

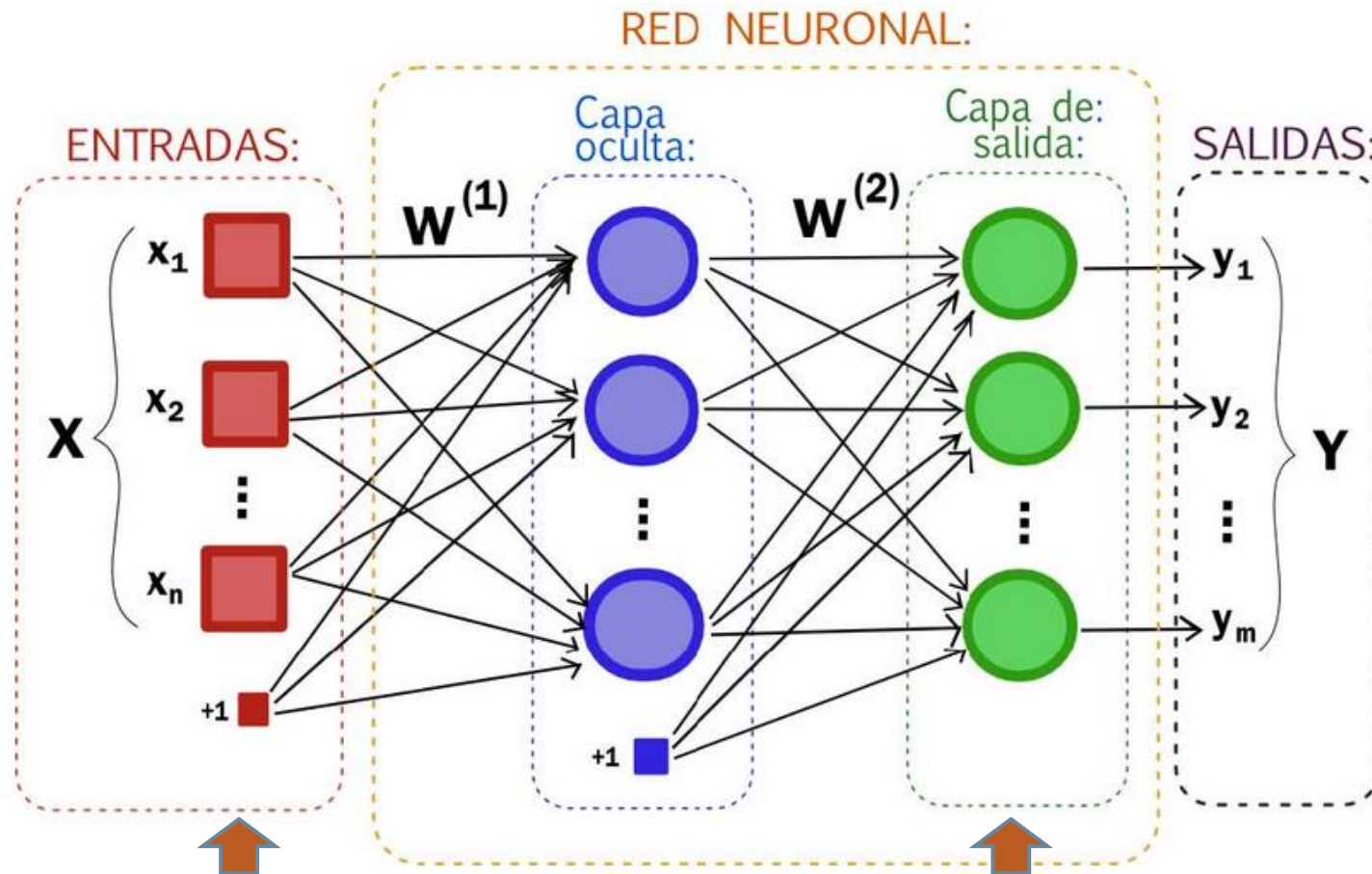
# Resumen

## Resolución de una tarea de clasificación

- Conjunto de datos etiquetados (aprendizaje supervisado)
- Definición de la arquitectura de la red
  - ▣ Número de capas y tamaño de cada una
  - ▣ Función de activación a usar en cada capa
- Entrenamiento
  - ▣ Función de error
  - ▣ Técnica de optimización para reducir el error
- Evaluar el modelo

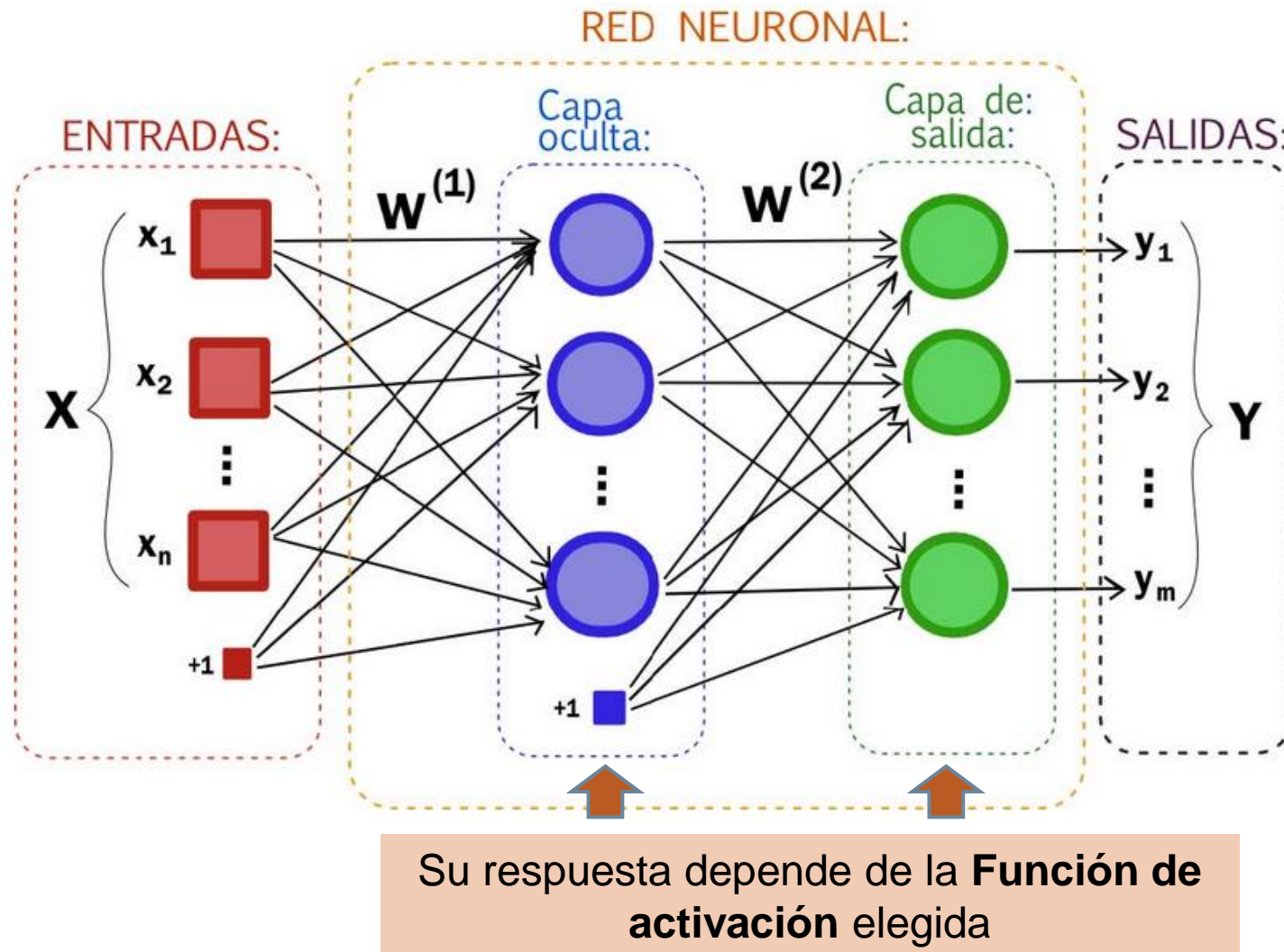


# Arquitectura de la red

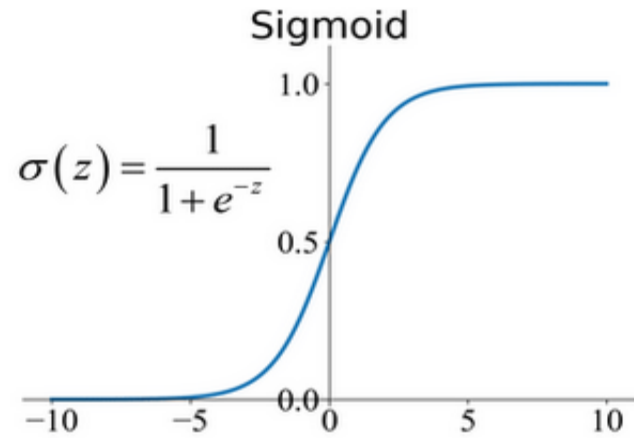


Las dimensiones de las capas de entrada y salida las define el problema

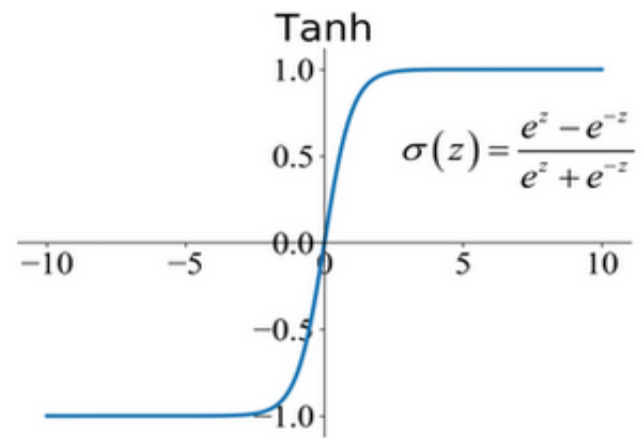
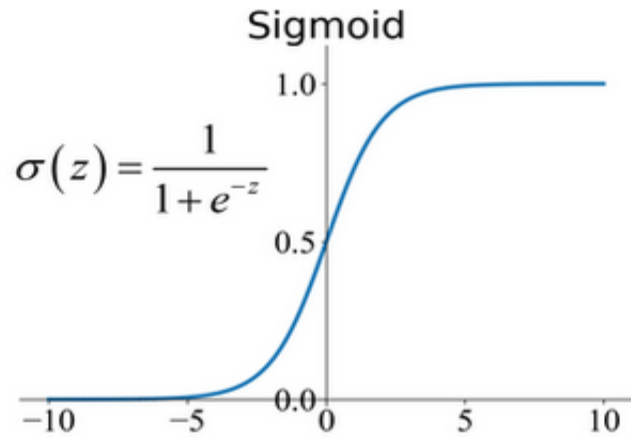
# Arquitectura de la red



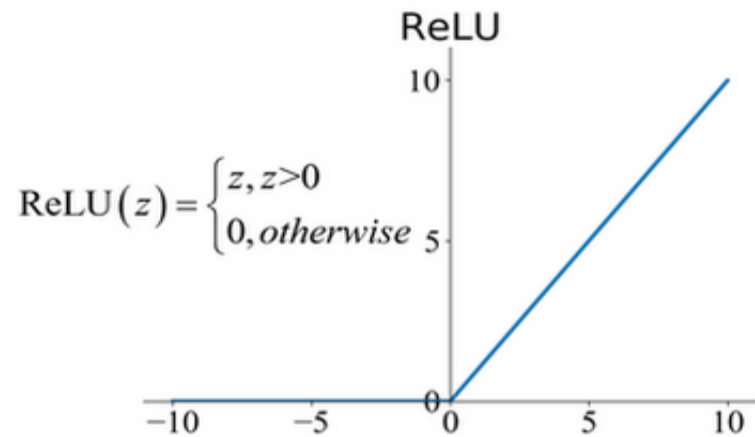
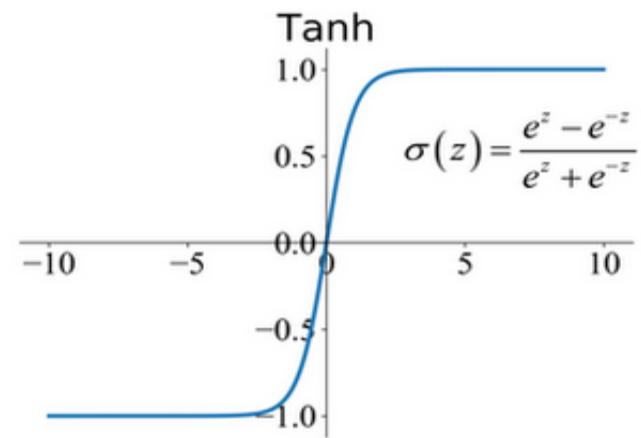
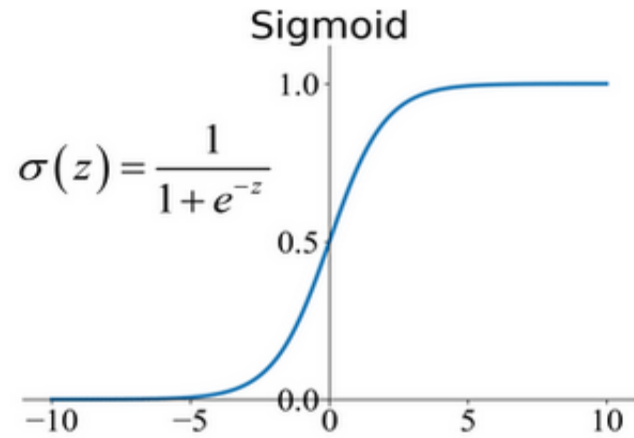
# Funciones de activación



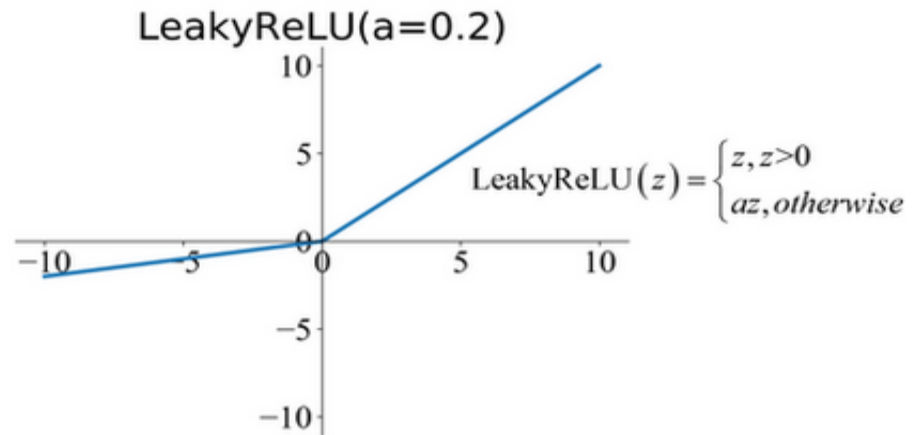
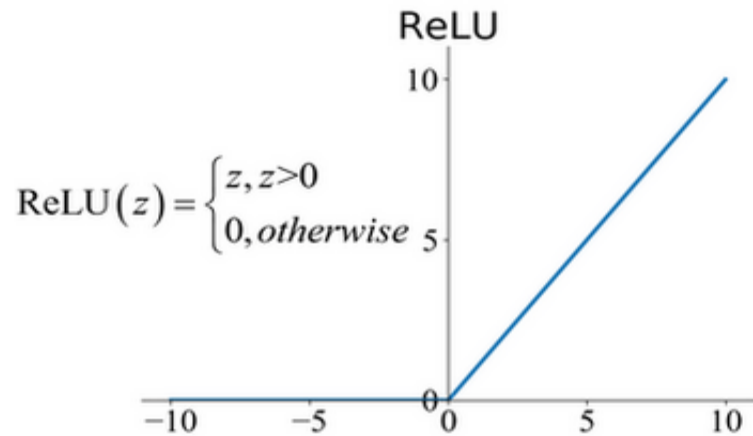
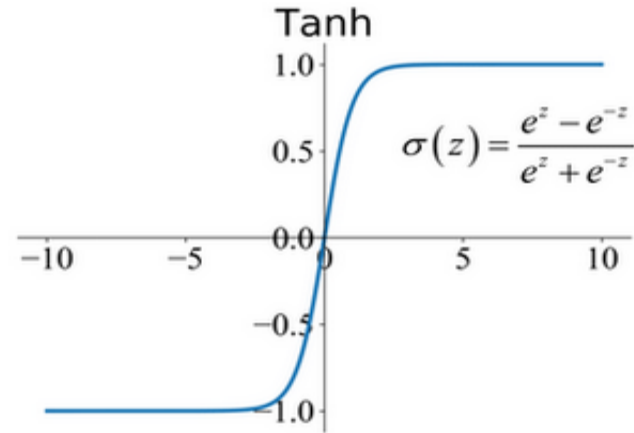
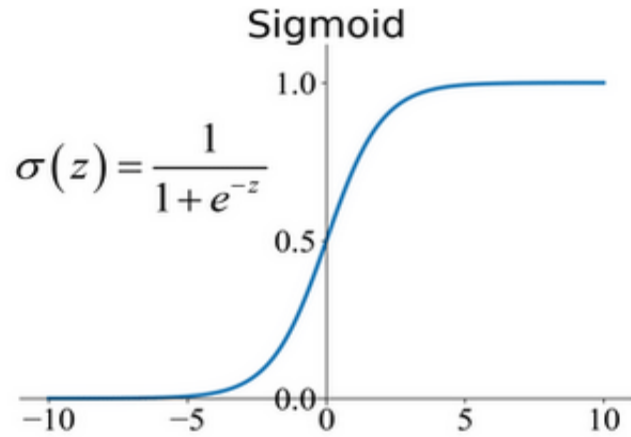
# Funciones de activación



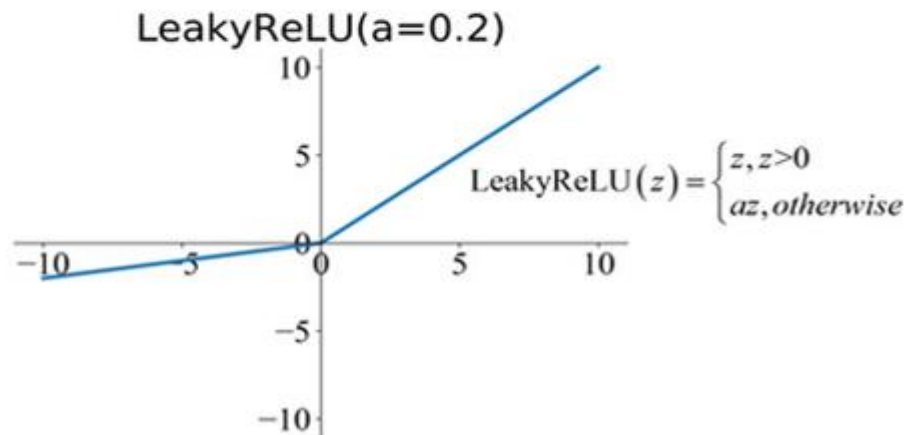
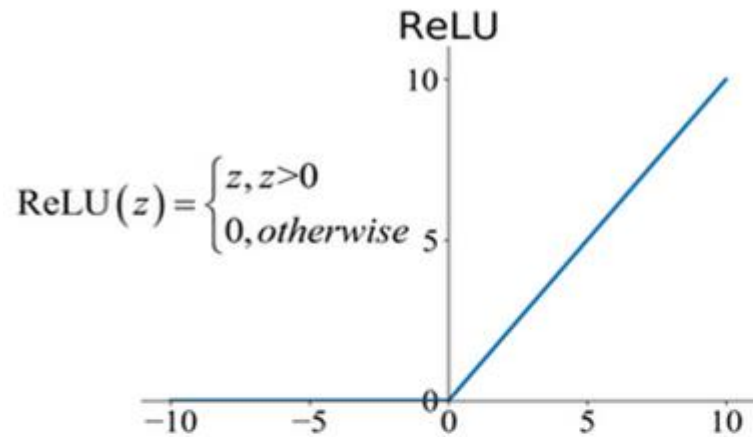
# Funciones de activación



# Funciones de activación



# ReLU (Unidad Lineal Rectificada)

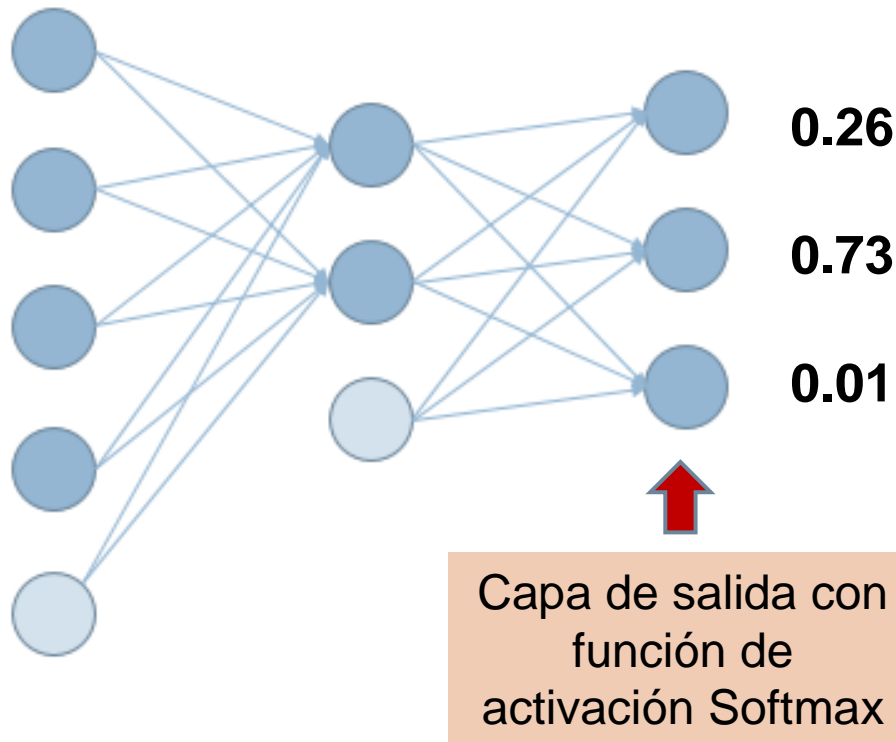


- Velocidad de aprendizaje (derivada)
- Velocidad de cómputo (fácil de calcular)
- Activa sólo algunas neuronas



# Función Softmax

- Se utiliza como función de activación en la última capa para normalizar la salida de la red de manera que los valores sumen 1.



$$neta_j = \sum_i w_{ji} x_i + b_j$$

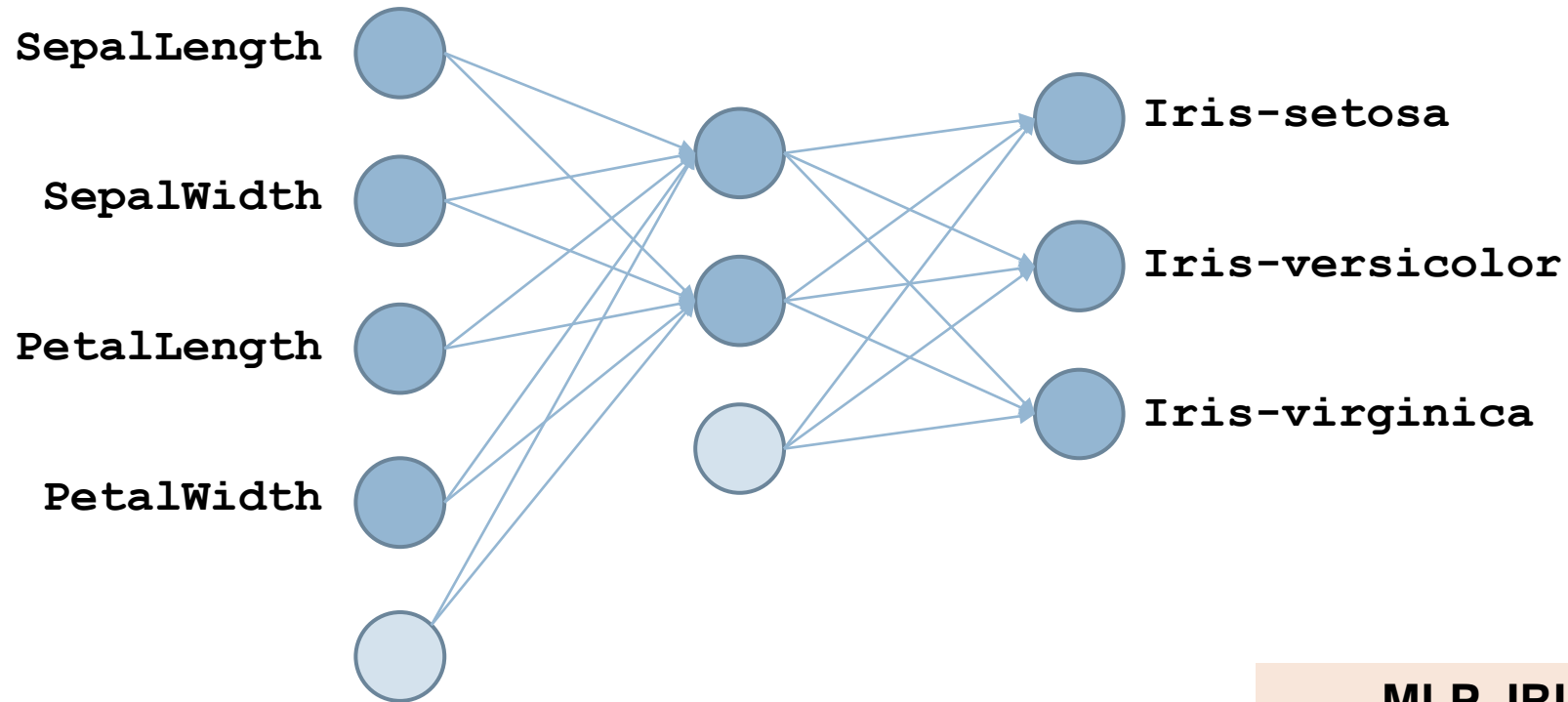
$$\hat{y}_j = \frac{e^{neta_j}}{\sum_k e^{neta_k}}$$

# 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

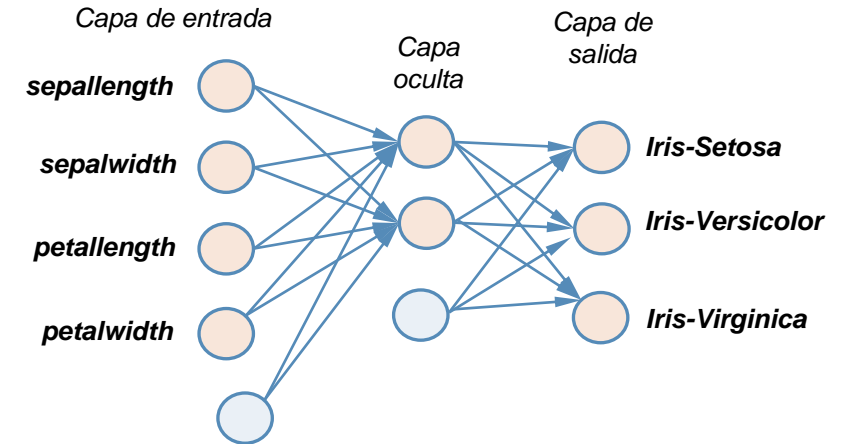
# Clasificación de flores de Iris

**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]]
```



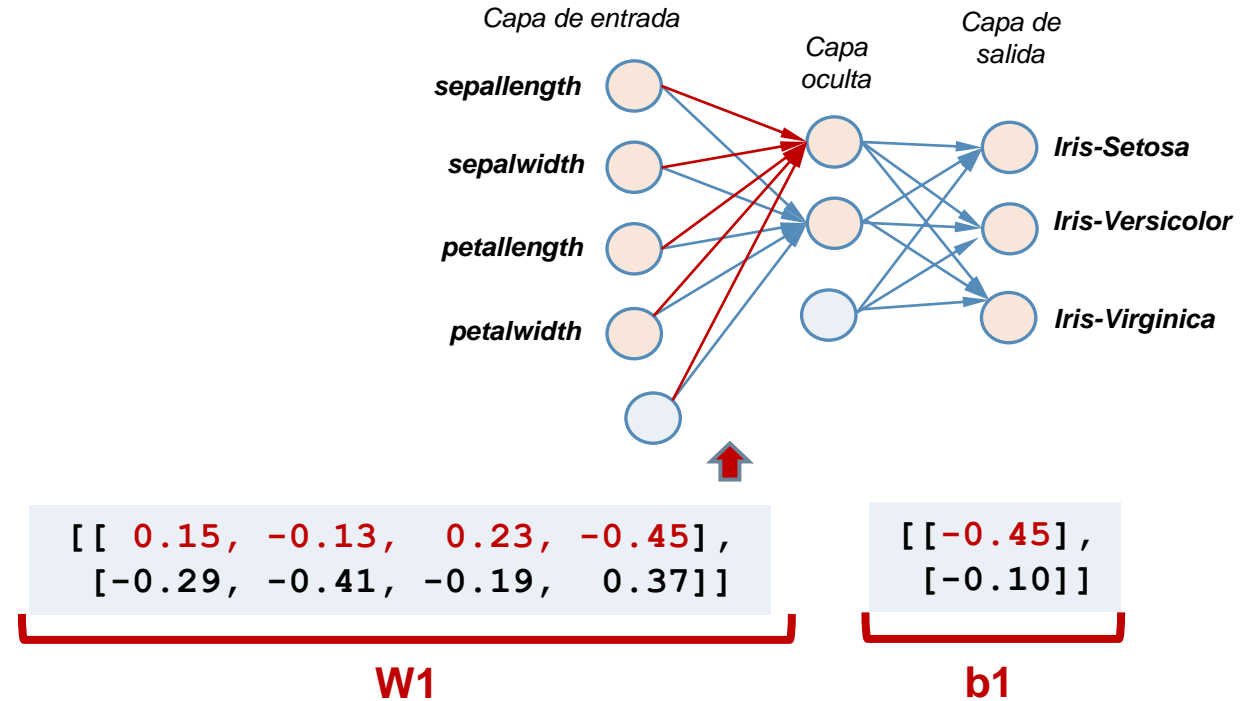
# Clasificación de flores de Iris

**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]]
```



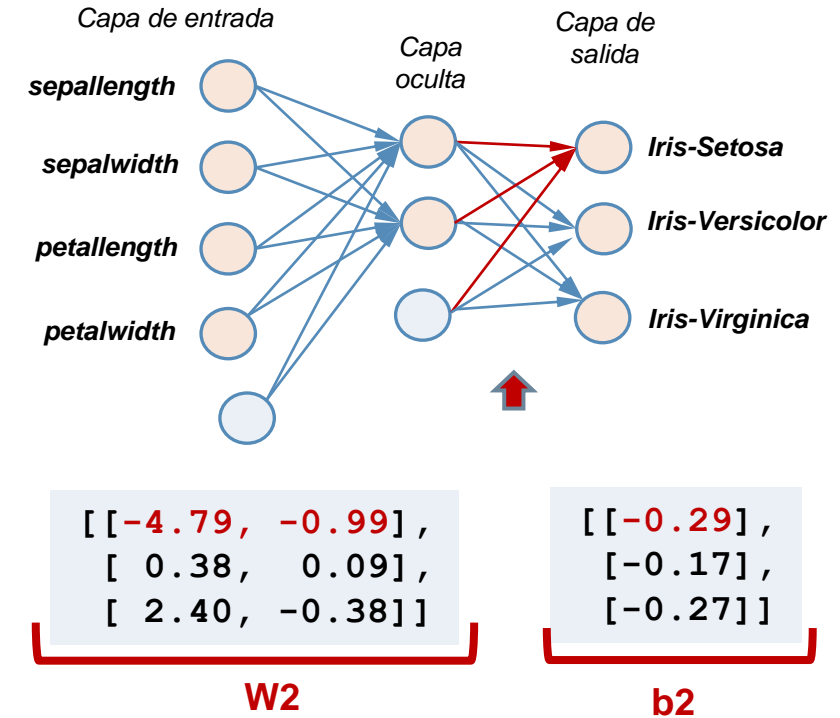
# Clasificación de flores de Iris

**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

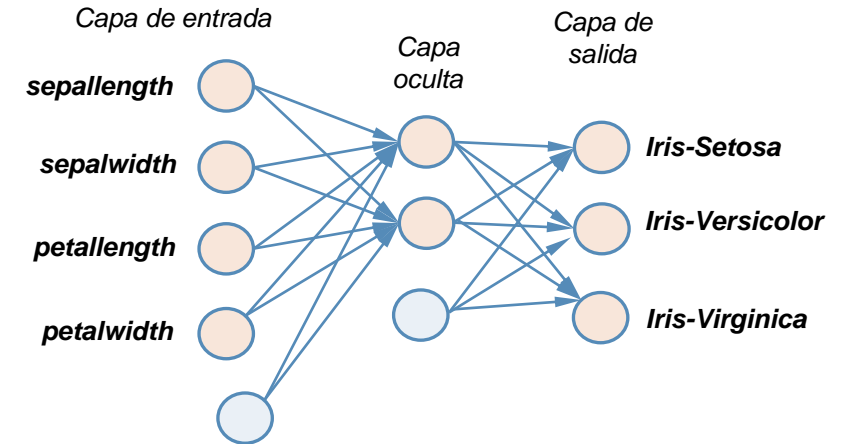
**X**

```
[[-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]]
```

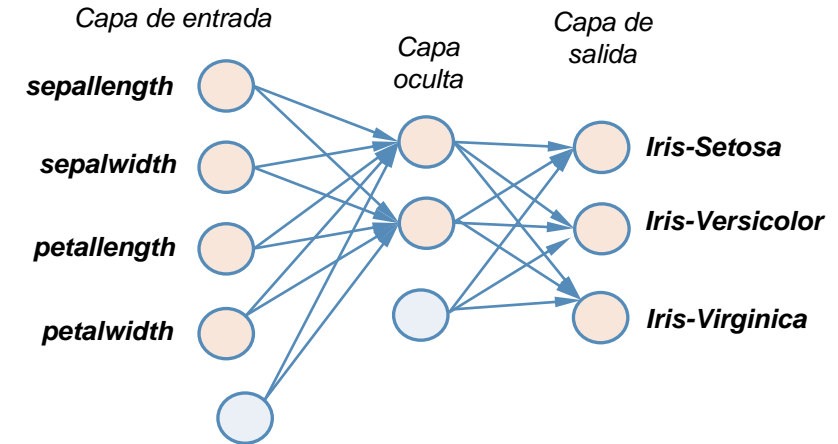
FunH='relu' ; FunO='sigmoid'



Ingresar el primer ejemplo a la red y  
calcular su salida

# Calculando la salida de la capa oculta

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



□ Salida de la capa oculta

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

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

FunH='relu' ; FunO='sigmoid'

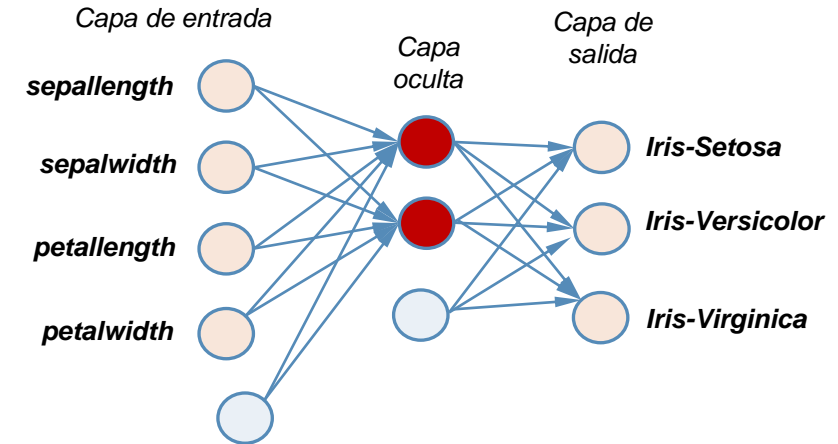
$$\text{netas}_{pj}^h = \sum_{i=1}^n w_{ji}^h x_{pi} + \theta_j^h$$

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



# Calculando la salida de la capa oculta

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



□ Salida de la capa oculta

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

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

FunH='relu' ; FunO='sigmoid'

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

$$\text{salidasH} = \text{netasH} * (\text{netasH} > 0) = \left[ \begin{array}{c} [0] \\ [0.1997] \end{array} \right]$$

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

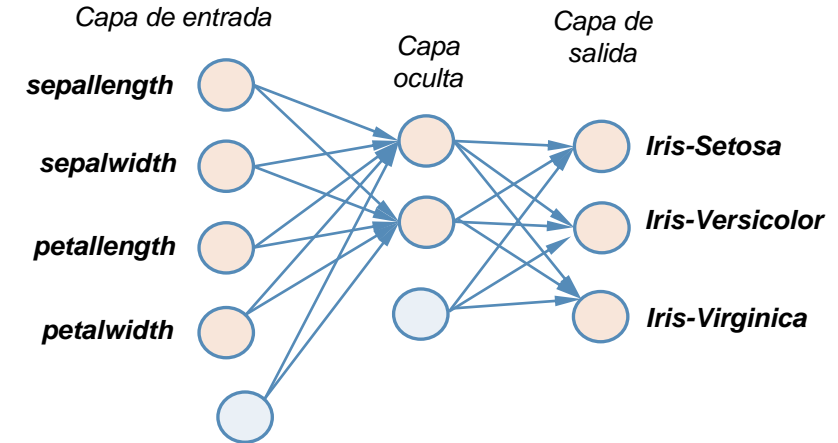
# Calculando la salida de la red (capa de salida)

**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]]
```

**Y**

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



□ Salida de red

`netasO = W2 * salidasH + b2`

**FunH='relu' ; 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   0.1997]]</pre>	+	<p><b>b2</b></p> <pre>[[ -0.29],  [ -0.17],  [ -0.27]]</pre>	=	<p><b>netasO</b></p> <pre>[[ -0.487703],  [ -0.152027],  [ -0.345886]]</pre>
---	---	---	---	--	---	--

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

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

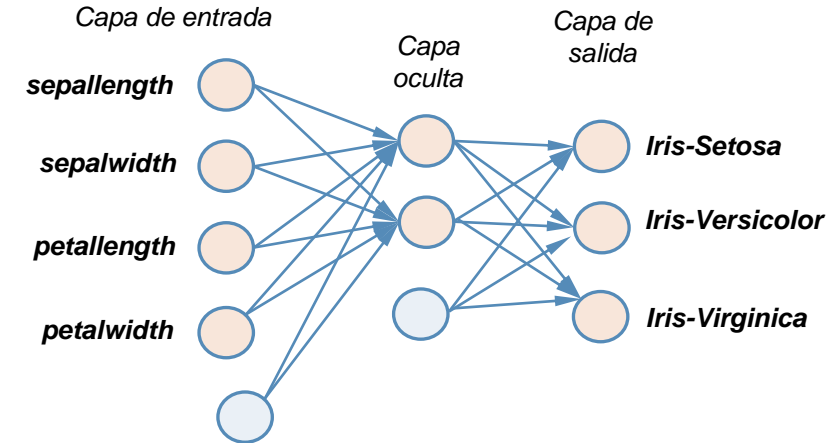
# Calculando la salida de la red (capa de salida)

**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='relu' ; 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   0.1997]]</pre>	<p>+</p>	<pre>[[ -0.29],  [ -0.17],  [ -0.27]]</pre> <p><b>b2</b></p>	<p>=</p>	<p><b>netasO</b></p> <pre>[[ -0.487703]  [ -0.152027]  [ -0.345886]]</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)) =
```

```
[[0.38043483]
 [0.46206628]
 [0.41438041]]
```



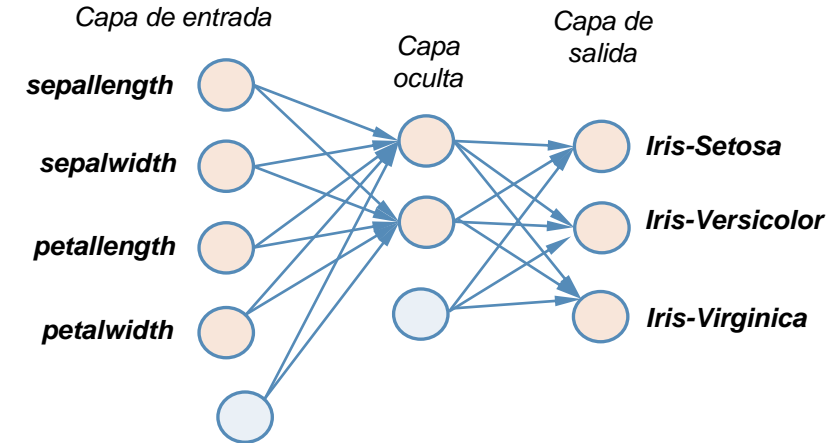
# Error de la capa de salida

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>[[0.38043483]</code>		<code>[[ 0.61956517]</code>
	<code>[0.0]</code>	<code>-</code>	<code>[0.46206628]</code>	<code>=</code>	<code>[-0.46206628]</code>
	<code>[0.0]]</code>		<code>[0.41438041]]</code>		<code>[-0.41438041]]</code>
	↑		<code>salidasO</code>		



`FunH='relu' ; 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**

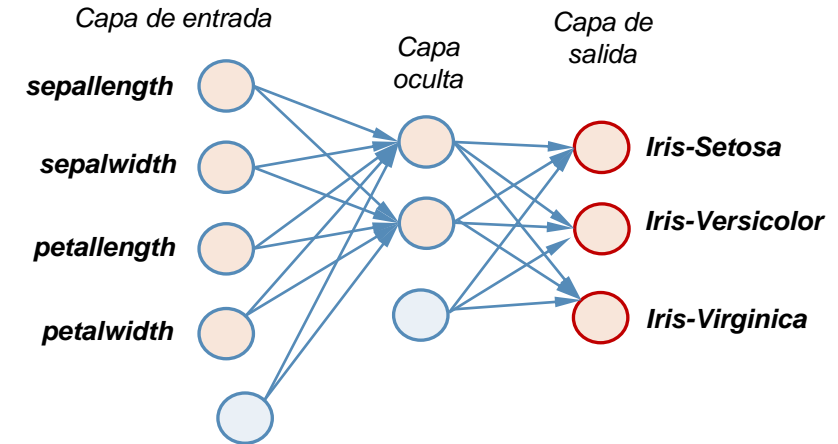
```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```

## Factores para corregir W2 y b2

`salidasO*(1-salidasO)`

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

```
deltaO = [[ 0.61956517]
           [-0.46206628]
           [-0.41438041]] .* [[0.23570417]
                               [0.24856103]
                               [0.24266929]] = [[ 0.14603409]
                                                  [-0.11485167]
                                                  [-0.1005574 ]]
```



**FunH='relu' ; FunO='sigmoid'**

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

# 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

`(salidasH > 0)*1`

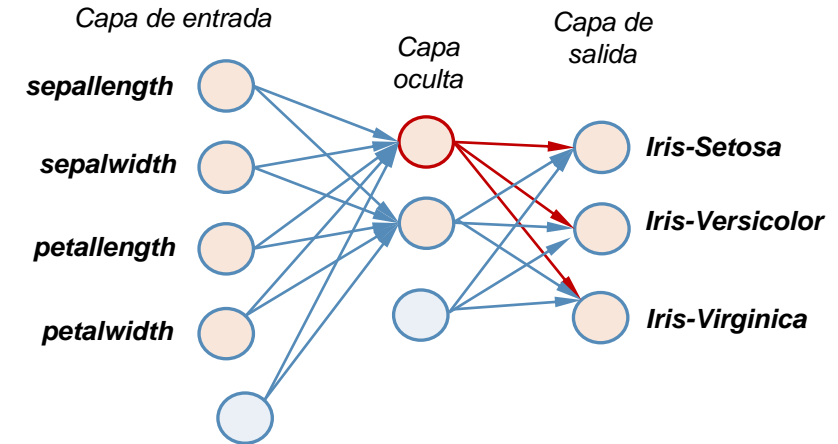
`deltaH = deriv_FunH .* (W2.T @ deltaO)`

`deltaH =`  $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$  `.*`

$\begin{bmatrix} -4.79 & 0.38 & 2.4 \\ -0.99 & 0.09 & -0.38 \end{bmatrix}$  `@`

$\begin{bmatrix} 0.14603409 \\ -0.11485167 \\ -0.1005574 \end{bmatrix}$

`deltaH =`  $\begin{bmatrix} 0 \\ -0.11669859 \end{bmatrix}$



`FunH='relu' ; FunO='sigmoid'`

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

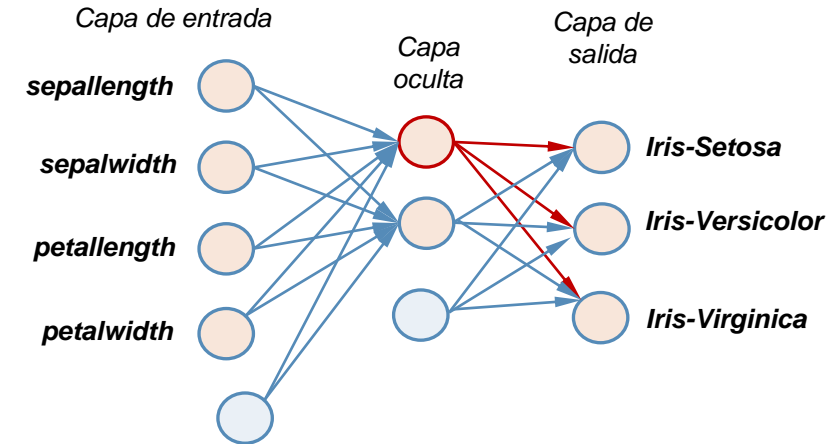
# Corrigiendo de los pesos

**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='relu' ; FunO='sigmoid'**

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

$$W2 = \begin{bmatrix} [-4.79, -0.99], \\ [0.38, 0.09], \\ [2.40, -0.38] \end{bmatrix} + \text{alfa} * \begin{bmatrix} [0.14603409] \\ [-0.11485167] \\ [-0.1005574] \end{bmatrix} @ \begin{bmatrix} [0] \\ [0.1997] \end{bmatrix} = \begin{bmatrix} [-4.79, -0.99] \\ [0.38, 0.09] \\ [2.4, -0.38] \end{bmatrix}$$

$$b2 = b2 + \text{alfa} * \text{deltaO} = \begin{bmatrix} [-0.29], \\ [-0.17], \\ [-0.27] \end{bmatrix} + \text{alfa} * \begin{bmatrix} [0.14603409] \\ [-0.11485167] \\ [-0.1005574] \end{bmatrix} = \begin{bmatrix} [-0.28], \\ [-0.18], \\ [-0.28] \end{bmatrix}$$

# Corrigiendo de los pesos

MLP\_IRIS\_algBPN\_RELU.ipynb

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>

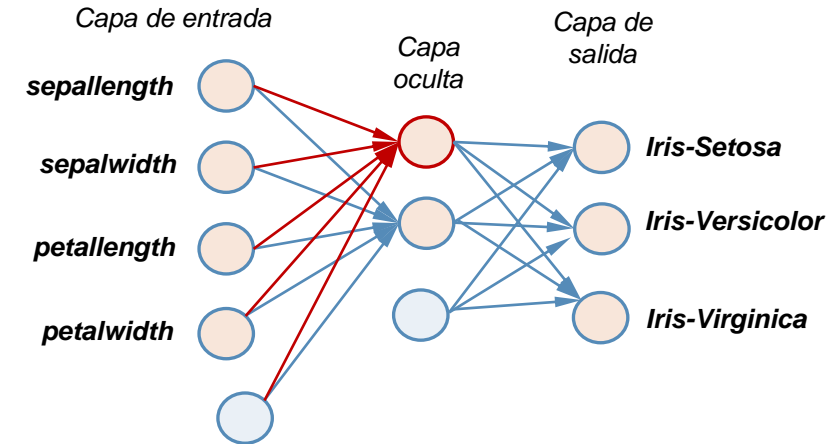
## □ Modificación de W1 y b1

`W1 = W1 + alfa * deltaH @ x`

`W1 = [[ 0.15,-0.13,0.23,-0.45],`  
`[-0.29,-0.41,-0.19,0.37]] + alfa *  $\begin{bmatrix} [ [ 0. & ] \\ [-0.11669859] ] \end{bmatrix} @ [-1.73,-0.05,-1.38,-1.31]$`

`W1 = [[ 0.15 -0.13 0.23 -0.45]`  
`[-0.27 -0.41 -0.17 0.39]]`

`b1 = b1 + alfa * deltaH =  $\begin{bmatrix} [-0.45] \\ [-0.10] \end{bmatrix} + alfa * \begin{bmatrix} [ 0. & ] \\ [-0.11669859] \end{bmatrix} = \begin{bmatrix} [-0.45] \\ [-0.11] \end{bmatrix}$`



`FunH='relu' ; FunO='sigmoid'`



# 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(-netasH))-1
netasO = W2 @ salidasH + b2
salidasO = 1.0/(1+np.exp(-netasO))
print("salidaO = \n", salidasO)
ErrorSalidaNew = yi.T-salidasO
print("ErrorSalida = \n", ErrorSalidaNew)
```

```
salidaO =
[[0.66514149]
 [0.436038 ]
 [0.30779953]]
ErrorSalida =
[[ 0.33485851]
 [-0.436038 ]
 [-0.30779953]]
```

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

```
Error inicial = 0.7690773719494183
Error luego de la correccion = 0.39699991207045215
```

Antes de modificar los pesos de la red

```
salidaO =
[[0.38043483]
 [0.46206628]
 [0.41438041]]
ErrorSalida =
[[ 0.61956517]
 [-0.46206628]
 [-0.41438041]]
```

Ver  
**MLP\_IRIS.ipynb**

# Keras

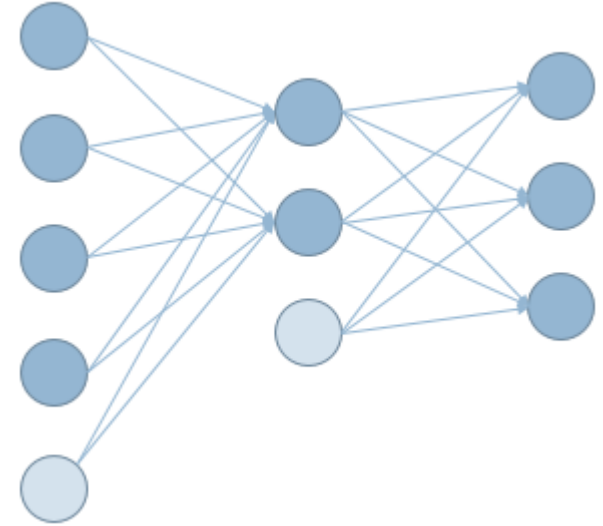
- Keras es una biblioteca de código abierto escrita en Python que facilita la creación de modelos complejos de aprendizaje profundo.
- Características
  - ▣ Prototipado rápido del modelo.
  - ▣ De alto nivel (programación a nivel de capa)
  - ▣ Usa las librerías de los frameworks vinculados
    - TensorFlow
    - Theano
    - Microsoft Cognitive Toolkit (CNTK)



# Construcción del modelo

```
from keras.models import Sequential  
from keras.layers import Dense
```

```
# Crear un modelo de capas secuenciales  
model=Sequential()
```



# Construcción del modelo

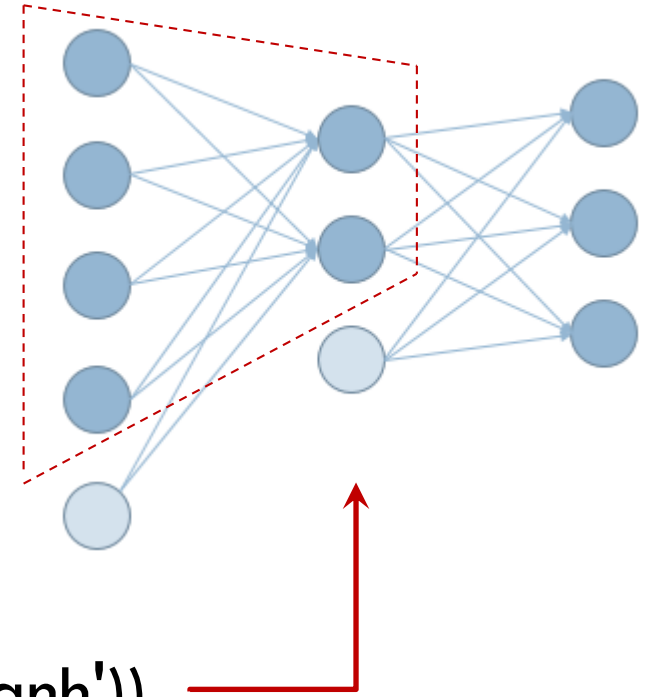
```
from keras.models import Sequential  
from keras.layers import Dense
```

**# Crear un modelo de capas secuenciales**

```
model=Sequential()
```

**# Agregar las capas al modelo**

```
model.add(Dense(2, input_shape=[4], activation='tanh'))
```



# Construcción del modelo

```
from keras.models import Sequential  
from keras.layers import Dense
```

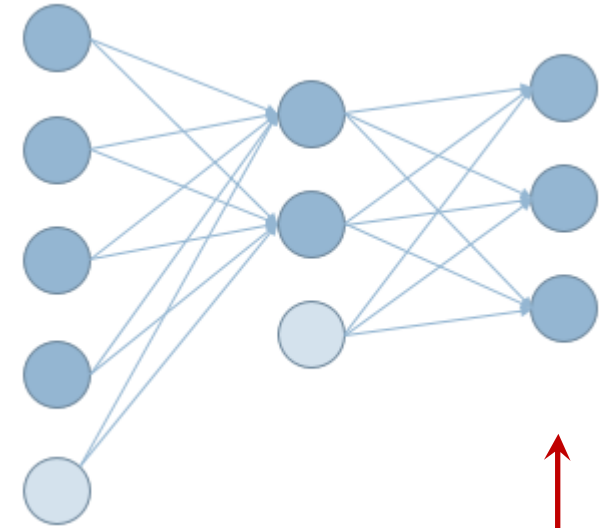
**# Crear un modelo de capas secuenciales**

```
model=Sequential()
```

**# Agregar las capas al modelo**

```
model.add(Dense(2, input_shape=[4], activation='tanh'))
```

```
model.add(Dense(3, activation='sigmoid'))
```



# Construcción del modelo

```
from keras.models import Sequential
from keras.layers import Dense
```

**# Crear un modelo de capas secuenciales**

```
model=Sequential()
```

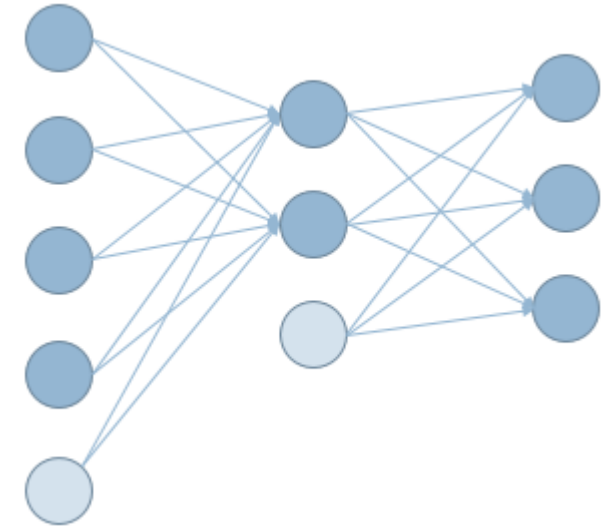
**# Agregar las capas al modelo**

```
model.add(Dense(2, input_shape=[4], activation='tanh'))
```

```
model.add(Dense(3, activation='sigmoid'))
```

**# Imprimir un resumen del modelo**

```
model.summary()
```



Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2)	10
dense_2 (Dense)	(None, 3)	9
Total params: 19		
Trainable params: 19		
Non-trainable params: 0		

# Indicando la función de activación por separado

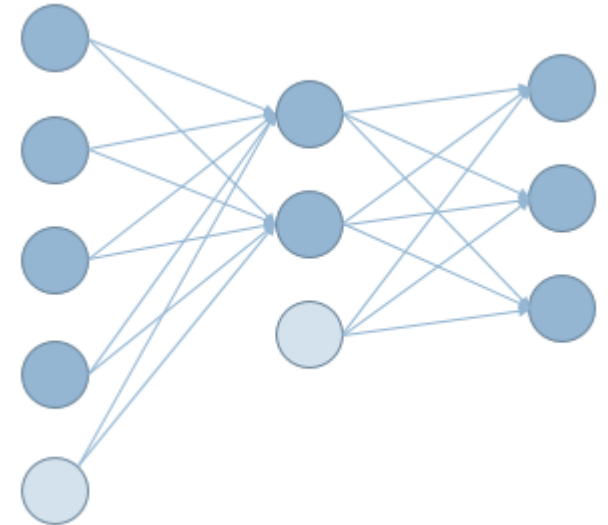
```
from keras.models import Sequential
from keras.layers import Dense, Activation

model=Sequential()

model.add(Dense(2, input_dim=4))
model.add(Activation('tanh'))

model.add(Dense(3))
model.add(Activation('sigmoid'))

model.summary()
```



Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 2)	10
activation (Activation)	(None, 2)	0
dense_1 (Dense)	(None, 3)	9
activation_1 (Activation)	(None, 3)	0

=====  
Total params: 19

Trainable params: 19

Non-trainable params: 0

# Usando una lista

```
from keras.models import Sequential
from keras.layers import Dense
```

```
model=Sequential([
    Dense(2, input_shape=[4], activation='tanh', name='Oculto'),
    Dense(3, activation='sigmoid', name='salida')])
```

```
model.summary()
```

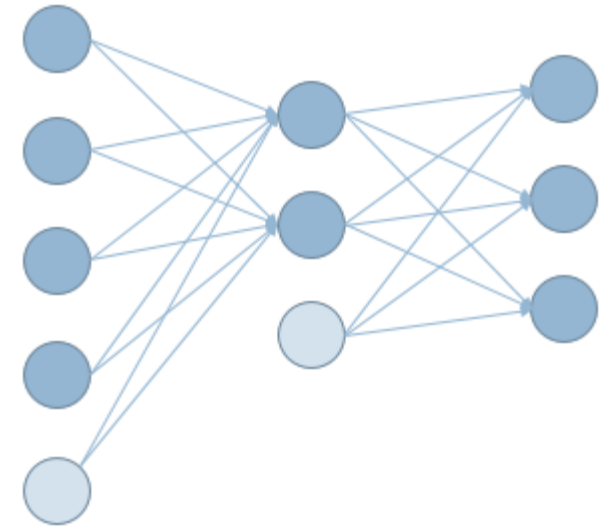
Model: "sequential"

Layer (type)	Output Shape	Param #
Oculto (Dense)	(None, 2)	10
salida (Dense)	(None, 3)	9

Total params: 19

Trainable params: 19

Non-trainable params: 0



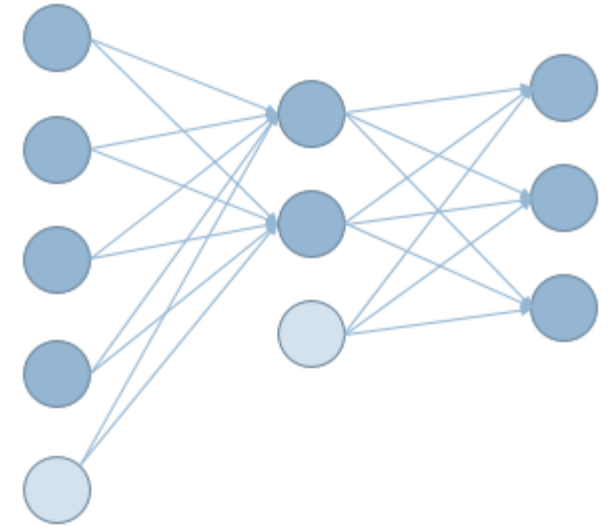


# Usando una lista

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model=Sequential([
    Dense(2, input_dim=4, name='Oculto'),
    Activation('tanh', name='FunH'),
    Dense(3, name='salida'),
    Activation('sigmoid', name='FunO')])

model.summary()
```



Layer (type)	Output Shape	Param #
Oculto (Dense)	(None, 2)	10
FunH (Activation)	(None, 2)	0
salida (Dense)	(None, 3)	9
FunO (Activation)	(None, 3)	0
Total params: 19		

# Funcional

```
from keras.models import Model
from keras.layers import Dense, Input
```

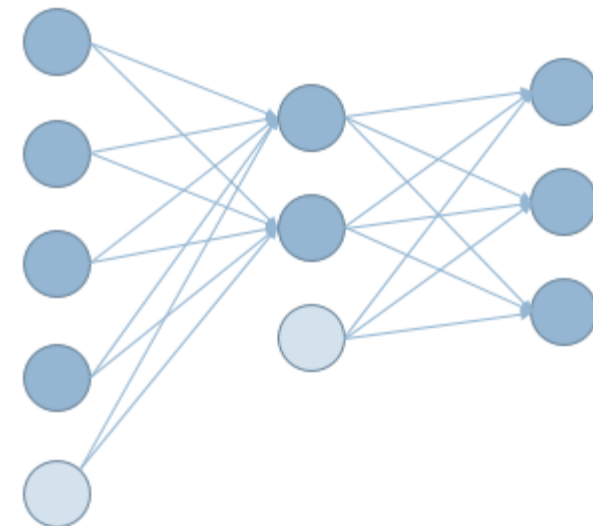
```
I = Input(shape=(4,))
```

```
L = Dense(units=2, activation='tanh', name='Oculto')(I)
```

```
salida=Dense(units=3, activation='sigmoid', name='salida')(L)
```

```
model = Model(inputs=I, outputs = salida)
```

```
model.summary()
```



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 4)]	0
Oculto (Dense)	(None, 2)	10
salida (Dense)	(None, 3)	9
Total params: 19		

# Funcional

```
from keras.models import Model
from keras.layers import Dense, Input, Activation

I = Input(shape=(4,), name='entrada')

L = Dense(units=2, name='Oculto')(I)

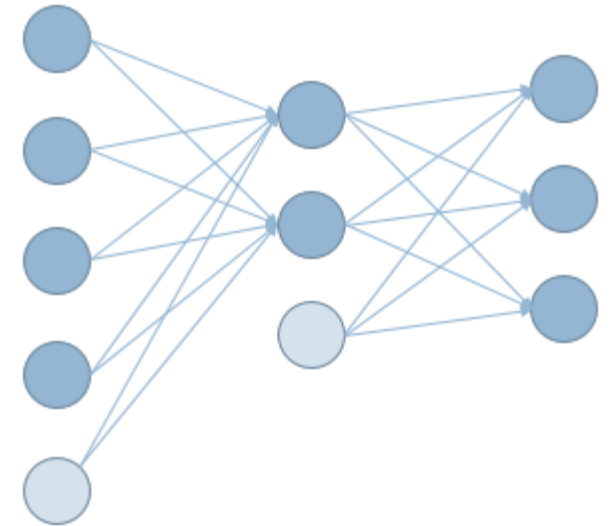
L = Activation('tanh', name='FunH')(L)

L = Dense(units=3, name='salida')(L)

salida=Activation('sigmoid', name='FunO')(L)

model = Model(inputs=I, outputs = salida)

model.summary()
```



Layer (type)	Output Shape	Param #
entrada (InputLayer)	[(None, 4)]	0
Oculto (Dense)	(None, 2)	10
FunH (Activation)	(None, 2)	0
salida (Dense)	(None, 3)	9
FunO (Activation)	(None, 3)	0
Total params: 19		

# Definiendo capas

```
from keras.models import Model
from keras.layers import Dense, Input

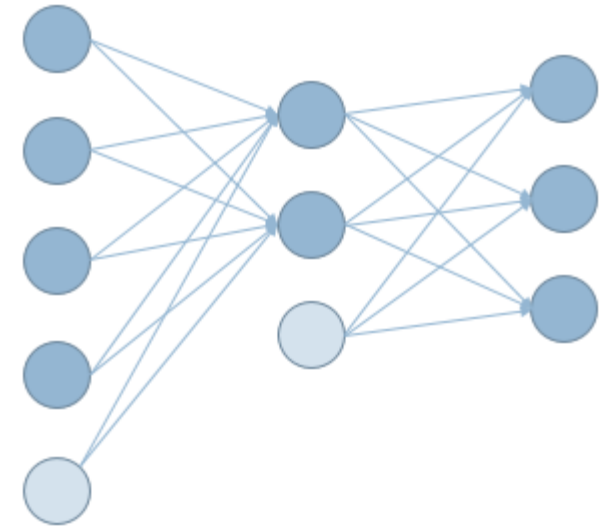
I = Input(shape=(4,), name='entrada')

oculta = Dense(units=2, activation='tanh', name='Oculto')
salida = Dense(units=3, activation='sigmoid', name='salida')

red = salida(oculta(I))

model = Model(inputs=I, outputs = red)

model.summary()
```



Layer (type)	Output Shape	Param #
entrada (InputLayer)	[(None, 4)]	0
Oculto (Dense)	(None, 2)	10
salida (Dense)	(None, 3)	9
Total params: 19		

# Resumen

## Resolución de una tarea de clasificación

- Conjunto de datos etiquetados (aprendizaje supervisado)
- Definición de la arquitectura de la red
  - ▣ Número de capas y tamaño de cada una
  - ▣ Función de activación a usar en cada capa
- Entrenamiento
  - ▣ Función de error
  - ▣ Técnica de optimización para reducir el error
- Evaluar el modelo



# Configuración para entrenamiento

```
from keras.models import Sequential
from keras.layers import Dense
```

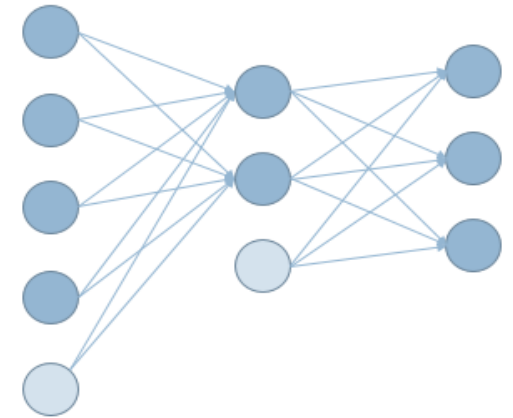
```
model=Sequential()
model.add(Dense(2, input_shape=[4], activation='tanh'))
model.add(Dense(3, activation='sigmoid'))
```

## # Configuración para entrenamiento

```
model.compile(optimizer='sgd', loss='mse', metrics='accuracy')
```

*Descenso de gradiente  
estocástico*

*Error Cuadrático  
Medio*



*Keras\_IRIS.ipynb*

# Configuración para entrenamiento

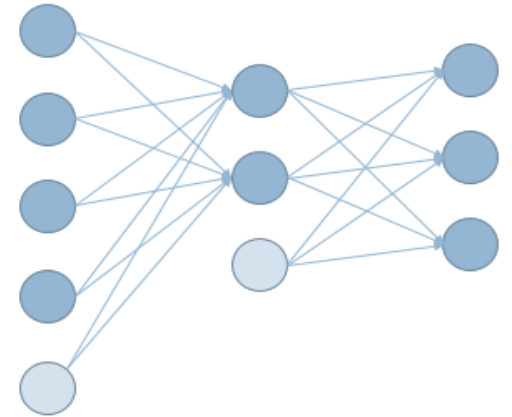
```
from keras.optimizers import SGD ←  
from keras.models import Sequential  
from keras.layers import Dense
```

```
model=Sequential()  
model.add(Dense(2, input_shape=[4], activation='tanh'))  
model.add(Dense(3, activation='sigmoid'))
```

## # Configuración para entrenamiento

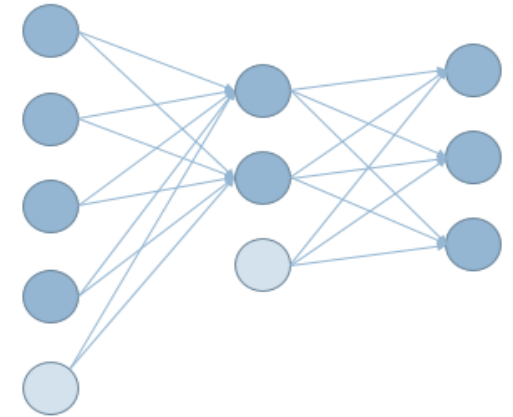
```
model.compile(optimizer=SGD(lr=0.1), loss='mse', metrics='accuracy')
```

*Tasa de aprendizaje  
(learning rate)*



*Keras\_IRIS\_SGD.ipynb*

# Configuración para entrenamiento



```
from keras.models import Sequential
from keras.layers import Dense
```

```
model=Sequential()
model.add(Dense(2, input_shape=[4], activation='tanh'))
model.add(Dense(3, activation='softmax'))
```

## # Configuración para entrenamiento

```
model.compile(loss='categorical_crossentropy', optimizer='sgd',
              metrics=['accuracy'])
```

*Keras\_Iris\_Softmax.ipynb*



# Carga de datos

`X, T = cargar_datos()`

`Y = keras.utils.to_categorical(T)`

*T debe ser un vector numérico. Puede usar lo siguiente para convertirlo de ser necesario:*

```
from sklearn import preprocessing
encoder = preprocessing.LabelEncoder()
T = encoder.fit_transform(T)
```

*X → Conjunto de ejemplos de entrada*

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4

*Y → Rtas esperadas para cada neurona de la capa de salida*

	0	1	2
0	1	0	0
1	0	1	0
2	0	0	1
3	0	0	1
4	0	1	0
5	0	0	1

***X e Y son matrices de numpy***

# Entrenamiento del modelo

```
X,Y = cargar_datos()  # X e Y son matrices de numpy  
# Entrenar el modelo  
model.fit(X,Y, epochs=100, batch_size=20)
```

# Predicción del modelo

`X,Y = cargar_datos()` *# X e Y son matrices de numpy*


*# Entrenar el modelo*

`model.fit(X,Y, epochs=100, batch_size=20)`

*# predecir la salida del modelo*

`Y_pred = model.predict(X)`

*Y\_pred tiene las mismas  
dimensiones que Y*



	0	1	2
0	0.967722	0.189344	0.00421873
1	0.0372113	0.510963	0.346058
2	0.00325751	0.261545	0.917956
3	0.00823823	0.319694	0.795647
4	0.0717264	0.611822	0.171516
5	0.0134856	0.482814	0.59486

# Error del modelo

```
X,Y = cargar_datos() # X e Y son matrices de numpy
```

```
# Entrenar el modelo
```

```
model.fit(X,Y, epochs=100, batch_size=20)
```

```
# predecir la salida del modelo
```

```
Y_pred = model.predict(X)
```

```
# Calcular el error del modelo
```

```
score = model.evaluate(X_train, Y_trainB)
```

```
print('Error :', score[0])
```

```
print('Accuracy:', score[1])
```

]

*Muestra el valor de la función de Costo y la precisión del modelo al finalizar el entrenamiento*

# Métricas

## **# Entrenar el modelo**

```
model.fit( X, Y, epochs=100, batch_size=20)
```

## **# Predicciones del modelo**

```
Y_pred = model.predict(X)
```

```
Y_pred_nro = np.argmax(Y_pred, axis=1) # conversión a entero
```

```
Y_true = np.argmax(Y, axis=1)
```

```
print("%%% aciertos %.3f" % metrics.accuracy_score(Y_true, Y_pred_nro))
```

ver  
***Keras\_IRIS.ipynb***

# Pesos de la red

```
model.fit(...)
```

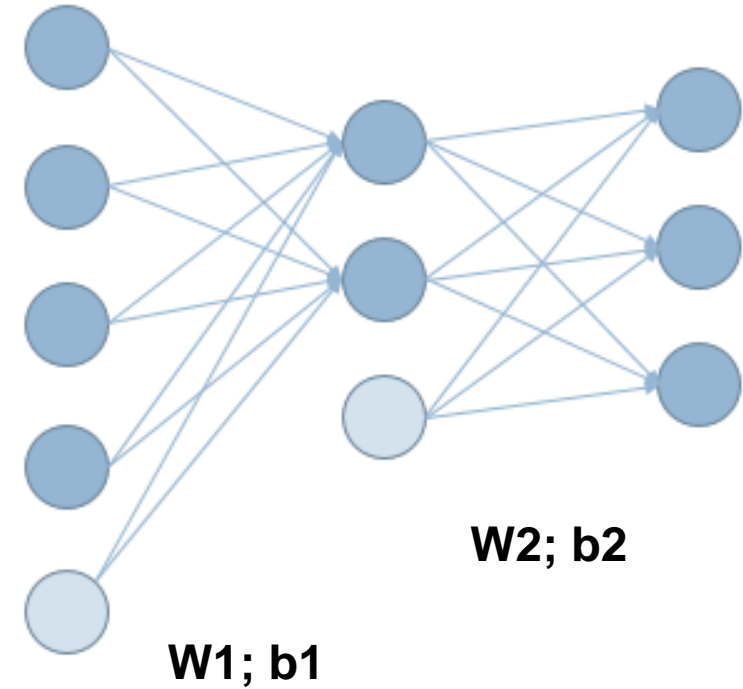
```
...
```

```
capaOculta = model.layers[0]
```

```
W1, b1 = capaOculta.get_weights()
```

```
capaSalida = model.layers[1]
```

```
W2, b2 = capaSalida.get_weights()
```



# Salvar el modelo

- Una vez entrenado el modelo, si los resultados han sido buenos lo guardamos para su uso posterior

## OPCION 1

Guardamos todo el modelo

```
model = ...  
model.fit( ... )  
...  
model.save("miModelo.h5")
```

## OPCION 2

Guardamos sólo los pesos

```
model = ...  
model.fit( ... )  
...  
model.save_weights("pesos_de_miModelo.h5")
```

*Requiere definir el modelo antes  
de cargar*

<https://filext.com/es/online-file-viewer.html>

# Cargar el modelo

## OPCION 1 – Carga el modelo completo


```
from keras.models import load_model  
model = load_model("miModelo.h5")
```

## OPCION 2 – Cargar sólo los pesos

```
model = ... (definir el modelo)  
...  
model.load_weights("pesos_de_miModelo.h5")
```



# Técnicas de optimización

- Descenso de gradiente estocástico (SGD) y el uso de mini-lotes 
- Capacidad de generalización de la red - Sobreajuste
- Mejoras introducidas
  - ▣ Momento: utiliza información de los gradientes anteriores
  - ▣ RMSProp: considera distintas magnitudes de cambio para reducir oscilaciones
  - ▣ Adam: combina los dos anteriores. Es el más usado.

# Descenso de gradiente en mini-lotes

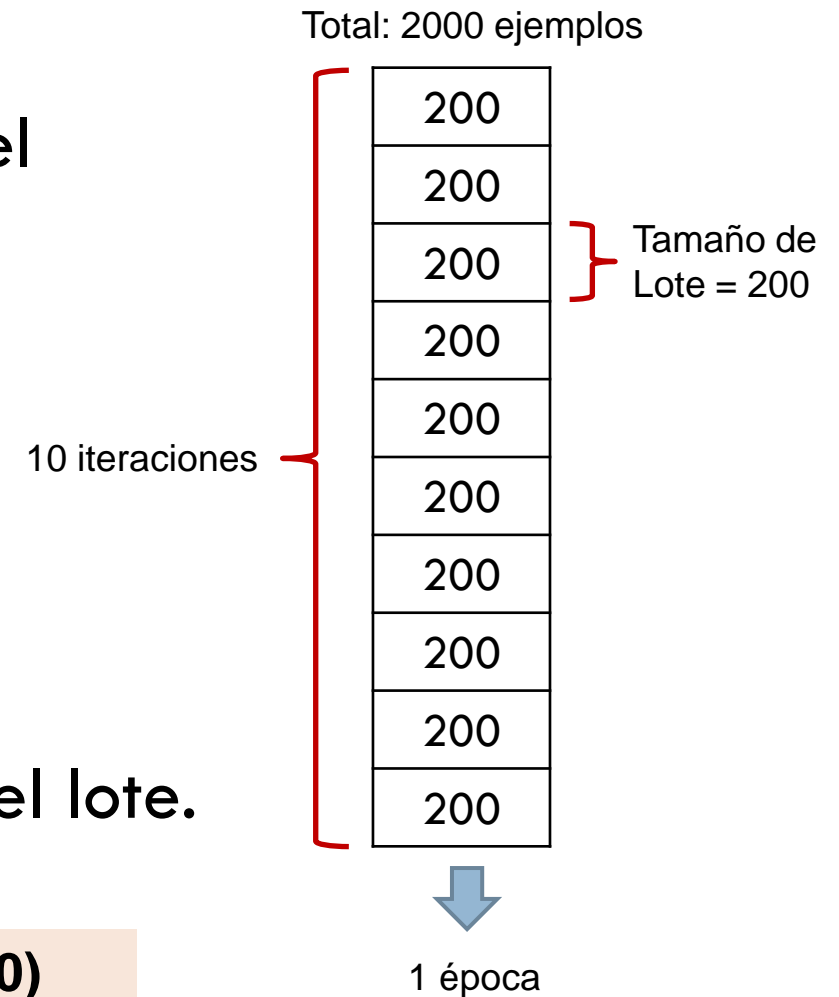
- En lugar de ingresar los ejemplos de a uno, ingresamos  $N$  a la red y buscamos minimizar el error cuadrático promedio del lote.

- La función de costo será

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

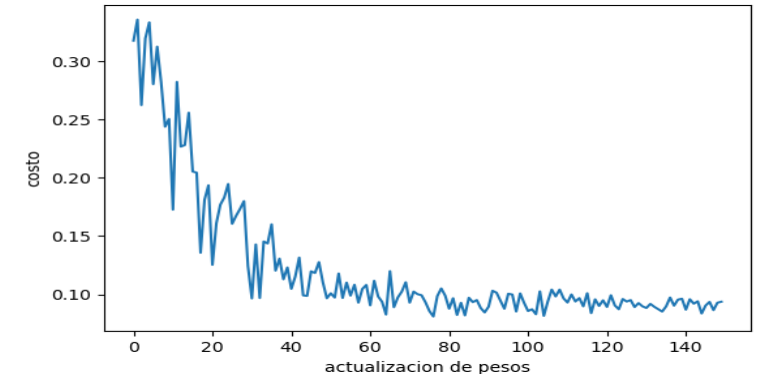
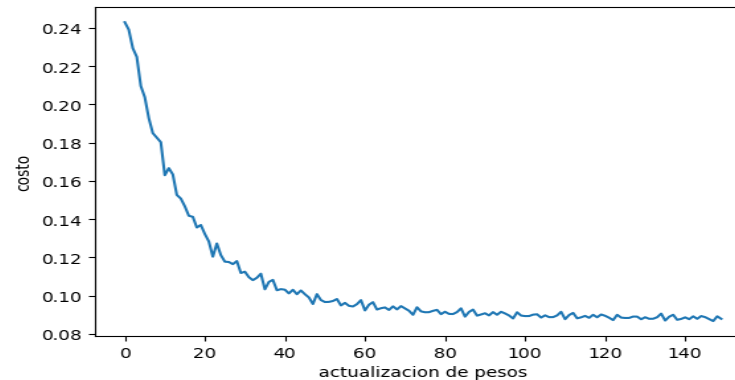
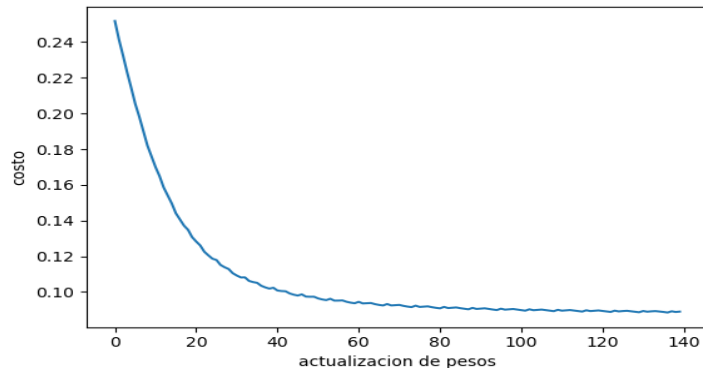
- $N$  es la cantidad de ejemplos que conforman el lote.

`model.fit(X, Y, epochs=2000, batch_size=200)`




# 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
$C = \frac{1}{M} \sum_{i=1}^M (d_i - f(neta_i))^2$	$C = \frac{1}{N} \sum_{i=1}^N (d_i - f(neta_i))^2 \quad N \ll M$	$C = (d - f(neta))^2$

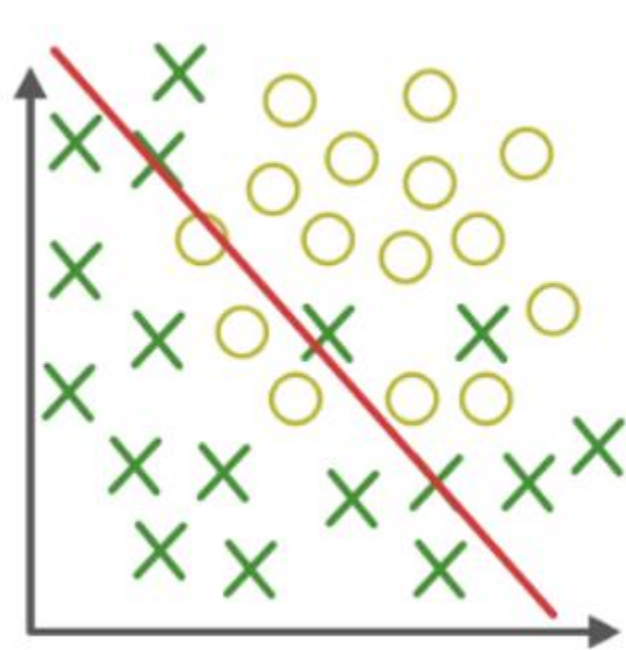


Ver [MLP\\_MNIST\\_8x8.ipynb](#) y [MLP\\_MNIST\\_8x8\\_miniLote.ipynb](#)

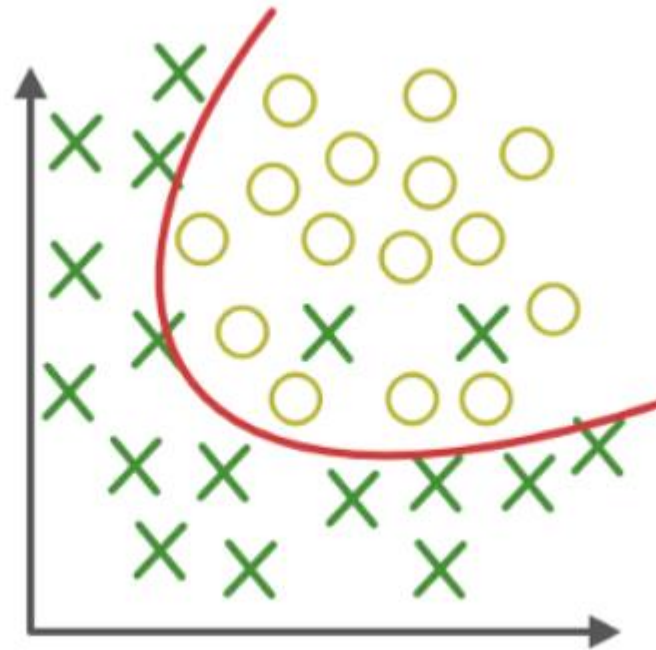
# Técnicas de optimización

- Descenso de gradiente estocástico (SGD) y el uso de mini-lotes
- Capacidad de generalización de la red - Sobreajuste 
- Mejoras introducidas
  - ▣ Momento: utiliza información de los gradientes anteriores
  - ▣ RMSProp: considera distintas magnitudes de cambio para reducir oscilaciones
  - ▣ Adam: combina los dos anteriores. Es el más usado.

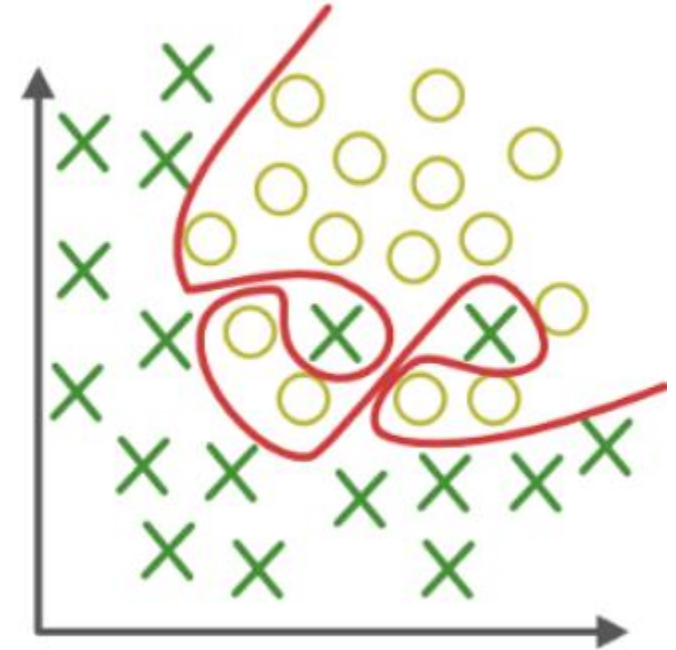
# Capacidad de generalización de la red



Underfitting  
(demasiado simple)



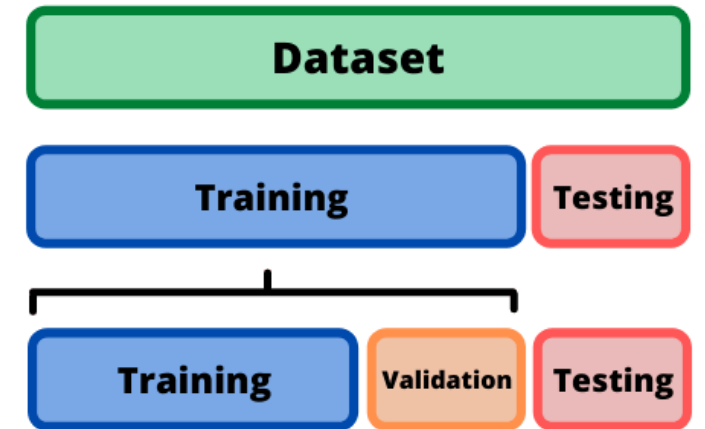
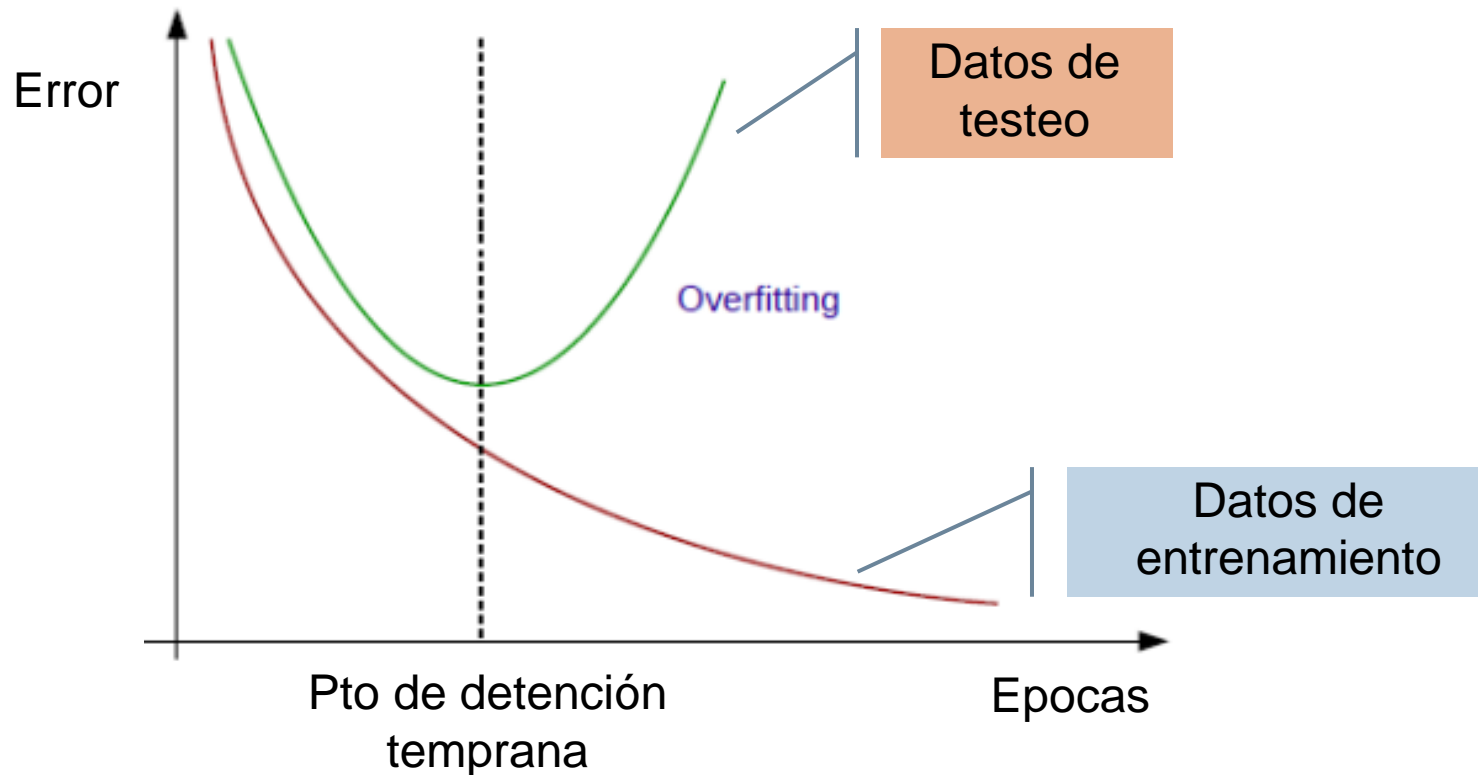
Generalización  
correcta



Overfitting  
(demasiados  
parámetros)

# Sobreaajuste

## □ Parada temprana (early-stopping)



# Parada temprana

```
from keras.callbacks import EarlyStopping  
  
model = ...  
model.compile( ... )  
  
es = EarlyStopping(monitor='val_accuracy', patience=30, min_delta=0.0001)  
  
H = model.fit(x = X_train, y = Y_train, epochs=4000, batch_size = 20,  
              validation_data = (X_test, Y_test), callbacks=[es])  
  
print("Epocas = %d" % es.stopped_epoch)
```

*Keras\_Iris\_softmax\_earlyStop.ipynb ; Keras\_IRIS\_Softmax\_earlyStop\_Valida.ipynb*

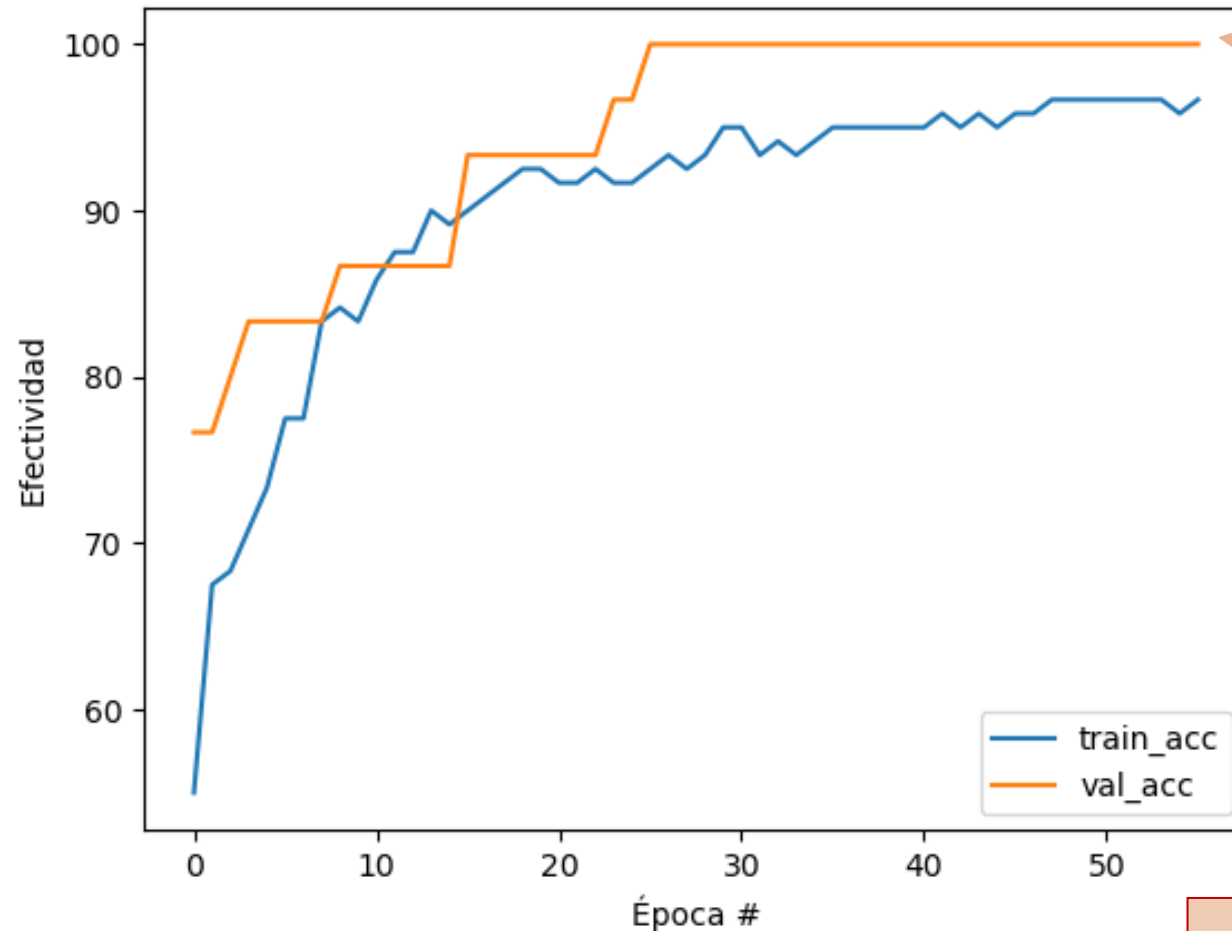
# EarlyStopping

- Detiene el entrenamiento cuando una métrica ha dejado de mejorar.
- Parámetros principales
  - ▣ **monitor:** valor a monitorear
  - ▣ **min\_delta:** un cambio absoluto en el valor monitoreado inferior a min\_delta, se considerará como que no hubo mejora.
  - ▣ **patience:** Número de épocas sin mejora tras las cuales se detendrá el entrenamiento.
  - ▣ **modo:** Uno de {"auto", "min", "max"}. En el modo "min", el entrenamiento se detendrá cuando el valor monitoreado haya dejado de disminuir; en el modo "max" se detendrá cuando el valor monitoreado haya dejado de aumentar; en el modo "auto", la dirección se infiere automáticamente del nombre del valor monitoreado.
  - ▣ **restore\_best\_weights:** Si se restauran los pesos del modelo de la época con el mejor resultado del valor monitoreado.

*[https://keras.io/api/callbacks/early\\_stopping/](https://keras.io/api/callbacks/early_stopping/)*



# Evolución del entrenamiento



***monitor='val\_accuracy'***  
***patience=30***  
***min\_delta=0.0001***

***Keras\_IRIS\_Softmax\_earlyStop.ipynb***

# Reducción del sobreajuste

- Si lo que se busca es reducir el sobreajuste puede probar
  - ▣ Incrementar la cantidad de ejemplos de entrenamiento.
  - ▣ Reducir la complejidad del modelo, es decir usar menos pesos (menos capas o menos neuronas por capa).
  - ▣ Aplicar una técnica de regularización
    - Regularización L2
    - Regularización L1
    - Dropout

*Tienen por objetivo que los pesos de la red se mantengan pequeños*

# Sobreaajuste - Regularización L2

- También conocida como técnica de decaimiento de pesos

$$C = C_o + \frac{\lambda}{2} \sum_k w_k^2$$

donde  $C_o$  es la función de costo original sin regularizar

- La derivada de la función de costo regularizada será

$$\frac{\partial C}{\partial w_k} = \frac{\partial C_o}{\partial w_k} + \lambda w_k$$

# Sobreajuste - Regularización L2

## Función de costo regularizada

$$C = C_o + \frac{\lambda}{2} \sum_k w_k^2$$

## Derivada

$$\frac{\partial C}{\partial w_k} = \frac{\partial C_o}{\partial w_k} + \lambda w_k$$

- Actualización de los pesos

$$w_k = w_k - \alpha \frac{\partial C_o}{\partial w_k} - \lambda w_k$$

$$w_k = (1 - \lambda) w_k - \alpha \frac{\partial C_o}{\partial w_k}$$

# Sobreajuste - Regularización L1

## Función de costo regularizada

$$C = C_0 + \lambda \sum_k |w_k|$$

## Derivada

$$\frac{\partial C}{\partial w_k} = \frac{\partial C_0}{\partial w_k} + \lambda \operatorname{sign}(w_k)$$

- Actualización de los pesos

$$w_k = w_k - \alpha \frac{\partial C_0}{\partial w_k} - \lambda \operatorname{sign}(w_k)$$

# Keras.regularizers

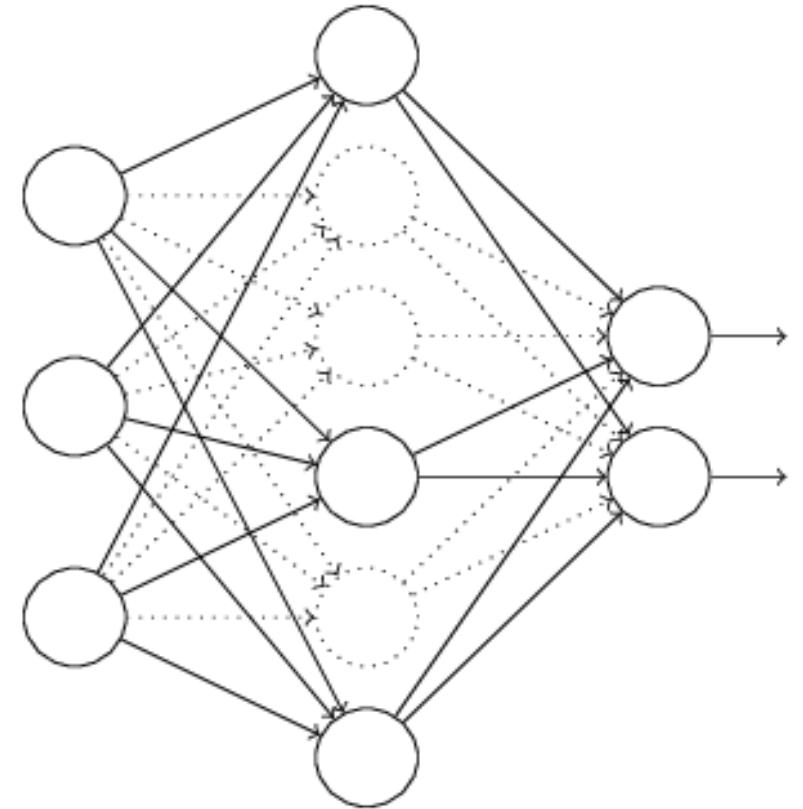
```
from keras.layers import Dense
from keras.regularizers import l2, l1, l1_l2
...
model.add(Dense(32, kernel_regularizer=l2(0.01),
                bias_regularizer=l2(0.01)))
```

- Se pueden aplicar ambos

```
model.add(Dense(32, kernel_regularizer=l1_l2(l1=0.01, l2=0.01),
                bias_regularizer=l1_l2(l1=0.01, l2=0.01)))
```

# Sobreajuste - Dropout

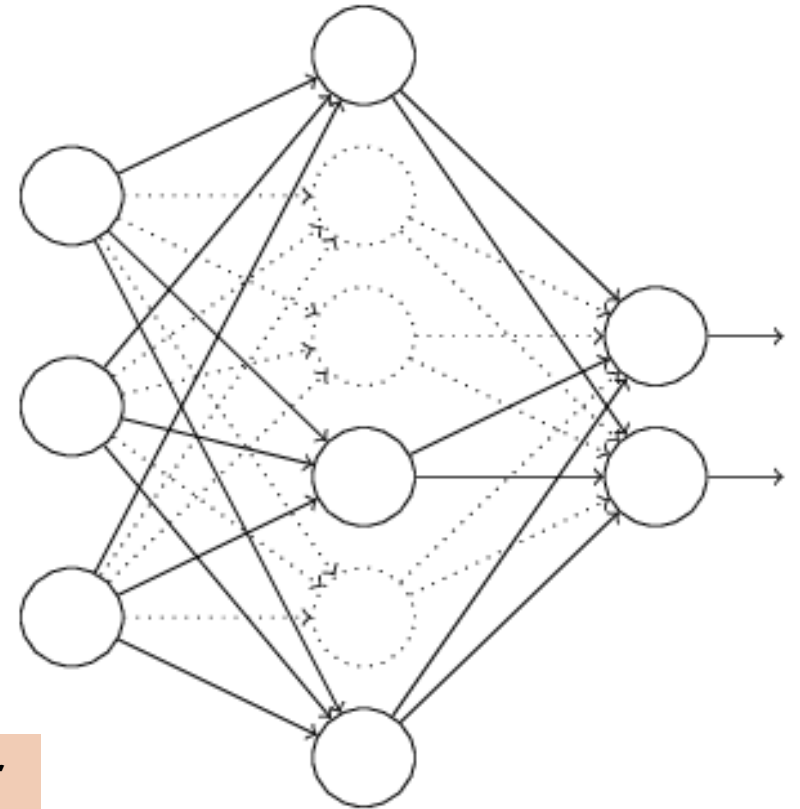
- No modifica la función de costo sino la arquitectura de la red.
- Proceso
  - ▣ Selecciona aleatoriamente las neuronas que no participarán en la próxima iteración y las “borra” temporalmente.
  - ▣ Actualiza los pesos (del mini lote si corresponde).
  - ▣ Restaura las neuronas “borradas”.
  - ▣ Repite hasta que se estabilice.



# Keras dropout

```
from keras.layers import Dense
from keras.layers import Dropout
...
model.add(Dense(6, input_shape=[3]))
model.add(Dropout(0.5))
model.add(Dense(2))
```

*Probabilidad de anular cada entrada de la capa anterior  
En este caso el 50% de las entradas serán anuladas*





# Técnicas de optimización

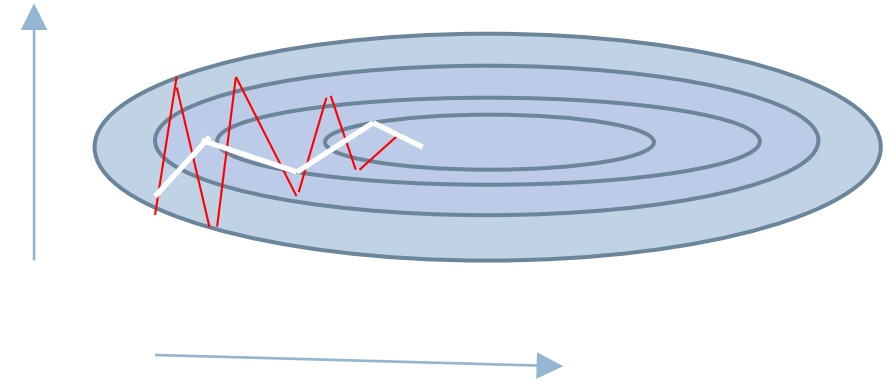
- Descenso de gradiente estocástico (SGD)
- Capacidad de generalización de la red - Sobreajuste
- Mejoras introducidas
  - ▣ Momento: utiliza información de los gradientes anteriores
  - ▣ RMSProp: considera distintas magnitudes de cambio para reducir oscilaciones
  - ▣ Adam: combina los dos anteriores. Es el más usado.



# SGD con momento

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla C)_t$$

$$W_t = W_t - \alpha v_t$$



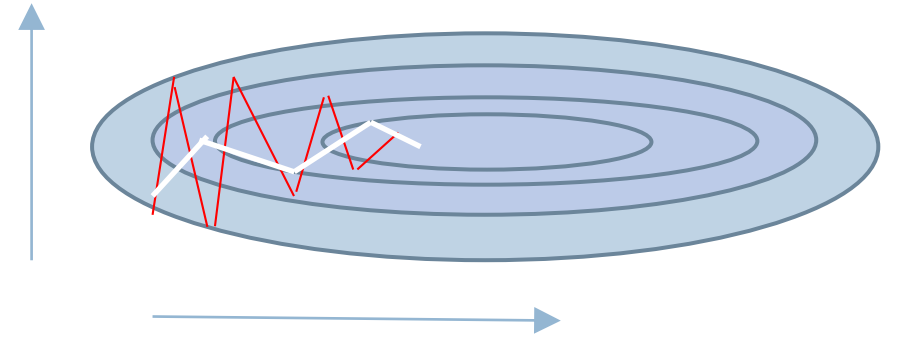
- Las modificaciones sobre  $W$  tienen en cuenta el promedio de los gradientes anteriores.
- La cantidad de gradientes anteriores a considerar son aprox.  $\frac{1}{1-\beta}$
- Esto reduce las oscilaciones.

# SGD con momento

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla C)_t$$

□ Usemos  $\beta = 0.9$  en la iteración  $t = 10$

$$v_{10} = 0.9 * v_9 + (1 - 0.9)(\nabla C)_{10}$$

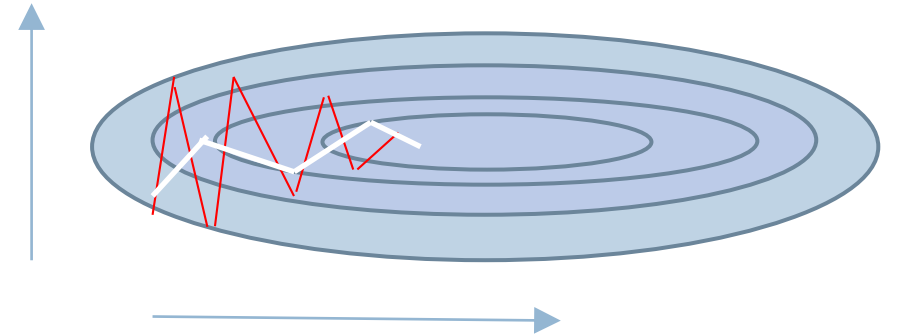


# SGD con momento

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla C)_t$$

□ Usemos  $\beta = 0.9$  en la iteración  $t = 10$

$$v_{10} = 0.9 * v_9 + (1 - 0.9)(\nabla C)_{10} = 0.1 \nabla C_{10} + 0.9 v_9$$



# SGD con momento

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla C)_t$$

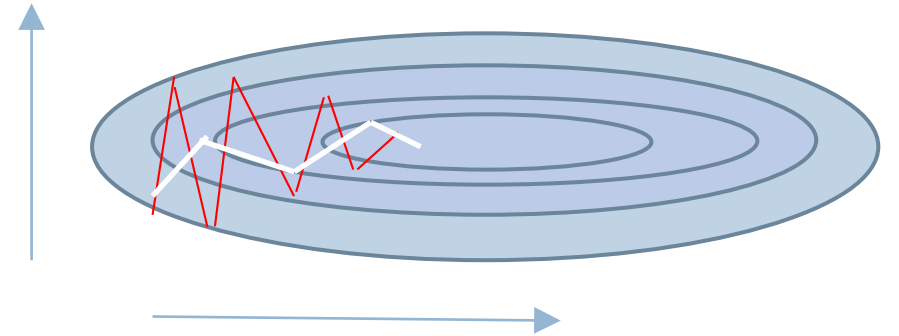
□ Usemos  $\beta = 0.9$  en la iteración  $t = 10$

$$v_{10} = 0.9 * v_9 + (1 - 0.9)(\nabla C)_{10} = 0.1 \nabla C_{10} + 0.9 v_9$$

$$v_{10} = 0.1 \nabla C_{10} + 0.9 (0.1 \nabla C_9 + 0.9 v_8)$$

$$v_{10} = 0.1 \nabla C_{10} + 0.1 * 0.9 \nabla C_9 + 0.9^2 v_8$$

$$v_{10} = 0.1 \nabla C_{10} + 0.1 * 0.9 \nabla C_9 + 0.9^2 (0.9 v_7 + 0.1 \nabla C_8)$$



# SGD con momento

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla C)_t$$

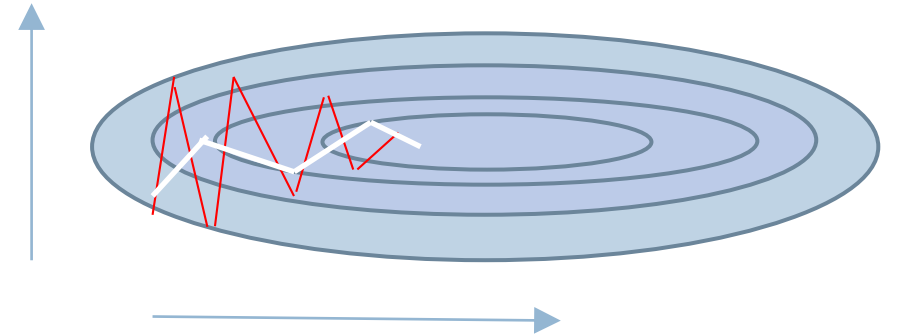
□ Usemos  $\beta = 0.9$  en la iteración  $t = 10$

$$v_{10} = 0.9 * v_9 + (1 - 0.9)(\nabla C)_{10} = 0.1 \nabla C_{10} + 0.9 v_9$$

$$v_{10} = 0.1 \nabla C_{10} + 0.9 (0.1 \nabla C_9 + 0.9 v_8)$$

$$v_{10} = 0.1 \nabla C_{10} + 0.1 * 0.9 \nabla C_9 + 0.9^2 v_8$$

$$v_{10} = 0.1 \nabla C_{10} + 0.1 * 0.9 \nabla C_9 + 0.1 * 0.9^2 \nabla C_8 + 0.9^3 v_7 + \dots$$



**La cantidad de gradientes anteriores a considerar son aprox.  $\frac{1}{1-\beta}$   $\therefore$  si  $\beta=0.9$  serán aprox. 10**

# SGD con momento

$V_w = 0$

$V_b = 0$

for t in range(iteraciones):

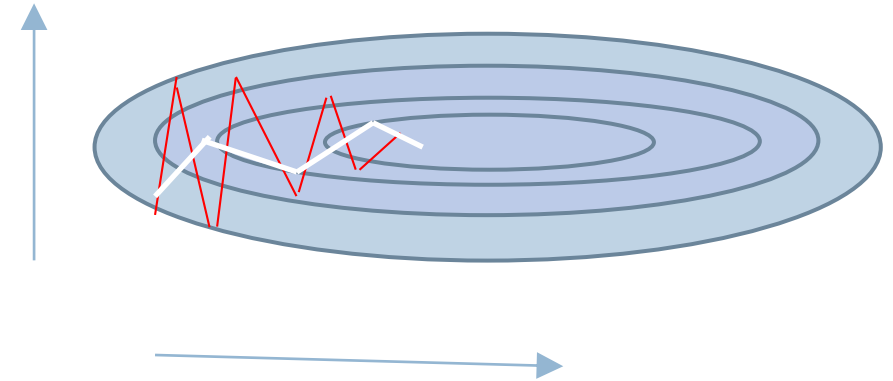
    Calcular gradientes  $\nabla_w$  y  $\nabla_b$

$V_w = \text{beta} * V_w + (1-\text{beta}) * \nabla_w$

$V_b = \text{beta} * V_b + (1-\text{beta}) * \nabla_b$

$W = W - \text{alfa} * V_w$

$b = b - \text{alfa} * V_b$

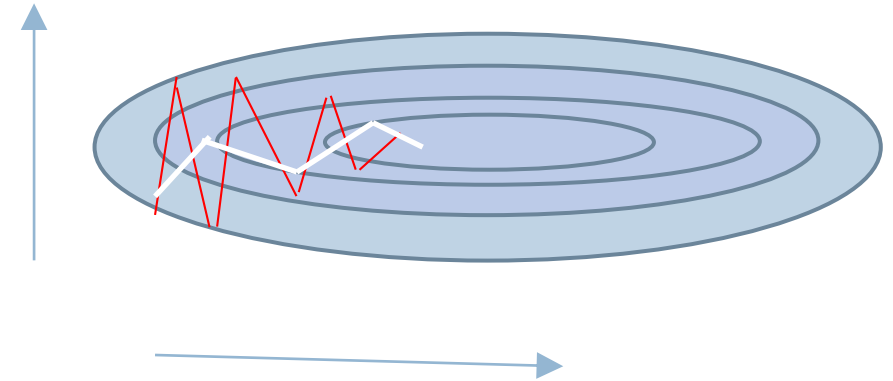


`keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)`

# RMSprop

$$s = \beta s + (1 - \beta) (\nabla C)^2$$

$$w = w - \alpha \frac{\nabla C}{\sqrt{s + \varepsilon}}$$



- Las modificaciones sobre  $W$  tienen en cuenta el promedio de los gradientes anteriores.
- Las modificaciones más grandes serán divididas por coeficientes más grandes; por lo tanto se reducen.
- Las modificaciones más chicas se incrementan.

***Es más eficiente que  
SGD+Momento***



# RMSprop

```
from keras.optimizers import RMSprop

X,Y = cargar_datos()

model = Sequential()
model.add(...)

model.compile(
    loss='categorical_crossentropy',
    optimizer = RMSprop(lr=0.001),
    metrics=['accuracy'])

model.fit(X,Y, epochs=10, batch_size=32)
```

# ADAM

<https://keras.io/api/optimizers/adam/>

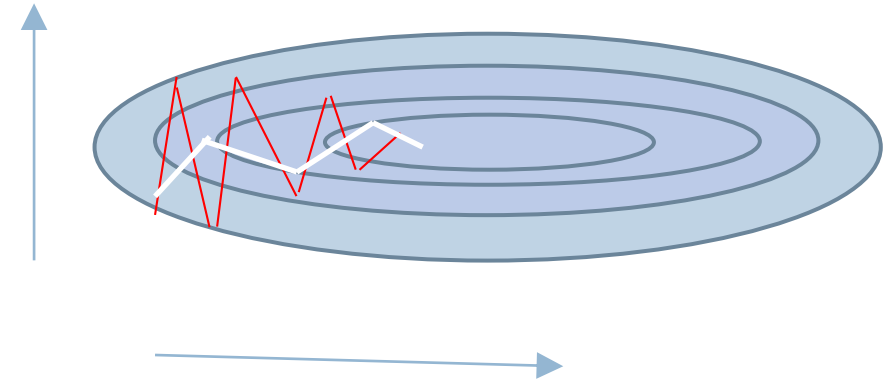
- Combina momento y RMSprop

$$v = \beta_1 v + (1 - \beta_1) \nabla C$$

$$s = \beta_2 s + (1 - \beta_2) (\nabla C)^2$$

$$w = w - \alpha \frac{v}{\sqrt{s + \epsilon}}$$


- Los valores recomendados son  $\beta_1 = 0.9$  y  $\beta_2 = 0.999$



```
model.compile(optimizer='adam', loss='mse')
```

# Resumen

## Resolución de una tarea de clasificación

- Conjunto de datos etiquetados (aprendizaje supervisado)
- Definición de la arquitectura de la red
  - ▣ Número de capas y tamaño de cada una
  - ▣ Función de activación a usar en cada capa
- Entrenamiento
  - ▣ Función de error
  - ▣ Técnica de optimización para reducir el error
- Evaluar el modelo 

# Evaluación del modelo

- Matriz de confusión
- Métricas
  - ▣ Accuracy
  - ▣ Precisión
  - ▣ Recall
  - ▣ F1 -score
  - ▣ AUC, Curva ROC

# Clasificación binaria

- Los resultados se etiquetan como positivos (P) o negativos (N)
- Luego, la matriz de confusión tendrá la siguiente forma:

	Predice P	Predice N	
Clase P	VP	FN	$P = VP + FN$
Clase N	FP	VN	$N = FP + VN$

- ▣ Tasa de verdaderos positivos  $\rightarrow TVP = VP / P$  (Sensibilidad)
- ▣ Tasa de Falsos Positivos  $\rightarrow TFP = FP / N$  (Falsas alarmas)
- ▣ Tasa de verdaderos negativos  $\rightarrow TVN = VN / N$  (Especificidad)

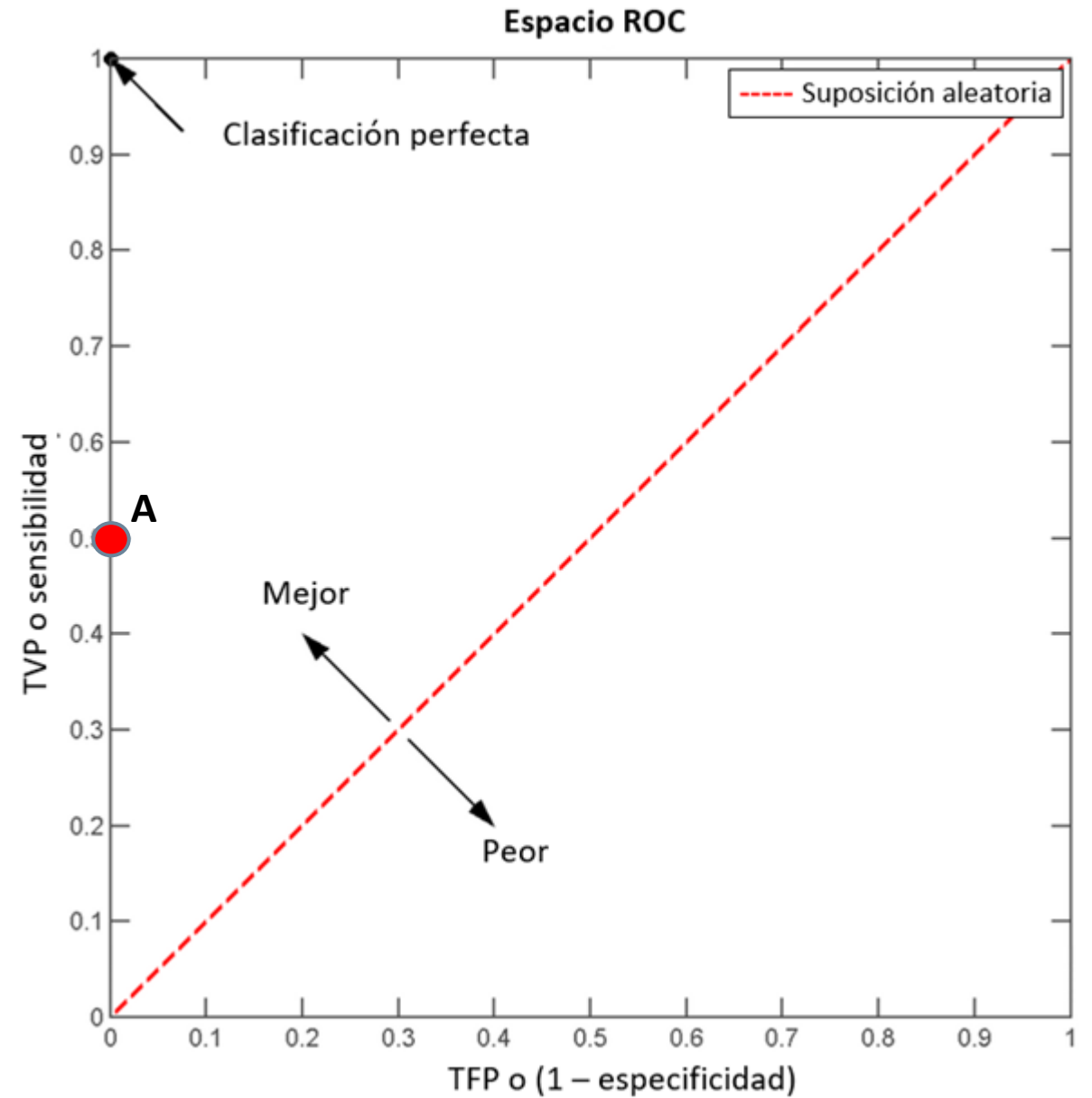
**A**

VP=6	FN=6	12
FP=0	VN=8	8

6          14      20

$$\text{TVP} = 6/12 = 0.5$$

$$\text{TFP} = 0/8 = 0$$



**A**

VP=6	FN=6	12
FP=0	VN=8	8

6      14      20

$$\text{TVP} = 6/12 = 0.5$$

$$\text{TFP} = 0/8 = 0$$

**B**

VP=9	FN=3	12
FP=6	VN=2	8

15      5      20

$$\text{TVP} = 9/12 = 0.75$$

$$\text{TFP} = 6/8 = 0.75$$

**C**

VP=12	FN=0	12
FP=2	VN=6	8

14      6      20

$$\text{TVP} = 12/12 = 1$$

$$\text{TFP} = 2/8 = 0.25$$

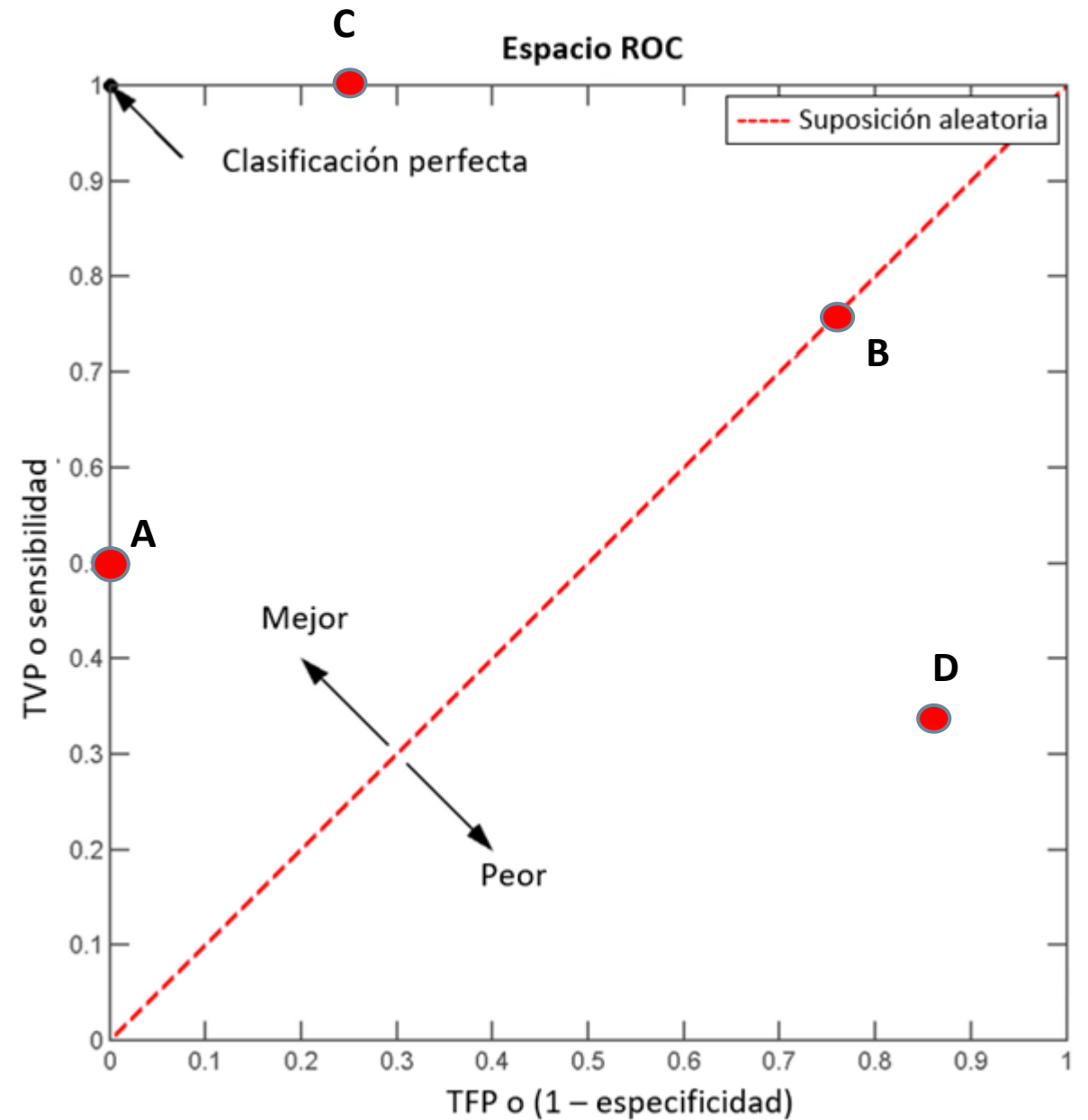
**D**

VP=4	FN=8	12
FP=7	VN=1	8

11      9      20

$$\text{TVP} = 4/12 = 0.33$$

$$\text{TFP} = 7/8 = 0.875$$



# Roca o Mina

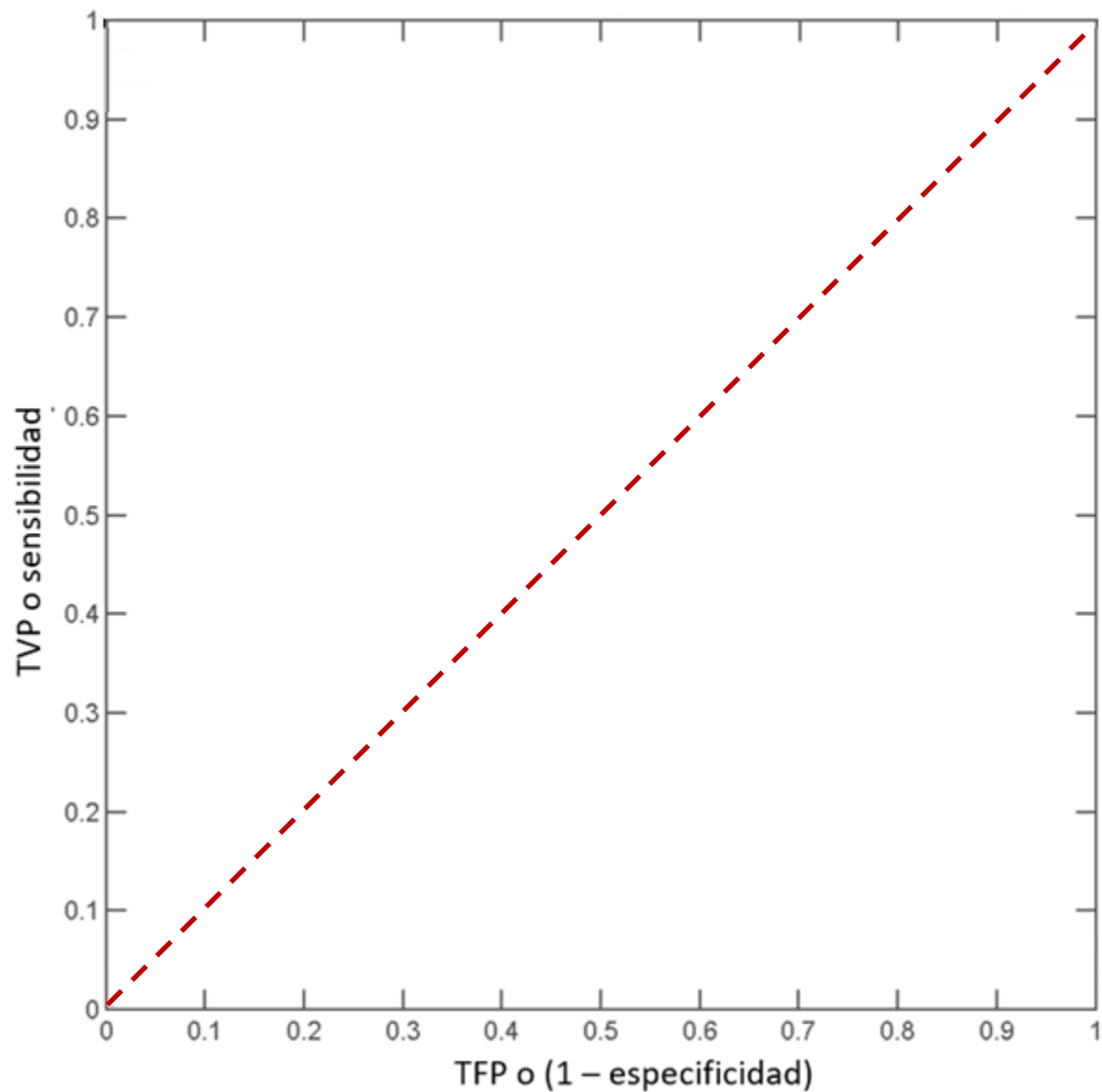
- A partir de los datos del archivo “Sonar.csv” se desea construir una red neurona multiperceptrón para discriminar entre señales de sonar rebotadas en un cilindro de metal (“Mine”) y aquellas rebotadas en una roca más o menos cilíndrica (“Rock”).
- Probar con distintas configuraciones
- Indicar cuál recomendaría a la hora de predecir si es una mina o no utilizando: accuracy, f1-score y AUC.



ID	Clase	Confianza	Predice
5	Mina	0.99	
7	Mina	0.99	
9	Mina	0.99	
1	Mina	0.9	
10	Mina	0.9	
20	Mina	0.9	
8	Roca	0.8	
14	Mina	0.8	
15	Mina	0.8	
18	Roca	0.8	
19	Mina	0.8	
3	Mina	0.7	
6	Mina	0.7	
12	Mina	0.65	
4	Roca	0.6	
16	Roca	0.6	
11	Roca	0.5	
2	Roca	0.4	
13	Roca	0.3	
17	Roca	0.1	

**12 minas** y **8 rocas**

**CURVA ROC**

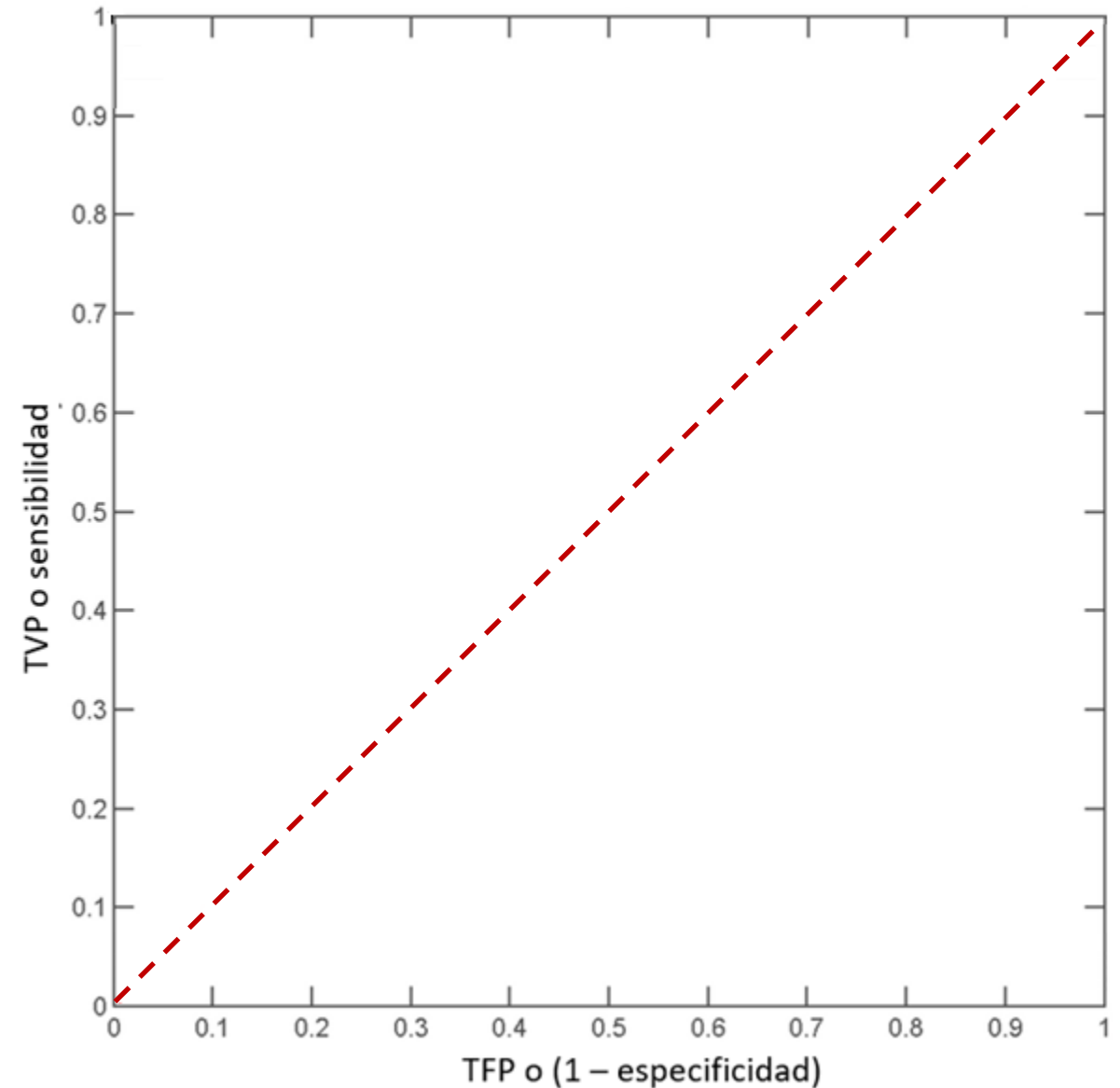


ID	Clase	Confianza	Predice
5	Mina	0.99	
7	Mina	0.99	
9	Mina	0.99	
1	Mina	0.9	
10	Mina	0.9	
20	Mina	0.9	
8	Roca	0.8	
14	Mina	0.8	
15	Mina	0.8	
18	Roca	0.8	
19	Mina	0.8	
3	Mina	0.7	
6	Mina	0.7	
12	Mina	0.65	
4	Roca	0.6	
16	Roca	0.6	
11	Roca	0.5	
2	Roca	0.4	
13	Roca	0.3	
17	Roca	0.1	

12 minas y 8 rocas

Umbral = 0.99

## CURVA ROC



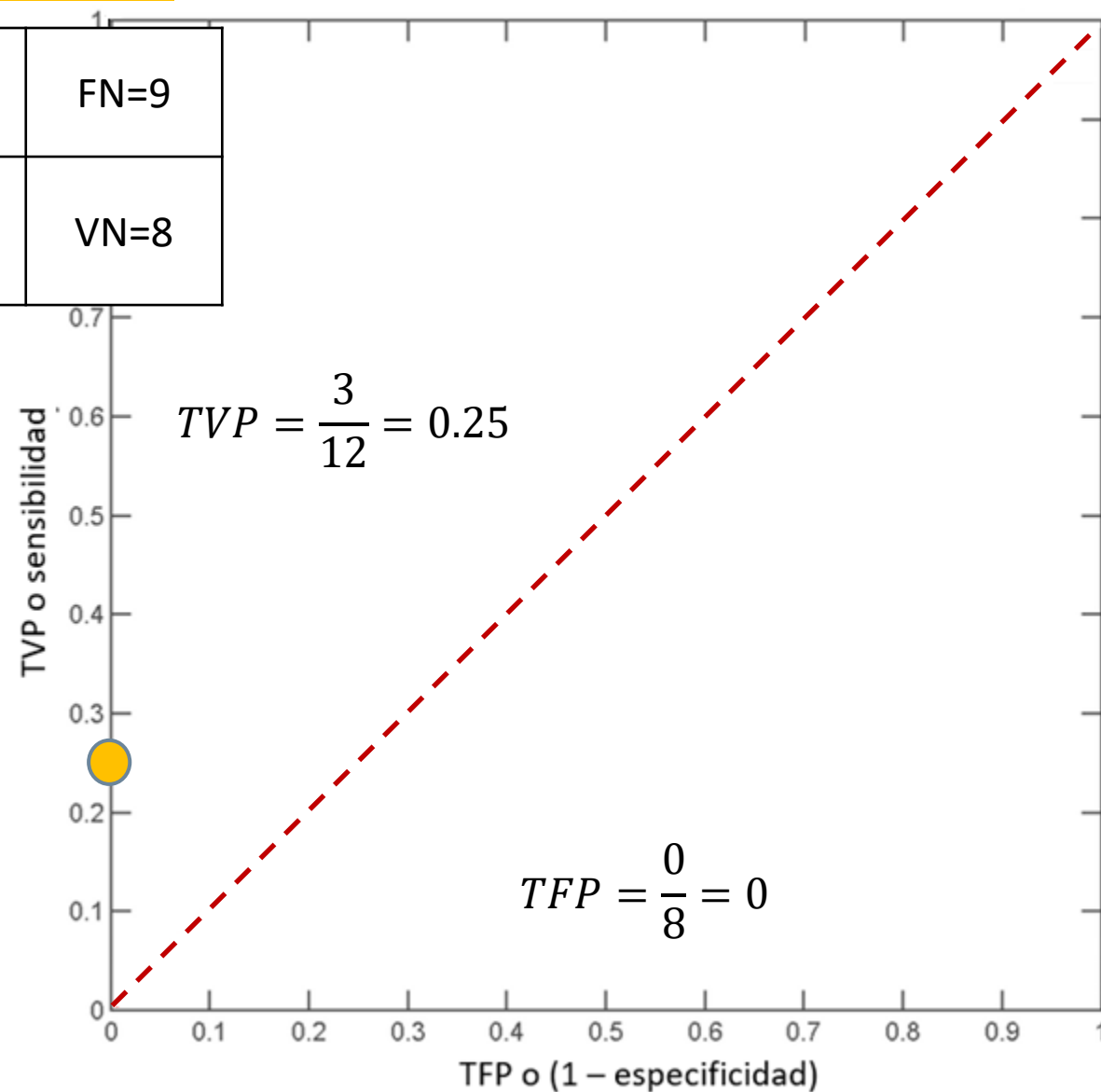
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Roca
10	Mina	0.9	Roca
20	Mina	0.9	Roca
8	Roca	0.8	Roca
14	Mina	0.8	Roca
15	Mina	0.8	Roca
18	Roca	0.8	Roca
19	Mina	0.8	Roca
3	Mina	0.7	Roca
6	Mina	0.7	Roca
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y 8 rocas

Umbral = 0.99

VP=3	FN=9
FP=0	VN=8

## CURVA ROC



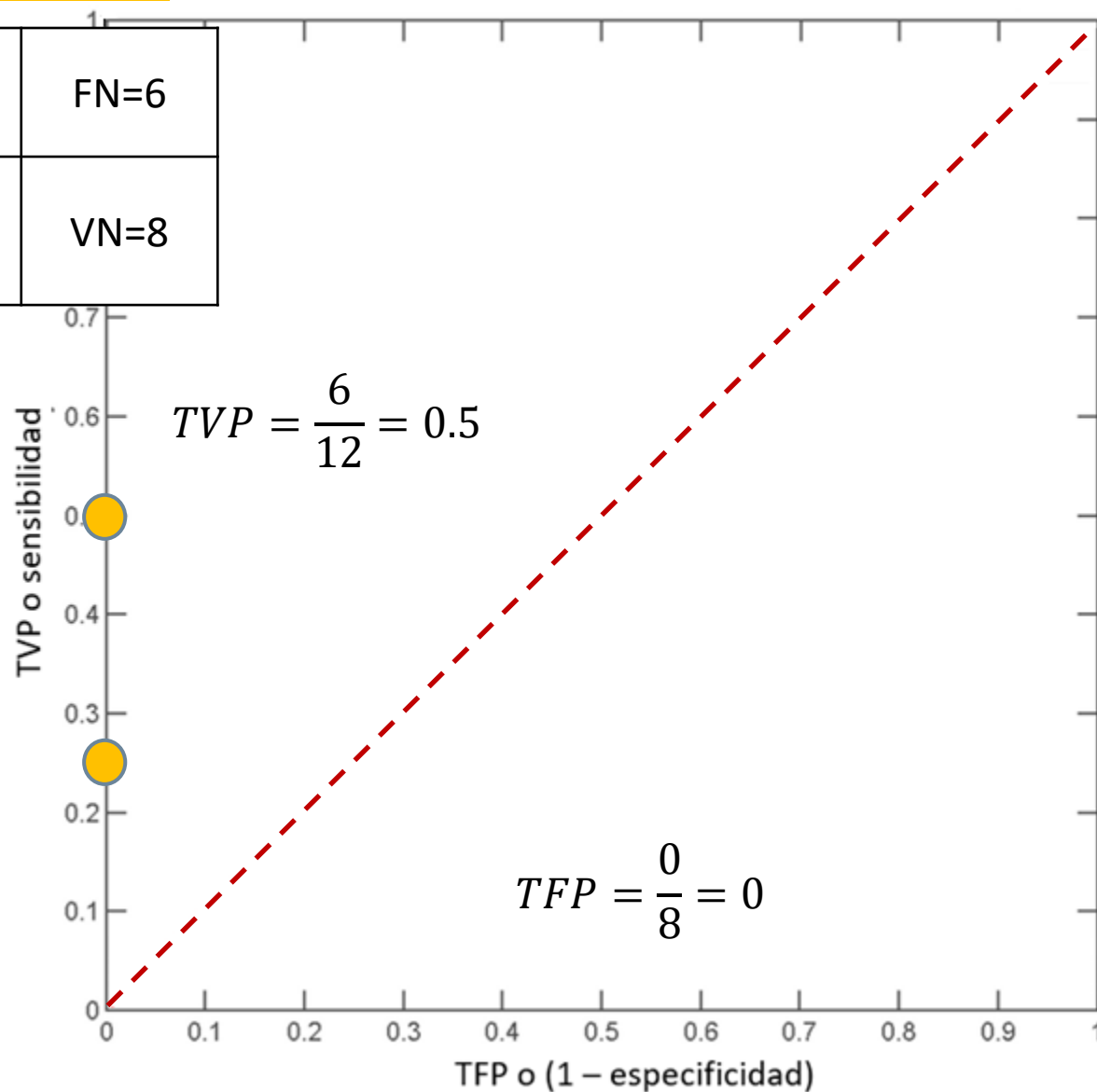
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Roca
14	Mina	0.8	Roca
15	Mina	0.8	Roca
18	Roca	0.8	Roca
19	Mina	0.8	Roca
3	Mina	0.7	Roca
6	Mina	0.7	Roca
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y 8 rocas

Umbral = 0.9

VP=6	FN=6
FP=0	VN=8

## CURVA ROC



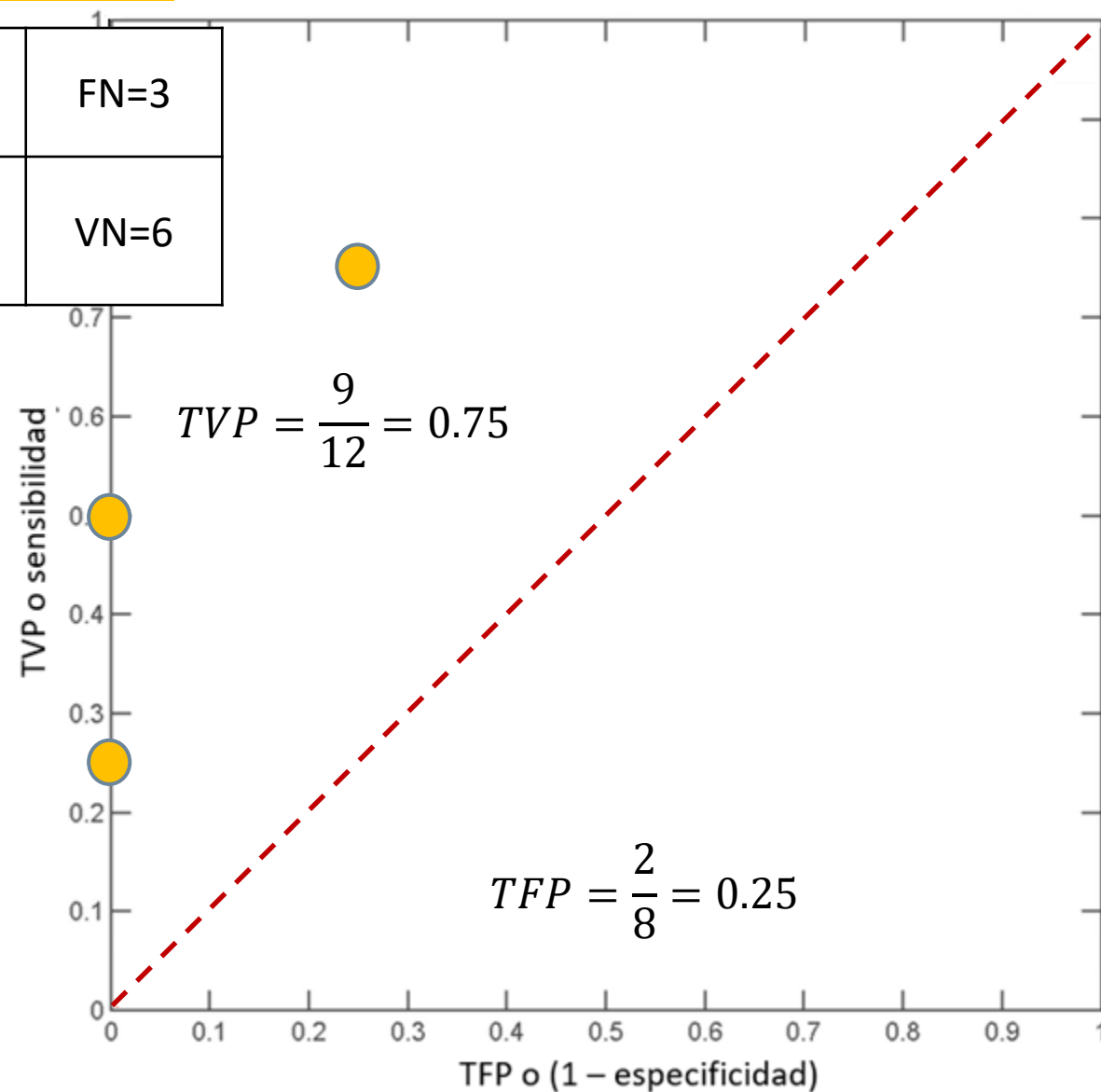
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Roca
6	Mina	0.7	Roca
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y 8 rocas

Umbral = 0.8

VP=9	FN=3
FP=2	VN=6

## CURVA ROC



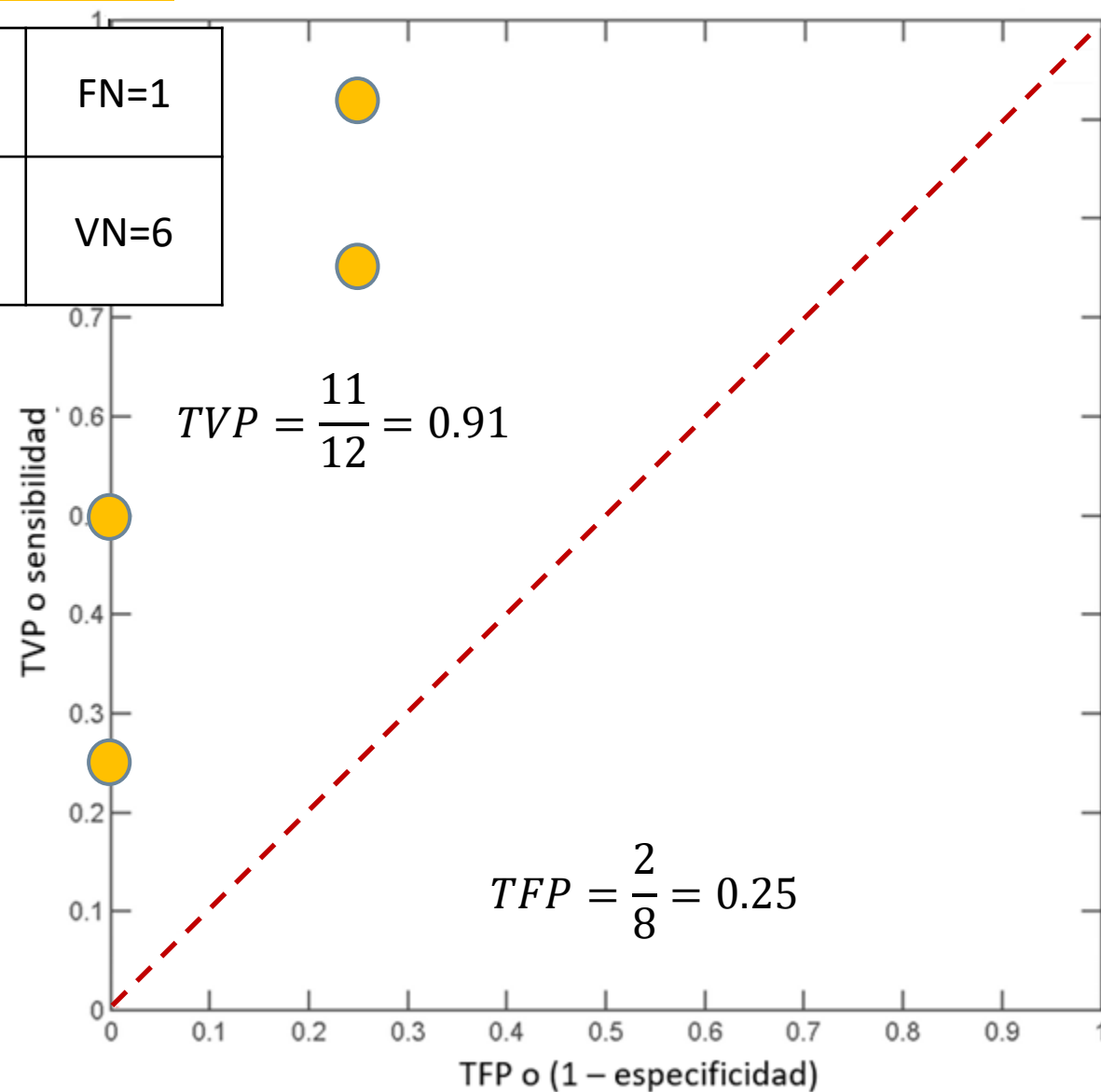
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y 8 rocas

Umbral = 0.7

VP=11	FN=1
FP=2	VN=6

## CURVA ROC



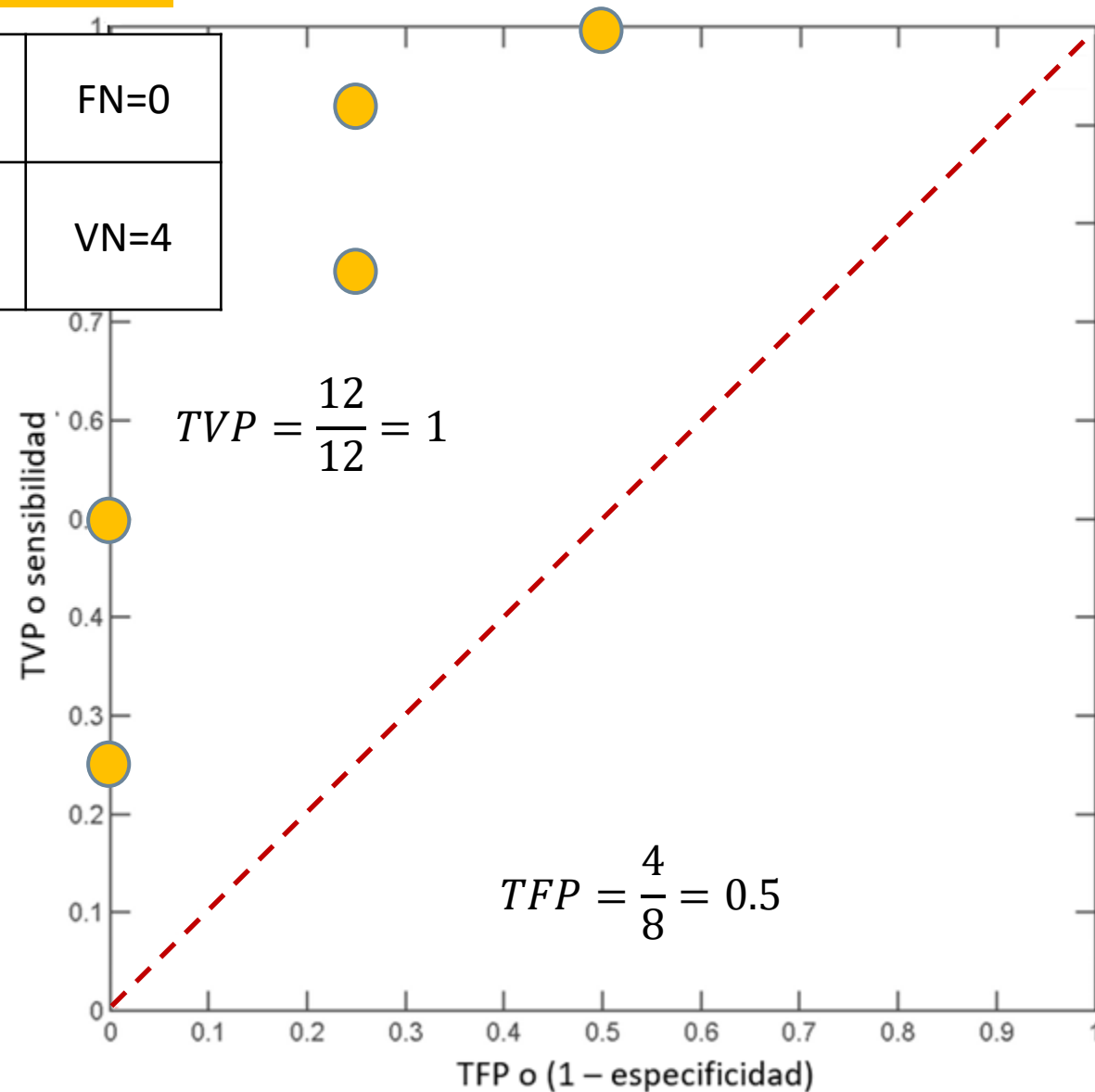
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y 8 rocas

Umbral = 0.6

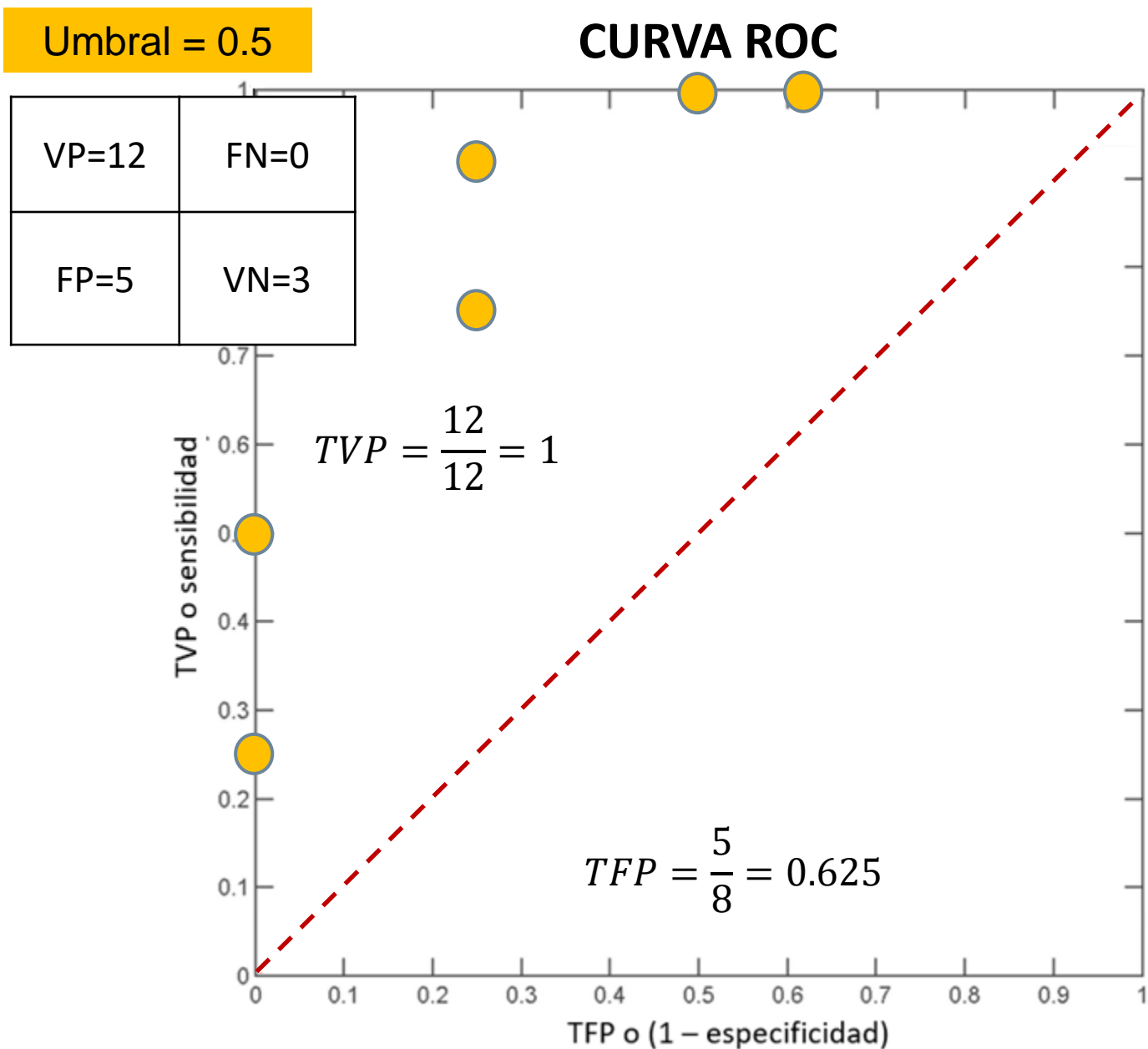
VP=12	FN=0
FP=4	VN=4

## CURVA ROC



ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

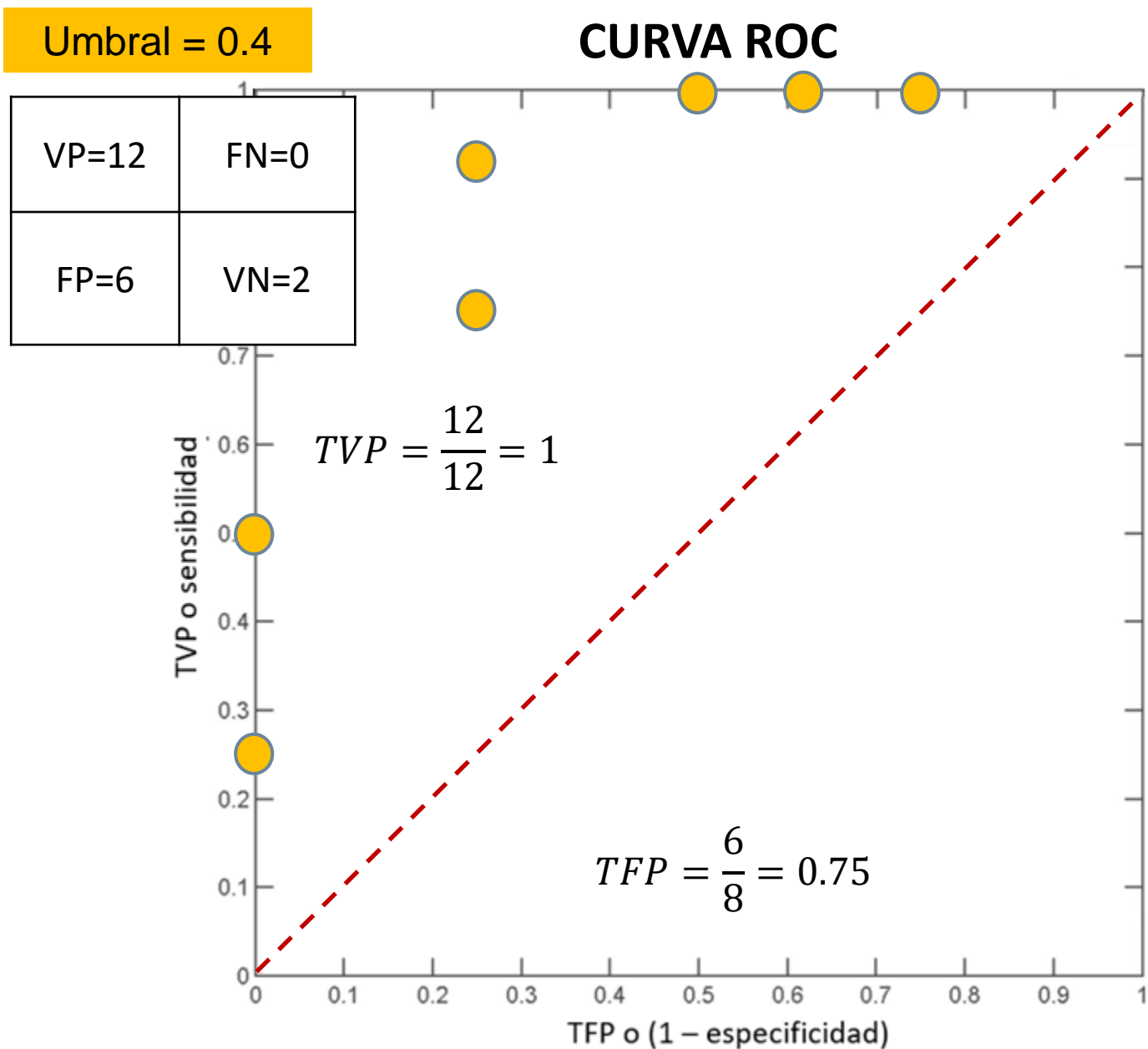
12 minas y 8 rocas





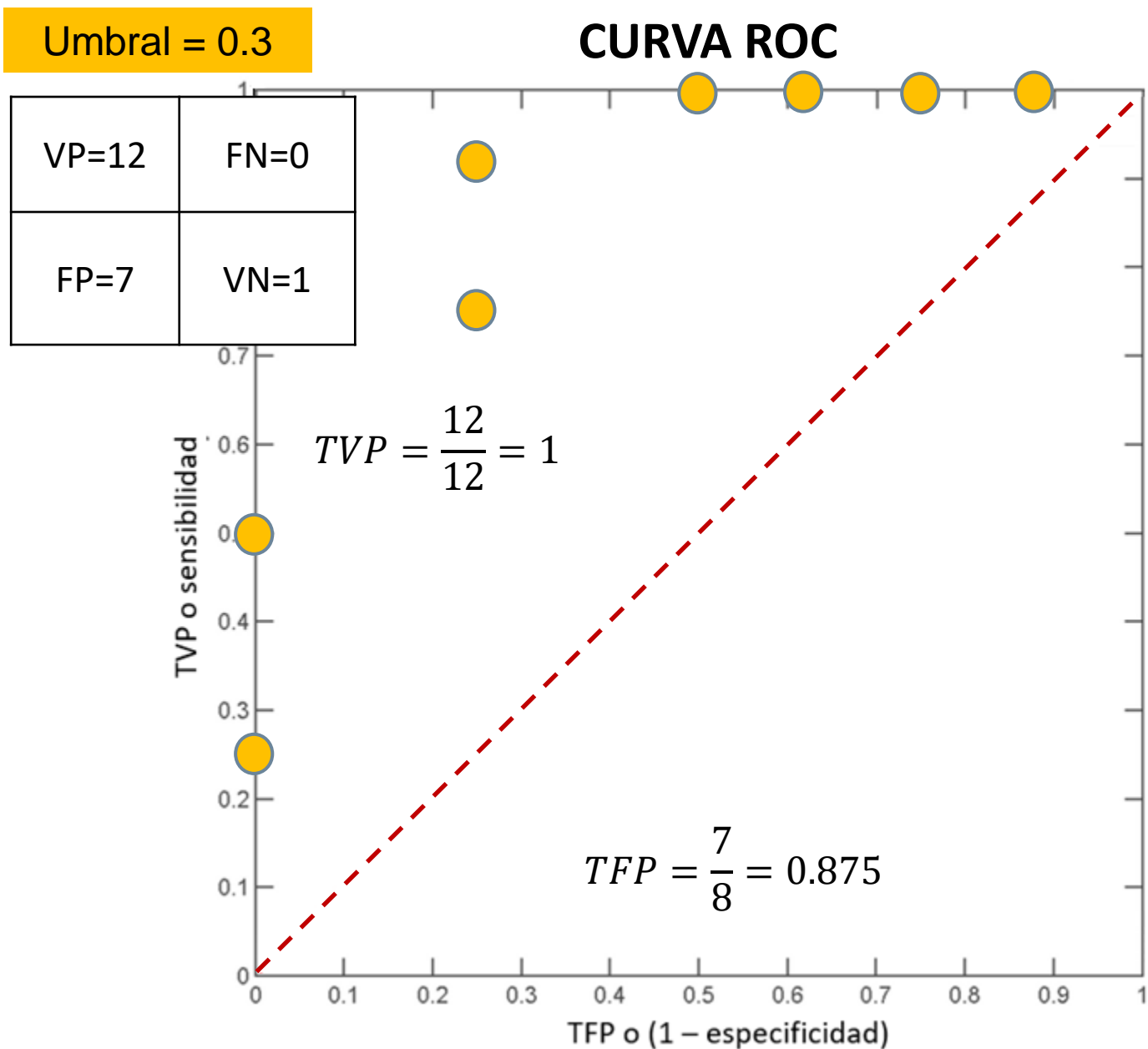
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y 8 rocas



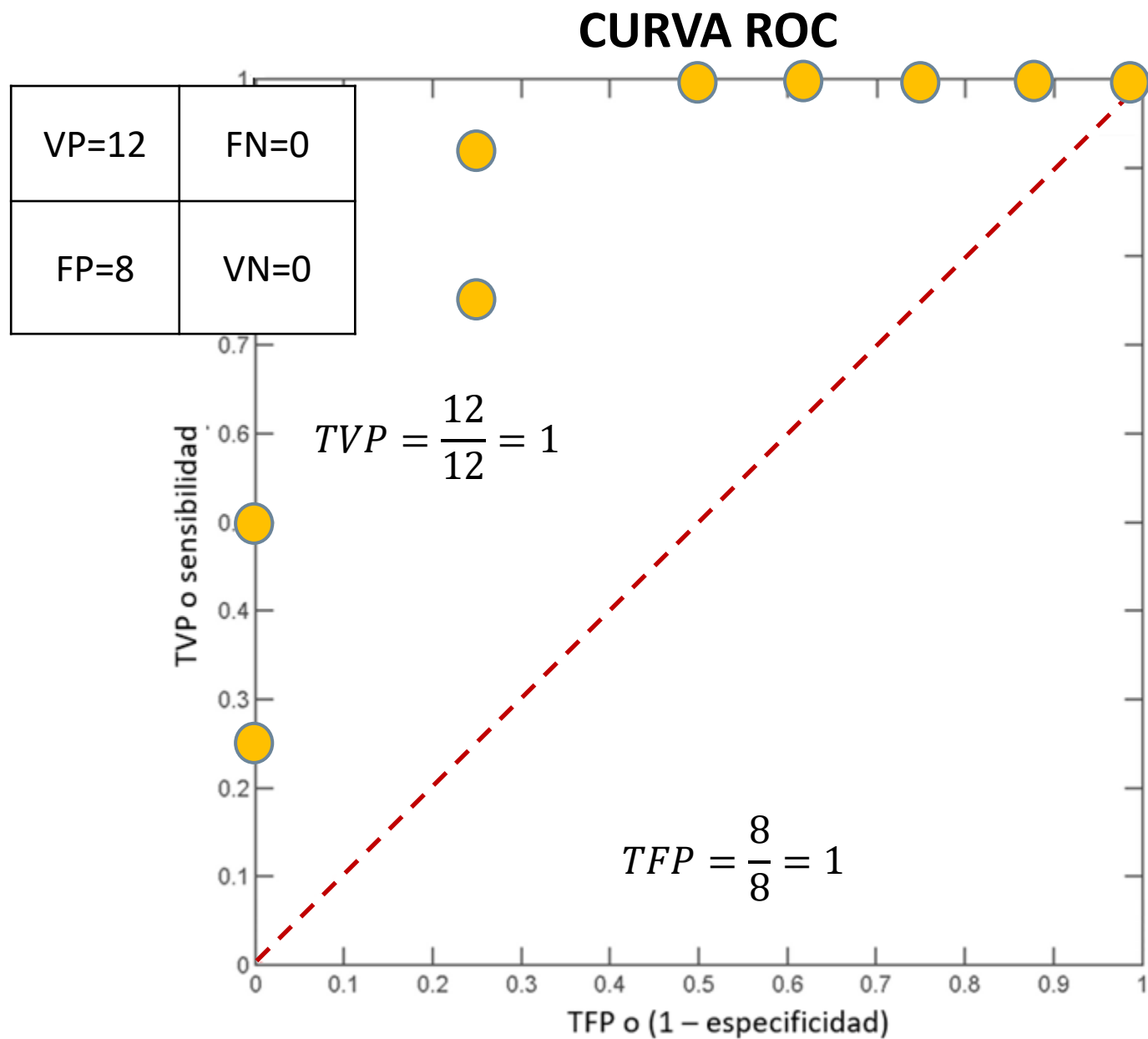
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Roca

12 minas y 8 rocas



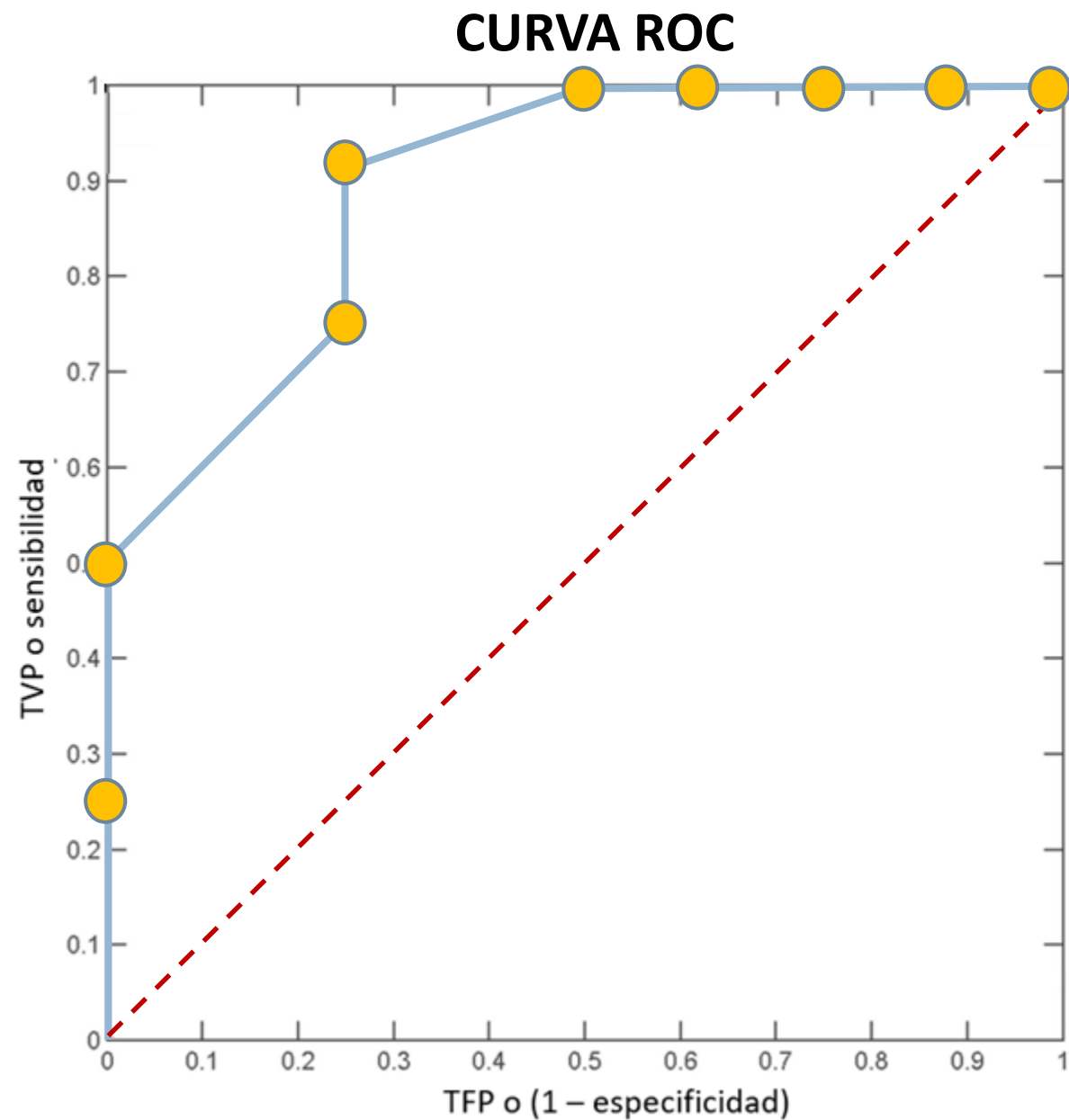
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Mina

12 minas y 8 rocas



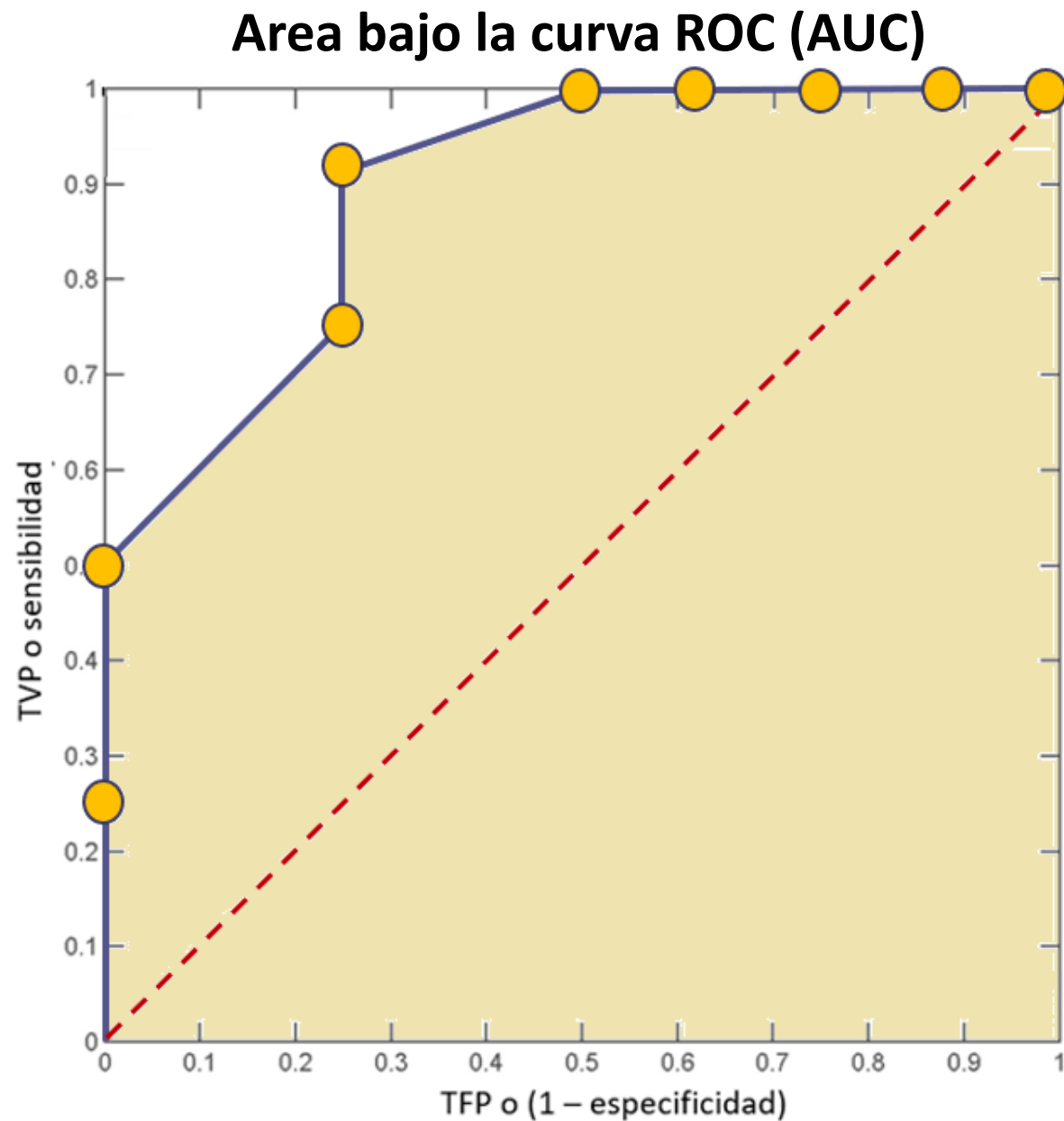
ID	Clase	Confianza	Predice
5	Mina	0.99	
7	Mina	0.99	
9	Mina	0.99	
1	Mina	0.9	
10	Mina	0.9	
20	Mina	0.9	
8	Roca	0.8	
14	Mina	0.8	
15	Mina	0.8	
18	Roca	0.8	
19	Mina	0.8	
3	Mina	0.7	
6	Mina	0.7	
12	Mina	0.65	
4	Roca	0.6	
16	Roca	0.6	
11	Roca	0.5	
2	Roca	0.4	
13	Roca	0.3	
17	Roca	0.1	

**12 minas** y **8 rocas**



ID	Clase	Confianza	Predice
5	Mina	0.99	
7	Mina	0.99	
9	Mina	0.99	
1	Mina	0.9	
10	Mina	0.9	
20	Mina	0.9	
8	Roca	0.8	
14	Mina	0.8	
15	Mina	0.8	
18	Roca	0.8	
19	Mina	0.8	
3	Mina	0.7	
6	Mina	0.7	
12	Mina	0.65	
4	Roca	0.6	
16	Roca	0.6	
11	Roca	0.5	
2	Roca	0.4	
13	Roca	0.3	
17	Roca	0.1	

**12 minas** y **8 rocas**



# Roca o Mina

- A partir de los datos del archivo “Sonar.csv” se desea construir una red neurona multiperceptrón para discriminar entre señales de sonar rebotadas en un cilindro de metal (“Mine”) y aquellas rebotadas en una roca más o menos cilíndrica (“Rock”).
- Probar con distintas configuraciones
- Indicar cuál recomendaría a la hora de predecir si es una mina o no utilizando: accuracy, f1-score y AUC.

# Curva ROC

```
fpr, tpr, threshold = metrics.roc_curve(Y_true, Y_prob)
roc_auc = metrics.auc(fpr, tpr)

plt.figure()
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
```

Keras\_SONAR\_softmax\_AUC.ipynb

# Curva ROC

Keras\_SONAR\_softmax\_AUC.ipynb

