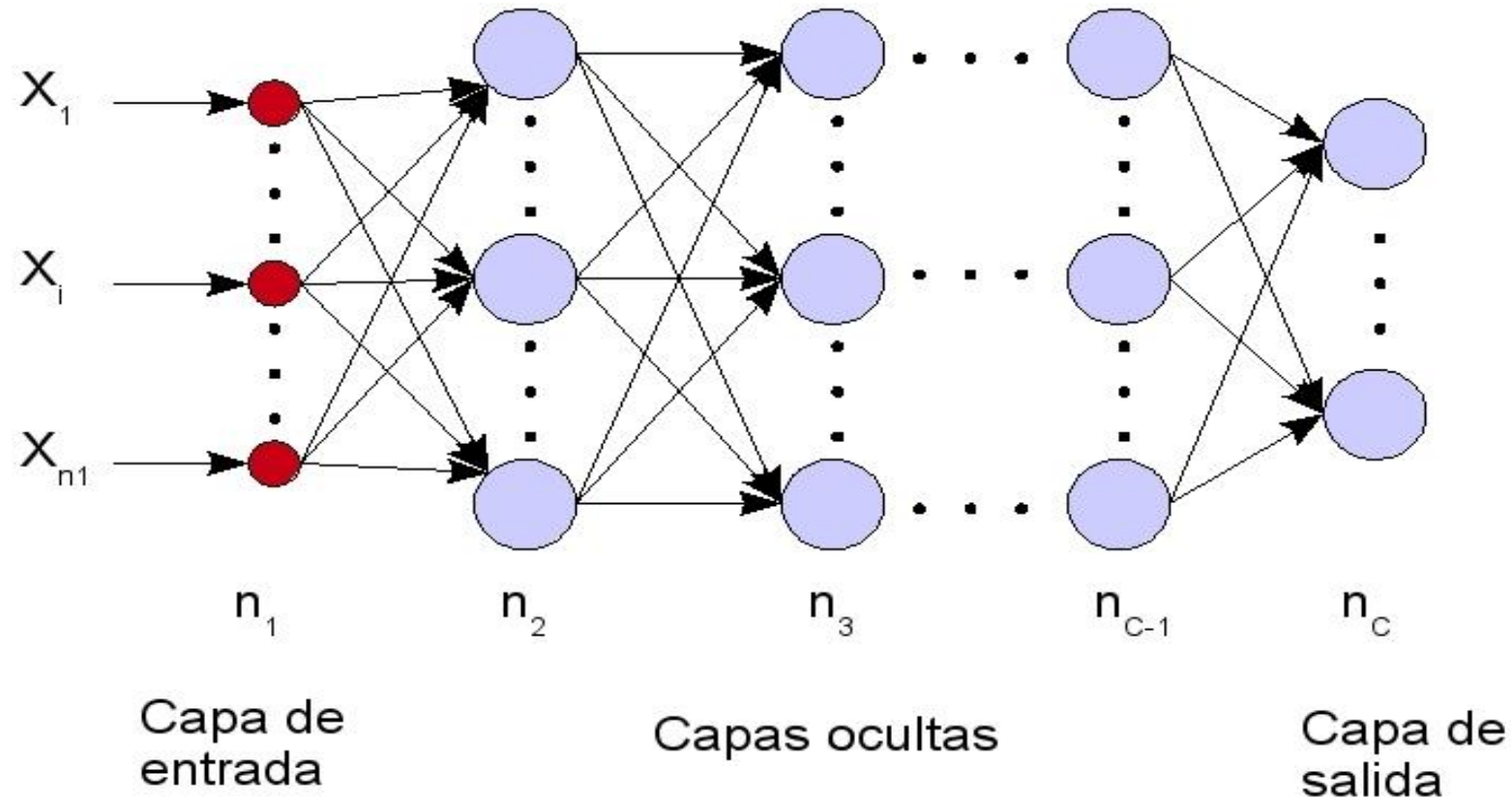


# MULTIPERCEPTRÓN Y EL ALGORITMO BACKPROPAGATION

# Multiperceptrón - Arquitectura

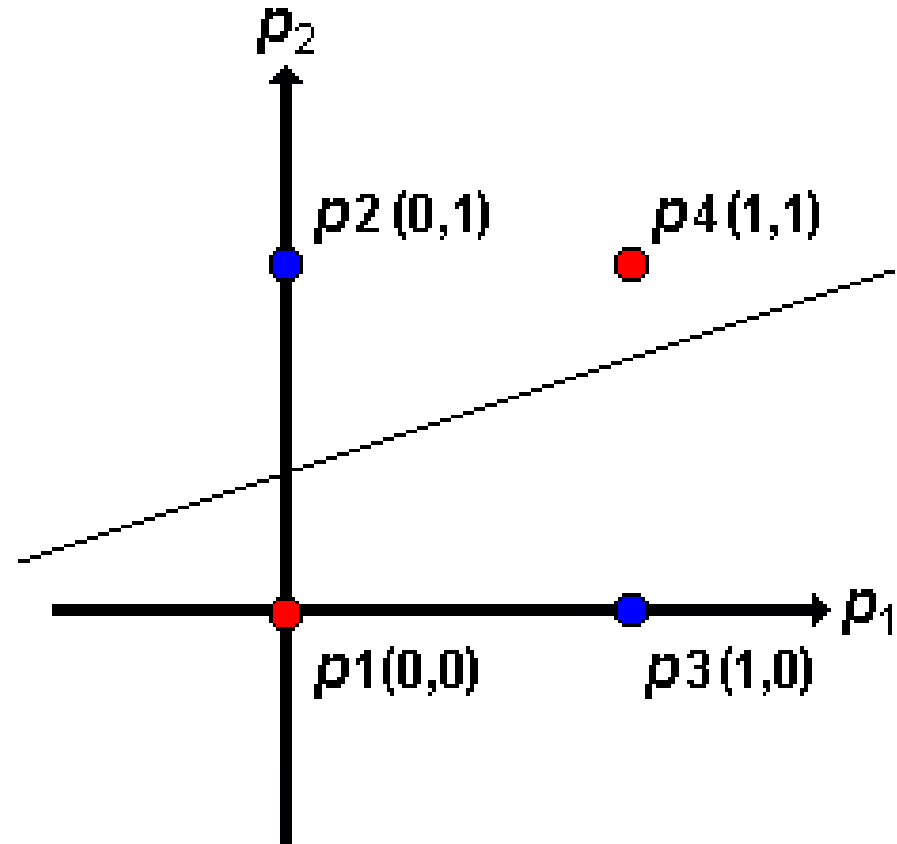
2



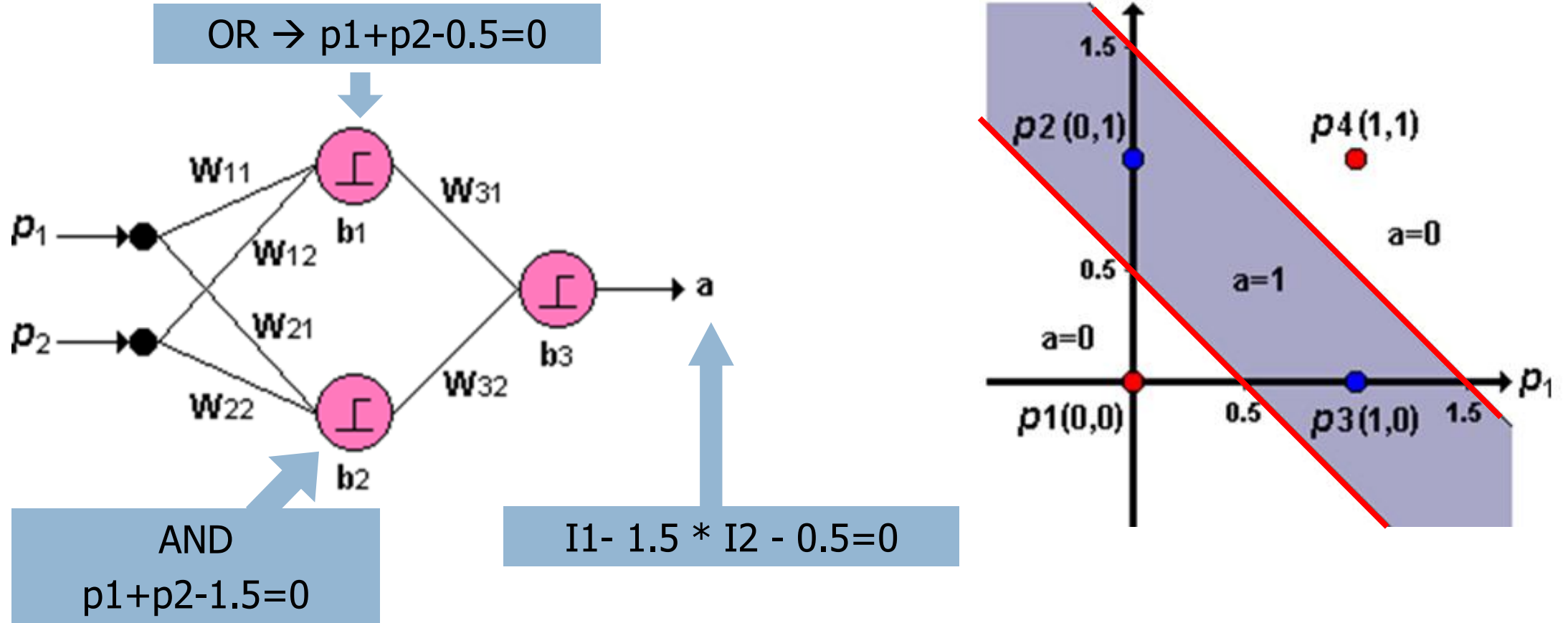
# Redes multicapa

3

- Con una sola neurona no se puede resolver el problema del XOR porque no es linealmente separable.



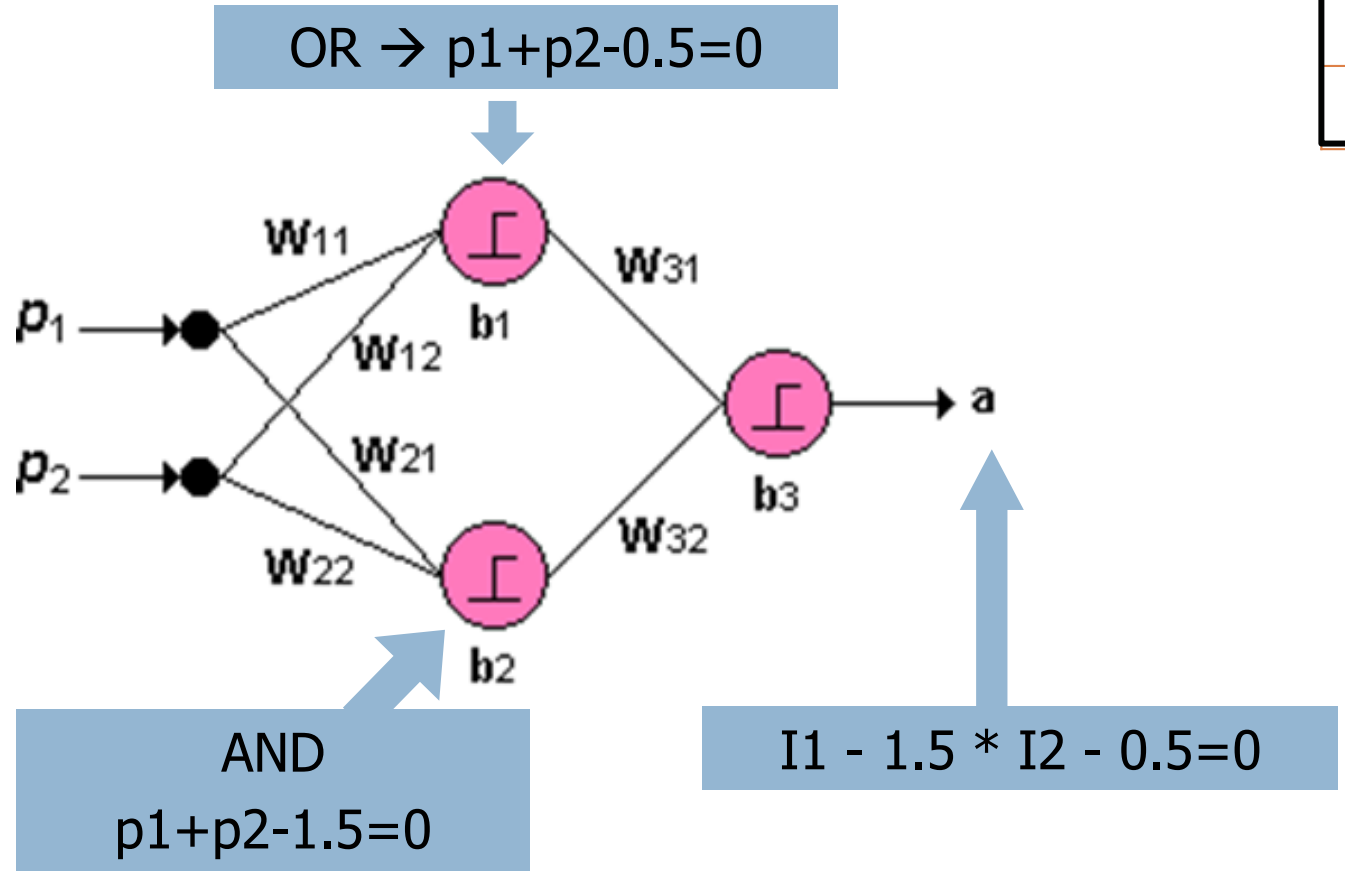
# XOR



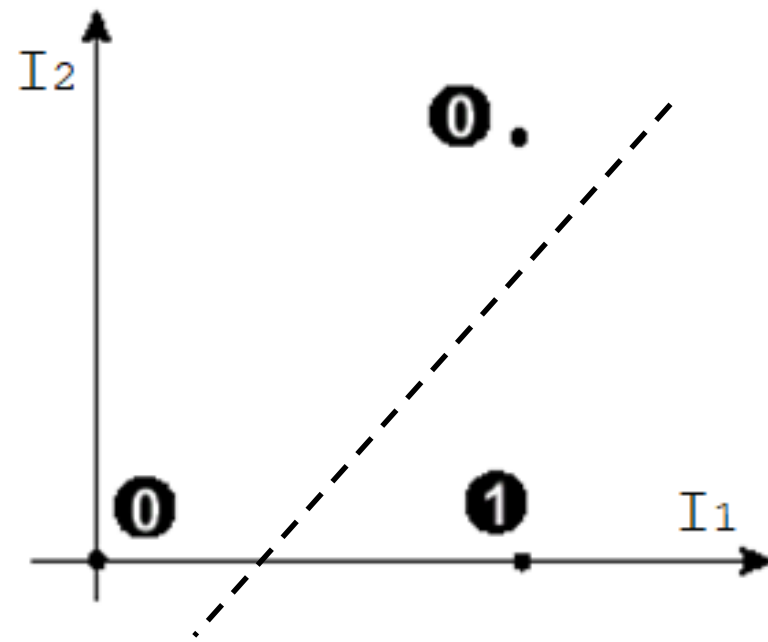
$$w_{11}=1 \quad w_{12}=1 \quad b_1=-0.5 \quad ; \quad w_{21}=1 \quad w_{22}=1 \quad b_2=-1.5 \quad ; \quad w_{31}=1 \quad w_{32}=-1.5 \quad b_3=-0.5$$

# XOR

5

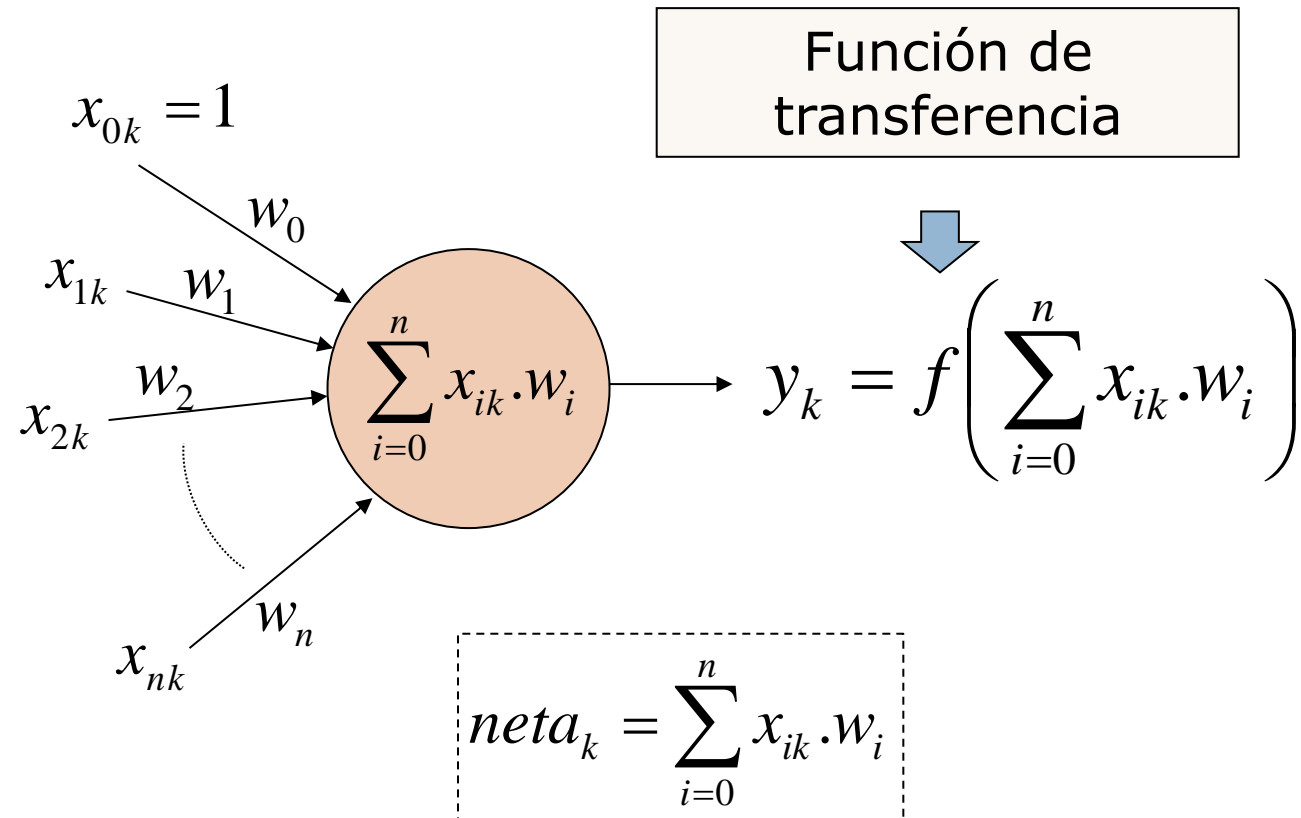


p1	p2	I1 (or)	I2 (AND)	a
1	0	1	0	1
1	1	1	1	0
0	0	0	0	0
0	1	1	0	1



# Neurona artificial

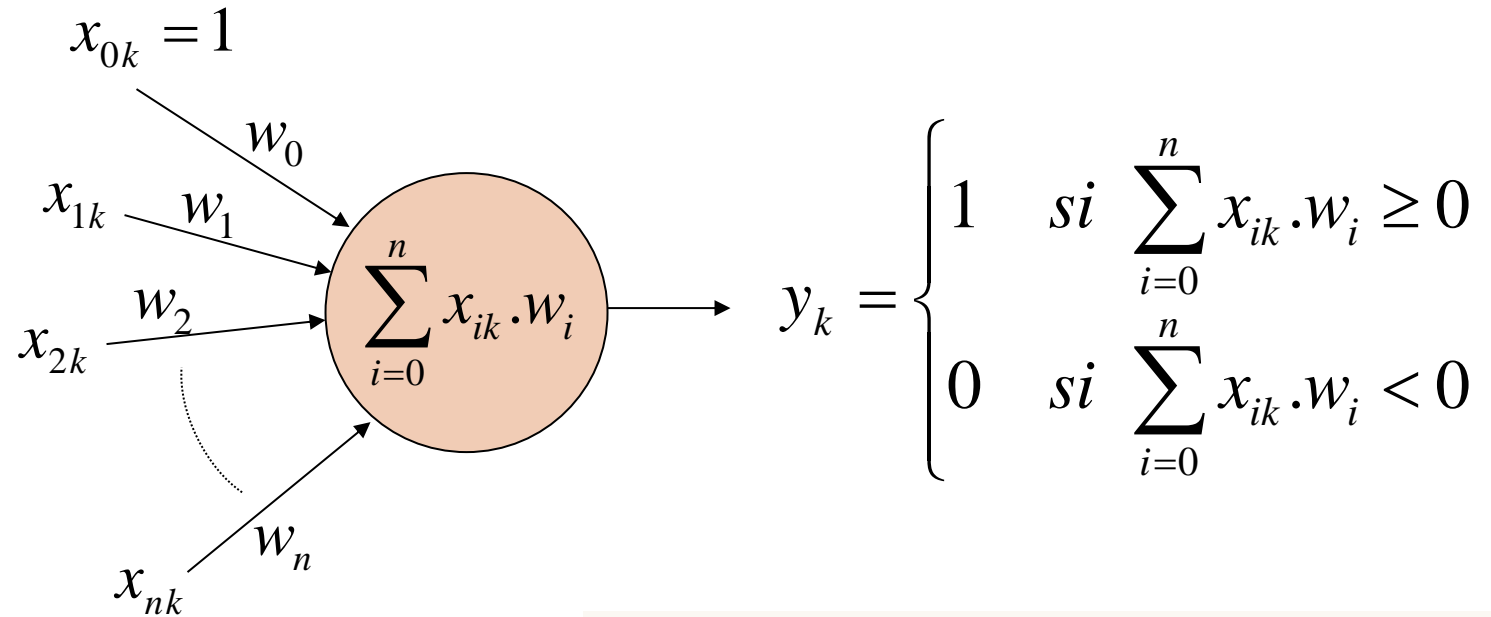
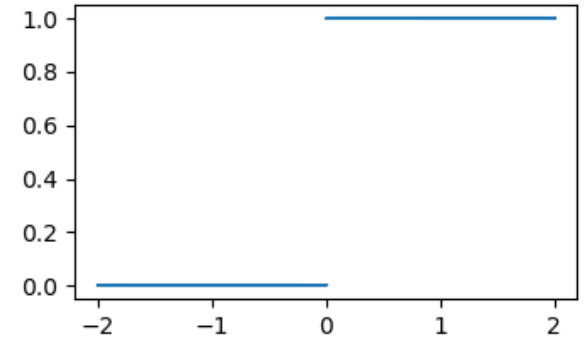
6



# Perceptrón

7

$$f(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$

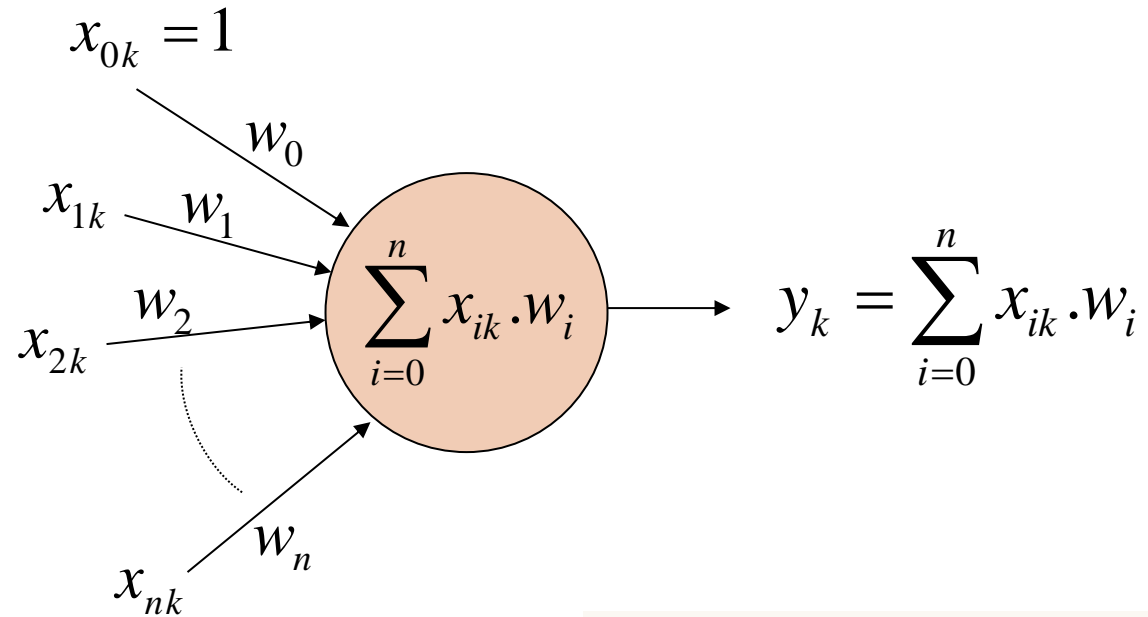
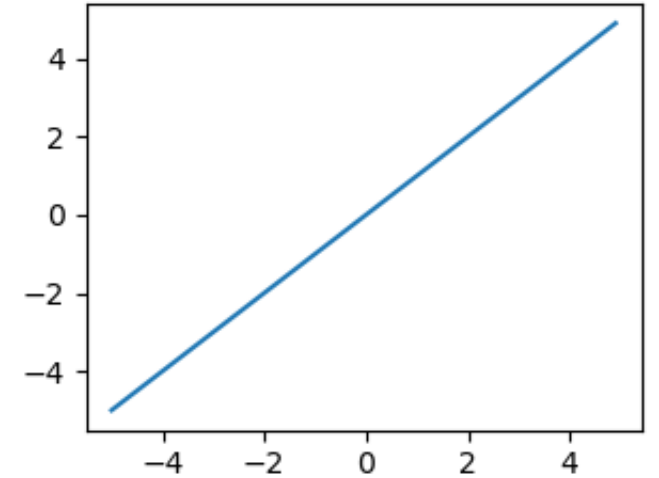


- Actualiza el vector  $W$  modificando el ángulo que forma con cada ejemplo  $X_k$

# Combinador lineal

8

$$f(x) = x$$



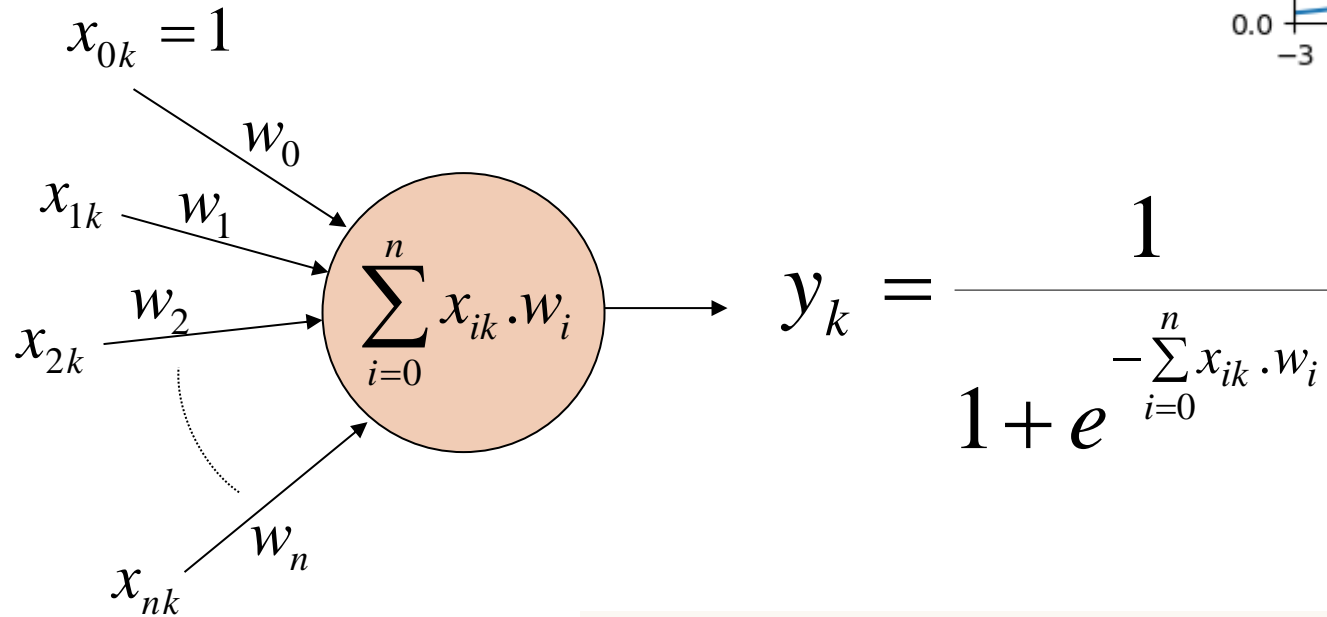
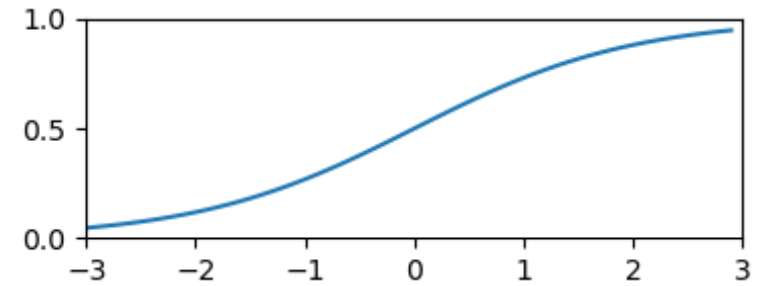
- Utiliza descenso por gradiente para actualizar el vector de pesos.



# Neurona no lineal (logsig)

9

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

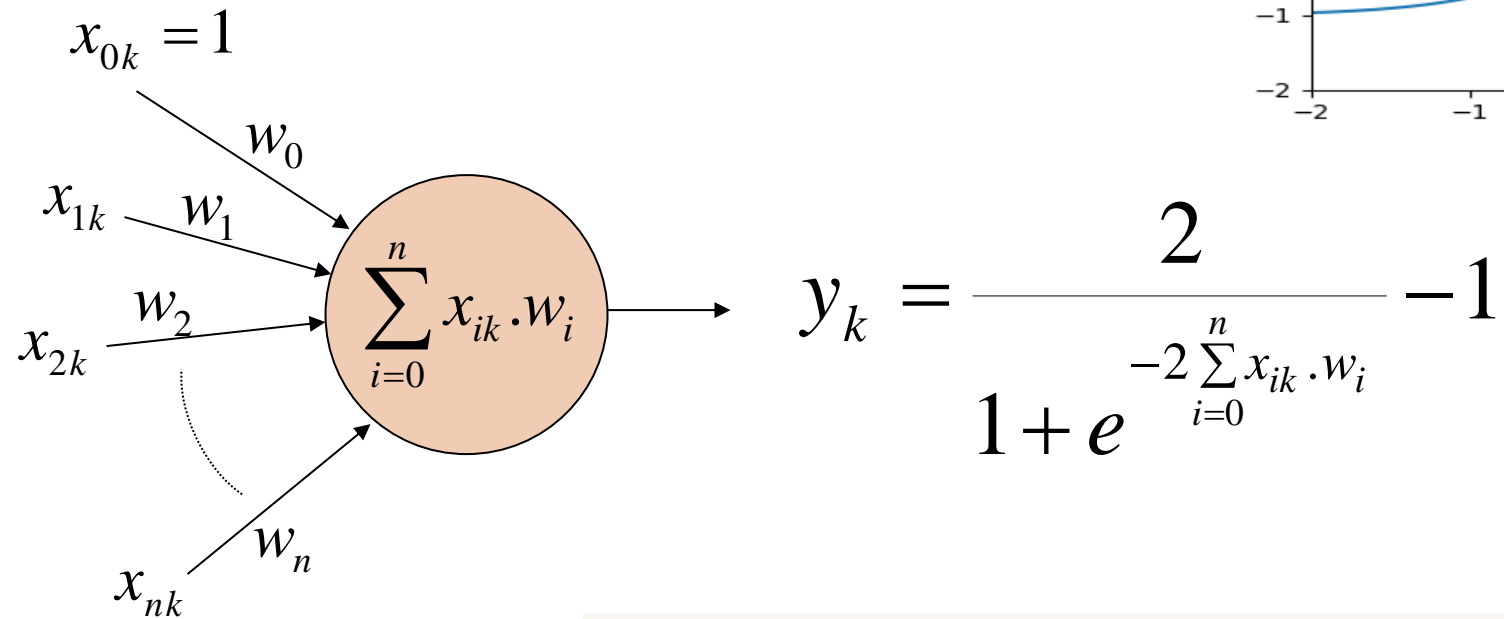
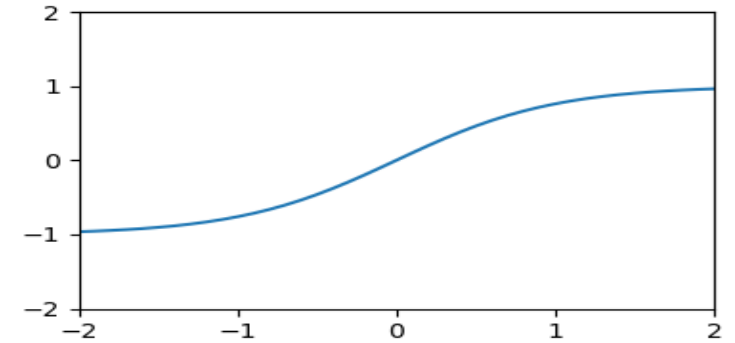


- Utiliza descenso por gradiente para actualizar el vector de pesos.

# Neurona no lineal (tansig)

10

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$



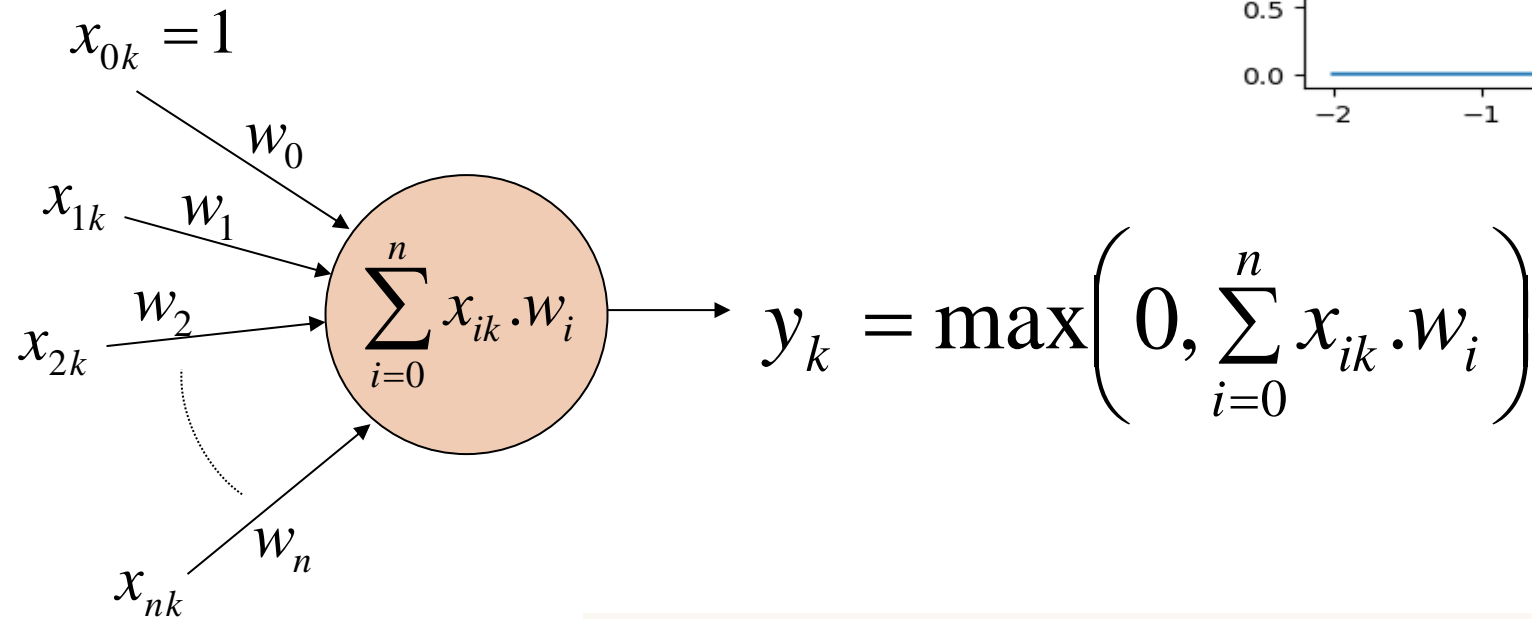
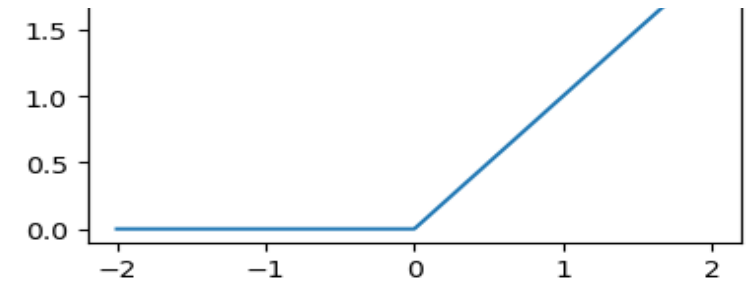
- Utiliza descenso por gradiente para actualizar el vector de pesos.

# Neurona no lineal (Relu)

11

$$f(x) = \max(0, x)$$

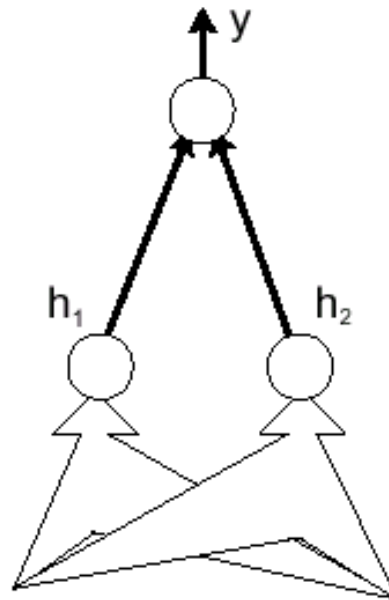
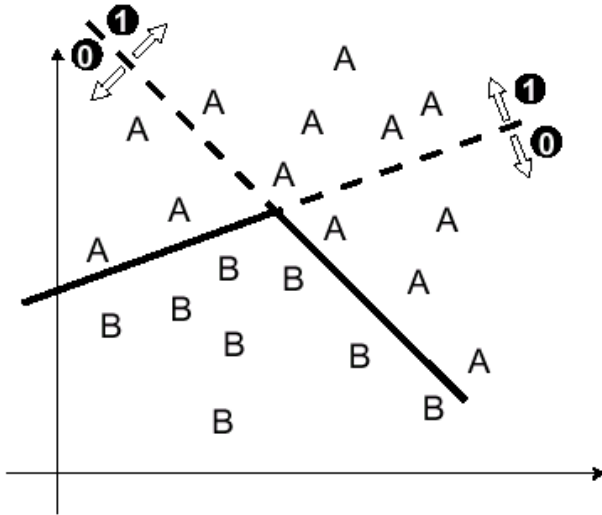
no es derivable



- Utiliza descenso por gradiente para actualizar el vector de pesos.

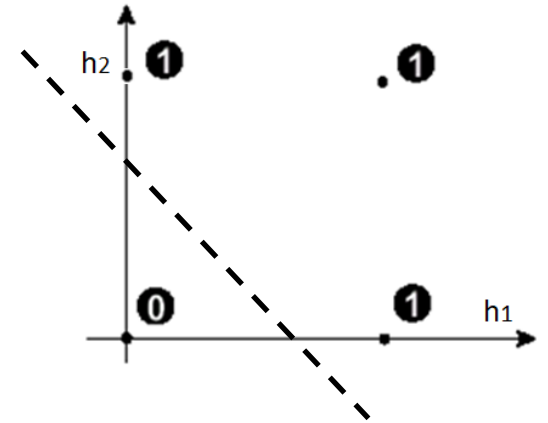
# Problema no separable linealmente

12



$h_1$	$h_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1

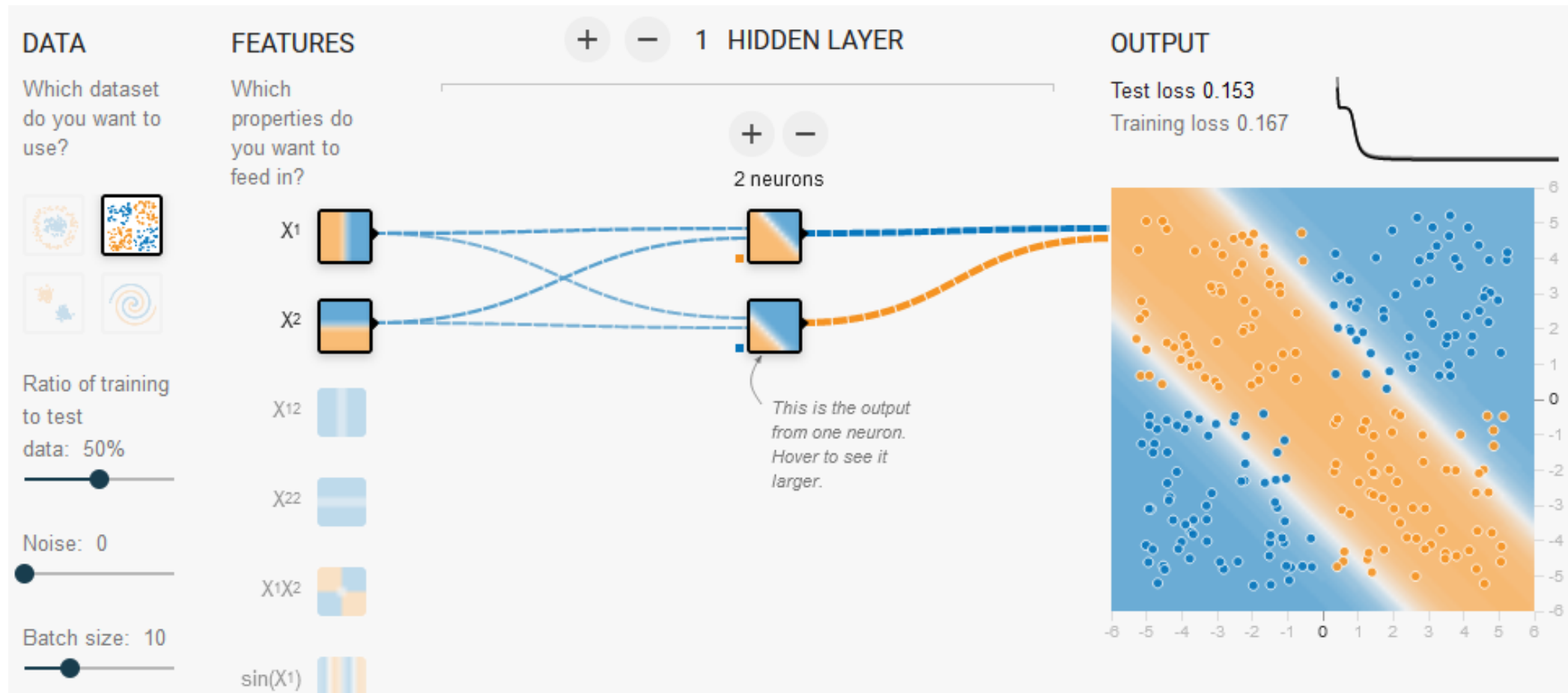
}  $A \leftrightarrow 1$



- Se busca obtener un algoritmo más general que permita integrar el aprendizaje entre las dos capas.

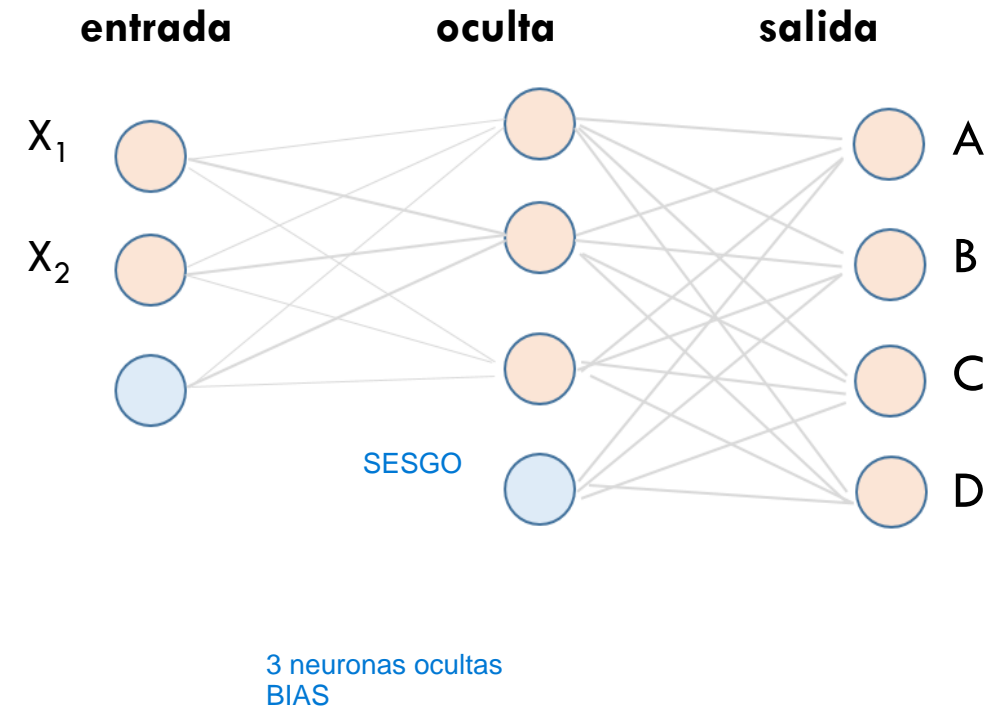
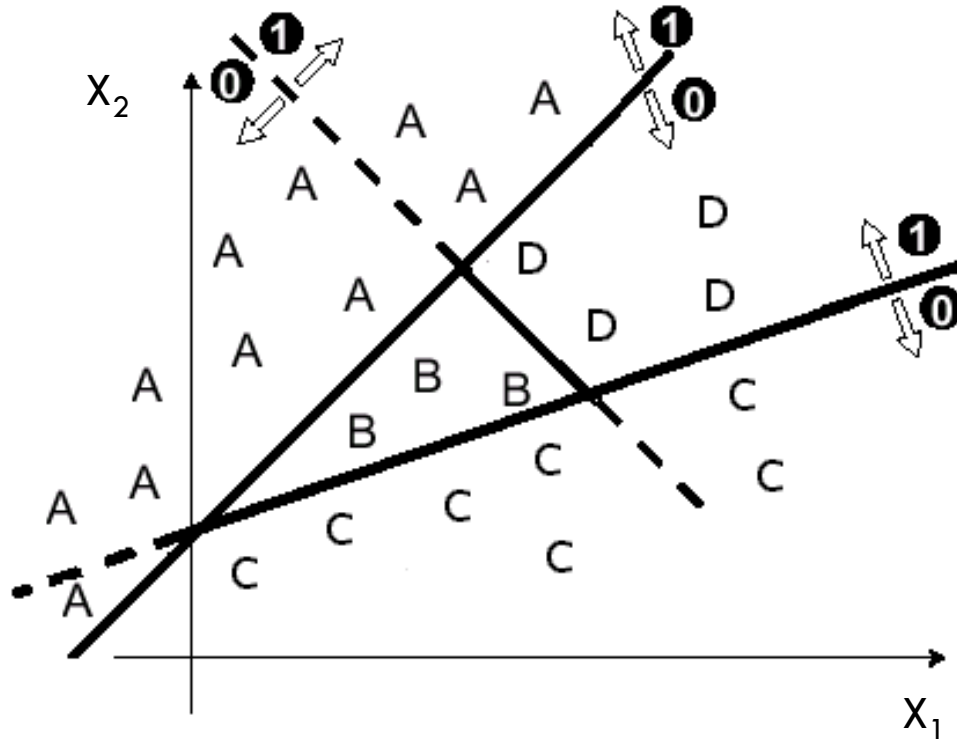
# Animación de una RN

## Tinker With a Neural Network Right Here in Your Browser



# Problema no separable linealmente

14



□ ¿Cuál es el tamaño de cada capa?

OCULTA: cuantos cortes en la capa de entrada, la cantidad de capas tiene que ser la menor necesaria

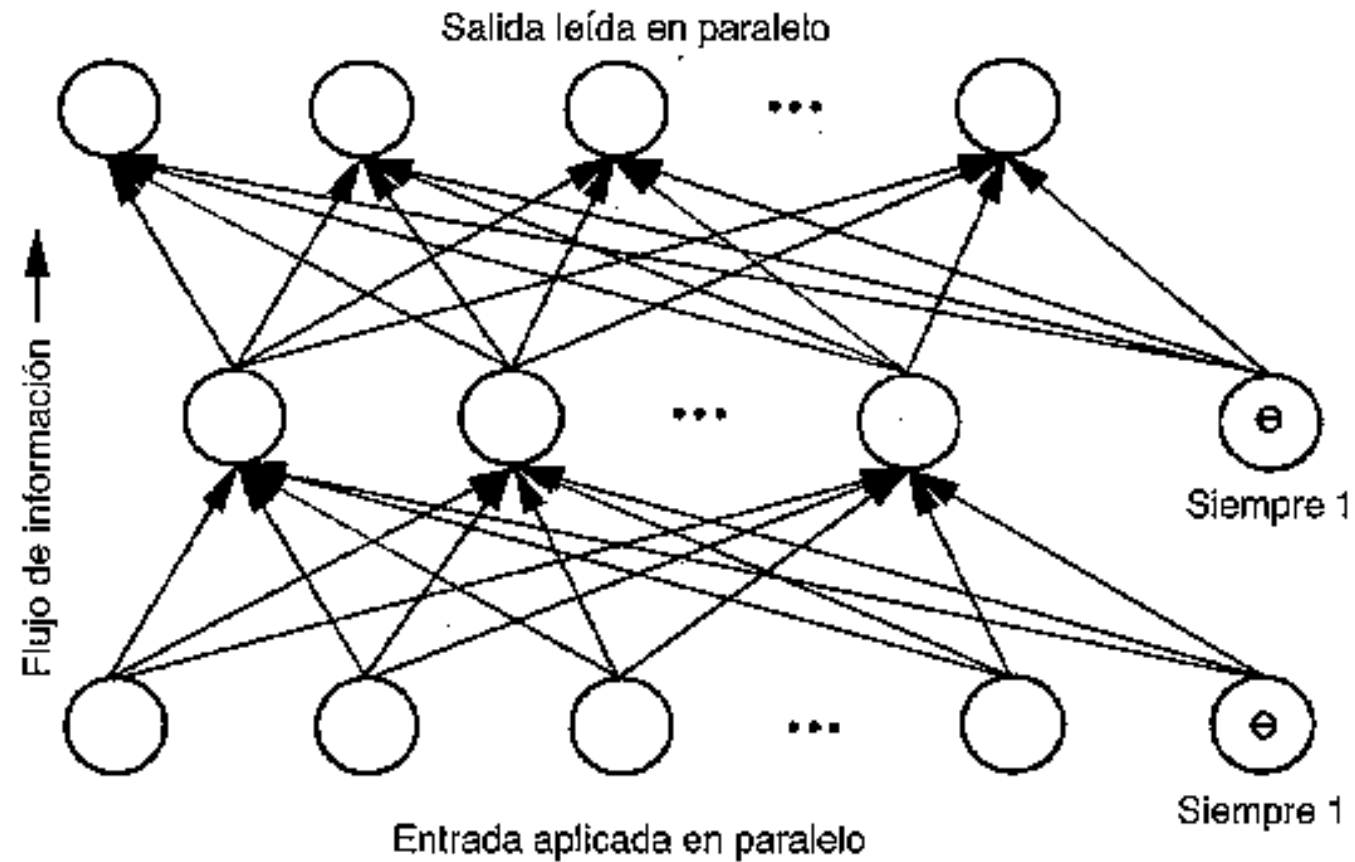
# Problema no separable linealmente

15

- La idea es aplicar un descenso en la dirección del gradiente sobre la superficie de error expresada como una función de los pesos.
- Deberán tenerse en cuenta los pesos de los arcos que unen AMBAS capas.
- Dado que el aprendizaje es supervisado, para los nodos de salida se conoce la respuesta esperada a cada entrada. Por lo tanto, puede aplicarse la **regla delta** vista para el Combinador Lineal y la Neurona No Lineal.

# Multiperceptrón - Arquitectura

16





# Algoritmo backpropagation

17

Dado el siguiente conjunto de vectores

$$\{(x_1, y_1), \dots, (x_p, y_p)\}$$

que son ejemplos de correspondencia funcional

$$y = \phi(x) \quad x \in R^N, y \in R^M$$

se busca entrenar la red para que aprenda una aproximación

$$y' = \phi'(x)$$

# Backpropagation. Capa Oculta

18

- Ejemplo de entrada

$$x_p = (x_{p1}, x_{p2}, \dots, x_{pN})^t$$

- Entrada neta de la j-ésima neurona de la capa oculta

$$neta_{pj}^h = \sum_{i=1}^n w_{ji}^h x_{pi} + \theta_j^h$$

- Salida de la j-ésima neurona de la capa oculta

$$i_{pj} = f_j^h(neta_{pj}^h)$$

# Backpropagation. Capa de Salida

19

- Entrada neta de la k-ésima neurona de la capa de salida

$$neta_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o$$

- Salida de la k-ésima neurona de la capa de salida

$$o_{pk} = f_k^o(neta_{pk}^o)$$

# Actualización de pesos

20

- Error en una sola unidad de la capa de salida

$$\delta_{pk} = (y_{pk} - o_{pk})$$

donde

- ▣  $y$  es la salida deseada
- ▣  $o$  es la salida real.
- ▣  $p$  se refiere al  $p$ -ésimo vector de entrada
- ▣  $k$  se refiere a la  $k$ -ésima unidad de salida

# Actualización de pesos

21

- Se busca minimizar

$$E_p = \frac{1}{2} \sum_{k=1}^M \delta_{pk}^2$$

$$E_p = \frac{1}{2} \sum_{k=1}^M (y_{pk} - o_{pk})^2$$

se tomará el valor negativo del gradiente

# Actualización de pesos

$$\frac{\partial E_p}{\partial w_{kj}^o} = -(y_{pk} - o_{pk}) \frac{\partial f_k^o}{\partial (neta_{pk}^o)} \frac{\partial (neta_{pk}^o)}{\partial w_{kj}^o}$$

$f_k^{o'}(neta_{pk}^o)$

$\frac{\partial}{\partial w_{kj}^o} \left( \sum_{j=1}^L w_{kj}^o i_{pj} + h_k^o \right) = i_{pj}$

$$\frac{\partial E_p}{\partial w_{kj}^o} = -(y_{pk} - o_{pk}) f_k^{o'}(neta_{pk}^o) i_{pj}$$

Salida de la neurona oculta j

Peso del arco que une la neurona j de la capa oculta y la neurona k de la capa de salida

# Actualización de pesos

- Por lo tanto, para la capa de salida se tiene

$$\delta_{pk}^o = (y_{pk} - o_{pk}) f_k^{o'}(neta_{pk}^o)$$

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \alpha \delta_{pk}^o i_{pj}$$

# Actualización de pesos

- Corrección para los pesos de los arcos entre la capa de entrada y la oculta

$$\Delta_p w_{ji}^h(t) = \alpha \delta_{pj}^h x_{pi}$$

serán de la forma:

$$\delta_{pj}^h = f_j^{h'}(neta_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o$$



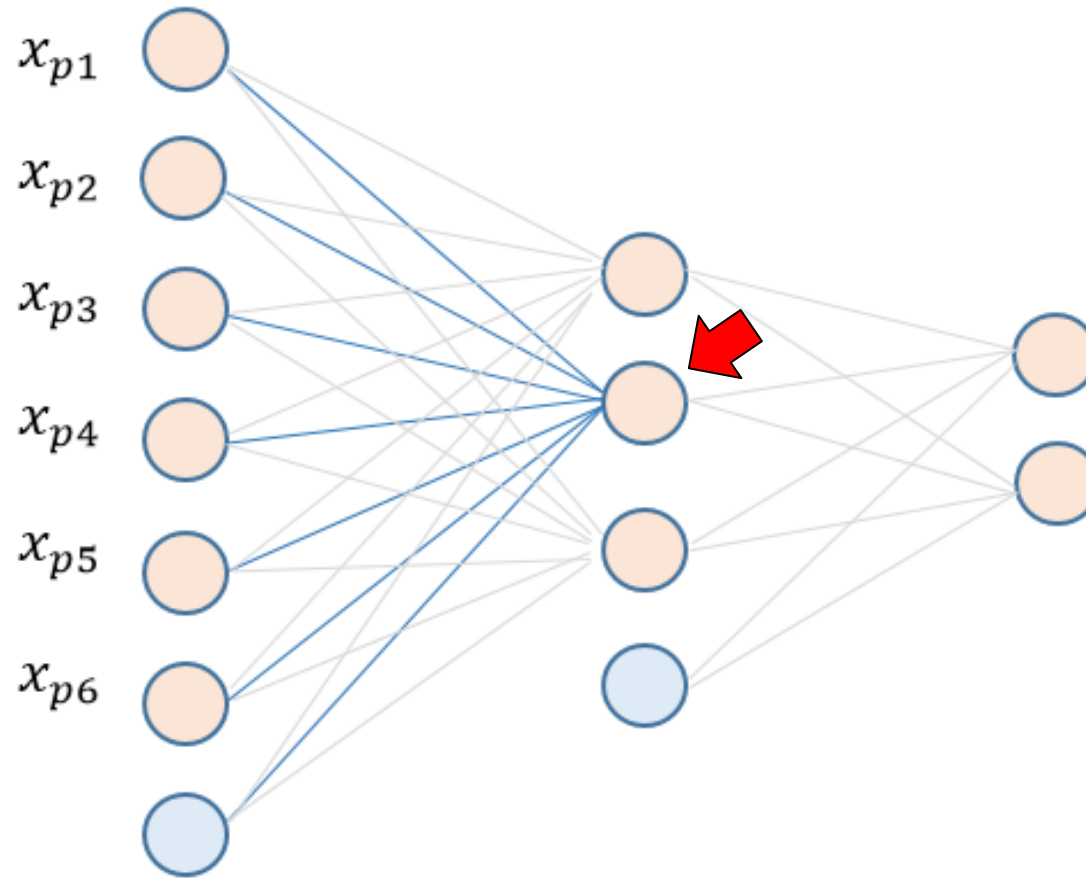
# Backpropagation. Entrenamiento

25

- Aplicar un vector de entrada y calcular su salida.
- Calcular el error.
- Determinar en qué dirección (+ o -) debe cambiarse los pesos para reducir el error.
- Determinar la cantidad en que es preciso cambiar cada peso.
- Corregir los pesos de las conexiones.
- Repetir los pasos anteriores para todos los ejemplos hasta reducir el error a un valor aceptable.

□ Propagar el ejemplo de entrada a través de la capa oculta

26

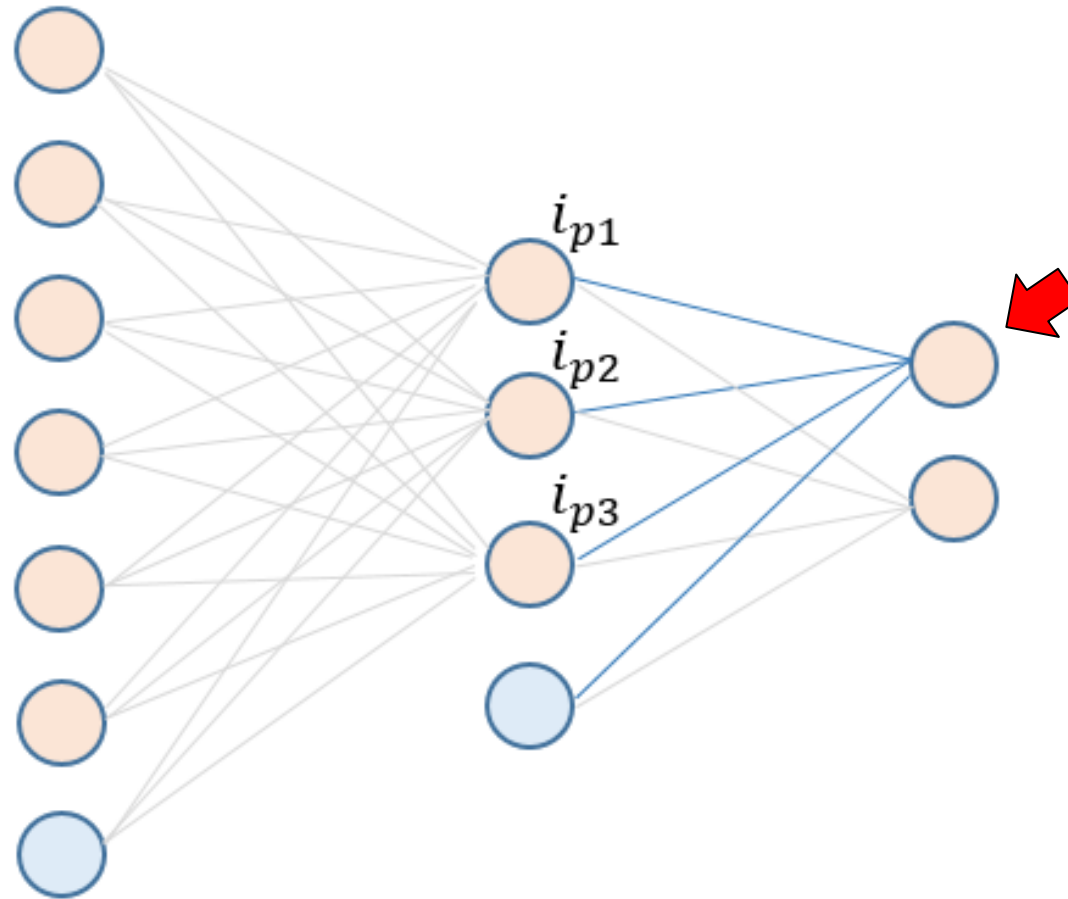


$$neta_{pj}^h = \sum_{i=1}^n w_{ji}^h x_{pi} + \theta_j^h$$

$$i_{pj} = f_j^h(neta_{pj}^h)$$

- Propagar las salidas de la capa oculta hacia la capa de salida

27

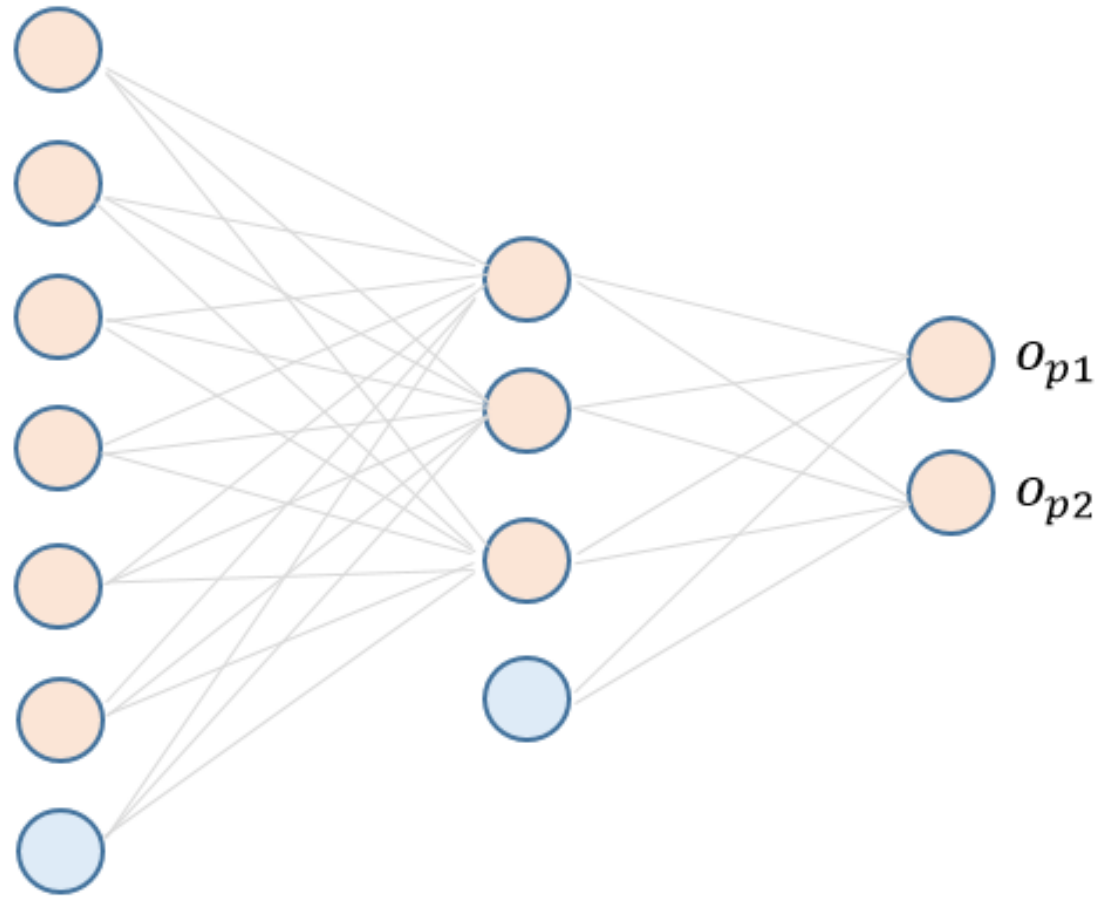


$$neta_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o$$

$$o_{pk} = f_k^o(neta_{pk}^o)$$

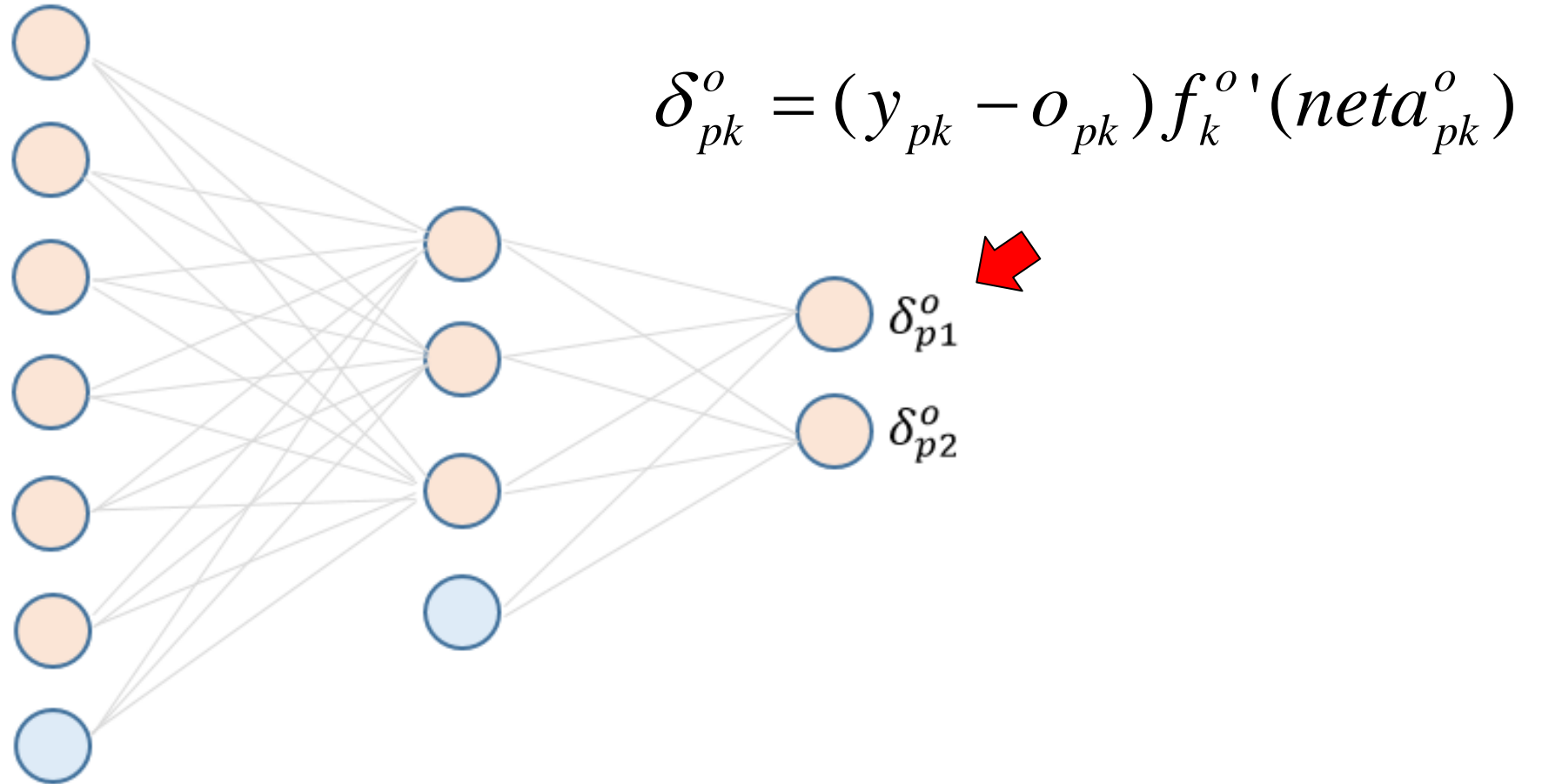
- Se obtienen los valores de salida

28



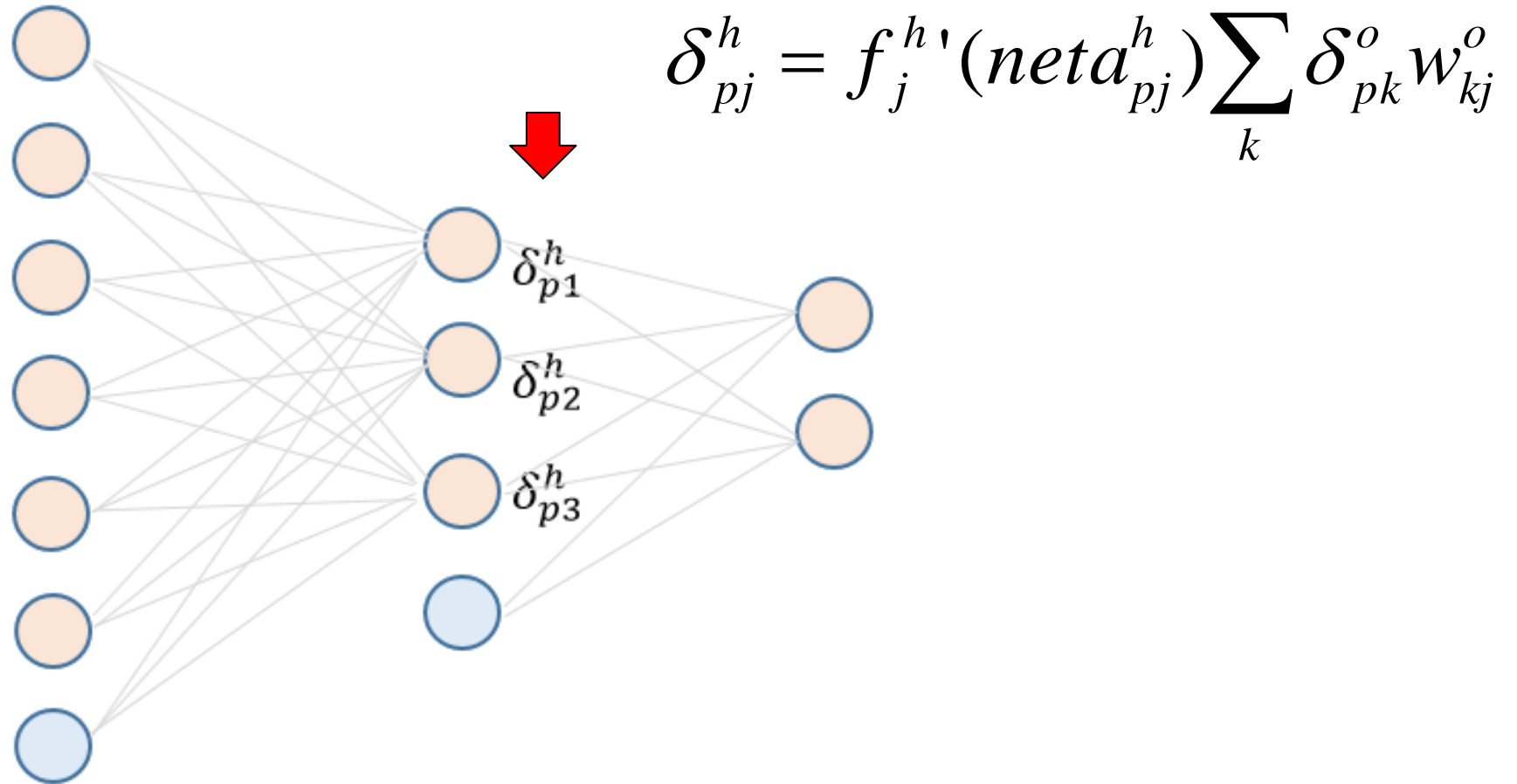
- Se calcula la corrección que se realizará al vector de pesos que llega a cada neurona de salida

29



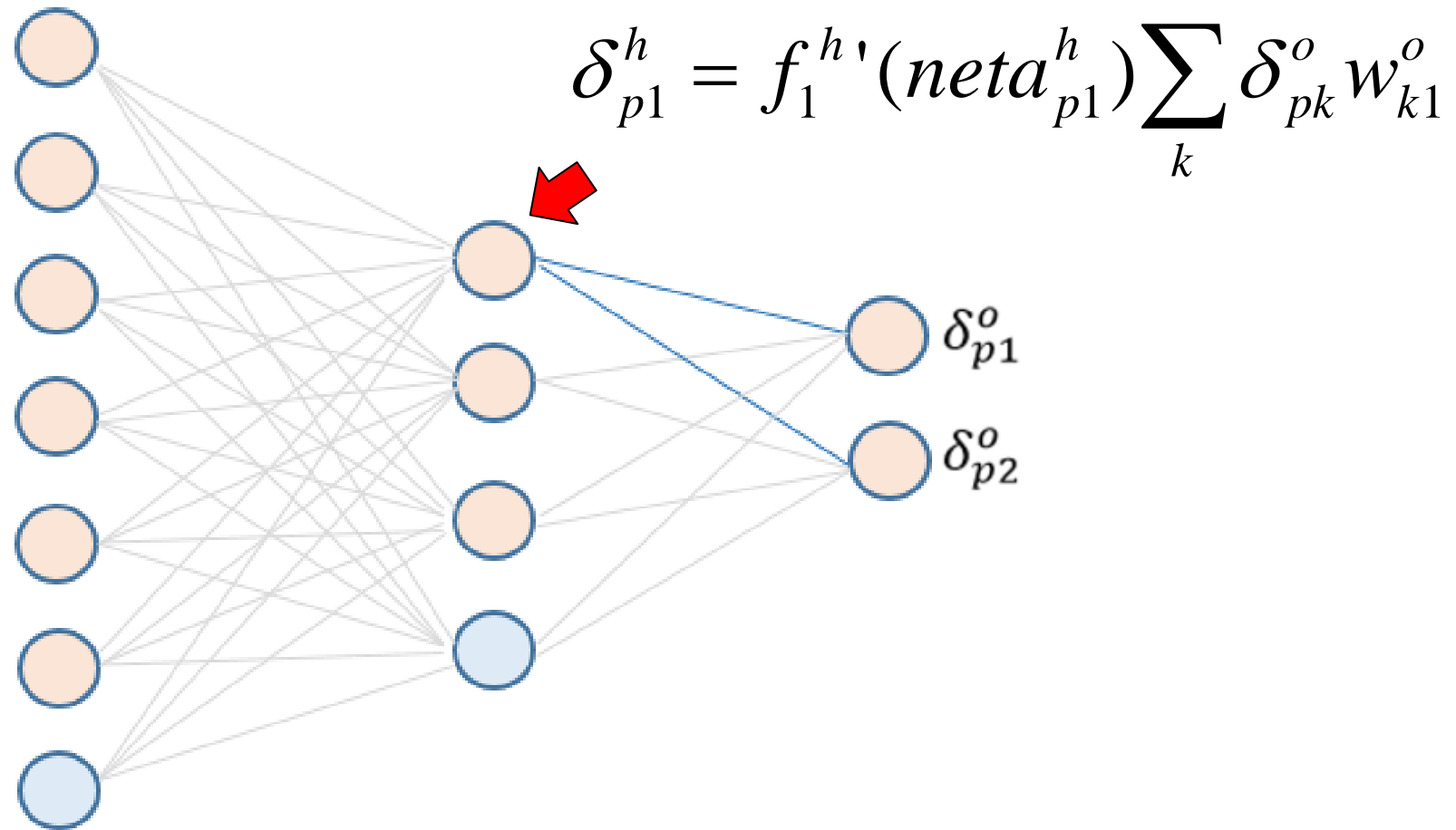
- Se calcula la corrección que se realizará al vector de pesos que llega a cada neurona oculta

30



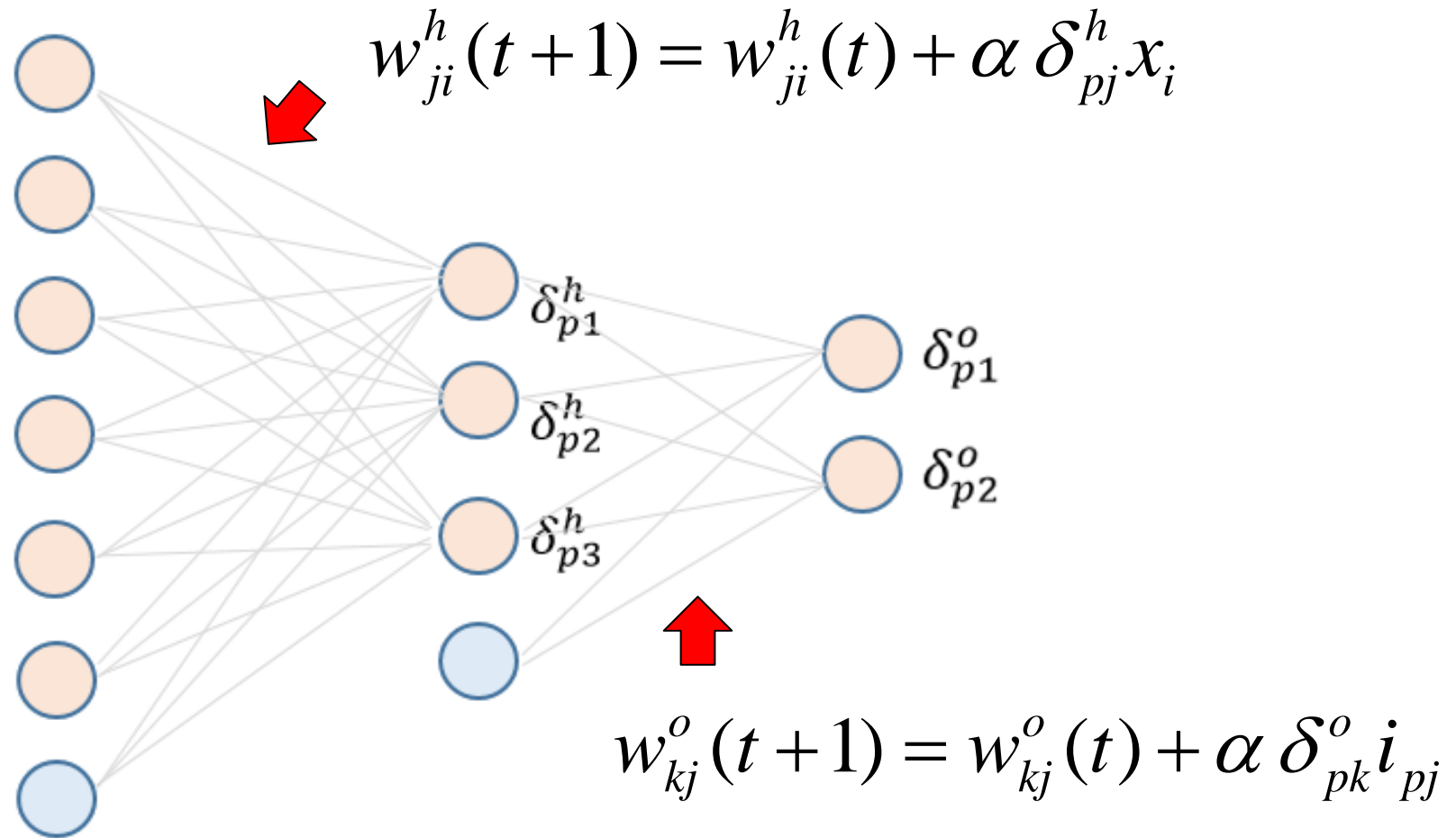
- Se calcula la corrección que se realizará al vector de pesos que llega a cada neurona de la capa oculta

31



- Se actualizan ambas matrices de pesos

33





# Backpropagation. Resumen

34

- Aplicar el vector de entrada

$$x_p = (x_{p1}, x_{p2}, \dots, x_{pN})^t$$

- Calcular los valores netos de las unidades de la capa oculta

$$neta_{pj}^h = \sum_{i=1}^n w_{ji}^h x_{pi} + \theta_j^h$$

- Calcular las salidas de la capa oculta

$$i_{pj} = f_j^h(neta_{pj}^h)$$

# Backpropagation. Resumen

35

- Calcular los valores netos de las unidades de la capa de salida

$$neta_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o$$

- Calcular las salidas

$$o_{pk} = f_k^o(neta_{pk}^o)$$

# Backpropagation. Resumen

36

- Calcular los términos de error para las unidades de salida

$$\delta_{pk}^o = (y_{pk} - o_{pk}) f_k^{o'}(neta_{pk}^o)$$

- Calcular los términos de error para las unidades ocultas

$$\delta_{pj}^h = f_j^{h'}(neta_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o$$

# Backpropagation. Resumen

37

- Se actualizan los pesos de la capa de salida

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \alpha \delta_{pk}^o i_{pj}$$

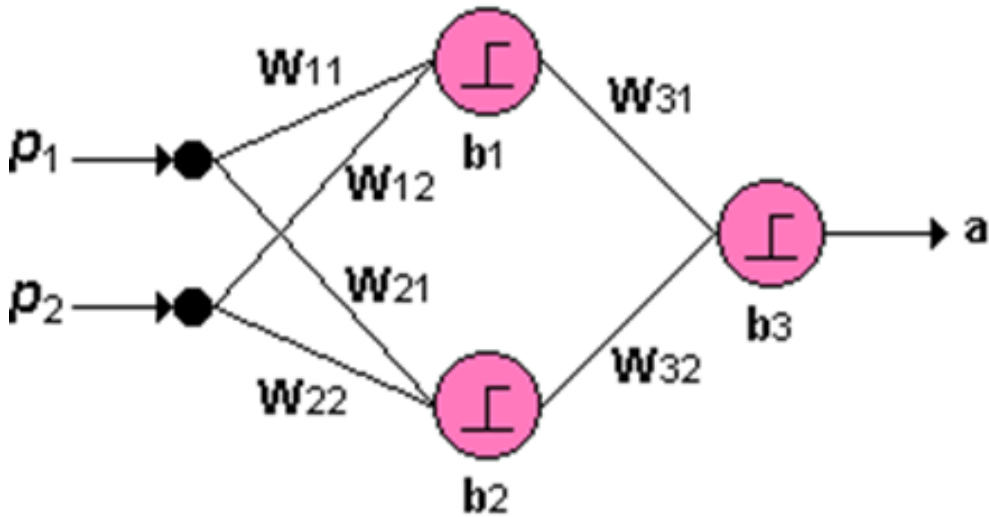
- Se actualizan los pesos de la capa oculta

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \alpha \delta_{pj}^h x_i$$

- Repetir hasta que el error resulte aceptable

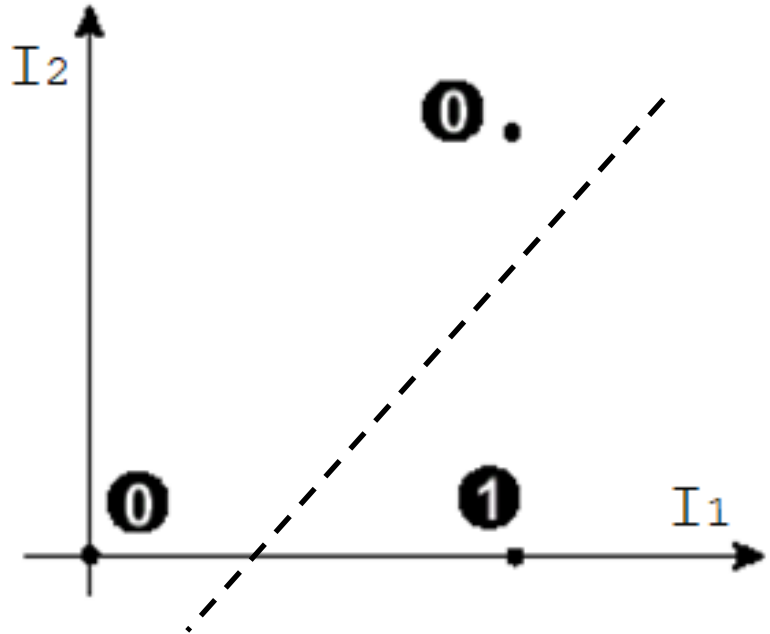
# XOR

38



MLP\_XOR.ipynb

$p_1$	$p_2$	$I_1$ (or)	$I_2$ (AND)	$a$
1	0	1	0	1
1	1	1	1	0
0	0	0	0	0
0	1	1	0	1



# Problema del XOR

39

```
import numpy as np
from graficaMLP import dibuPtos_y_2Rectas
from Funciones import evaluar, evaluarDerivada

X = np.array([ [-1, -1], [-1, 1], [1, -1], [1, 1]])
Y = np.array([-1, 1, 1, -1]).reshape(-1,1)

entradas = X.shape[1]
ocultas = 2
salidas = Y.shape[1]
```

```
In [13]: X
```

```
Out[13]:
```

```
array([[ -1,  -1],
       [ -1,   1],
       [  1,  -1],
       [  1,   1]])
```

```
In [14]: Y
```

```
Out[14]:
```

```
array([[ 1],
       [-1],
       [-1],
       [ 1]])
```

# Pesos iniciales

40

```
W1 = np.random.uniform(-0.5,0.5,[ocultas, entradas])
b1 = np.random.uniform(-0.5,0.5, [ocultas,1])
W2 = np.random.uniform(-0.5,0.5,[salidas, ocultas])
b2 = np.random.uniform(-0.5,0.5, [salidas,1])
```

```
In [17]: W1
Out[17]: array([[ -0.09705477, -0.48156505],
               [  0.14081924,  0.17185576]])

In [18]: b1
Out[18]: array([[ 0.28208473],
               [ 0.07973888]])

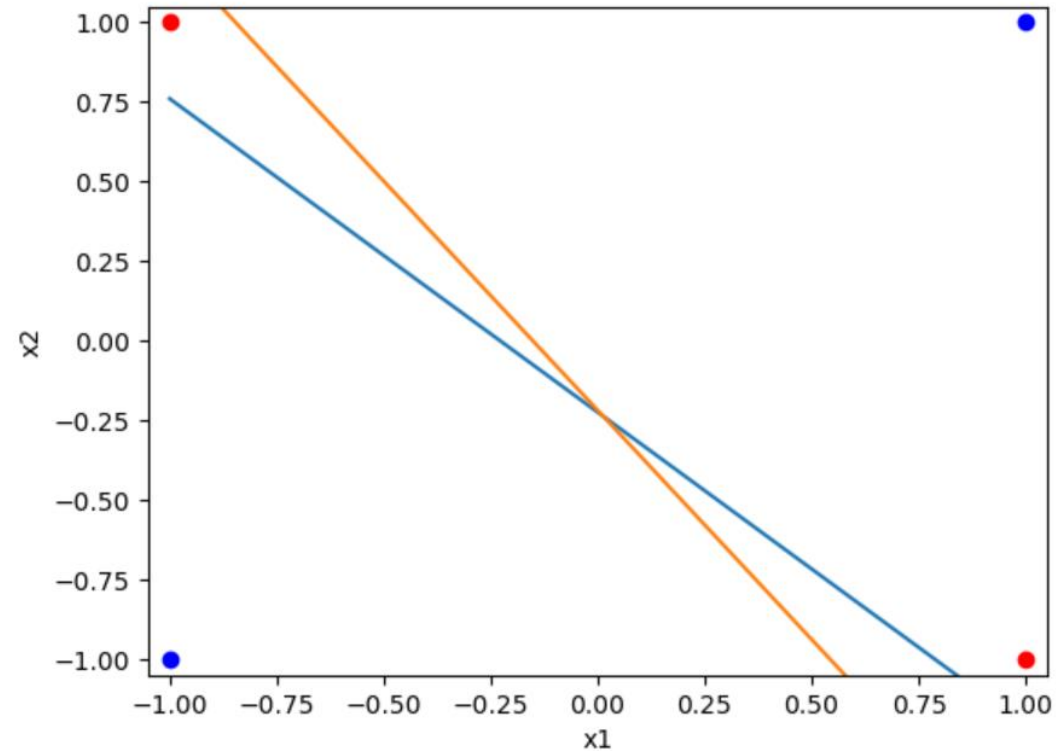
In [19]: W2
Out[19]: array([[ -0.48098277,  0.45515249]])

In [20]: b2
Out[20]: array([[ 0.15708682]])
```

# Graficar W1 y b1

41

```
ph = dibuPtos_y_2Rectas(X,Y, W1, b1, ph)
```





```
alfa = 0.15
CotaError = 0.001
MAX_ITERA = 300
ite = 0

while ( abs(ErrorAVG-ErrorAnt)>Cota) and ( ite < MAX_ITERA ):
    for p in range(len(P)):    #para cada ejemplo
        # propagar el ejemplo hacia adelante
        # calcular los errores en ambas capas
        # corregir los todos los pesos

    # Recalcular AVGError
    ite = ite + 1
    print(ite, AVGError)
    # Graficar las rectas
```

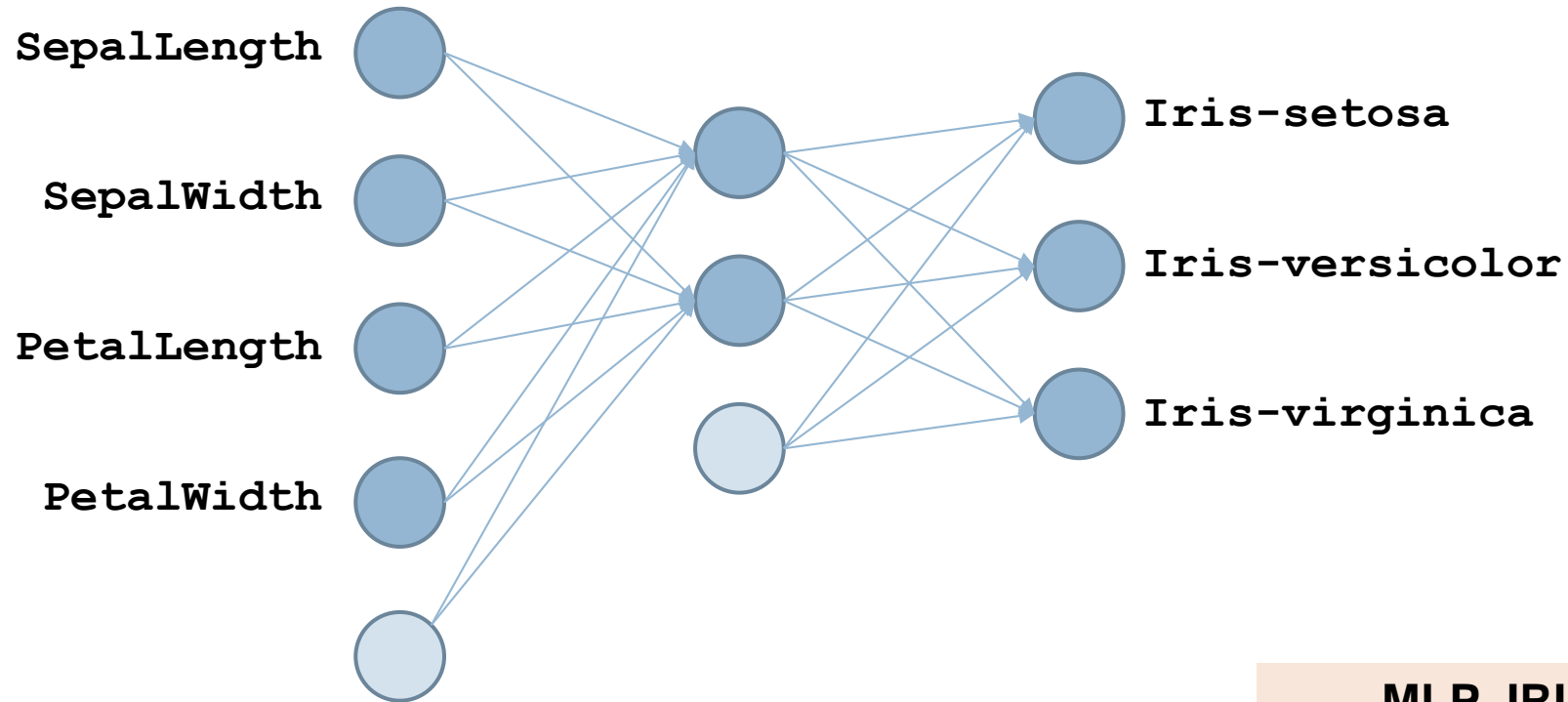
Ver  
**MLP\_XOR.ipynb**

# Ejemplo: Clasificación de flores de Iris

Id	sepalength	sepalwidth	petallength	petalwidth	class
1	5,1	3,5	1,4	0,2	Iris-setosa
2	4,9	3,0	1,4	0,2	Iris-setosa
...	...	...	...	...	...
95	5,6	2,7	4,2	1,3	Iris-versicolor
96	5,7	3,0	4,2	1,2	Iris-versicolor
97	5,7	2,9	4,2	1,3	Iris-versicolor
...	...	...	...	...	...
149	6,2	3,4	5,4	2,3	Iris-virginica
150	5,9	3,0	5,1	1,8	Iris-virginica

<https://archive.ics.uci.edu/ml/datasets/Iris>

# Ejemplo: Clasificación de flores de Iris



MLP\_IRIS.ipynb

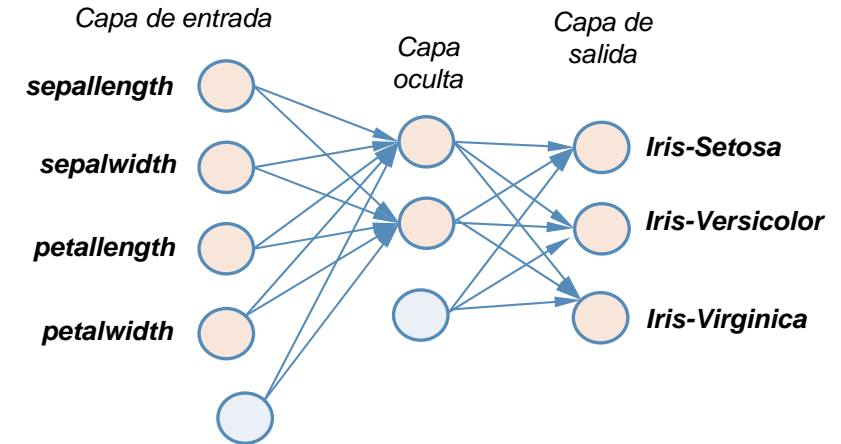
# Clasificación de flores de Iris

**X**

```
[[-1.73,-0.05,-1.38,-1.31],  
 [-0.37,-1.62, 0.22, 0.18],  
 [ 1.11,-0.05, 0.93, 1.54],  
 [-0.99, 0.39,-1.44,-1.31],  
 [ 1.73, 1.29, 1.46, 1.81]]
```

**Y**

```
[[1,0,0],  
 [0,1,0],  
 [0,0,1],  
 [1,0,0],  
 [0,0,1]]
```



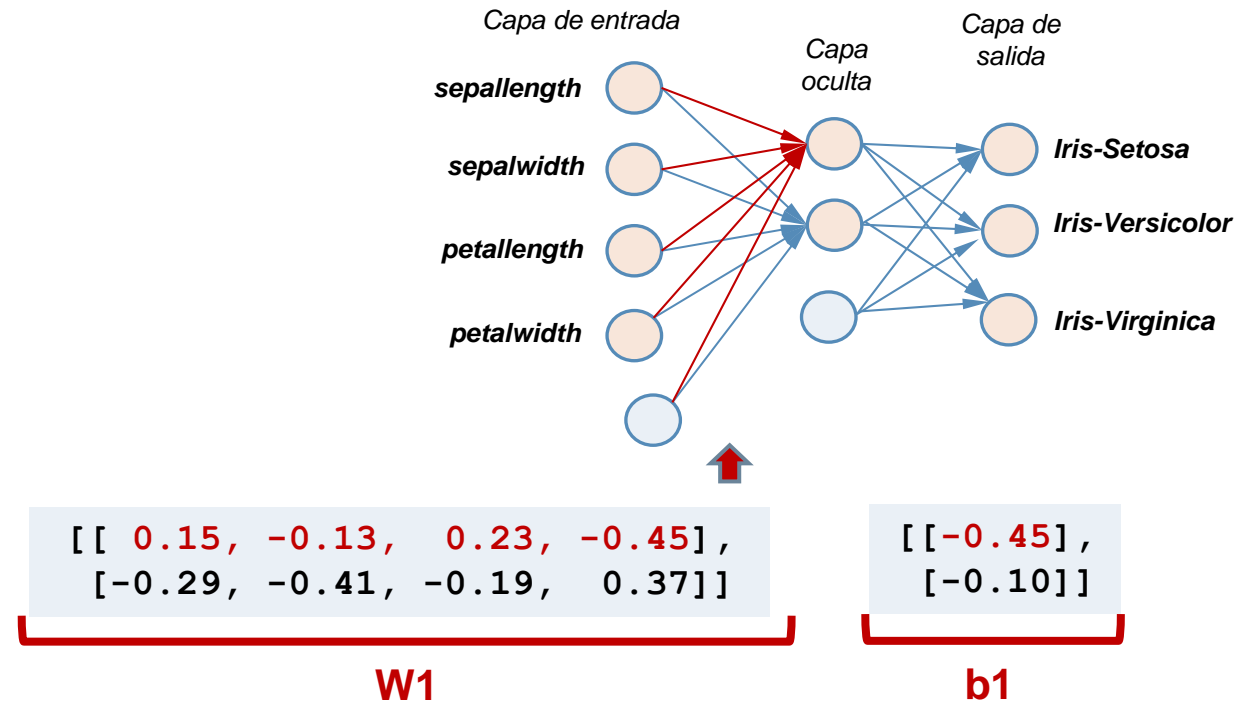
# Clasificación de flores de Iris

**X**

```
[[-1.73, -0.05, -1.38, -1.31],  
 [-0.37, -1.62, 0.22, 0.18],  
 [ 1.11, -0.05, 0.93, 1.54],  
 [-0.99, 0.39, -1.44, -1.31],  
 [ 1.73, 1.29, 1.46, 1.81]]
```

**Y**

```
[[1, 0, 0],  
 [0, 1, 0],  
 [0, 0, 1],  
 [1, 0, 0],  
 [0, 0, 1]]
```



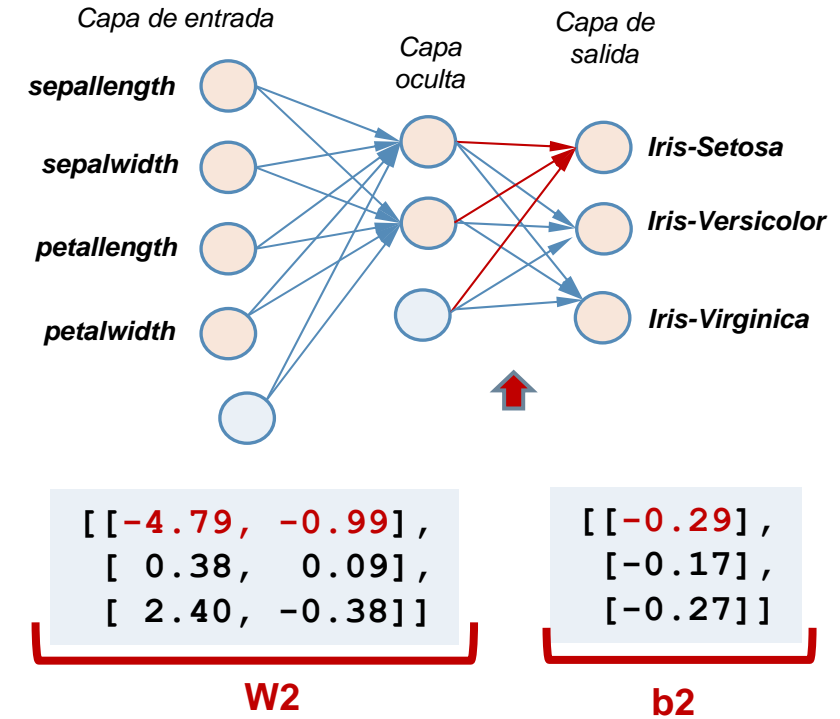
# Clasificación de flores de Iris

**X**

```
[[-1.73, -0.05, -1.38, -1.31],  
 [-0.37, -1.62, 0.22, 0.18],  
 [ 1.11, -0.05, 0.93, 1.54],  
 [-0.99, 0.39, -1.44, -1.31],  
 [ 1.73, 1.29, 1.46, 1.81]]
```


**Y**

```
[[1, 0, 0],  
 [0, 1, 0],  
 [0, 0, 1],  
 [1, 0, 0],  
 [0, 0, 1]]
```



# Clasificación de flores de Iris

**X**

$\left[ \begin{bmatrix} -1.73, -0.05, -1.38, -1.31 \end{bmatrix}, \right.$   **Y**

$\left[ \begin{bmatrix} 1, 0, 0 \end{bmatrix}, \right.$

$\left[ \begin{bmatrix} -0.37, -1.62, 0.22, 0.18 \end{bmatrix}, \right.$

$\left[ \begin{bmatrix} 1.11, -0.05, 0.93, 1.54 \end{bmatrix}, \right.$

$\left[ \begin{bmatrix} -0.99, 0.39, -1.44, -1.31 \end{bmatrix}, \right.$

$\left[ \begin{bmatrix} 1.73, 1.29, 1.46, 1.81 \end{bmatrix} \right]$

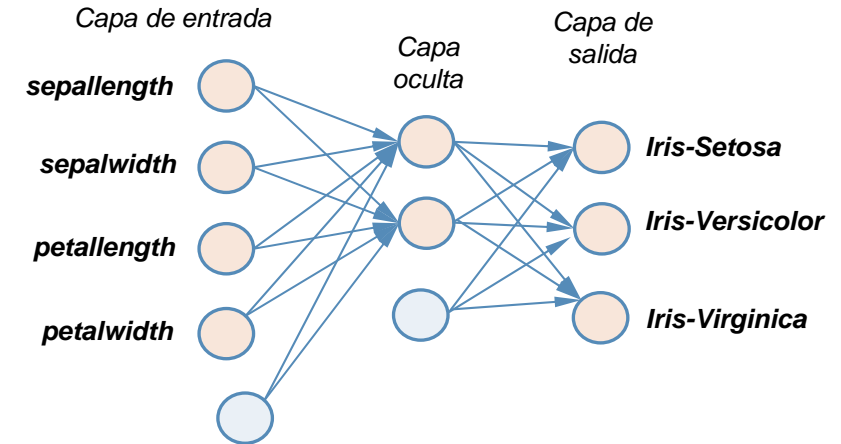
$\left[ \begin{bmatrix} 0, 1, 0 \end{bmatrix}, \right.$

$\left[ \begin{bmatrix} 0, 0, 1 \end{bmatrix}, \right.$

$\left[ \begin{bmatrix} 1, 0, 0 \end{bmatrix}, \right.$

$\left[ \begin{bmatrix} 0, 0, 1 \end{bmatrix} \right]$

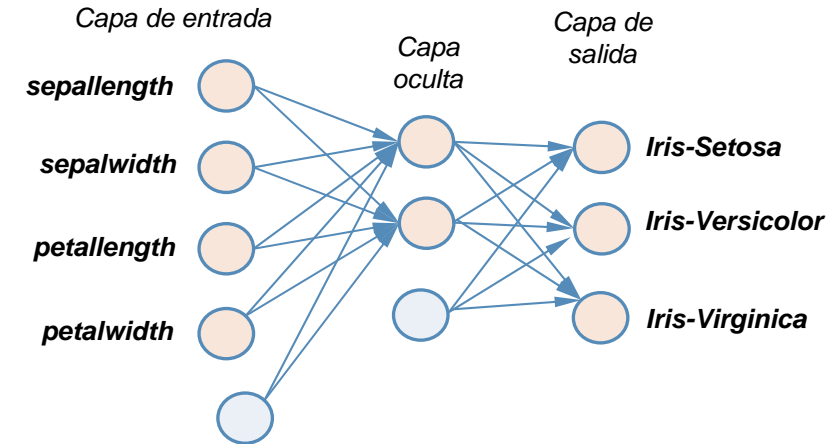
FunH='tanh' ; FunO='sigmoid'



Ingresar el primer ejemplo a la red y  
calcular su salida

# Calculando la salida de la capa oculta

$$\begin{array}{c} \mathbf{X} \\ \left[ \begin{array}{cccc} [-1.73, -0.05, -1.38, -1.31], \\ [-0.37, -1.62, 0.22, 0.18], \\ [1.11, -0.05, 0.93, 1.54], \\ [-0.99, 0.39, -1.44, -1.31], \\ [1.73, 1.29, 1.46, 1.81] \end{array} \right] \end{array} \quad \leftarrow \quad \begin{array}{c} \mathbf{Y} \\ \left[ \begin{array}{ccc} [1, 0, 0], \\ [0, 1, 0], \\ [0, 0, 1], \\ [1, 0, 0], \\ [0, 0, 1] \end{array} \right] \end{array}$$



□ Salida de la capa oculta

$$\text{netasH} = \mathbf{W1} * \mathbf{x.T} + \mathbf{b1}$$

$$\begin{array}{c} \left[ \begin{array}{cccc} 0.15, & -0.13, & 0.23, & -0.45 \\ -0.29, & -0.41, & -0.19, & 0.37 \end{array} \right] \\ \mathbf{W1} \end{array} * \begin{array}{c} \mathbf{x^T} \\ \left[ \begin{array}{c} [-1.73], \\ [-0.05], \\ [-1.38], \\ [-1.31] \end{array} \right] \end{array} + \begin{array}{c} \left[ \begin{array}{c} [-0.45], \\ [-0.10] \end{array} \right] \\ \mathbf{b1} \end{array} = \begin{array}{c} \left[ \begin{array}{c} [-0.4309] \\ [0.1997] \end{array} \right] \end{array}$$

FunH='tanh' ; FunO='sigmoid'

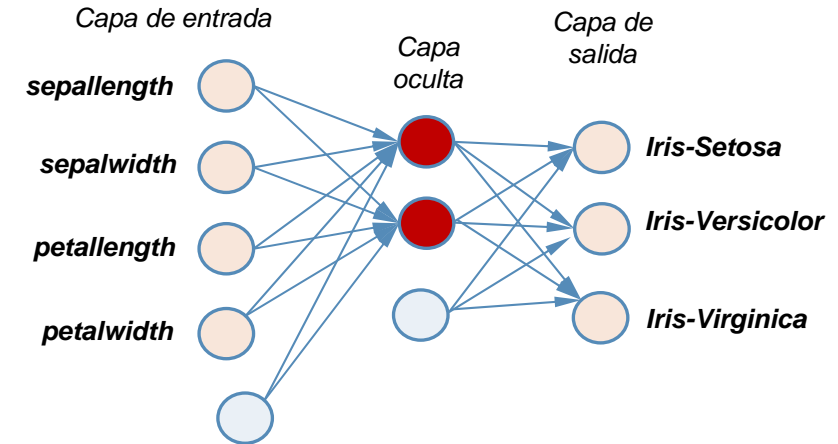
$$\text{netas}_{pj}^h = \sum_{i=1}^n w_{ji}^h x_{pi} + \theta_j^h$$

$$i_{pj} = f_j^h(\text{netas}_{pj}^h)$$



# Calculando la salida de la capa oculta

$$\begin{array}{c} \mathbf{X} \\ \left[ \begin{array}{cccc} [-1.73, -0.05, -1.38, -1.31], \\ [-0.37, -1.62, 0.22, 0.18], \\ [1.11, -0.05, 0.93, 1.54], \\ [-0.99, 0.39, -1.44, -1.31], \\ [1.73, 1.29, 1.46, 1.81] \end{array} \right] \end{array} \quad \leftarrow \quad \begin{array}{c} \mathbf{Y} \\ \left[ \begin{array}{ccc} [1, 0, 0], \\ [0, 1, 0], \\ [0, 0, 1], \\ [1, 0, 0], \\ [0, 0, 1] \end{array} \right] \end{array}$$



□ Salida de la capa oculta

$$\text{netasH} = \mathbf{W1} * \mathbf{x.T} + \mathbf{b1}$$

$$\begin{array}{c} \left[ \begin{array}{cccc} 0.15, & -0.13, & 0.23, & -0.45 \\ -0.29, & -0.41, & -0.19, & 0.37 \end{array} \right] \quad * \quad \begin{array}{c} \mathbf{x^T} \\ \left[ \begin{array}{c} [-1.73], \\ [-0.05], \\ [-1.38], \\ [-1.31] \end{array} \right] \end{array} \quad + \quad \begin{array}{c} \mathbf{b1} \\ \left[ \begin{array}{c} [-0.45], \\ [-0.10] \end{array} \right] \end{array} \quad = \quad \left[ \begin{array}{c} [-0.4309] \\ [0.1997] \end{array} \right] \end{array}$$

FunH='tanh' ; FunO='sigmoid'

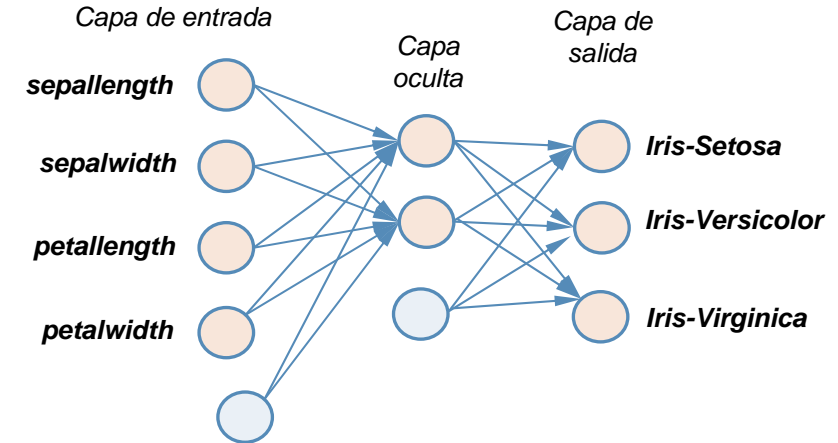
$$netas_{pj}^h = \sum_{i=1}^n w_{ji}^h x_{pi} + \theta_j^h$$

$$\text{salidasH} = 2 / (1 + \text{np.exp}(-\text{netasH})) - 1 = \left[ \begin{array}{c} [-0.21217712] \\ [0.09951948] \end{array} \right]$$

$$i_{pj} = f_j^h(\text{netas}_{pj}^h)$$

# Calculando la salida de la red (capa de salida)

X	Y
<code>[[-1.73, -0.05, -1.38, -1.31],</code>	<code>[[1, 0, 0],</code>
<code>[-0.37, -1.62, 0.22, 0.18],</code>	<code>[0, 1, 0],</code>
<code>[ 1.11, -0.05, 0.93, 1.54],</code>	<code>[0, 0, 1],</code>
<code>[-0.99, 0.39, -1.44, -1.31],</code>	<code>[1, 0, 0],</code>
<code>[ 1.73, 1.29, 1.46, 1.81]]</code>	<code>[0, 0, 1]]</code>



## Salida de red

`netasO = W2 * salidasH + b2`

**FunH='tanh' ; FunO='sigmoid'**

W2	*	salidasH	+	b2	=	netasO
<code>[[-4.79, -0.99],</code> <code>[ 0.38, 0.09],</code> <code>[ 2.40, -0.38]]</code>		<code>[[-0.21217712]</code> <code>[ 0.09951948]]</code>		<code>[[-0.29],</code> <code>[-0.17],</code> <code>[-0.27]]</code>		<code>[ [ 0.62780411]</code> <code>[-0.24167055]</code> <code>[-0.81704249]]</code>

$$neta_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o$$

$$o_{pk} = f_k^o(neta_{pk}^o)$$

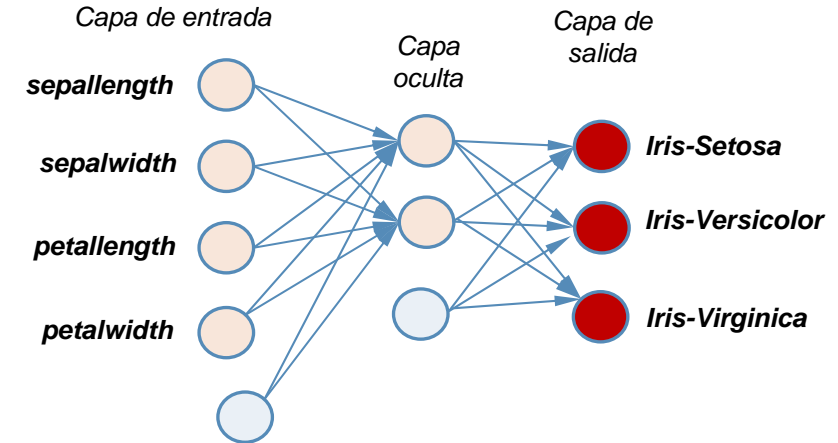
# Calculando la salida de la red (capa de salida)

**X**

```
[[ -1.73, -0.05, -1.38, -1.31],
 [ -0.37, -1.62,  0.22,  0.18],
 [  1.11, -0.05,  0.93,  1.54],
 [ -0.99,  0.39, -1.44, -1.31],
 [  1.73,  1.29,  1.46,  1.81]]
```

**Y**

```
[[1, 0, 0],
 [0, 1, 0],
 [0, 0, 1],
 [1, 0, 0],
 [0, 0, 1]]
```



## Salida de red

```
netasO = W2 * salidasH + b2
```

FunH='tanh' ; FunO='sigmoid'

<pre>[[ -4.79, -0.99],  [  0.38,  0.09],  [  2.40, -0.38]]</pre> <p><b>W2</b></p>	<p><b>salidasH</b></p> <pre>[[ -0.21217712]  [  0.09951948]]</pre>	<p>+</p>	<pre>[[ -0.29],  [ -0.17],  [ -0.27]]</pre> <p><b>b2</b></p>	<p>=</p>	<p><b>netasO</b></p> <pre>[[ 0.62780411]  [-0.24167055]  [-0.81704249]]</pre>
---	--	----------	--	----------	---

$$neta_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o$$

$$o_{pk} = f_k^o(neta_{pk}^o)$$

```
salidasO = 1 / (1+np.exp(-netasO)) =
```

```
[[0.65199139]
 [0.43987471]
 [0.30639182]]
```

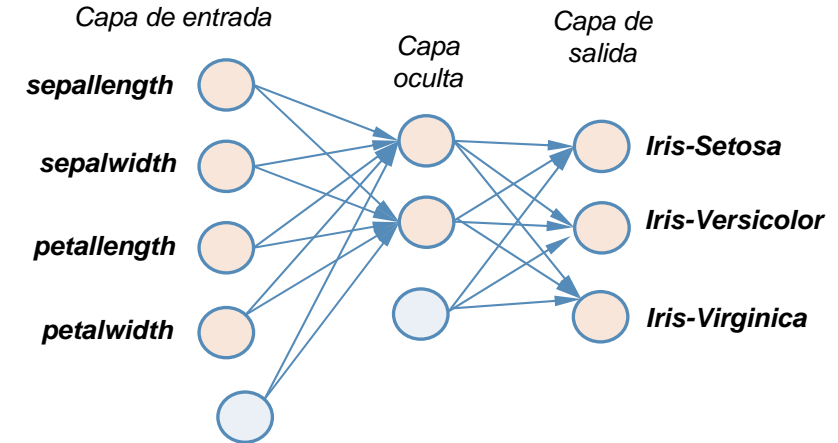
# Error de la capa de salida

X	Y
<code>[[-1.73, -0.05, -1.38, -1.31],</code>	<code>[[1, 0, 0],</code> ←
<code>[-0.37, -1.62, 0.22, 0.18],</code>	<code>[0, 1, 0],</code>
<code>[ 1.11, -0.05, 0.93, 1.54],</code>	<code>[0, 0, 1],</code>
<code>[-0.99, 0.39, -1.44, -1.31],</code>	<code>[1, 0, 0],</code>
<code>[ 1.73, 1.29, 1.46, 1.81]]</code>	<code>[0, 0, 1]]</code>

- Error en la respuesta de la red para este ejemplo

`ErrorSalida = y.T - salidasO`

<code>ErrorSalida =</code>	<code>[[1.0]</code>		<code>[[0.65199139]</code>		<code>[[ 0.34800861]</code>
	<code>[0.0]</code>	<code>-</code>	<code>[0.43987471]</code>	<code>=</code>	<code>[-0.43987471]</code>
	<code>[0.0]]</code>		<code>[0.30639182]]</code>		<code>[-0.30639182]]</code>
	↑		<code>salidasO</code>		



`FunH='tanh' ; FunO='sigmoid'`

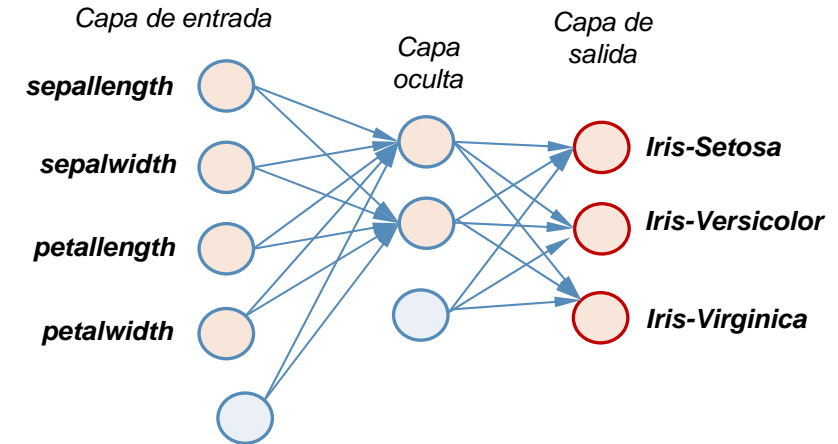
# Factores de corrección de los pesos

**X**

```
[[ -1.73, -0.05, -1.38, -1.31],
 [ -0.37, -1.62,  0.22,  0.18],
 [  1.11, -0.05,  0.93,  1.54],
 [ -0.99,  0.39, -1.44, -1.31],
 [  1.73,  1.29,  1.46,  1.81]]
```

**Y**

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



FunH='tanh' ; FunO='sigmoid'

## Factores para corregir W2 y b2

```
deltaO = ErrorSalida .* derivada_FunO
```

```
deltaO = [[ 0.34800861]
 [-0.43987471]
 [-0.30639182]] .* [[0.22689862]
 [0.24638495]
 [0.21251587]] = [[ 0.07896267]
 [-0.10837851]
 [-0.06511312]]
```

salidasO\*(1-salidasO)

$$\delta_{pk}^o = (y_{pk} - o_{pk}) f_k^{o'}(neta_{pk}^o)$$

# Factores de corrección de los pesos

**X**

```
[[ -1.73, -0.05, -1.38, -1.31],
 [ -0.37, -1.62,  0.22,  0.18],
 [  1.11, -0.05,  0.93,  1.54],
 [ -0.99,  0.39, -1.44, -1.31],
 [  1.73,  1.29,  1.46,  1.81]]
```

**Y**

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```

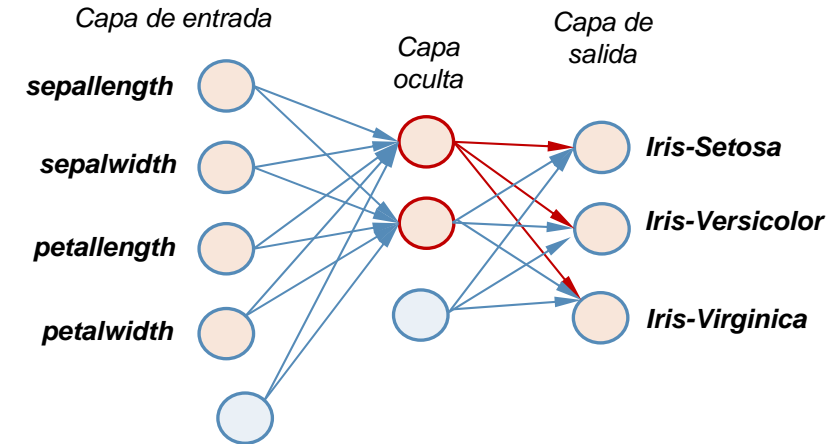
## Factores para corregir W1 y b1

`(1-salidasH**2)`

`deltaH = deriv_FunH .* (W2.T @ deltaO)`

`deltaH =`  $\begin{bmatrix} 0.95498087 \\ 0.99009587 \end{bmatrix} \cdot \begin{bmatrix} -4.79 & 0.38 & 2.4 \\ -0.99 & 0.09 & -0.38 \end{bmatrix} @ \begin{bmatrix} 0.07896267 \\ -0.10837851 \\ -0.06511312 \end{bmatrix}$

`deltaH =`  $\begin{bmatrix} -0.54976963 \\ -0.06255834 \end{bmatrix}$



`FunH='tanh' ; FunO='sigmoid'`

$$\delta_{pj}^h = f_j^{h'}(neta_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o$$

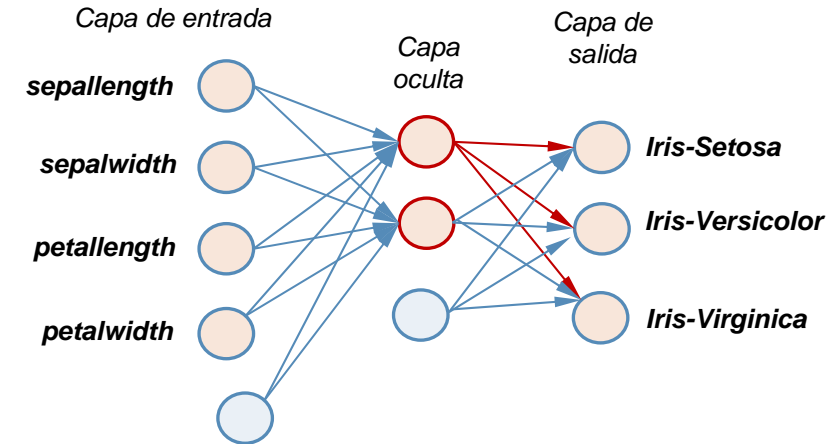
# Factores de corrección de los pesos

**X**

```
[[ -1.73, -0.05, -1.38, -1.31],
 [ -0.37, -1.62,  0.22,  0.18],
 [  1.11, -0.05,  0.93,  1.54],
 [ -0.99,  0.39, -1.44, -1.31],
 [  1.73,  1.29,  1.46,  1.81]]
```

**Y**

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



FunH='tanh' ; FunO='sigmoid'

## Factores para corregir W1 y b1

(1-salidasH\*\*2)

$\text{deltaH} = \text{deriv\_FunH} .* (\text{W2.T} @ \text{deltaO})$

$\text{deltaH} = \begin{bmatrix} 0.95498087 \\ 0.99009587 \end{bmatrix} .* \begin{bmatrix} -4.79 & 0.38 & 2.4 \\ -0.99 & 0.09 & -0.38 \end{bmatrix} @ \begin{bmatrix} 0.07896267 \\ -0.10837851 \\ -0.06511312 \end{bmatrix}$

$\text{deltaH} = \begin{bmatrix} -0.54976963 \\ -0.06255834 \end{bmatrix}$

Note que las derivadas de ambas funciones de activación sigmoides (derivada\_FunO y deriv\_FunH) son siempre positivas y actúan como factor de escala

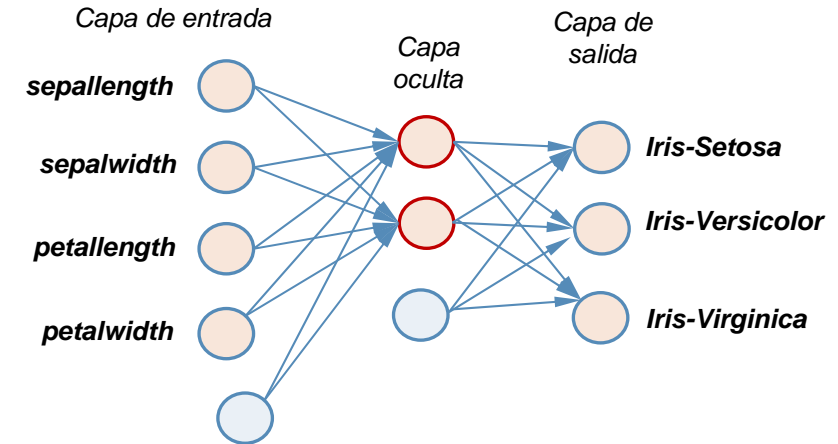
# Corrigiendo de los pesos

**X**

```
[[-1.73, -0.05, -1.38, -1.31],
 [-0.37, -1.62, 0.22, 0.18],
 [ 1.11, -0.05, 0.93, 1.54],
 [-0.99, 0.39, -1.44, -1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

**Y**

```
[[1, 0, 0],
 [0, 1, 0],
 [0, 0, 1],
 [1, 0, 0],
 [0, 0, 1]]
```



FunH='tanh' ; FunO='sigmoid'

## □ Modificación de W2 y b2

$W2 = W2 + \text{alfa} * \text{deltaO} @ \text{salidasH.T}$

$$W2 = \begin{bmatrix} -4.79 & -0.99 \\ 0.38 & 0.09 \\ 2.40 & -0.38 \end{bmatrix} + \text{alfa} * \begin{bmatrix} 0.07896267 \\ -0.10837851 \\ -0.06511312 \end{bmatrix} @ \begin{bmatrix} -0.212177 & 0.099519 \end{bmatrix} = \begin{bmatrix} -4.79 & -0.99 \\ 0.38 & 0.09 \\ 2.4 & -0.38 \end{bmatrix}$$

$$b2 = b2 + \text{alfa} * \text{deltaO} = \begin{bmatrix} -0.29 \\ -0.17 \\ -0.27 \end{bmatrix} + \text{alfa} * \begin{bmatrix} 0.07896267 \\ -0.10837851 \\ -0.06511312 \end{bmatrix} = \begin{bmatrix} -0.28 \\ -0.18 \\ -0.28 \end{bmatrix}$$



# Corrigiendo de los pesos

**X**

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

**Y**

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```

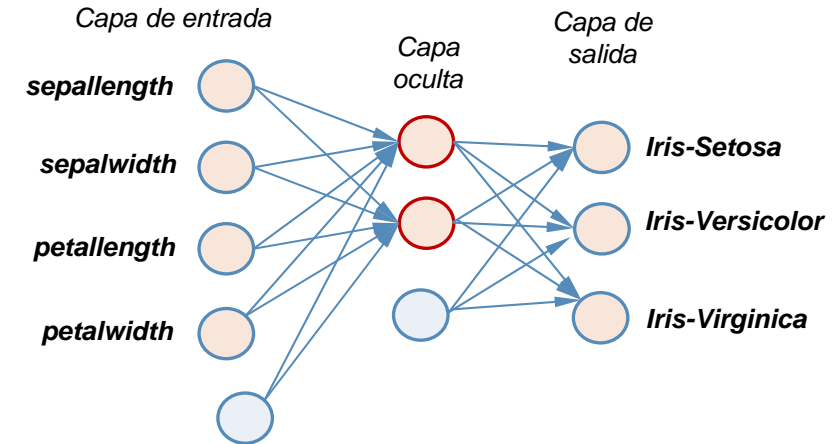
## □ Modificación de W1 y b1

```
W1 = W1 + alfa * deltaH @ x
```

```
W1 = [[ 0.15,-0.13,0.23,-0.45],
       [-0.29,-0.41,-0.19,0.37]] + alfa *  $\begin{bmatrix} [-0.54976963] \\ [-0.06255834] \end{bmatrix} @ [-1.73,-0.05,-1.38,-1.31]$ 
```

```
W1 = [[ 0.25 -0.13  0.31 -0.38]
       [-0.28 -0.41 -0.18  0.38]]
```

```
b1 = b1 + alfa * deltaH =  $\begin{bmatrix} [-0.45] \\ [-0.10] \end{bmatrix} + alfa * \begin{bmatrix} [-0.54976963] \\ [-0.06255834] \end{bmatrix} = \begin{bmatrix} [-0.5] \\ [-0.11] \end{bmatrix}$ 
```



```
FunH='tanh' ; FunO='sigmoid'
```

# Si se ingresa el mismo ejemplo luego de modificar los pesos de la red ...

```
1 netasH = W1 @ xi.T + b1
2 salidasH = 2.0/(1+np.exp(-netasH))-1
3 netasO = W2 @ salidasH + b2
4 salidasO = 1.0/(1+np.exp(-netasO))
5 print("salidaO = \n", salidasO)
6 ErrorSalidaNew = yi.T-salidasO
7 print("ErrorSalida = \n", ErrorSalidaNew)
```

```
salidaO =
[[0.82694546]
 [0.41887838]
 [0.21955387]]
ErrorSalida =
[[ 0.17305454]
 [-0.41887838]
 [-0.21955387]]
```

```
1 print("Error inicial = ", np.sum(ErrorSalida**2))
2 print("Error luego de la correccion = ", np.sum(ErrorSalidaNew**2))
```

```
Error inicial = 0.40847570307169573
Error luego de la correccion = 0.25361086753371476
```

Antes de modificar los pesos de la red

```
salidaO =
[[0.65199139]
 [0.43987471]
 [0.30639182]]
ErrorSalida =
[[ 0.34800861]
 [-0.43987471]
 [-0.30639182]]
```

Ver  
**MLP\_IRIS.ipynb**

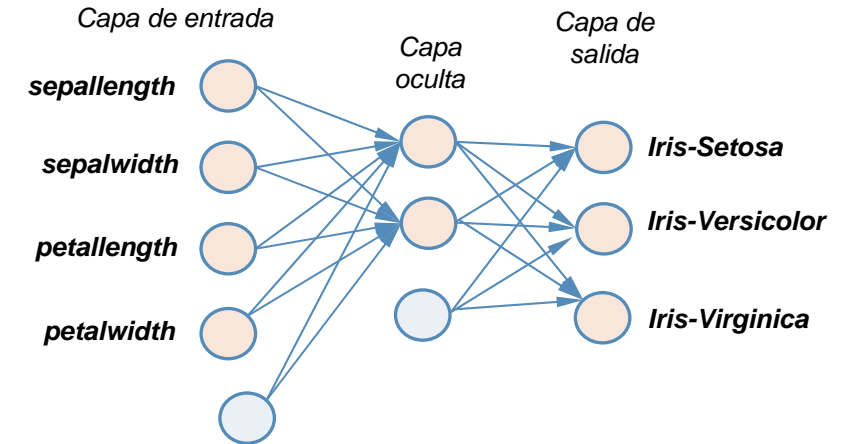
# Clasificación de flores de Iris

**X**

```
[[-1.73,-0.05,-1.38,-1.31],  
 [-0.37,-1.62, 0.22, 0.18],  
 [ 1.11,-0.05, 0.93, 1.54],  
 [-0.99, 0.39,-1.44,-1.31],  
 [ 1.73, 1.29, 1.46, 1.81]]
```

**Y**

```
[[1,0,0],  
 [0,1,0],  
 [0,0,1],  
 [1,0,0],  
 [0,0,1]]
```

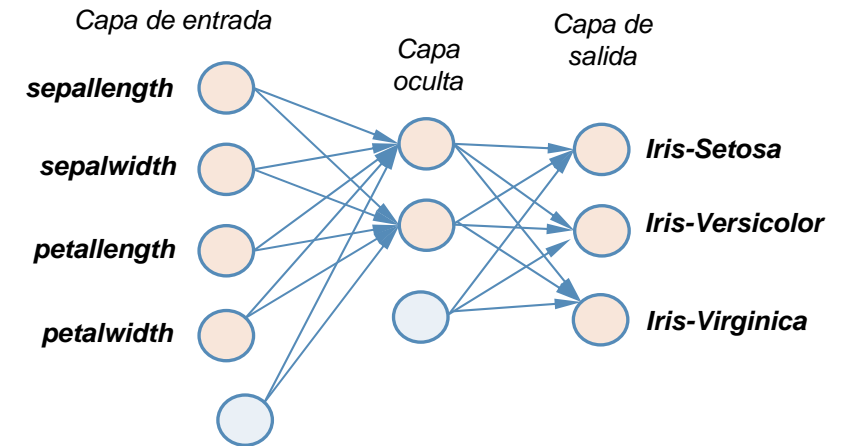


# Clasificación de flores de Iris

**X**

$\left[ \begin{bmatrix} -1.73, -0.05, -1.38, -1.31 \end{bmatrix}, \right. \leftarrow \left. \begin{bmatrix} 1, 0, 0 \end{bmatrix}, \right.$   
 $\begin{bmatrix} -0.37, -1.62, 0.22, 0.18 \end{bmatrix}, \quad \begin{bmatrix} 0, 1, 0 \end{bmatrix},$   
 $\begin{bmatrix} 1.11, -0.05, 0.93, 1.54 \end{bmatrix}, \quad \begin{bmatrix} 0, 0, 1 \end{bmatrix},$   
 $\begin{bmatrix} -0.99, 0.39, -1.44, -1.31 \end{bmatrix}, \quad \begin{bmatrix} 1, 0, 0 \end{bmatrix},$   
 $\begin{bmatrix} 1.73, 1.29, 1.46, 1.81 \end{bmatrix} \quad \begin{bmatrix} 0, 0, 1 \end{bmatrix}]$

FunH= 'tanh' ; FunO= 'softmax'



Ingresar el primer ejemplo a la red y  
calcular su salida

# Calculando la salida de la capa oculta

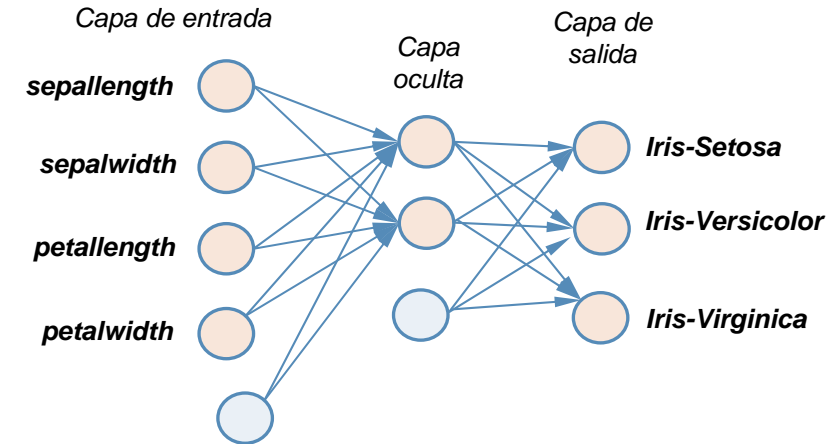
$$\begin{matrix} \mathbf{X} & \mathbf{Y} \\ \left[ \begin{array}{cccc} -1.73, & -0.05, & -1.38, & -1.31 \\ -0.37, & -1.62, & 0.22, & 0.18 \\ 1.11, & -0.05, & 0.93, & 1.54 \\ -0.99, & 0.39, & -1.44, & -1.31 \\ 1.73, & 1.29, & 1.46, & 1.81 \end{array} \right] & \left[ \begin{array}{ccc} 1, & 0, & 0 \\ 0, & 1, & 0 \\ 0, & 0, & 1 \\ 1, & 0, & 0 \\ 0, & 0, & 1 \end{array} \right] \end{matrix}$$

□ Salida de la capa oculta

$$\text{netasH} = \mathbf{W1} * \mathbf{x}^T + \mathbf{b1}$$

$$\begin{bmatrix} 0.15, & -0.13, & 0.23, & -0.45 \\ -0.29, & -0.41, & -0.19, & 0.37 \end{bmatrix} * \begin{bmatrix} -1.73 \\ -0.05 \\ -1.38 \\ -1.31 \end{bmatrix} + \begin{bmatrix} -0.45 \\ -0.10 \end{bmatrix} = \begin{bmatrix} -0.4309 \\ 0.1997 \end{bmatrix}$$

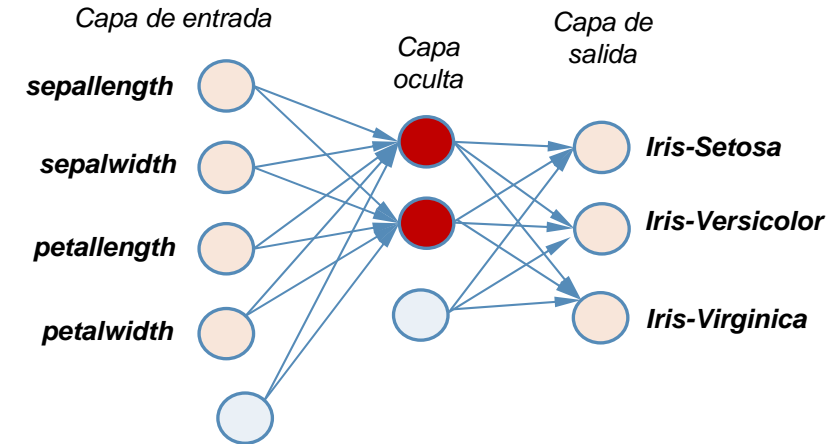
$\mathbf{W1}$   $\mathbf{x}^T$   $\mathbf{b1}$



FunH='tanh' ; FunO='softmax'

# Calculando la salida de la capa oculta

$$\begin{array}{c} \mathbf{X} \\ \left[ \begin{array}{cccc} [-1.73, -0.05, -1.38, -1.31], \\ [-0.37, -1.62, 0.22, 0.18], \\ [1.11, -0.05, 0.93, 1.54], \\ [-0.99, 0.39, -1.44, -1.31], \\ [1.73, 1.29, 1.46, 1.81] \end{array} \right] \end{array} \quad \leftarrow \quad \begin{array}{c} \mathbf{Y} \\ \left[ \begin{array}{ccc} [1, 0, 0], \\ [0, 1, 0], \\ [0, 0, 1], \\ [1, 0, 0], \\ [0, 0, 1] \end{array} \right] \end{array}$$



□ Salida de la capa oculta

$$\text{netasH} = \mathbf{W1} * \mathbf{x.T} + \mathbf{b1}$$

$$\begin{bmatrix} 0.15 & -0.13 & 0.23 & -0.45 \\ -0.29 & -0.41 & -0.19 & 0.37 \end{bmatrix}$$

$\mathbf{W1}$

$\mathbf{x}^T$

$$\begin{bmatrix} -1.73 \\ -0.05 \\ -1.38 \\ -1.31 \end{bmatrix}$$

+

$$\begin{bmatrix} -0.45 \\ -0.10 \end{bmatrix}$$

$\mathbf{b1}$

$$\text{FunH} = \text{'tanh'} ; \text{FunO} = \text{'softmax'}$$

$$= \begin{bmatrix} -0.4309 \\ 0.1997 \end{bmatrix}$$

$$\text{salidasH} = 2 / (1 + \text{np.exp}(-\text{netasH})) - 1 = \begin{bmatrix} -0.21217712 \\ 0.09951948 \end{bmatrix}$$

# Calculando la salida de la red (capa de salida)

**X**

```
[[-1.73, -0.05, -1.38, -1.31],  
 [-0.37, -1.62, 0.22, 0.18],  
 [ 1.11, -0.05, 0.93, 1.54],  
 [-0.99, 0.39, -1.44, -1.31],  
 [ 1.73, 1.29, 1.46, 1.81]]
```

**Y**

```
[[1, 0, 0],  
 [0, 1, 0],  
 [0, 0, 1],  
 [1, 0, 0],  
 [0, 0, 1]]
```

□ Salida de red

```
netasO = W2 * salidasH + b2
```

```
[[-4.79, -0.99],  
 [ 0.38, 0.09],  
 [ 2.40, -0.38]]
```

**W2**

**salidasH**

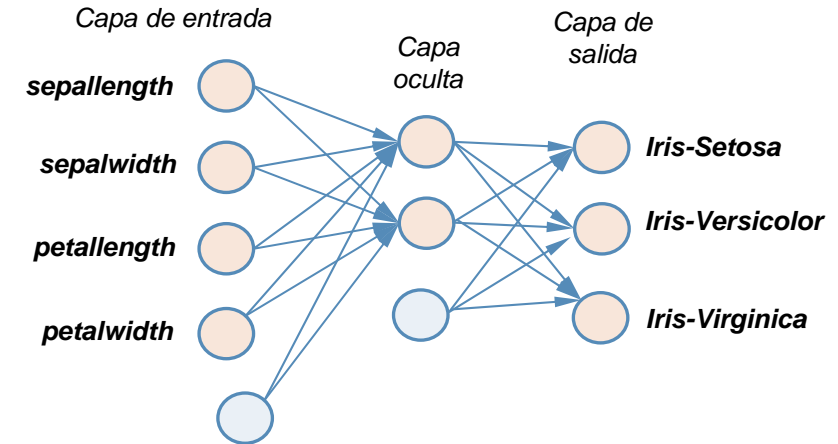
```
* [[-0.21217712]  
   [ 0.09951948]]
```

+

```
[[ -0.29],  
 [-0.17],  
 [-0.27]]
```

**b2**

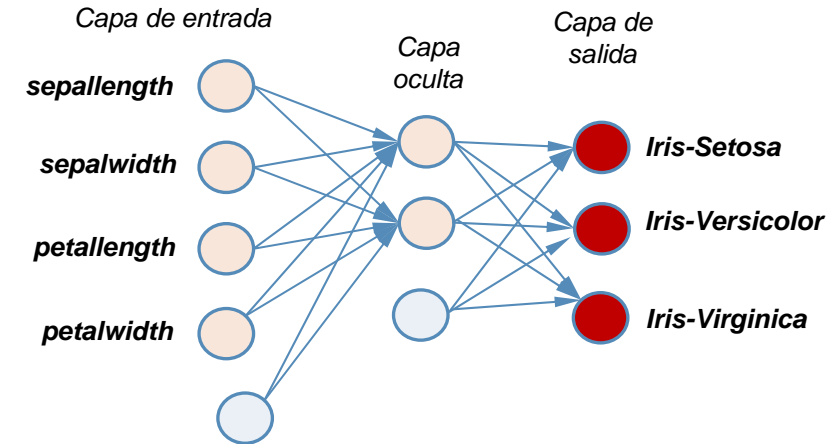
**FunH='tanh' ; FunO='softmax'**



```
= [[ 0.62780411]  
   [-0.24167055]  
   [-0.81704249]]
```

# Calculando la salida de la red (capa de salida)

$$\begin{array}{c} \text{X} \\ \begin{bmatrix} [-1.73, -0.05, -1.38, -1.31], \\ [-0.37, -1.62, 0.22, 0.18], \\ [1.11, -0.05, 0.93, 1.54], \\ [-0.99, 0.39, -1.44, -1.31], \\ [1.73, 1.29, 1.46, 1.81] \end{bmatrix} \end{array} \quad \begin{array}{c} \text{Y} \\ \begin{bmatrix} [1, 0, 0], \\ [0, 1, 0], \\ [0, 0, 1], \\ [1, 0, 0], \\ [0, 0, 1] \end{bmatrix} \end{array} \leftarrow$$



□ Salida de red

$$\text{netasO} = \text{W2} * \text{salidasH} + \text{b2}$$

FunH='tanh' ; FunO='softmax'

$$\begin{array}{c} \text{salidasH} \\ \begin{bmatrix} [-4.79, -0.99], \\ [0.38, 0.09], \\ [2.40, -0.38] \end{bmatrix} \end{array} * \begin{array}{c} \begin{bmatrix} [-0.21217712] \\ [0.09951948] \end{bmatrix} \end{array} + \begin{array}{c} \text{netasO} \\ \begin{bmatrix} [-0.29], \\ [-0.17], \\ [-0.27] \end{bmatrix} \end{array} = \begin{array}{c} \begin{bmatrix} [0.62780411] \\ [-0.24167055] \\ [-0.81704249] \end{bmatrix} \end{array}$$

$$\begin{array}{c} \text{W2} \\ \text{salidasO} = \text{np.exp}(\text{netasO}) / (\text{np.sum}(\text{np.exp}(\text{netasO}))) = \begin{bmatrix} [0.60424642] \\ [0.253283] \\ [0.14247058] \end{bmatrix} \end{array} \leftarrow$$



# Error de la capa de salida

**X**

```
[[ -1.73, -0.05, -1.38, -1.31],
 [-0.37, -1.62,  0.22,  0.18],
 [  1.11, -0.05,  0.93,  1.54],
 [-0.99,  0.39, -1.44, -1.31],
 [  1.73,  1.29,  1.46,  1.81]]
```

**Y**

```
[[1, 0, 0],
 [0, 1, 0],
 [0, 0, 1],
 [1, 0, 0],
 [0, 0, 1]]
```

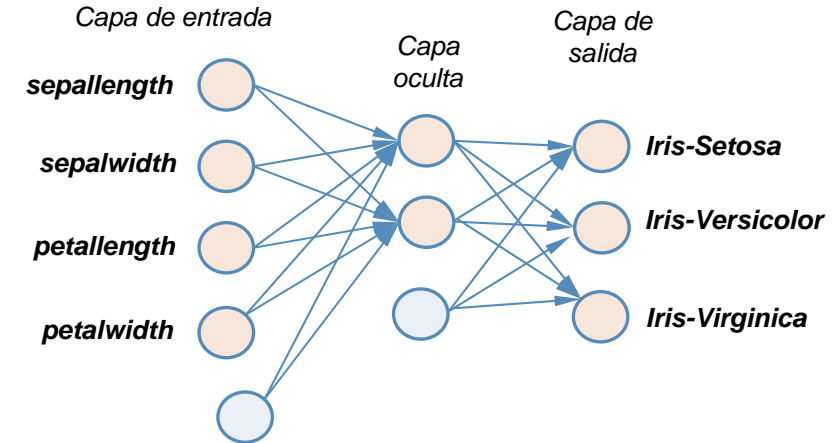
←

- Error en la respuesta de la red para este ejemplo

```
ErrorSalida = - y.T * np.log(salidasO)
```

```
ErrorSalida = [[1.0]  [[-0.50377318]  [[ 0.50377318]
                 [0.0]  [-1.37324784]  [-0.
                 [0.0]  [-1.94861978]]  [-0.
                                     ]
                                     ]]
```

↑ `np.log(salidasO)`



FunH= 'tanh' ; FunO= 'softmax'

$$C = - \sum_{k=1}^3 y_k \ln \hat{y}_k$$

$$C = - \ln \hat{y}_s$$

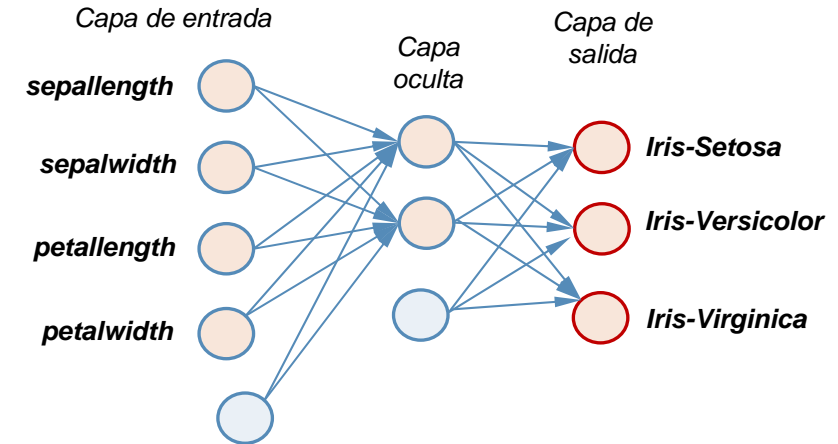
# Factores de corrección de los pesos

**X**

```
[[ -1.73, -0.05, -1.38, -1.31],
 [ -0.37, -1.62,  0.22,  0.18],
 [  1.11, -0.05,  0.93,  1.54],
 [ -0.99,  0.39, -1.44, -1.31],
 [  1.73,  1.29,  1.46,  1.81]]
```

**Y**

```
[[1, 0, 0],
 [0, 1, 0],
 [0, 0, 1],
 [1, 0, 0],
 [0, 0, 1]]
```



FunH= 'tanh' ; FunO= 'softmax'

## Factores para corregir W2 y b2

```
deltaO = (y.T - salidasO)
```

```
deltaO = [[1.0]
           [0.0]
           [0.0]] - [[0.60424642]
                    [0.253283 ]
                    [0.14247058]] = [[ 0.39575358]
                                     [-0.253283 ]
                                     [-0.14247058]]
```

$$\frac{\partial C}{\partial w_{jk}} = -(y_j - \hat{y}_j) x_k$$

$$\frac{\partial C}{\partial b_j} = -(y_j - \hat{y}_j)$$

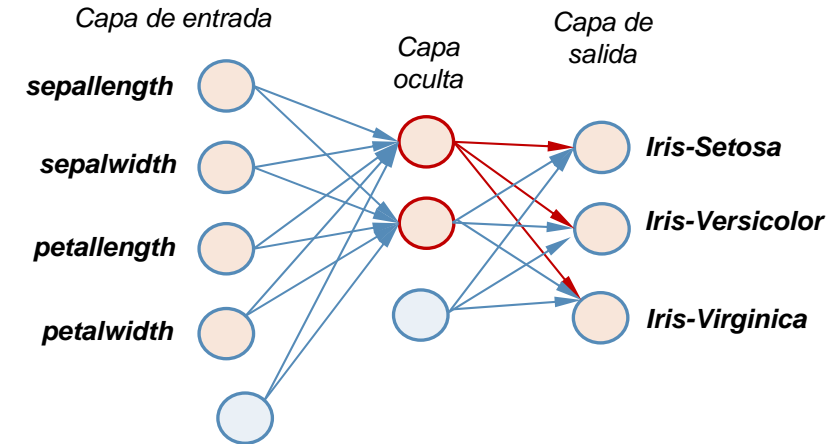
# Factores de corrección de los pesos

**X**

```
[[ -1.73, -0.05, -1.38, -1.31],
 [ -0.37, -1.62,  0.22,  0.18],
 [  1.11, -0.05,  0.93,  1.54],
 [ -0.99,  0.39, -1.44, -1.31],
 [  1.73,  1.29,  1.46,  1.81]]
```

**Y**

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



## Factores para corregir W1 y b1

$(1 - \text{salidasH}^2)$

$\text{deltaH} = \text{deriv\_FunH} .* (\text{W2.T} @ \text{deltaO})$

$\text{deltaH} = \begin{bmatrix} 0.95498087 \\ 0.99009587 \end{bmatrix} .* \begin{bmatrix} -4.79 & 0.38 & 2.4 \\ -0.99 & 0.09 & -0.38 \end{bmatrix} @ \begin{bmatrix} 0.39575358 \\ -0.253283 \\ -0.14247058 \end{bmatrix}$

$\text{deltaH} = \begin{bmatrix} -2.22876926 \\ -0.35688272 \end{bmatrix}$

$\text{FunH} = \text{'tanh'}$  ;  $\text{FunO} = \text{'softmax'}$

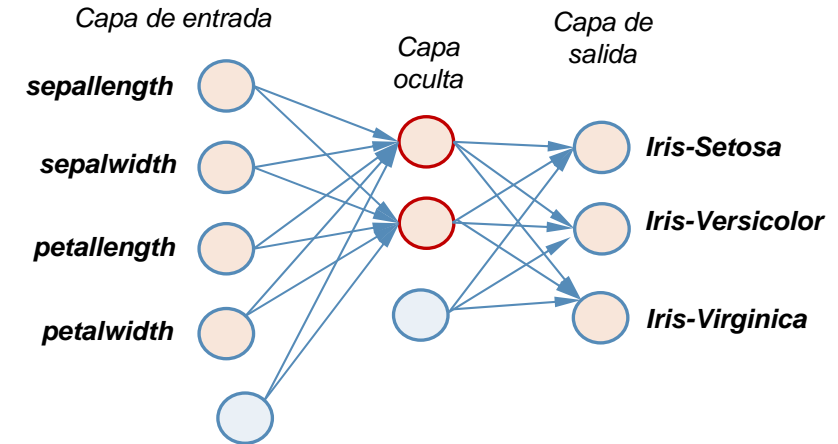
# Corrigiendo de los pesos

**X**

```
[[ -1.73, -0.05, -1.38, -1.31],
 [ -0.37, -1.62,  0.22,  0.18],
 [  1.11, -0.05,  0.93,  1.54],
 [ -0.99,  0.39, -1.44, -1.31],
 [  1.73,  1.29,  1.46,  1.81]]
```

**Y**

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



## □ Modificación de W2 y b2

FunH='tanh' ; FunO='softmax'

$W2 = W2 + \text{alfa} * \text{deltaO} @ \text{salidasH.T}$

$$W2 = \begin{bmatrix} [-4.79, -0.99], \\ [0.38, 0.09], \\ [2.40, -0.38] \end{bmatrix} + \text{alfa} * \begin{bmatrix} [0.39575358] \\ [-0.253283] \\ [-0.14247058] \end{bmatrix} @ \begin{bmatrix} [-0.212177], [0.099519] \end{bmatrix} = \begin{bmatrix} [-4.8 & -0.99] \\ [0.39 & 0.09] \\ [2.4 & -0.38] \end{bmatrix}$$

$$b2 = b2 + \text{alfa} * \text{deltaO} = \begin{bmatrix} [-0.29], \\ [-0.17], \\ [-0.27] \end{bmatrix} + \text{alfa} * \begin{bmatrix} [0.39575358] \\ [-0.253283] \\ [-0.14247058] \end{bmatrix} = \begin{bmatrix} [-0.25] \\ [-0.2] \\ [-0.28] \end{bmatrix}$$

# Corrigiendo de los pesos

**X**

```
[[ -1.73, -0.05, -1.38, -1.31],
 [ -0.37, -1.62,  0.22,  0.18],
 [  1.11, -0.05,  0.93,  1.54],
 [ -0.99,  0.39, -1.44, -1.31],
 [  1.73,  1.29,  1.46,  1.81]]
```

**Y**

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```

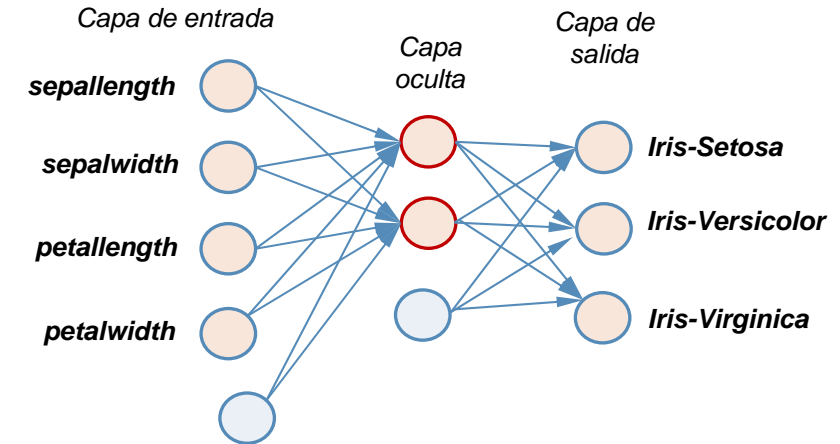
## □ Modificación de W1 y b1

```
W1 = W1 + alfa * deltaH @ x
```

```
W1 = [[ 0.15, -0.13, 0.23, -0.45],
       [-0.29, -0.41, -0.19, 0.37]] + alfa *  $\begin{bmatrix} [-2.22876926] \\ [-0.35688272] \end{bmatrix} @ [-1.73, -0.05, -1.38, -1.31]$ 
```

```
W1 = [[ 0.54 -0.12  0.54 -0.16]
       [-0.23 -0.41 -0.14  0.42]]
```

```
b1 = b1 + alfa * deltaH =  $\begin{bmatrix} [-0.45] \\ [-0.10] \end{bmatrix} + alfa * \begin{bmatrix} [-2.22876926] \\ [-0.35688272] \end{bmatrix} = \begin{bmatrix} [-0.67] \\ [-0.14] \end{bmatrix}$ 
```



```
FunH='tanh' ; FunO='softmax'
```

# Si se ingresa el mismo ejemplo luego de modificar los pesos de la red ...

```
1 netasH = W1 @ xi.T + b1
2 salidasH = 2.0/(1+np.exp(-netasH))-1
3 netasO = W2 @ salidasH + b2
4 salidasO = np.exp(netasO)/(np.sum(np.exp(netasO)))
5 print("salidaO = \n", salidasO)
6 ErrorSalidaNew = -yi.T*np.log(salidasO+EPS)
7 print("ErrorSalida = \n", ErrorSalidaNew)
```

```
salidaO =
[[0.9799741 ]
 [0.01682763]
 [0.00319826]]
ErrorSalida =
[[ 0.02022913]
 [-0.         ]
 [-0.         ]]
```

```
1 print("Error inicial = ", np.max(ErrorSalida))
2 print("Error luego de la correccion = ", np.max(ErrorSalidaNew))
```

```
Error inicial = 0.5037731768932103
Error luego de la correccion = 0.02022913433699978
```

Antes de modificar los pesos de la red

```
salidaO =
[[0.60424642]
 [0.253283   ]
 [0.14247058]]
ErrorSalida =
[[ 0.50377318]
 [-0.         ]
 [-0.         ]]
```

Ver  
**MLP\_IRIS\_SoftMax.ipynb**

# Multiperceptrón en Python

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier( solver='sgd',
                    learning_rate_init=0.15,
                    hidden_layer_sizes=(5),
                    max_iter=700, verbose=False,
                    tol=1.0e-09, activation = 'tanh')

clf.fit(X_train,T_train)
y_pred= clf.predict(X_train)
```

# Matriz de Confusión

	Predice Clase 1	Predice Clase 2	Recall
True Clase 1	<b>A</b>	<b>B</b>	$A/(A+B)$
True Clase 2	<b>C</b>	<b>D</b>	$D/(C+D)$
Precision	$A/(A+C)$	$D/(B+D)$	$(A+D)/(A+B+C+D)$ accuracy

- Los **aciertos** del modelo están sobre la **diagonal** de la matriz.
- **Precision**: la proporción de **predicciones correctas** sobre **una clase**.
- **Recall**: la proporción de **ejemplos** de **una clase** que son **correctamente clasificados**.
- **Accuracy**: la performance general del modelo, sobre **todas las clases**. Es la cantidad de **aciertos** sobre el **total** de ejemplos.



# Sklearn.metrics.accuracy\_score

```
from sklearn import metrics

Y_train = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
Y_pred  = [0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 2, 3]

aciertos = metrics.accuracy_score(Y_train, Y_pred)
print("%% accuracy = %.3f" % aciertos)
```

# Sklearn.metrics.accuracy\_score

```
from sklearn import metrics

Y_train = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
Y_pred  = [0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 2, 3]

aciertos = metrics.accuracy_score(Y_train, Y_pred)
print("%% accuracy = %.3f" % aciertos)
```

Rtas esperadas

Rtas obtenidas

- De los 12 valores sólo 9 fueron identificados correctamente.
- La tasa de aciertos es  $9/12 = 0.75$

# Sklearn.metrics.confusion\_matrix

```
Y_train = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
Y_pred  = [0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 2, 3]
```

```
MM = metrics.confusion_matrix(Y_train, Y_pred)
print("Matriz de confusión:\n%s" % MM)
```

```
Matriz de confusión:
 0 [[3 0 0 0]
 1 [0 2 1 0]
 2 [0 1 2 0]
 3 [1 0 0 2]]
    0  1  2  3
  PREDICE
```

Esperaba obtener 1  
como respuesta pero  
la red respondió 2

# Sklearn.metrics.confusion\_matrix

```
Y_train = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
Y_pred  = [0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 2, 3]
```

```
MM = metrics.confusion_matrix(Y_train, Y_pred)
print("Matriz de confusión:\n%s" % MM)
```

```
Matriz de confusión:
 0 [[3 0 0 0]
 1 [0 2 1 0]
 2 [0 1 2 0]
 3 [1 0 0 2]]
    0  1  2  3
  PREDICE
```

La respuesta correcta  
es 2 pero la red  
respondió 1

# Sklearn.metrics.confusion\_matrix

```
Y_train = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
Y_pred  = [0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 2, 3]
```

```
MM = metrics.confusion_matrix(Y_train, Y_pred)
print("Matriz de confusión:\n%s" % MM)
```

```
Matriz de confusión:
 0 [[3 0 0 0]
 1 [0 2 1 0]
 2 [0 1 2 0]
 3 [1 0 0 2]]
   0  1  2  3
  PREDICE
```

Esperaba un 3 pero la red respondió 0

# Sklearn.metrics.confusion\_matrix

```
Y_train = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
Y_pred  = [0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 2, 3]
```

```
MM = metrics.confusion_matrix(Y_train, Y_pred)
print("Matriz de confusión:\n%s" % MM)
```

```
Matriz de confusión:
 0 [[3 0 0 0]
 1 [0 2 1 0]
 2 [0 1 2 0]
 3 [1 0 0 2]]
 0 1 2 3
PREDICE
```

Los valores fuera de la diagonal principal son errores

# Sklearn.metrics.classification\_report

```
Y_train = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
Y_pred  = [0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 2, 3]
report = metrics.classification_report(Y_train, Y_pred)
print("Resultado de la clasificación:\n%s" % report)
```

Resultado de la clasificación:

	precision	recall	f1-score	support
0	0.75	1.00	0.86	3
1	0.67	0.67	0.67	3
2	0.67	0.67	0.67	3
3	1.00	0.67	0.80	3
accuracy			0.75	12
macro avg	0.77	0.75	0.75	12
weighted avg	0.77	0.75	0.75	12

Matriz de confusión:

```
[[3 0 0 0]
 [0 2 1 0]
 [0 1 2 0]
 [1 0 0 2]]
```

# Sklearn.metrics.classification\_report

```
Y_train = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
Y_pred  = [0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 2, 3]
report = metrics.classification_report(Y_train, Y_pred)
print("Resultado de la clasificación:\n%s" % report)
```

Resultado de la clasificación:



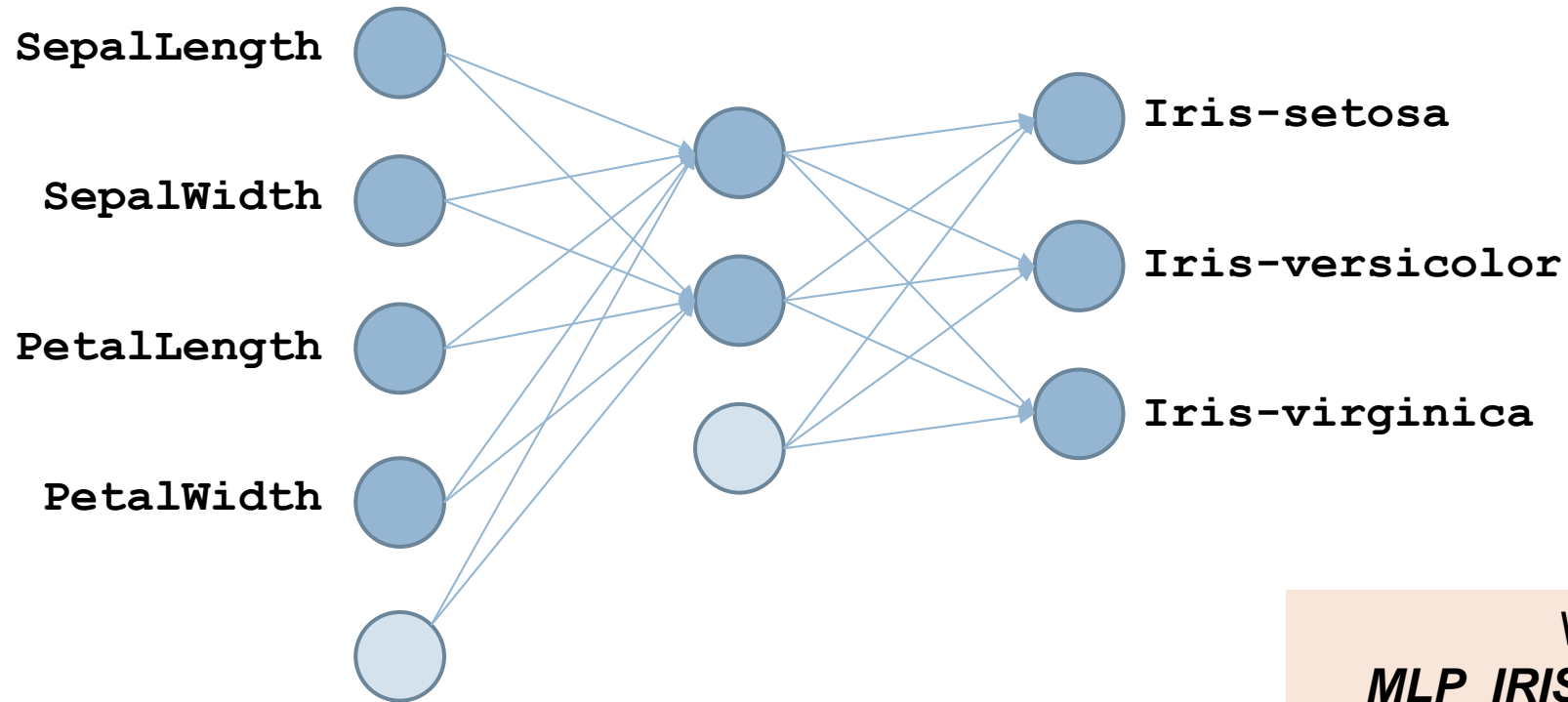
	precision	recall	f1-score	support
0	0.75	1.00	0.86	3
1	0.67	0.67	0.67	3
2	0.67	0.67	0.67	3
3	1.00	0.67	0.80	3
accuracy			0.75	12
macro avg	0.77	0.75	0.75	12
weighted avg	0.77	0.75	0.75	12

## F1-score

$$F1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precisión} + \text{recall}}$$



# Ejemplo: Clasificación de flores de Iris



Ver  
*MLP\_IRIS\_RN.ipynb*

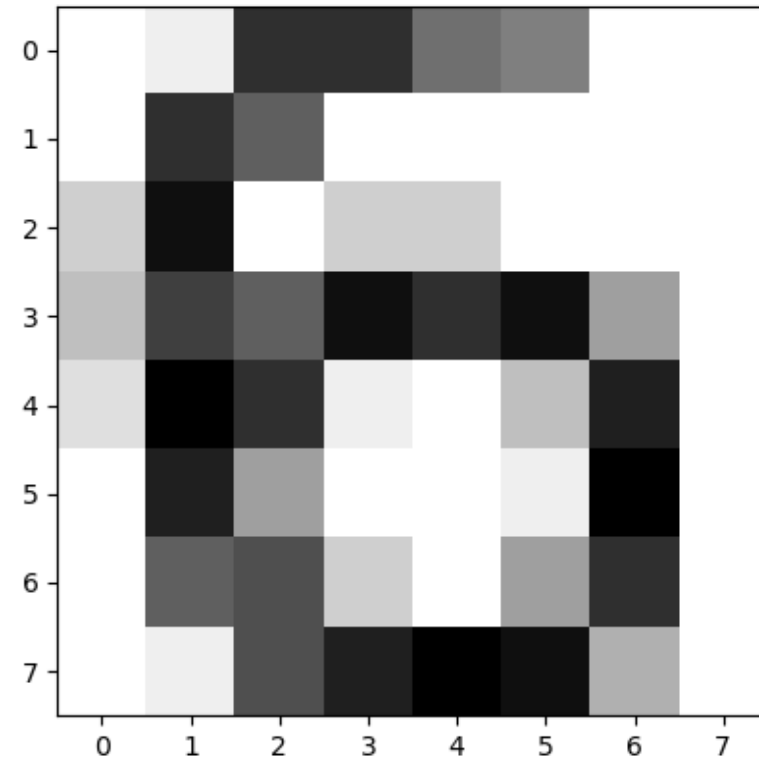
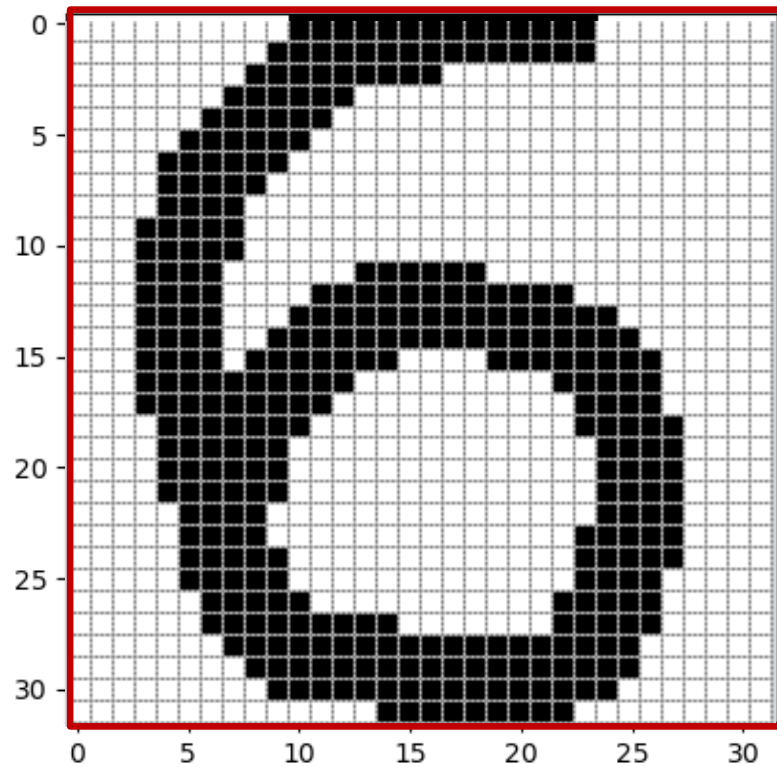
# Reconocedor de dígitos escritos a mano

- Se desea entrenar un multiperceptrón para reconocer dígitos escritos a mano. Para ello se dispone de los mapas de bits correspondientes a 3823 dígitos escritos a mano por 30 personas diferentes en el archivo “optdigits\_train.csv”.
- El desempeño de la red será probado con los dígitos del archivo “optdigits\_test.csv” escritos por otras 13 personas.



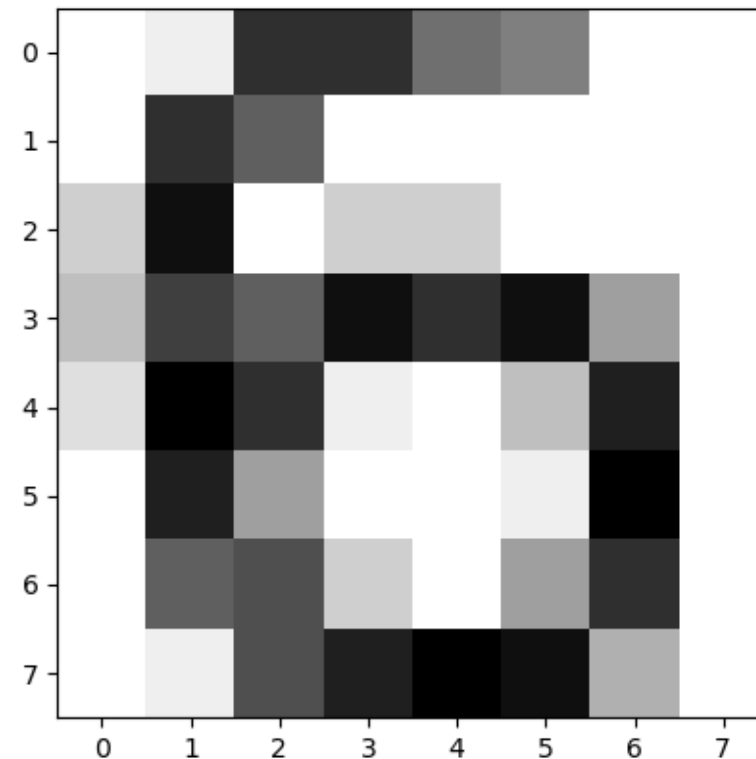
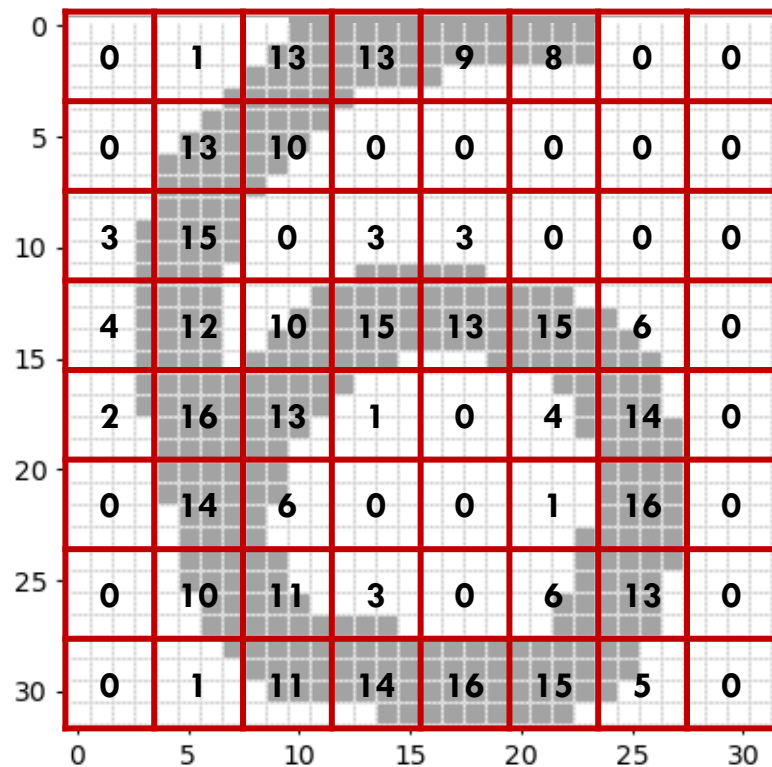
# “optdigits\_train.csv” y “optdigits\_test.csv”

- Cada dígito está representado por una matriz numérica de 8x8



# “optdigits\_train.csv” y “optdigits\_test.csv”

- Cada dígito está representado por una matriz numérica de 8x8



# RN para reconocer dígitos manuscritos

```
In [90]: print("ite = %d %% aciertos X_train : %.3f" % (ite,
metrics.accuracy_score(Y_train,Y_pred)))
ite = 200  % aciertos X_train : 0.982
```

```
In [91]: MM = metrics.confusion_matrix(Y_train,Y_pred)
....: print("Matriz de confusión TRAIN:\n%s" % MM)
```

Matriz de confusión TRAIN: im\_32x32\_a\_8x8.ipynb

```
[[375  0  0  0  0  0  0  0  1  0]
 [ 7 382  0  0  0  0  0  0  0  0]
 [ 2  0 378  0  0  0  0  0  0  0]
 [ 7  0  0 380  0  2  0  0  0  0]
 [ 3  0  0  0 383  0  1  0  0  0]
 [ 6  0  0  1  0 369  0  0  0  0]
 [ 2  2  0  0  0  0 373  0  0  0]
 [ 0  0  0  1  0  0  0 386  0  0]
 [17  2  0  0  0  0  0  0 361  0]
 [13  1  0  1  0  1  0  0  0 366]]
```

Entrenamiento  
muy lento  
¿Se puede  
acelerar?

MLP\_MNIST\_8x8.ipynb