. Write MATLAB to solve the question 3 (OR Function) in Exercise 3.

  2. Write MATLAB to solve the question 4 (FAT Child Problem) in Exercise 3.

Limitation of Perceptron (XOR Function)

No.	P1	P2	Output/Target
1.	0	0	0
2.	0	1	1
3.	1	0	1
4.	1	1	0



?