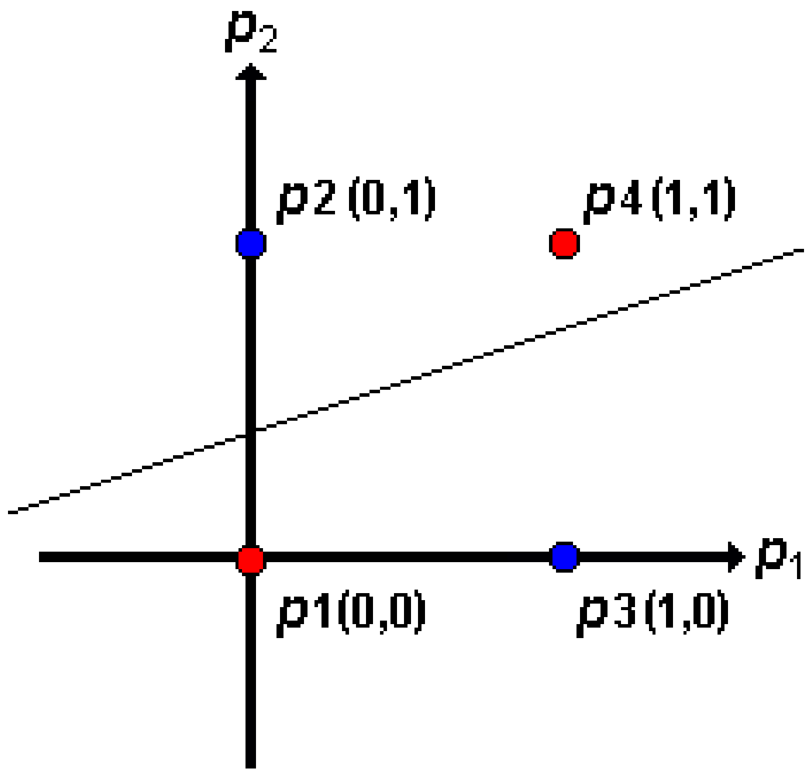
# MULTIPERCEPTRÓN

## ALGORITMO BACKPROPAGATION

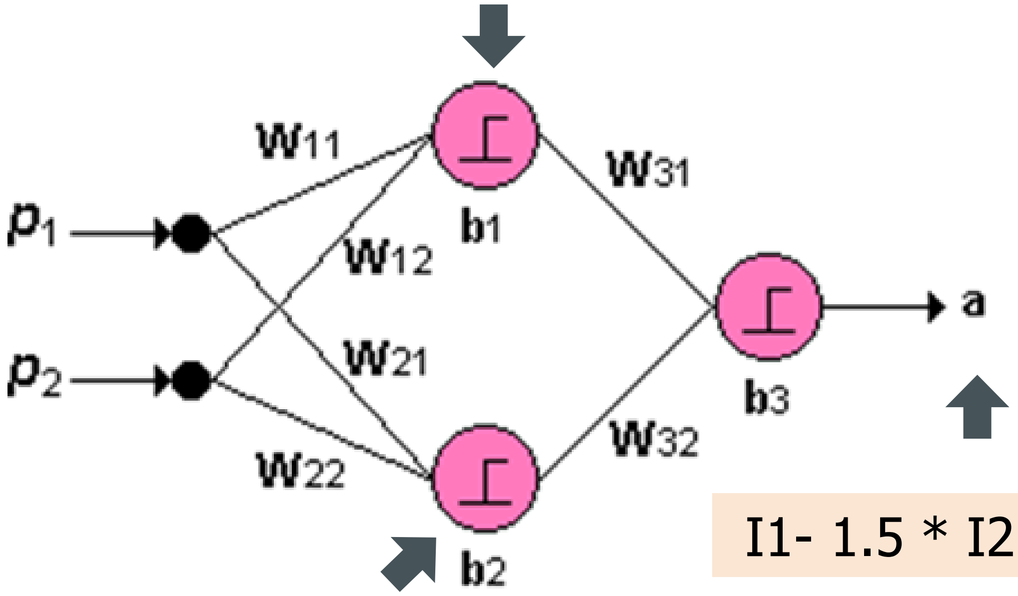
# XOR

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



# XOR

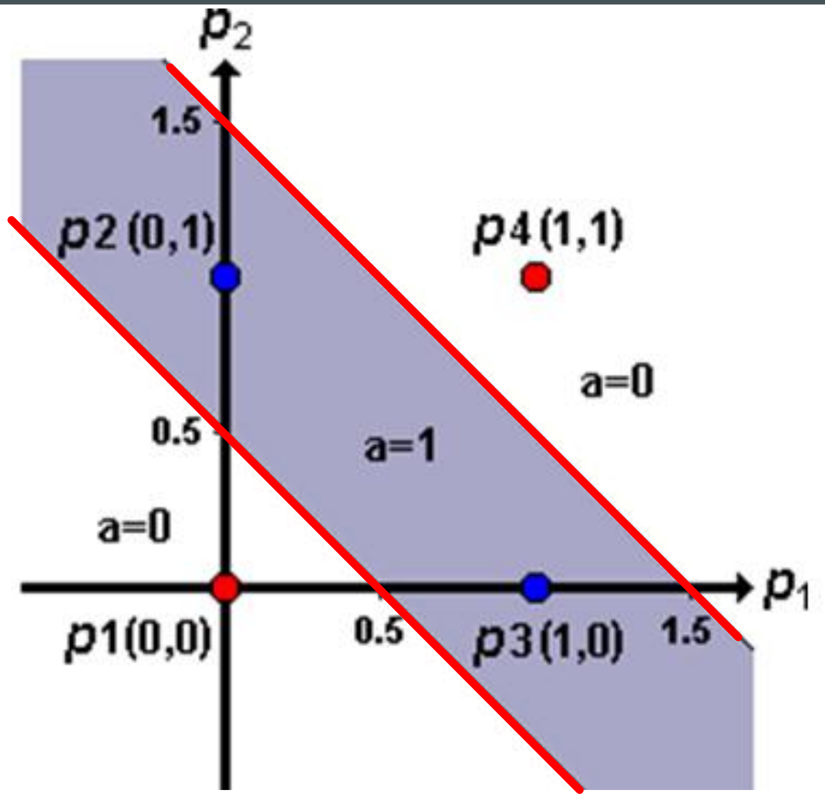
OR → p1+p2-0.5=0




I1- 1.5 * I2 - 0.5=0

AND

p1+p2-1.5=0

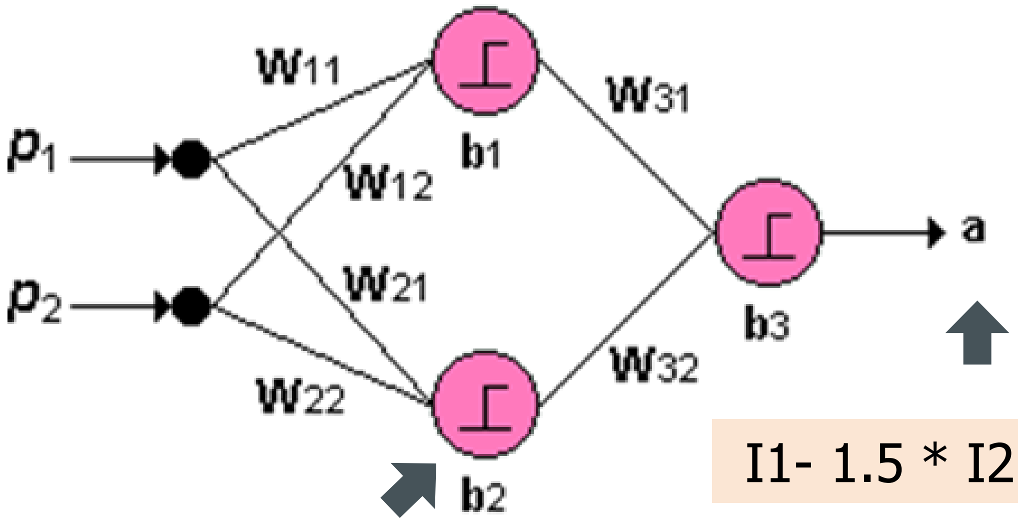



$w_{11}$=1 $w_{12}$=1 $b_1$=-0.5 ; $w_{21}$=1 $w_{22}$=1 $b_2$=-1.5 ; $w_{31}$=1 $w_{32}$=-1.5 $b_3$=-0.5

# XOR

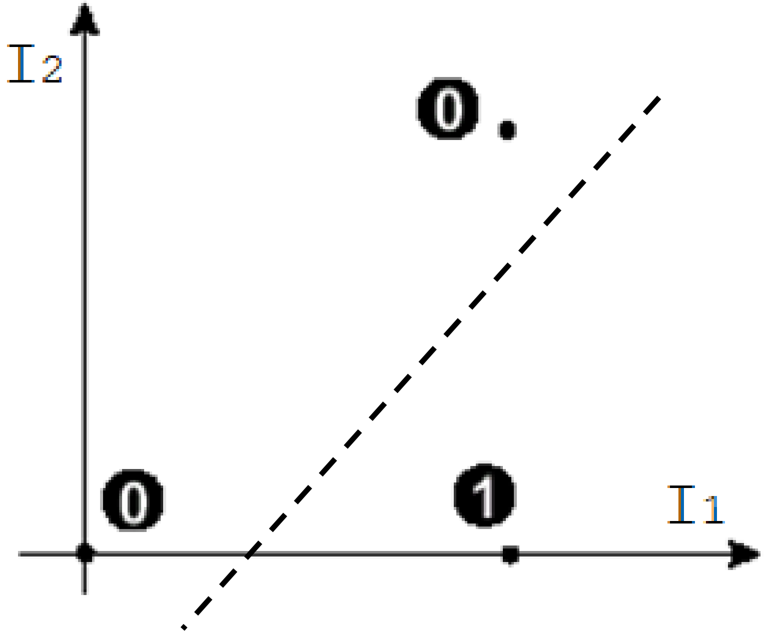
| p1 | p2 | I1 (or) | I2 (AND) | a |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 |

OR → p1+p2-0.5=0



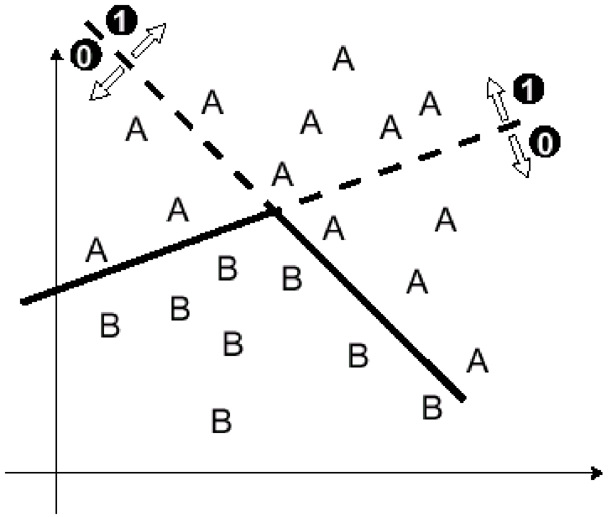

AND
p1+p2-1.5=0

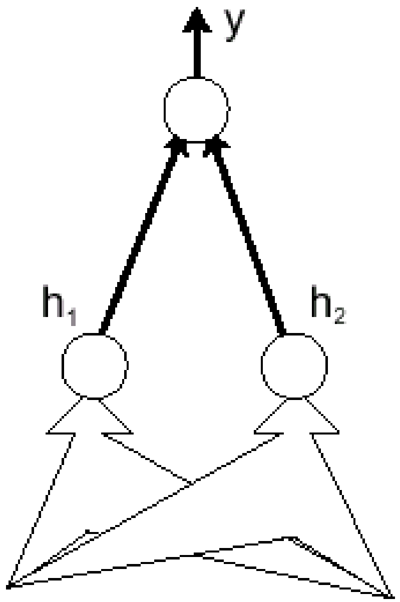
I1- 1.5 * I2 - 0.5=0




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

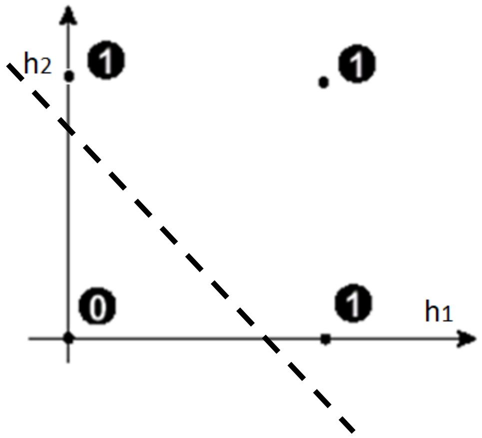
# PROBLEMA NO SEPARABLE LINEALMENTE






| $h_1$ | $h_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

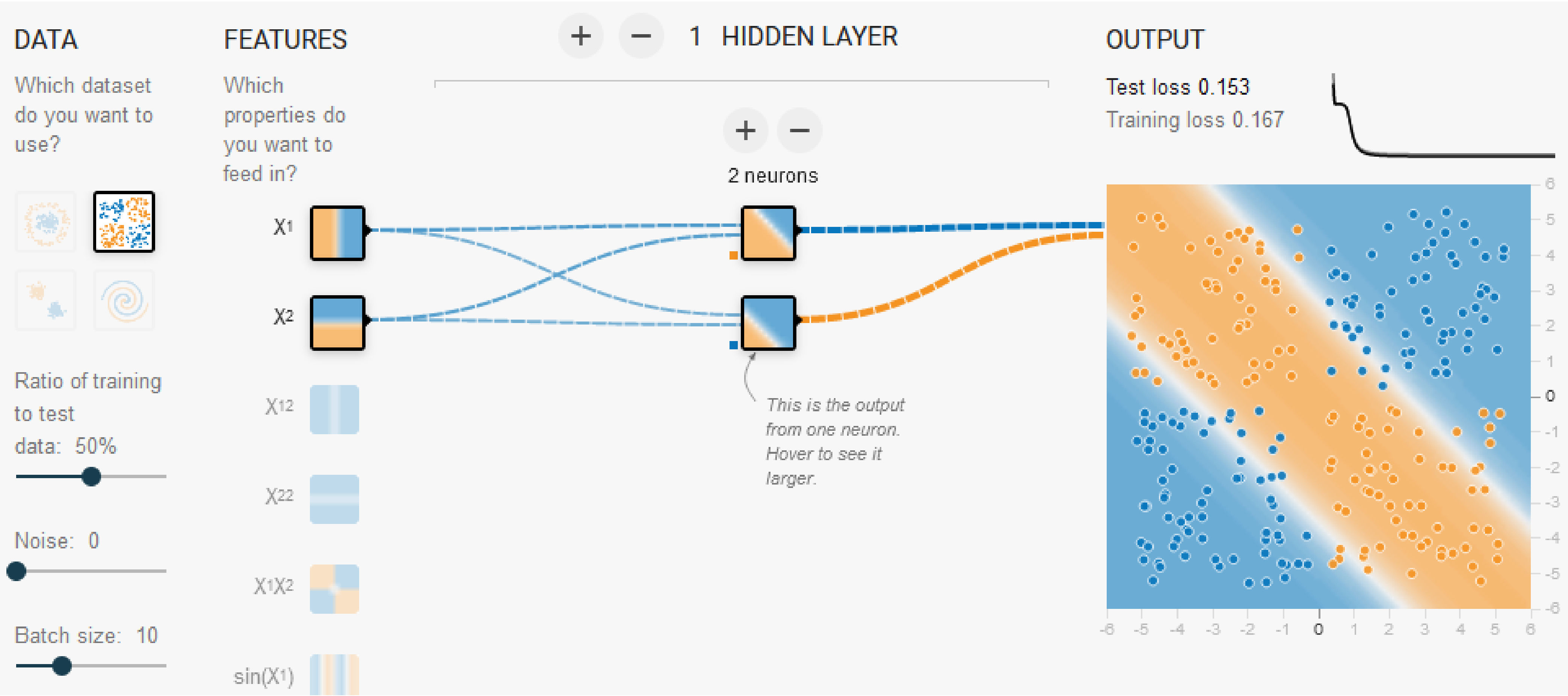
$A \leftrightarrow 1$



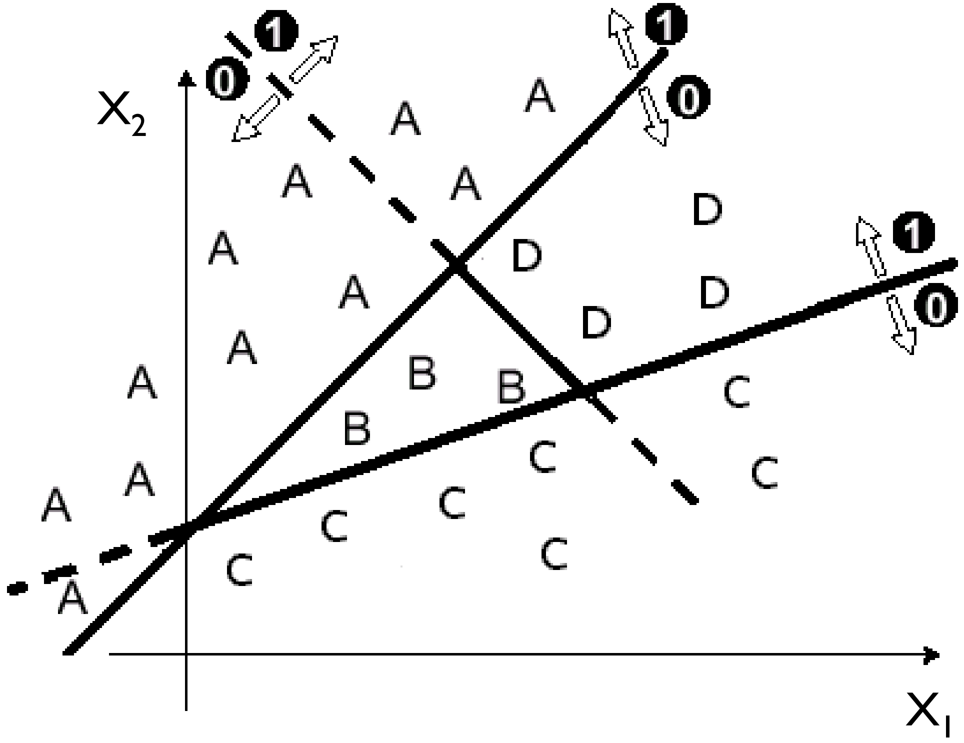

- Se busca obtener un algoritmo más general que permita integrar el aprendizaje entre las dos capas.

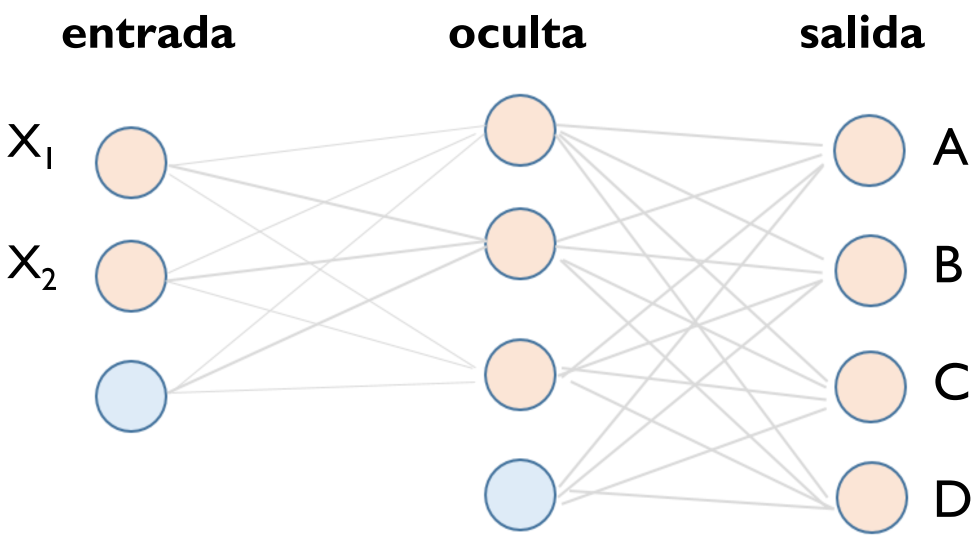
# ANIMACIÓN DE UNA RN

Tinker With a Neural Network Right Here in Your Browser



# PROBLEMA NO SEPARABLE LINEALMENTE



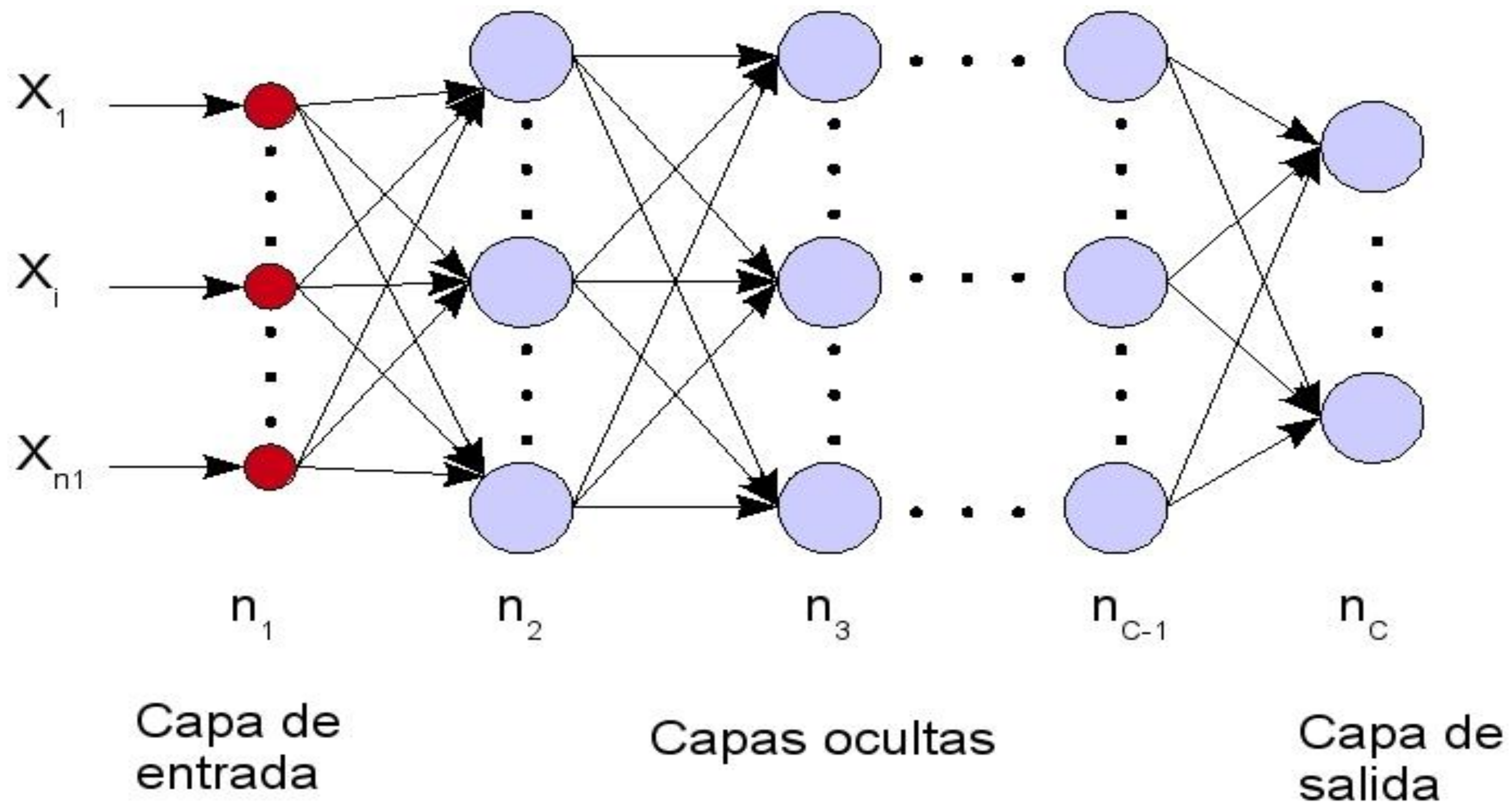




- ¿Cuál es el tamaño de cada capa?

# PROBLEMA NO SEPARABLE LINEALMENTE

- 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 que unen AMBAS capas.
- Dado que el aprendizaje es supervisado, para los nodos de salida se conoce la respuesta esperada a cada entrada (**regla delta**).
- Es preciso definir una manera de medir la participación de las neuronas ocultas en el valor de salida para determinar cómo corregir los pesos que llegan a ellas.

# MULTIPERCEPTRÓN - ARQUITECTURA



# ALGORITMO BACKPROPAGATION

Dado el siguiente conjunto de vectores

$$\{(x_1, t_1),\ \ldots,\ (x_p, t_p)\}$$

que son ejemplos de correspondencia funcional

$$t = \varphi(x) \qquad x \in R^N, t \in R^M$$

se busca entrenar la red para que aprenda una aproximación

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

# BACKPROPAGATION. CAPA OCULTA

- 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

- 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

$$y_{pk} = f_{k}^{o}(neta_{pk}^{o})$$

# ACTUALIZACIÓN DE PESOS

- Error en una sola unidad de la capa de salida

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

**salida esperada** para el vector de entrada $p$ en la neurona de salida $k$

**salida obtenida** para el vector de entrada $p$ en la neurona de salida $k$

# ACTUALIZACIÓN DE PESOS

- Se busca minimizar

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

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

se tomará el valor negativo del gradiente

# ACTUALIZACIÓN DE PESOS

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

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

$$f_k^{o\prime}(neta_{pk}^o)$$

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

Salida de la neurona oculta j

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

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

$$\delta_{pk}^{o} = (t_{pk} - y_{pk}) f_k^{o\prime}(neta_{pk}^{o})$$

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

# ACTUALIZACIÓN DE PESOS

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

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

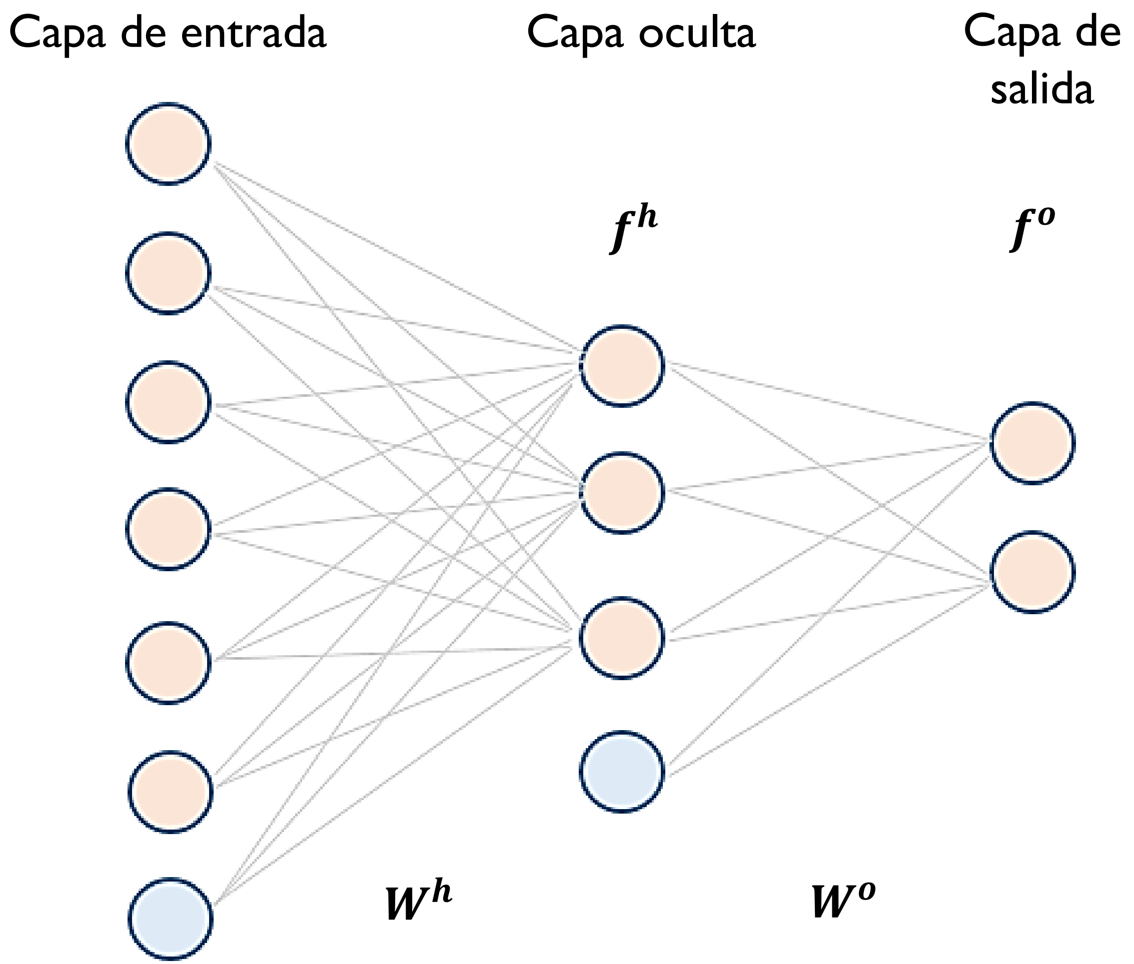
serán de la forma:

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

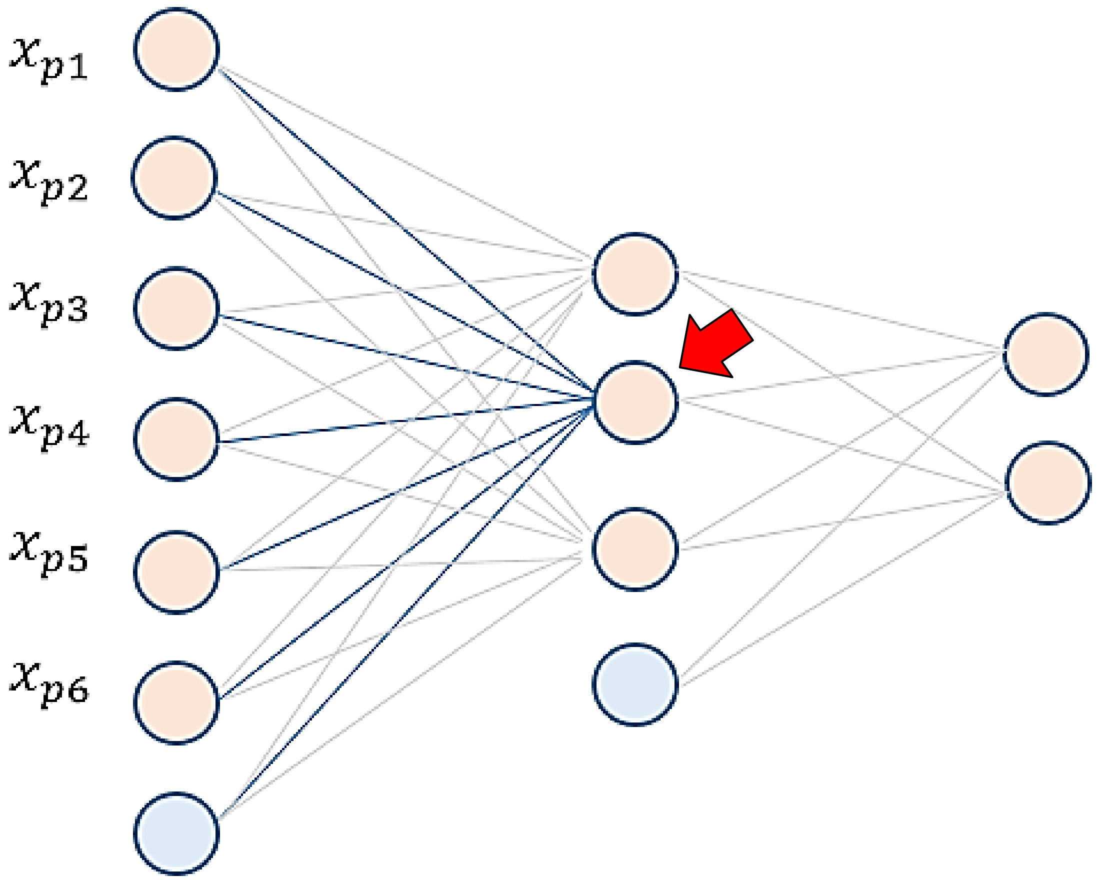
# ALGORITMO BACKPROPAGATION

```
Definir alfa, MAX_ITE y COTA
Inicializar los pesos de forma aleatoria
Mientras ( la variación de la función de costo supere cierta COTA ):
    Para cada ejemplo
        - propagar el ejemplo hacia adelante
        - calcular los gradientes para cada capa (parte común)
        - corregir todos los pesos
    Recalcular la función de costo
```

# Problema de clasificación en 2 clases



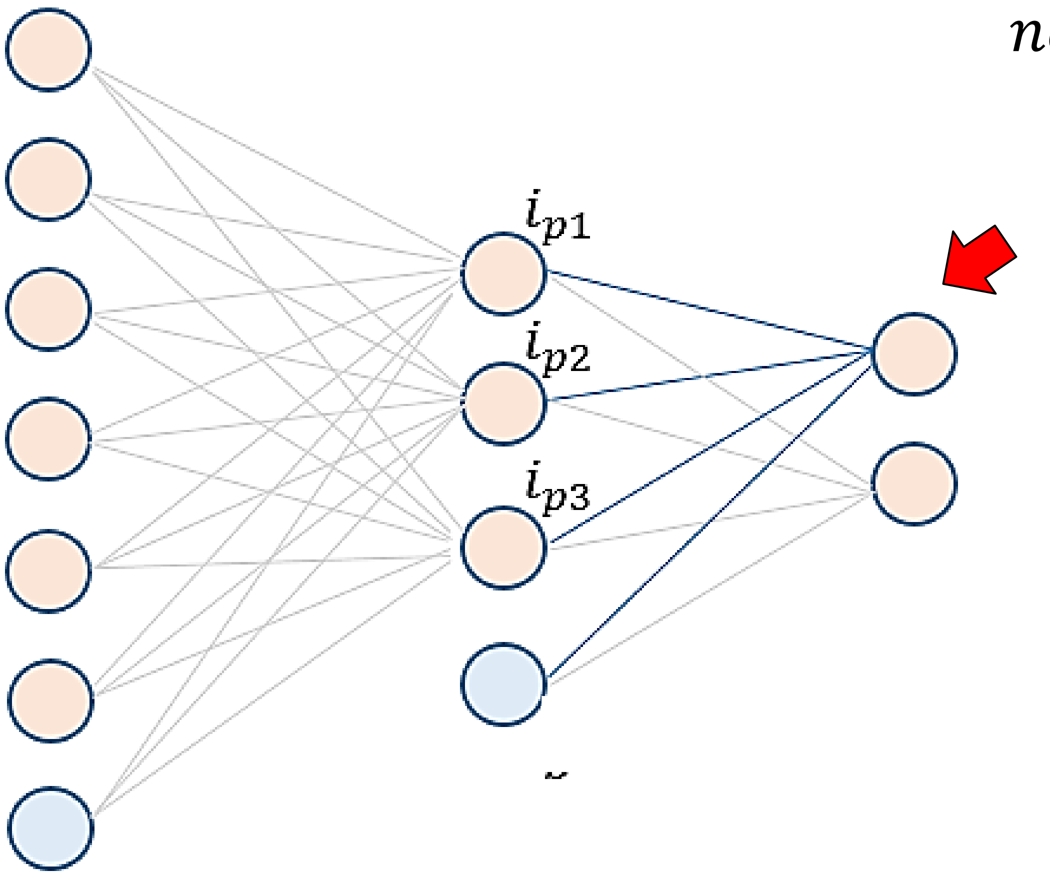
# Propagar el ejemplo de entrada a través de la capa oculta




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

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

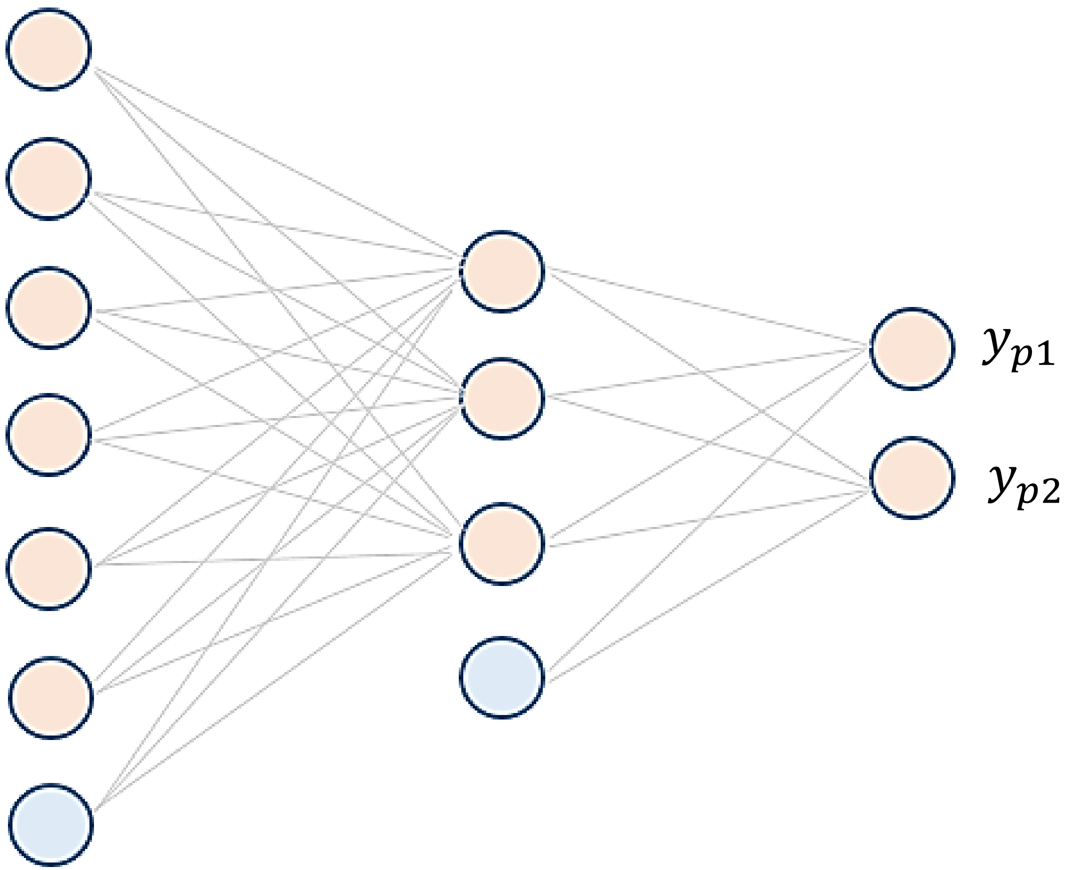
# Propagar las salidas de la capa oculta hacia la capa de salida




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

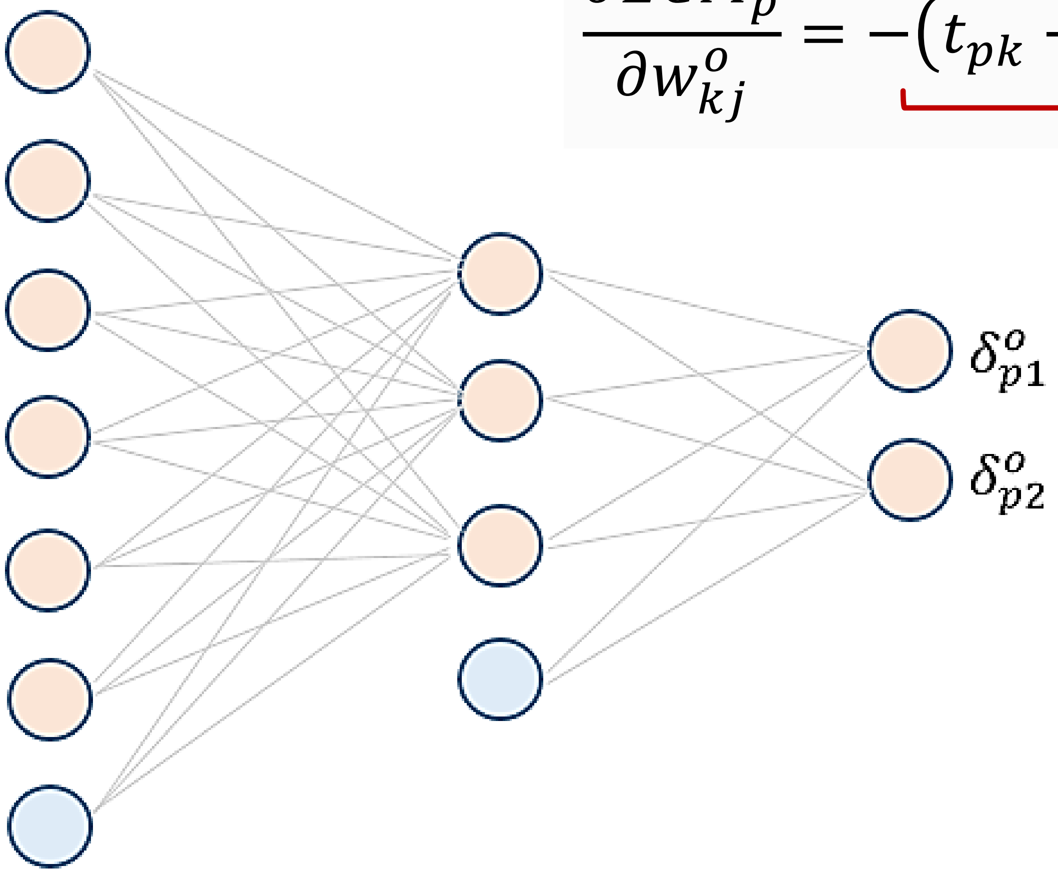
$$y_{pk} = f_{k}^{o}(neta_{pk}^{o})$$

# Se obtienen los valores de salida



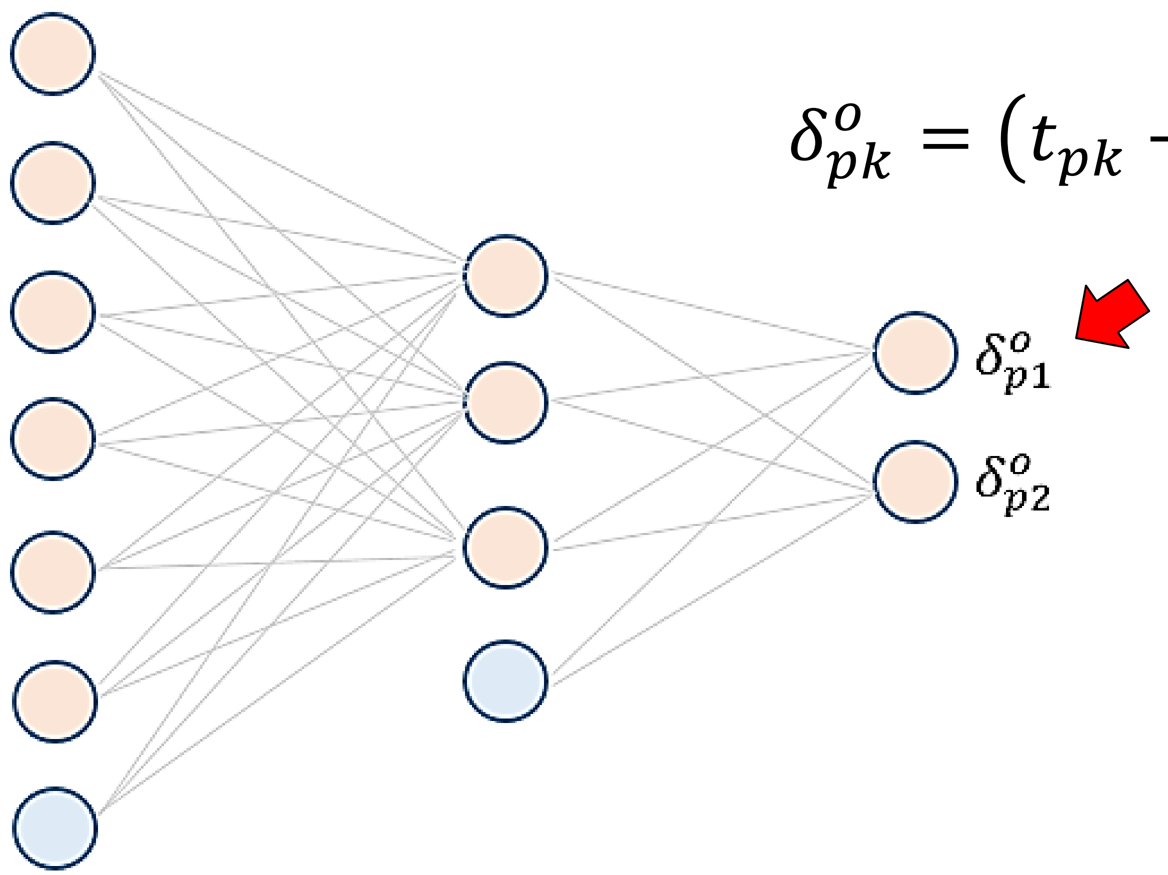
# Corrección que se realizará a los pesos de cada neurona de salida

$$\frac{\partial ECM_p}{\partial w^o_{kj}} = \underbrace{-\left(t_{pk} - y_{pk}\right) f^{o\prime}_k\left(neta^o_{pk}\right)}_{\delta^o_{pk}}\ i_{pj}$$



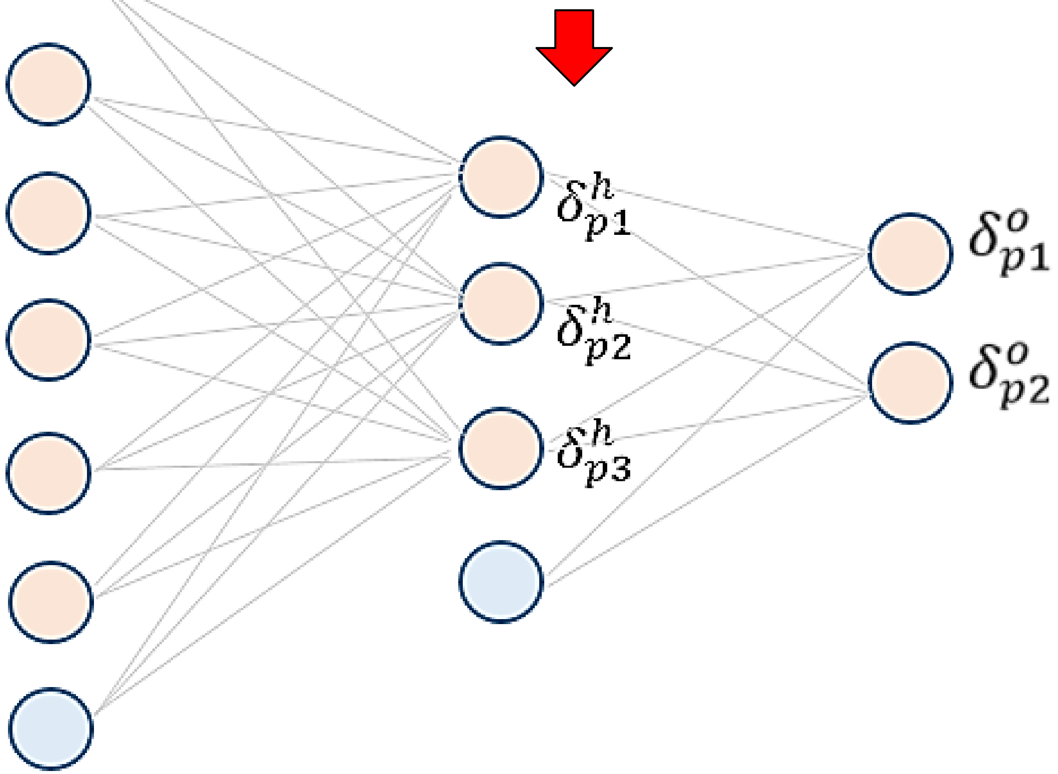
# Corrección que se realizará a los pesos de cada neurona de salida

$$\delta_{pk}^{o} = \left(t_{pk} - y_{pk}\right) f_k^{o\prime}(neta_{pk}^{o})$$



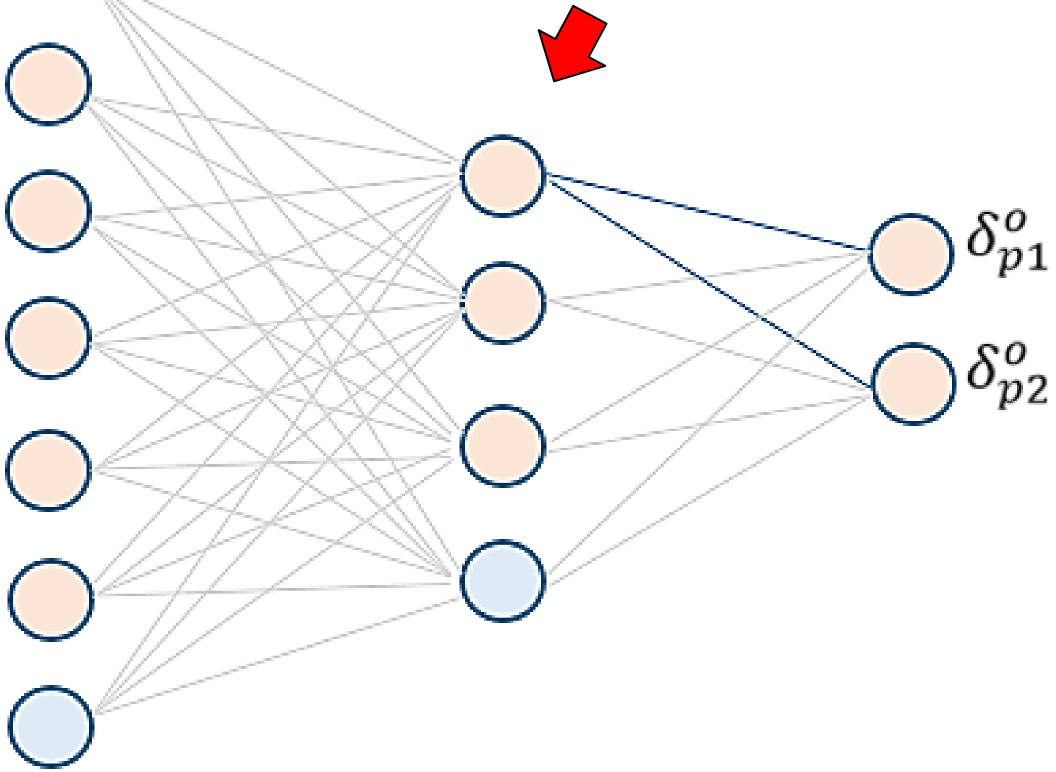
# Corrección que se realizará a los pesos que llega a cada neurona oculta

$$\delta_{pj}^{h} = f_j^{h\prime}(neta_{pj}^{h}) \sum_{k} \delta_{pk}^{o} w_{kj}^{o}$$



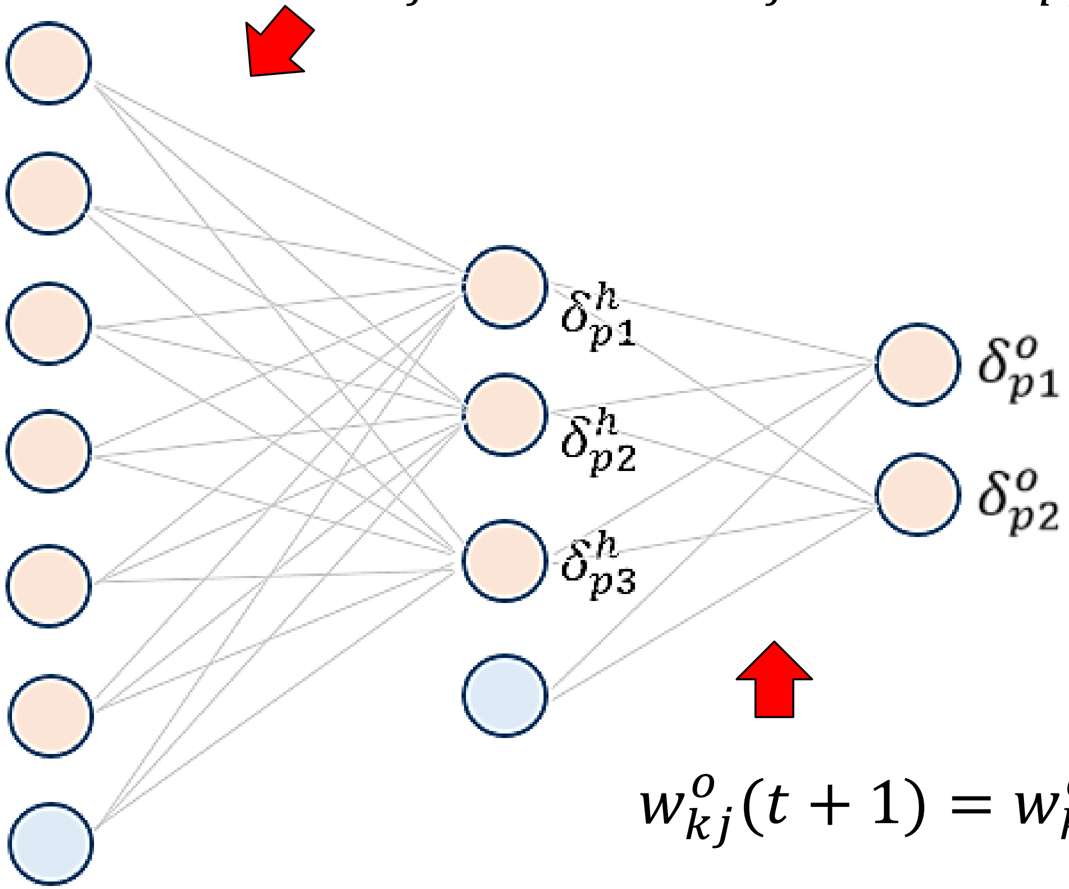
# Corrección que se realizará a los pesos que llega a cada neurona oculta

$$\delta^h_{p1} = f^{h\prime}_1(neta^h_{p1}) \sum_k \delta^o_{pk} w^o_{k1}$$



# Se actualizan ambas matrices de pesos

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




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

# BACKPROPAGATION. RESUMEN

- Aplicar el vector de entrada

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

- 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

- 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

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

# BACKPROPAGATION. RESUMEN

- Calcular la parte del gradiente común a cada neurona de la capa de salida

$$\delta_{pk}^{o} = (t_{pk} - y_{pk}) f_k^{o\prime}(neta_{pk}^{o})$$

- Calcular la parte del gradiente común a cada neurona de la capa oculta

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

# BACKPROPAGATION. RESUMEN

- Se actualizan los pesos de la capa de salida

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

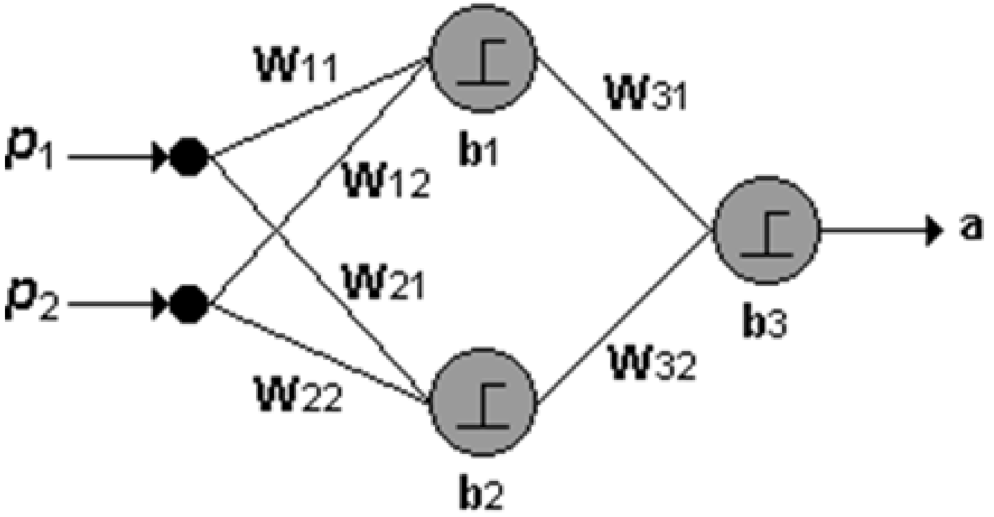
- Se actualizan los pesos de la capa oculta

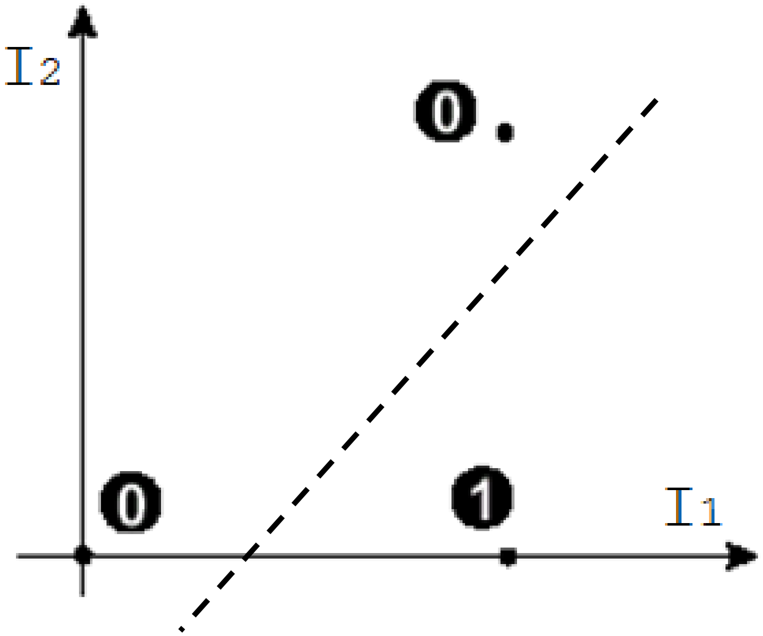
$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \alpha\, \delta_{pj}^{h}\, x_{i}$$

- Repetir mientras la reducción del ECM supere cierta COTA

# XOR

| 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 |



**MLP_XOR.ipynb**

# PROBLEMA DEL XOR

```
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]])
T = np.array([-1, 1, 1, -1]).reshape(-1,1)

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

**X**

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

**T**

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

# PESOS INICIALES

**MLP_XOR.ipynb**

```
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])
```

```
W1
array([[-0.09705477, -0.48156505],
       [ 0.14081924,  0.17185576]])
```

```
b1
array([[ 0.28208473],
       [ 0.07973888]])
```

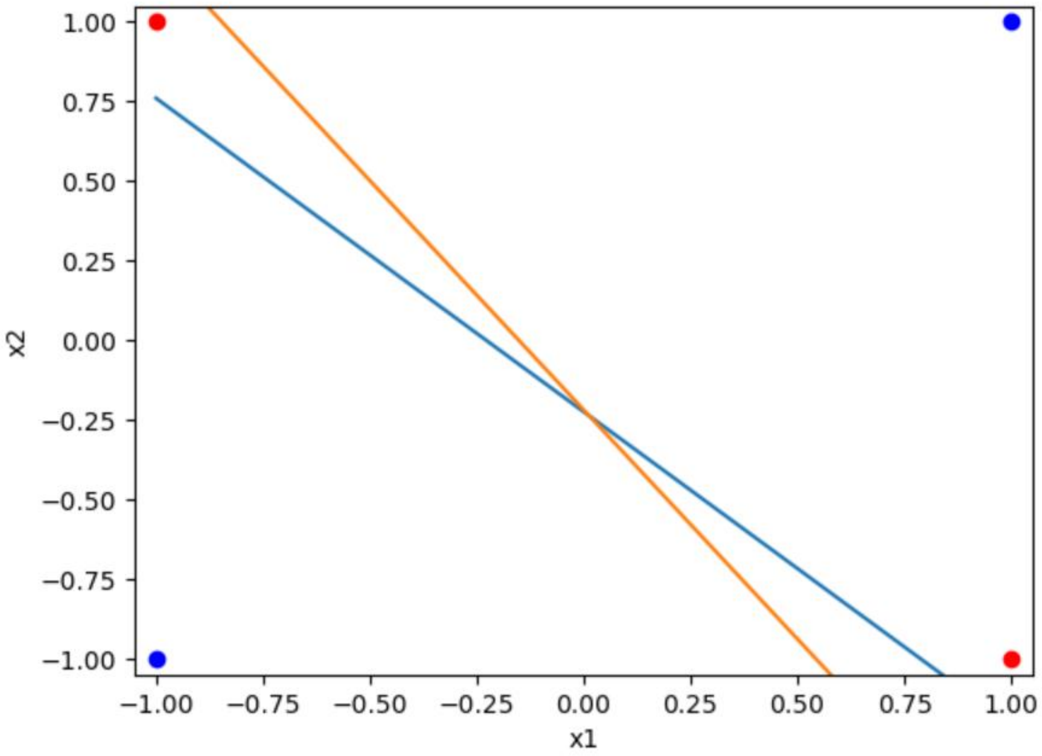
```
W2
array([[-0.48098277,  0.45515249]])
```

```
b2
array([[ 0.15708682]])
```

# GRAFICAR W1 Y B1

**MLP_XOR.ipynb**

```
ph = dibuPtos_y_2Rectas(X,T, 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(X)):    #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
```

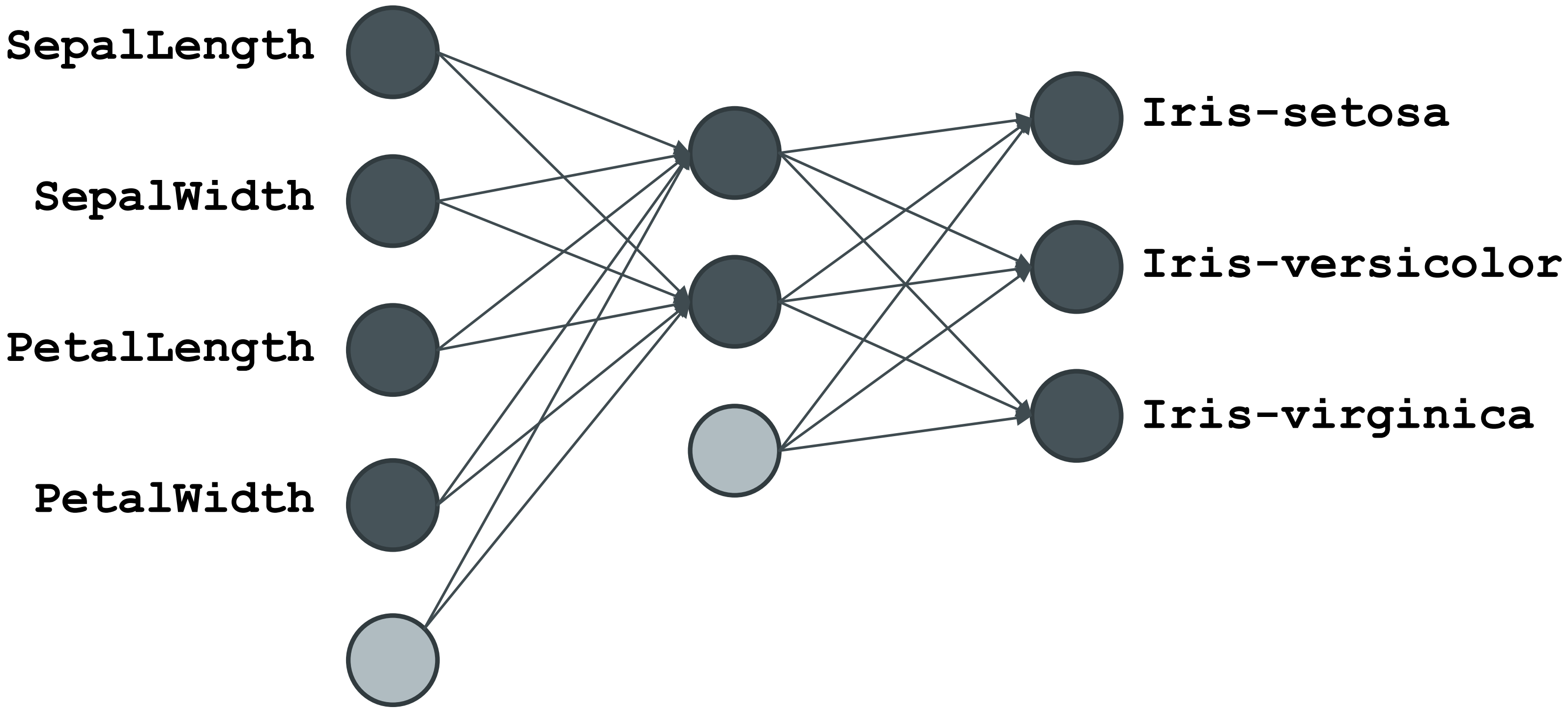
Ver
***MLP_XOR.ipynb***

# EJEMPLO: CLASIFICACIÓN DE FLORES DE IRIS

| Id | sepallength | 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



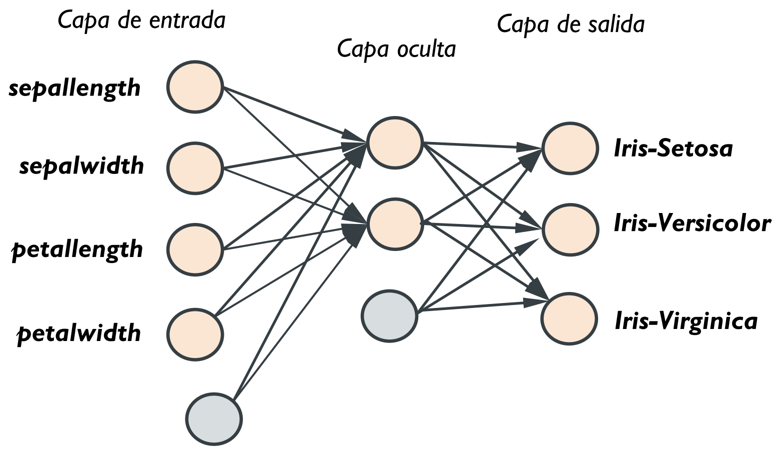
# CLASIFICACIÓN DE FLORES DE IRIS

**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]]
```

**T**

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



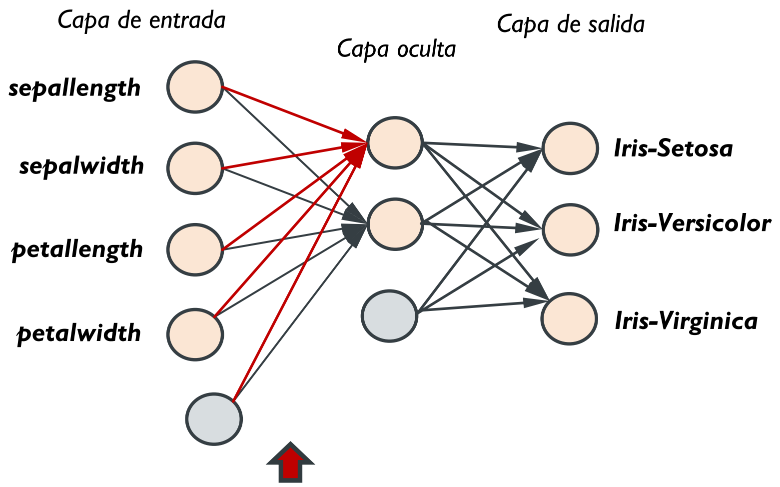
# CLASIFICACIÓN DE FLORES DE IRIS

**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]]
```

**T**

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




```
[[ 0.15, -0.13,  0.23, -0.45],
 [-0.29, -0.41, -0.19,  0.37]]
```

**W1**

```
[[-0.45],
 [-0.10]]
```

**b1**

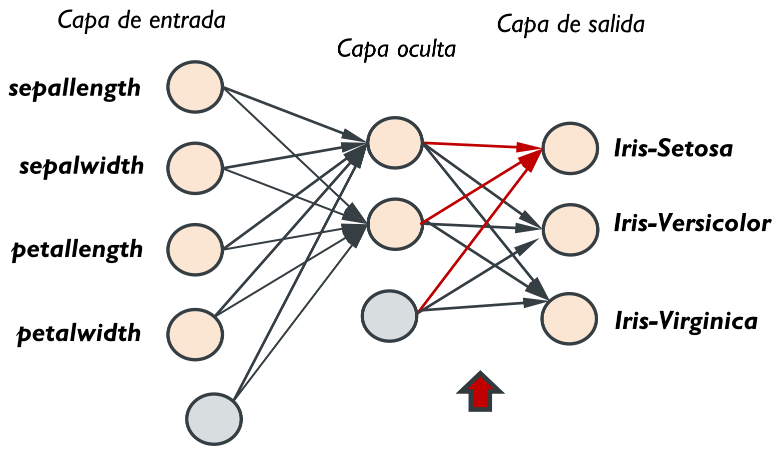
# CLASIFICACIÓN DE FLORES DE IRIS

**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]]
```

**T**

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




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

**W2**

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

**b2**

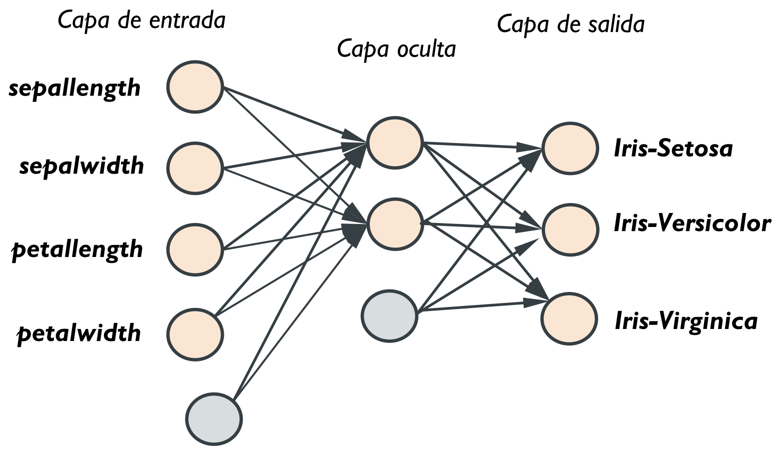
# CLASIFICACIÓN DE FLORES DE IRIS

**X**

**T**

```
[[-1.73,-0.05,-1.38,-1.31],      [[1,0,0],
 [-0.37,-1.62, 0.22, 0.18],       [0,1,0],
 [ 1.11,-0.05, 0.93, 1.54],       [0,0,1],
 [-0.99, 0.39,-1.44,-1.31],       [1,0,0],
 [ 1.73, 1.29, 1.46, 1.81]]       [0,0,1]]
```

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




Ingresar el primer ejemplo a la red y calcular su salida

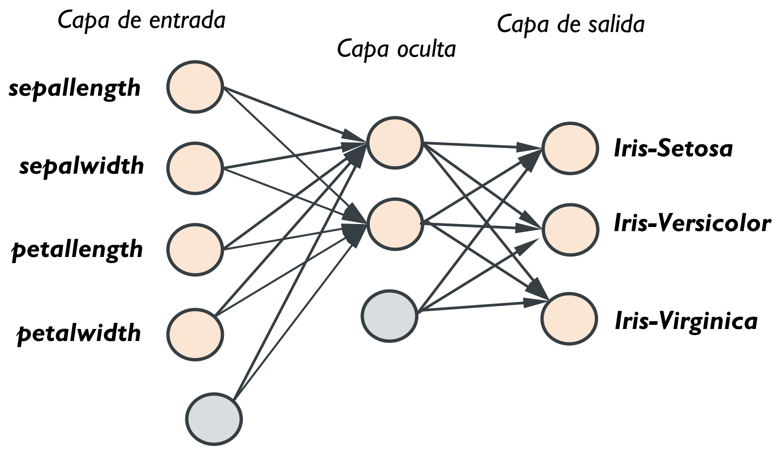
# CALCULANDO LA SALIDA DE LA CAPA OCULTA

**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]]
```

**T**

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




- Salida de la capa oculta

```
netasH = W1 * x.T + b1
```

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

$x^T$

```
[[ 0.15, -0.13,  0.23, -0.45],     [[-1.73],     [[-0.45],     [[-0.4309]
 [-0.29, -0.41, -0.19,  0.37]]  *   [-0.05],  +   [-0.10]]  =   [ 0.1997]]
                                    [-1.38],
                                    [-1.31]]
          W1                                        b1
```

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

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

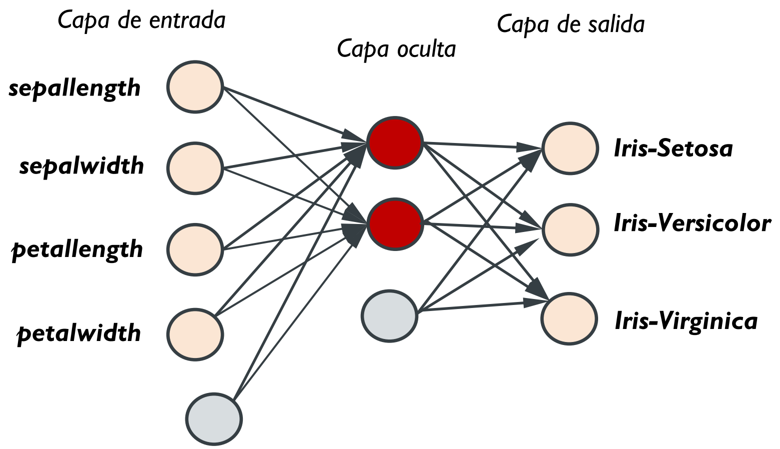
# CALCULANDO LA SALIDA DE LA CAPA OCULTA

**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]]
```

**T**

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




- Salida de la capa oculta

```
netasH = W1 * x.T + b1
```

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

**W1**

```
[[ 0.15, -0.13,  0.23, -0.45],
 [-0.29, -0.41, -0.19,  0.37]]
```

*

**$x^T$**

```
[[-1.73],
 [-0.05],
 [-1.38],
 [-1.31]]
```

\+

**b1**

```
[[-0.45],
 [-0.10]]
```

=

```
[[-0.4309]
 [ 0.1997]]
```

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

```
salidasH = 2.0/(1+np.exp(-2*netasH))-1 =
```

```
[[-0.40607318]
 [ 0.19708699]]
```

$$i_{pj} = f_{j}^{h}(neta_{pj}^{h})$$

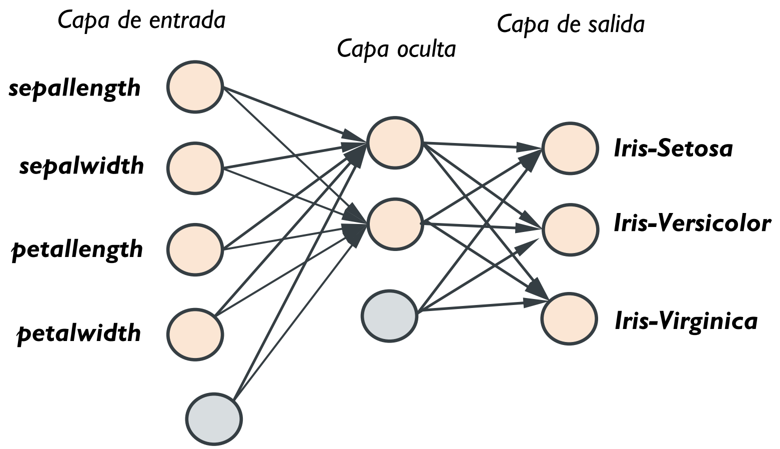
# CALCULANDO LA SALIDA DE LA RED (CAPA DE SALIDA)

**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]]
```

**T**

```
[[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'
```

| W2 | | salidasH | | b2 | | netasO |
|---|---|---|---|---|---|---|
| [[-4.79, -0.99],<br>[ 0.38, 0.09],<br>[ 2.40, -0.38]] | * | [[-0.40607318]<br>[ 0.19708699]] | + | [[-0.29],<br>[-0.17],<br>[-0.27]] | = | [[ 1.4599744 ]<br>[-0.30656998]<br>[-1.31946868]] |

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

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

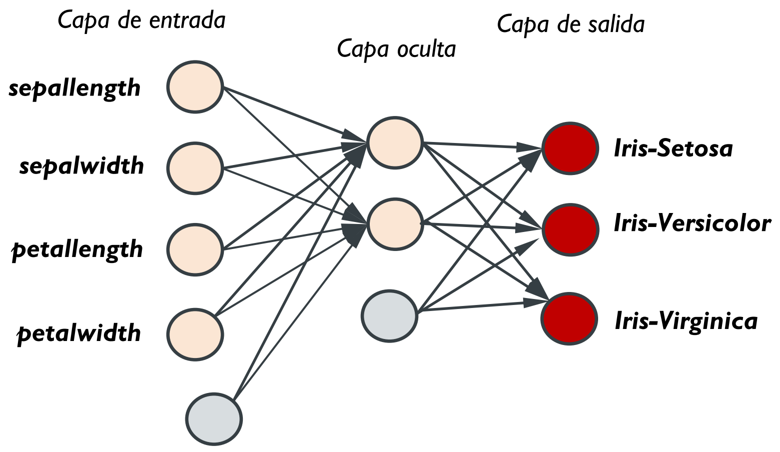
# CALCULANDO LA SALIDA DE LA RED (CAPA DE SALIDA)

**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]]
```

**T**

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




- Salida de red

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

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

| W2 | | salidasH | | b2 | | netasO |
|---|---|---|---|---|---|---|
| [[-4.79, -0.99],<br>[ 0.38, 0.09],<br>[ 2.40, -0.38]] | * | [[-0.40607318]<br>[ 0.19708699]] | + | [[-0.29],<br>[-0.17],<br>[-0.27]] | = | [[ 1.4599744 ]<br>[-0.30656998]<br>[-1.31946868]] |

```
salidasO = 1 / (1+np.exp(-netasO)) = [[0.81152876]
                                      [0.42395219]
                                      [0.2109067 ]]
```

$$neta^o_{pk} = \sum_{j=1}^{L} w^o_{kj} i_{pj} + \theta^o_k$$

$$y_{pk} = f^o_k(neta^o_{pk})$$

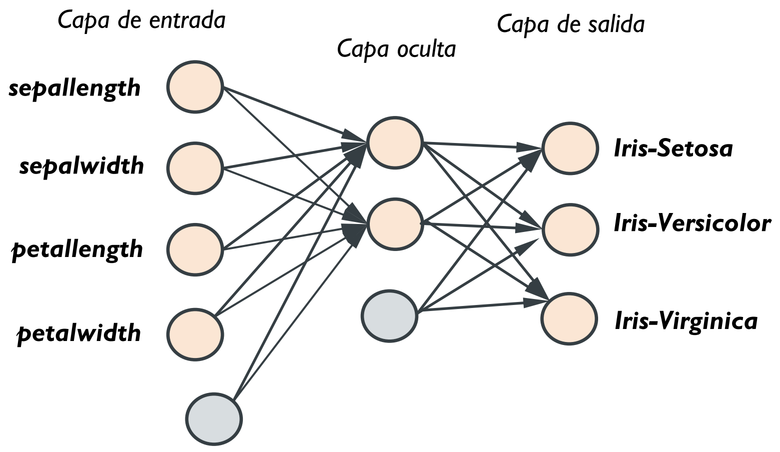
# ERROR DE LA RED PARA ESTE EJEMPLO

**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]]
```

**T**

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




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

- Error en la respuesta de la red para este ejemplo

```
ErrorSalida =  t.T - salidasO
```

```
                [[1.0]       [[0.81152876]        [[ 0.18847124]
ErrorSalida =    [0.0]   -    [0.42395219]    =    [-0.42395219]
                 [0.0]]       [0.2109067 ]]        [-0.2109067 ]]

                               salidasO
```

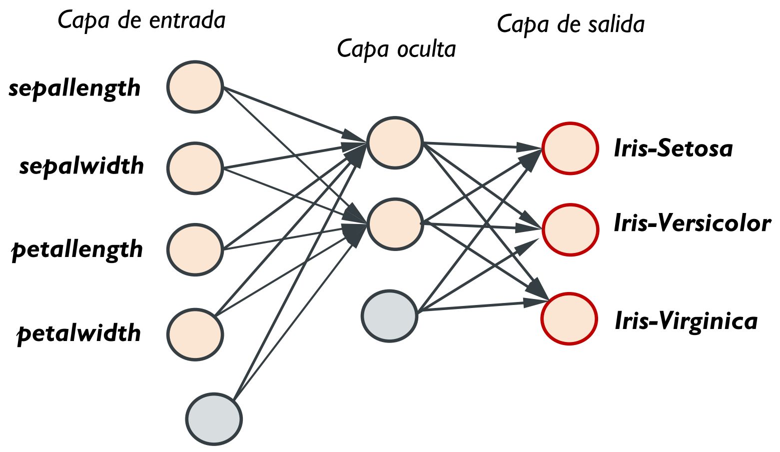
# FACTORES DE CORRECCIÓN DE LOS PESOS

**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]]
```

**T**

```
[[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**

```
salidasO*(1-salidasO)
```

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

```
          [[ 0.18847124]        [[0.15294983]        [[ 0.02882664]
deltaO =   [-0.42395219]   .*    [0.24421673]    =    [-0.10353622]
           [-0.2109067 ]]        [0.16642507]]        [-0.03510016]]
```

$$\delta_{pk}^{o} = (t_{pk} - y_{pk}) f_k^{o\prime}(neta_{pk}^{o})$$

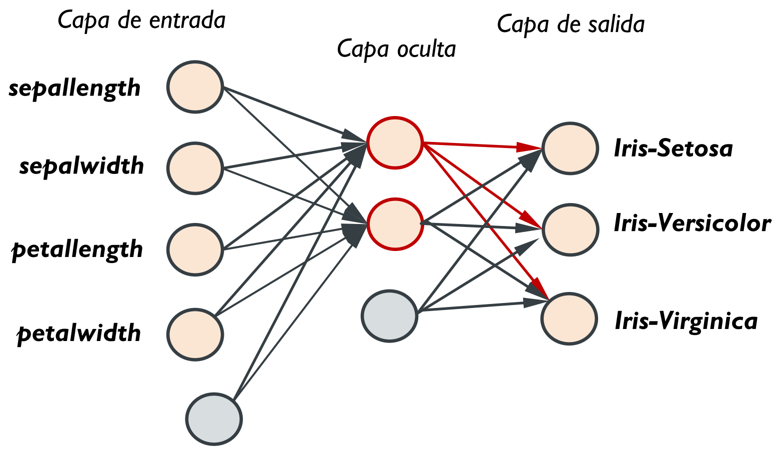
# FACTORES DE CORRECCIÓN DE LOS PESOS

**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]]
```

**T**

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




- **Factores para corregir W1 y b1**

```
(1-salidasH**2)
```

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

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

```
deltaH =  [[0.83510457]    .*  ( [[-4.79  0.38  2.4 ]   @  [[ 0.02882664]  )
           [0.96115672]]          [-0.99  0.09 -0.38]]      [-0.10353622]
                                                            [-0.03510016]]
```

```
deltaH =  [[-0.21851662]
           [-0.02356619]]
```

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

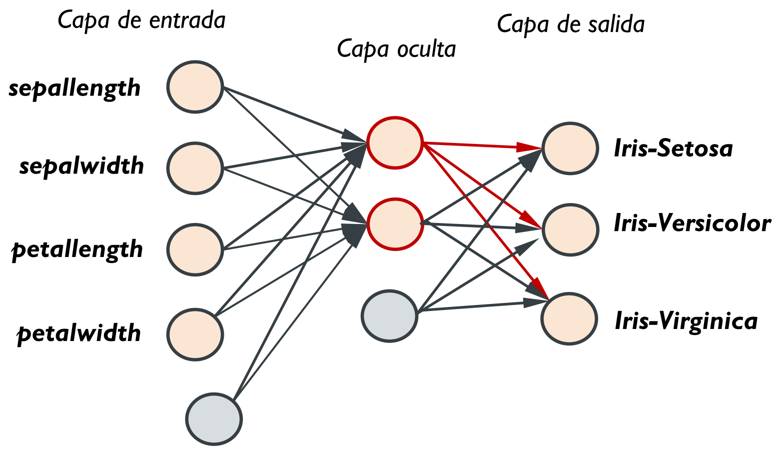
# FACTORES DE CORRECCIÓN DE LOS PESOS

**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]]
```

**T**

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




- **Factores para corregir W1 y b1**

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

```
(1-salidasH**2)
deltaH = deriv_FunH .* (W2.T @ deltaO)
```

```
deltaH =  [[0.83510457]      .*  ( [[-4.79  0.38  2.4 ]   @   [[ 0.02882664]
           [0.96115672]]              [-0.99  0.09 -0.38]]        [-0.10353622]
                                                                  [-0.03510016]] )
```

```
deltaH =  [[-0.21851662]
           [-0.02356619]]
```

*Note que las derivadas de ambas funciones de activación sigmoides (`derivada_FunO` y `deriv_FunH`) son siempre positivas y actúan como factor de escala*

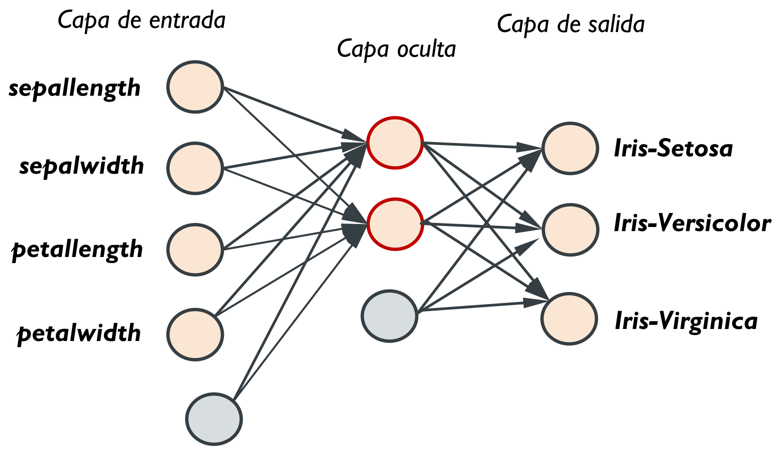
# CORRECCIÓN DE LOS PESOS

**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]]
```

**T**

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




- **Modificación de W2 y b2**

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

```
W2 = W2 + alfa * deltaO @ salidasH.T
```

```
W2 =  [[-4.79, -0.99],                  [[ 0.02882664]                                              [[-4.79, -0.99]
       [ 0.38,  0.09], + alfa *  (        [-0.10353622] @ [[-0.406073], [0.197087]] )  =             [ 0.38,  0.09]
       [ 2.40, -0.38]]                    [-0.03510016]]                                              [ 2.4 , -0.38]]
```

```
b2 = b2 + alfa * deltaO =  [[-0.29],           [[ 0.02882664]         [[-0.29],
                            [-0.18], + alfa *   [-0.10353622]    =     [-0.18],
                            [-0.27]]            [-0.03510016]]         [-0.27]]
```

# CORRIGIENDO DE LOS PESOS

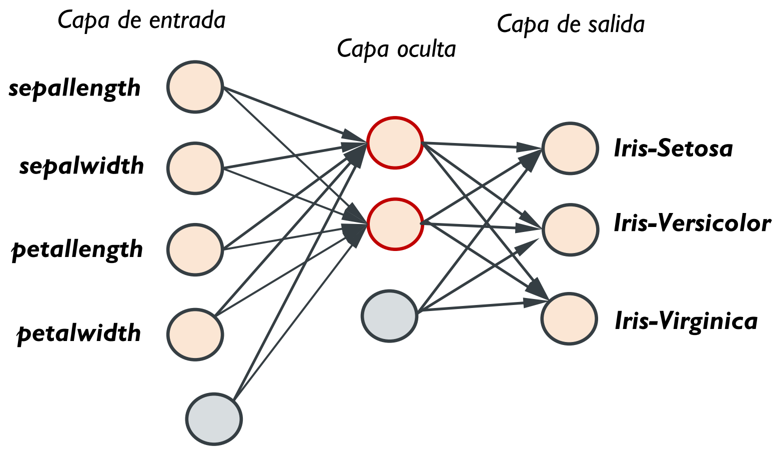
**MLP_IRIS_algBPN.ipynb**

**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]]
```

**T**

```
[[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
```

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

```
W1 = [[ 0.15,-0.13,0.23,-0.45],
      [-0.29,-0.41,-0.19,0.37]] + alfa * ( [[-0.21851662]
                                            [-0.02356619]] @ [[-1.73,-0.05,-1.38,-1.31] )

W1 = [[ 0.19 -0.13  0.26 -0.42]
      [-0.29 -0.41 -0.19  0.37]]

b1 = b1 + alfa * deltaH = [[-0.45],  + alfa * [[-0.21851662]  = [[-0.47],
                          [-0.10]]            [-0.02356619]]     [-0.1 ]]
```

# SI SE INGRESA EL MISMO EJEMPLO LUEGO DE MODIFICAR LOS PESOS DE LA RED …

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

```
salidaO =
 [[0.89080126]
 [0.40850633]
 [0.16423515]]
ErrorSalida =
 [[ 0.10919874]
 [-0.40850633]
 [-0.16423515]]
```

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

```
Error inicial =  0.25973850457967945
Error luego de la correccion =  0.2057749693024508
```

Antes de modificar los pesos de la red

```
salidaO =
 [[0.81152876]
 [0.42395219]
 [0.2109067 ]]

ErrorSalida =
 [[ 0.18847124]
 [-0.42395219]
 [-0.2109067 ]]
```

*Ver*
***MLP_IRIS_algBPN.ipynb***

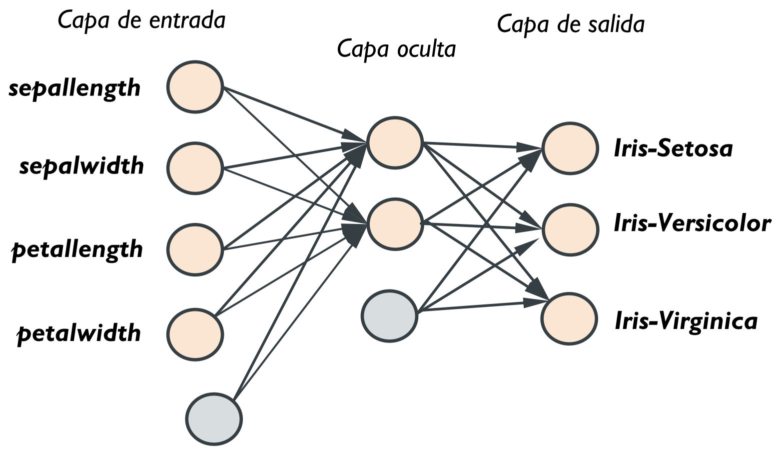
# CLASIFICACIÓN DE FLORES DE IRIS

**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]]
```

**T**

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



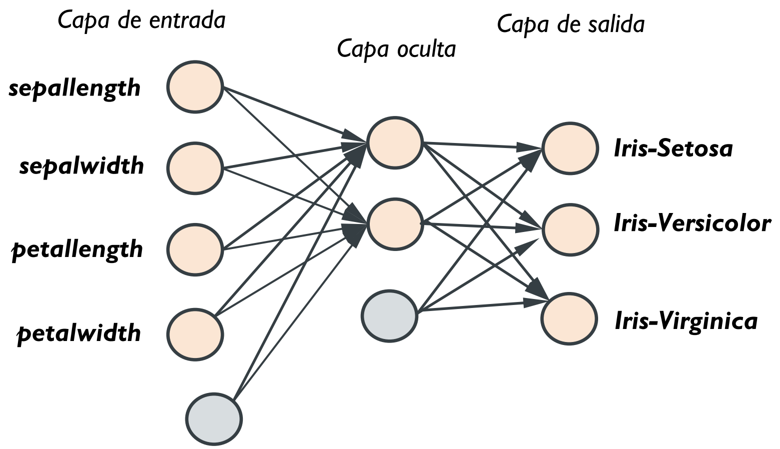
# CLASIFICACIÓN DE FLORES DE IRIS

**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]]
```

**T**

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




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

Ingresar el primer ejemplo a la red y calcular su salida

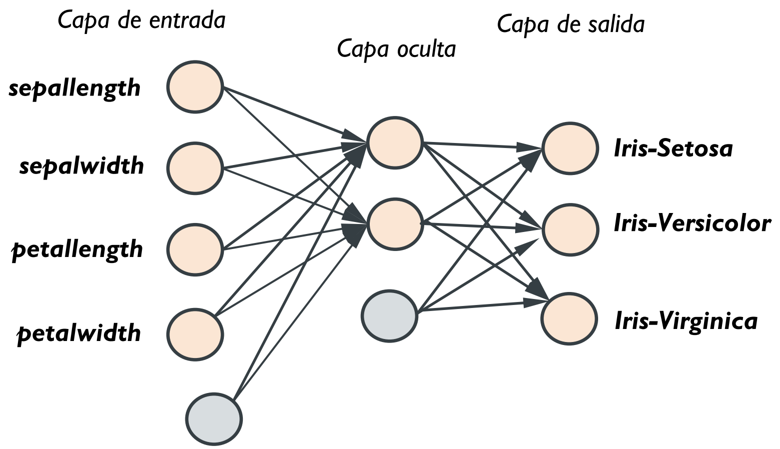
# CALCULANDO LA SALIDA DE LA CAPA OCULTA

**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]]
```

**T**

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




- Salida de la capa oculta

```
netasH = W1 * x.T + b1
```

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

**W1**

```
[[ 0.15, -0.13,  0.23, -0.45],
 [-0.29, -0.41, -0.19,  0.37]]
```

*

$x^T$

```
[[-1.73],
 [-0.05],
 [-1.38],
 [-1.31]]
```

\+

**b1**

```
[[-0.45],
 [-0.10]]
```

=

```
[[-0.4309]
 [ 0.1997]]
```

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

$$i_{pj} = f_{j}^{h}(neta_{pj}^{h})$$

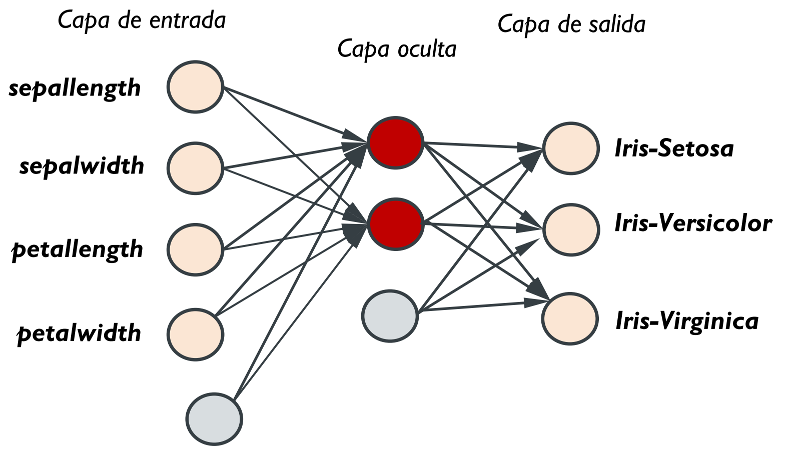
# CALCULANDO LA SALIDA DE LA CAPA OCULTA

**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]]
```

**T**

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




- Salida de la capa oculta

```
netasH = W1 * x.T + b1
```

FunH='tanh' ; FunO='softmax'

```
[[ 0.15, -0.13,  0.23, -0.45],     [[-1.73],     [[-0.45],     [[-0.4309]
 [-0.29, -0.41, -0.19,  0.37]]  *   [-0.05],  +   [-0.10]]  =   [ 0.1997]]
                                    [-1.38],
                                    [-1.31]]
```

W1 · $x^T$ · b1

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

```
salidasH = 2.0/(1+np.exp(-2*netasH))-1 =   [[-0.40607318]
                                            [ 0.19708699]]
```

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

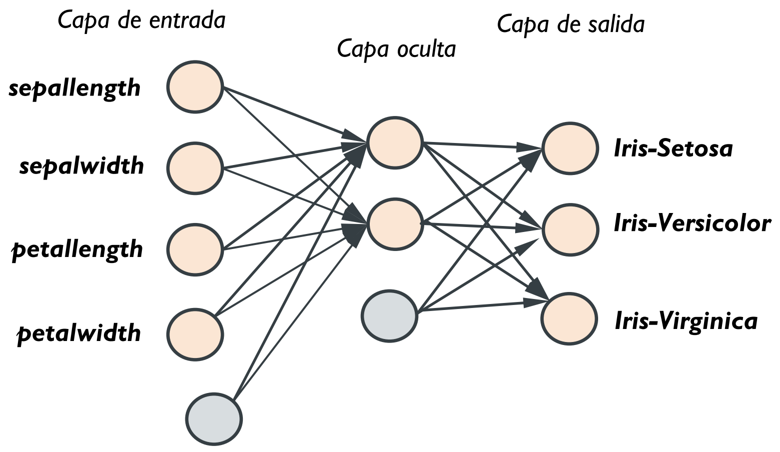
# CALCULANDO LA SALIDA DE LA RED (CAPA DE SALIDA)

**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]]
```

**T**

```
[[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='softmax'
```

W2 * salidasH + b2 = netasO

```
W2:
[[-4.79, -0.99],
 [ 0.38,  0.09],
 [ 2.40, -0.38]]

salidasH:
[[-0.40607318]
 [ 0.19708699]]

b2:
[[-0.29],
 [-0.17],
 [-0.27]]

netasO:
[[ 1.4599744 ]
 [-0.30656998]
 [-1.31946868]]
```

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

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

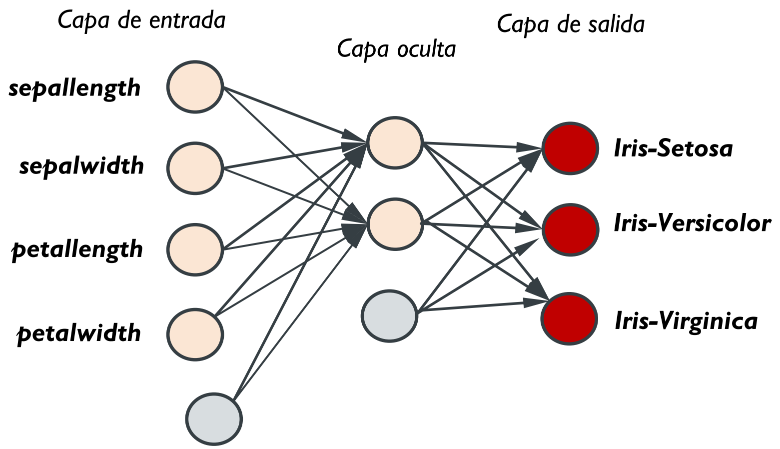
# CALCULANDO LA SALIDA DE LA RED (CAPA DE SALIDA)

**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]]
```

**T**

```
[[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='softmax'
```

| W2 | | salidasH | | b2 | | netasO |
|---|---|---|---|---|---|---|
| [[-4.79, -0.99],<br>[ 0.38, 0.09],<br>[ 2.40, -0.38]] | * | [[-0.40607318]<br>[ 0.19708699]] | + | [[-0.29],<br>[-0.17],<br>[-0.27]] | = | [[ 1.4599744 ]<br>[-0.30656998]<br>[-1.31946868]] |

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

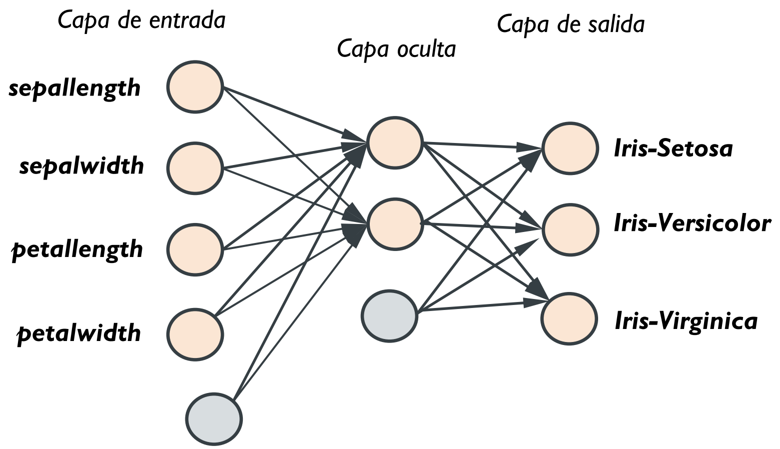
$$y_{pk} = f_k^{o}(neta_{pk}^{o})$$

```
salidasO = np.exp(netasO)/(np.sum(np.exp(netasO))) = [[0.81103285]
                                                      [0.13862385]
                                                      [0.0503433 ]]
```

# ERROR DE LA RED PARA ESTE EJEMPLO

**X** **T**

```
[[-1.73,-0.05,-1.38,-1.31],      [[1,0,0],
 [-0.37,-1.62, 0.22, 0.18],       [0,1,0],
 [ 1.11,-0.05, 0.93, 1.54],       [0,0,1],
 [-0.99, 0.39,-1.44,-1.31],       [1,0,0],
 [ 1.73, 1.29, 1.46, 1.81]]       [0,0,1]]
```




- Valor de la función de costo para este ejemplo

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

```
ErrorSalida =  Ti.T * (- np.log(salidasO+EPS))
```

```
                [[1.0]           [[-0.20944672]      [[ 0.20944672]
ErrorSalida =    [0.0]   *  (-1)  [-1.9759911 ]   =   [-0.        ]
                 [0.0]]           [-2.9888898 ]]      [-0.        ]]
```

```
(- np.log(salidasO+EPS))
```

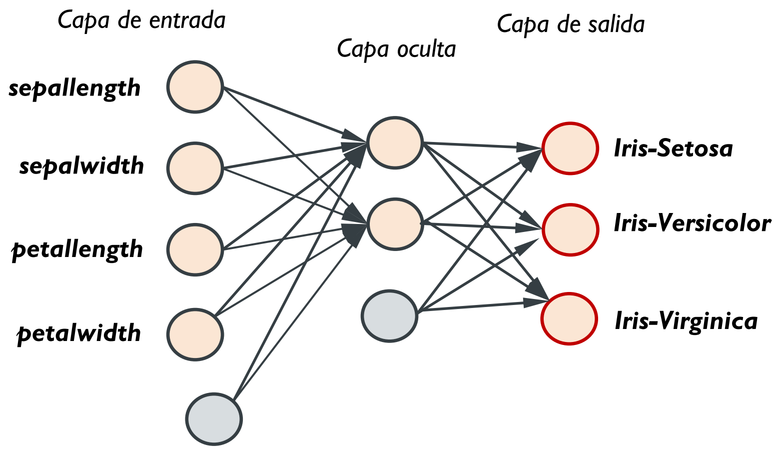
# FACTORES DE CORRECCIÓN DE LOS PESOS

**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]]
```

**T**

```
[[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 = (Ti.T - salidasO)
```

```
          [[1.0]       [[0.81103285]       [[ 0.18896715]
deltaO =   [0.0]   -    [0.13862385]   =    [-0.13862385]
           [0.0]]       [0.0503433 ]]       [-0.0503433 ]]
```

$$\delta_{pk}^{o} = (t_{pk} - y_{pk})$$

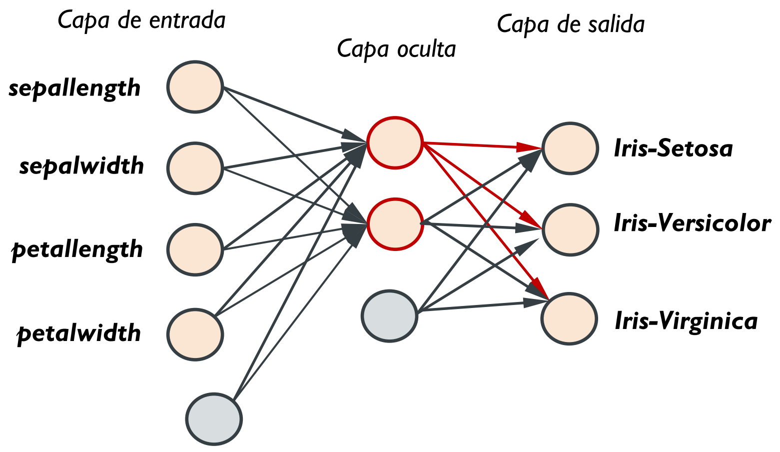
# FACTORES DE CORRECCIÓN DE LOS PESOS

**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]]
```

**T**

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




- **Factores para corregir W1 y b1**

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

```
(1-salidasH**2)

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

deltaH =  [[0.83510457]    .*  ( [[-4.79  0.38  2.4 ]  @  [[ 0.18896715]
           [0.96115672]]           [-0.99  0.09 -0.38]]     [-0.13862385]
                                                            [-0.0503433 ]] )

deltaH =  [[-0.90078858]
           [-0.17341495]]
```

$$\delta^h_{pj} = f^{h}_{j}{}'(neta^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

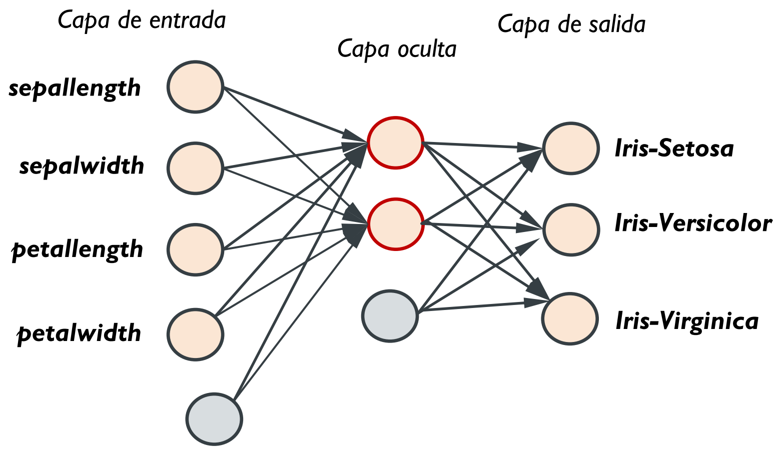
# CORRECCIÓN DE LOS PESOS

**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]]
```

**T**

```
[[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 + alfa * deltaO @ salidasH.T
```

```
W2 = [[-4.79, -0.99],                ( [[ 0.18896715]                                  )   [[ -4.8   -0.99]
      [ 0.38,  0.09],  + alfa *      (  [-0.13862385]  @ [[-0.406073], [ 0.197087]]    ) =  [ 0.39   0.09]
      [ 2.40, -0.38]]                (  [-0.0503433 ]]                                 )    [ 2.4   -0.38]]
```

```
b2 = b2 + alfa * deltaO =  [[-0.29],          [[ 0.18896715]        [[-0.27],
                            [-0.17], + alfa *  [-0.13862385]   =     [-0.18],
                            [-0.27]]           [-0.0503433 ]]        [-0.28]]
```

# CORRIGIENDO DE LOS PESOS

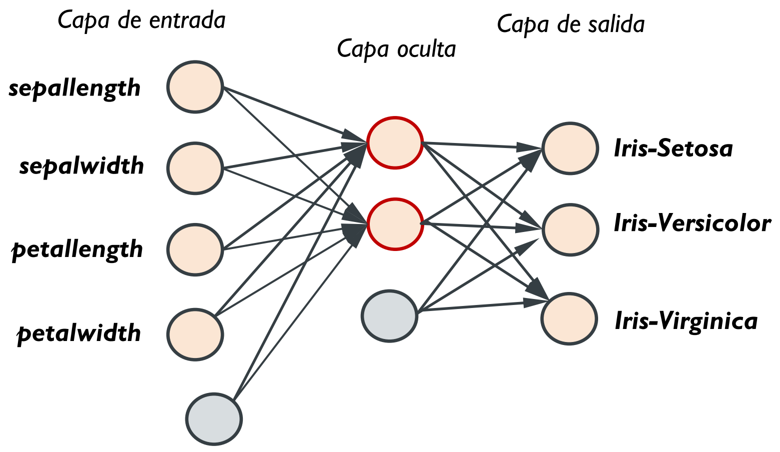
**MLP_IRIS_algBPN_softmax.ipynb**

**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]]
```

**T**

```
[[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
```

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

```
W1 = [[ 0.15,-0.13,0.23,-0.45],
      [-0.29,-0.41,-0.19,0.37]] + alfa * ( [[-0.90078858]
                                             [-0.17341495]]  @  [[-1.73,-0.05,-1.38,-1.31] )

W1 = [[ 0.31 -0.13  0.35 -0.33]
      [-0.26 -0.41 -0.17  0.39]]

b1 = b1 + alfa * deltaH = [[-0.45],
                           [-0.10]] + alfa * [[-0.90078858]
                                              [-0.17341495]]  =  [[-0.54],
                                                                  [-0.12]]
```

# SI SE INGRESA EL MISMO EJEMPLO LUEGO DE MODIFICAR LOS PESOS DE LA RED …

```
netasH = W1 @ xi.T + b1
salidasH = 2.0/(1+np.exp(-2*netasH))-1
netasO =  W2 @ salidasH + b2
salidasO = np.exp(netasO)/(np.sum(np.exp(netasO)))
print("salidaO = \n", salidasO)
ErrorSalidaNew = Ti.T-salidasO
print("ErrorSalida = \n", ErrorSalidaNew)
```

```
salidaO =
 [[0.97937129]
 [0.01757216]
 [0.00305655]]
ErrorSalida =
 [[ 0.02062871]
 [-0.01757216]
 [-0.00305655]]
```

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

```
Error inicial =  0.04386792904265554
Error luego de la correccion =  0.0007436667531155563
```

Antes de modificar los pesos de la red

```
salidaO =
 [[0.81103285]
 [0.13862385]
 [0.0503433 ]]
ErrorSalida =
 [[ 0.18896715]
 [-0.13862385]
 [-0.0503433 ]]
```

*Ver*
***MLP_IRIS.ipynb***

# SKLEARN - MLPCLASSIFIER

```
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)
Y_prob = clf.predict_proba(X_train)
```

MLPClassifier.html

# SKLEARN - MLPCLASSIFIER

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='sgd', learning_rate_init=0.05,
                    hidden_layer_sizes=(2,), random_state=1,
                    max_iter=2000, batch_size = 20,
                    tol=1.0e-05, n_iter_no_change = 30,
                    early_stopping = False, verbose = False,
                    activation='tanh')

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

Ver

***MLP_IRIS_RN.ipynb***

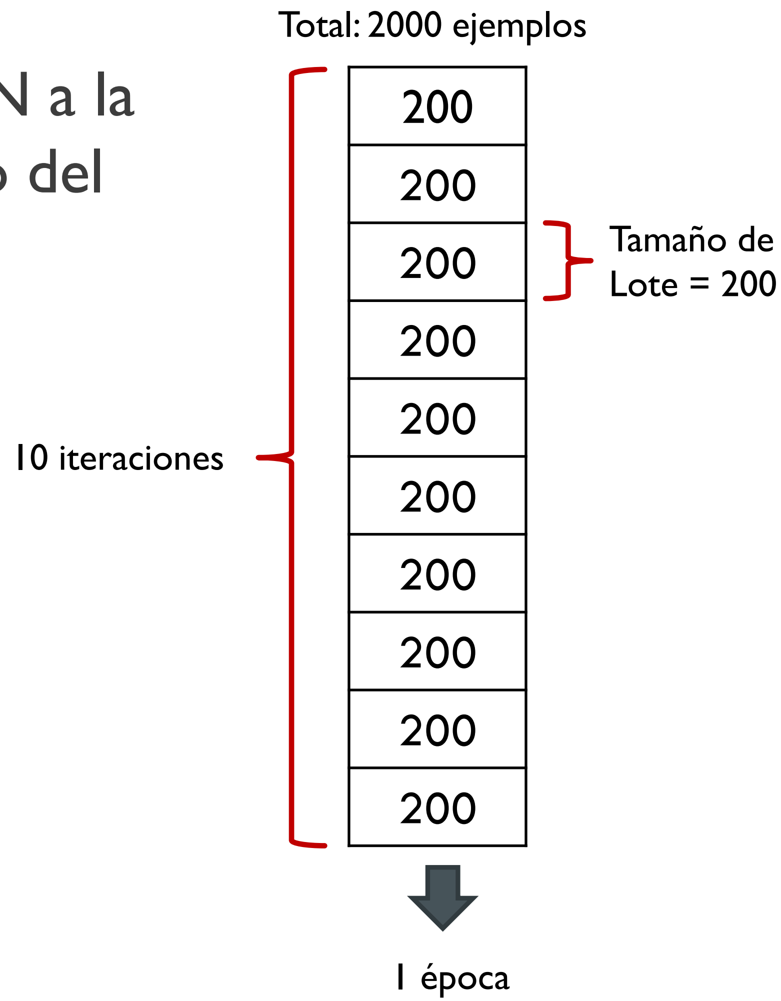
# DESCENSO DE GRADIENTE CON MINI-LOTE

- En lugar de ingresar los ejemplos de a uno, ingresamos N a la red y buscamos minimizar el error cuadrático promedio del lote.

- La función de costo será

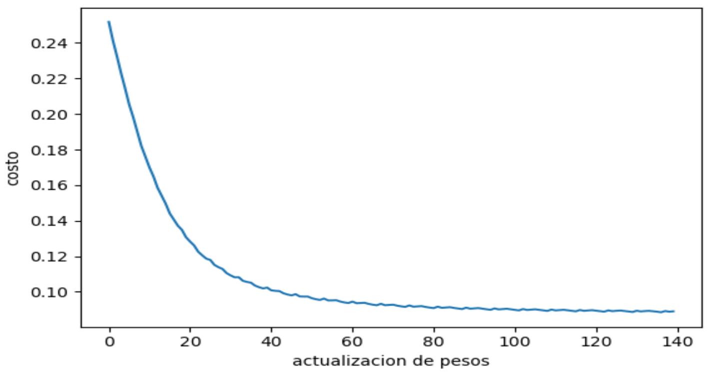
$$C = \frac{1}{N}\sum_{i=1}^{N}(t_i - f(neta_i))^2$$

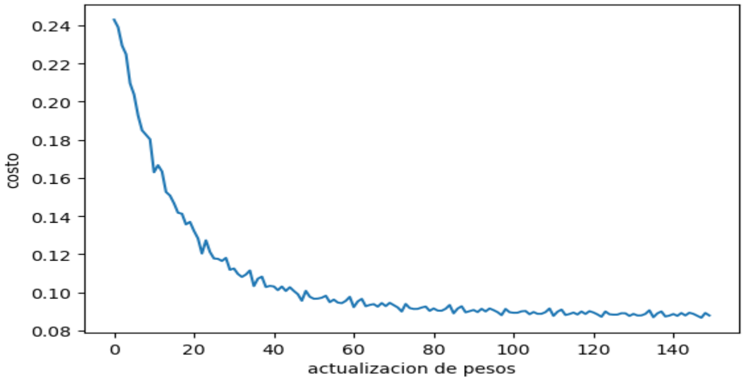
- N es la cantidad de ejemplos que conforman el lote.

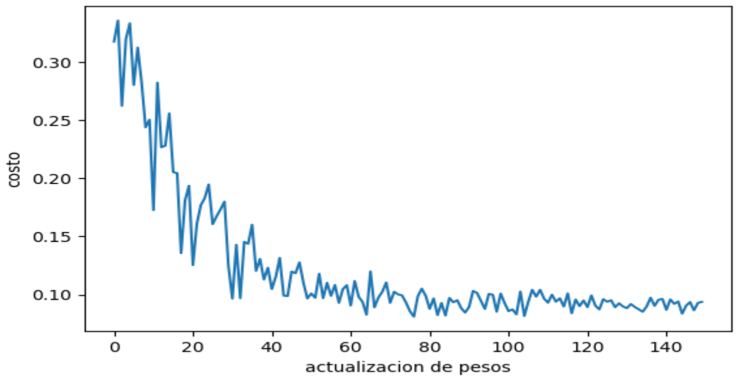


# DESCENSO DE GRADIENTE

| Batch | Mini-batch | Stochastic |
|---|---|---|
| Ingresa TODOS los ejemplos y luego actualiza los pesos. | Ingresa un LOTE de N ejemplos y luego actualiza los pesos | Ingresa UN ejemplo y luego actualiza los pesos |
| $C = \frac{1}{M}\sum_{i=1}^{M}(t_i - f(neta_i))^2$ | $C = \frac{1}{N}\sum_{i=1}^{N}(t_i - f(neta_i))^2 \quad N \ll M$ | $C = (\mathrm{t} - f(neta))^2$ |



# RECONOCEDOR DE DÍGITOS ESCRITOS A MANO

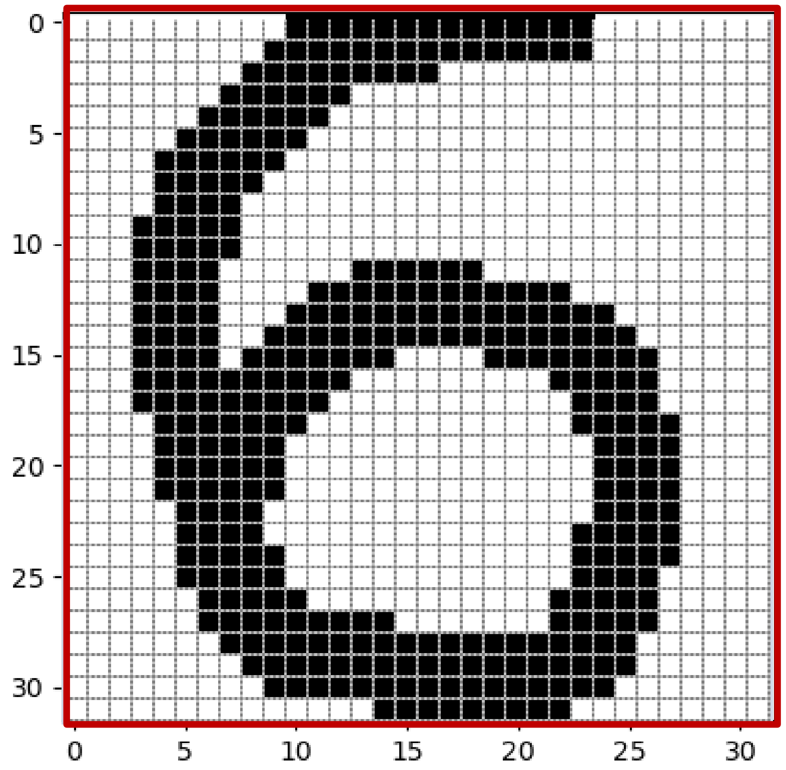
- Se desea entrenar un multiperceptrón para reconocer dígitos escritos a mano. Para ello se dispone de los mapas de bits correspondientes a 3823 dígitos escritos a mano por 30 personas diferentes en el archivo “optdigits_train.csv”.

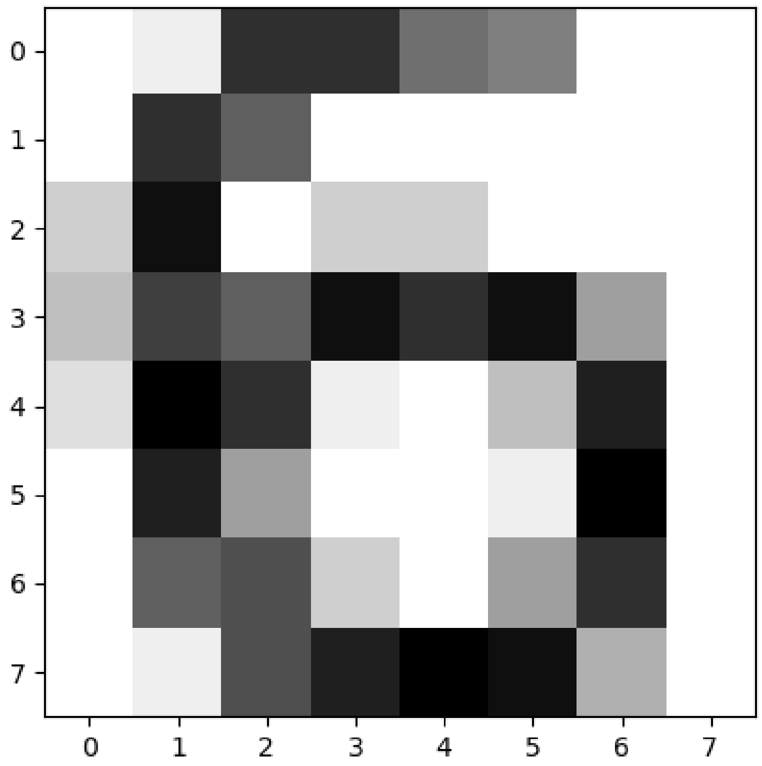
- El desempeño de la red será probado con los dígitos del archivo “optdigits_test.csv” escritos por otras 13 personas.

https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits

# “OPTDIGITS_TRAIN.CSV” Y “OPTDIGITS_TEST.CSV”

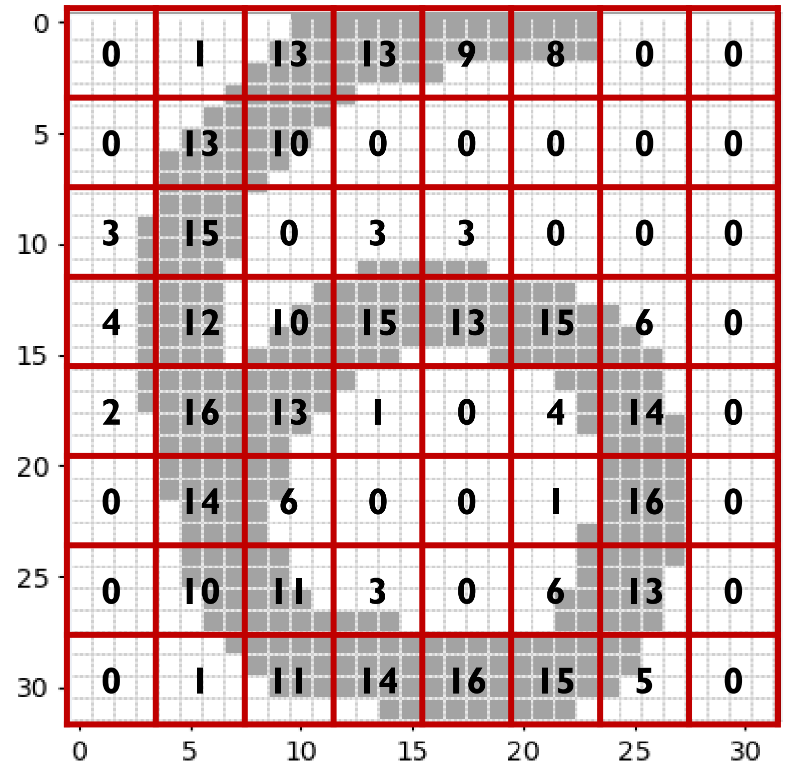
- Cada dígito está representado por una matriz numérica de 8x8

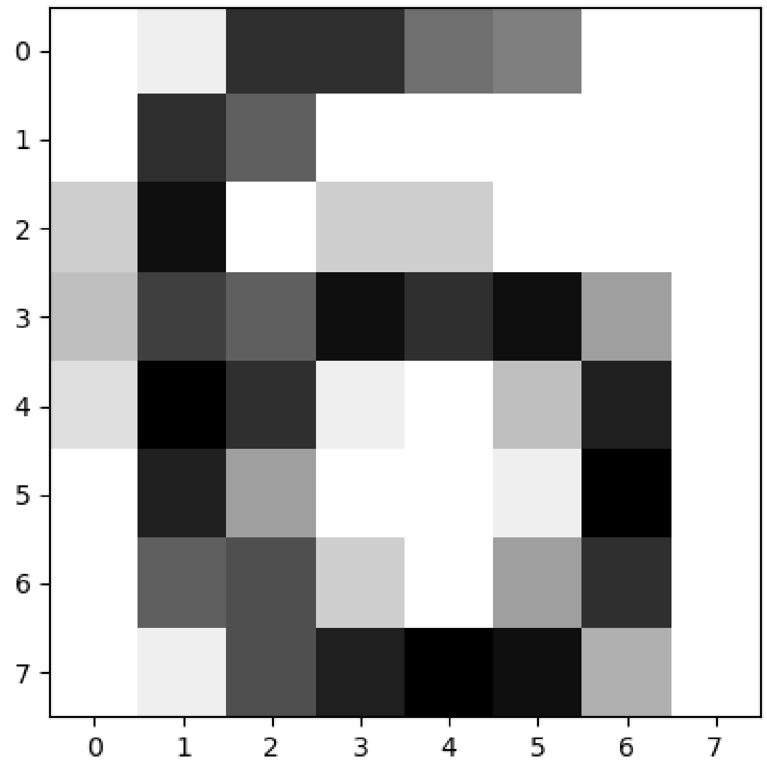


# “OPTDIGITS_TRAIN.CSV” Y “OPTDIGITS_TEST.CSV”

- 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:
[[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]]
```

El tiempo de entrenamiento es menor si se usa mini-lote

***MLP_MNIST_8x8.ipynb***
***MLP_MNIST_8x8_miniLote.ipynb***

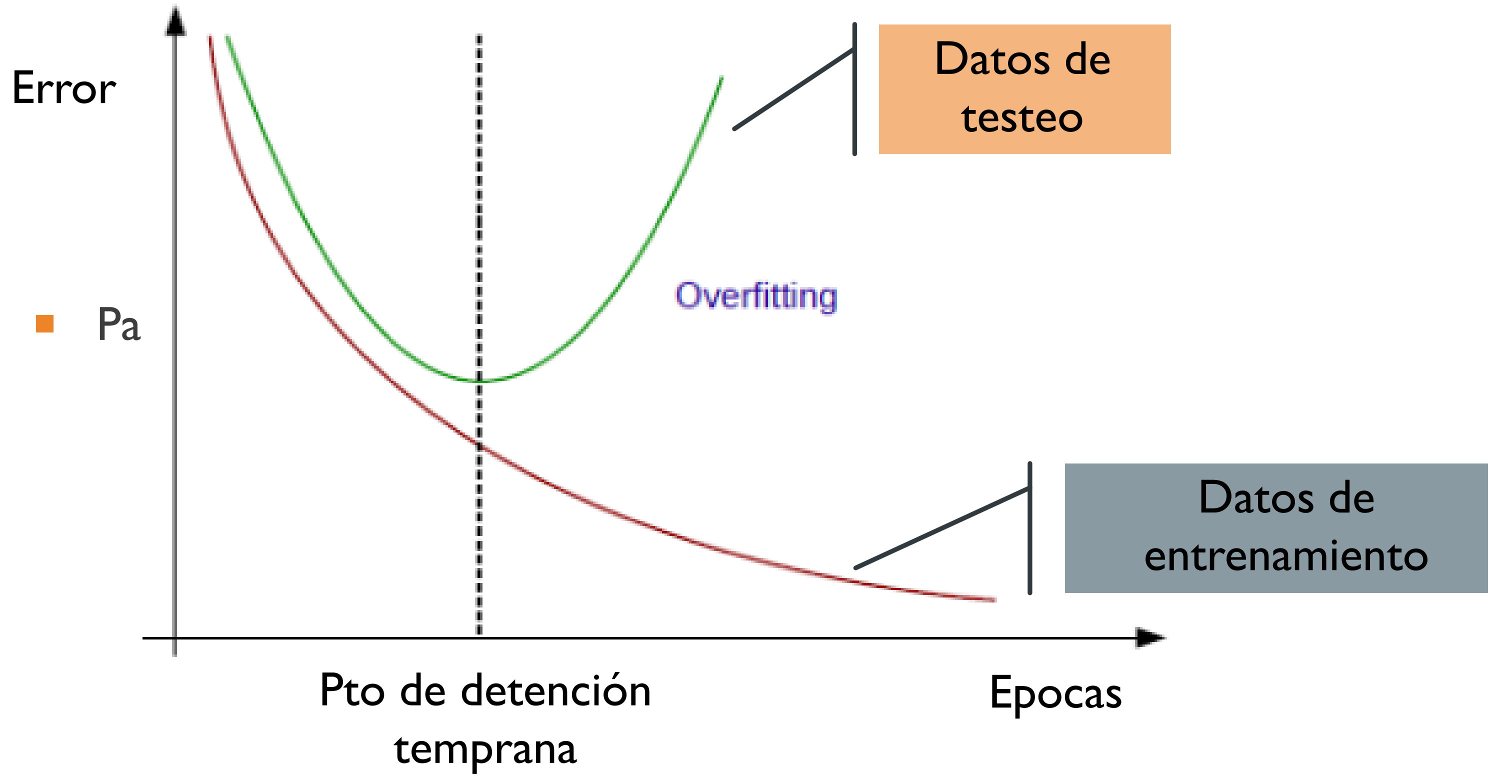
# SKLEARN - MLPCLASSIFIER

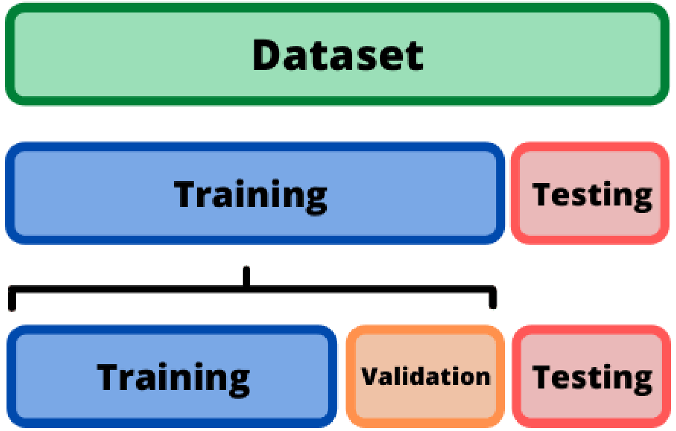
```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='sgd', learning_rate_init=0.05,
                    hidden_layer_sizes=(15,), random_state=1,
                    max_iter=2000, batch_size = 200,
                    tol=1.0e-05, n_iter_no_change = 30,
                    early_stopping = True, validation_fraction=0.2,
                    activation='tanh')

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

[MLPClassifier.html]()

# SOBREAJUSTE

- Pa



# SKLEARN - MLPCLASSIFIER

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='sgd', learning_rate_init=0.05,
                    hidden_layer_sizes=(15,), random_state=1,
                    max_iter=2000, batch_size = 200,
                    tol=1.0e-05, n_iter_no_change = 30,
                    early_stopping = True, validation_fraction=0.2,
                    activation='tanh')

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

*Ver*
***MLP_MNIST_RN.ipynb***

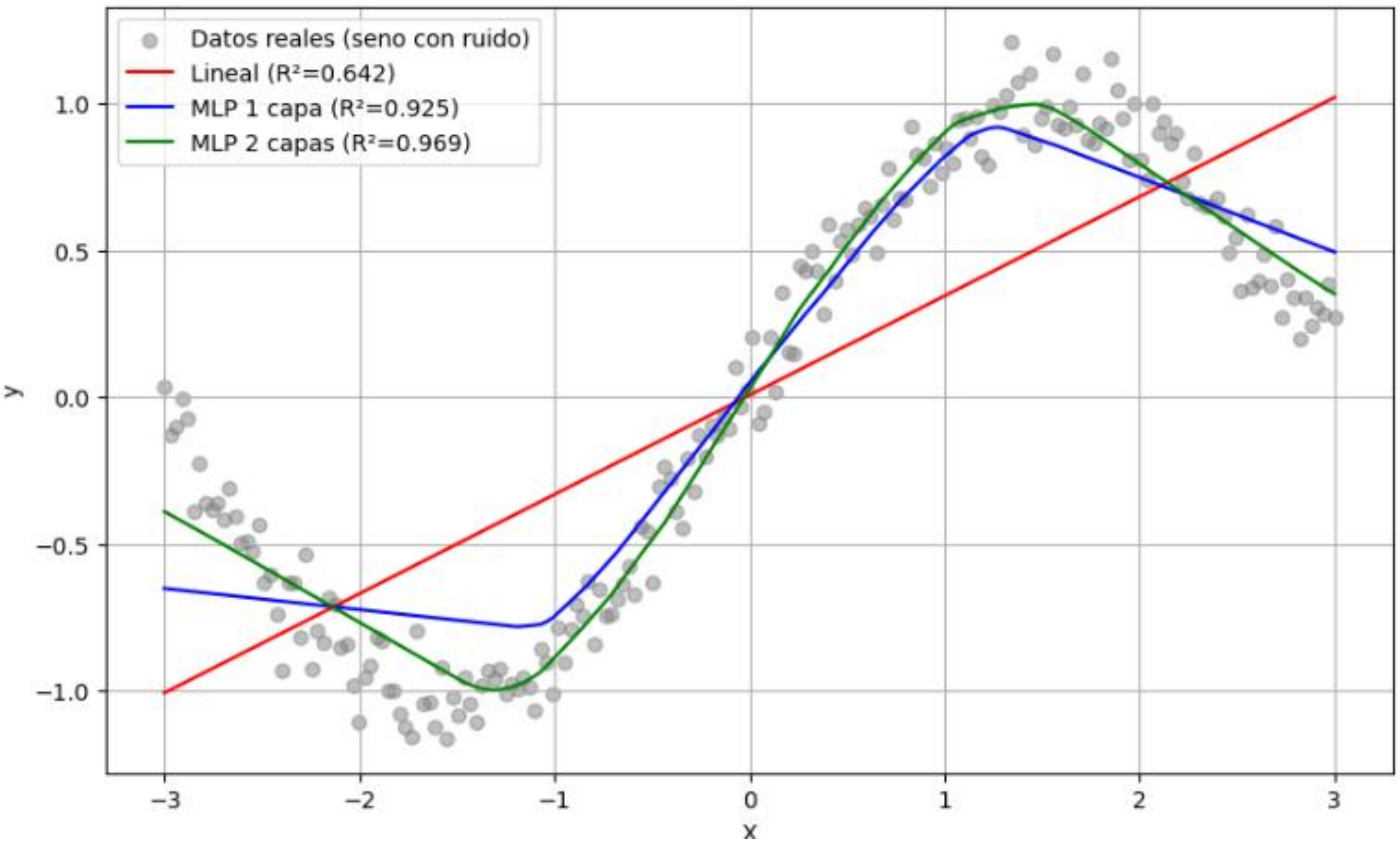
# SKLEARN - MLPREGRESSOR

```
from sklearn.neural_network import MLPRegressor

clf = MLPRegressor( solver='sgd',
                    learning_rate_init=0.15,
                    hidden_layer_sizes=(50),
                    max_iter=700, verbose=False,
                    tol=1.0e-09, activation = 'relu')

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

[MLPRegressor.html]()

MLPRegressor_hiperparam.ipynb

# MLPREGRESSOR – ARQUITECTURAS CON TAMAÑO DIFERENTE



*MLPRegressor_hiperparam.ipynb*

# MLPREGRESSOR – DISTINTAS FUNCIONES DE ACTIVACIÓN