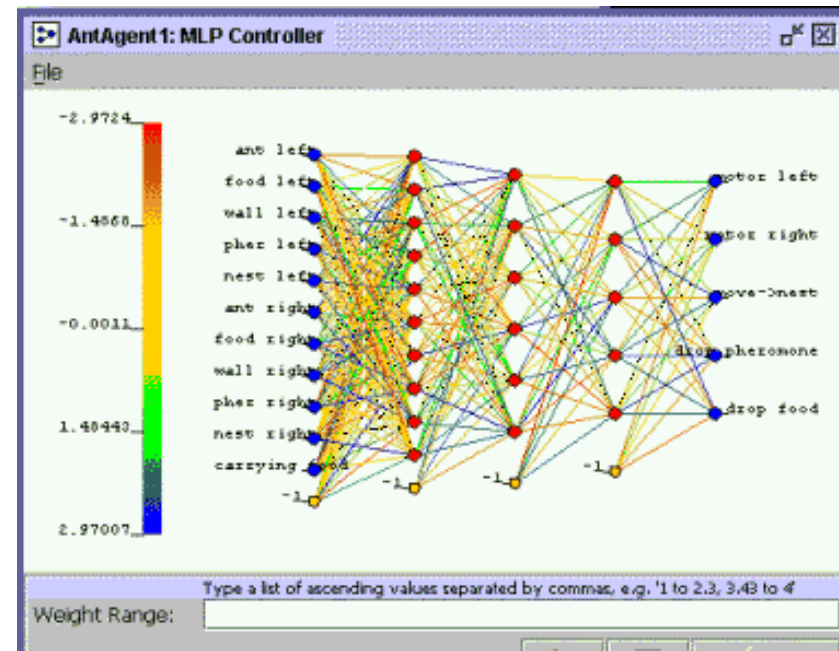


Redes neurais

Francisco Javier Roper Peláez

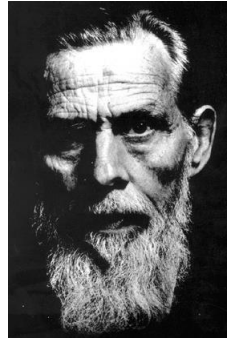


- Bibliografia :

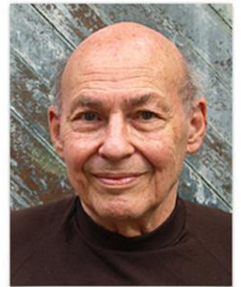
- Redes Neurais Artificiais. Teoria e Aplicações. Antônio de Pádua Braga. André Ponce de Leon F. de Carvalho. Teresa Bernarda Ludemir. Editora: LTC. 2007
- Machine Learning. An Algorithmic Perspective. Stephen Marsland. CRC Press. 2009.
- Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Rumelhart & McClelland. Cambridge MIT Press. 1986

História das redes neurais

- O neurônio de McCulloch-Pitts (1943)
- O perceptron de Rosemblatt (1957)



- Marvin Minski e o desafio da ou-exclusiva (1969)
- Os anos de silencio (Amari, Grossberg...)
- O livro PDP (1985)





Sun ichi Amar



Geoffrey Hinton



Stephen Grossberg



Teuvo Kohonen



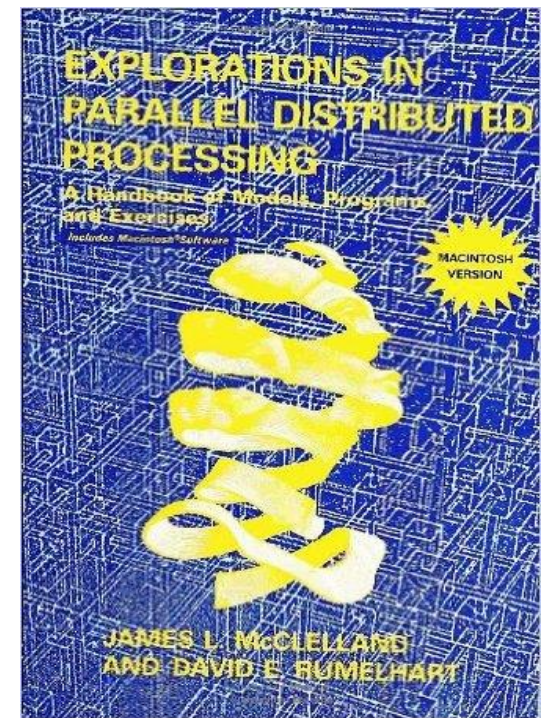
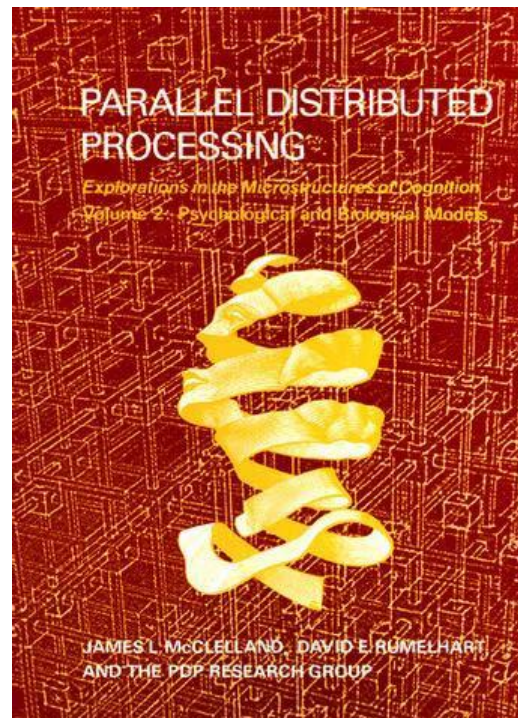
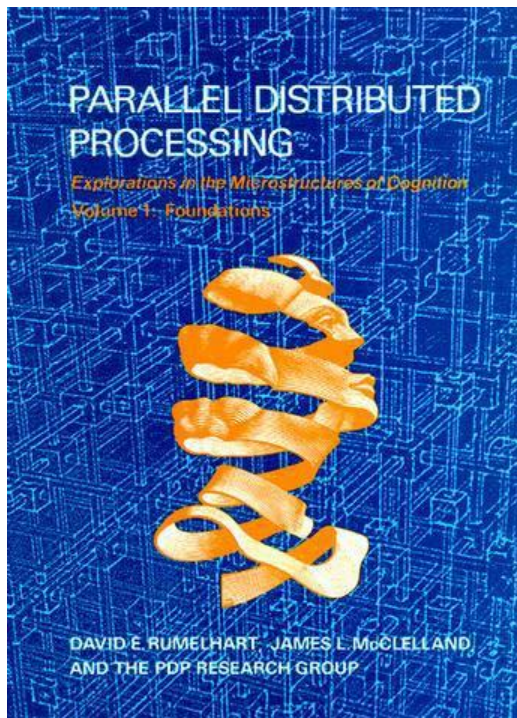
John Hopfield



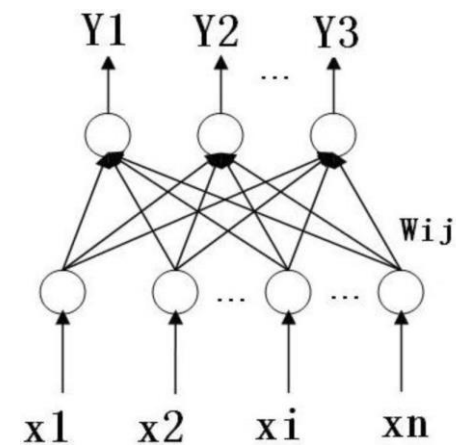
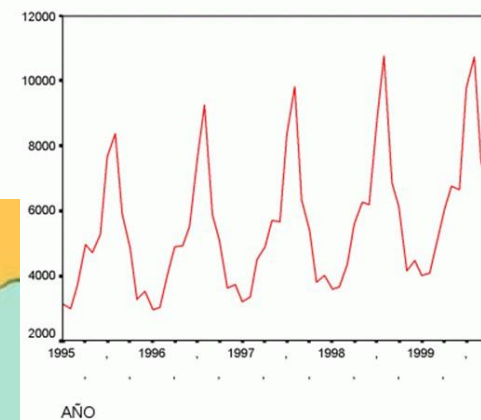
Kunihiko Fukushima

Parallel Distributed Processing (1985)

Explorations in Parallel Distributed Processing (1989)

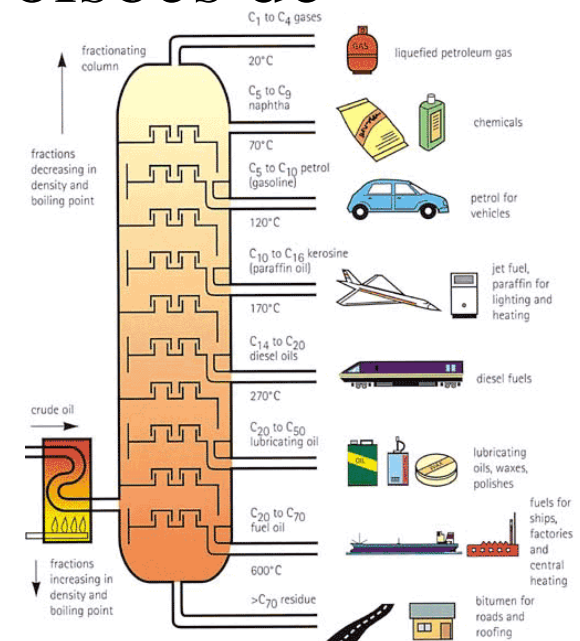
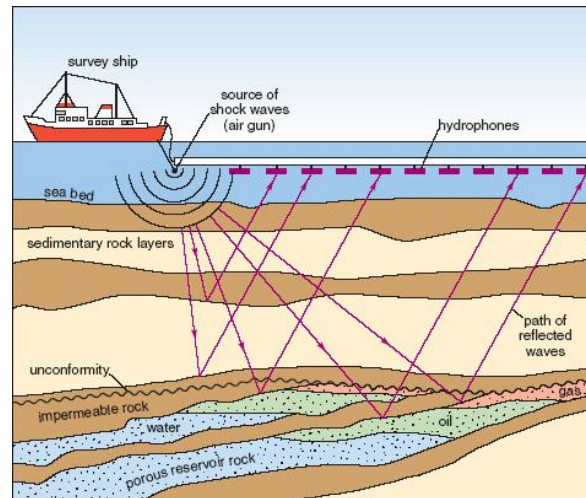


<https://web.stanford.edu/group/pdplab/pdphandbook/handbook.pdf>



Aplicações das redes neurais em empresa petrolífera

- Previsão de vendas.
- Interpretação do eco de explosão controlada para identificação de bolsões de petróleo
- Controle de processos (destilação fraccionada).

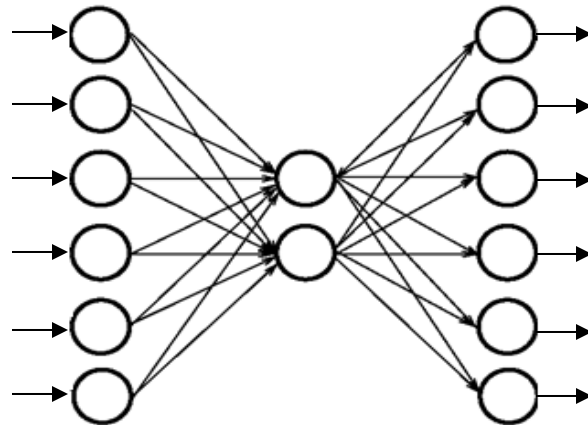


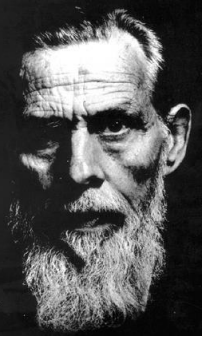
...dados, dados, dados: não posso fazer
tijolos sem barro.

Sherlock Holmes (filme, 2009)

Manutenção preventiva de máquinas ATM

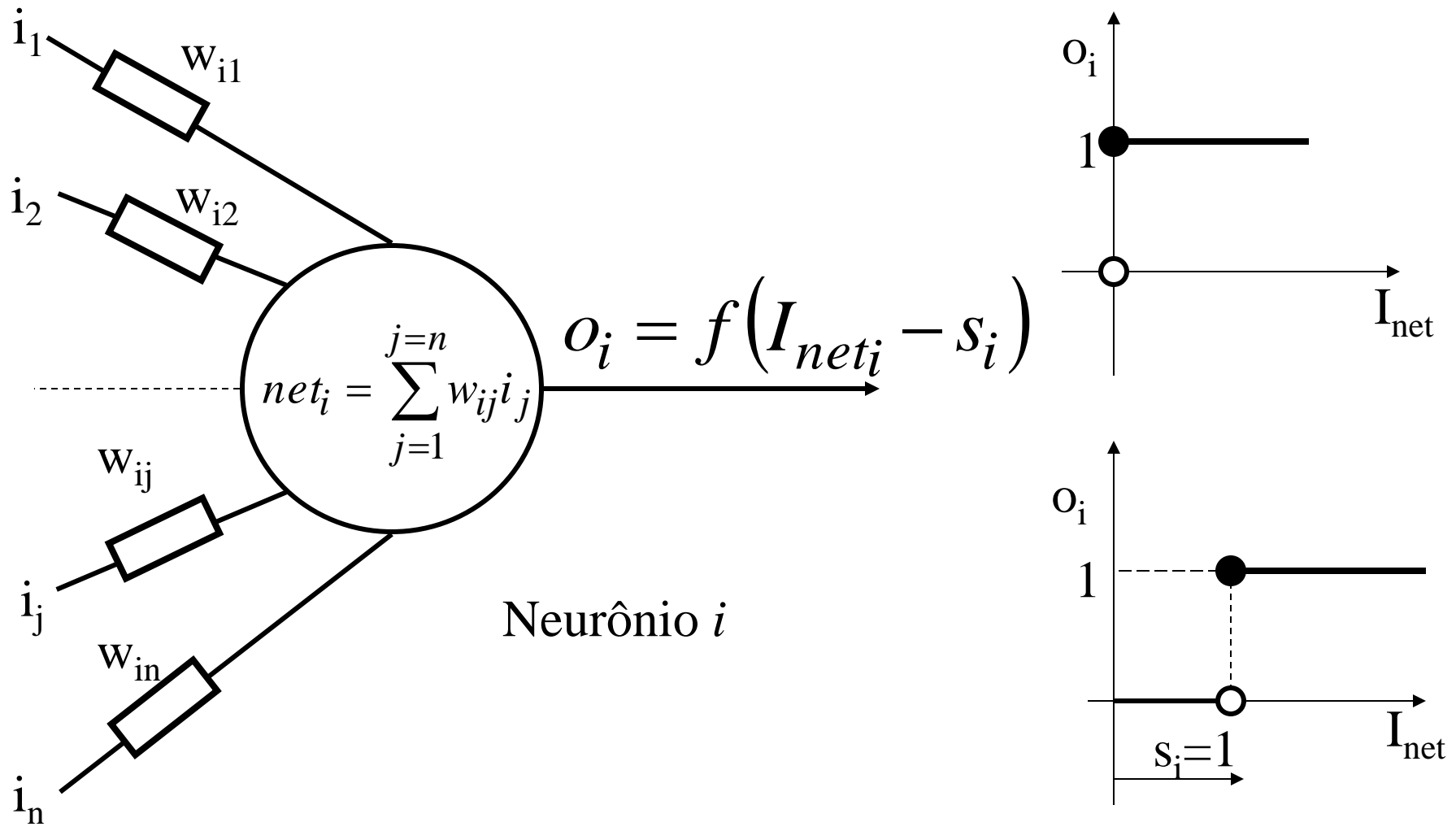
- **PELÁEZ, F. J. R. ; AGUIAR, M. A. ; DESTRO, R. C. ; KOVÁCS, Z. L. ; SIMÕES, M. G. .** Predictive Maintenance Oriented Neural Network System(PREMON). In: IECON 2001, 2001, Denver (Colorado). Proceedings of the IECON 2001.

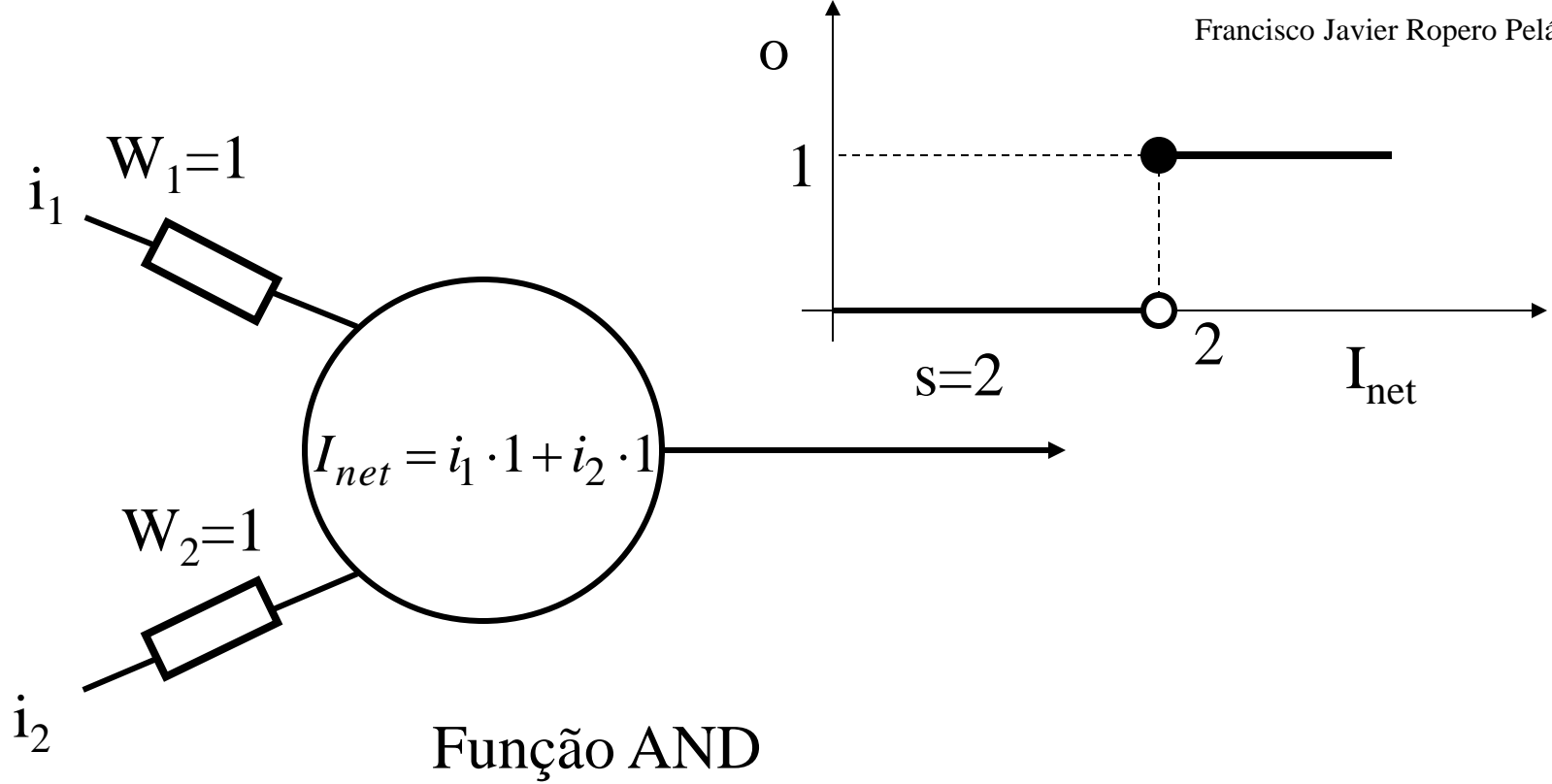




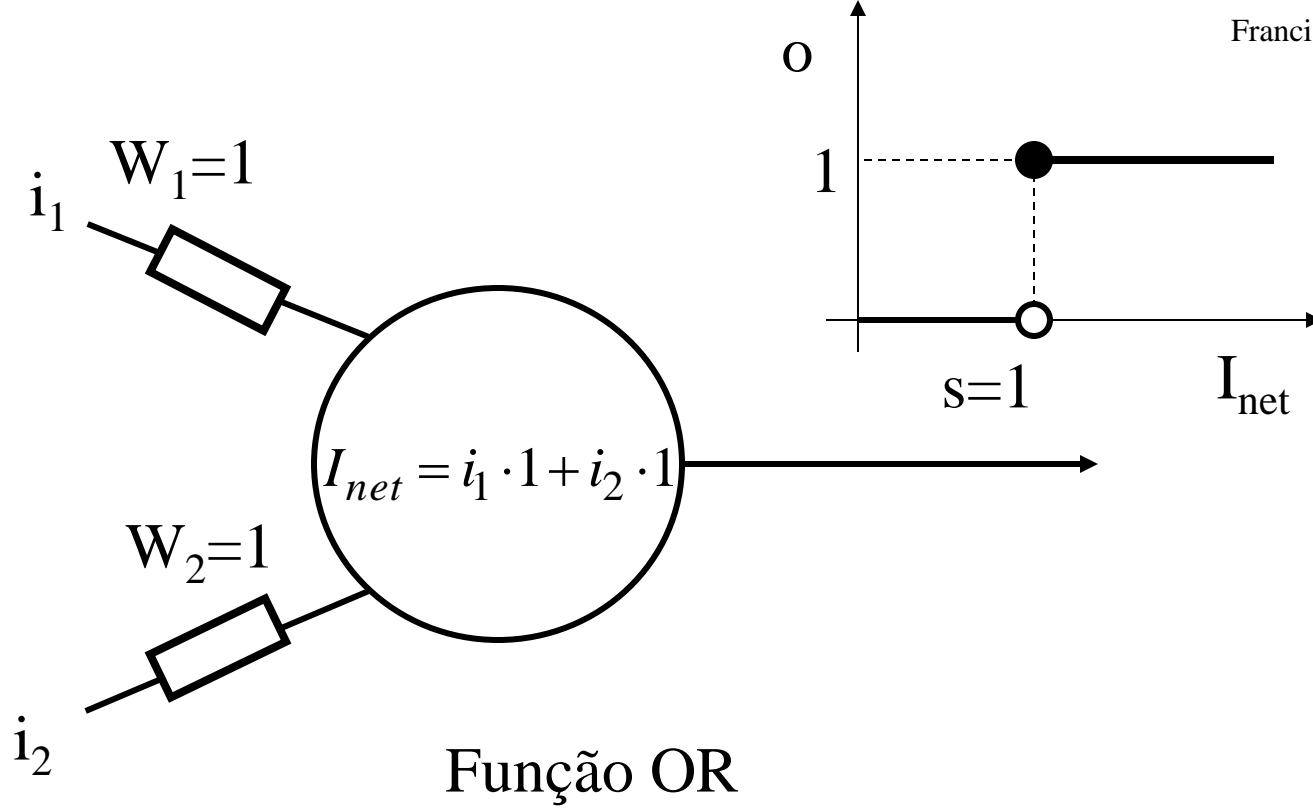
O neurônio de McCulloch- Pitts

“Um neurônio pode realizar qualquer função aritmética ou lógica”

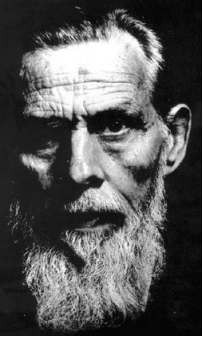




i_1	i_2	net	$f(\text{net}-2)$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

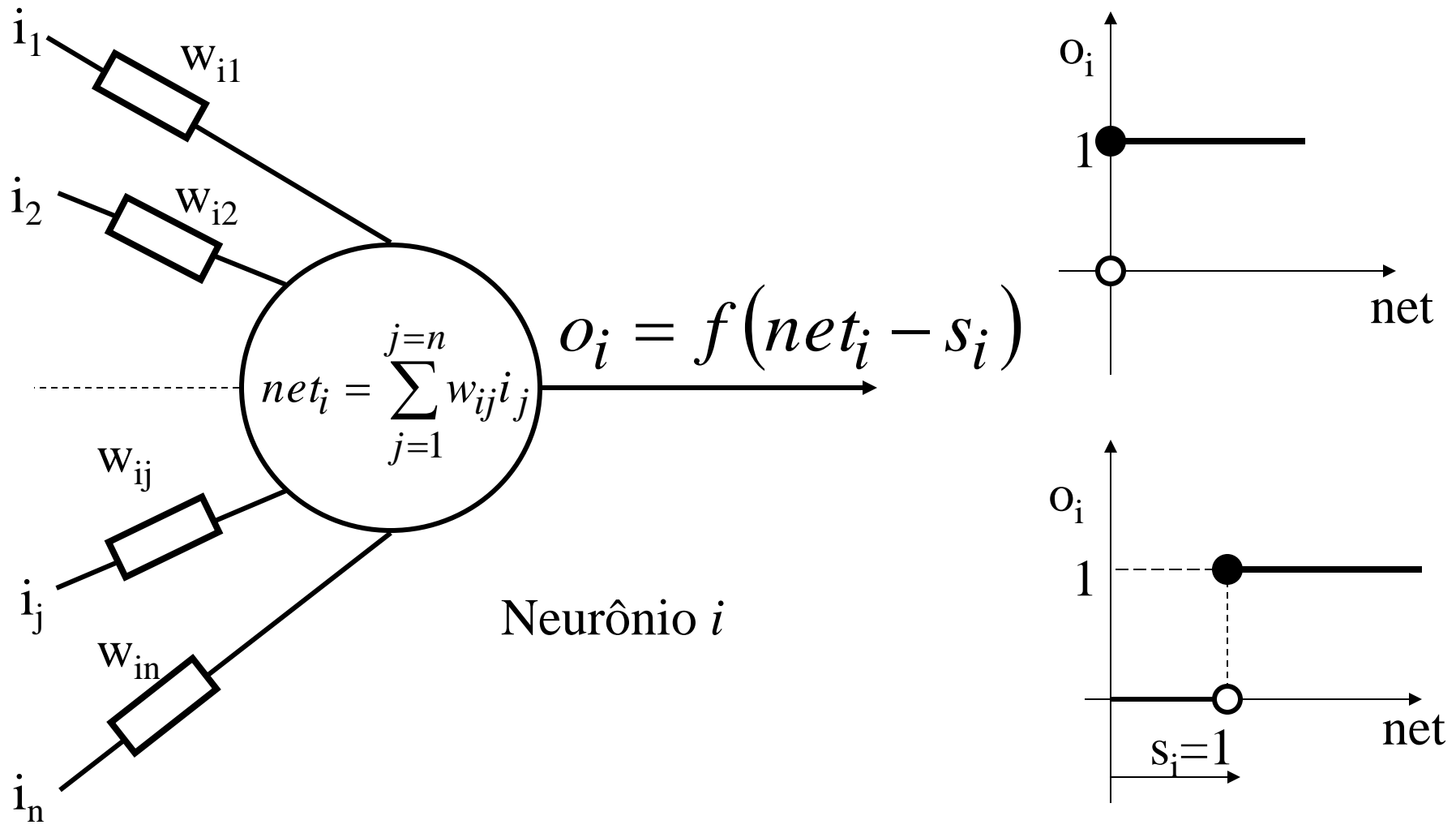


i_1	i_2	net	$f(\text{net}-1)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1

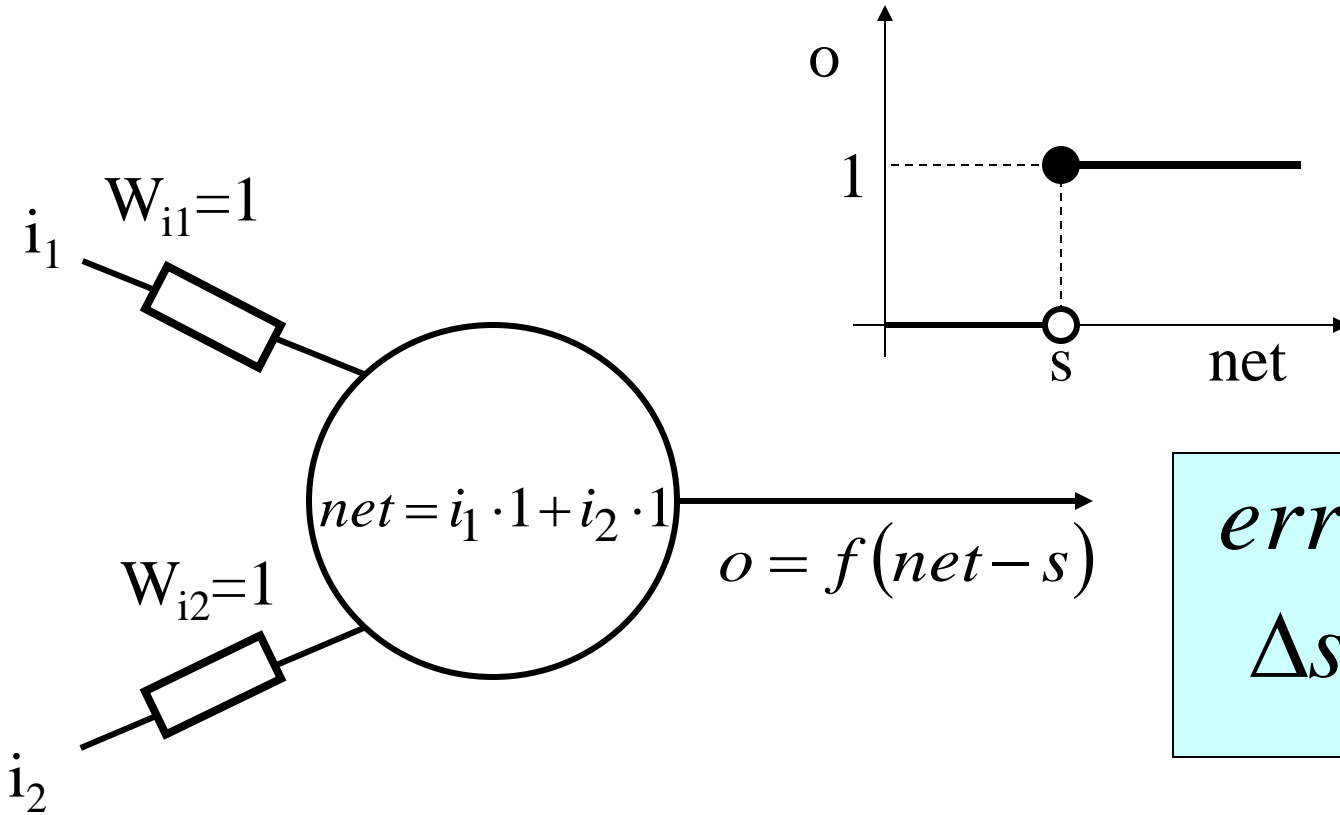


O neurônio de McCulloch- Pitts

“Um neurônio pode realizar qualquer função aritmética ou lógica”

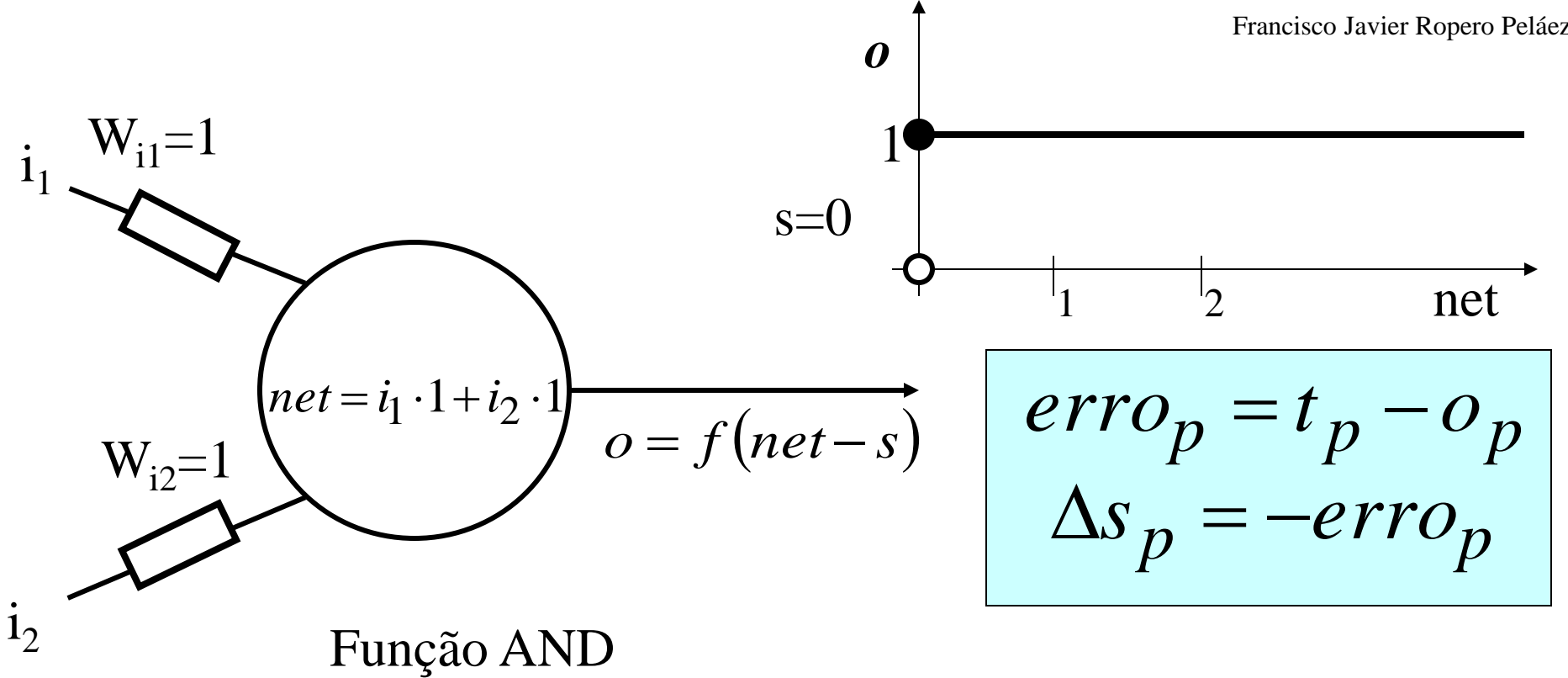


Ajustando o deslocamento da função degrau no neurônio de McCulloch

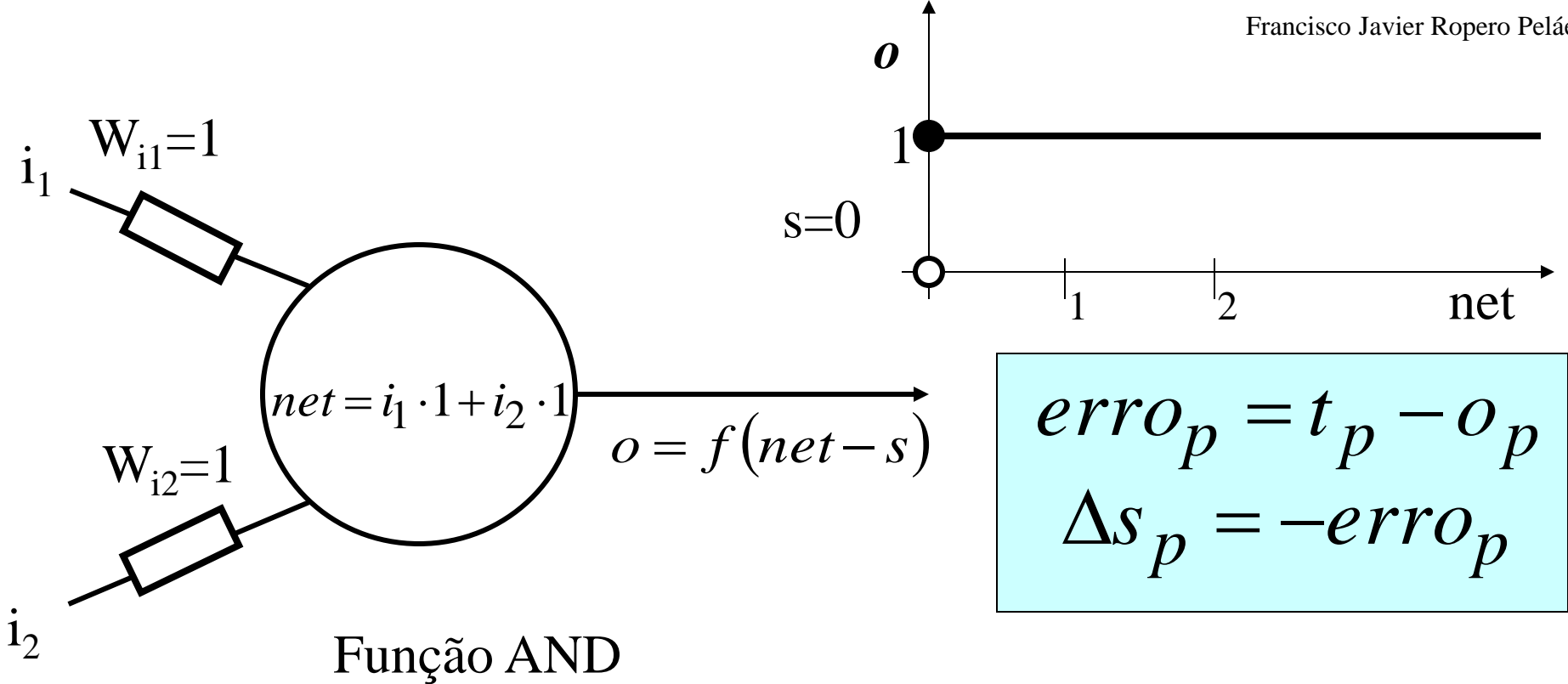


$$error_p = t_p - o_p$$

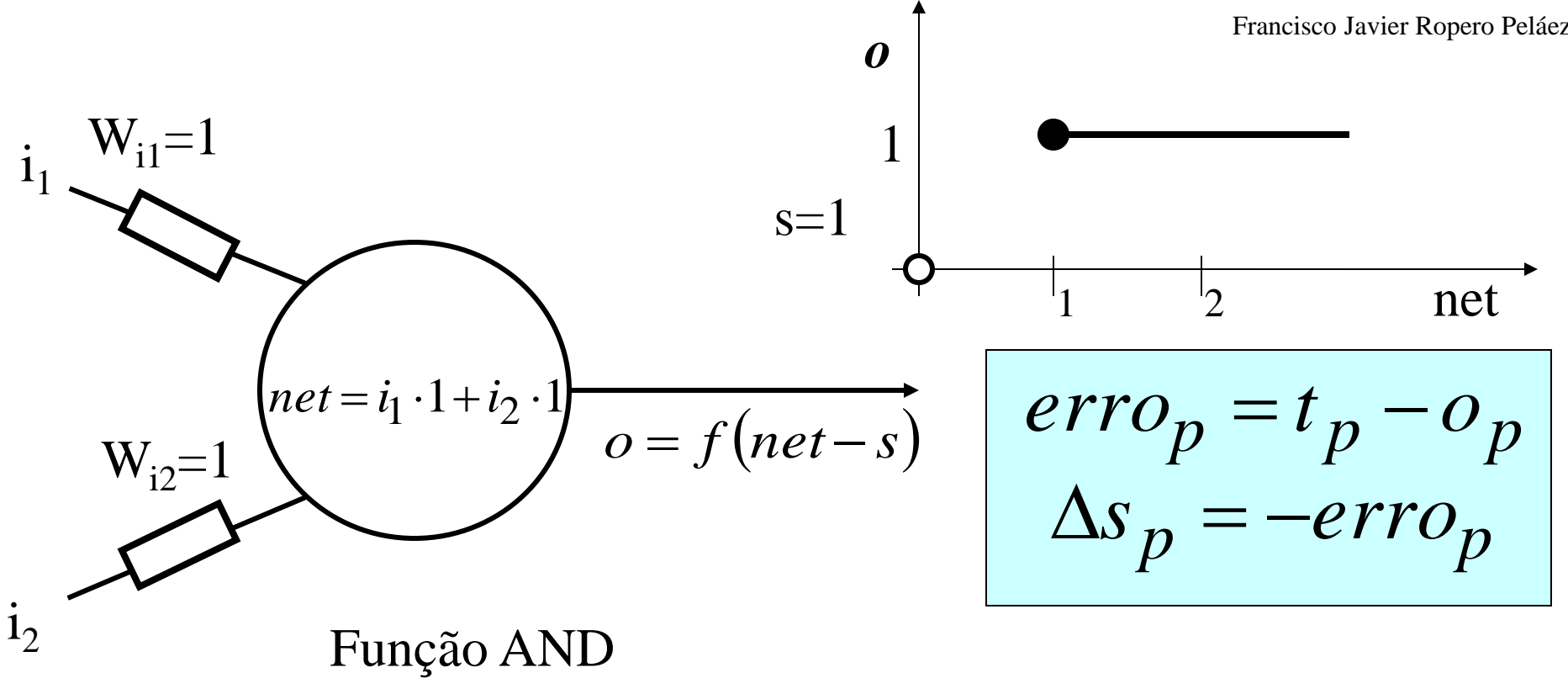
$$\Delta s_p = -error_p$$



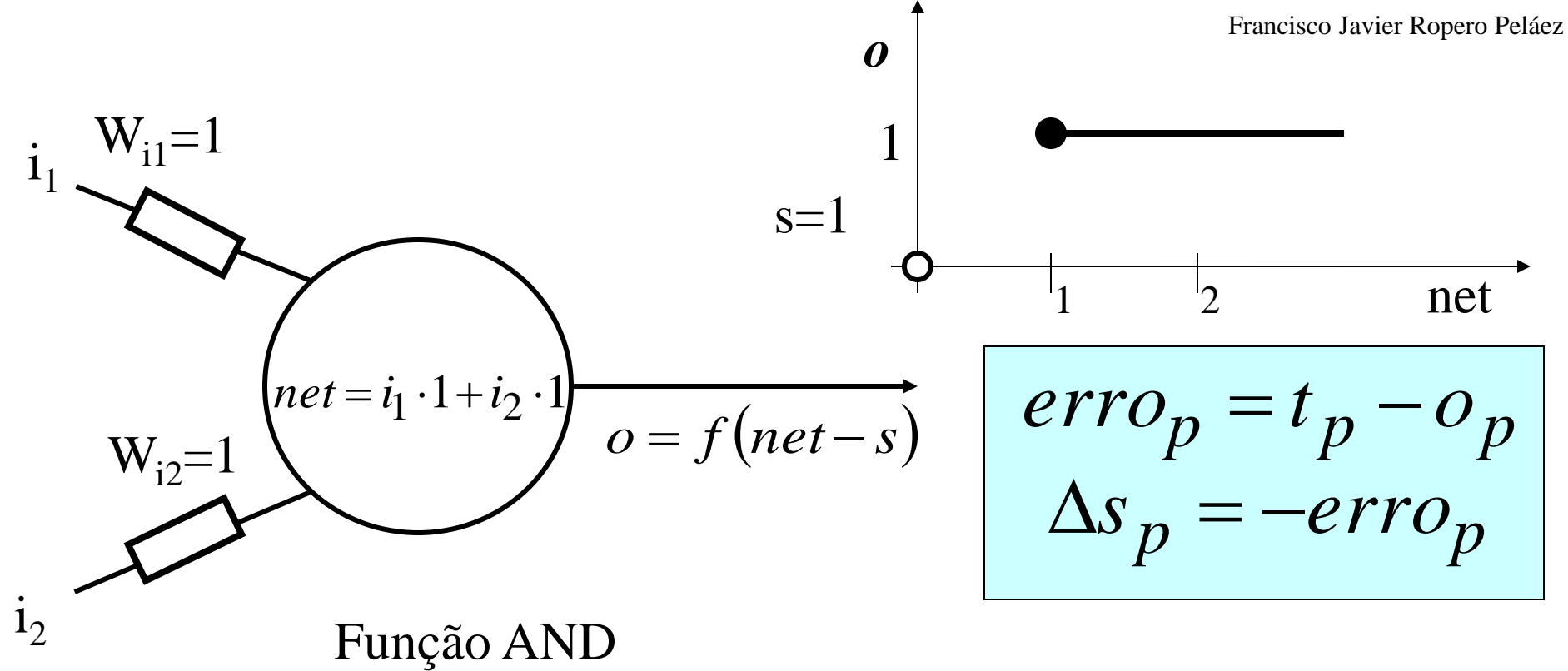
p	i_1	i_2	t (target)	net	s	$o=f(net-s)$	erro	Δs
1	0	0	0	0	0			
2	0	1	0					
3	1	0	0					
4	1	1	1					



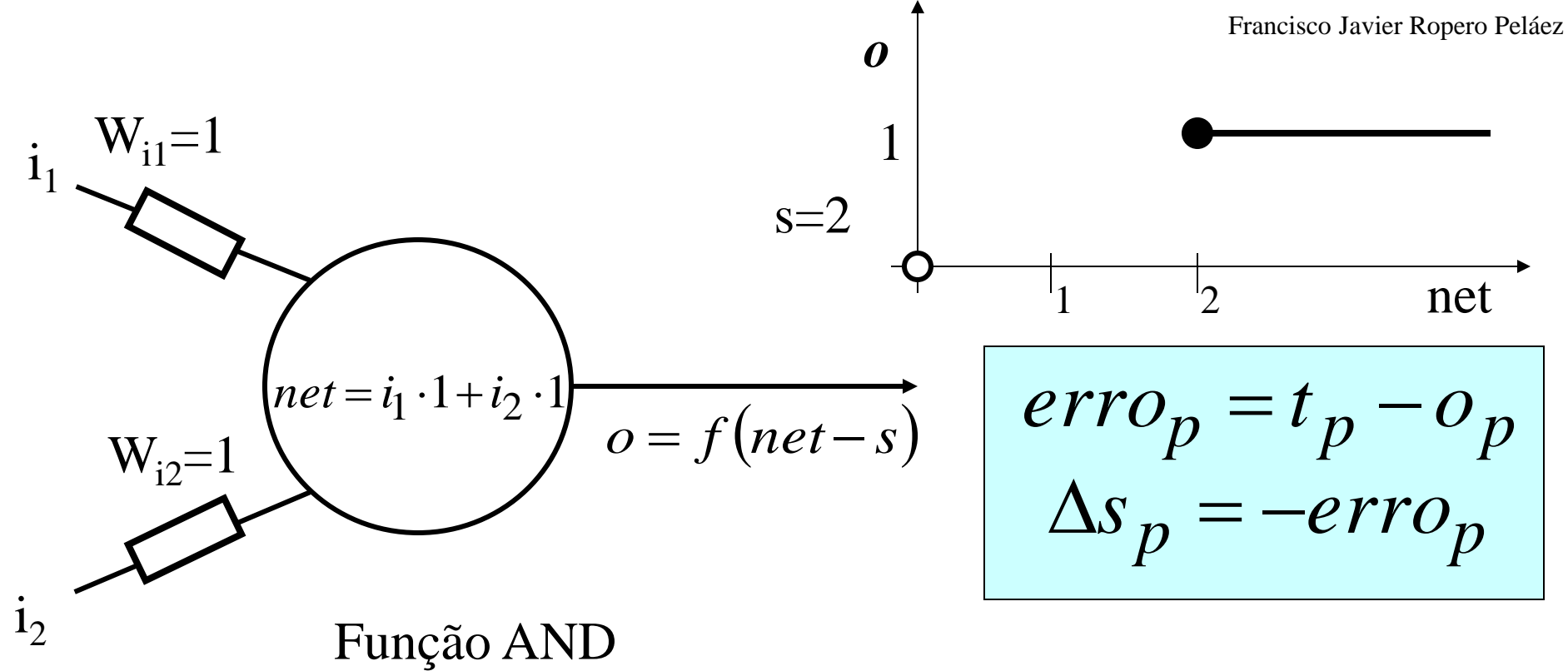
p	i_1	i_2	t (target)	net	s	$o=f(net-s)$	erro	Δs
1	0	0	0	0	0	1	-1	1
2	0	1	0					
3	1	0	0					
4	1	1	1					



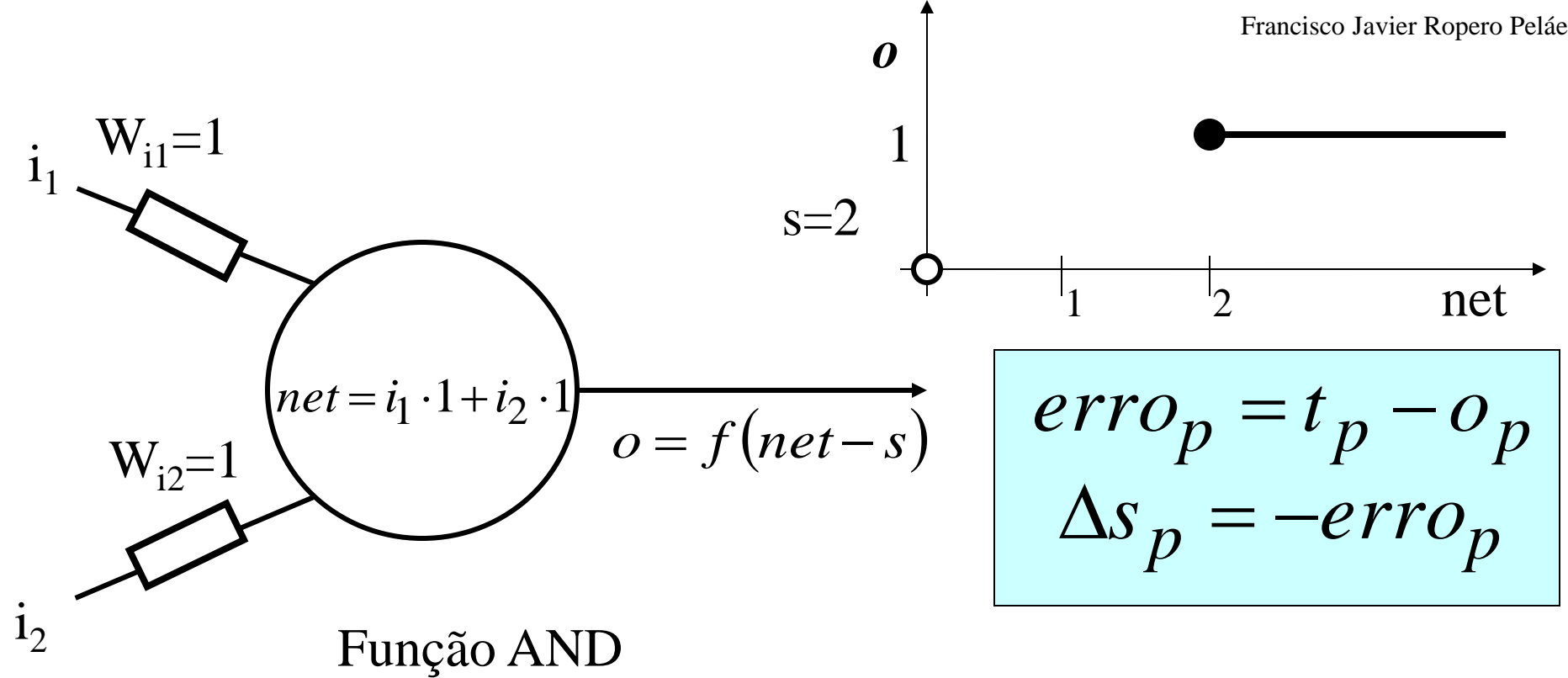
p	i_1	i_2	t (target)	net	s	$o=f(net-s)$	erro	Δs
1	0	0	0	0	0	1	-1	1
2	0	1	0	1	1			
3	1	0	0					
4	1	1	1					



p	i_1	i_2	t (target)	net	s	$o=f(net-s)$	erro	Δs
1	0	0	0	0	0	1	-1	1
2	0	1	0	1	1	1	-1	1
3	1	0	0					
4	1	1	1					

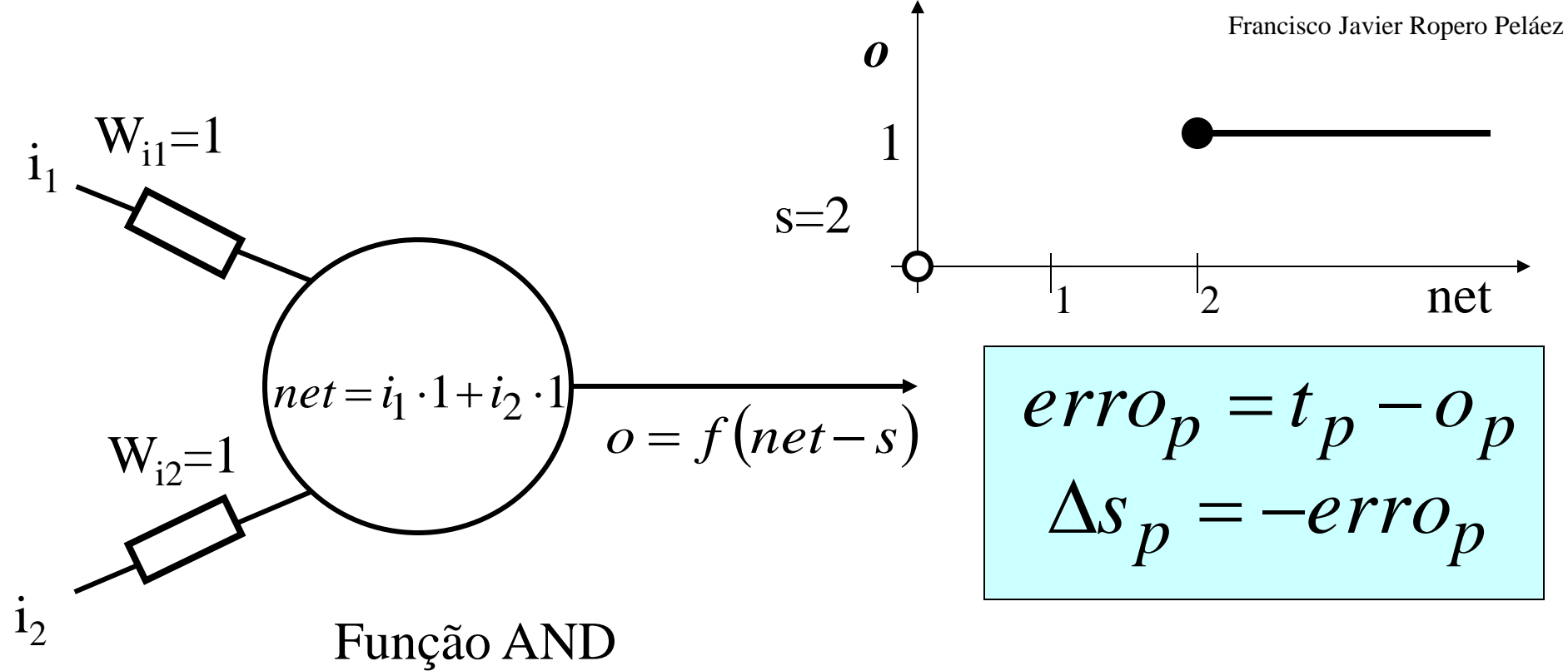


p	i_1	i_2	t (target)	net	s	$o=f(net-s)$	erro	Δs
1	0	0	0	0	0	1	-1	1
2	0	1	0	1	1	1	-1	1
3	1	0	0	1	2			
4	1	1	1					

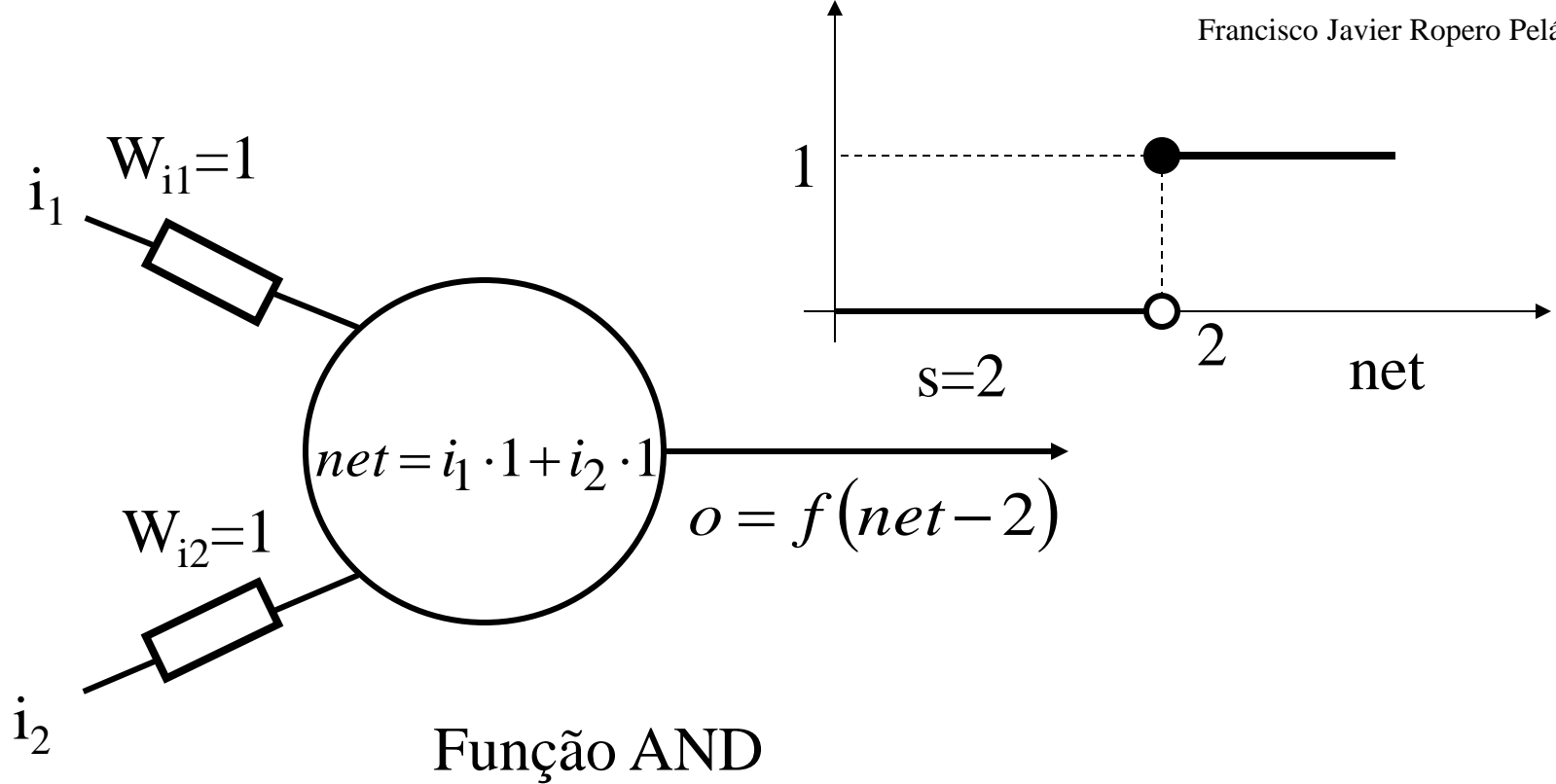


$$erro_p = t_p - o_p$$
$$\Delta s_p = -erro_p$$

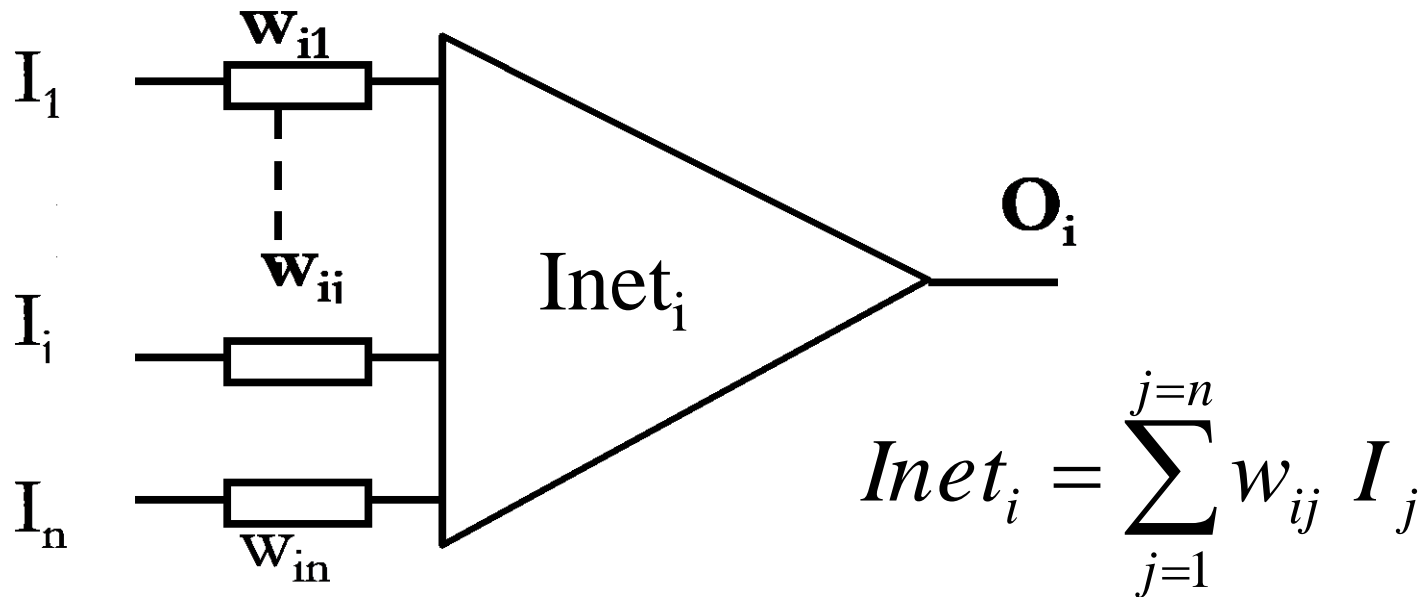
p	i_1	i_2	t (target)	net	s	$o=f(net-s)$	erro	Δs
1	0	0	0	0	0	1	-1	1
2	0	1	0	1	1	1	-1	1
3	1	0	0	1	2	0	0	0
4	1	1	1					



p	i_1	i_2	t (target)	net	s	$o=f(net-s)$	erro	Δs
1	0	0	0	0	0	1	-1	1
2	0	1	0	1	1	1	-1	1
3	1	0	0	1	2	0	0	0
4	1	1	1	2	2	1	0	0



i_1	i_2	t	o
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

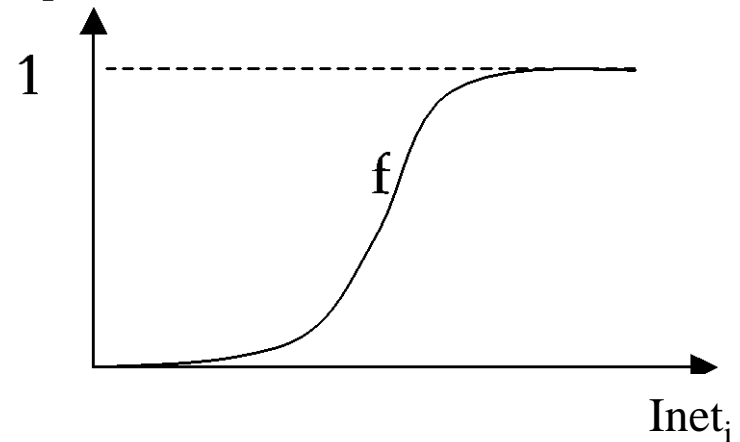


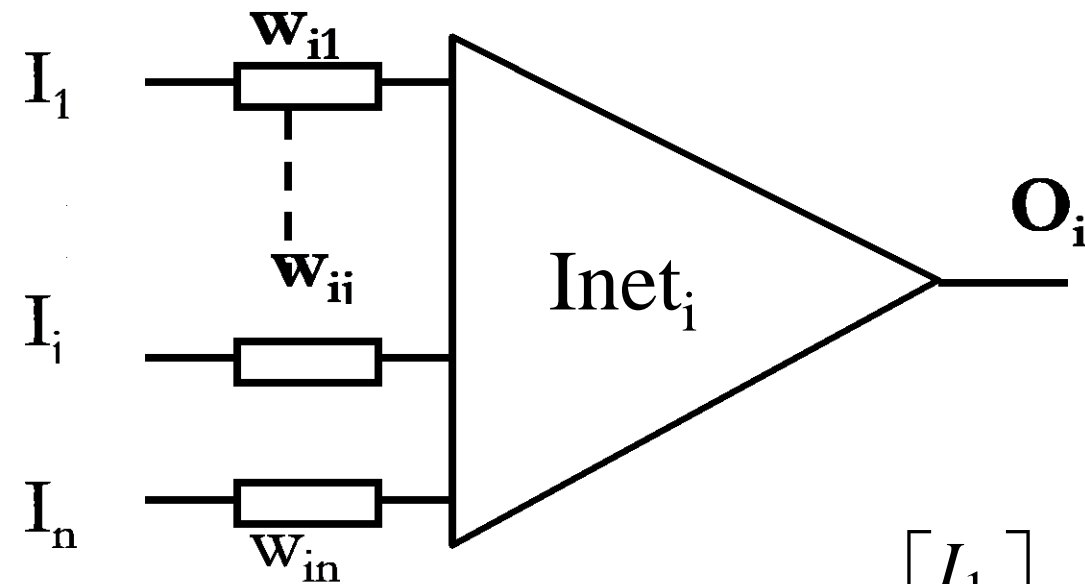
Em cada w_{ij} , o primeiro índice, i , significa o neurônio do qual falamos.

Ex: w_{13} significa o peso da entrada que vai ao neurônio 1 vinda do neurônio 3

Em ocasiões é possível omitir o primeiro índice

O output do neurônio i , O_i , é calculado aplicando a $Inet_i$ uma função não linear $f()$ como a do desenho, $O_i = f(Inet_i)$



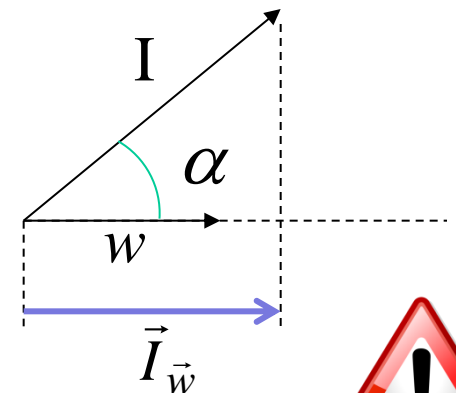


$$Inet_i = \sum_{j=1}^{j=n} w_{ij} I_j$$

$$Inet_i = \begin{bmatrix} w_{i1} & w_{i2} & \dots & w_{ij} & \dots & w_{in} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_j \\ \vdots \\ I_n \end{bmatrix} = \vec{I} \cdot \vec{w}$$

Se w for um vetor unitário, o produto Iw seria a projeção de I sobre w , ou seja:

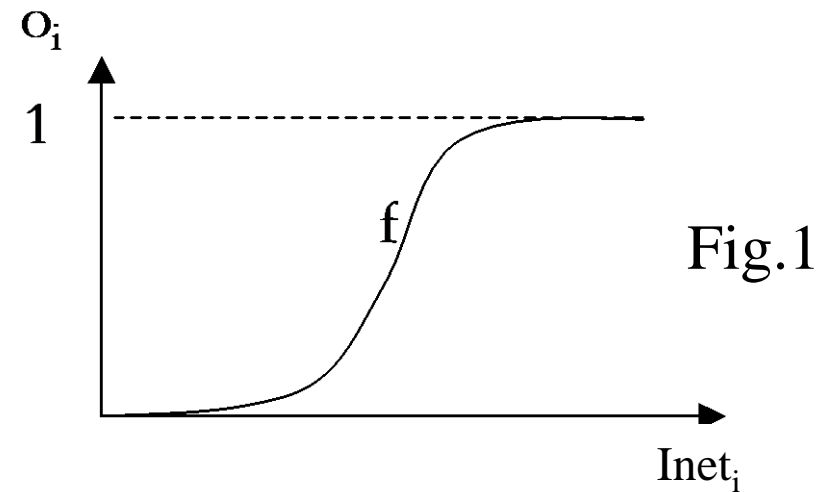
$$\begin{aligned} Inet_i &= \vec{I} \cdot \vec{w} = \\ &= \|\vec{I}\| \|\vec{w}\| \cos \alpha = \vec{I}_{\vec{w}} \end{aligned}$$



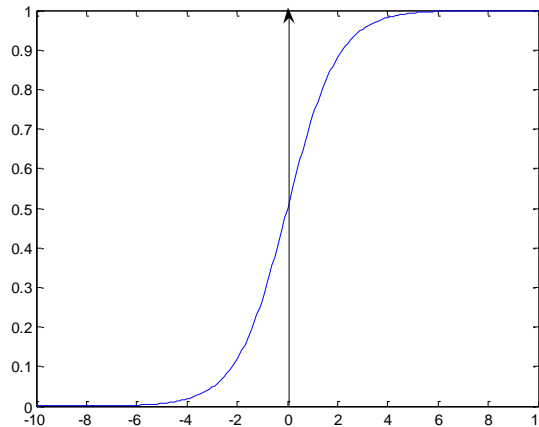
Se o w for uma componente principal teríamos a projeção sobre a componente principal



O output do neurônio i , O_i , é calculado aplicando a $Inet_i$ uma função não linear $f()$ como a do desenho, $O_i=f(Inet_i)$.



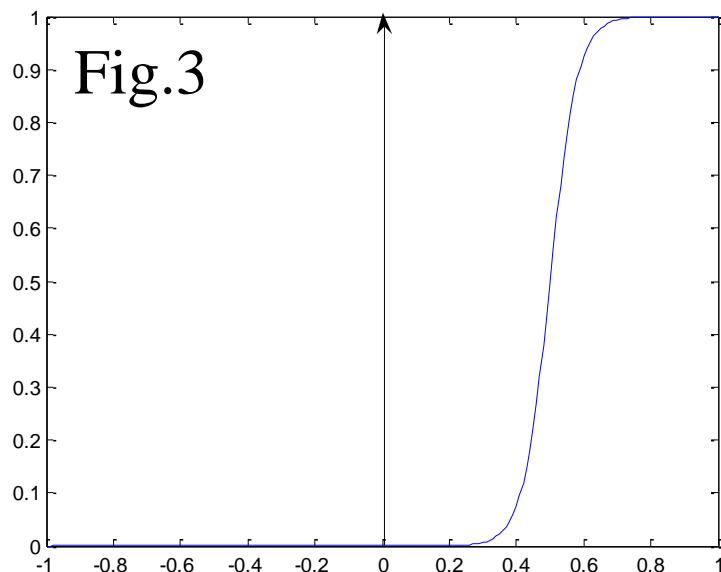
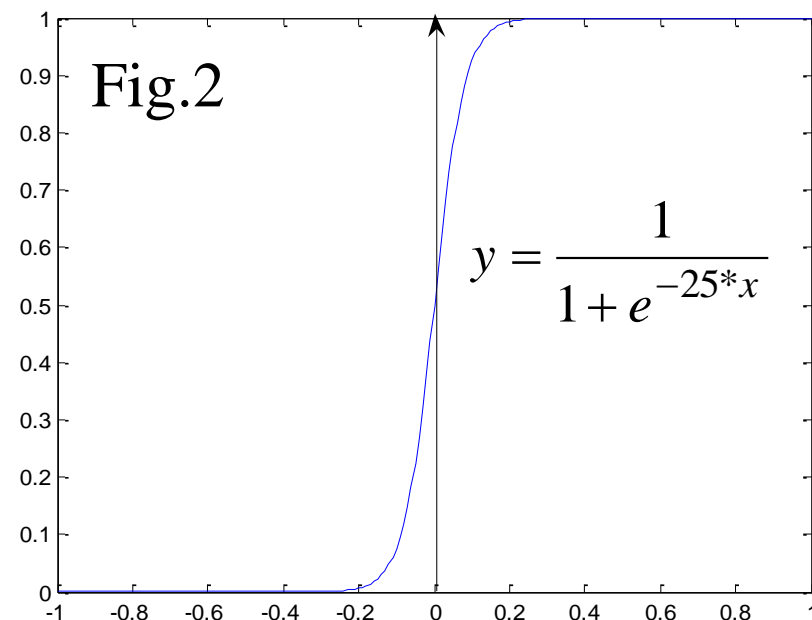
Para conseguir uma função não linear como a da figura 1 acima partiremos de uma função sigmóide típica.



$$y = \frac{1}{1 + e^{-x}}$$

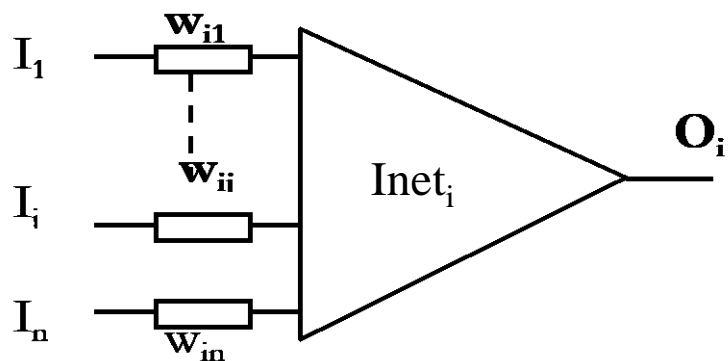
Para transformar a sigmóide numa sigmóide como a da fig .1 comprimimos a função multiplicando o x por um coeficiente maior que 1 (por exemplo, 25) obtendo a fig. 2.

Um coeficiente muito alto faria a função sigmoide parecer com uma função degrau.



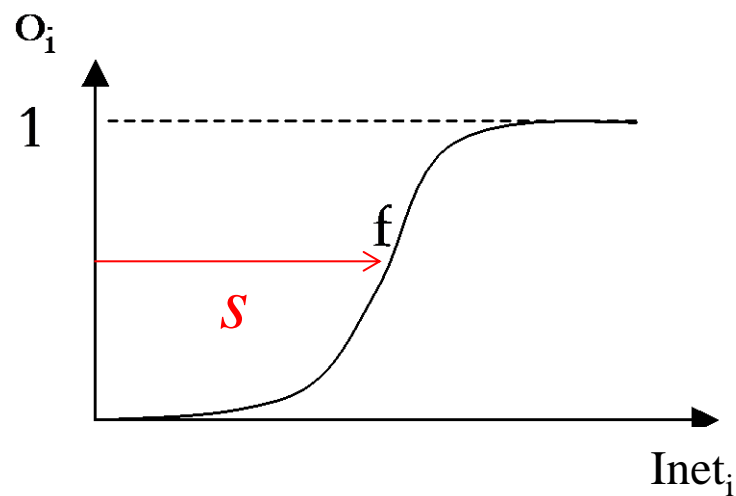
Finalmente deslocamos a curva 0.5 unidades para a direita (Fig. 3) subtraindo do x o valor do deslocamento, shift= 0.5, e obtemos uma curva parecida à da Fig.1.

$$y = \frac{1}{1 + e^{-25*(x-0.5)}}$$



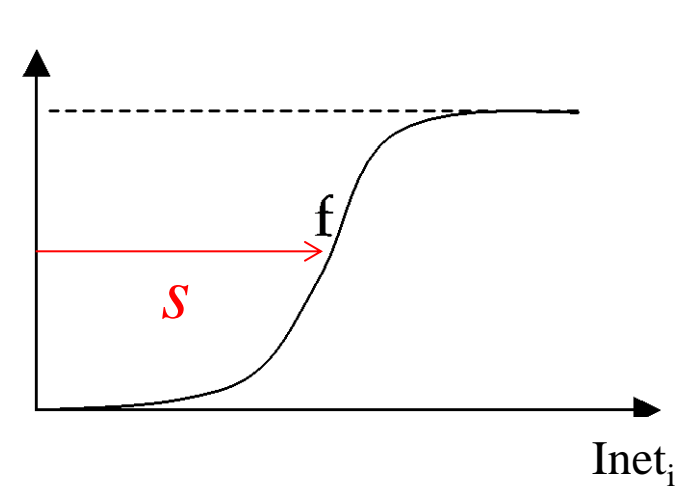
$$Inet_i = \sum_{j=1}^{j=n} w_{ij} I_j$$

$$O_i = \frac{1}{1 + e^{-25*(Inet_i - s)}}$$



$$O_i = \frac{1}{1 + e^{-25*(w_{i1}I_1 + w_{i2}I_2 + \dots + w_{in}I_n - s)}}$$

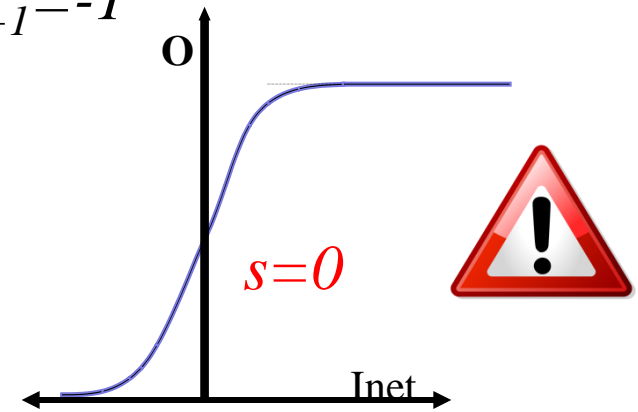
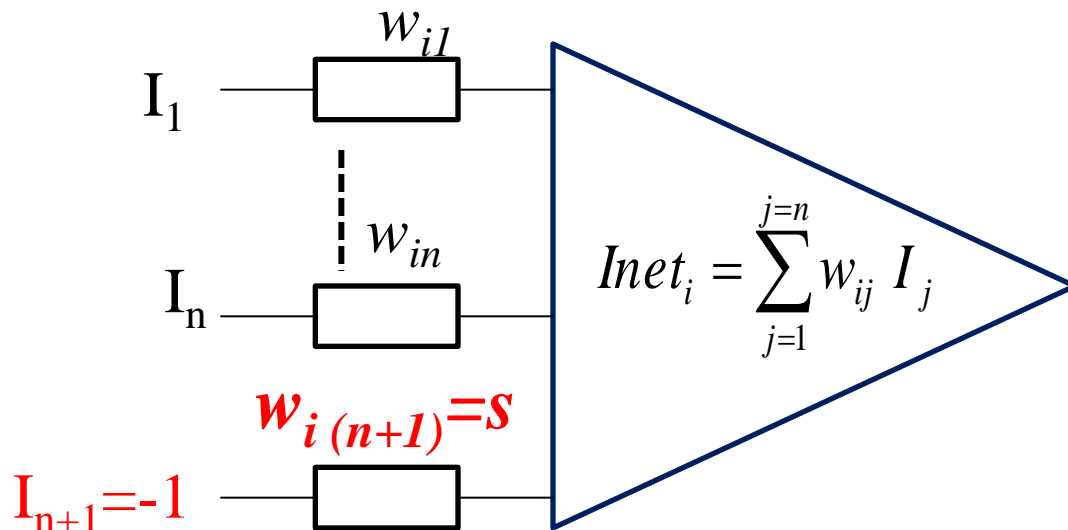
$$O_i = \frac{1}{1 + e^{-25 * (w_{i1}I_1 + w_{i2}I_2 + \dots + w_{in}I_n - s)}} - 1$$

$$Inet_i = \sum_{j=1}^{j=n} w_{ij} I_j - s = \sum_{j=1}^{j=n} w_{ij} I_j + s(-1) =$$


$$= \left[\sum_{j=1}^{j=n} w_{ij} I_j \right] + w_{i(n+1)} I_{n+1} = \sum_{j=1}^{j=n+1} w_{ij} I_j$$

Onde $w_{i(n+1)} = s$
e $I_{n+1} = -1$

Nas redes neurais artificiais



$$O_i = \frac{1}{1 + e^{-25 * (\sum_{i=1}^{n+1} w_{i1} I_1 \cancel{- s})}}$$



Aprendizado de Hebb

- “Quando um axônio de uma célula A está perto suficiente de uma célula B, como para excita-la, e participa repetida o persistentemente no seu disparo, acontece algum processo de crescimento ou mudança metabólica, em uma o em ambas células, de tal modo que a eficiência de A disparando sobre B aumenta...”

Modelos artificiais de plasticidade sináptica

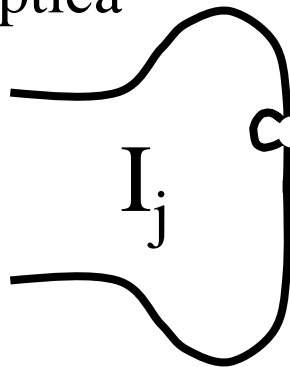
Fator de aprendizado



1. Modelo de Hebb

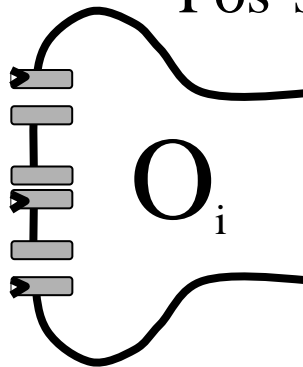
$$\Delta w_{ij} = \varepsilon O_i I_j$$

Membrana
Pré-sináptica



W

Membrana
Pós-sináptica



$\varepsilon = 0,1$

w_{ij}	I_j	O_i	Δw_{ij}
0	1	1	

Modelos artificiais de plasticidade sináptica

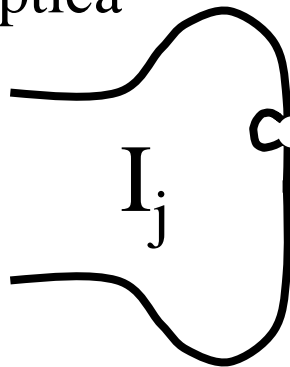
Fator de aprendizado



1. Modelo de Hebb

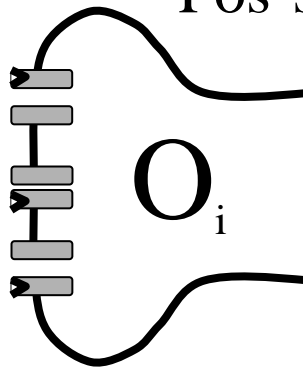
$$\Delta w_{ij} = \varepsilon O_i I_j$$

Membrana
Pré-sináptica



W

Membrana
Pós-sináptica



$\varepsilon = 0,1$

w_{ij}	I_j	O_i	Δw_{ij}
0	1	1	0,1
0,1			

Modelos artificiais de plasticidade sináptica

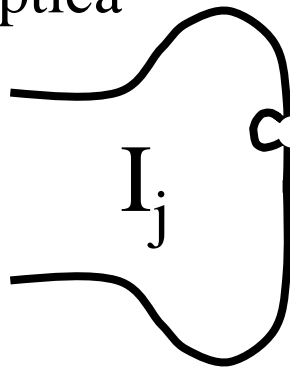
Fator de aprendizado



1. Modelo de Hebb

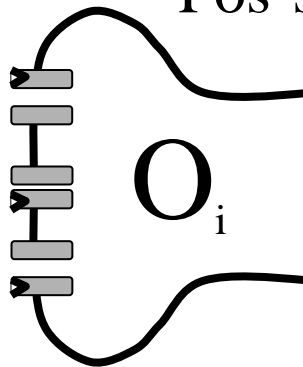
$$\Delta w_{ij} = \varepsilon O_i I_j$$

Membrana
Pré-sináptica



W

Membrana
Pós-sináptica



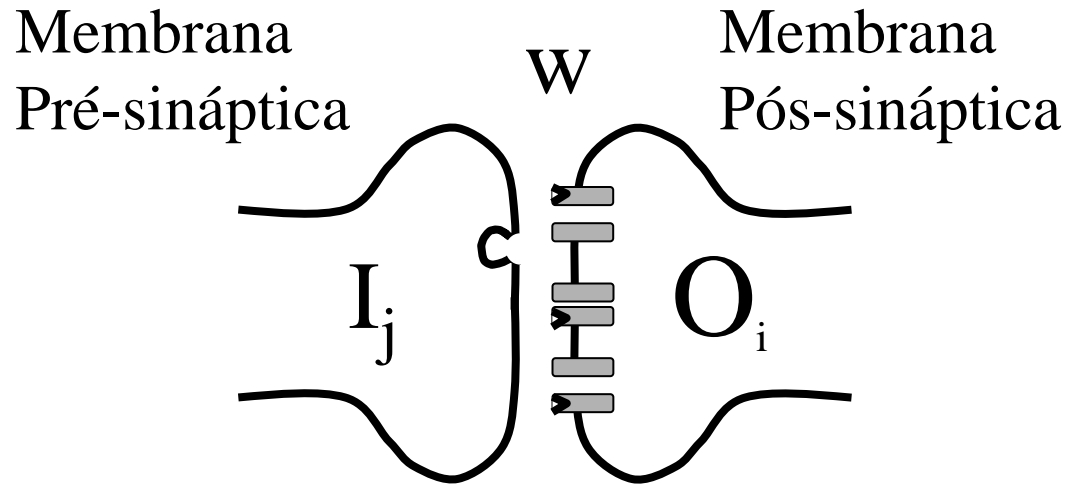
$\varepsilon = 0,1$

w_{ij}	I_j	O_i	Δw_{ij}
0	1	1	0,1
0,1	1	0,5	

Modelos artificiais de plasticidade sináptica

1. Modelo de Hebb

$$\Delta w_{ij} = \varepsilon O_i I_j$$



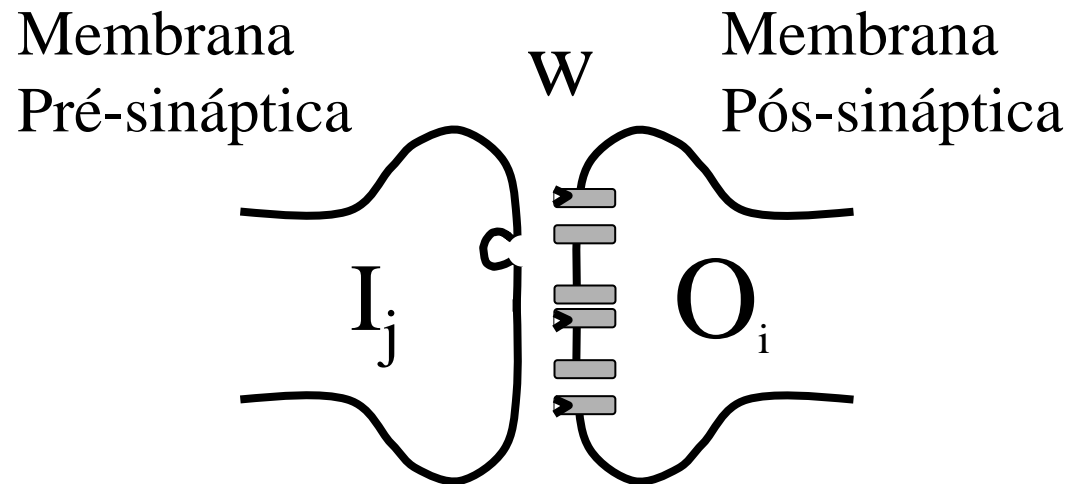
$$\varepsilon = 0,1$$

w_{ij}	I_j	O_i	Δw_{ij}
0	1	1	0,1
0,1	1	0,5	0,05
0,15			

Modelos artificiais de plasticidade sináptica

1. Modelo de Hebb

$$\Delta w_{ij} = \varepsilon O_i I_j$$



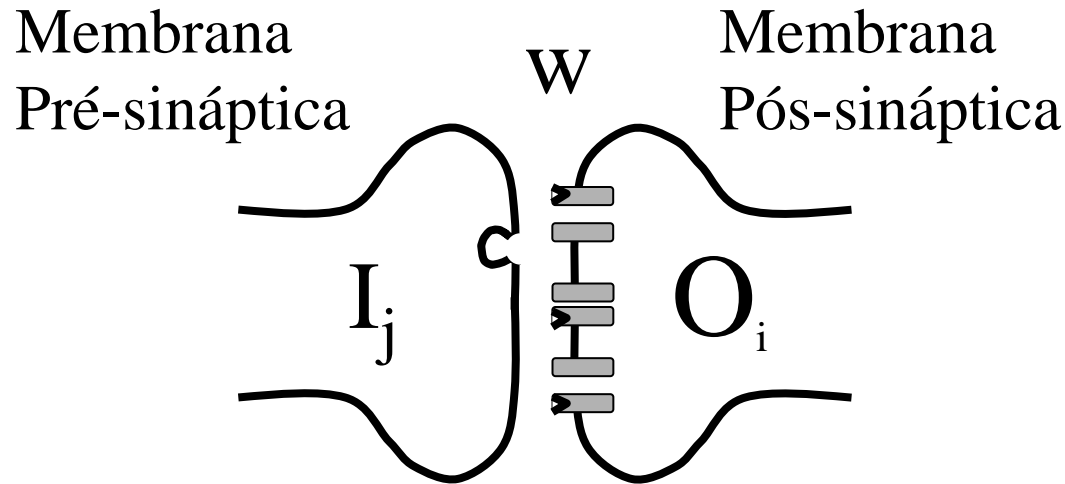
$$\varepsilon = 0,1$$

w_{ij}	I_j	O_i	Δw_{ij}
0	1	1	0,1
0,1	1	0,5	0,05
0,15	0,1	0,5	

Modelos artificiais de plasticidade sináptica

1. Modelo de Hebb

$$\Delta w_{ij} = \varepsilon O_i I_j$$



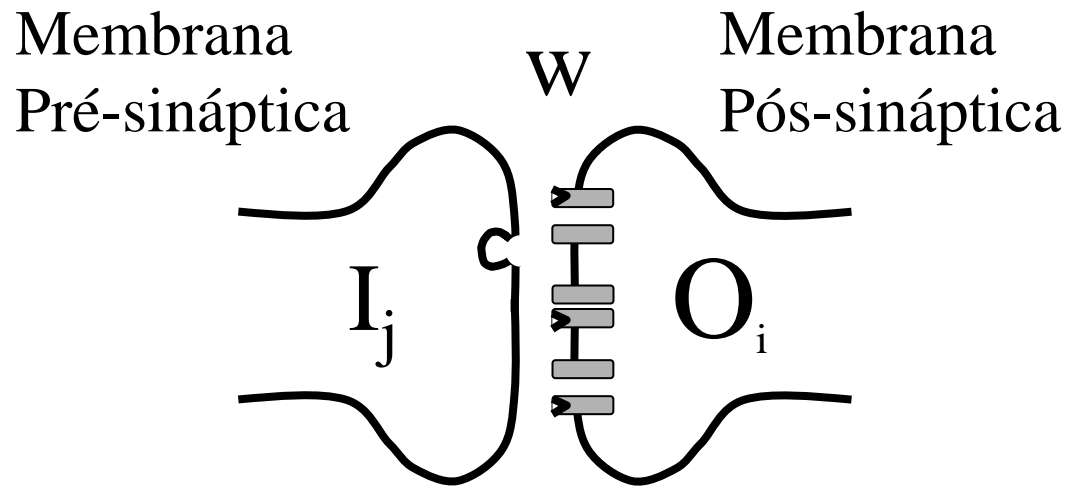
$$\varepsilon = 0,1$$

w_{ij}	I_j	O_i	Δw_{ij}
0	1	1	0,1
0,1	1	0,5	0,05
0,15	0,1	0,5	0,005
0,155			

Modelos artificiais de plasticidade sináptica

1. Modelo de Hebb

$$\Delta w_{ij} = \varepsilon O_i I_j$$



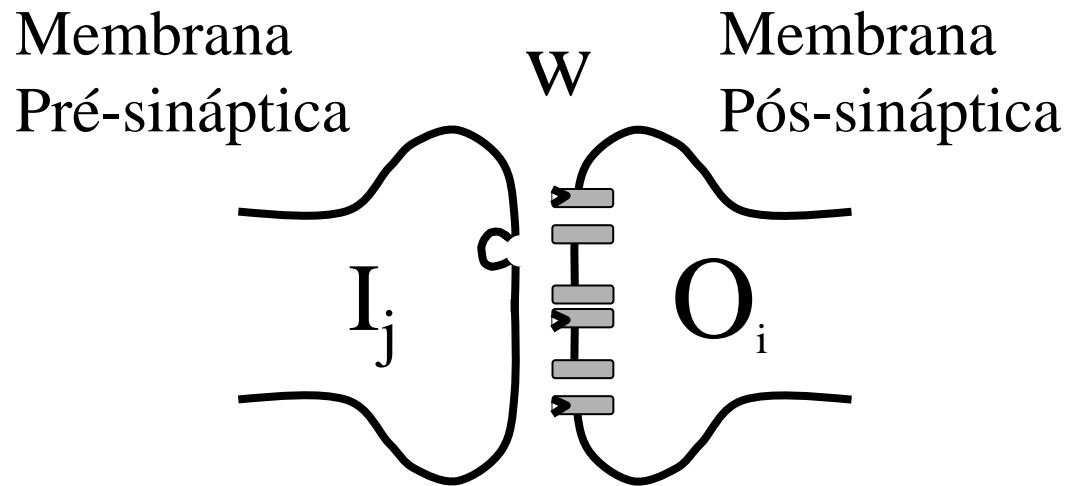
$$\varepsilon = 0,1$$

w_{ij}	I_j	O_i	Δw_{ij}
0	1	1	0,1
0,1	1	0,5	0,05
0,15	0,1	0,5	0,005
0,155	1	1	

Modelos artificiais de plasticidade sináptica

1. Modelo de Hebb

$$\Delta w_{ij} = \varepsilon O_i I_j$$



$$\varepsilon = 0,1$$

w_{ij}	I_j	O_i	Δw_{ij}
0	1	1	0,1
0,1	1	0,5	0,05
0,15	0,1	0,5	0,005
0,155	1	1	0,1
0,255			

Neste exemplo colocam-se valores arbitrários de I_j e O_i em cada instante

Redes neurais

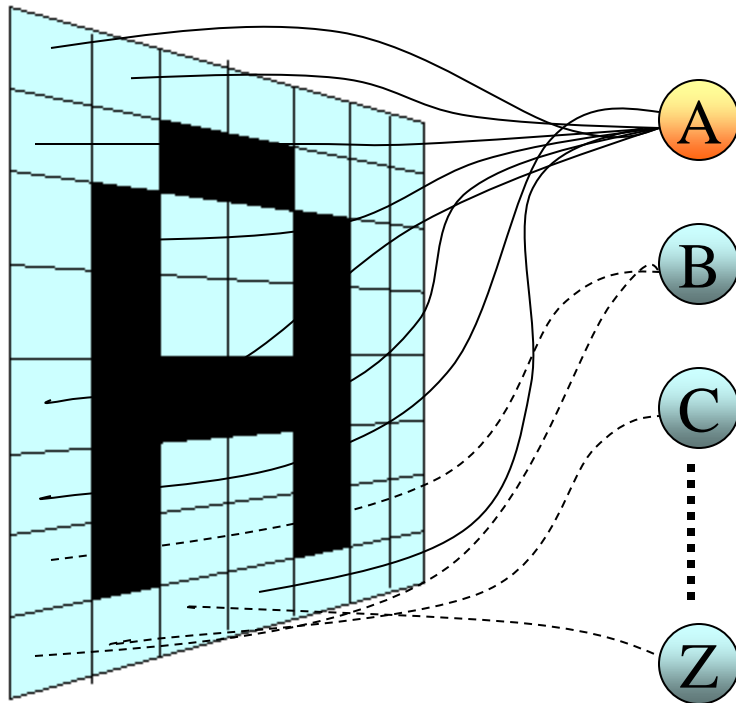
- Redes supervisionadas
 - Perceptron de Rosemblatt
 - Metodos de descida de gradiente.
 - Perceptron multicamada (método de back-propagation)
- Redes não-supervisionadas
 - Rede SOM (Self Organizing Feature Map)



Redes supervisionadas:

Perceptron de Rosemblatt

$$o_{i,p} = f\left(\text{net}_{i,p} - s_i\right) = f\left(\sum_{j=1}^{j=n} w_{ij} i_{jp} - s_i\right)$$



$$\Delta s_{ip} = -\xi \cdot \text{erro}_{ip}$$

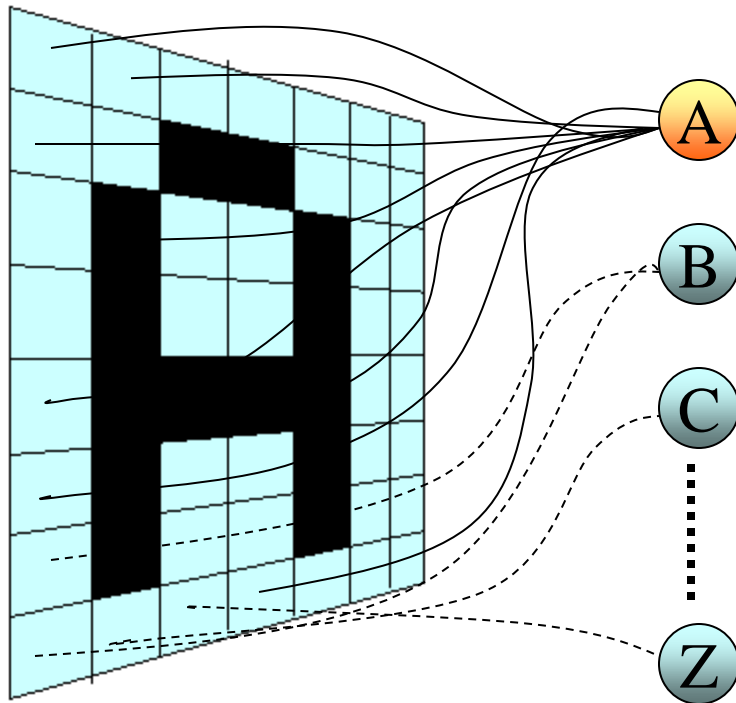
$$\Delta w_{ij,p} = \xi \cdot \text{erro}_{ip} \cdot i_{j,p}$$



Redes supervisionadas:

Perceptron de Rosemblatt

$$o_{i,p} = f\left(\text{net}_{i,p} - s_i\right) = f\left(\sum_{j=1}^{j=n} w_{ij} i_{jp} - s_i\right)$$



$$\begin{aligned}\Delta s_{ip} &= -\xi \cdot \text{erro}_{ip} = \\ &= \xi \cdot \text{erro}_{ip}(-1) \\ &= \xi \cdot \text{erro}_{ip} \cdot i_{n+1,p}\end{aligned}$$

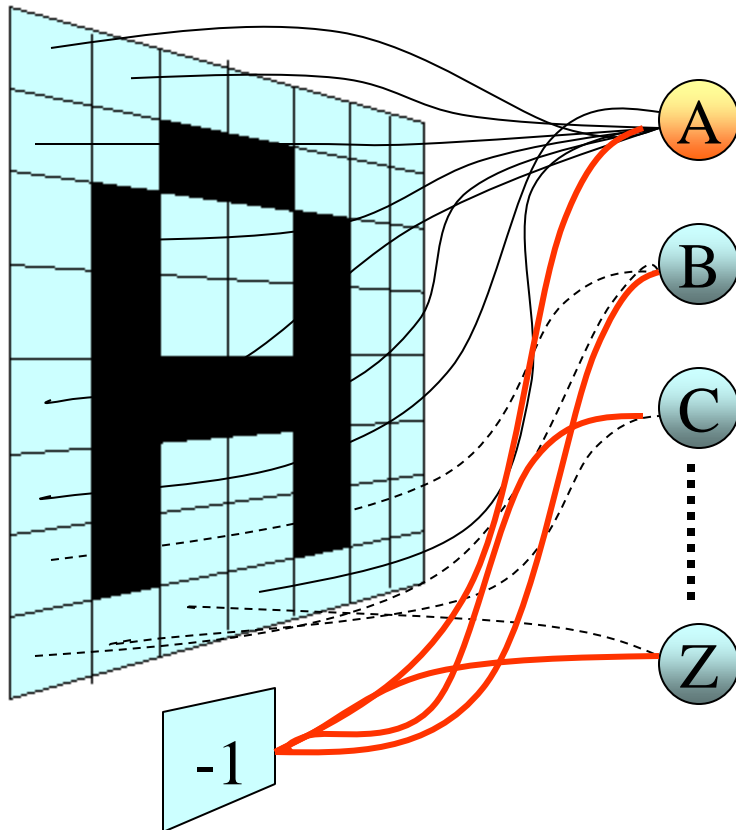
$$\Delta w_{ij,p} = \xi \cdot \text{erro}_{ip} \cdot i_{j,p}$$



Redes supervisionadas:

Perceptron de Rosemblatt

$$o_{i,p} = f(net_{i,p} - s_i) = f\left(\sum_{j=1}^{j=n} w_{ij} i_{jp} + w_{i(n+1)}(-1)\right)$$



$$\begin{aligned}\Delta s_{ip} &= -\xi \cdot erro_{ip} = \\ &= \xi \cdot erro_{ip}(-1) \\ &= \xi \cdot erro_{ip} \cdot i_{n+1,p}\end{aligned}$$

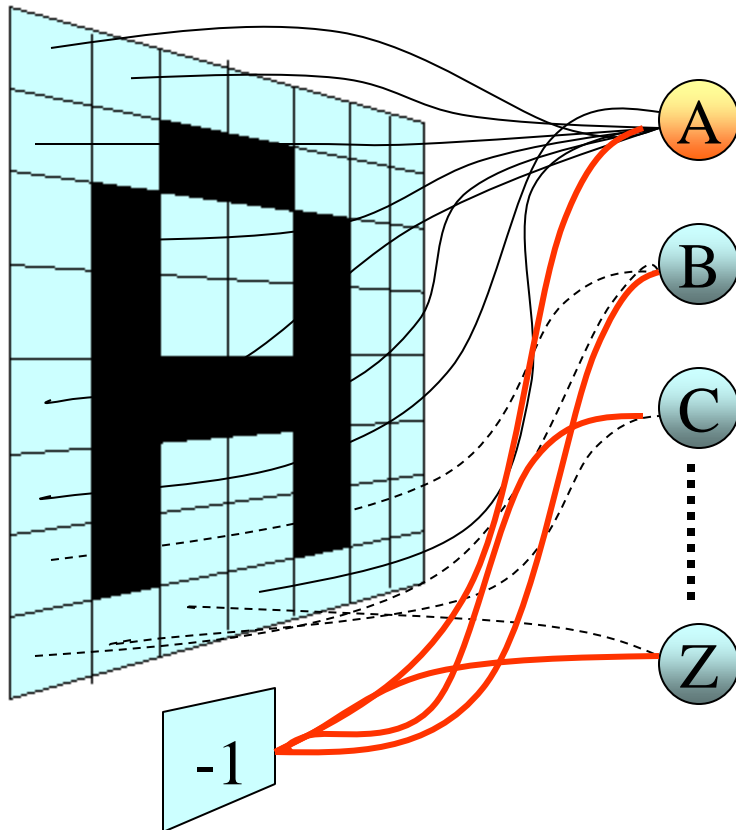
$$\Delta w_{ij,p} = \xi \cdot erro_{ip} \cdot i_{j,p}$$



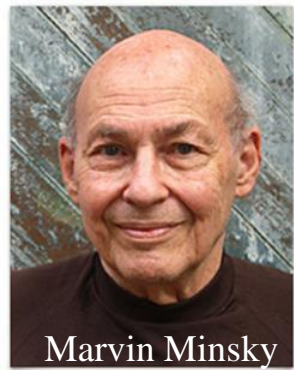
Redes supervisionadas:

Perceptron de Rosemblatt

$$o_{i,p} = f(\text{net}_{i,p}) = f\left(\sum_{j=1}^{j=n+1} w_{ij} \cdot i_{jp}\right)$$



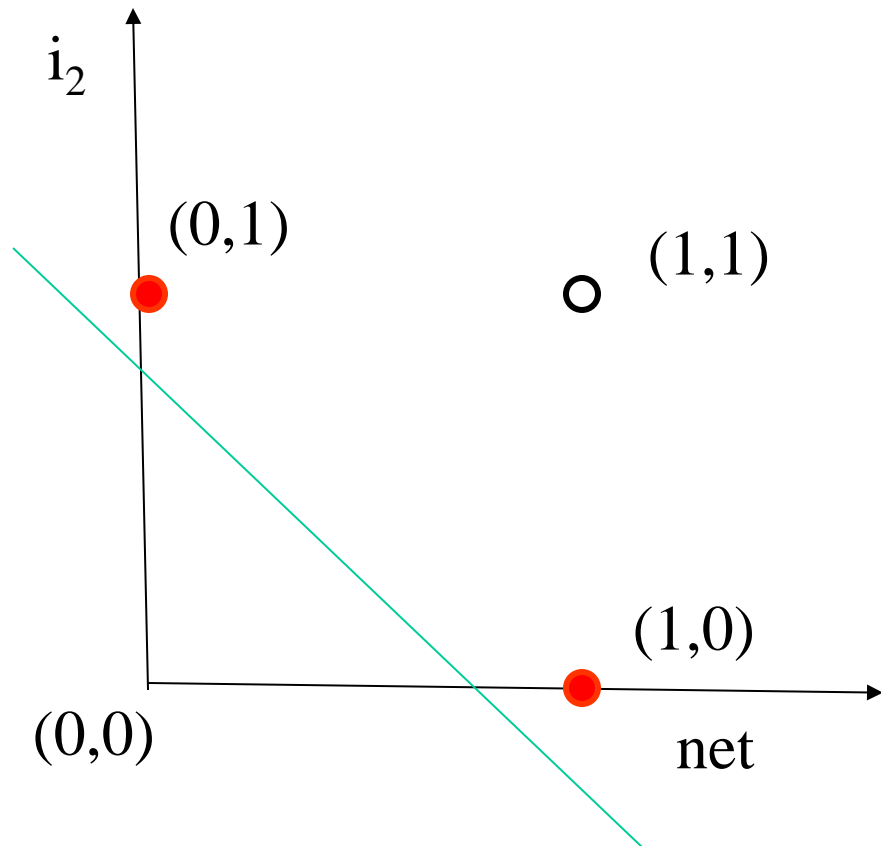
$$\Delta w_{ij,p} = \xi \cdot \text{erro}_{ip} \cdot i_{j,p}$$

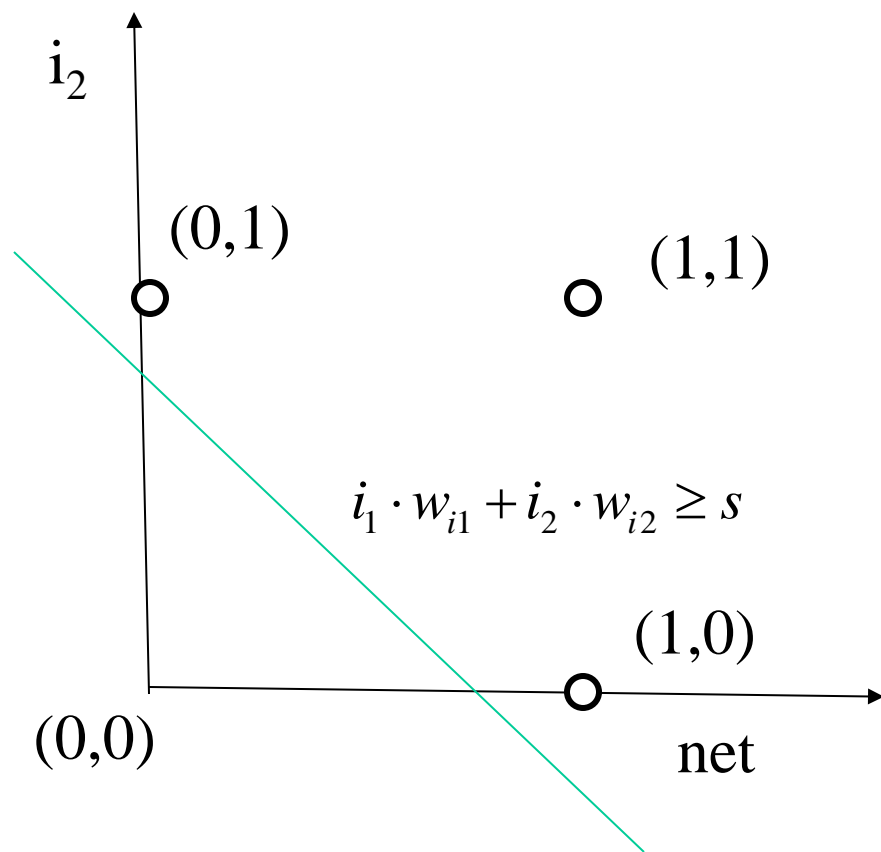


Marvin Minsky e o desafio do “ou exclusivo”

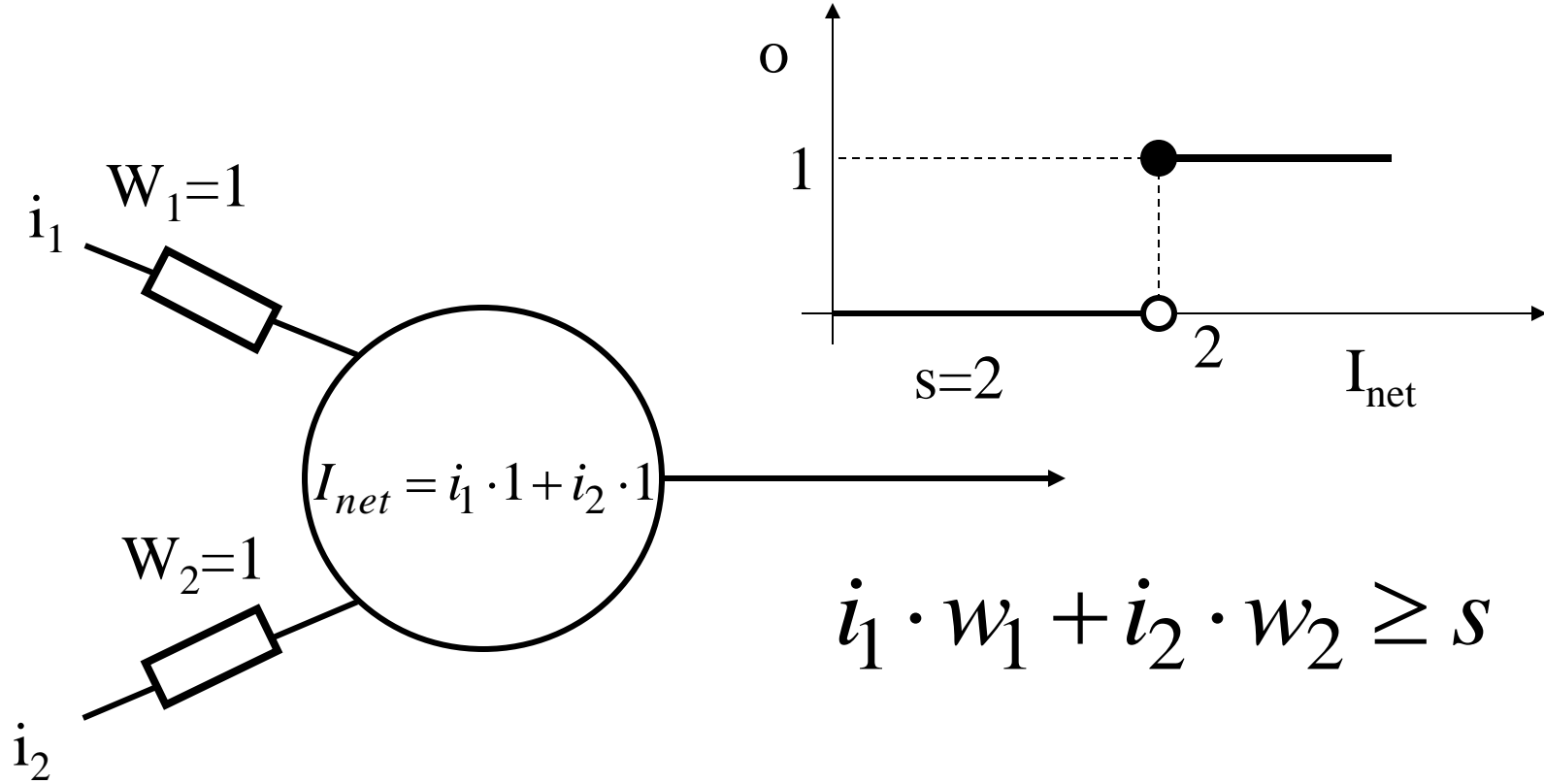
Função EX-OR

i_1	i_2	O
0	0	0
0	1	1
1	0	1
1	1	0





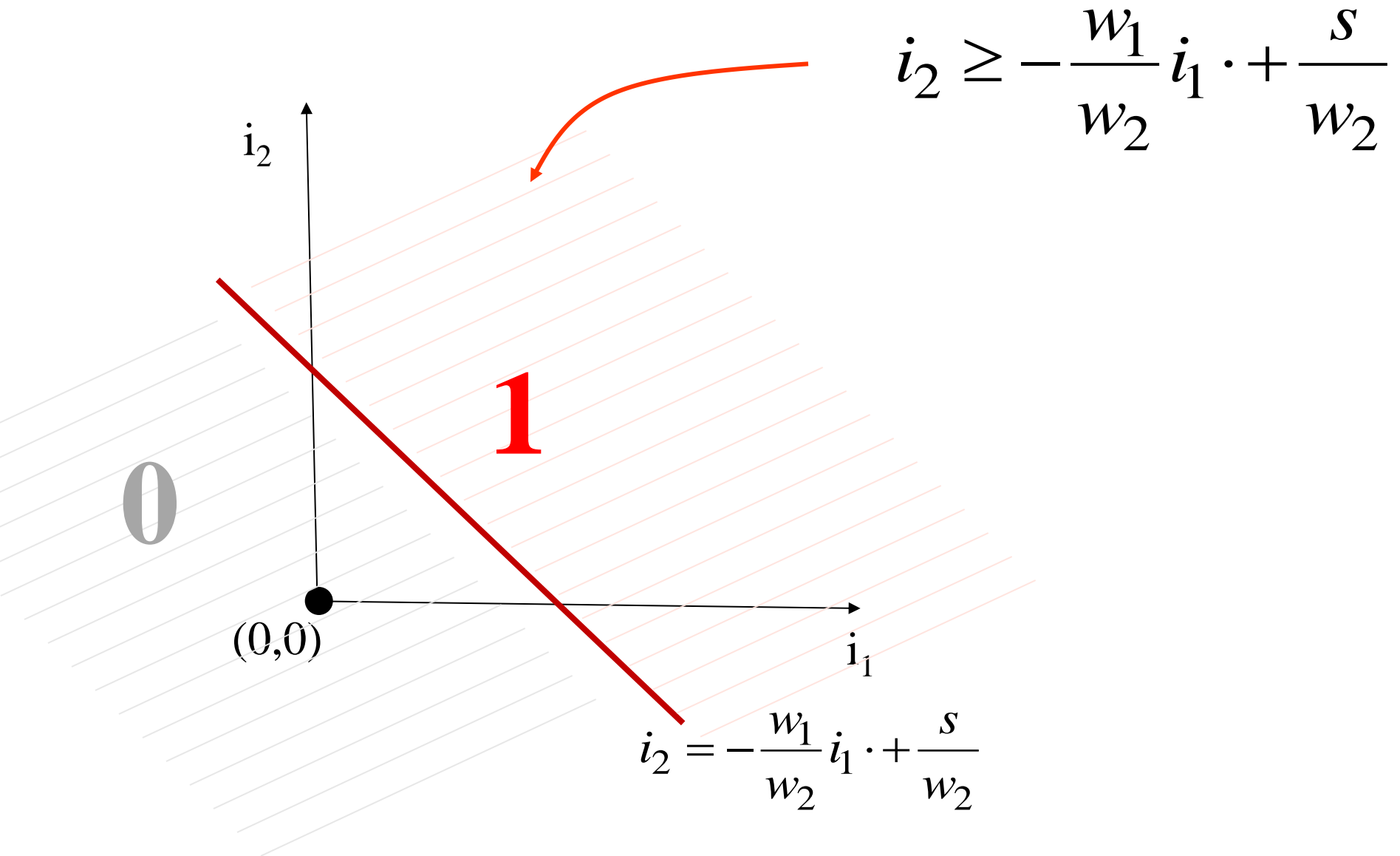
O neurônio como discriminador linear (I)



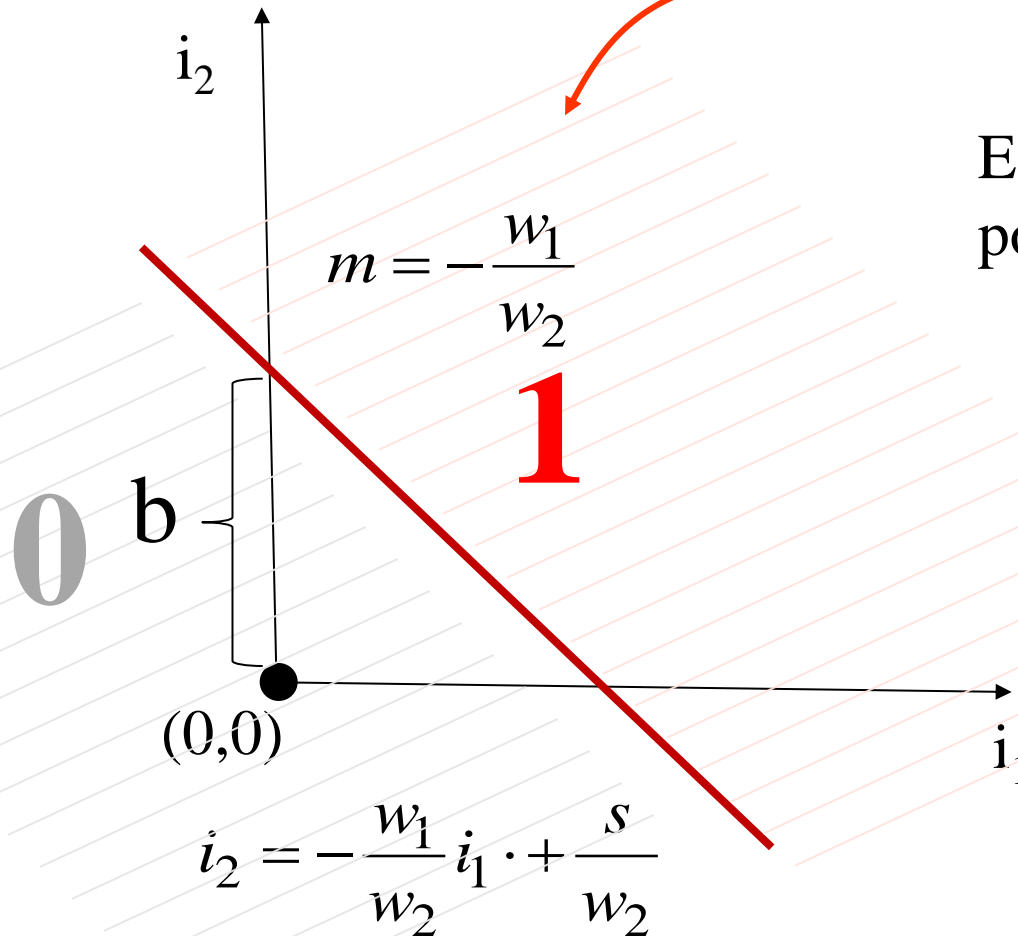
$$i_1 \cdot w_1 + i_2 \cdot w_2 \geq s$$

⚠
$$i_2 \geq -\frac{w_1}{w_2} i_1 + \frac{s}{w_2}$$

O neurônio como discriminador linear (II)



O neurônio como discriminador linear (II)



$$i_2 \geq -\frac{w_1}{w_2} i_1 + \frac{s}{w_2}$$

Equação da reta em: forma
ponto-inclinação

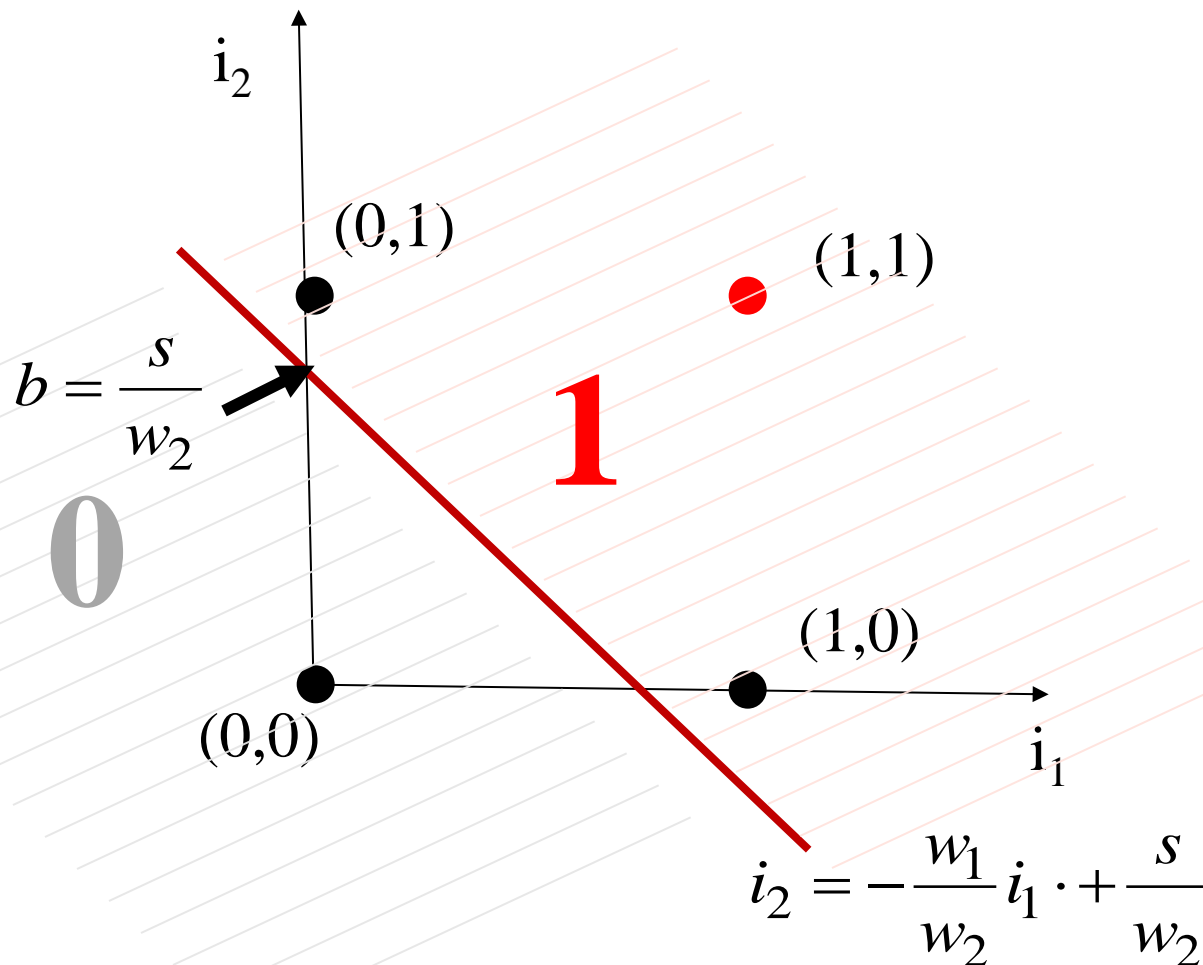
$$i_2 = m i_1 + b$$

sendo m e b :

$$m = -\frac{w_1}{w_2}$$

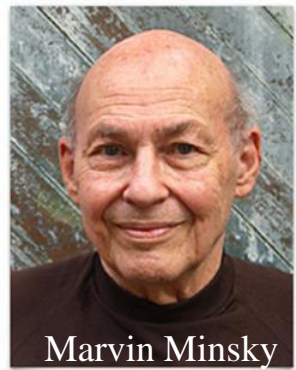
$$b = \frac{s}{w_2}$$

Exercício: Deixando w_2 fixo, $w_2=1$, quais seriam os possíveis valores de s e w_1 para ter um discriminador linear que implemente a função AND
Mostrar num sistema de coordenadas w e s os valores possíveis e as regiões de erro



Função AND

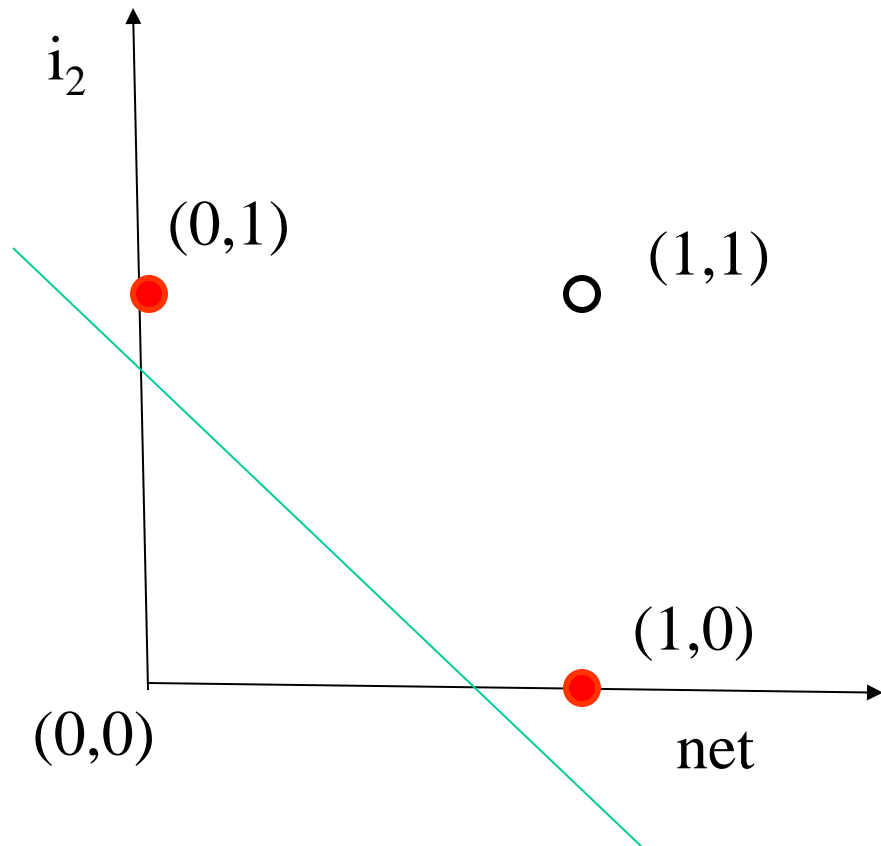
i_1	i_2	$f(\text{Inet})$
0	0	0
0	1	0
1	0	0
1	1	1



Marvin Minsky e o desafio do “ou exclusivo”

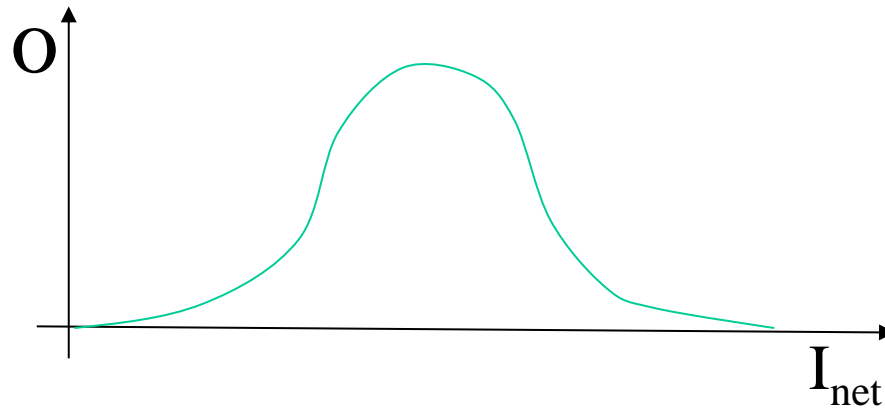
Função EX-OR

i_1	i_2	O
0	0	0
0	1	1
1	0	1
1	1	0



Exclusive-or (1º método de resolução)

- Utilizando uma função de ativação tipo gaussiana.

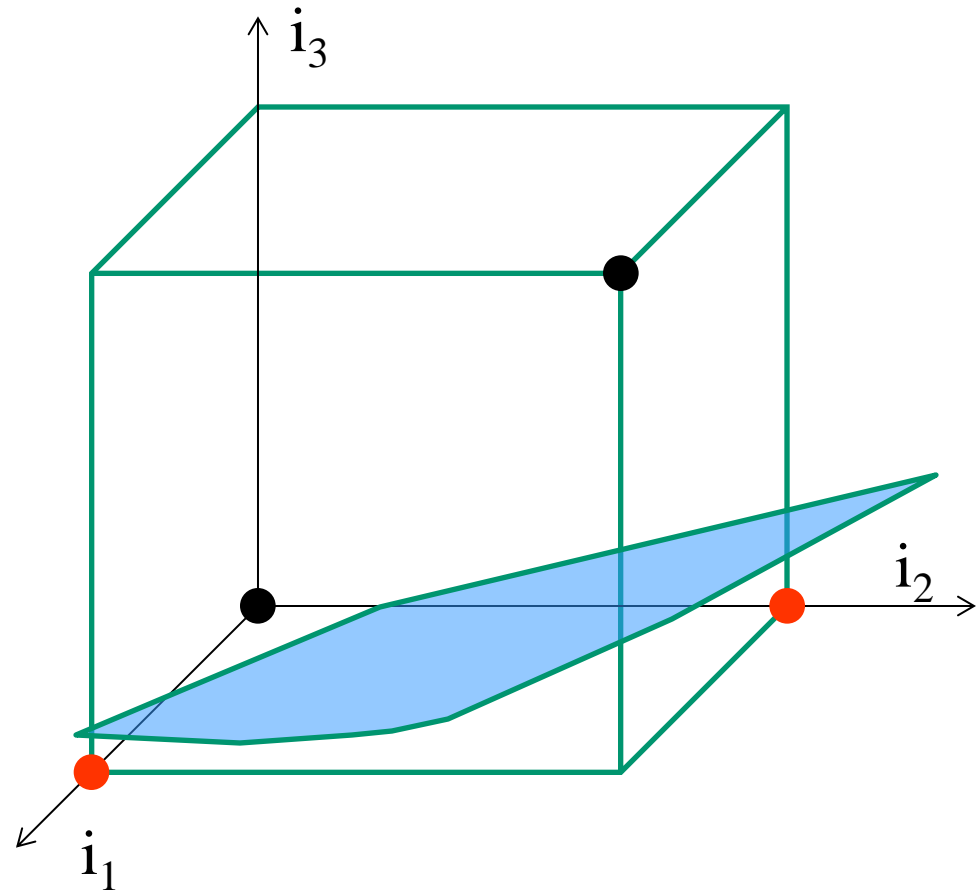


Exclusive-or (2º método de resolução)

- Utilizando uma novo input que é uma função dos dois anteriores: $i_3 = i_1 \& i_2$.

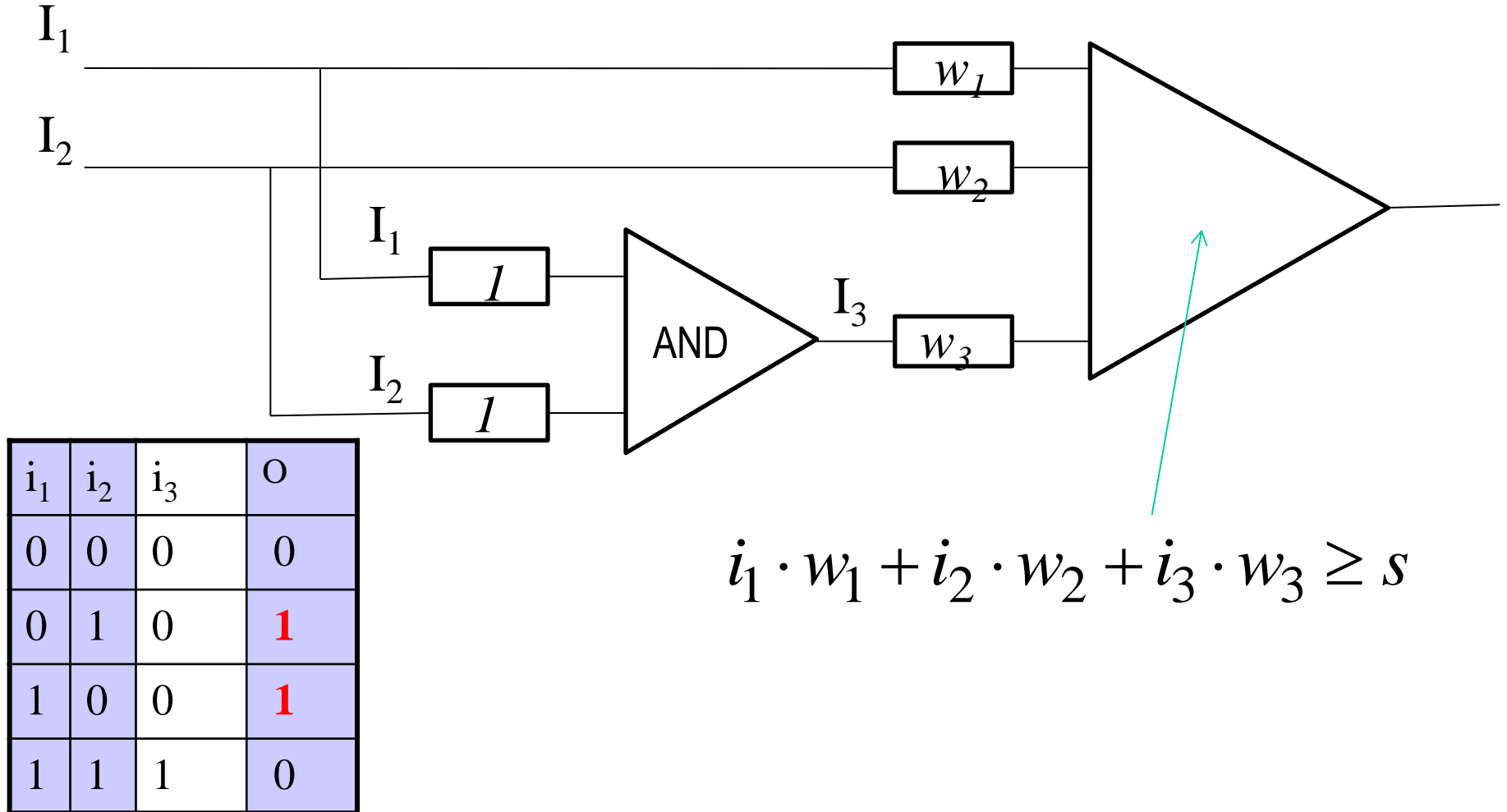
i_1	i_2	i_3	O
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$i_1 \cdot w_1 + i_2 \cdot w_2 + i_3 \cdot w_3 \geq s$$

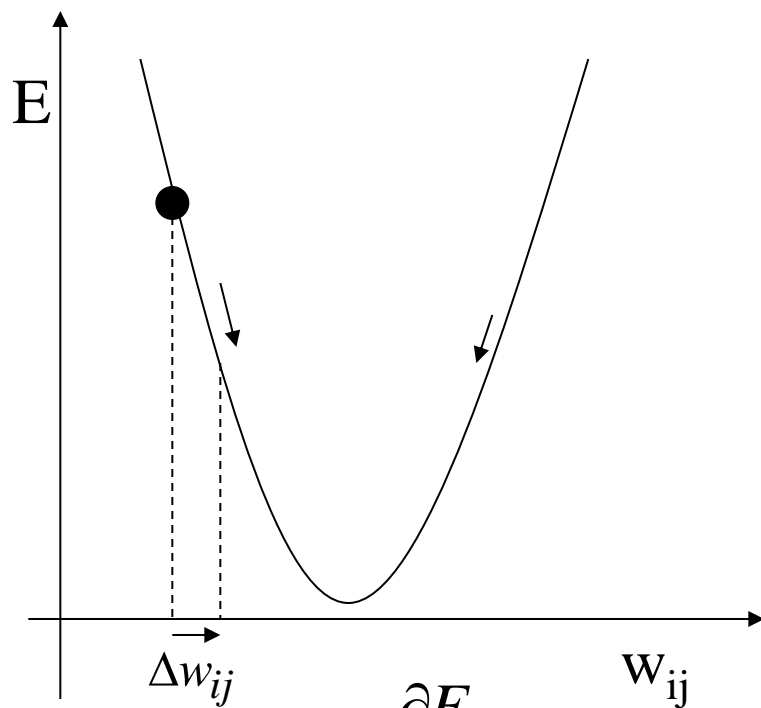


Exclusive-or (2º método de resolução)

- Utilizando uma novo input que é uma função dos dois anteriores: ex: $i_3 = i_1 \& i_2$.



Ajuste dos pesos mediante
método de gradiente descendente
no perceptron e no perceptron
multicamada



$$\Delta w_{ij} = -k \frac{\partial E_p}{\partial w_{ij}}$$



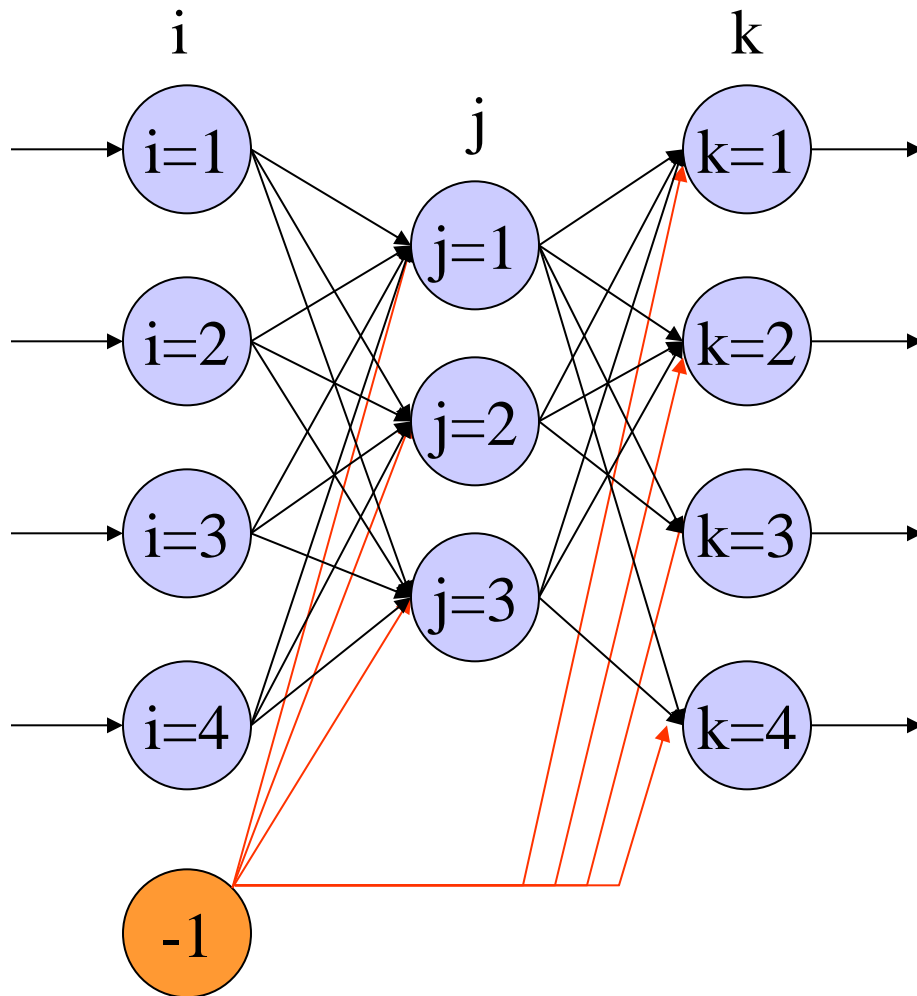
$$E = \sum_p error_p = \sum_p \sum_i (t_{ip} - o_{ip})^2 =$$

$$= \sum_p \sum_i \left(t_{ip} - \sum_j w_{ij} i_j \right)^2$$

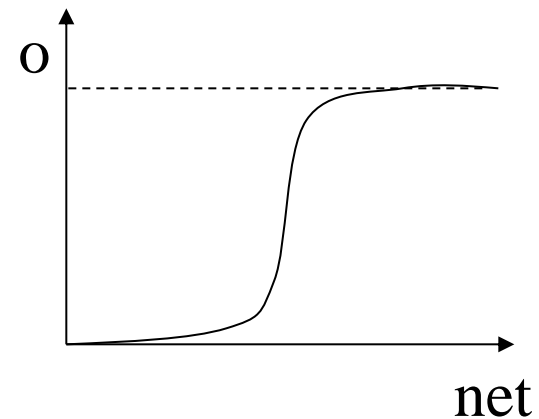
$$\Delta w_{ij} = -k \frac{\partial E_p}{\partial w_{ij}} = -k \cdot 2(t_{ip} - o_{ip})(-i_{jp}) =$$

$$= \xi \cdot error_{ip} \cdot i_{jp} = \xi \cdot \delta_{ip} \cdot i_{jp}$$

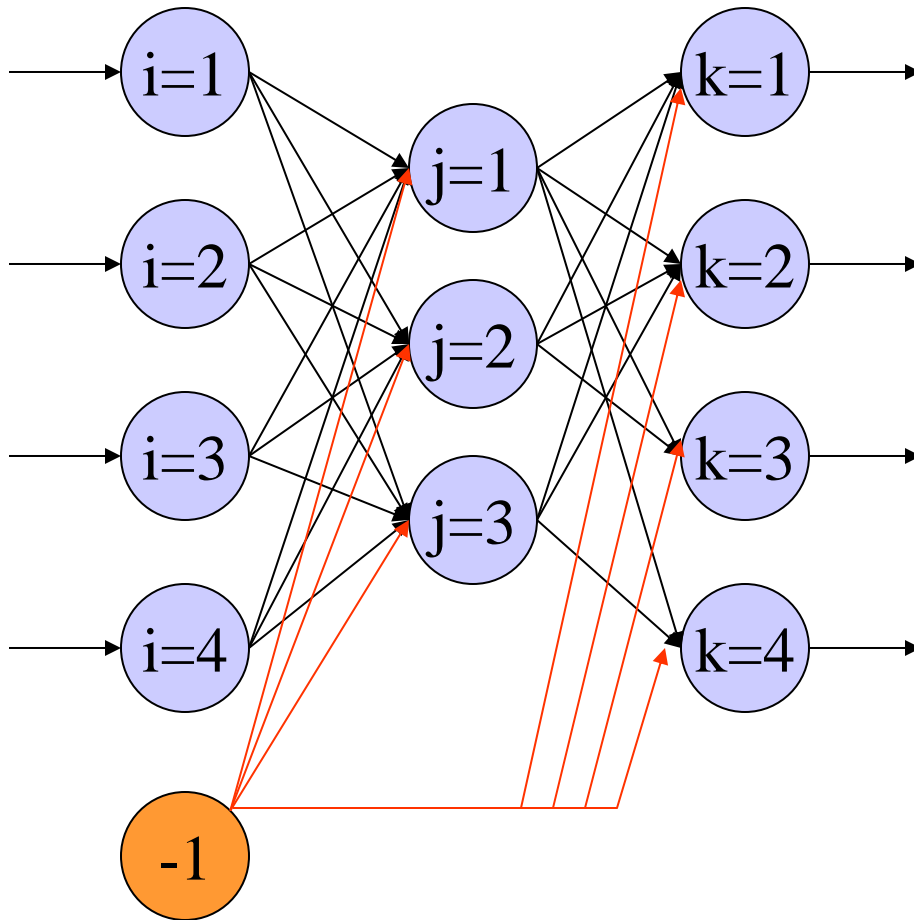
Perceptron multicamada (método de back-propagation)



$$f(net) = \frac{1}{1 + e^{(-net)}}$$



Perceptron multicamada (método de back-propagation)



$$\Delta w_{ij} = \xi \cdot \delta_i \cdot i_j$$

Neurônios última camada

$$\delta_k = (t_k - o_k) f'(net_k)$$

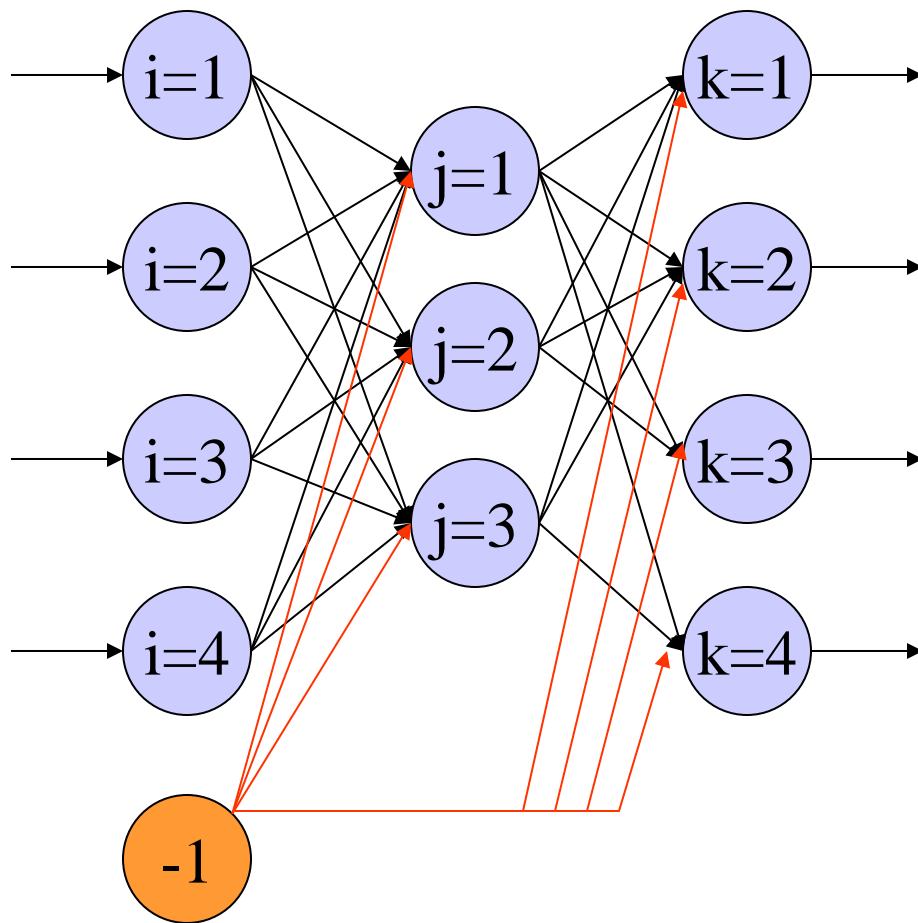
$$\Delta w_{kj} = \xi \cdot \delta_k \cdot o_j$$

Neurônios camadas anteriores

$$\delta_j = \left(\sum_k w_{kj} \delta_k \right) f'(net_j)$$

$$\Delta w_{ji} = \xi \cdot \delta_j \cdot o_i$$

Back-propagation com momentum



$$M = \alpha \Delta w_{ij}(t-1) = \alpha (w_{ij}(t) - w_{ij}(t-1))$$

$$\Delta w_{ij} = \xi \cdot \delta_i \cdot i_j + M$$

Neurônios última camada

$$\delta_k = (t_k - o_k) f'(net_k)$$

$$\Delta w_{kj} = \xi \cdot \delta_k \cdot o_j + M$$

Neurônios camadas anteriores

$$\delta_j = \left(\sum_k w_{kj} \delta_k \right) f'(net_j)$$

$$\Delta w_{ji} = \xi \cdot \delta_j \cdot o_i + M$$

Como ajustar o fator de aprendizado ξ

- Com valores de ξ pequenos <0.01 o aprendizado é mais lento mais converge melhor para um mínimo global.
- Com valores de ξ grandes >0.01 o aprendizado é mais rápido mas converge pior para um mínimo global.



Como organizar o conjunto de padrões de treino

- Do conjunto dos padrões de entrada/saída totais separamos aproximadamente um 50% para treino, 25% para teste (no final) e 25% para validação.
- O grupo de validação é aplicado á rede durante o treino para saber como o treino está indo.
- O conjunto de todos os padrões de treino é chamado de época (a contagem é feita por épocas)

Número de neurônios e camadas

- Normalmente é suficiente colocar uma camada escondida embora mais camadas possam ser necessárias se os dados são altamente não lineares.
- O número de neurônios da camada intermediária normalmente é bem menor do que o número de neurônios da camada de entrada. Se a rede não consegue diminuir suficientemente o erro pode ser necessário repetir o treino com mais neurônios

Exemplo Matlab:


```
p = [-1 -1 2 2;0 5 0 5];  
t = [-1 -1 1 1]; %Padrões de teino e de teste (em colunas)  
  
net=newff(minmax(p),[3,1],{'tansig','purelin'},'traingd'); % Arquitetura rede.  
%minmax serve para ajustar o range dos dados  
  
net.trainParam.show = 50;  
net.trainParam.lr = 0.05; %Taxa de aprendizado (learning rate, lr )  
net.trainParam.epochs = 300; %Numero de épocas  
net.trainParam.goal = 1e-5; %Erro global a ser atingido para parar treino  
  
[net,tr]=train(net,p,t); %Instrução para treinar rede  
  
a = sim(net,p) %Depois do treino aplicamos os o padrões de teste na entrada  
a = -1.0010 -0.9989 1.0018 0.9985 % O resultado é obtido na saída  
  
%Para treino com padrões de validação consultar Help do Matlab
```

Memorização versus generalização

- Com poucos neurônios na camada intermediária é possível generalizar (e porem interpolar) melhor os dados e eliminar o ruído dos dados.
- Com muitos neurônios pode acontecer um sobretreino indesejado e memorizar mesmo o ruído dos dados.



Redes neurais autoassociativas

- Os pesos dos neurônios da camada intermediária das redes autoassociativas convergem para os componentes principais (Ver “Principal Component Neural Networks” de Diamantaras & Kung. 
- Redes autoassociativas são úteis para extrair resíduos para, por exemplo, aplicação das redes para manutenção de equipamentos

Aplicação a previsão de séries temporais

- Usa-se para prever o valor de $x(t+t)$ a partir dos valores:

$$x(t), x(t-\tau), x(t-2\tau).... (t-k\tau)$$