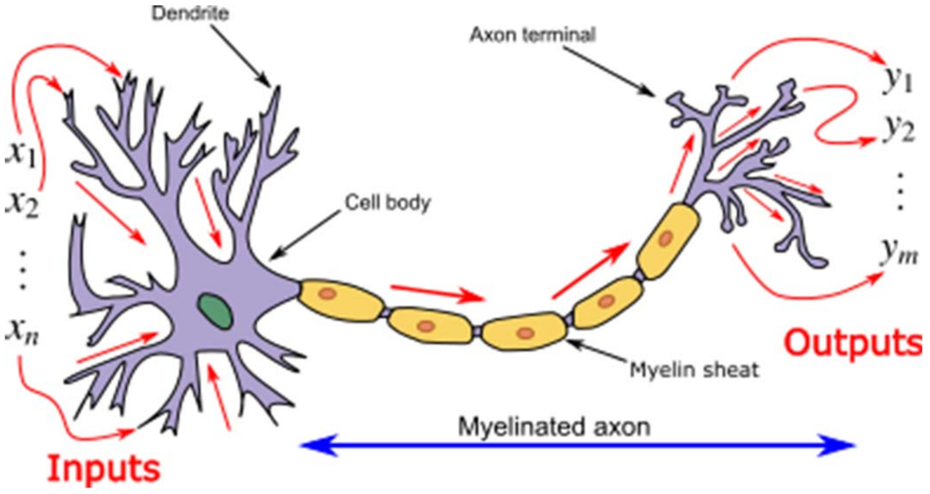
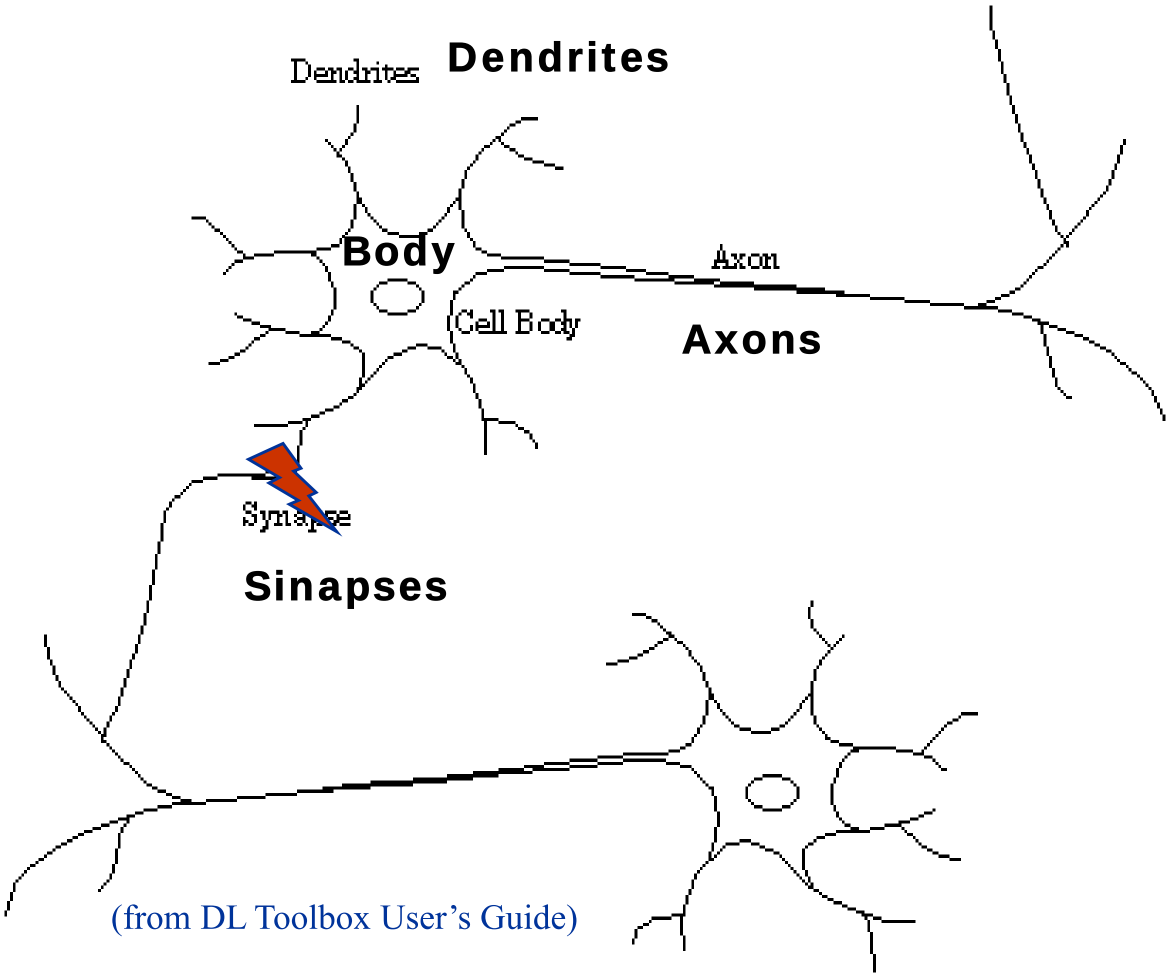
Chapter 4

Neurons, Layers, (Neural) Networks



https://en.wikipedia.org/wiki/Artificial_neuron 11 Sept 2023

Biological Neurons



(from DL Toolbox User's Guide)

Brain:

– 10^{11} neurons

– 10^{4} connections by neuron

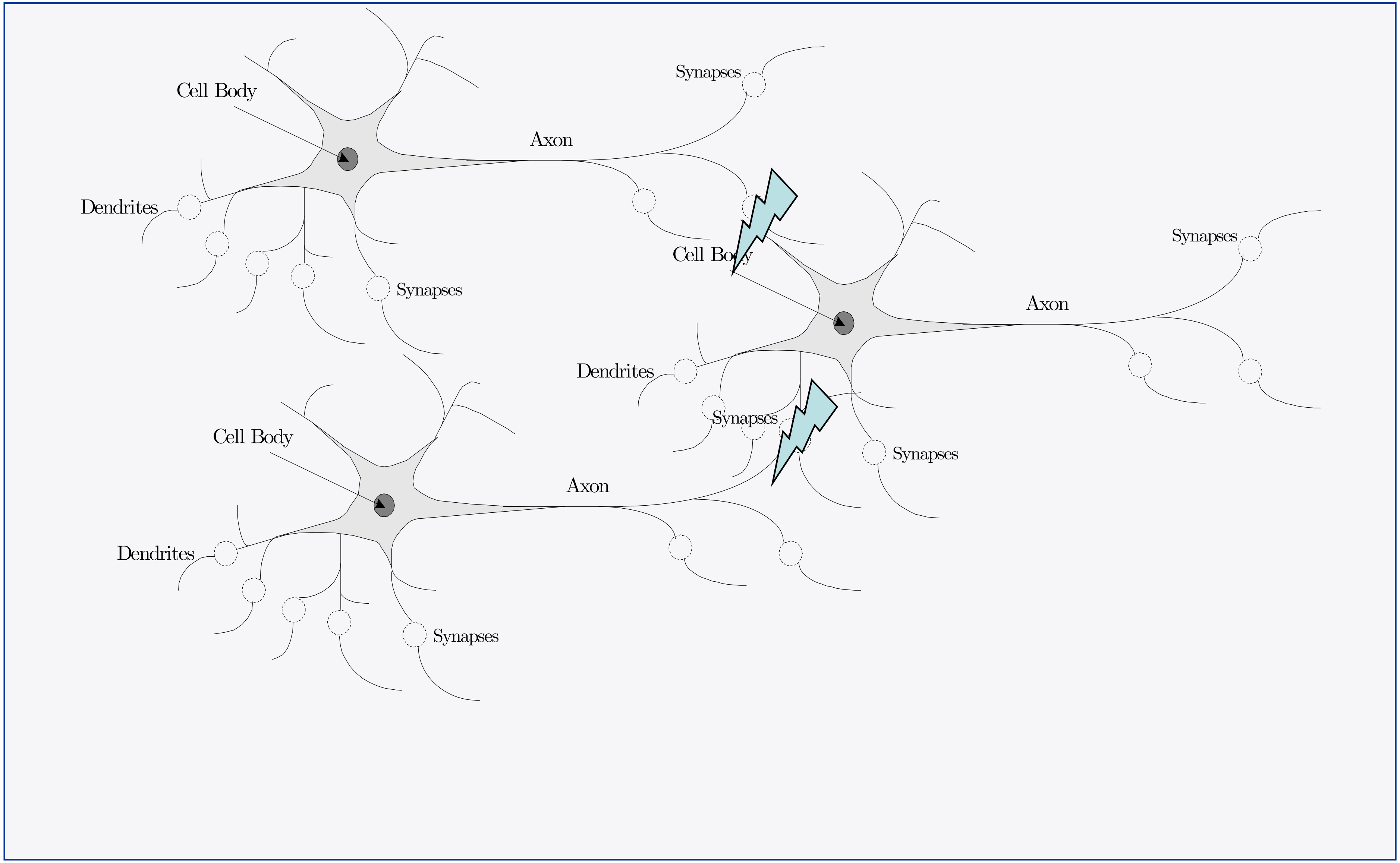
- parallel computation

Response Time:

– 10^{-3} s , the biological

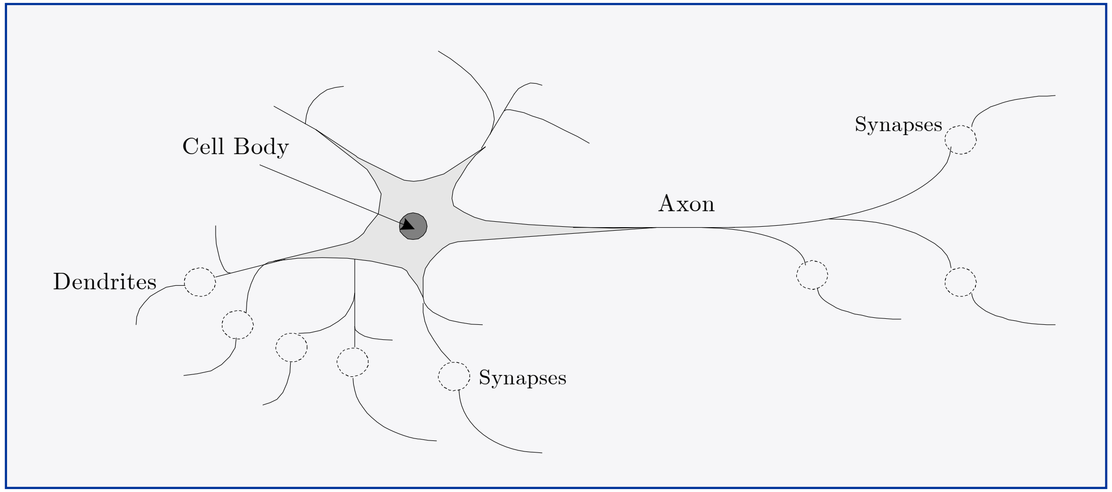
- 10^{-9} s , the electrical circuits

Biological Network

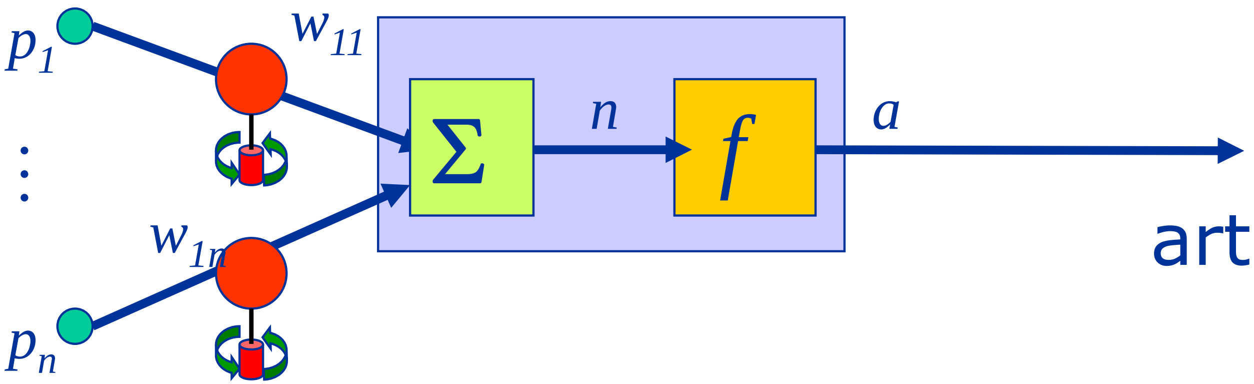


Artificial Neuron: mathematical model

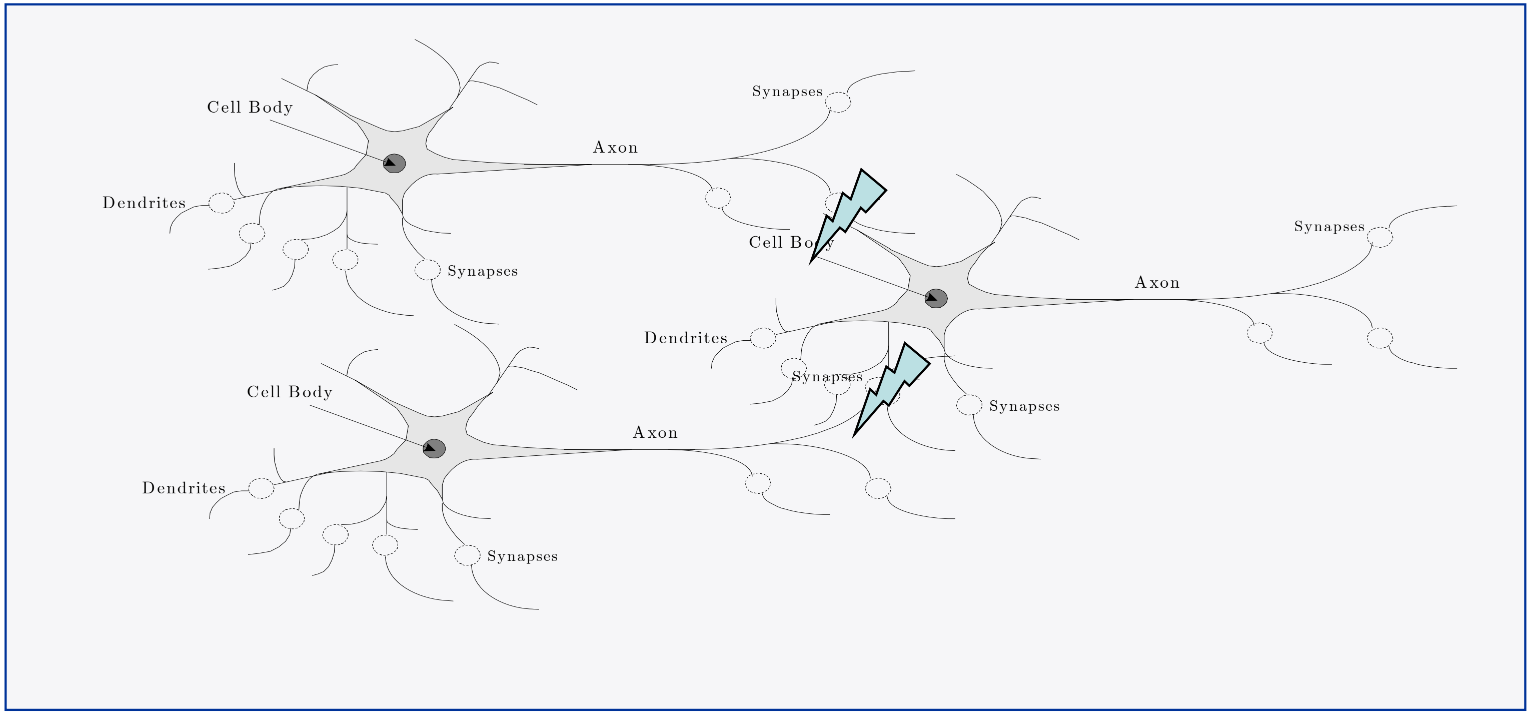
dendrites sinapses body axon

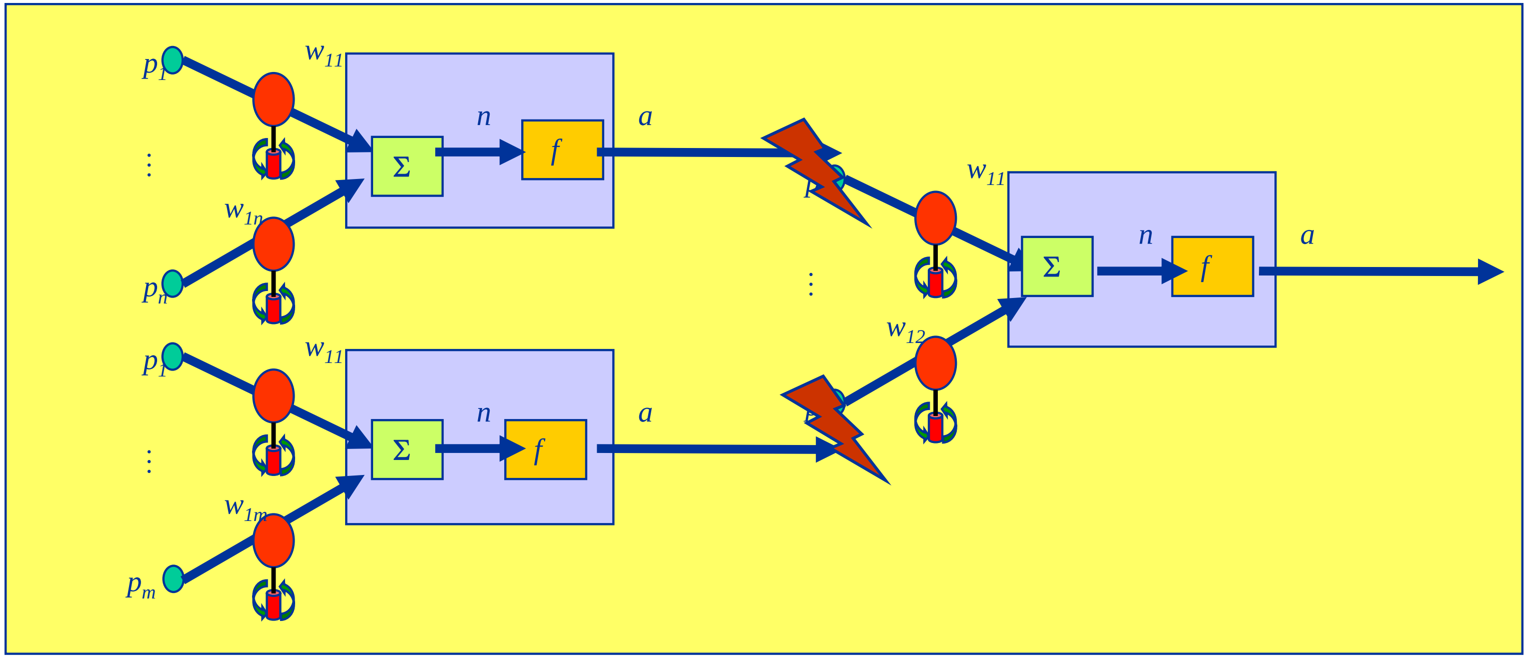


inputs weights Sum+activation *f* output

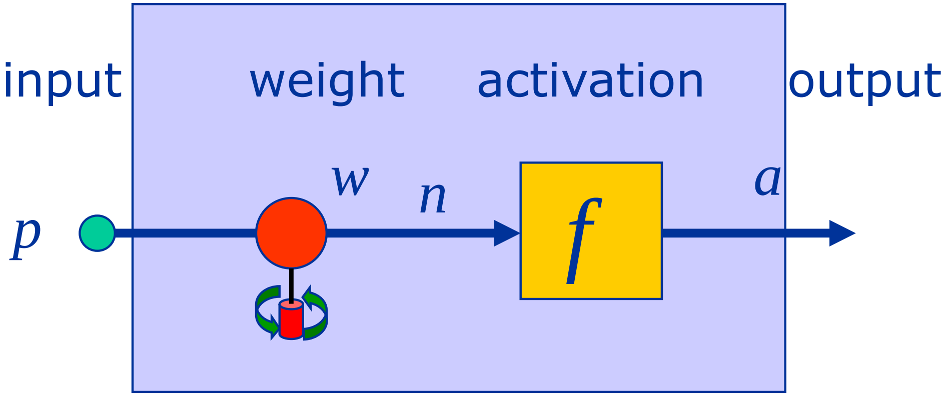


artificial


Cell Body
Synapses
Axon
Dendrites
Synapses
Cell Body
Synapses
Axon
Dendrites
Synapses
Synapses
Cell Body
Axon
Dendrites
Synapses


p_1
w_{11}
n
f
a
Σ
w_{1n}
p_n
w_{11}
n
a
w_{12}
p_1
w_{11}
n
f
a
Σ
w_{1m}
p_m

4.1. Neuron with a single input



$$n = wp$$

$$a = f(n) = f(wp)$$

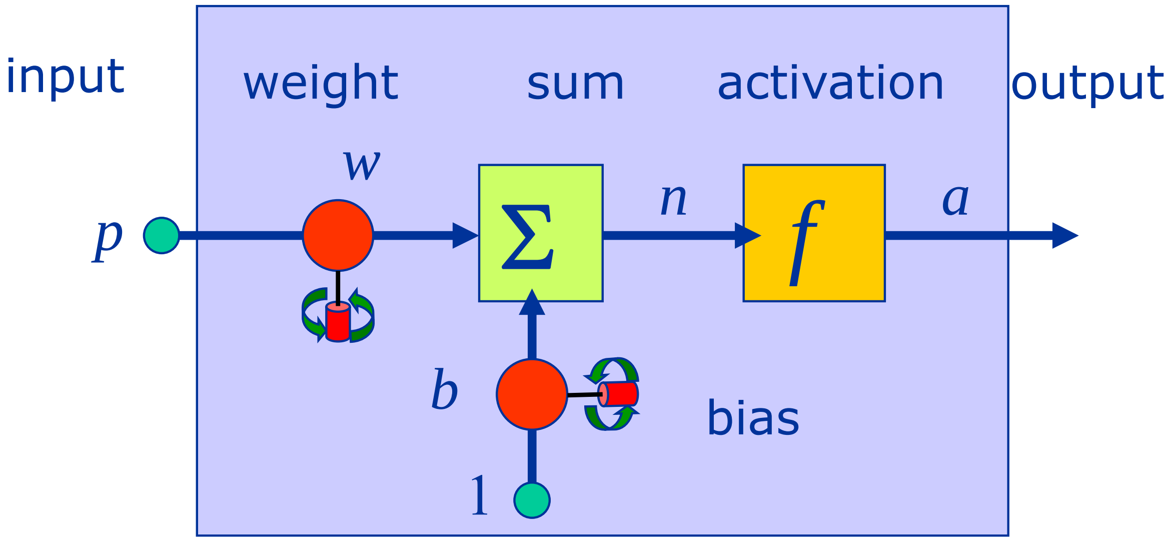
$$p = 0 \quad \Leftrightarrow \quad a = 0$$

$$\text{if } f(0) = 0$$

f : activation function (or transfer function)

w : input weight

Neuron with a single input and a bias



$$n = w \times p + b \times 1 = [w \; b] \times \begin{bmatrix} p \\ 1 \end{bmatrix} = w' p'$$

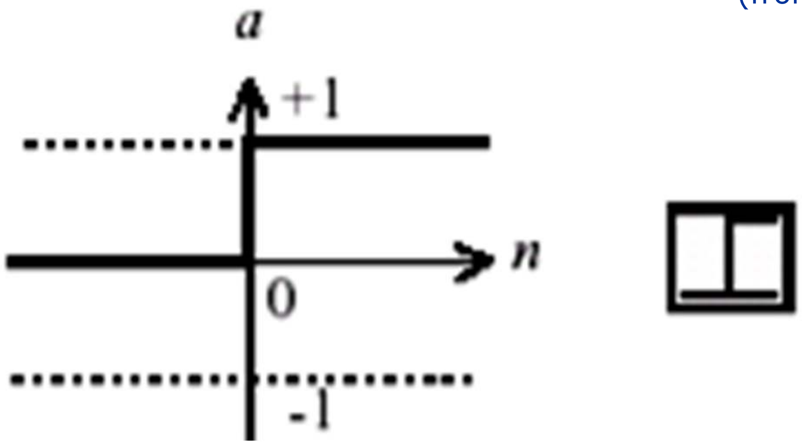
$$a = f(n) = f(wp + b) = f(w'p')$$

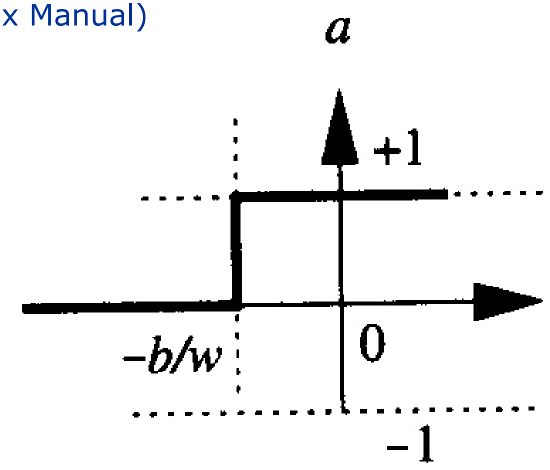
The bias allows the output to be nonzero even if the inputs are zero.

4.2. Activation functions

binary

(from DL Toolbox Manual)





$$a = \mathrm{hardlim}(n)$$

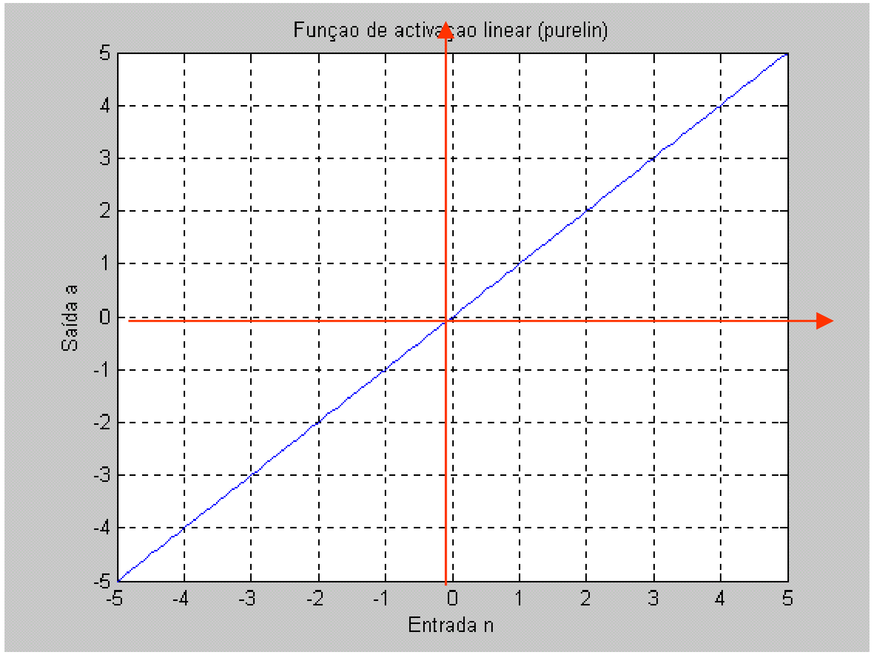
$$\begin{aligned} a &= 0 \qquad n < 0 \\ a &= 1 \qquad n \geq 0 \end{aligned}$$

$$a = \mathrm{hardlim}(wp + b)$$

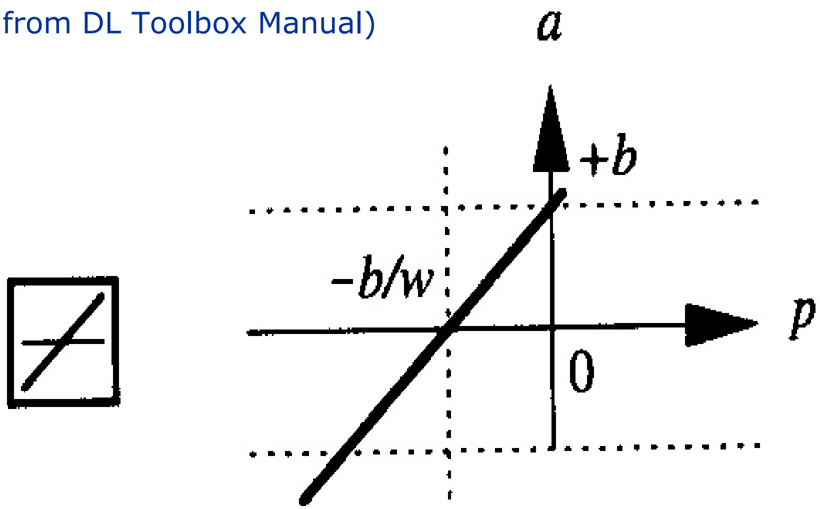
The *bias b* makes horizontal displacements of the step function

Activation functions

continuous linear



(from DL Toolbox Manual)



$$a = \text{purelin}(n)$$

$$a = n$$

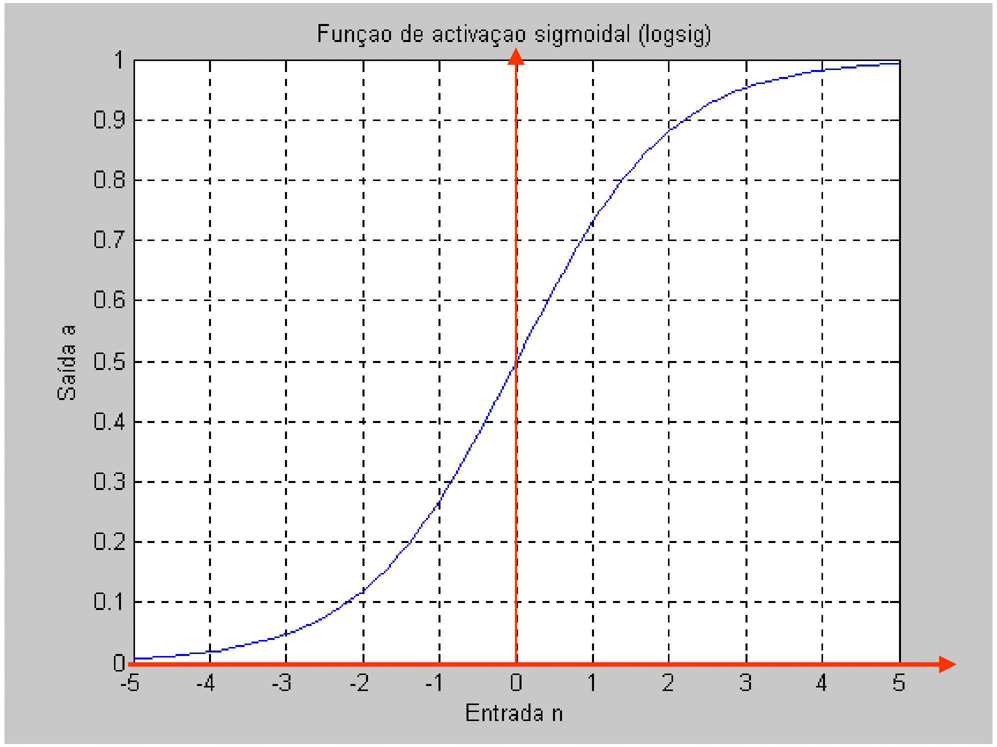
$$a = \text{purelin}(wp + b)$$

The *bias b* makes horizontal displacements of the linear function

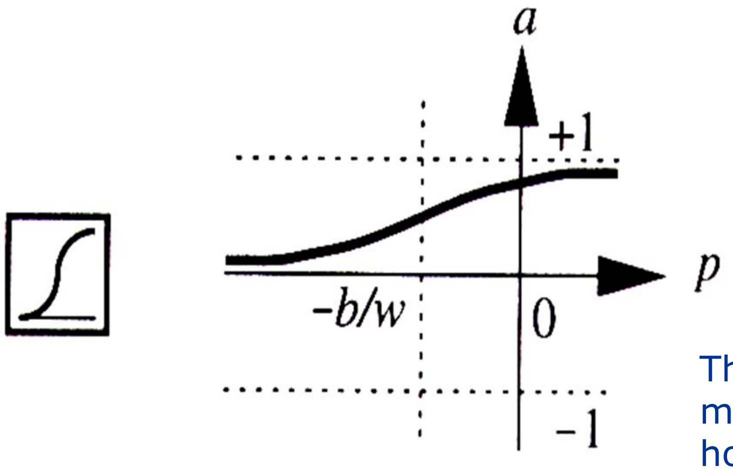
Activation functions

Continuous nonlinear monotones (differentiable)

Sigmoid unipolar



(from DL Toolbox Manual)



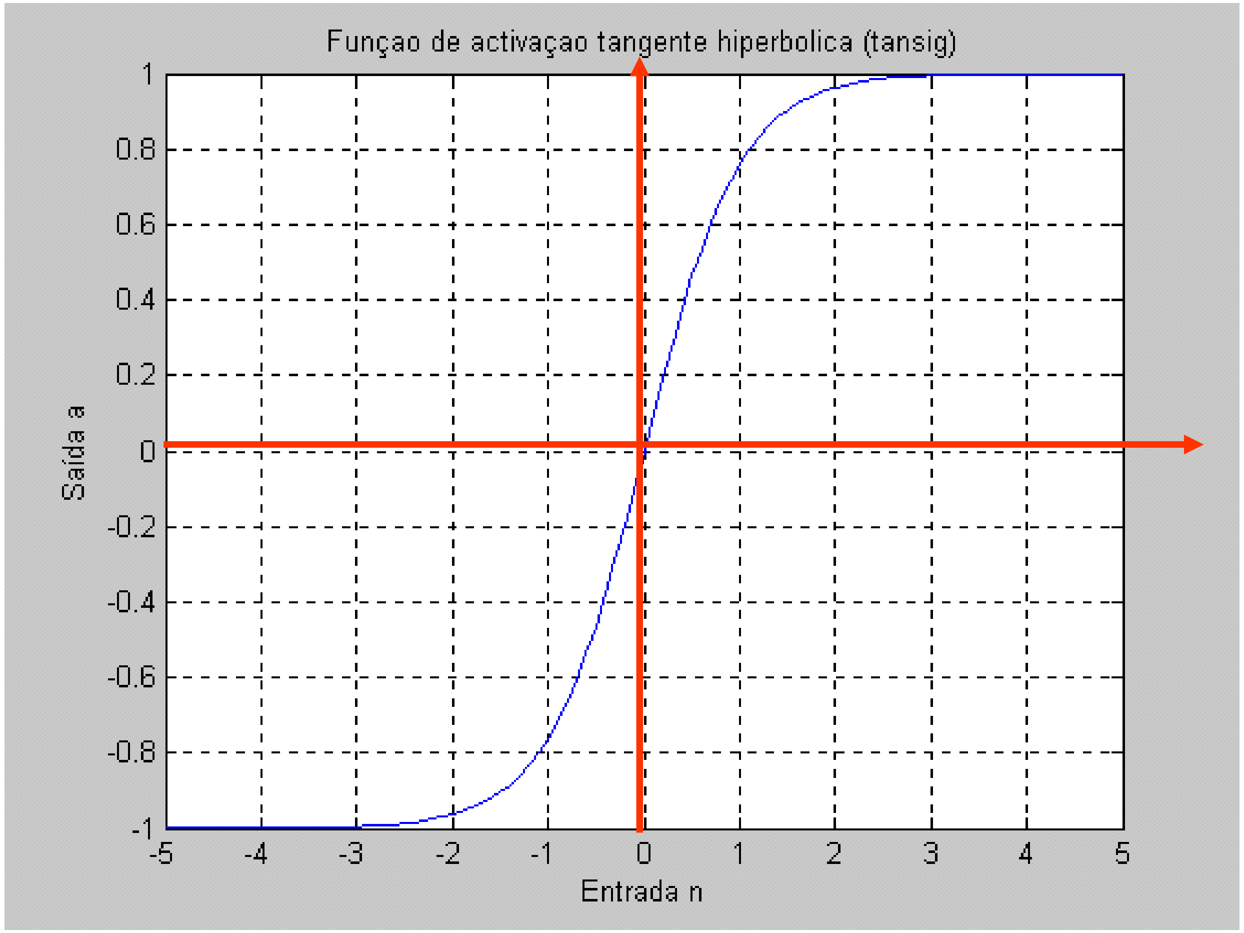
$$a = logsig(wp+b)$$

The *bias b* makes horizontal displacements of the sigmoid function

$$a = \text{logsig}(n) = \frac{1}{1+e^{-n}}$$

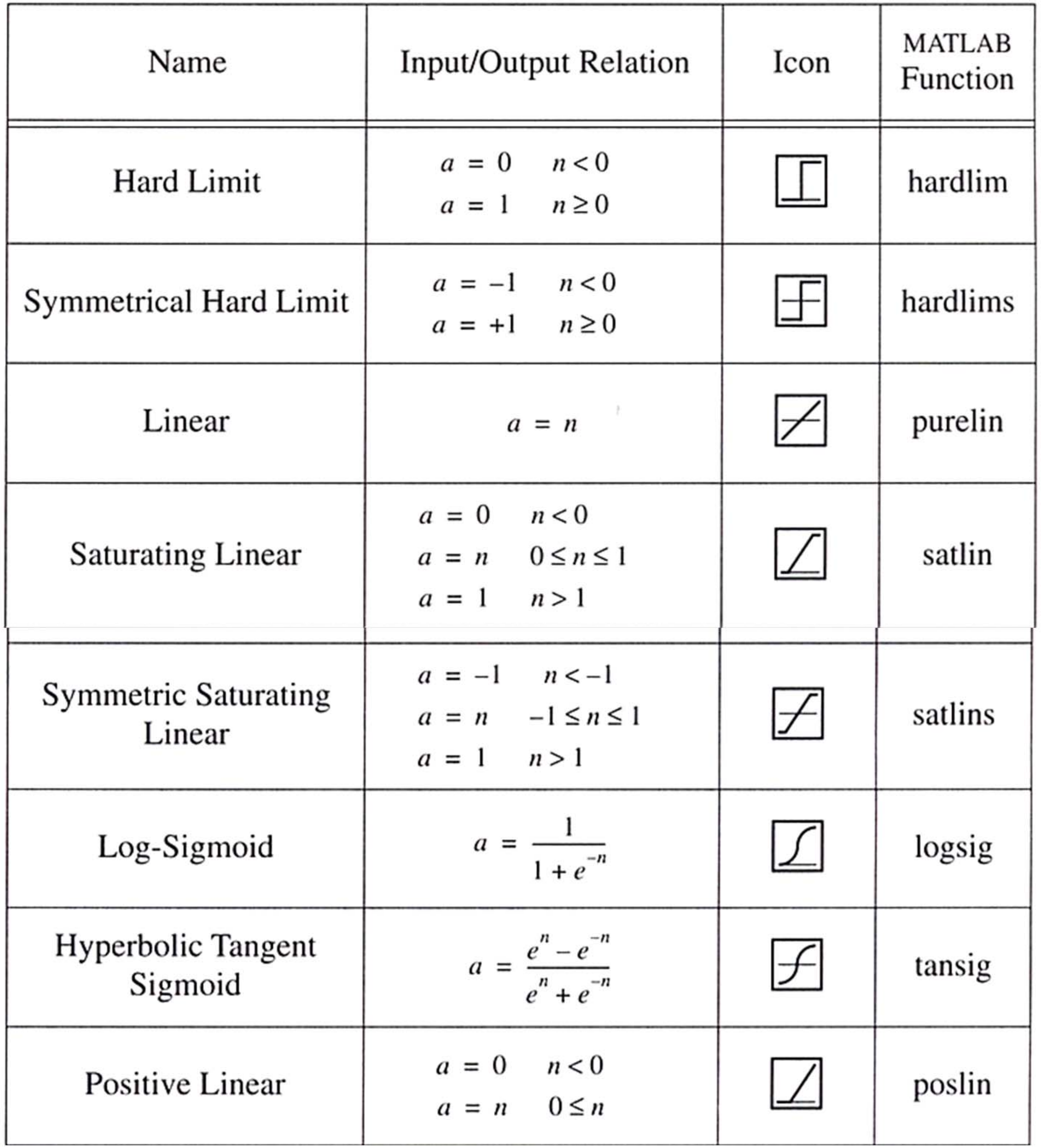
$$a = \frac{1}{1+e^{-(wp+b)}}$$

Sigmoid bipolar (hiperbolic tangent)



$$a = \mathrm{tansig}(n) = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}, \quad \text{in Matlab}$$

Activation functions

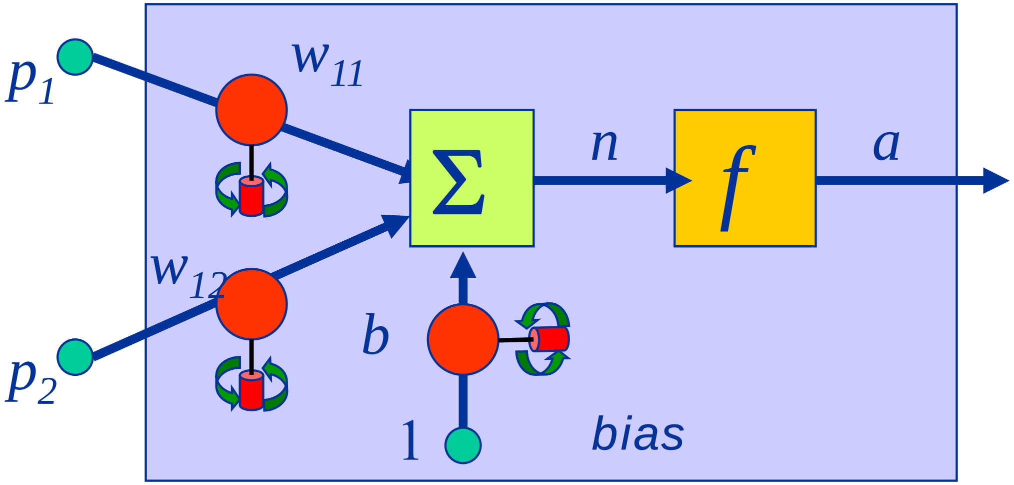


Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1 + e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin

(from Hagan&coll.)

4.3. Neuron with several inputs

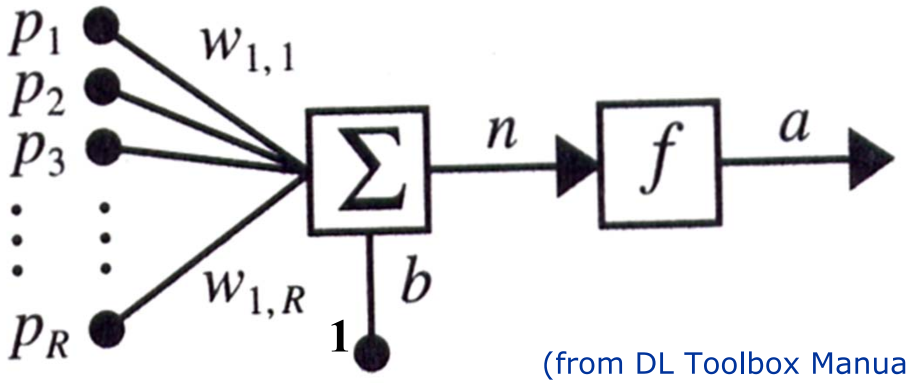
2 inputs 2 weights sum activation output



$$n = w_{11}p_1 + w_{12}p_2 + b = \begin{bmatrix} w_{11} & w_{12} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + b = \mathrm{W}p + b$$

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

Neuron with R inputs

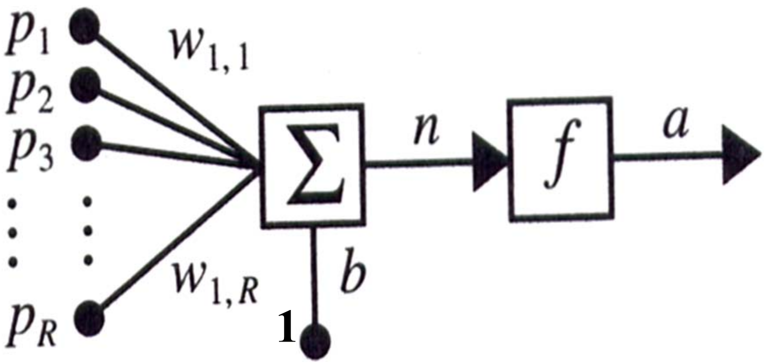


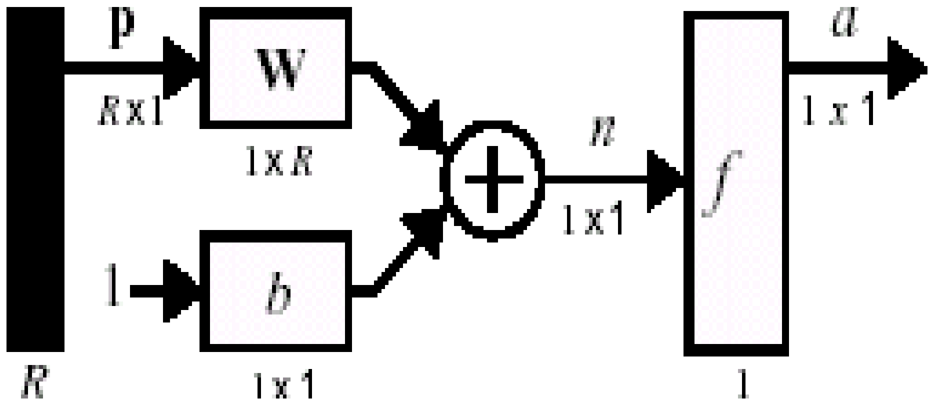
(from DL Toolbox Manual)

$$n = w_{11}\,p_1 + w_{12}\,p_2 + \ldots + w_{1R}\,p_R + b \quad \Leftrightarrow \quad n = \mathrm{W}\;p + b$$

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

Neuron with R inputs





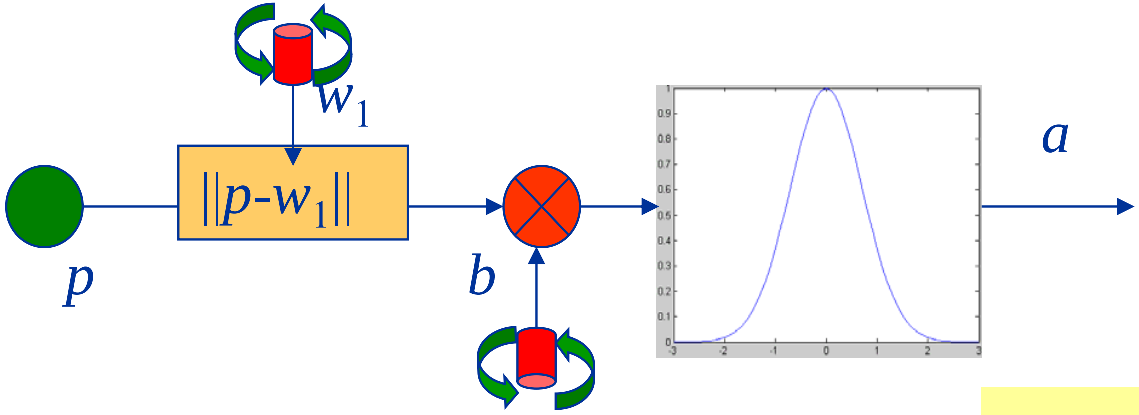
(De DL Toolbox Manual)

compact notation (vectors and matrices)

4.4. Neuron RBF (Radial Basis Function)

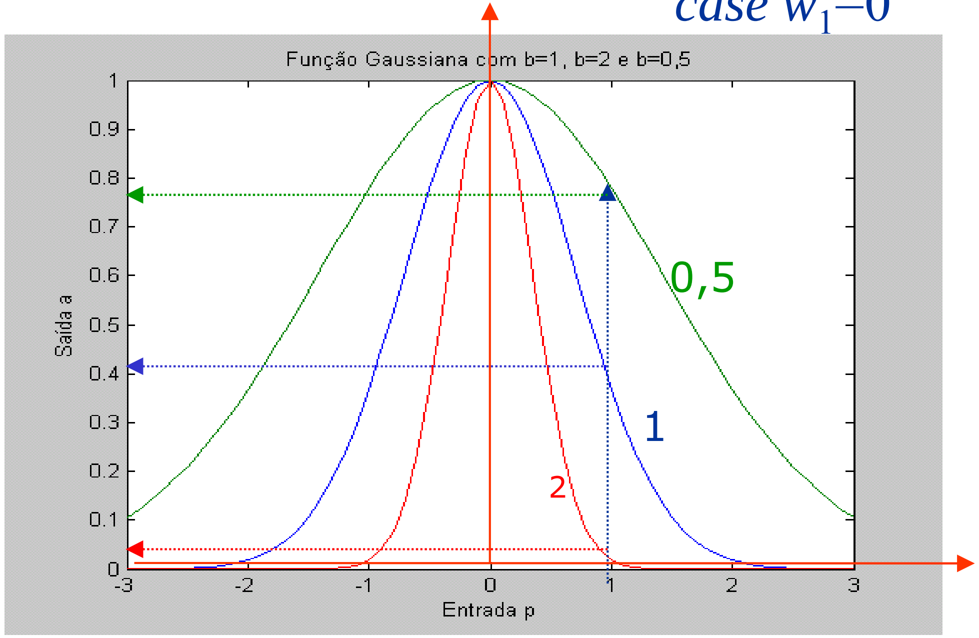
Activation function continuous non linear, non monotone

One RBF neuron, gaussian, with one input and a scale factor

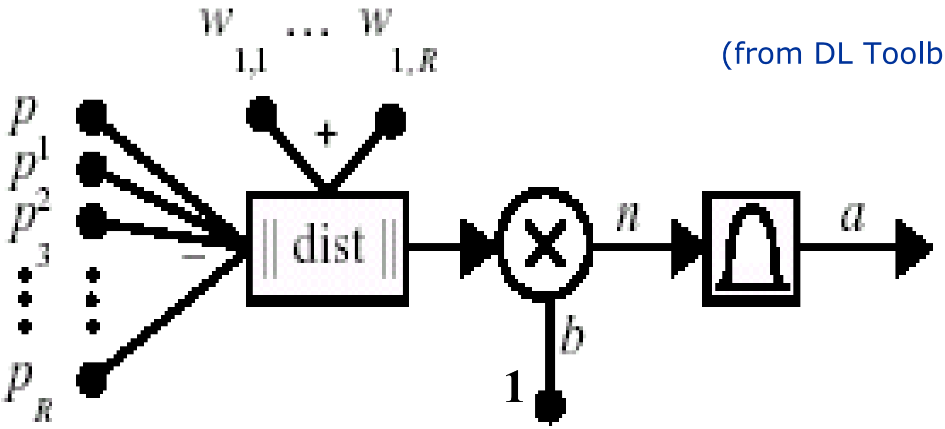


$$a = e^{-(\|p-w_1\| \times b)^2}$$

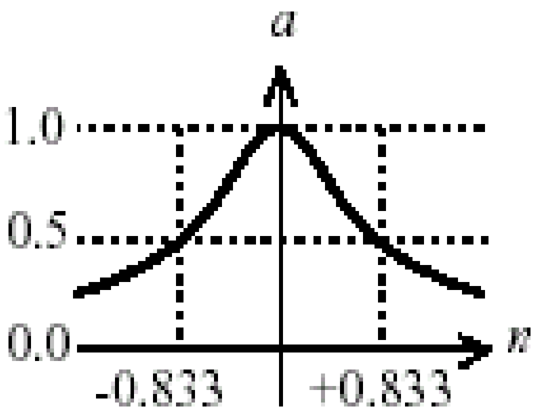
case $w_1=0$



One RBF neuron, gaussian, with R inputs and scale factor



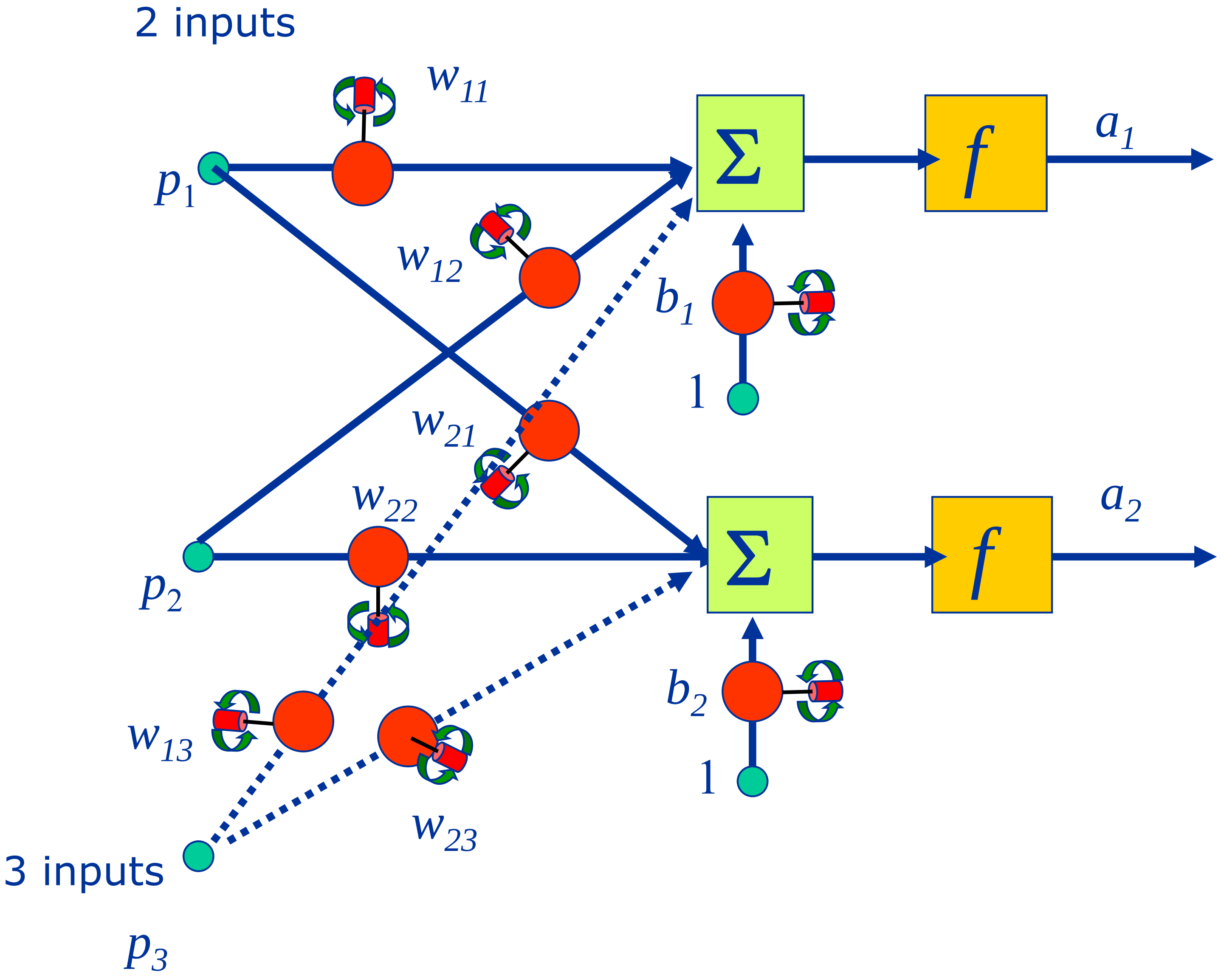
(from DL Toolbox Manual)



$$a = radbas(n) = radbas(\|w - p\| b)$$

$$a = radbas(n)$$
$$= e^{-n^2}$$

4.5. Layer of neurons

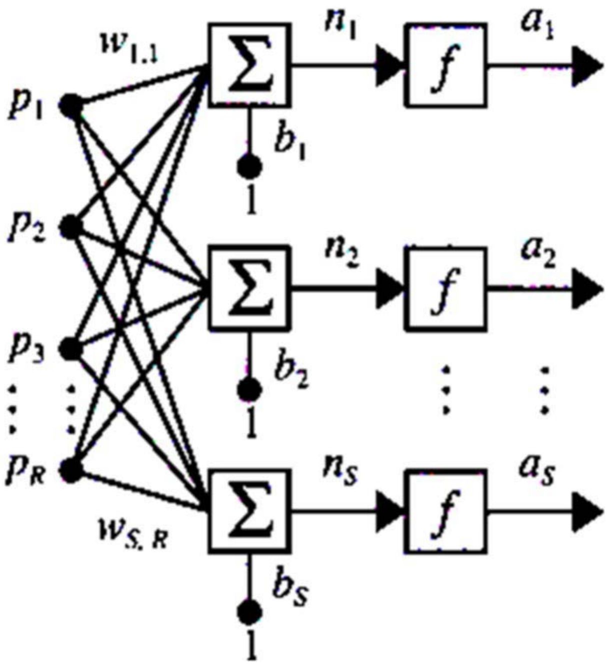


4.5. Layer of neurons

S neurons

R

inputs



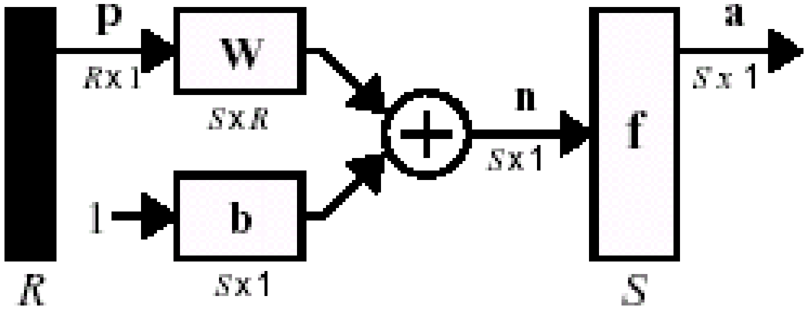
$$a = f(\mathrm{W}p + b)$$

$$n_i = w_{i1}p_1 + w_{i2}p_2 + \ldots + w_{iR}p_R + b_i$$

$$a_i = f_i(n_i) \qquad\qquad i = 1, 2, \ldots, S$$

Compact (matrices)

(from DL Toolbox Manual)



neuron

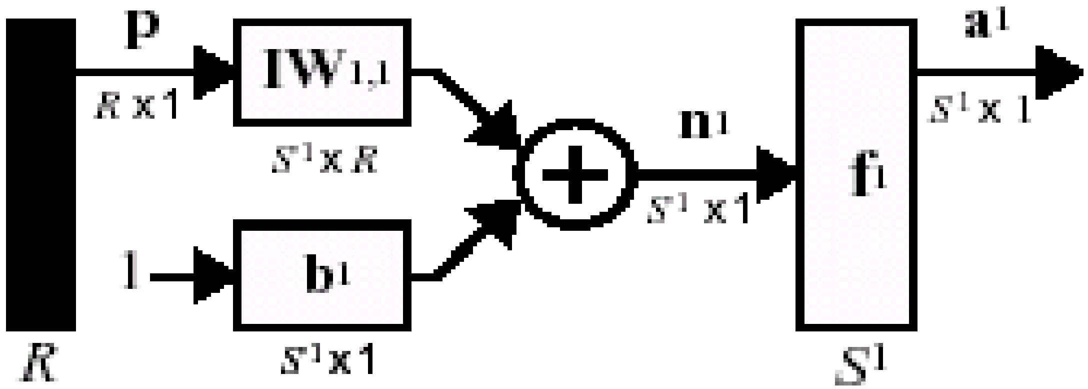
$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1R} \\ w_{21} & w_{22} & \dots & w_{2R} \\ \dots & \dots & \dots & \dots \\ w_{S1} & w_{S2} & \dots & w_{SR} \end{bmatrix}$$

input

$$a = f(Wp+b)$$

$$p = \begin{bmatrix} p_1 \\ p_2 \\ \dots \\ p_R \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_S \end{bmatrix} \quad a = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_S \end{bmatrix}$$

Compact notation (matrices) with layer index



(De DL Toolbox Manual)

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

$IW^{1,1} \triangleq Input\ \ Weight\ \ Matrix$

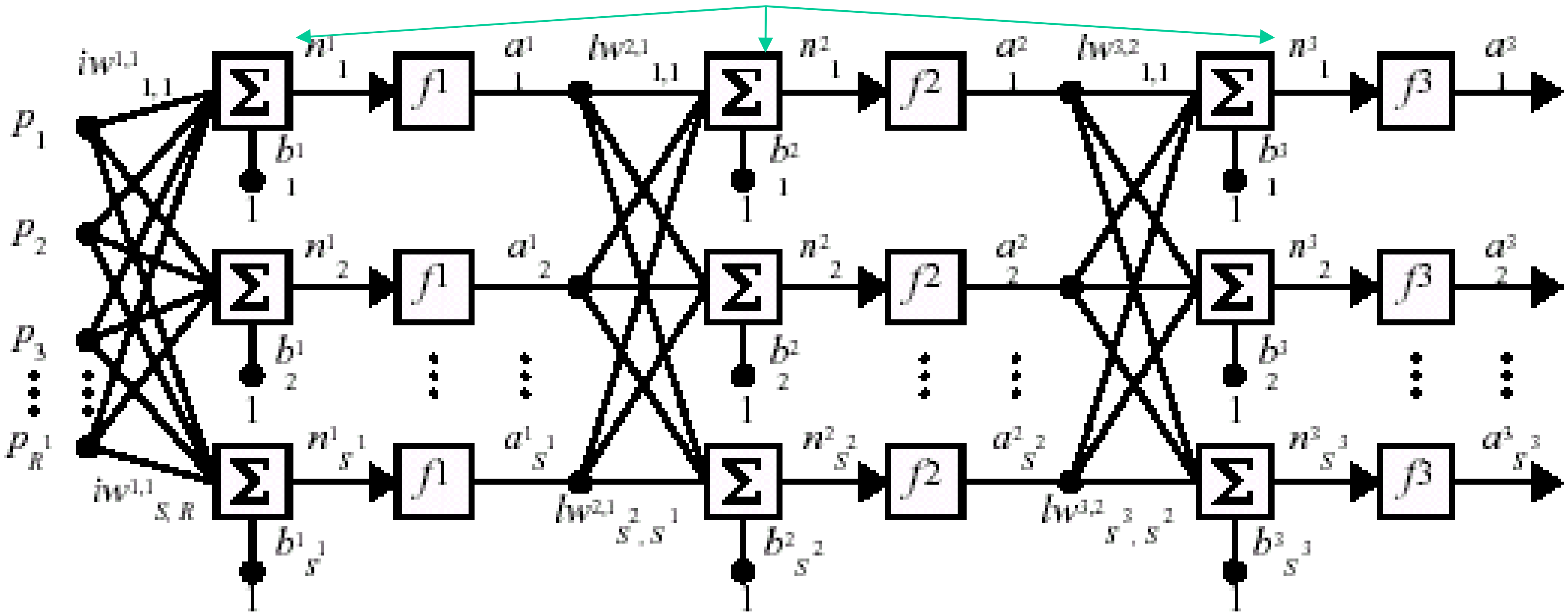
from the origin 1 (second index) to the destination 1 (first index)

$LW^{i,j} \triangleq Layer\ \ Weight\ \ Matrix$

from origin j to the destination i

4.6. Multilayer network

3 layers



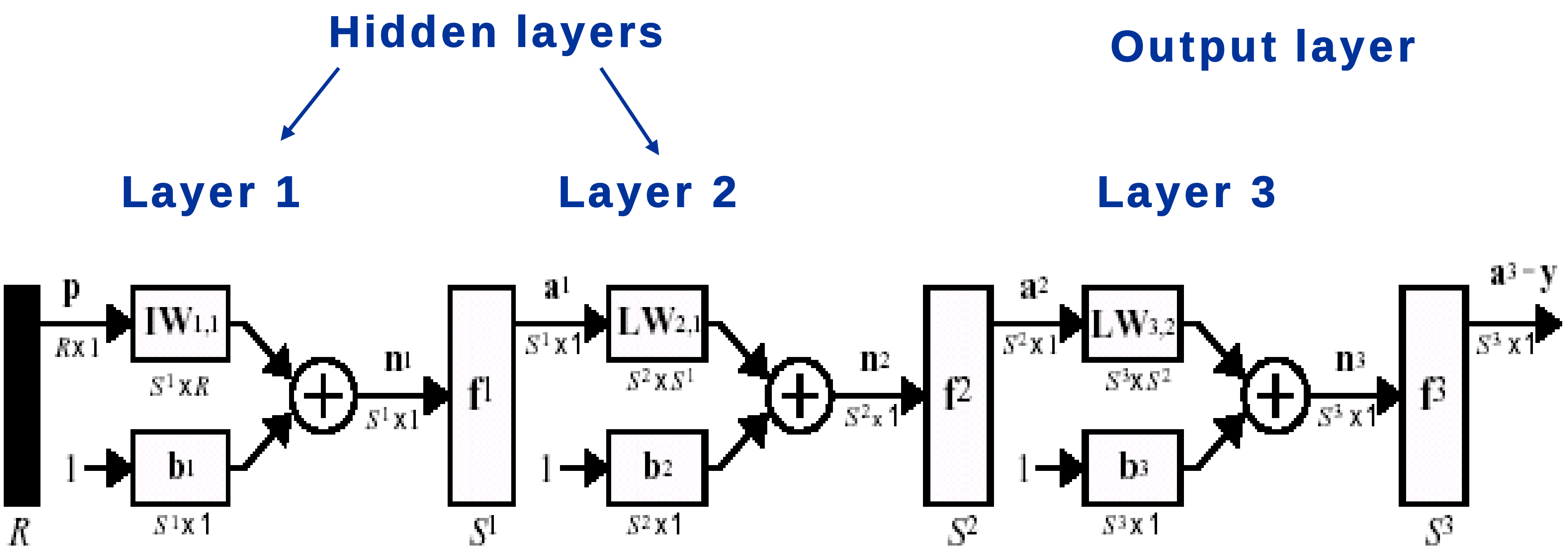
from DL Toolbox UG)

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

$$a^3 = f^3(LW^{3,2}f^2(LW^{2,1}f^1(IW^{1,1}p+b_1)+b_2)+b_3)$$

f^3 can model any nonlinear relation between input *p* and output *a*

Compact notation



Hidden layers

Output layer

Layer 1

Layer 2

Layer 3

(from DL Toolbox Manual)

hidden layer: its output is not seen from the outside

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

$$a^3 = y = f^3(LW^{3,2}f^2(LW^{2,1}f^1(IW^{1,1}p+b_1)+b_2)+b_3)$$

f^3 can model any nonlinear relation between input *p* and output *y*

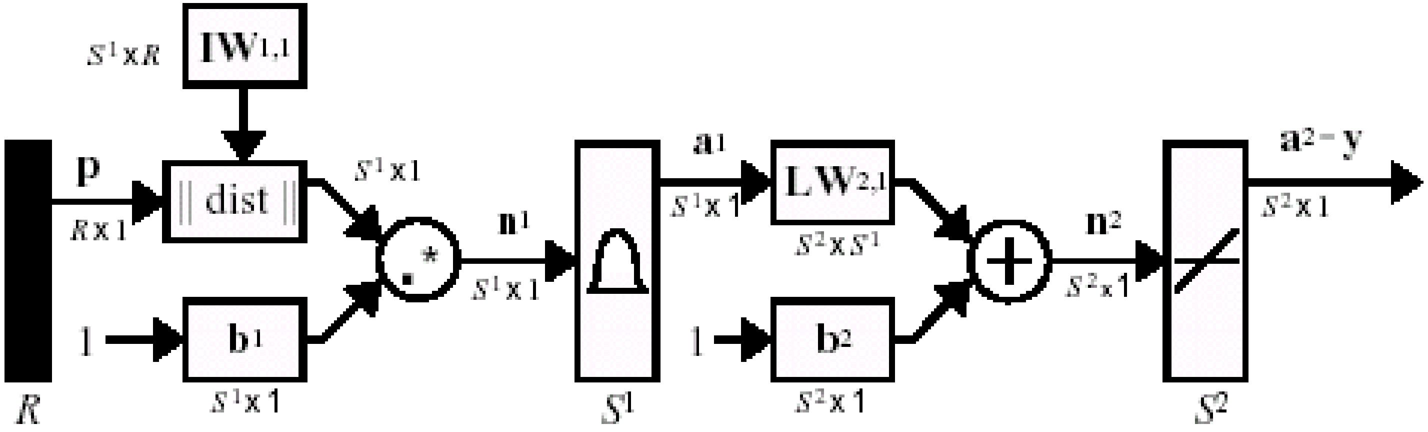
4.7. RBFNN- Radial Basis Function Neural Network

Layer of S_1 RBF neurons

Linear layer with S_2 neurons

R

entradas



(De DL Toolbox Manual)

$$a_i^1 = radbas(\|_i\mathrm{IW}^{1,1} - p\| b_i^1) \qquad \mathrm{a}_2 = purelin(\mathrm{LW}^{2,1}\mathrm{a}^1 + \mathrm{b}^2))$$

$$a_i^1 = \mathrm{i}^{th} \text{ element of } \mathrm{a}_1$$

$$_i\mathrm{IW}^{1,1} \triangleq \text{vector composed by the } i^{th} \text{ row of } \mathrm{IW}^{1,1}$$

4.8. The binary perceptron: training and learning

Learning rule or training algorithm:

- a systematic procedure to modify the weights and the bias of a NN such that it will work as we need

The most common learning approaches:

supervised learning

reinforcement learning

unsupervised learning

- Supervised learning

A set of *Q* examples of correct behavior of the NN is given (inputs *p*, outputs *t*)

$$\left\{p^{1},t_{1}\right\},\ \left\{p^{2},t_{2}\right\},\ \ldots,\ \left\{p^{Q},t_{Q}\right\}$$

- Reinforcement learning

The NN receives only a classification that favors good performances.

- Unsupervised learning

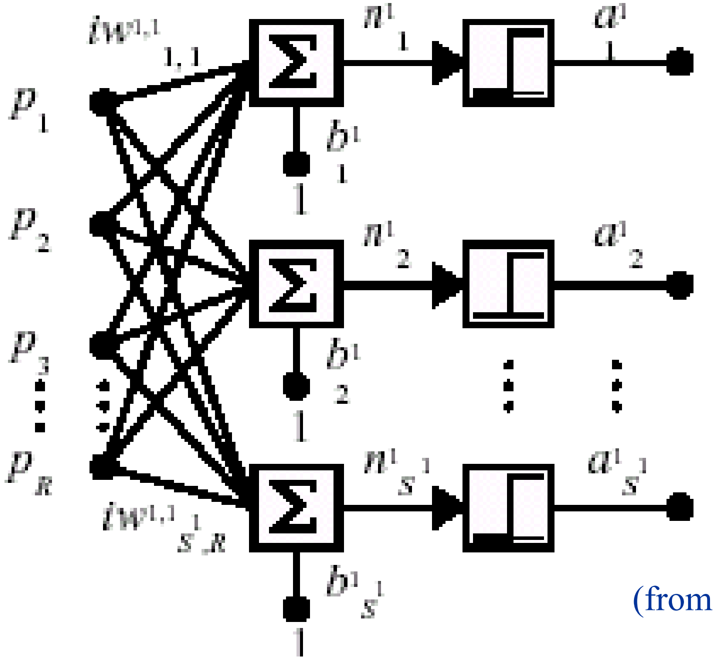
The NN has only inputs, not outputs, and learns to categorize them (dividing the inputs in classes, as in clustering)

Binary Perceptron learning

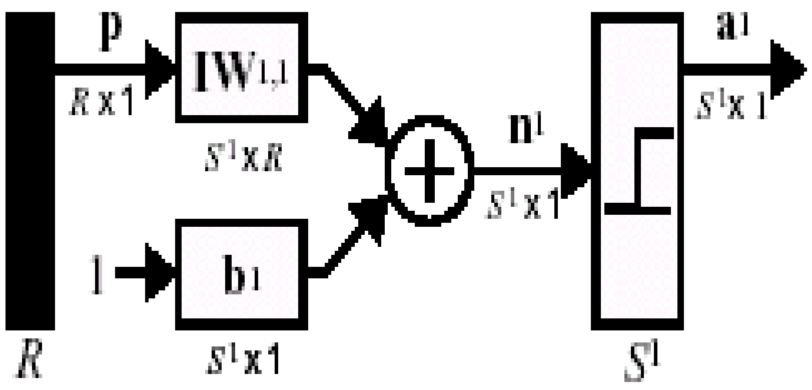
Perceptron is the first ANN for which mathematical developments have been made. Its training is still illustrative of many issues in any ANN training.

S neurons

R Inputs



S neurons

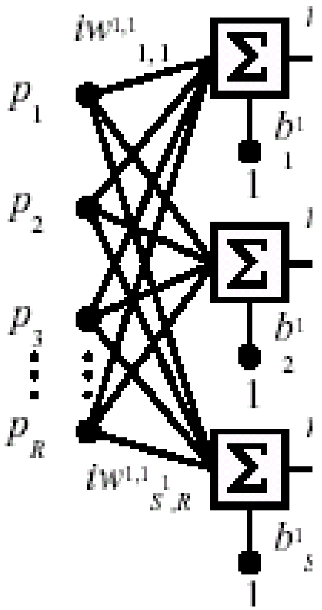


(from DL Toolbox U.G.)

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

For a single layer one may eliminate the layer index ^{1}

For a single layer one may eliminate the layer index ^{1}



$$W = IW^{1,1} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1R} \\ w_{21} & w_{22} & \cdots & w_{2R} \\ \cdots & \cdots & \cdots & \cdots \\ w_{S1} & w_{S2} & \cdots & w_{SR} \end{bmatrix}$$

$${}_i w = \begin{bmatrix} w_{1i} \\ w_{2i} \\ \cdots \\ w_{Si} \end{bmatrix}$$

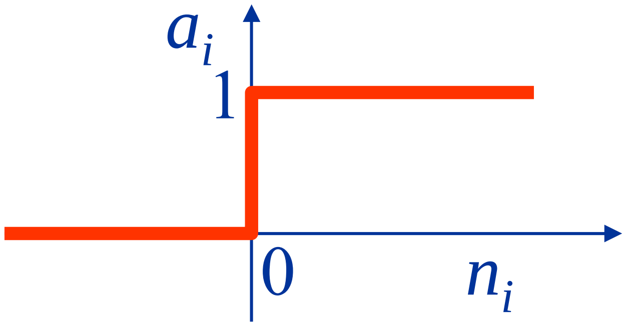
Column i of W

Line i of W

$${}_i w^T = \begin{bmatrix} w_{i1} & w_{i2} & \cdots & w_{iR} \end{bmatrix}$$

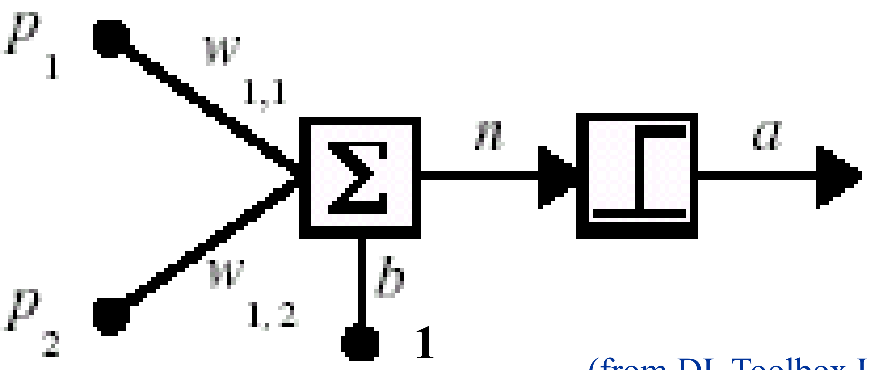
$$W = \begin{bmatrix} {}_1 w^T \\ {}_2 w^T \\ \cdots \\ {}_S w^T \end{bmatrix}$$

$$a_i = \text{hardlim}(n_i) = \text{hardlim}({}_i w^T p + b_i)$$



Each neuron divides the space in two regions

One neuron, two inputs



(from DL Toolbox U.G.)

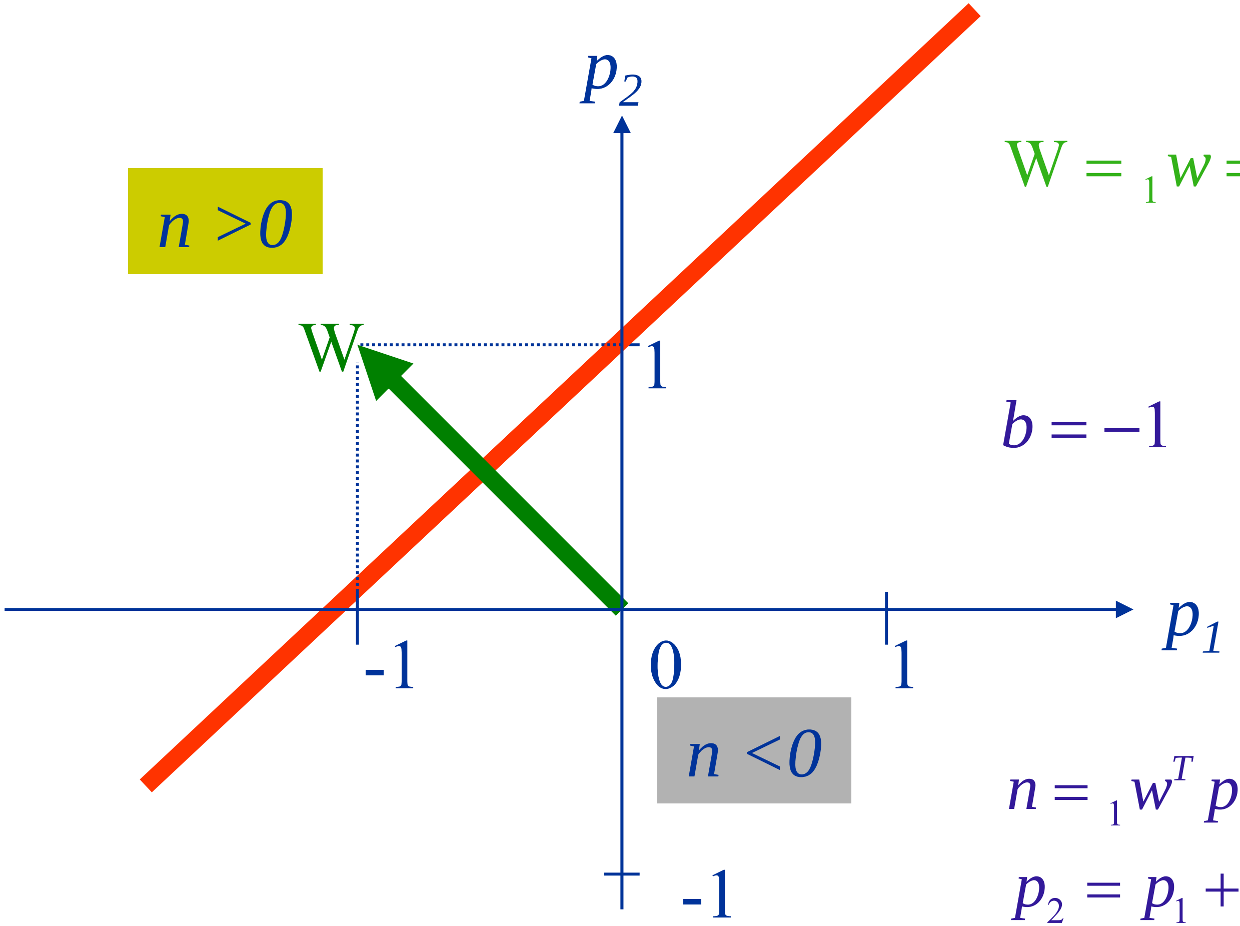
$$\begin{aligned} a &= \text{hardlim}(\mathrm{W}p + b) \\ &= \text{hardlim}({}_1w^T p + b) \\ &= \text{hardlim}(w_{11}p_1 + w_{12}p_2 + b) \end{aligned}$$

Decision boundary, $n = 0$: $n = {}_1w^T p + b = w_{11}p_1 + w_{12}p_2 + b = 0$

$$p_2 = -\frac{w_{11}}{w_{12}}p_1 - \frac{b}{w_{12}}$$

… One straight line $p_2 = mp_1 + d$

$$p_2 = -\frac{w_{11}}{w_{12}} p_1 - \frac{b}{w_{12}}$$



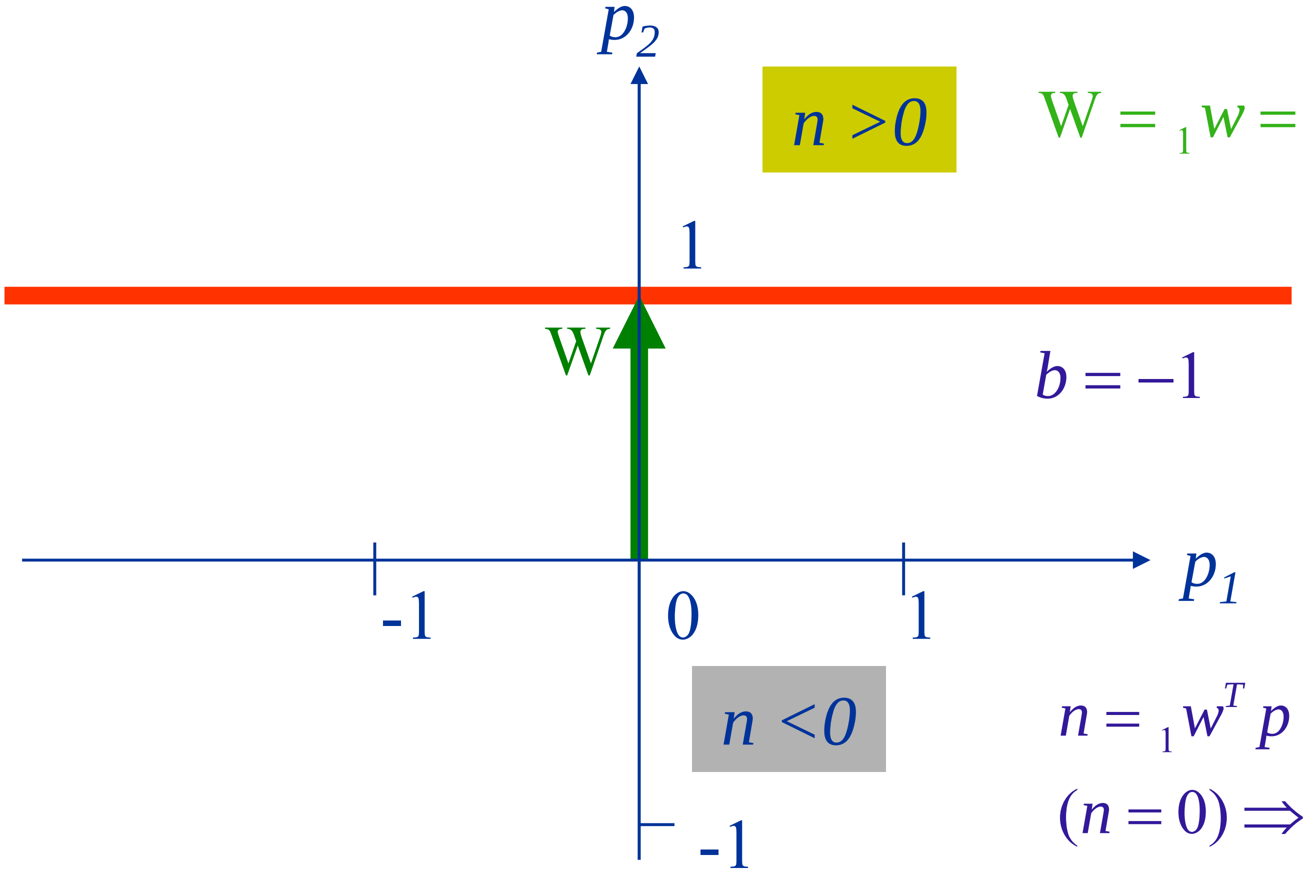
$$\mathrm{W} = {}_1w = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$b = -1$$

$$n = {}_1w^T p + b = -p_1 + p_2 - 1$$

$$p_2 = p_1 + 1 \quad (n = 0)$$

$$p_2 = -\frac{w_{11}}{w_{12}} p_1 - \frac{b}{w_{12}}$$



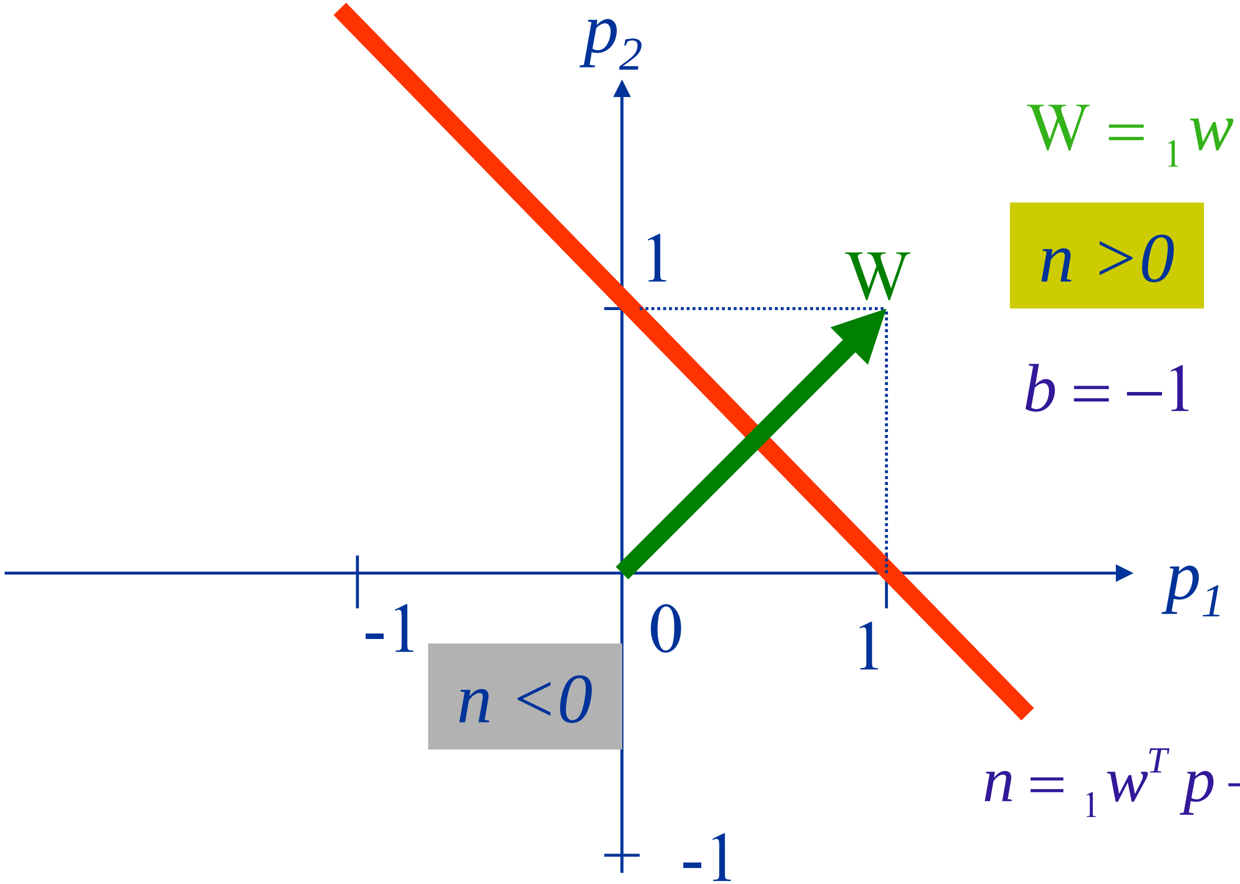
$$W = {}_1w = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$b = -1$$

$$n = {}_1w^T p + b = p_2 - 1$$

$$(n = 0) \Rightarrow p_2 = 1$$

$$p_2 = -\frac{w_{11}}{w_{12}} p_1 - \frac{b}{w_{12}}$$



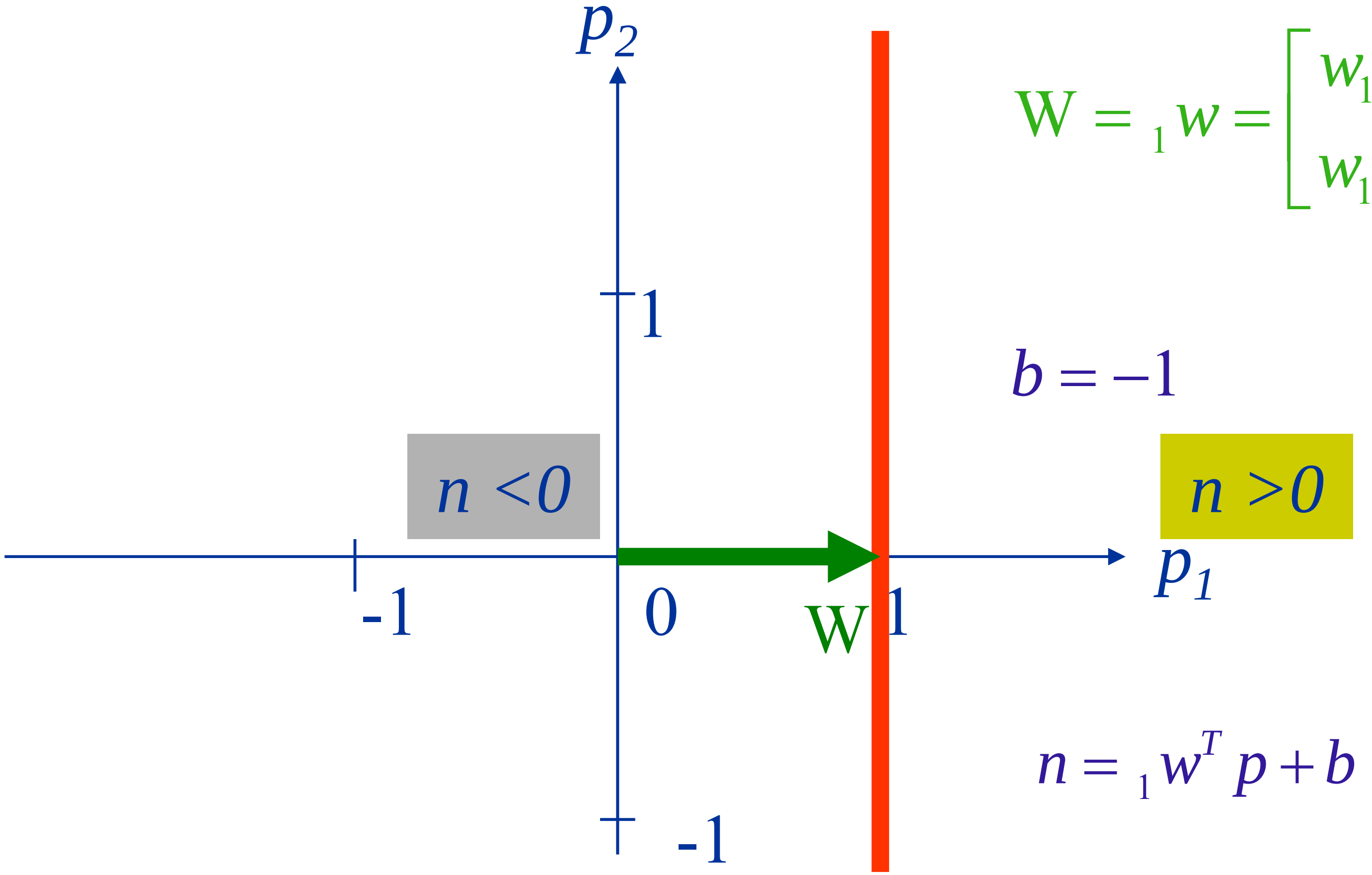
$$\mathrm{W} = {}_1w = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$n > 0$

$$b = -1$$

$$n = {}_1w^T p + b = p_1 + p_2 - 1$$

$$w_{11}p_1 + w_{12}p_2 + b = 0$$



$$W = {}_1w = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

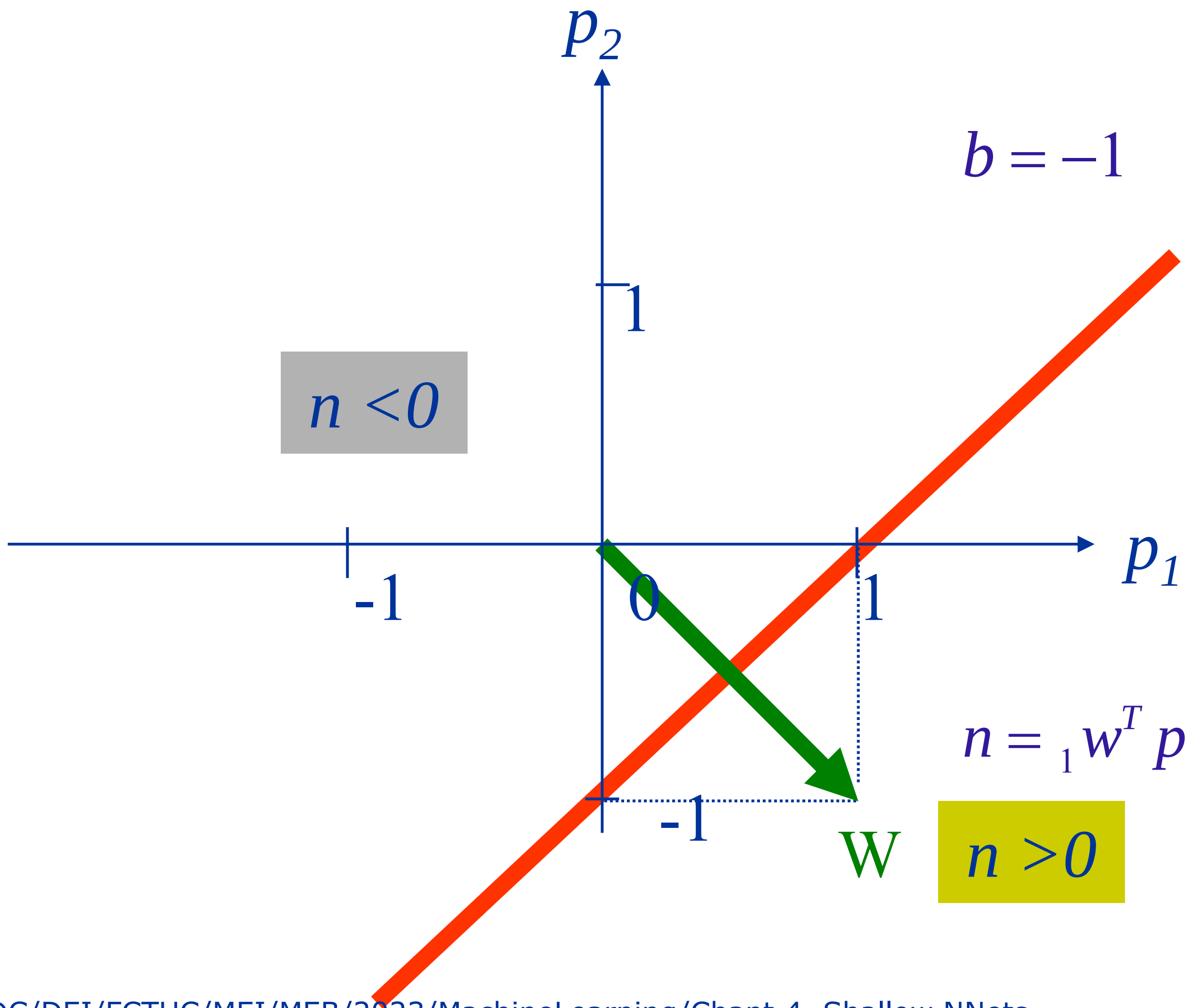
$$b = -1$$

$$n = {}_1w^T p + b = p_1 - 1$$

$$p_2 = -\frac{w_{11}}{w_{12}} p_1 - \frac{b}{w_{12}}$$

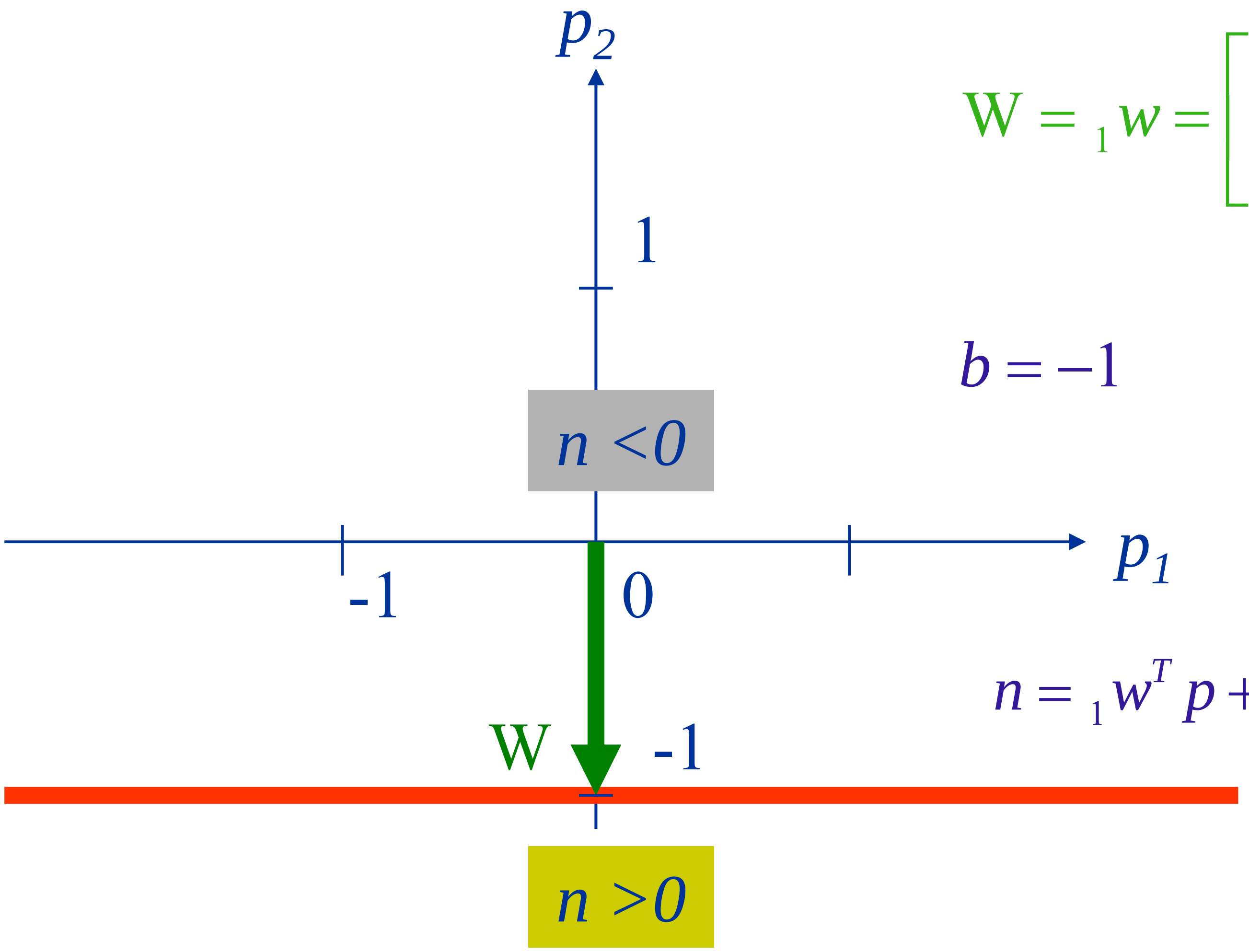
$$W = {}_1w = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$b = -1$$



$$n = {}_1w^T p + b = p_1 - p_2 - 1$$

$$p_2 = -\frac{w_{11}}{w_{12}} p_1 - \frac{b}{w_{12}}$$

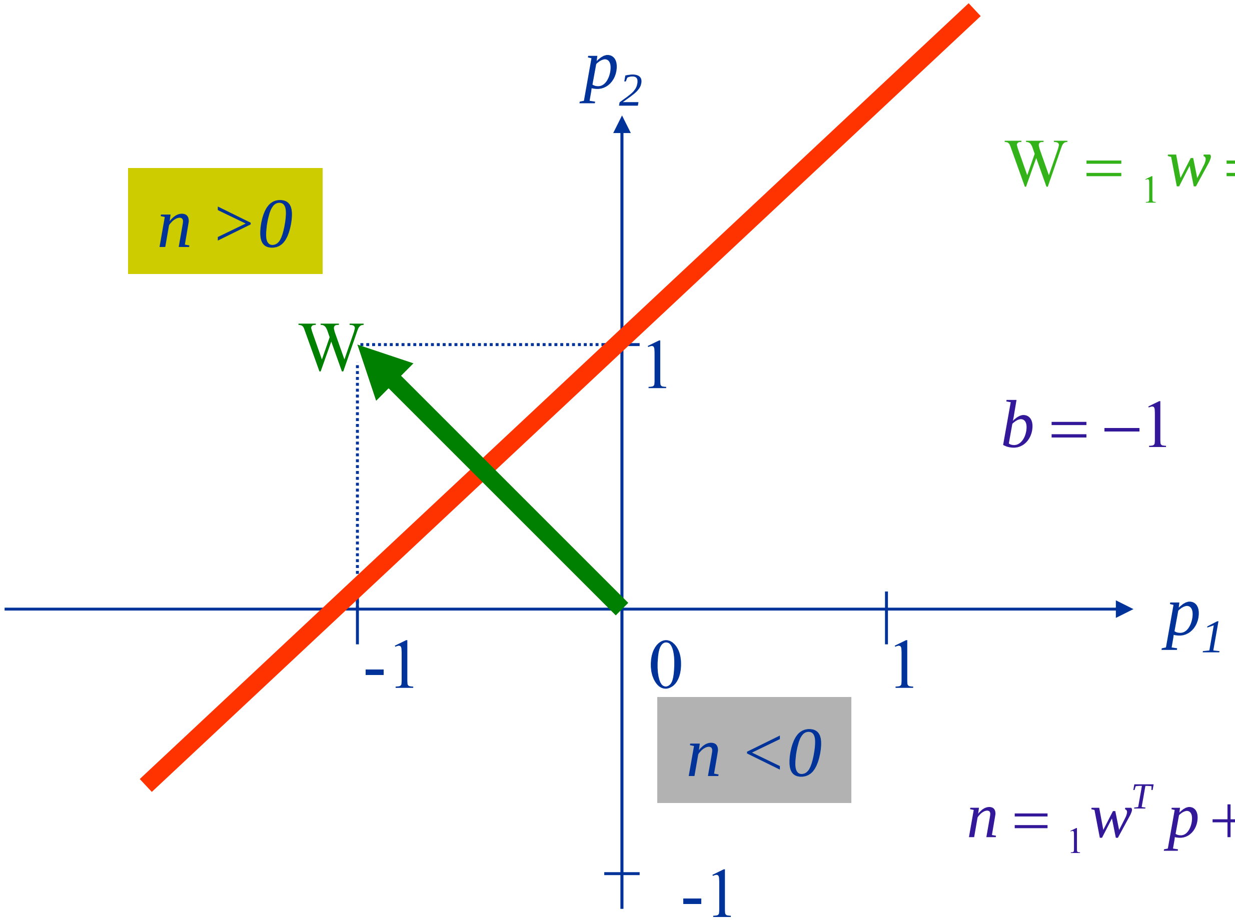


$$\mathrm{W} = {}_1w = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$b = -1$$

$$n = {}_1w^T p + b = -p_2 - 1$$

$$p_2 = -\frac{w_{11}}{w_{12}} p_1 - \frac{b}{w_{12}}$$



$$W = {}_1w = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$b = -1$$

$$n = {}_1w^T p + b = -p_1 + p_2 - 1$$

In any problem:

- Draw a boundary straight line

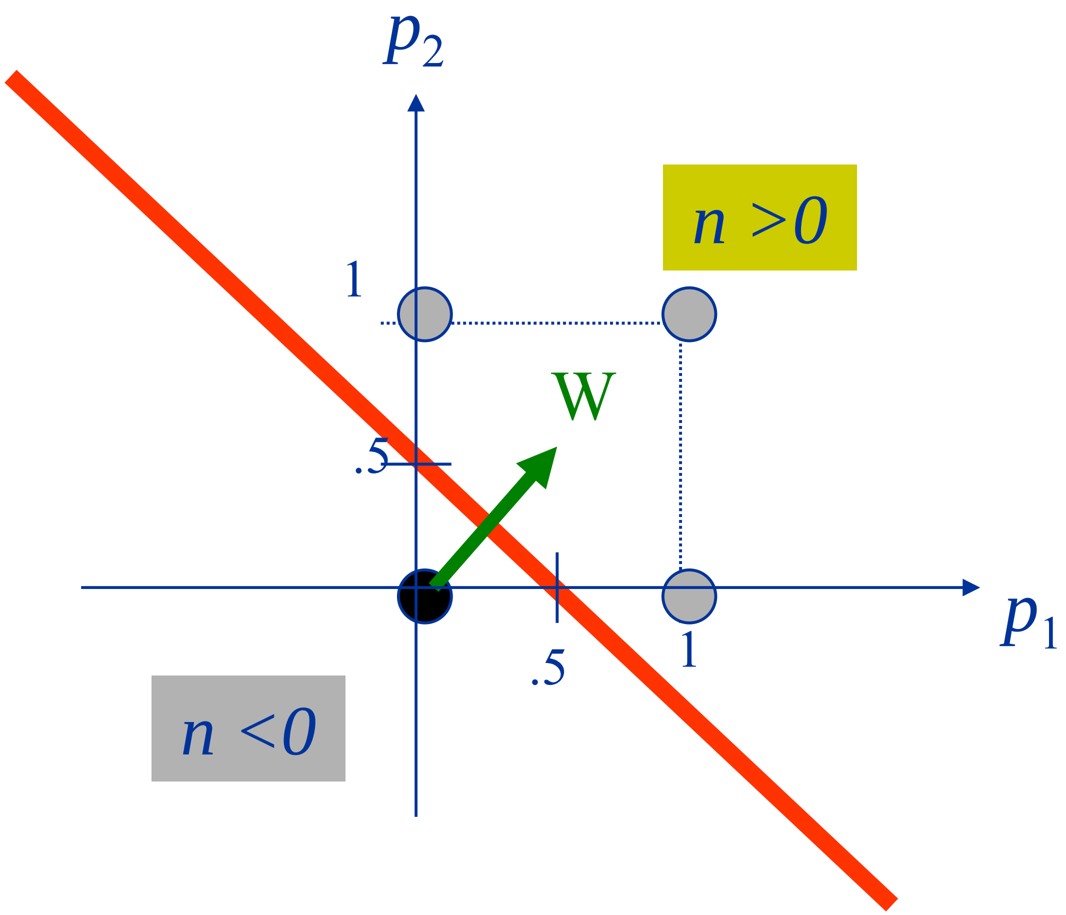
- Select the weights vector W perpendicular to that boundary, of any magnitude (what matters is its direction and sense)

- Calculate now the needed b making the calculations for a point of the boundary.

- The vector W points to the region $n>0$

Example 4.1: logical **OR**

p_1	p_2	p_1 **OR** p_2
0	0	0
0	1	1
1	0	1
1	1	1



○ True, 1

● False, 0

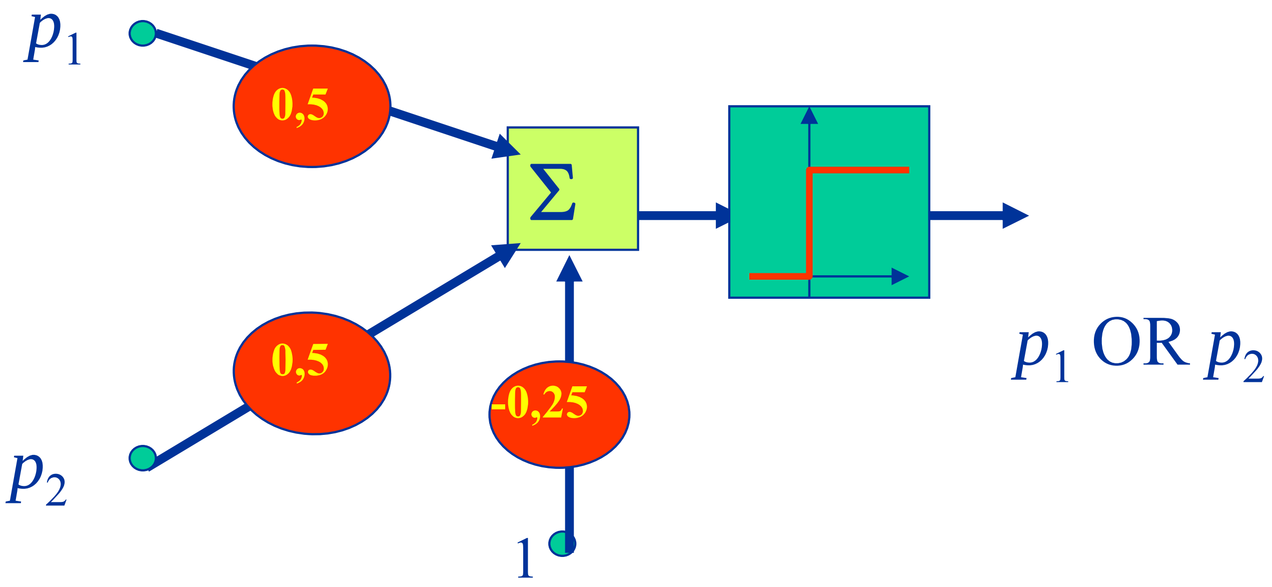
$$W = \begin{bmatrix} 0,5 \\ 0,5 \end{bmatrix}$$

$$point[0, 0.5]$$

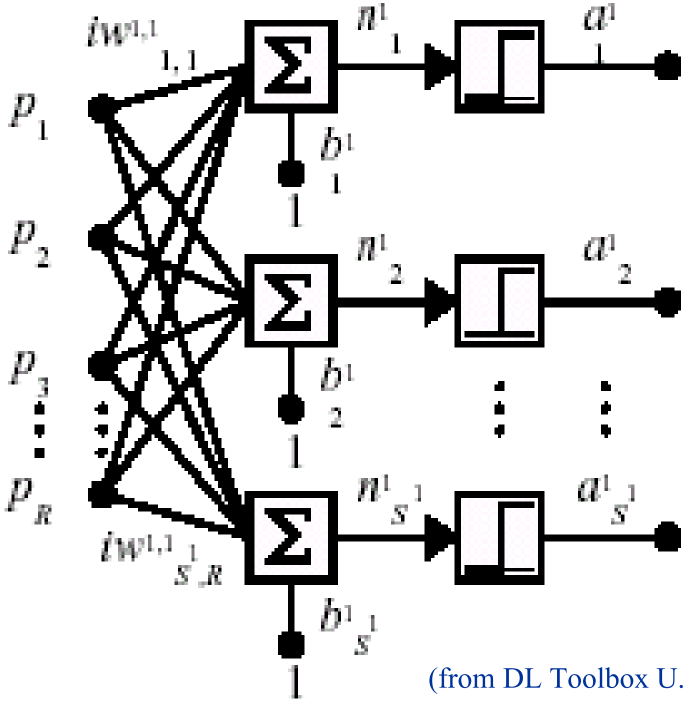
$$0,5 * 0 + 0,5 * 0,5 + b = 0$$

$$b = -0.25$$

Example 4.1: logical **OR**



Perceptron with several neurons



(from DL Toolbox U.G.)

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

- One decision boundary per neuron. For the neuron *i*, it will be

$$_i w^T p + b_i = 0$$

- It can classify into 2^S categories, with S neurons.

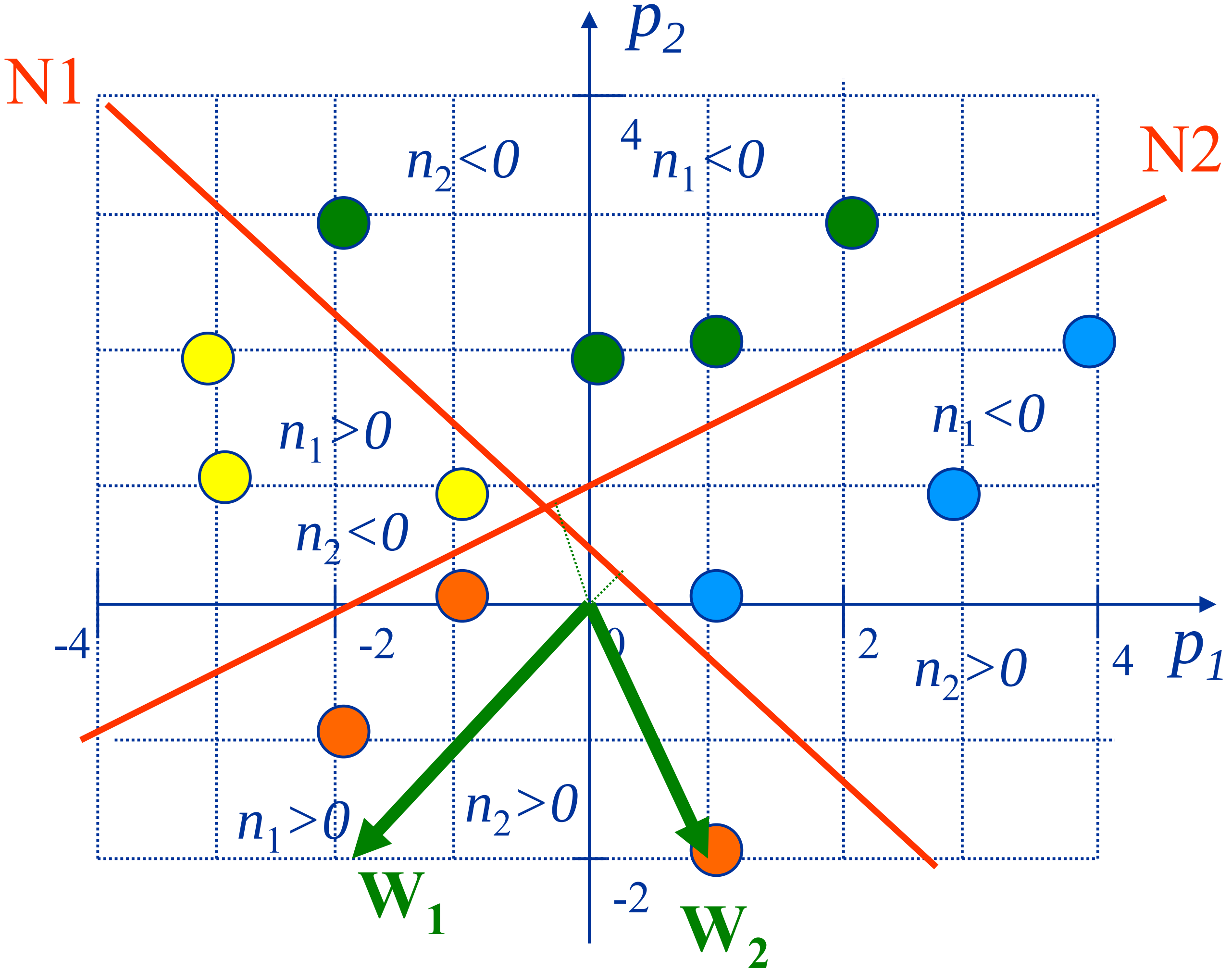
Example 4.2

Class 1: $\left\{\begin{bmatrix}-2\\3\end{bmatrix},\begin{bmatrix}0\\2\end{bmatrix},\begin{bmatrix}1\\2\end{bmatrix},\begin{bmatrix}2\\3\end{bmatrix}\right\}$ 🟢

Class 2: $\left\{\begin{bmatrix}-3\\2\end{bmatrix},\begin{bmatrix}-3\\1\end{bmatrix},\begin{bmatrix}-1\\1\end{bmatrix}\right\}$ 🟡

Class 3: $\left\{\begin{bmatrix}-2\\-1\end{bmatrix},\begin{bmatrix}-1\\0\end{bmatrix},\begin{bmatrix}1\\-2\end{bmatrix}\right\}$ 🟠

Class 4: $\left\{\begin{bmatrix}1\\0\end{bmatrix},\begin{bmatrix}3\\1\end{bmatrix},\begin{bmatrix}4\\2\end{bmatrix}\right\}$ 🔵

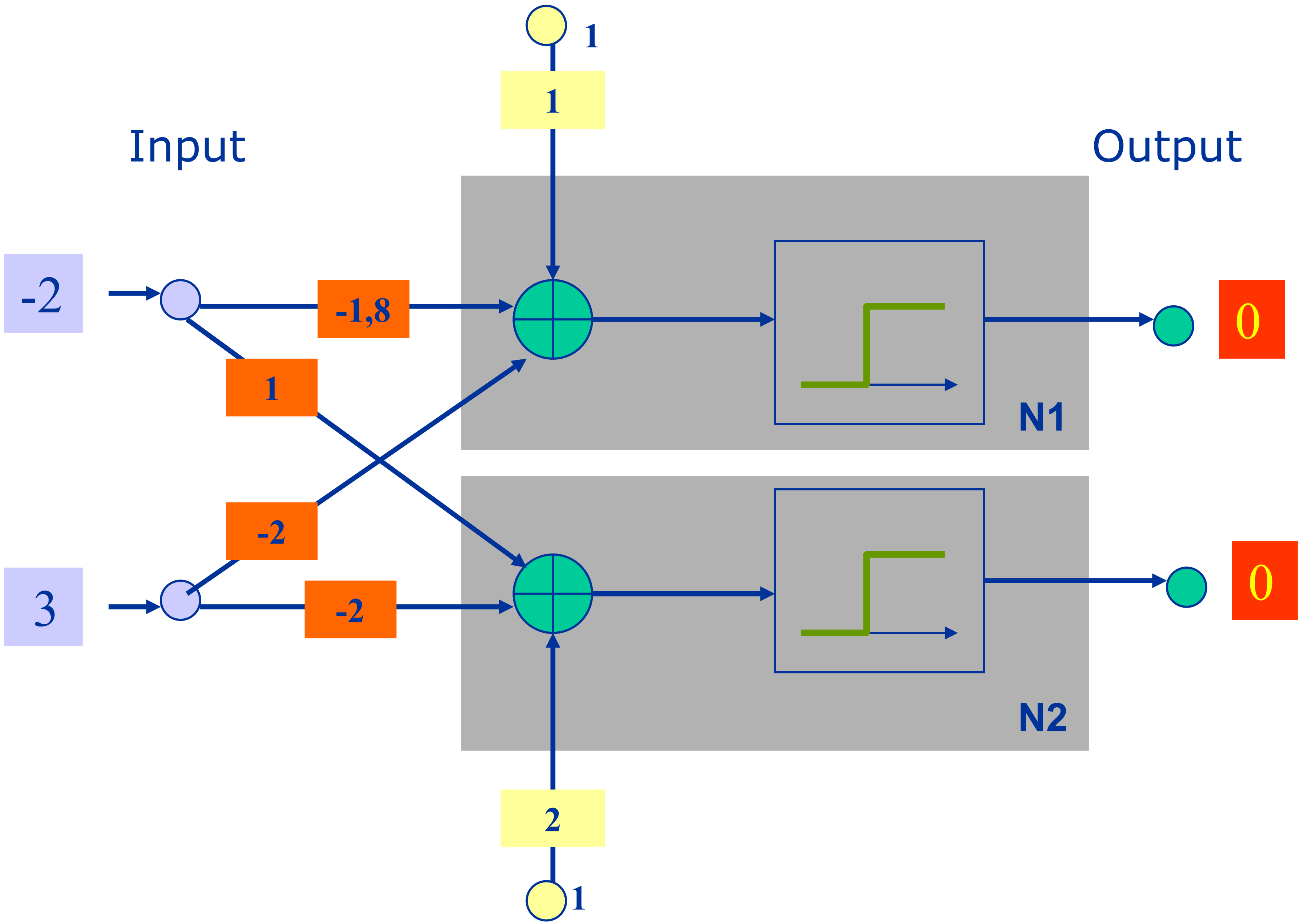


Two neurons with outputs

$\begin{bmatrix}0\\0\end{bmatrix}$,	$\begin{bmatrix}1\\0\end{bmatrix}$,	$\begin{bmatrix}0\\1\end{bmatrix}$,	$\begin{bmatrix}1\\1\end{bmatrix}$
🟢	🟡	🔵	🟠
$n_1<0$	$n_1>0$	$n_1<0$	$n_1>0$
$n_2<0$	$n_2<0$	$n_2>0$	$n_2>0$

$$W_1=\begin{bmatrix}-1,8\\-2\end{bmatrix}\qquad W_2=\begin{bmatrix}1\\-2\end{bmatrix}$$

$$b_1=1\qquad b_2=2$$


1
1
Input
Output
-2
-1,8
1
0
N1
-2
3
-2
0
N2
2
1

Learning rule (automatic learning)

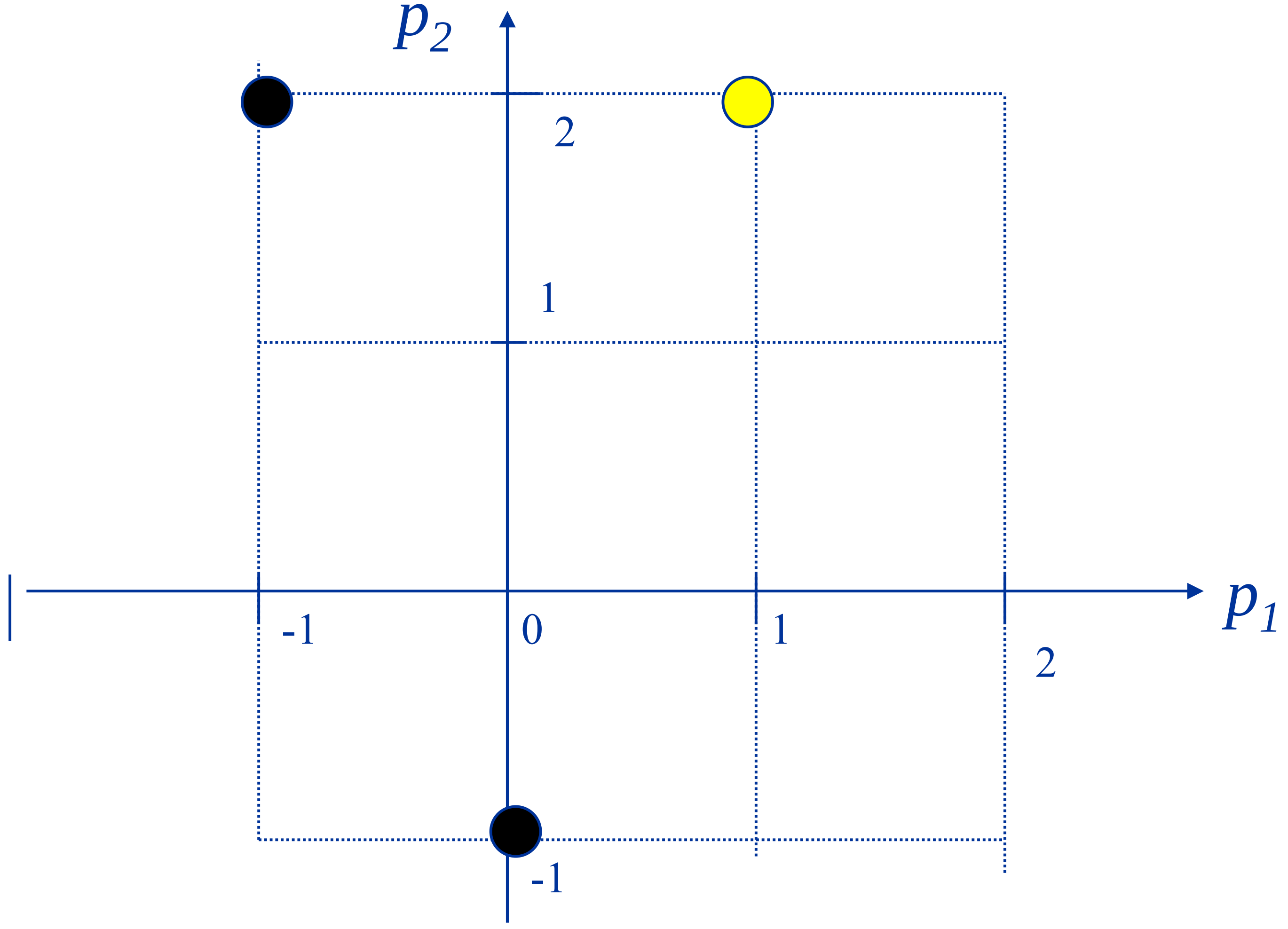
Given a training dataset: correct pairs of {input, output}:

$$\left\{p^1, t_1\right\},\ \left\{p^2, t_2\right\}, \ldots, \left\{p^Q, t_Q\right\}$$

Example 4.3:

$$\left\{p^1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, t_1 = 1\right\},\ \left\{p^2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_2 = 0\right\}, \left\{p^3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, t_3 = 0\right\}$$

$$\left\{ p^1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, t_1 = 1 \right\}, \; \left\{ p^2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_2 = 0 \right\}, \left\{ p^3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, t_3 = 0 \right\}$$

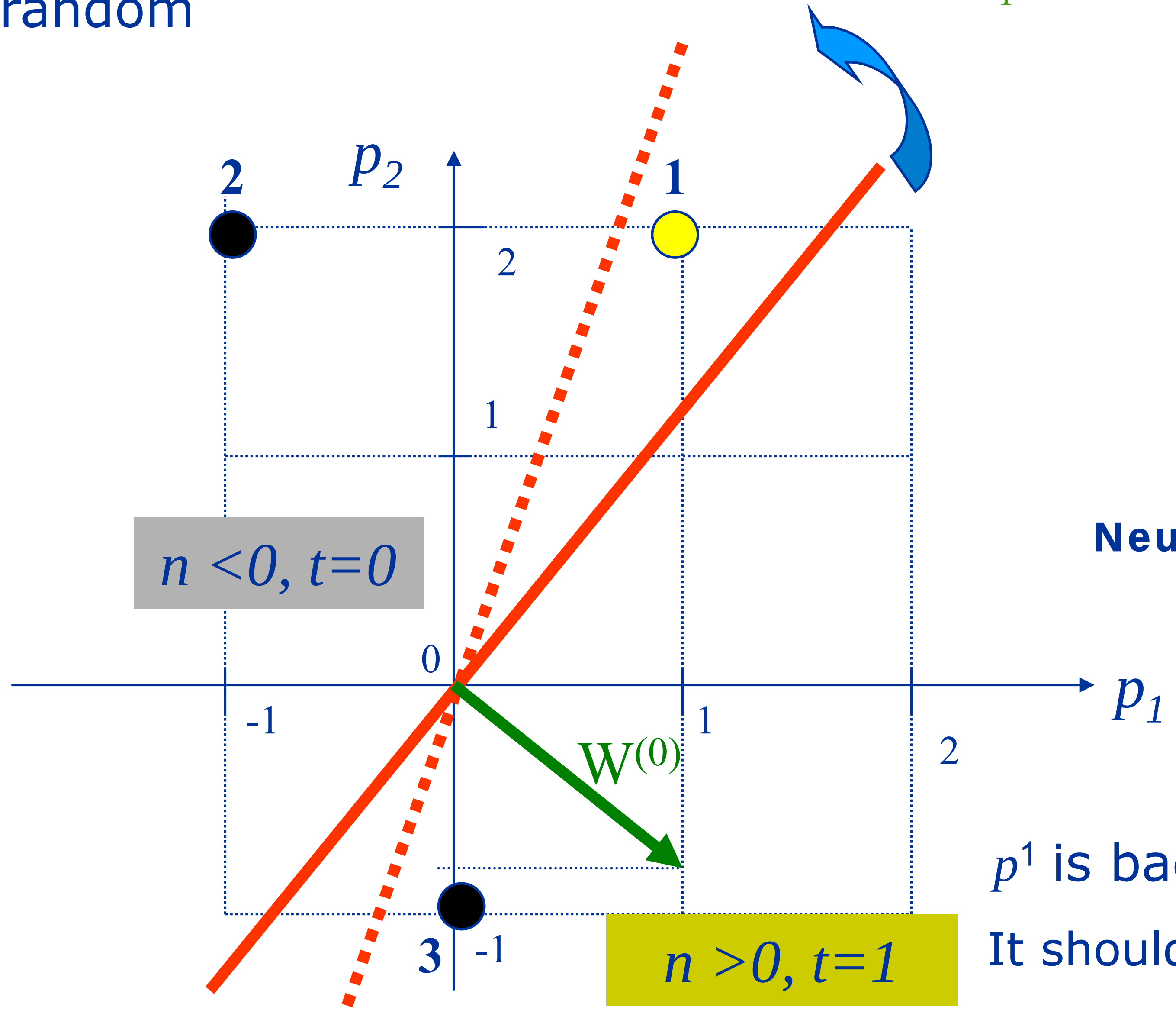


● $t = 0$

● $t = 1$

Initialization of the weights: random

$$ {}_1w^{(0)} = [1.0 \quad -0.8]^T $$



● $t=0$

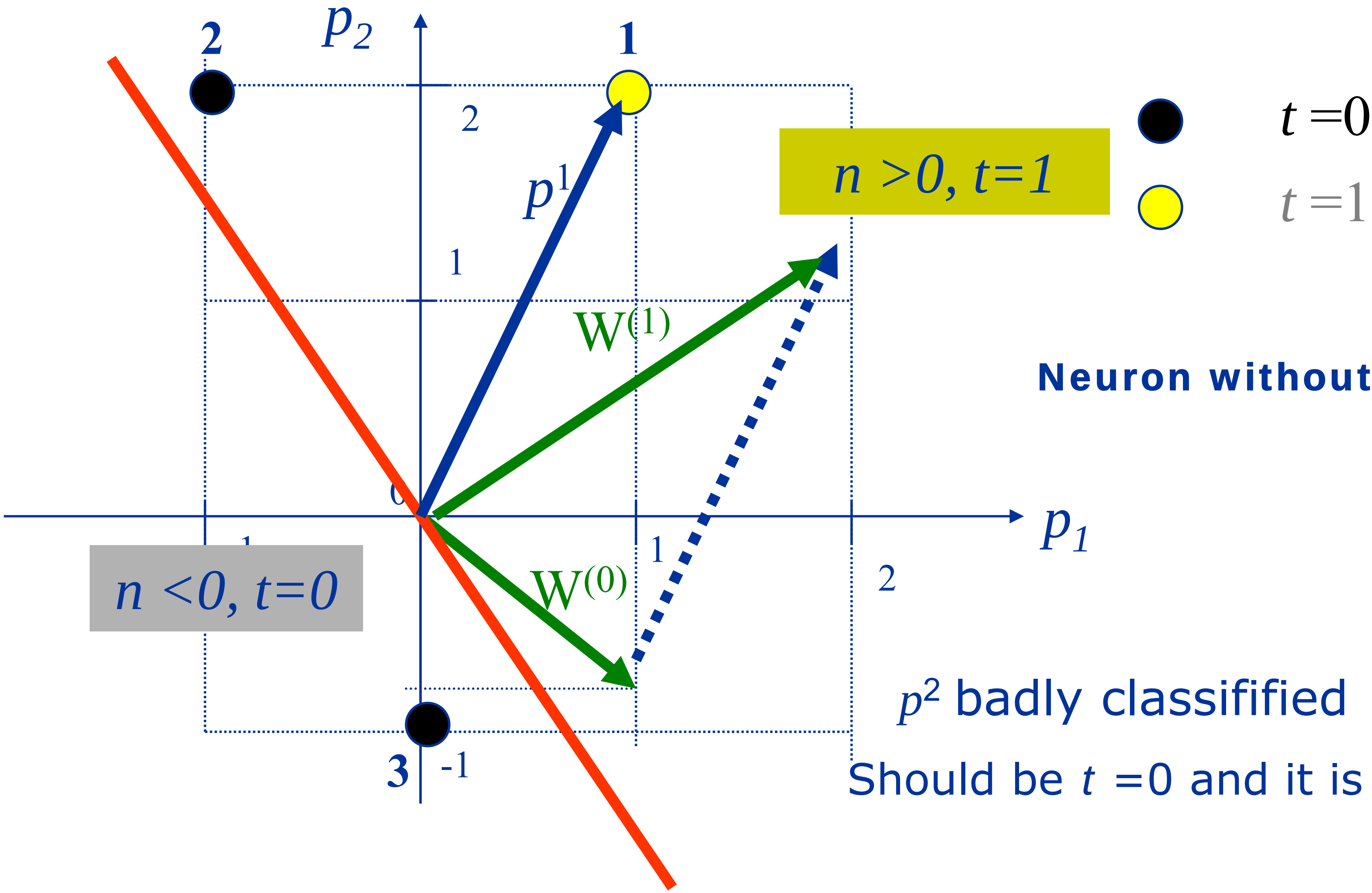
🟡 $t=1$

Neuron without bias

p^1 is badly classified !!! +

It should be $t=1$ and it is $t=0$. ⬆

$${}_1w^{(1)} = {}_1w^{(0)} + p_1$$

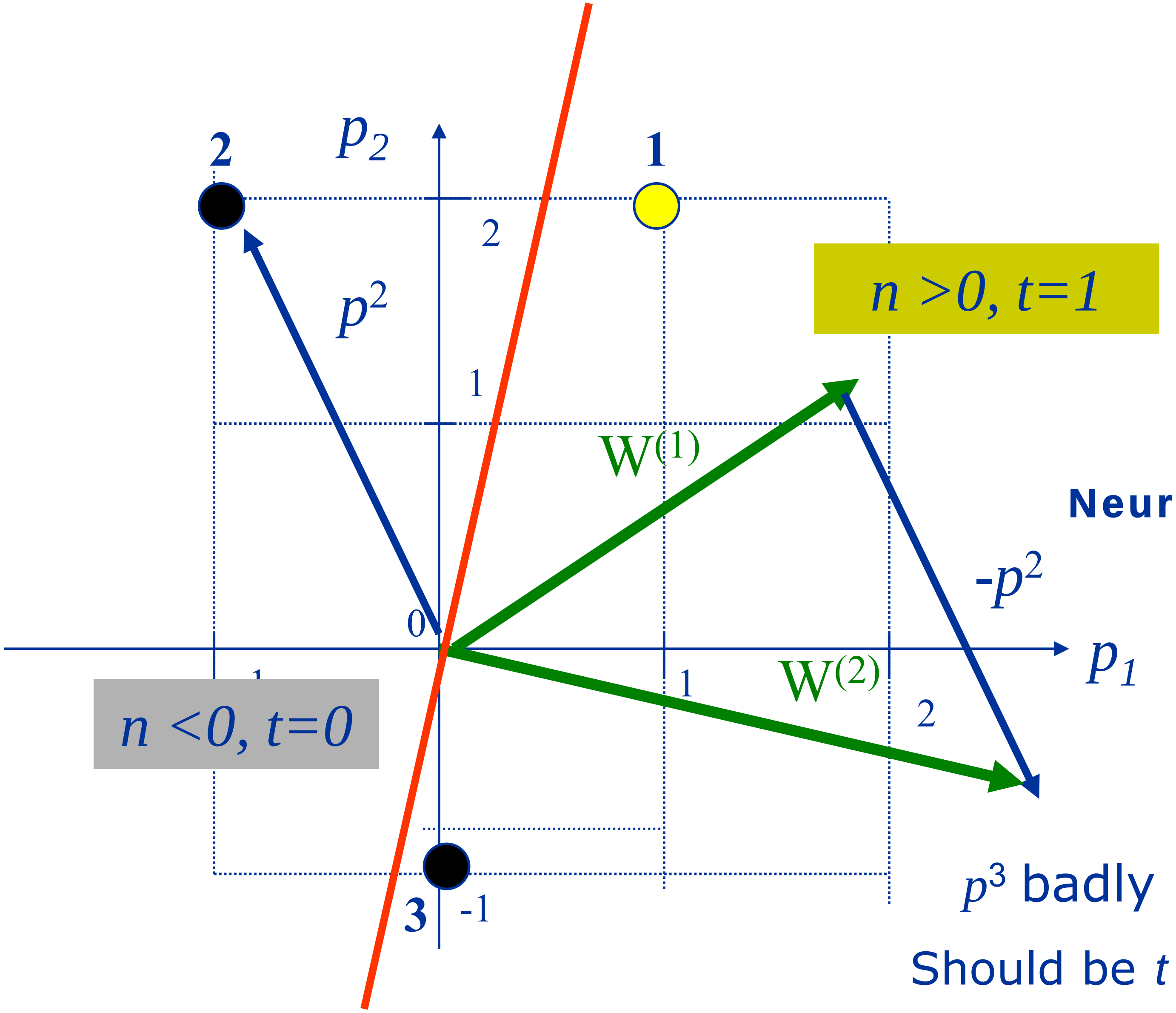


Neuron without bias

p^2 badly classified !!!

Should be $t = 0$ and it is $t = 1$

$$ {}_1w^{(2)} = {}_1w^{(1)} - p_2 $$



● t =0

● t =1

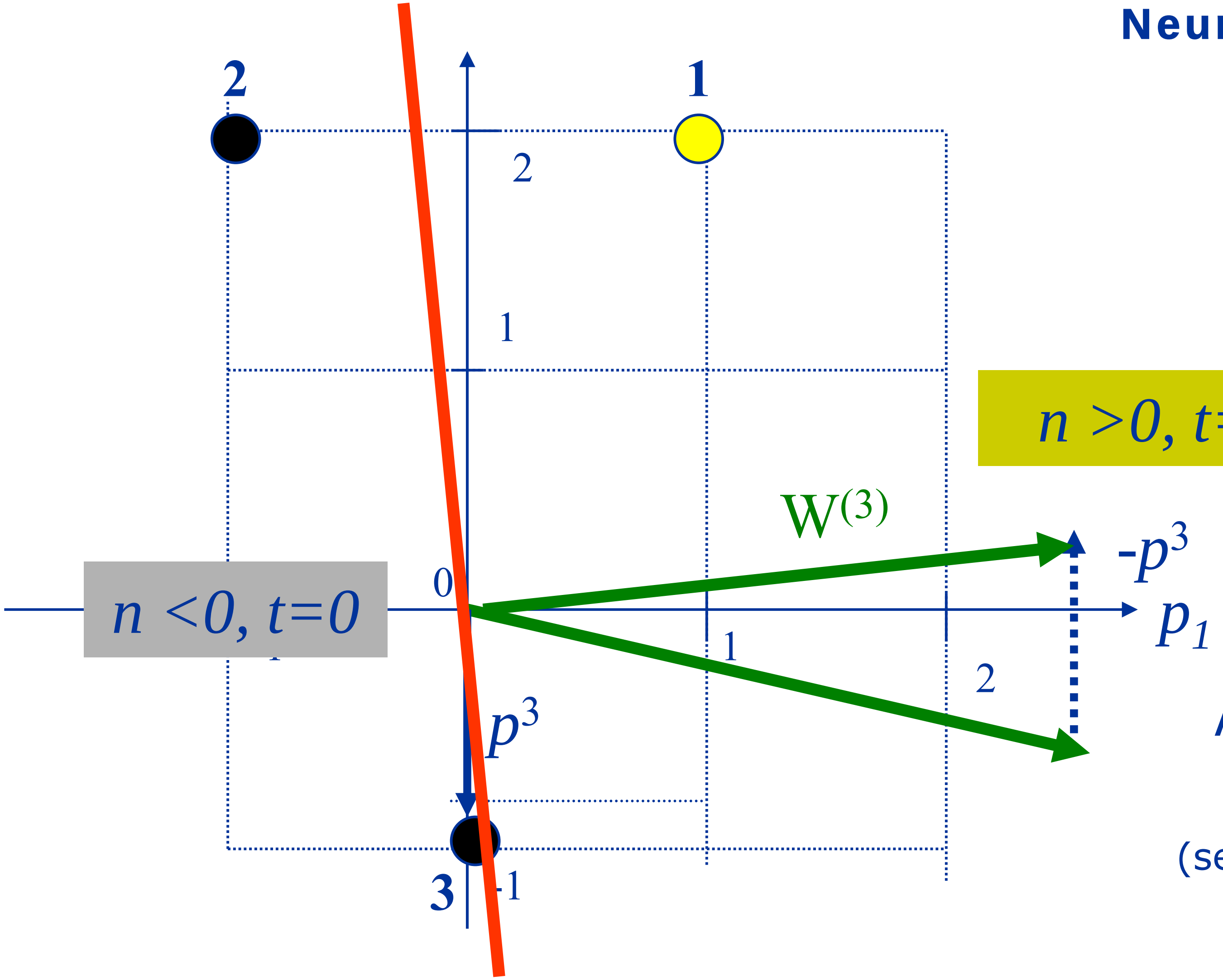
Neuron without bias

p^3 badly classified!!!

Should be t =0, it is t =*1.*

$$ {}_1w^{(3)} = {}_1w^{(2)} - p_3 $$

Neuron without bias



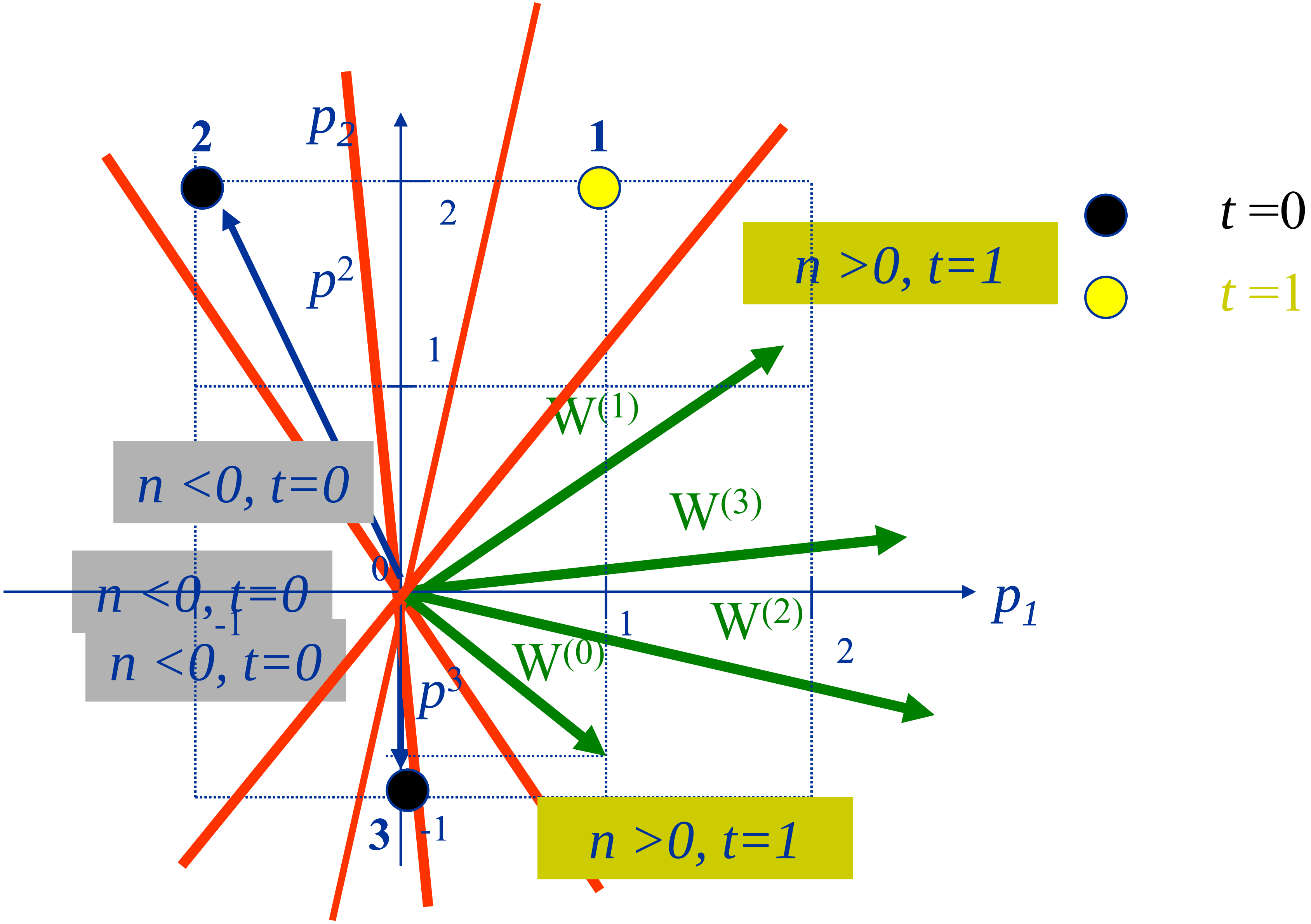
● $t=0$

● $t=1$

All points are now well classified

(see numerical calculations in Hagan 4-10)

Example 4.3:


2
p_2
1
2
p^2
1
W^(1)
n >0, t=1
n <0, t=0
W^(3)
0
n <0, t=0
-1
n <0, t=0
1
W^(2)
p_1
W^(0)
2
p^3
3
-1
n >0, t=1
● t =0
● t =1

And if a point, when analyzed, is correct ? – nothing is changed

If $t = 1$ and $a = 0$, then do $\quad {}_1w^{(new)} = {}_1w^{(old)} + p \quad e = +1$

If $t = 0$ and $a = 1$, then do $\quad {}_1w^{(new)} = {}_1w^{(old)} - p \quad e = \text{-}1$

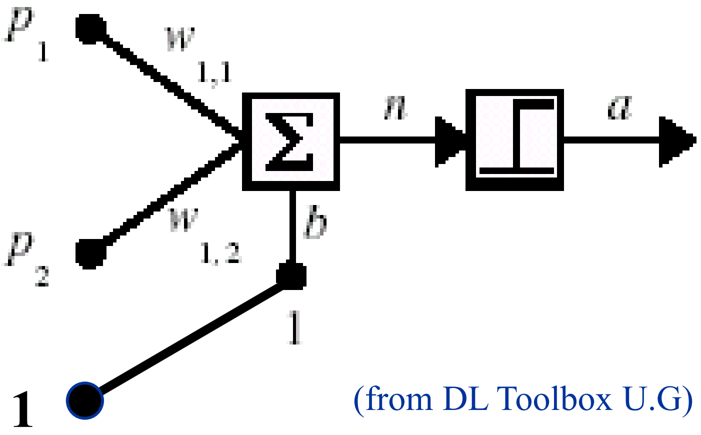
If $t = a$ then do $\quad {}_1w^{(new)} = {}_1w^{(old)} \quad e = 0$

Defining $e = t\text{-}a$ $\quad {}_1w^{(new)} = {}_1w^{(old)} + e.p$

Only some problems can be solved by one neuron without bias

What happens if the neuron has bias *b* ?

The bias is a weight of an input equal to 1 !



(from DL Toolbox U.G)

$$_1\theta = \begin{bmatrix} _1w \\ b \end{bmatrix} \qquad z = \begin{bmatrix} p \\ 1 \end{bmatrix}$$

$$a = \text{hardlim}(_1\theta^T.z)$$

Perceptron learning rule

$$_1\theta^{(new)} = {_1\theta^{(old)}} + e.z$$

Perceptron with several neurons

For the neuron i, $i = 1, \dots, S$

$$e_i = t_i - a_i \qquad z_i = \begin{bmatrix} p \\ 1 \end{bmatrix} \qquad {}_i\theta = \begin{bmatrix} {}_i w \\ b_i \end{bmatrix}$$

$${}_i\theta^{(new)} = {}_i\theta^{(old)} + e_i . z_i \quad \Longrightarrow \quad {}_i\theta^{(new)T} = {}_i\theta^{(old)T} + e_i . z_i^{\,T}$$

$$\Theta = \begin{bmatrix} {}_1\theta^T \\ \dots \\ {}_S\theta^T \end{bmatrix} \qquad z^T = \begin{bmatrix} z_1^T \\ \dots \\ z_S^T \end{bmatrix}$$

$$\boxed{\Theta^{new} = \Theta^{old} + e.z^T}$$

$$e = \begin{bmatrix} e_1 & e_2 & \dots & e_S \end{bmatrix}$$

Perceptron rule !!!

Does the procedure converge always ?

Yes, if there exists a solution: see the proof in Hagan, 4-15.

Normalized perceptron rule

- All the inputs have the same importance (problem of *outliers*)

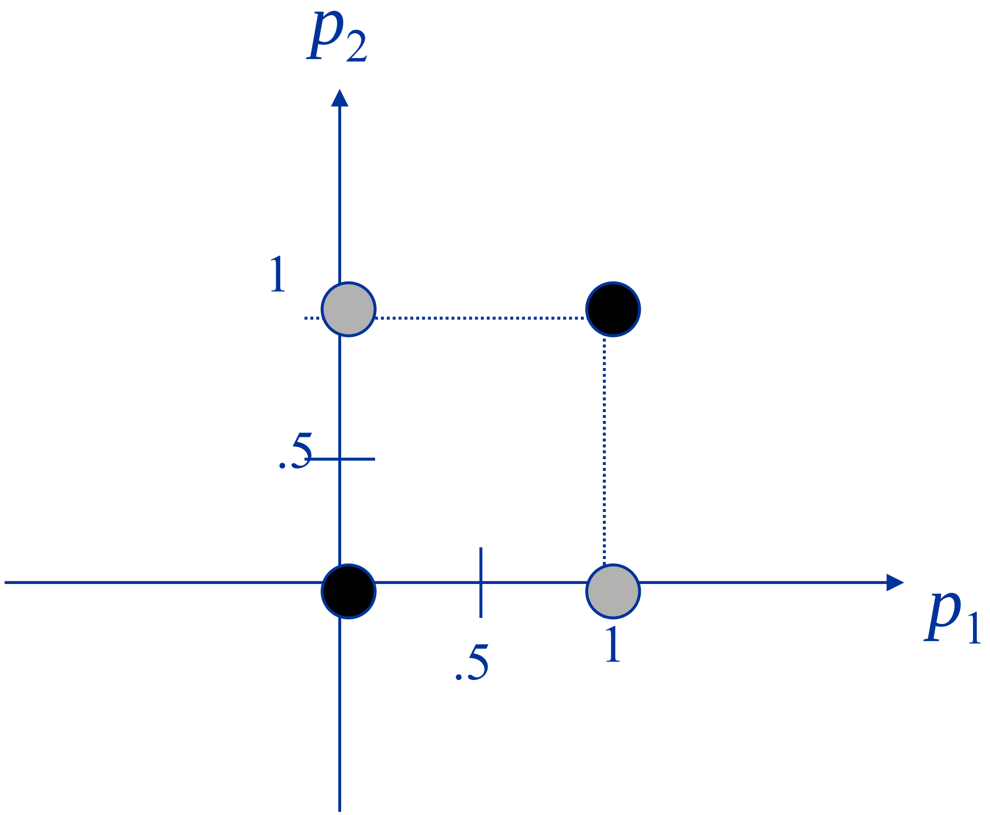
$$\Theta^{new} = \Theta^{old} + e.\frac{z^T}{\|z\|}$$

Limitations of the perceptron

It solves only linearly separable problems

Example 4.4. : exclusive **OR**, **XOR**

p_1	p_2	p_1**XOR**p_2
0	0	0
0	1	1
1	0	1
1	1	0



???

Conclusions about the perceptron

- The perceptron learning rule is a supervised one

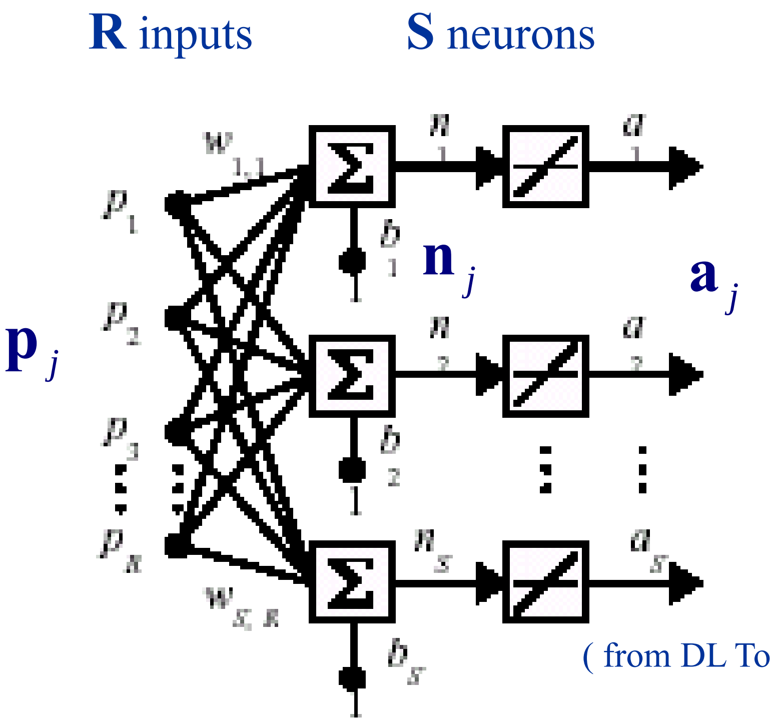
- It is simple, but powerful

- With one single layer, the binary perceptron can solve only linearly separable problems.

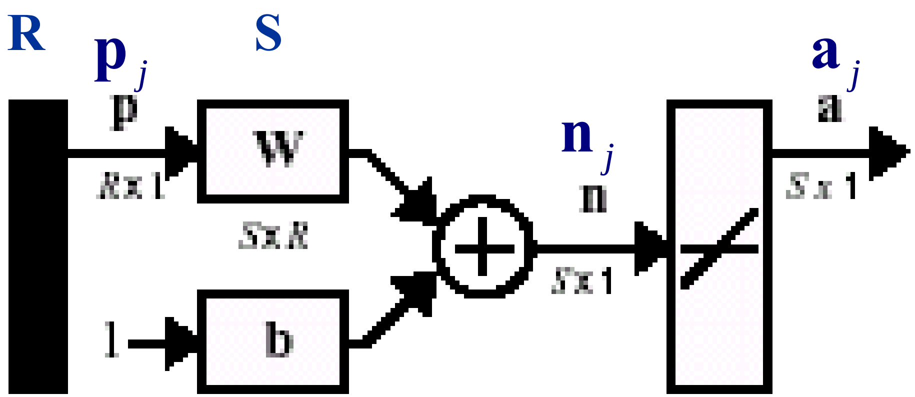
- The problems nonlinearly separable can be solved by multilayer architectures.

4.9 General Single Layer Networks

4.9.1. The ADALINE , Adaptive Linear Network



(from DL Toolbox U.G.)



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

$$a_{ij} = purelin(\mathbf{w}_i^T \mathbf{p}_j + b_i)$$

$$\mathbf{w}_i^T \triangleq i\text{-th row of } \mathbf{W}$$

Notation for one layer of S neurons

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1R} \\ w_{21} & w_{22} & \cdots & w_{2R} \\ w_{31} & w_{32} & \cdots & w_{3R} \\ w_{S1} & w_{S2} & \cdots & w_{SR} \end{bmatrix}_{S\times R} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \cdots \\ \mathbf{w}_S^T \end{bmatrix}_{S\times R} \qquad \boldsymbol{\Theta} = \begin{bmatrix} w_{11} & \cdots & w_{1R} & b_1 \\ w_{21} & \cdots & w_{2R} & b_2 \\ \cdots & \cdots & \cdots & \cdots \\ w_{S1} & \cdots & w_{SR} & b_S \end{bmatrix}_{S\times(R+1)} = \begin{bmatrix} \boldsymbol{\theta}_1^T \\ \boldsymbol{\theta}_2^T \\ \cdots \\ \boldsymbol{\theta}_S^T \end{bmatrix} = \begin{bmatrix} \mathbf{W} & \mathbf{b} \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_Q \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1Q} \\ p_{21} & p_{22} & \cdots & p_{2Q} \\ \cdots & \cdots & \cdots & \cdots \\ p_{R1} & p_{R2} & \cdots & p_{RQ} \end{bmatrix}_{RxQ} \qquad \mathbf{P}^T = \begin{bmatrix} p_{11} & p_{21} & \cdots & p_{R1} \\ p_{12} & p_{22} & \cdots & p_{R2} \\ \cdots & \cdots & \cdots & \cdots \\ p_{1Q} & p_{2Q} & \cdots & p_{RQ} \end{bmatrix}_{Q\times R} = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \cdots \\ \mathbf{p}_Q^T \end{bmatrix}_{Q\times R}$$

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_Q \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1Q} \\ \cdots & \cdots & \cdots & \cdots \\ p_{R1} & p_{R2} & \cdots & p_{RQ} \\ 1 & 1 & \cdots & 1 \end{bmatrix}_{(R+1)xQ} = \begin{bmatrix} \mathbf{P} \\ \mathbf{1} \end{bmatrix}$$

Notation for one layer of S neurons

$$\mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & \dots & t_{1Q} \\ t_{21} & t_{22} & \dots & t_{2Q} \\ \dots & \dots & \dots & \dots \\ t_{S1} & t_{S2} & \dots & t_{SQ} \end{bmatrix}_{S\times Q} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \dots & \mathbf{t}_Q \end{bmatrix}_{S\times Q} = \begin{bmatrix} \mathbf{T}_1^T \\ \mathbf{T}_2^T \\ \dots \\ \mathbf{T}_S^T \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1Q} \\ a_{21} & a_{22} & \dots & a_{2Q} \\ \dots & \dots & \dots & \dots \\ a_{S1} & a_{S2} & \dots & a_{SQ} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1^T \\ \mathbf{A}_2^T \\ \dots \\ \mathbf{A}_S^T \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_Q \end{bmatrix}_{S\times Q}$$

Notation for one layer of S neurons

$\mathbf{p}_j \triangleq$ j-the input vector of dimension R

$p_{ij} \triangleq$ i-th component of the $j-$th input vector,

$\mathbf{a}_j \triangleq$ j-th output vector obtained with the j-th input $\mathbf{p}_j$

$\mathbf{A}_i^T \triangleq$ row vector of the outputs of the i-th neuron for all the inputs from 1 to Q

$a_{ij} \triangleq$ i-th component da j-th output $\mathbf{a}_j$, output of the i-th neuron

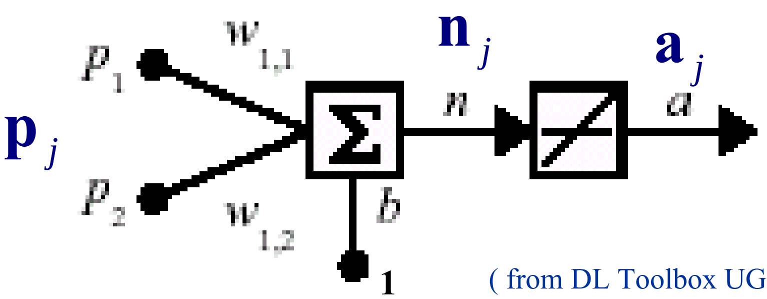
$\mathbf{t}_j \triangleq$ j-th target output, when the input $\mathbf{p}_j$ is applied

$\mathbf{T}_i^T \triangleq$ row vector of the target of i-th neuron for all the inputs from 1 to Q

$t_{ij} \triangleq$ component i of the j-th target $\mathbf{t}_j$

$w_{ij} \triangleq$ weight between the neuron i and the component j of the input vector

Case of one neuron with two inputs

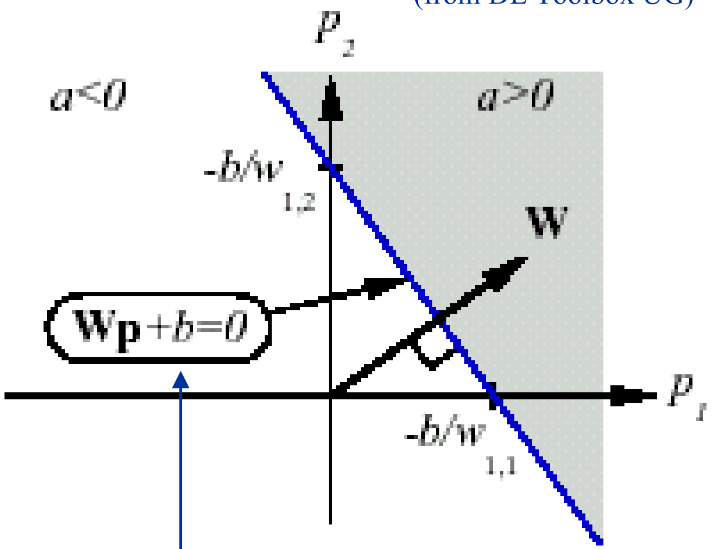


(from DL Toolbox UG)

$$a > 0 \quad \text{if} \quad n > 0$$
$$a = 0 \quad \text{if} \quad n = 0$$
$$a < 0 \quad \text{if} \quad n < 0$$

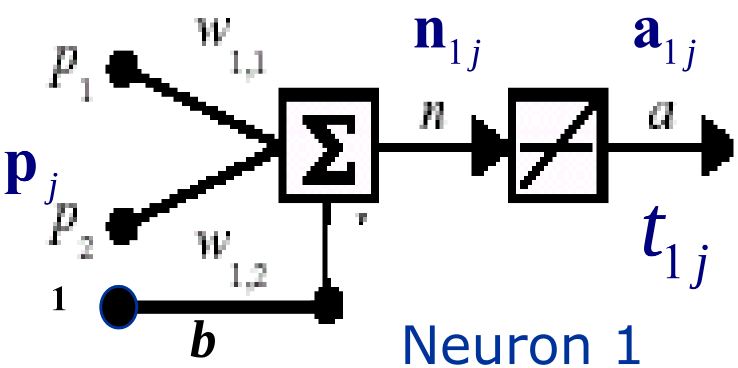
$$a_{1j} = purelin(n_j) = purelin(\mathbf{w}_1^T \mathbf{p}_j + b) = \mathbf{w}_1^T \mathbf{p}_j + b = w_{11} p_{1j} + w_{12} p_{2j} + b$$

(from DL Toolbox UG)



Decision boundary (W=w$_1^T$)

- Divides the plane into two zones.
- Can classify patterns linearly separable.
- The LMSE training optimizes the position of the boundary with respect to the training patterns (advantage over the perceptron).



Neuron 1

$$\boldsymbol{\theta}_1 = \begin{bmatrix}\mathbf{w}_1\\ b\end{bmatrix} = \begin{bmatrix} w_{11}\\ w_{12}\\ b\end{bmatrix} \qquad \mathbf{z}_j = \begin{bmatrix}p_{1j}\\ p_{2j}\\ 1\end{bmatrix} = \begin{bmatrix}\mathbf{p}_j\\ 1\end{bmatrix}$$

$$a_{1j} = {}_1\mathbf{w}^T\mathbf{p}_j + b = w_{11}p_{1j} + w_{12}p_{2j} + b = \begin{bmatrix}p_{1j} & p_{2j} & 1\end{bmatrix}\begin{bmatrix} w_{11}\\ w_{12}\\ b\end{bmatrix} = \mathbf{z}_j{}^T\boldsymbol{\theta}_1 =$$

3 unknowns

x

$$= \begin{bmatrix} w_{11} & w_{12} & b\end{bmatrix}\begin{bmatrix}p_{1j}\\ p_{2j}\\ 1\end{bmatrix} = \boldsymbol{\theta}_1{}^T\mathbf{z}_j$$

The 3 needed equations are obtained by the application of 3 inputs z_1,z_2, z_3:

1st $$a_{11} = t_{11} = w_{11}p_{11} + w_{12}p_{21} + b = \begin{bmatrix} w_{11} & w_{12} & b\end{bmatrix}\begin{bmatrix}p_{11}\\ p_{21}\\ 1\end{bmatrix} = \boldsymbol{\theta}_1{}^T\mathbf{z}_1$$

2nd $$a_{12} = t_{12} = w_{11} p_{12} + w_{12} p_{22} + b = \begin{bmatrix} w_{11} & w_{12} & b \end{bmatrix} \begin{bmatrix} p_{12} \\ p_{22} \\ 1 \end{bmatrix} = \boldsymbol{\theta}_1^T \mathbf{z}_2$$

3rd $$a_{13} = t_{13} = w_{11} p_{13} + w_{12} p_{23} + b = \begin{bmatrix} w_{11} & w_{12} & b \end{bmatrix} \begin{bmatrix} p_{13} \\ p_{23} \\ 1 \end{bmatrix} = \boldsymbol{\theta}_1^T \mathbf{z}_3$$

1st 2nd 3rd

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & b \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \boldsymbol{\theta}_1^T \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \mathbf{z}_3 \end{bmatrix} = \boldsymbol{\theta}_1^T \mathbf{Z}$$

$$\mathrm{A}_1^T = T_1^T = \boldsymbol{\theta}_1^T \mathbf{Z}$$ (supervised training, the output A should be equal to the target T)

$$\mathbf{A}_1^T = \mathbf{T}_1^T = \boldsymbol{\theta}_1{}^T \mathbf{Z} \longrightarrow \boldsymbol{\theta}_1{}^T = \mathbf{T}_1^T \mathbf{Z}^{-1}$$

Does exist $(\mathbf{Z})^{-1}$??? → Almost never!!!

What to do, to find a solution ???

More inputs → More equations than unknowns

One can use the pseudo-inverse of $\mathbf{Z}$ instead of the inverse, for a solution with non-null error:

$$\boldsymbol{\theta}_1{}^T = \mathbf{T}_1^T \mathbf{Z}^+ = \mathbf{T}_1^T \mathbf{Z}^T (\mathbf{Z}\mathbf{Z}^T)^{-1} \quad (Q > R + \underset{\uparrow \atop bias}{1})$$

Is it possible? Exists $(\mathbf{ZZ}^T)^{-1}$? Computational cost ?

It would be better to process the data iteratively

Minimization of the mean square error: LMSE (Least Mean Square Error)

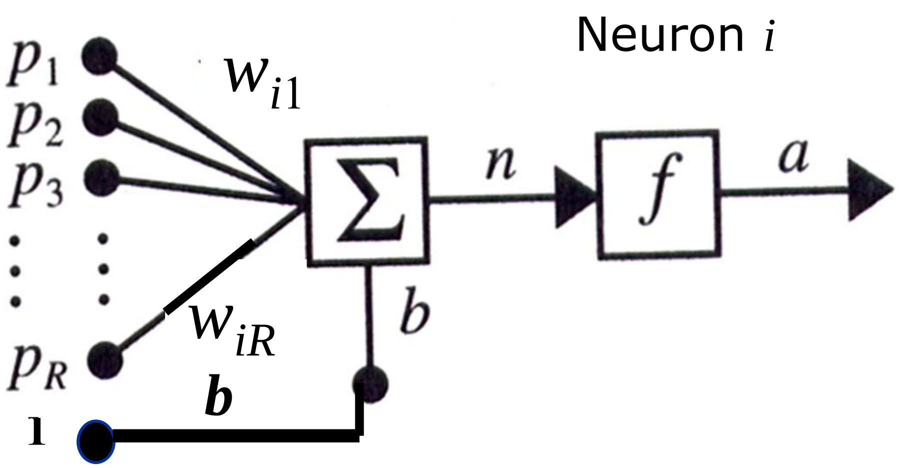
- Supervised training: one gives to the network a set *Q* of input-target pairs; for neuron *i*

$$\{\mathbf{p}_1, t_{i1}\}, \{\mathbf{p}_2, t_{i2}\}, \ldots, \{\mathbf{p}_Q, t_{iQ}\}$$

- The obtained output a_{ik} is compared with the target t_{ik}, and one obtains the error $e_{ik} = t_{ik} - a_{ik}$, $k=1, \ldots, Q$

- The LMSE algorithm will adjust the weights and the bias in order to minimize the mean square error, *mse*, penalizing in the same way the positive and the negative errors, i.e., minimizing

$$mse = \frac{1}{Q}\sum_{k=1}^{Q} e_{ik}{}^{2} = \frac{1}{Q}\sum_{k=1}^{Q}(t_{ik} - a_{ik})^{2}$$



$$\boldsymbol{\theta}_i = \begin{bmatrix} \mathbf{w}_i \\ b \end{bmatrix} = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \dots \\ w_{iR} \\ b \end{bmatrix}_{R+1} \qquad \mathbf{z}_j = \begin{bmatrix} \mathbf{p}_j \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1j} \\ p_{2j} \\ \dots \\ p_{Rj} \\ 1 \end{bmatrix}$$

$$a_{ij} = \mathbf{w}_i^T p_j + b = w_{i1} p_{1j} + w_{i2} p_{2j} + \dots + w_{iR} p_{Rj} + 1b = \boldsymbol{\theta}_i^T \mathbf{z}_j = \mathbf{z}_j^T \boldsymbol{\theta}_i$$

$$e_{ij}{}^2 = (t_{ij} - a_{ij})^2 = (t_{ij} - \mathbf{z}_j{}^T \boldsymbol{\theta}_i)^2 = (t_{ij} - \mathbf{z}_j{}^T \boldsymbol{\theta}_i)^T (t_{ij} - \mathbf{z}_j{}^T \boldsymbol{\theta}_i)$$

$$= (t_{ij}{}^T - \boldsymbol{\theta}_i{}^T \mathbf{z}_j)(t_{ij} - \mathbf{z}_j{}^T \boldsymbol{\theta}_i) = t_{ij}{}^2 - 2t_{ij} \mathbf{z}_j{}^T \boldsymbol{\theta}_i + \boldsymbol{\theta}_i{}^T \mathbf{z}_j \mathbf{z}_j{}^T \boldsymbol{\theta}_i$$

For a set of Q training pairs, the sum of all squared errors will be

$$\sum_{k=1}^{Q} e_{ik}{}^2 = e_{i1}{}^2 + e_{i2}{}^2 + \dots + e_{iQ}{}^2 = \begin{bmatrix} e_{i1} & e_{i2} & \dots & e_{iQ} \end{bmatrix} \begin{bmatrix} e_{i1} \\ e_{i2} \\ \dots \\ e_{iQ} \end{bmatrix} = \mathbf{e}_i{}^T \mathbf{e}_i$$

Consider the concatenated vectors of inputs and outputs

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_Q \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1Q} \\ p_{22} & p_{22} & \dots & p_{2Q} \\ \dots & \dots & \dots & \dots \\ p_{R1} & p_{R2} & \dots & p_{RQ} \\ 1 & 1 & \dots & 1 \end{bmatrix}_{(R+1)\times Q} \qquad \mathbf{Z}^T = \begin{bmatrix} \mathbf{z}_1{}^T \\ \mathbf{z}_2{}^T \\ \dots \\ \mathbf{z}_Q{}^T \end{bmatrix}_{Q\times(R+1)} \qquad \boldsymbol{\theta}_i = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \dots \\ w_{iR} \\ b \end{bmatrix}_{(R+1)}$$

$$\mathbf{A}_i^T = \begin{bmatrix} a_{i1} & a_{i2} & \dots & a_{iQ} \end{bmatrix}_{1\times Q} \qquad \mathbf{T}_i = \begin{bmatrix} t_{i1} & t_{i2} & \dots & t_{iQ} \end{bmatrix}_{1\times Q}$$

$$\mathbf{A}_i^T = \boldsymbol{\theta}_i{}^{\mathrm{T}}\mathbf{Z}$$

$$Q \geq R+1$$

(More equations than unknowns, there is no exact solution. We look for the one that minimizes the squared errors (MSE))

$$\mathbf{e}_i^T = \mathbf{T}_i^T - \mathbf{A}_i^T = \mathbf{T}_i^T - \boldsymbol{\theta}_i{}^{\mathrm{T}}\mathbf{Z}$$

Now $$F(\boldsymbol{\theta}_i) = \mathbf{e}_i{}^T\mathbf{e}_i = (\mathbf{T}_i^T - \mathbf{A}_i^T)(\mathbf{T}_i - \mathbf{A}_i) = (\mathbf{T}_i^T - \boldsymbol{\theta}_i{}^T\mathbf{Z})(\mathbf{T}_i - \mathbf{Z}^T\boldsymbol{\theta}_i)$$

$$= \mathbf{T}_i^T\mathbf{T}_i - \mathbf{T}_i^T\mathbf{Z}^T\boldsymbol{\theta}_i - \boldsymbol{\theta}_i{}^T\mathbf{Z}\mathbf{T}_i + \boldsymbol{\theta}_i{}^T\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i$$

$$= \mathbf{T}_i^T\mathbf{T}_i - 2\mathbf{T}_i^T\mathbf{Z}^T\boldsymbol{\theta}_i + \boldsymbol{\theta}_i{}^T\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i$$

This expression must be minimized with respect to θ_i :

gradient: $$\nabla F(\boldsymbol{\theta}_i) = \nabla(\mathbf{T}_i^T\mathbf{T}_i - 2\mathbf{T}_i^T\mathbf{Z}^T\boldsymbol{\theta}_i + \boldsymbol{\theta}_i{}^T\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i) =$$

$$= \nabla(\mathbf{T}_i^T\mathbf{T}_i) - 2\nabla(\mathbf{T}_i^T\mathbf{Z}^T\boldsymbol{\theta}_i) + \nabla(\boldsymbol{\theta}_i{}^T\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i) =$$

$$= -2\mathbf{Z}\mathbf{T}_i + 2\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i$$

$$-2\mathbf{Z}\mathbf{T}_i + 2\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i = 0 \quad \Leftrightarrow \quad \mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i = \mathbf{Z}\mathbf{T}_i \Leftrightarrow$$

$$\boldsymbol{\theta}_i = (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}_i^T = (\mathbf{Z}^T)^+\mathbf{T}_i$$

$$\Downarrow$$

$$((R+1)\times Q)\times(Q\times(R+1) = (R+1)\times(R+1)$$

Note:

$$\nabla(\mathbf{a}^T\mathbf{x}) = \nabla(\mathbf{x}^T\mathbf{a}) = \mathbf{a}$$

$$\nabla(\mathbf{x}^T\mathbf{A}\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{A}^T\mathbf{x} = 2\mathbf{A}\mathbf{x}$$

(if A is symmetric)

Is it a minimum? Is it a maximum? Is it a saddle point ?

Second derivative

$$\frac{\partial^2(\mathbf{e}_i^{\,T}\mathbf{e}_i)}{\partial\boldsymbol{\theta}_i^{\,2}} = \frac{\partial(-2\mathbf{Z}\mathbf{T}_i + 2\mathbf{Z}\mathbf{Z}^T\boldsymbol{\theta}_i)}{\partial\boldsymbol{\theta}_i} = \mathbf{Z}\mathbf{Z}^T$$

$\mathbf{Z}\mathbf{Z}^T > \mathbf{0},$ has a unique global minimum

$\mathbf{Z}\mathbf{Z}^T \geq \mathbf{0},$ has a weak global minimum or has no stationary point.

Note about the signal of matrices

The signal of a symmetric matrix A is related to the signal of its eigenvalues:

A>0, positive definite, all eigenvalues are >0
A≥0, positive semidefinite, eigenvalues are ≥0
A≤0, negative semidefinite, eigenvalues are ≤ 0
A<0, negative definite, eigenvalues are <0
A is indefinite if it has positive and negative eigenvalues

Interpretations of the condition of minimum:

$$F(\boldsymbol{\theta}_i) = \mathbf{T}_i^T \mathbf{T}_i - 2\mathbf{T}_i^T \mathbf{Z}^T \boldsymbol{\theta}_i + \boldsymbol{\theta}_i{}^T \mathbf{Z}\mathbf{Z}^T \boldsymbol{\theta}_i$$

If the inputs of the network are random vectors, then the error will be random and the objective function to be minimized will be:

$$F(\boldsymbol{\theta}_i) = E[\mathbf{T}_i^T \mathbf{T}_i - 2\mathbf{T}_i^T \mathbf{Z}^T \boldsymbol{\theta}_i + \boldsymbol{\theta}_i{}^T \mathbf{Z}\mathbf{Z}^T \boldsymbol{\theta}_i]$$

$$F(\boldsymbol{\theta}_i) = E[\mathbf{T}_i{}^T \mathbf{T}_i] - 2E[\mathbf{T}_i^T \mathbf{Z}^T]\boldsymbol{\theta}_i + \boldsymbol{\theta}_i{}^T E[\mathbf{Z}\mathbf{Z}^T]\boldsymbol{\theta}_i$$

Matrix of the correlation between the inputs and the targets

Matrix of the inputs auto-correlation

A correlation matrix is positive definite (>0) or is positive semidefinite (≥0).

If the inputs applied to the network are uncorrelated, the correlation matrix $\mathbf{ZZ}^T$ is diagonal, being the diagonal elements the squares of the inputs. In these conditions the correlation matrix is positive definite, and the global minimum exists, and it is unique.

The existence or not of a unique global minimum depends on the characteristics of the input training set.

For a network with S neurons,

$$\mathbf{\Theta} = \begin{bmatrix} \boldsymbol{\theta}_1{}^T \\ \cdots \\ \boldsymbol{\theta}_S{}^T \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots & w_{1R} & b_1 \\ w_{21} & \cdots & w_{2R} & b_2 \\ \cdots & \cdots & \cdots & \cdots \\ w_{S1} & \cdots & ..w_{SR}. & b_S \\ & & \cdots & \end{bmatrix}_{S\times(R+1)}$$

$$\xrightarrow{\text{neuron}}$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1Q} \\ t_{21} & t_{22} & \cdots & t_{2Q} \\ \cdots & \cdots & \cdots & \cdots \\ t_{S1} & t_{S2} & \cdots & t_{SQ} \end{bmatrix}_{S\times Q} = \begin{bmatrix} T_1^T \\ T_2^T \\ T_S^T \end{bmatrix}$$

$$\mathbf{T}^T = \begin{bmatrix} T_1^T \\ T_2^T \\ T_S^T \end{bmatrix}^T = \begin{bmatrix} T_1 & T_2 & \cdots & T_S \end{bmatrix}$$

$$(\mathrm{Q} > (\mathrm{R}+1))$$

$$\boldsymbol{\theta}_i = (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}_i = (\mathbf{Z}^T)^+\mathbf{T}_i$$

$$\begin{aligned}\boldsymbol{\Theta}^T &= \begin{bmatrix}\boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \dots & \boldsymbol{\theta}_S\end{bmatrix} = \begin{bmatrix}(\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}_1 & (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}_2 & \dots & (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}_S\end{bmatrix} = \\ &= (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\begin{bmatrix}\mathbf{T}_1 & \mathbf{T}_2 & \dots & \mathbf{T}_S\end{bmatrix} \\ &= (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}^T\end{aligned}$$

$$\boldsymbol{\Theta} = ((\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{T}^T)^T = \mathbf{T}\mathbf{Z}^+$$

Possible? Exists $(\mathbf{ZZ}^T)^{-1}$? Computational cost ?

The computation of the inverse, for a high number *R* of inputs, is difficult. Does it exist a (computationally) more simple algorithm?

Iterative LMSE algorithm

Initialize the weights (for example randomly)

$k=1$ *(iteration 1)* $w_{ij}{}^{k}= w_{ij}{}^{1}$

For q from 1 to Q do (case of one neuron i) :

For each training pair (p_{iq} , t_{iq})

1st compute the neuron output a_{iq} for the input p_{iq}

2nd compute the squared error $e_{ip}{}^{2}=(t_{ip}-a_{ip})^{2}$

3rd compute the gradient of the squared error with respect to the weights and the bias :

$$\left[\nabla e_{iq}{}^{2}\right] = \frac{\partial e_{iq}{}^{2}}{\partial w_{ij}{}^{k}} = 2e_{iq}\,\frac{\partial e_{iq}}{\partial w_{ij}{}^{k}} \quad \text{for the weights} \quad j=1,2,\ldots,R$$

$$\left[\nabla e_{iq}{}^{2}\right]_{R+1} = \frac{\partial e_{iq}{}^{2}}{\partial b} = 2e_{iq}\,\frac{\partial e_{iq}}{\partial b} \quad \text{for the bias}$$

Computing the derivatives (case of linear neuron)

$$\frac{\partial e_{iq}}{\partial w_{ij}{}^{k}} = \frac{\partial \left[t_{iq} \text{ - } a_{iq} \right]}{\partial w_{ij}{}^{k}} = \frac{\partial}{\partial w_{ij}{}^{k}} [t_{iq} - (\mathbf{w}_i{}^{\mathbf{T}k} \mathbf{p}_q + b)]$$

$$= \frac{\partial}{\partial w_{ij}{}^{k}} \left[t_{iq} - \left(\sum_{j=1}^{R} w_{ij}{}^{k} p_{jq} + b \right) \right] = -p_{jq} \qquad j = 1, 2, \ldots R$$

$$\frac{\partial e_{iq}}{\partial b} = -1, \quad \text{for the bias } b$$

we will have $\left[\nabla e_{iq}{}^{2} \right] = -2 e_{iq} \mathbf{z}_q$

$$\mathbf{z}_q = \begin{bmatrix} p_{1q} \\ p_{2q} \\ \ldots \\ p_{Rq} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_q \\ 1 \end{bmatrix}$$

4th Apply the gradient method to minimize $F(\boldsymbol{\theta}_{\boldsymbol{i}})$, being k the iteration index

$$\boldsymbol{\theta}_i^{k+1} = \boldsymbol{\theta}_i^{k} - \alpha \nabla F(\boldsymbol{\theta}_i^{k})$$

in the present case

$$\boldsymbol{\theta}_i = \begin{bmatrix} \mathbf{w}_i \\ b \end{bmatrix} \qquad F(\boldsymbol{\theta}_i^{k}) = e_{iq}^{2}$$

$$\left[\nabla e_{iq}^{2}\right] = -2e_{iq}\mathbf{z}_q = -2e_{iq}\begin{bmatrix} \mathbf{p}_q \\ 1 \end{bmatrix}$$

resulting in

$$\mathbf{w}_i^{k+1} = \mathbf{w}_i^{k} + 2\alpha e_{iq}\mathbf{p}_q$$

$$b_i^{k+1} = b_i^{k} + 2\alpha e_{iq}$$

But

$$\mathbf{w}_i^{k+1} = \mathbf{w}_i^{\,k} + 2\alpha e_{iq}\mathbf{p}_q \Rightarrow (\mathbf{w}_i^T)^{k+1} = (\mathbf{w}_i^{\,T})^k + 2\alpha e_{iq}\mathbf{p}_q^T$$

For one layer of *S* neurons will come

$$\begin{bmatrix} \mathbf{w}_1^T \\ \dots \\ \mathbf{w}_S^T \end{bmatrix}^{k+1} = \begin{bmatrix} \mathbf{w}_1^T \\ \dots \\ \mathbf{w}_S^T \end{bmatrix}^{k} + 2\alpha \begin{bmatrix} e_{1q} \\ \dots \\ e_{sq} \end{bmatrix} \mathbf{p}_q^T \qquad \begin{bmatrix} b_1 \\ \dots \\ b_S \end{bmatrix}^{k+1} = \begin{bmatrix} b_1 \\ \dots \\ b_S \end{bmatrix}^{k} + 2\alpha \begin{bmatrix} e_{1q} \\ \dots \\ e_{sq} \end{bmatrix}$$

Or, more compact,

$$\mathbf{W}^{k+1} = \mathbf{W}^k + 2\alpha \mathbf{e}_q \mathbf{p}_q^{\ T}$$

$$\mathbf{b}^{k+1} = \mathbf{b}^k + 2\alpha \mathbf{e}_q$$

Taking into account that

$$\boldsymbol{\Theta} = \begin{bmatrix} \mathbf{W} & \mathbf{b} \end{bmatrix} \quad (\mathbf{z}_q)^T = \begin{bmatrix} (\mathbf{p}_q)^T & 1 \end{bmatrix}$$

a more compact form can be written,

$$\boldsymbol{\Theta}^{k+1} = \boldsymbol{\Theta}^{k} + 2\alpha \mathbf{e}_q \mathbf{z}_q^T$$

Remark: this LMSE algorithm is also known as the Widrow-Hoff rule (see more in Hagan).

- The LMSE iterative algorithm is an approximation of the gradient method, because the computed gradient in each iteration is an approximation of the true gradient.

- Its convergence depends on the **learning coefficient** α**.**

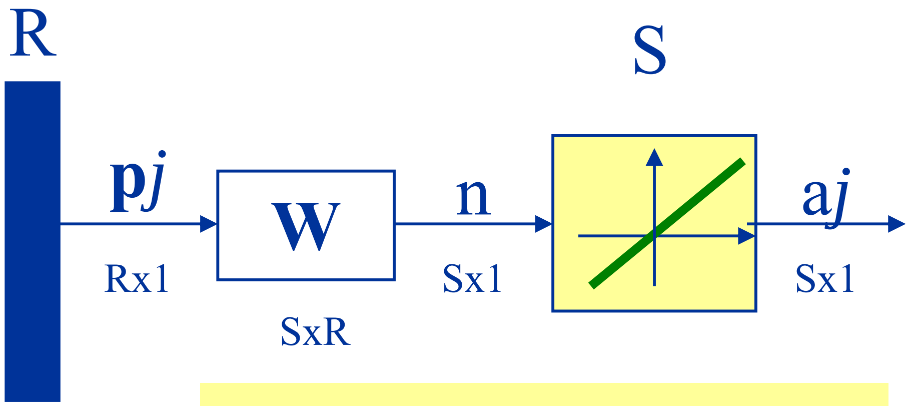
- If the successive input vectors are statistically independent, and if $\theta(k)$ and $\mathbf{z}(k)$ are statistically independent, it converges.

- The learning coefficient must verify

$$0 < \alpha < \frac{1}{\lambda_{\max}}$$

where $\lambda_{\max}$ is the maximum eigenvalue of the input correlation matrix $\mathbf{R}=\mathbf{Z}\mathbf{Z}^T$ (see more in *Hagan, 10-9*).

4.9.2. The particular case of the associative memory (one layer without bias)



$$\mathbf{a} = \mathbf{n} = purelin(\mathbf{W}\mathbf{p})$$

$$\mathbf{a}_j = \mathbf{W}\mathbf{p}_j$$

$$a_{ij} = \sum_{k=1}^{R} w_{ik} p_{kj}$$

Associative memory: learns Q prototype pairs of input-output vectors

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Giving an input prototype, the output is the correct one. Giving an input approximate to a prototype, the output will also be approximate to the corresponding output: a small change in the input will produce a small change in the output.

Supervised training: WP=A and we want A=T , so WP=T

If **R > Q** (more inputs than prototypes), **P** is rectangular, with more rows than columns. We have a system with more unknowns than equations.

If **P** has maximum characteristic (if its columns are linearly independent), the pseudo-inverse ca be used to find an **exact solution** for the system of equations.

$$\mathbf{W}\mathbf{P} = \mathbf{T} \Rightarrow \quad \mathbf{W} = \mathbf{T}\mathbf{P}^{+} = \mathbf{T}(\mathbf{P}^{T}\mathbf{P})^{-1}\mathbf{P}^{T}$$

$\mathbf{P}^{+}$ is the pseudo-inversa of Moore-Penrose.

.

If **R=Q** ,the number *Q* of prototypes is equal to the number *R* of network inputs, and if the prototypes are linearly independent, then **P** can be inverted.

Solving in order to $\mathbf{W}$, using $\mathbf{P}^{-1}$:

$$\mathbf{WP} = \mathbf{T} \Leftrightarrow \mathbf{WPP}^{-1} = \mathbf{TP}^{-1} \Leftrightarrow \mathbf{W}(\mathbf{PP}^{-1}) = \mathbf{TP}^{-1}$$

$$\Rightarrow \mathbf{W} = \mathbf{TP}^{-1}, \qquad \text{if } \mathbf{P} \text{ is invertible}$$

If **R < Q** (more prototypes than inputs), **P** is rectangular, with more columns than rows. We have a system with more equations than unknowns.

In general there is no exact solution. Only an approximated solution can be found, using the Penrose pseudo-inverse that minimizes the sum of the squared errors.

$$\mathbf{W}\,\mathbf{P} = \mathbf{T} \Rightarrow \qquad \boxed{\mathbf{W} = \mathbf{T}\mathbf{P}^{+} = \mathbf{T}\mathbf{P}^{T}(\mathbf{P}\mathbf{P}^{T})^{-1}}$$

Note that the pseudo-inverse does not have the same formula as in the previous case R>Q.

It gives the solution that minimizes the sum of the squared errors (see slides 211 and 213).

The formulae of the associative memory are a particular case of the ADALINE; here there is no bias, and as a consequence instead of the ADALINE Θ, one has the W of the associative memory.

The recursive version will be the same of ADALINE

$$\mathbf{W}^{k+1} = \mathbf{W}^{k} + 2\alpha \mathbf{e}_q \mathbf{p}_q{}^{T}$$

There is an historic algorithm, **the Hebb's rule**, that has a different form

$$\mathbf{W}^{k+1} = \mathbf{W}^{k} + \alpha \cdot \mathbf{t}_q \cdot \mathbf{p}_q^{T}, \ \mathrm{q} = 1, 2, \ldots, Q$$

The ADALINE rule comes from the mathematical development for the minimization of the squared error (LMSE). The Hebb's rule results from an empirical principle proposed by the neurobiologist Hebb in its book *The Organization of Behavior*, 1949.

The general Gradient Method

Iterative minimization of a general function g(x) with respect to x :

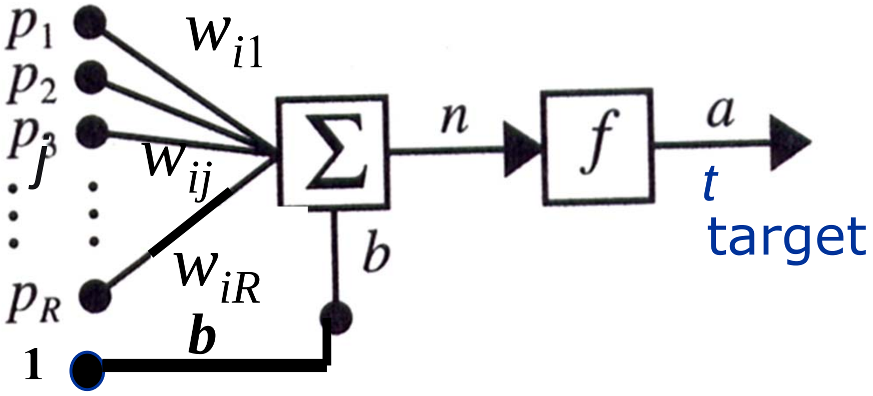
At iteration k,

$$x^{k+1} = x^k - \alpha \frac{\partial g}{\partial x}\bigg|_{x^k} = x^k - \alpha \nabla g(x)\Big|_{x^k}$$

α is a constant to be fixed by the user.

This is the **gradient method** and is the base of many ML learning algorithms.

4.10. One layer with any activation function



$$a = f(n)$$

$$e = t - a$$

$$F = e^2 \quad \text{to be minimized}$$

For the weight w_{ij} between neuron i and input p$_j$

$$\frac{\partial F}{\partial w_{ij}} = \nabla e^2 \Big|_{w_{ij}} = \frac{\partial F}{\partial e}\frac{\partial e}{\partial a}\frac{\partial a}{\partial n}\frac{\partial n}{\partial w_{ij}} =$$

$$= 2e.(\text{-}1).\dot{f}.p_j = -2e.\dot{f}.p_j$$

For the bias b

$$\frac{\partial F}{\partial b} = \nabla e^2 \Big|_{b} = \frac{\partial F}{\partial e}\frac{\partial e}{\partial a}\frac{\partial a}{\partial n}\frac{\partial n}{\partial b} =$$

$$= 2e.(\text{-}1).\dot{f}.1 = -2e.\dot{f}.1$$

for the input $\boldsymbol{p}_q$ $q=1, \;...., \; \boldsymbol{Q}$

$$\frac{\partial F}{\partial w_{ij}} = \nabla e^2 \Big|_{w_{ij}} = \frac{\partial F}{\partial e}\frac{\partial e}{\partial a}\frac{\partial a}{\partial n}\frac{\partial n}{\partial w_{ij}}$$

at iteration *k* :

$$\frac{\partial e_{iq}}{\partial w_{ij}{}^{k}} = \frac{\partial [t_{iq} - a_{iq}]}{\partial w_{ij}{}^{k}} = \frac{\partial}{\partial w_{ij}{}^{k}}[t_{iq} - f(n^k)]$$

$$= \frac{\partial}{\partial w_{ij}{}^{k}}\left[t_{iq} - f\left(\sum_{j=1}^{R} w_{ij}{}^{k} p_{jq} + b^k\right)\right] = \overbrace{-\frac{\partial f}{\partial n_q} \cdot \frac{\partial n_q^k}{\partial w_{ij}^k}}^{\text{chain rule}} \qquad j = 1, 2, \ldots R$$

$$= -\dot{f} \, .p_{jq} \qquad j = 1, 2, \ldots R$$

$$\frac{\partial e_{iq}}{\partial b} = -\dot{f} \, .1, \quad \text{for the bias } b$$

The gradient of the squared error, for all weights and bias, will come

$$\left[\nabla e_{iq}{}^{2}\right] = -2e_{iq} . \dot{f} . \mathbf{z}_q = \text{-}2e_{iq} . \dot{f} . \begin{bmatrix} \mathbf{p}_q \\ 1 \end{bmatrix}$$

and the update formula will be, for a layer of neurons

$$\boldsymbol{\Theta}^{k+1} = \boldsymbol{\Theta}^{k} + 2\alpha \mathbf{e}_q \,.\, \dot{f} \,.\mathbf{z}_q^T$$

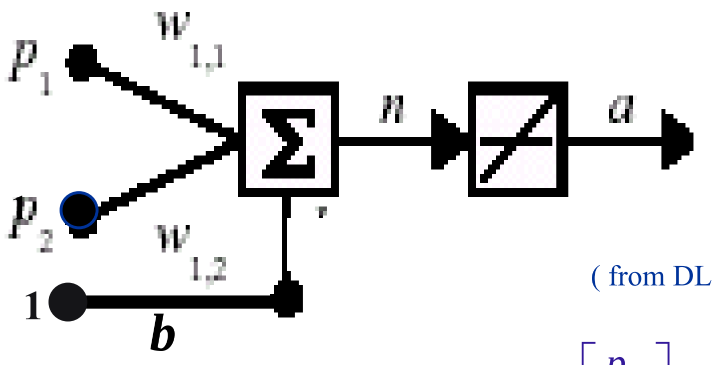
This is the general iterative gradient method for one layer, also known as the LMSE (least minimum squared error).

The Widraw-Hoff algorithm is the particular case of the gradient method when the activation function is linear, and so its derivative is one.

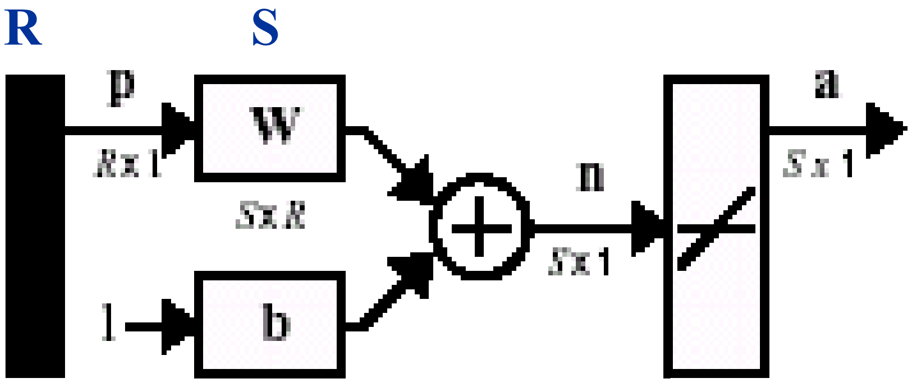
After passing through all the inputs (1.... Q) we say that an epoch of training is complete.
The process restarts with the actual parameters and again for the inputs (1 ... Q), the second epoch is ended.
And so on, until the convergence criteria is reached or a fixed maximum number of epochs is attained.

4.11. Synthesis of the learning techniques of a network with a single layer of linear neurons

Consider the unified notation Θ for the parameter matrix (having or not bias) and Z the input matrix (with or without bias). The similarity of the several learning methods becomes clear:



(from DL Toolbox UG)



$$\boldsymbol{\theta}_i^T = \begin{bmatrix} \mathbf{w}_i^T & b \end{bmatrix} \qquad \mathbf{z}_j = \begin{bmatrix} \mathbf{p}_j \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1j} \\ p_{2j} \\ \dots \\ p_{Rj} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{W} & \mathbf{b} \end{bmatrix} = \Theta \qquad \begin{bmatrix} \mathbf{P} \\ 1^T \end{bmatrix} = \mathbf{Z}$$