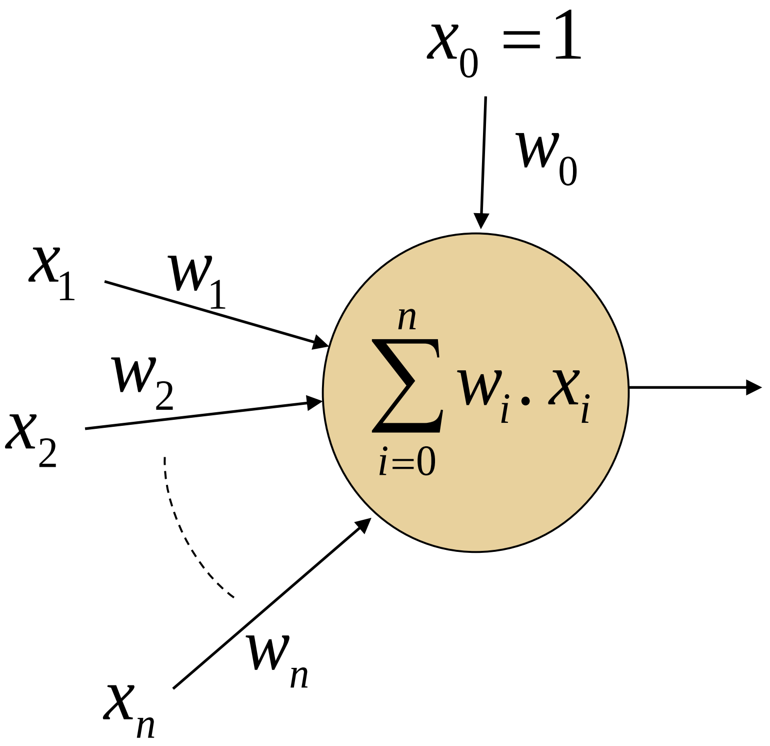
Combinador Lineal

- Resuelve un problema de **Regresión Lineal**
- Función de Error
  - Error cuadrático medio
- Técnica de optimización
  - Descenso de gradiente estocástico



$$y = \sum_{i=0}^{n} x_i . w_i$$

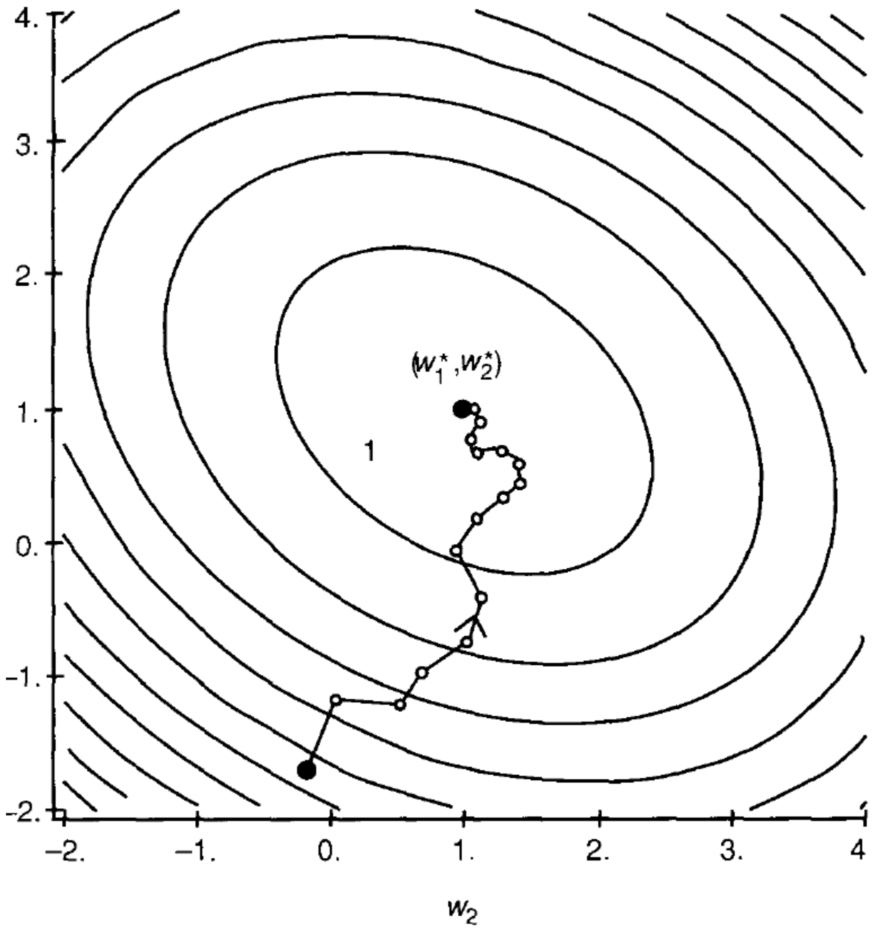
Técnica del descenso del gradiente estocástico

$$w(t+1) = w(t) + \Delta w(t)$$

$$w(t+1) = w(t) - \mu \nabla \xi(w(t))$$

- se utiliza

$$\xi =< \varepsilon_k^2 > \approx \varepsilon_k^2 = (d_k - \sum_{i=0}^{N} x_{ik} w_i)^2$$



Técnica del descenso del gradiente estocástico

$$w(t+1) = w(t) + \Delta w(t)$$

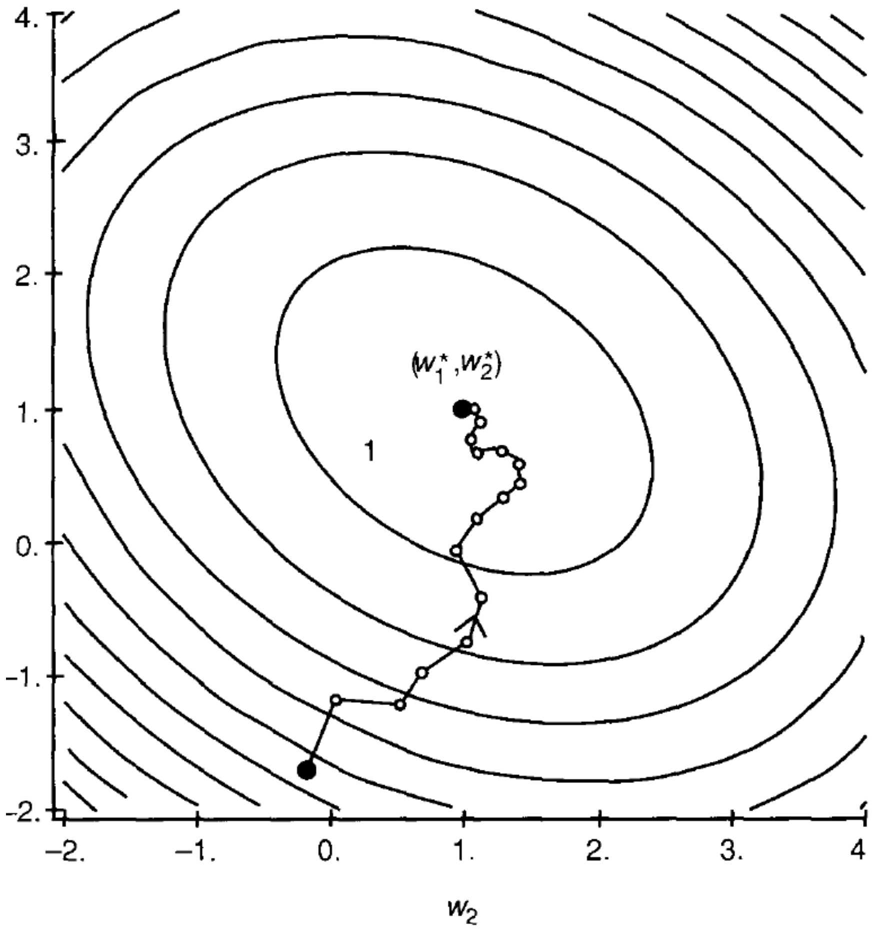
$$w(t+1) = w(t) - \mu \nabla \xi(w(t))$$

- se utiliza

$$\xi =< \varepsilon_k^2 > \approx \varepsilon_k^2 = (d_k - \sum_{i=0}^{N} x_{ik} w_i)^2$$

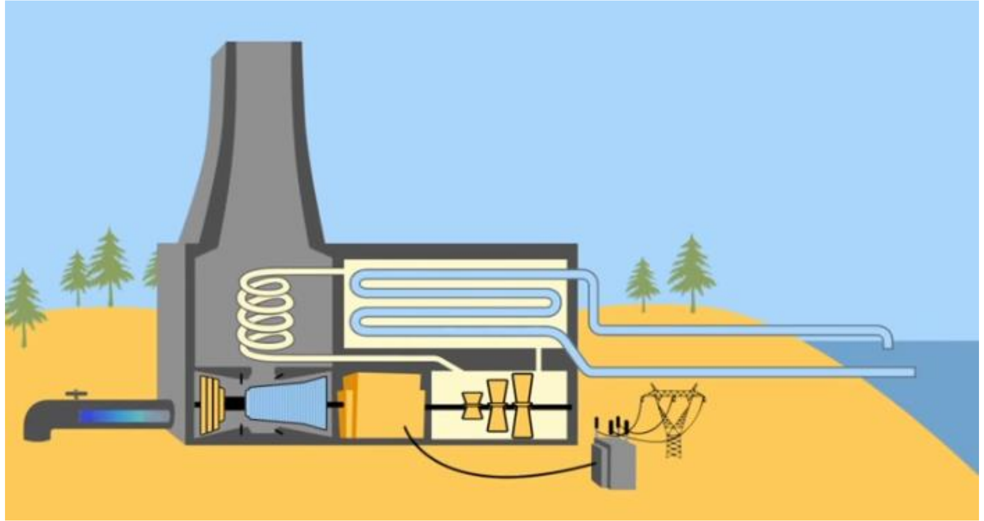
- veamos que

$$\nabla \varepsilon_k^2(t) = -2\varepsilon_k(t) x_k$$



Ejemplo - Central de energía eléctrica

- El archivo **CCPP.csv** contiene 9568 datos de una Central de Ciclo Combinado recolectados entre 2006 y 2011.
- Objetivo: Predecir la producción neta de energía eléctrica por hora (PE) de la planta
- Las características relevadas son:
  - Temperatura ambiente (AT)
  - Presión ambiente (AP)
  - Humedad relativa (RH)
  - Vacío de escape (V)



(ver cómo funciona la central)

Ejemplo - Central de energía eléctrica

- Matriz de Correlación

Index	AT	V	AP	RH	PE
AT	1	0.844107	-0.507549	-0.542535	-0.948128
V	0.844107	1	-0.413502	-0.312187	-0.86978
AP	-0.507549	-0.413502	1	0.0995743	0.518429
RH	-0.542535	-0.312187	0.0995743	1	0.389794
PE	-0.948128	-0.86978	0.518429	0.389794	1

Predicción de la Producción de Energía (PE)

X

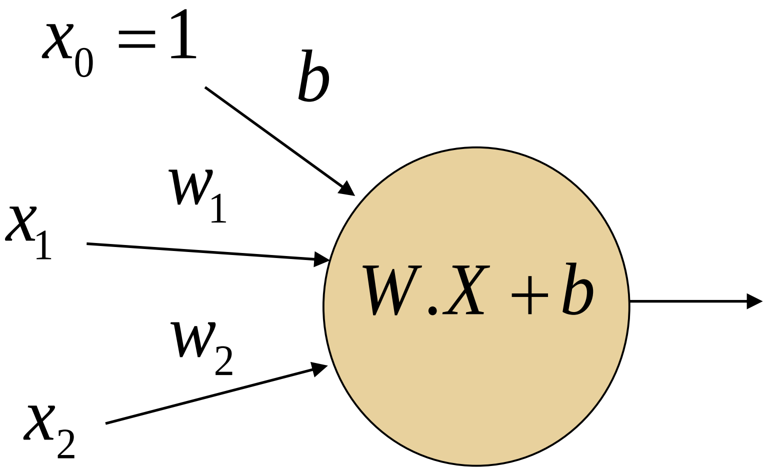
```
[[14.96, 41.76],
 [25.18, 62.96],
 [ 5.11, 39.4 ],
 ...,
 [31.32, 74.33],
 [24.48, 69.45],
 [21.6 , 62.52]]
```

AT V

Y

```
[463.26,
 444.37,
 488.56,
 ...,
 429.57,
 435.74,
 453.28]
```

PE



$$\hat{y} = \sum_{i=1}^{2} w_i . x_i + b$$

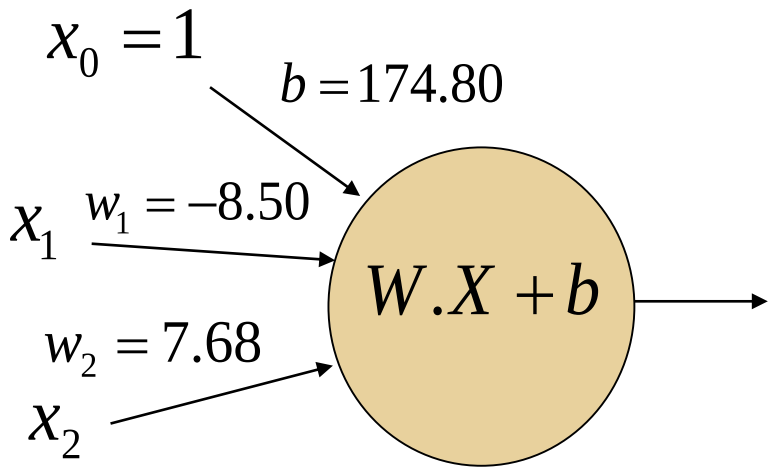
Predicción de la Producción de Energía (PE)

X

```
[[14.96, 41.76],
 [25.18, 62.96],
 [ 5.11, 39.4 ],
 ...,
 [31.32, 74.33],
 [24.48, 69.45],
 [21.6 , 62.52]]
```

Y

```
[463.26,
 444.37,
 488.56,
 ...,
 429.57,
 435.74,
 453.28]
```



$$\hat{y} = \sum_{i=1}^{2} w_i . x_i + b$$

$$\underbrace{(-8.50 \quad 7.68)}_{\mathbf{W}} * \begin{pmatrix} 14.96 \\ 41.73 \end{pmatrix} + \underbrace{174.80}_{\mathbf{b}}$$

$$\hat{y} = 368.49$$

$$y = 463.26$$

Error cometido en este ejemplo $\longrightarrow$ $Error = (y - \hat{y}) = 94.76$

Descenso de gradiente estocástico

- Se busca minimizar el ECM para el ejemplo actual

$$E = (y - \hat{y})^2 = \left(y - (w_1 . x_1 + w_2 . x_2 + b)\right)^2$$

- Debemos calcular el gradiente para saber cómo modificar los pesos

$$\frac{\partial E}{\partial w_1} = -2(y - \hat{y}) \frac{\partial (w_1 . x_1 + w_2 . x_2 + b)}{\partial w_1} = -2(y - \hat{y}) x_1$$

$$\frac{\partial E}{\partial w_2} = -2(y - \hat{y}) \frac{\partial (w_1 . x_1 + w_2 . x_2 + b)}{\partial w_2} = -2(y - \hat{y}) x_2$$

$$\frac{\partial E}{\partial b} = -2(y - \hat{y}) \frac{\partial (w_1 . x_1 + w_2 . x_2 + b)}{\partial b} = -2(y - \hat{y})$$

- La forma gral. es

$$\boldsymbol{\frac{\partial E}{w_i} = -2(y - \hat{y}) x_i}$$

Descenso de gradiente estocástico

- Se busca minimizar el ECM para el ejemplo actual

$$E = (y - \hat{y})^2 = \left(y - (w_1.x_1 + w_2.x_2 + b)\right)^2$$

- Debemos calcular el gradiente para saber cómo modificar los pesos

- Luego

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w_1}$$

$$w_1 = w_1 + 2\alpha.(y - \hat{y}).x_1 = -8.50 + 2\alpha * 94.76 * 14.96 = -8.47$$

Descenso de gradiente estocástico

- Se busca minimizar el ECM para el ejemplo actual

$$E = (y - \hat{y})^2 = \left(y - (w_1 . x_1 + w_2 . x_2 + b)\right)^2$$

- Debemos calcular el gradiente para saber cómo modificar los pesos

- Luego

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w_1} \qquad w_2 = w_2 - \alpha \frac{\partial E}{\partial w_2} \qquad b = b - \alpha \frac{\partial E}{\partial b}$$

$$w_1 = -8.47 \qquad w_2 = 7.76 \qquad b = 174.80$$

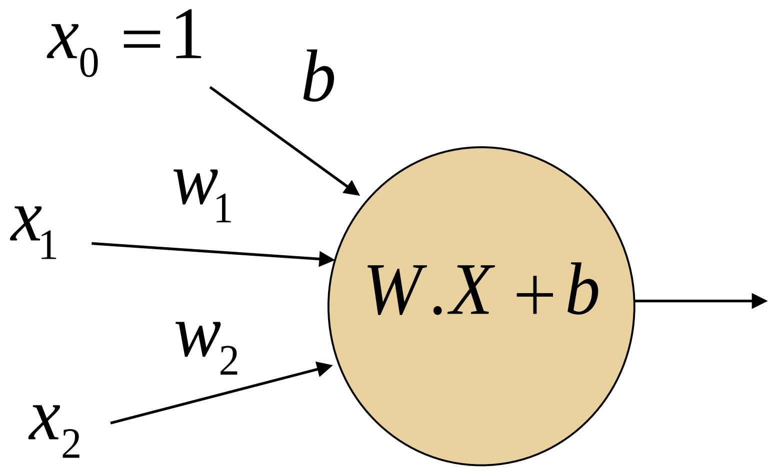
Predicción de la Producción de Energía (PE)

X

```
[[14.96, 41.76],
 [25.18, 62.96],
 [ 5.11, 39.4 ],
 ...,
 [31.32, 74.33],
 [24.48, 69.45],
 [21.6 , 62.52]]
```

Y

```
[463.26,
 444.37,
 488.56,
 ...,
 429.57,
 435.74,
 453.28]
```



$$\hat{y} = \sum_{i=1}^{2} w_i . x_i + b$$

$$\underbrace{(-8.50 \quad 7.68)}_{\text{W}} * \begin{pmatrix} 14.96 \\ 41.73 \end{pmatrix} + \underbrace{174.80}_{\text{b}}$$

$$\hat{y} = 368.49$$

$$y = 463.26$$

Error cometido en este ejemplo $\longrightarrow$ $Error = (y - \hat{y}) = 94.76$

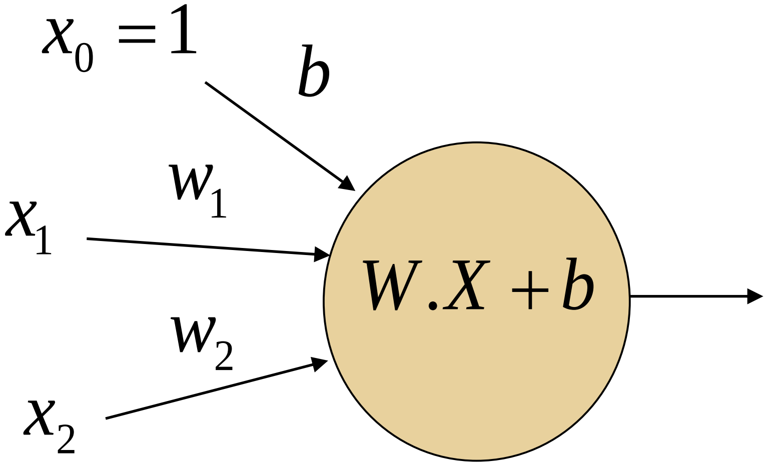
Predicción de la Producción de Energía (PE)

X

```
[[14.96, 41.76],
 [25.18, 62.96],
 [ 5.11, 39.4 ],
 ...,
 [31.32, 74.33],
 [24.48, 69.45],
 [21.6 , 62.52]]
```

Y

```
[463.26,
 444.37,
 488.56,
 ...,
 429.57,
 435.74,
 453.28]
```



$$\hat{y} = \sum_{i=1}^{2} w_i . x_i + b$$

$$\underbrace{(-8.47 \quad 7.76)}_{\mathbf{W}} * \begin{pmatrix} 14.96 \\ 41.73 \end{pmatrix} + \underbrace{174.80}_{\mathbf{b}}$$

$$\hat{y} = 372.23$$

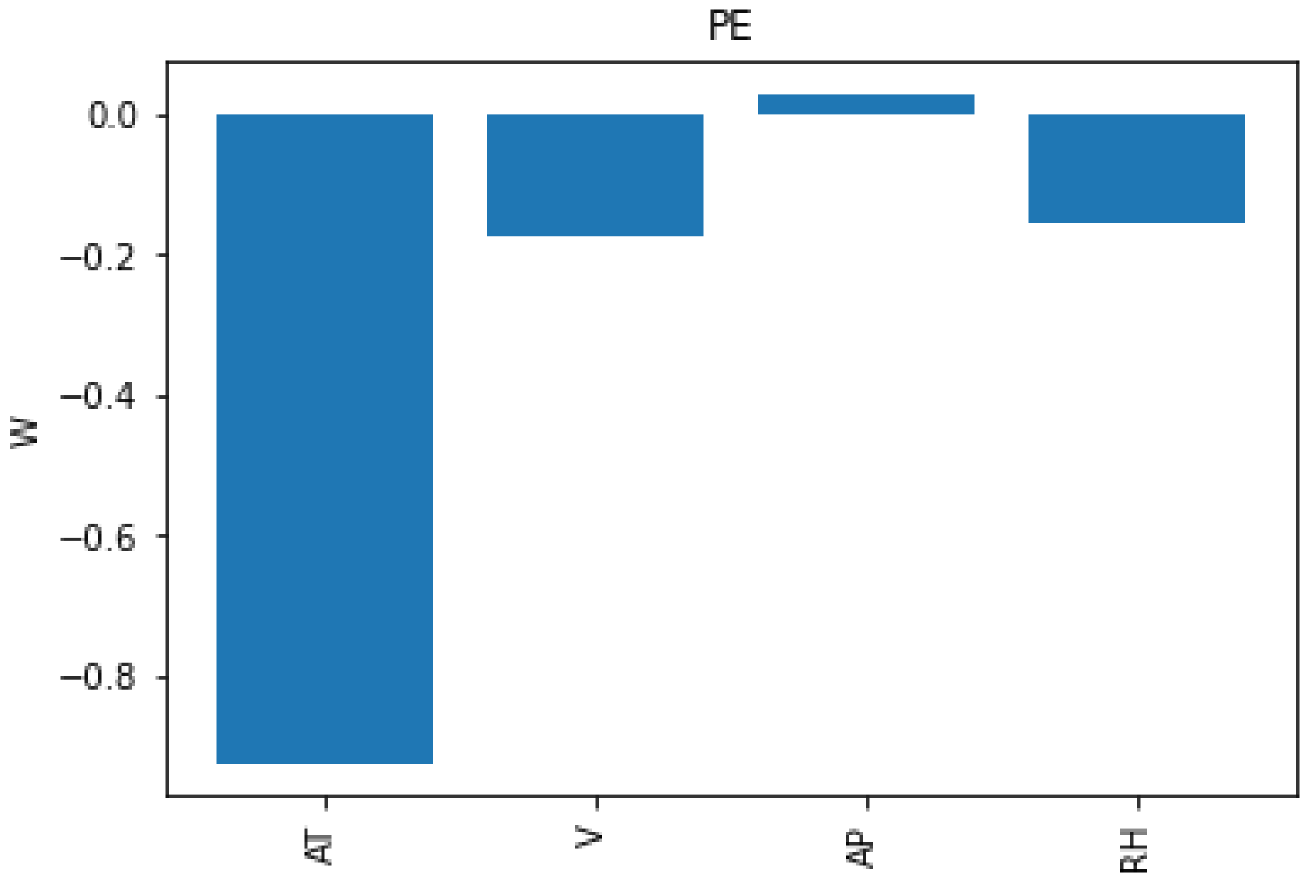
$$y = 463.26$$

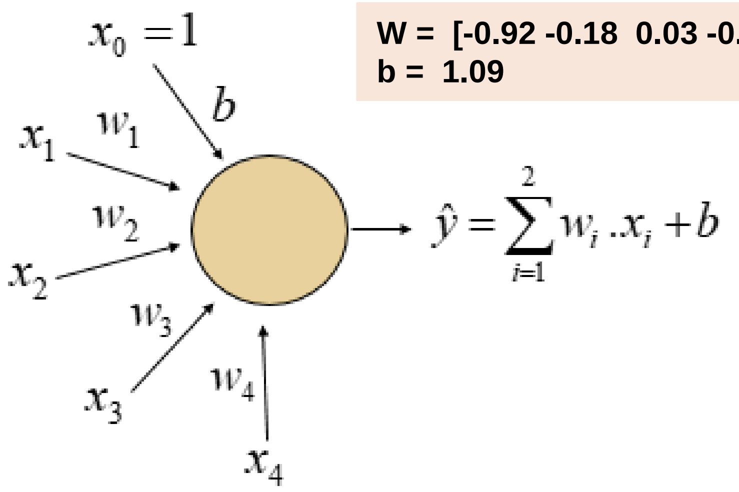
Error cometido en este ejemplo $\longrightarrow$ $Error = (y - \hat{y}) = 91.03$

Predicción de la Producción de Energía (PE)

(Atributos normalizados linealmente entre 0 y 1)

- PE = -0.92 * AT -0.18 * V + 0.03 * AP - 0.15 * RH + 1.09





W = [-0.92 -0.18 0.03 -0.15]
b = 1.09

Comb_LINEAL_CCPP_RN.ipynb

Comb_LINEAL_HOUSING_RN.ipynb

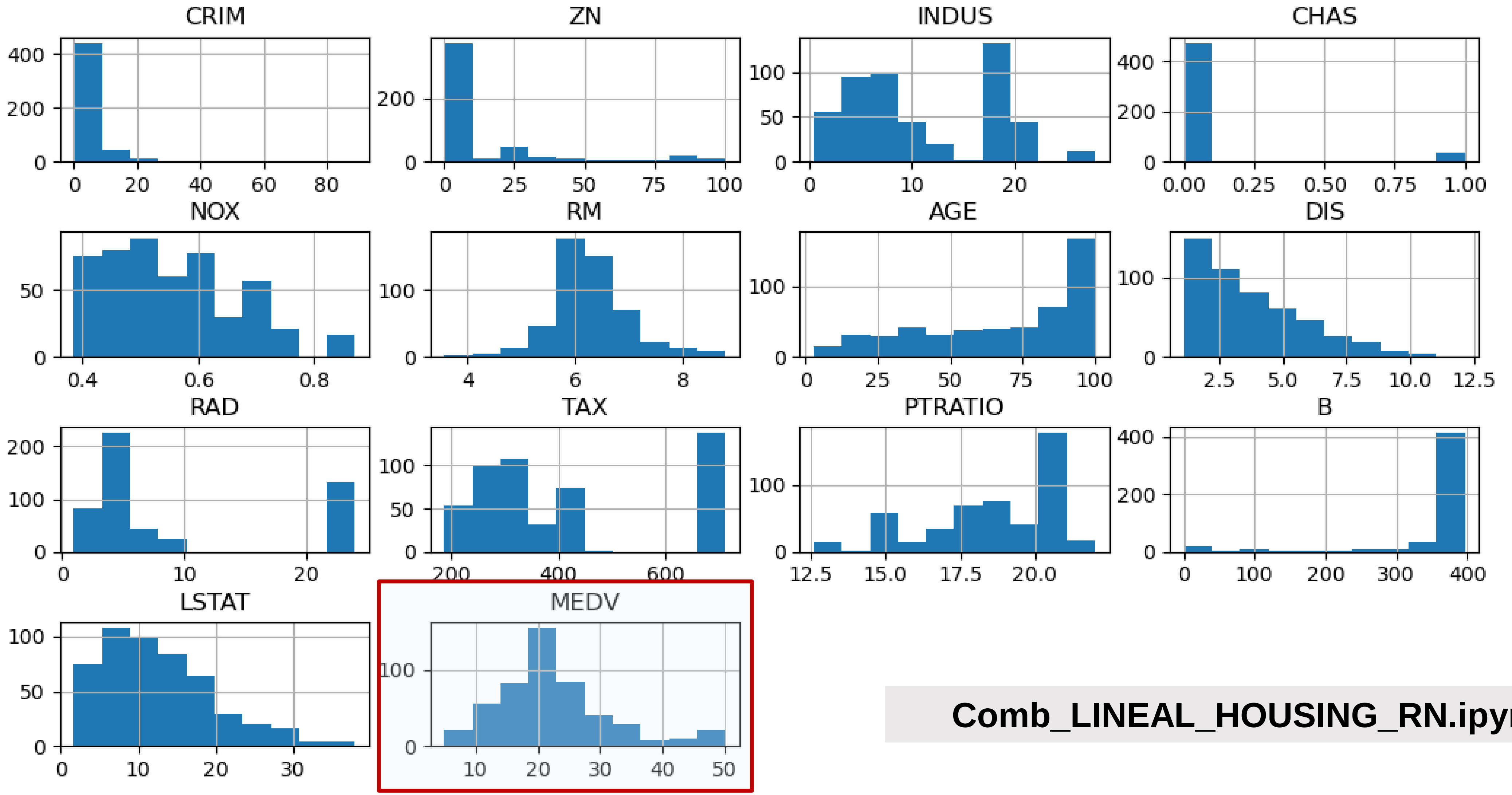
Boston Housing data set

- Contiene datos de las viviendas de 506 secciones censales de Boston del censo de 1970.

- Objetivo: Predecir el precio de una casa.
- Analizar los atributos antes de usarlos.
- Selección de atributos por correlación.

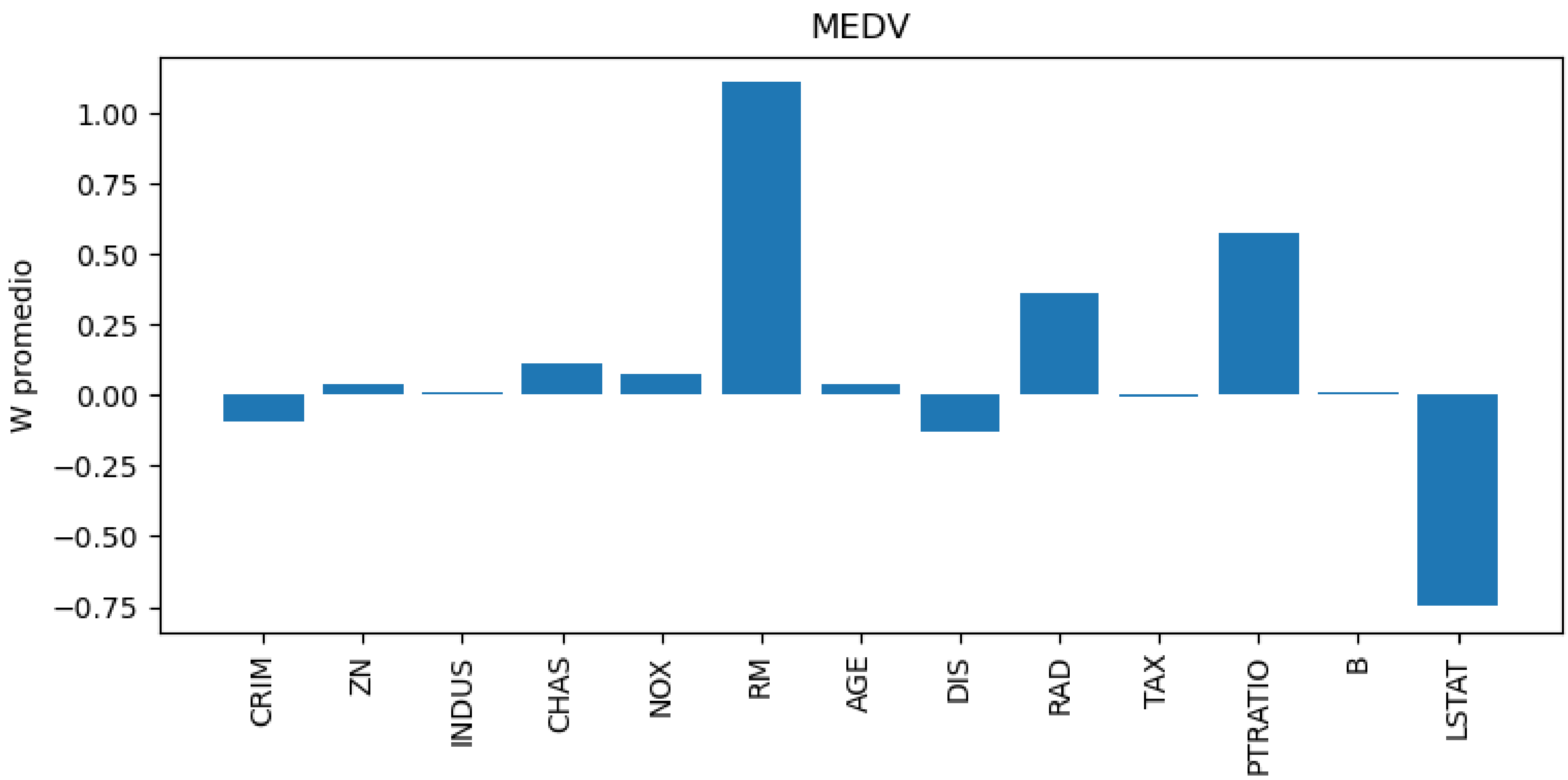
- CRIM - Crimen per cápita por ciudad
- ZN - proporción de terrenos residenciales
- INDUS - proporción de negocios no minoristas
- CHAS - 1 si el tramo limita el río, 0 si no
- NOX - concentración de óxidos nítricos
- RM – nro. promedio de habitaciones por vivienda
- AGE - proporción de unidades construidas antes de 1940
- DIS - Distancias promedio a centros de empleo
- RAD - accesibilidad a las autopistas radiales
- TAX - tasa de impuesto
- PTRATIO - colegios por localidad
- B - proporción de personas negras (año 1980!)
- LSTAT - porcentaje de personas de bajo estatus.
- MEDV - valor medio de las viviendas ocupadas por sus dueños

Histogramas de los atributos relevados

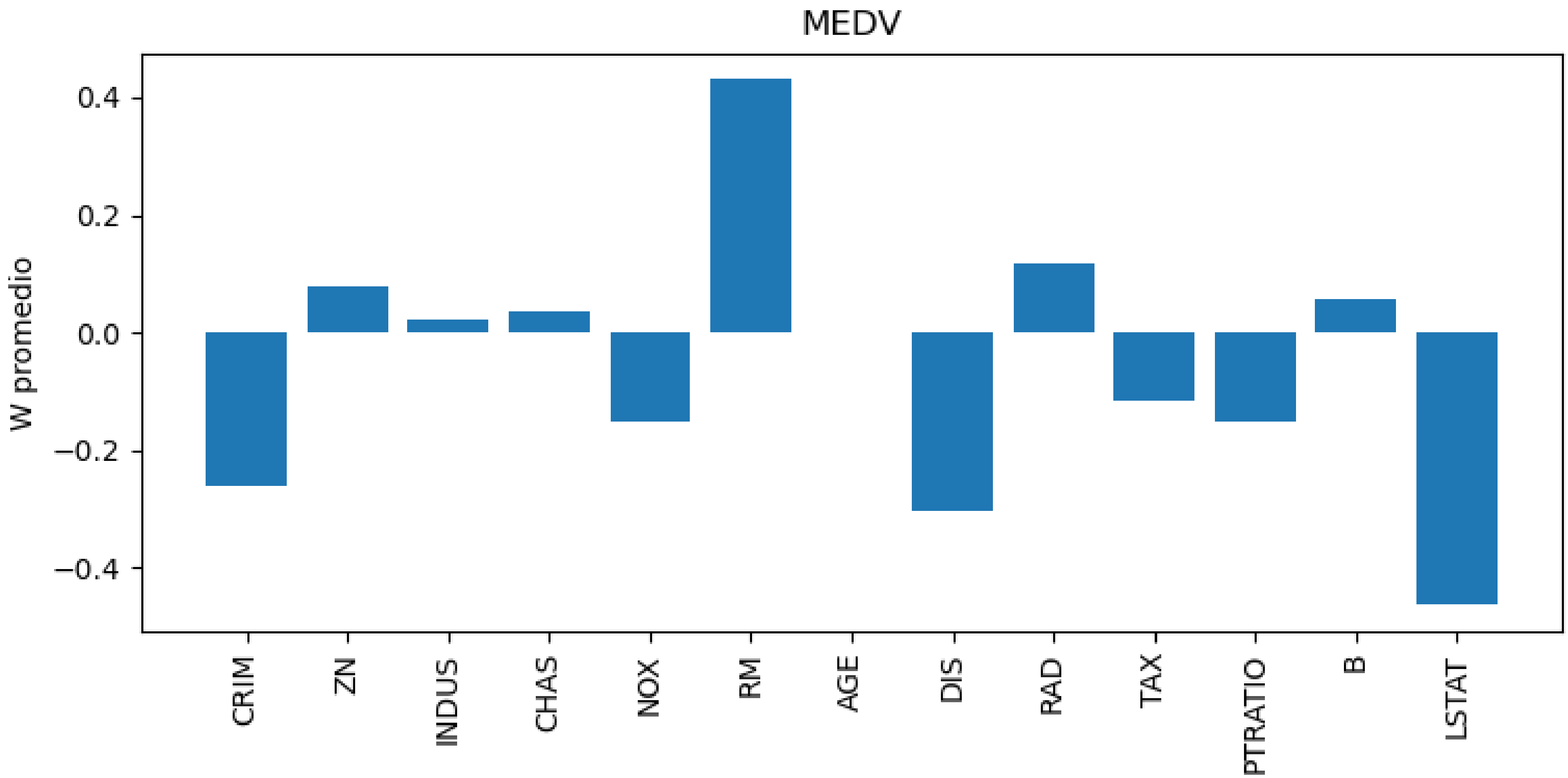


Comb_LINEAL_HOUSING_RN.ipynb

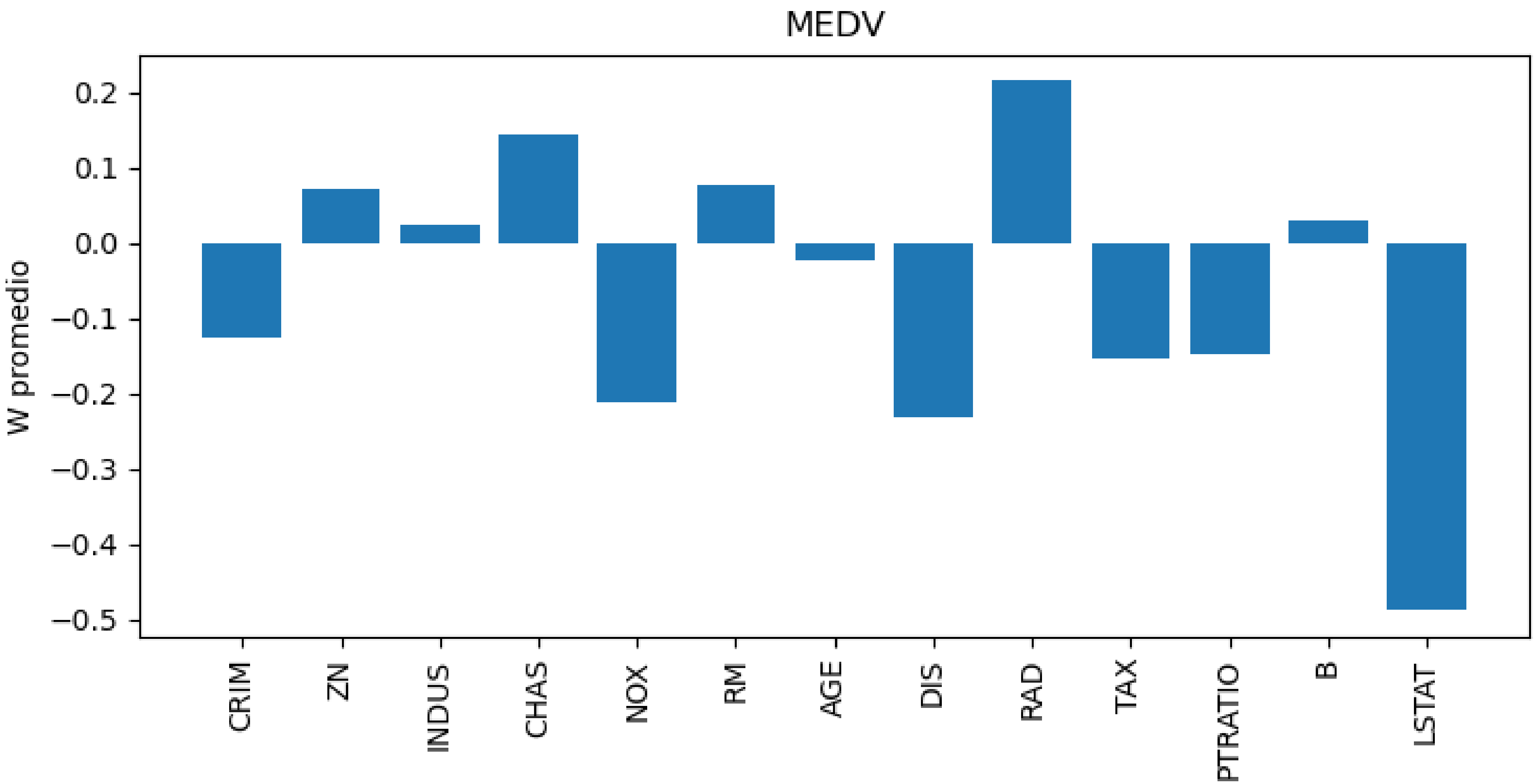
Vector W – sin normalizar los ejemplos



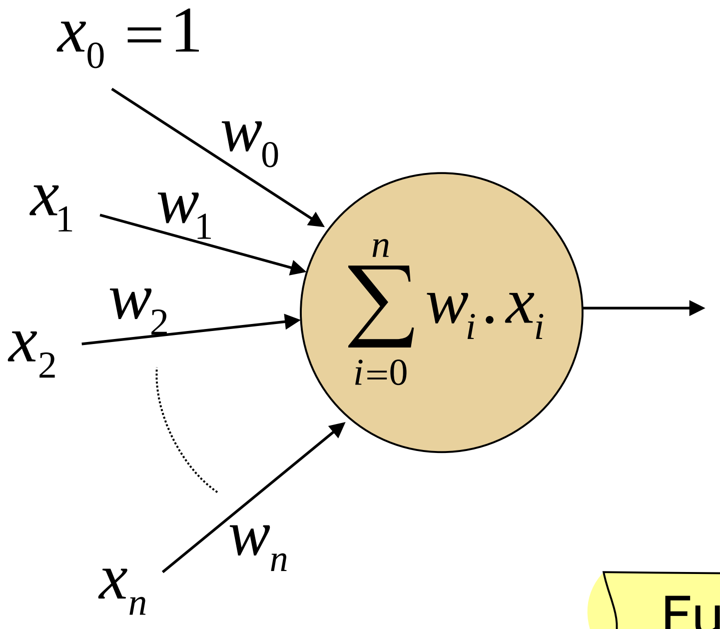
Vector W – ejemplos normalizados linealmente



Vector W – normalización con media y desvío



Neurona General



$$y_k = f\left(\sum_{i=0}^{n} w_i . x_i\right)$$

Función derivable

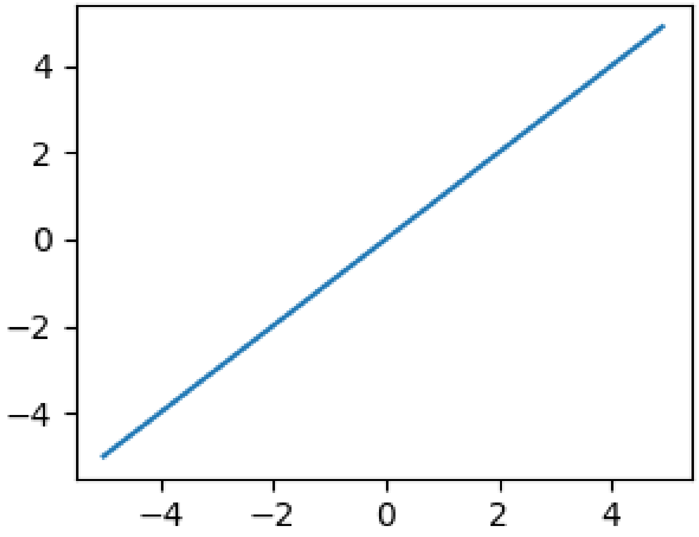
Función de Salida LINEAL

$$f(x) = x \qquad\qquad f'(x) = 1$$

desde Python

```
import numpy as np
import grafica as gr
from matplotlib import pyplot as plt

x = np.array(range(-50,50,1))/10.0
y = gr.evaluar('purelin', x)
plt.plot(x,y,'-')
```



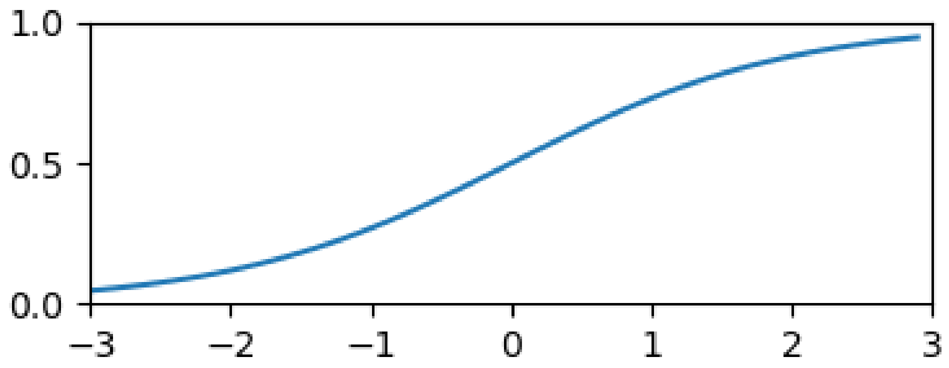
Función SIGMOIDE $\in (0,1)$

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

desde Python

```
import numpy as np
import grafica as gr
from matplotlib import pyplot as plt

x = np.array(range(-30,30,1))/10.0
y = gr.evaluar('logsig', x)
plt.plot(x,y,'-')
plt.axis([-3, 3, 0, 1])
plt.show()
```



Función SIGMOIDE ∈ (-1,1)

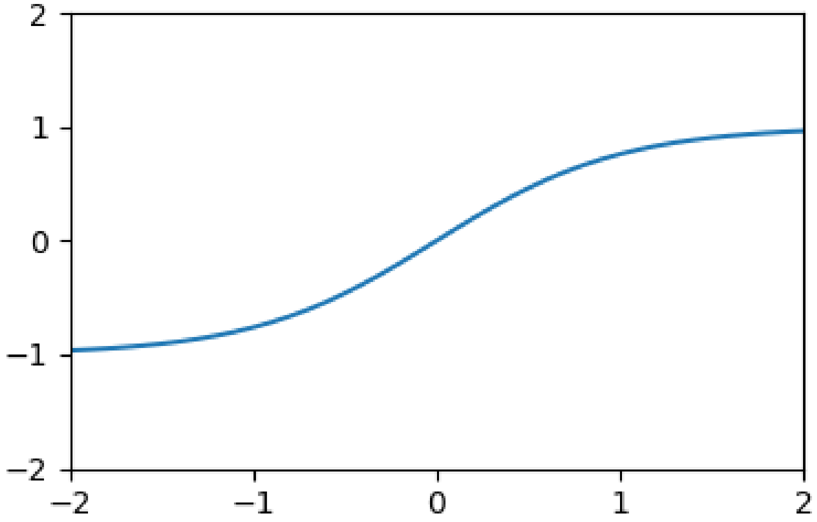
$$f(x) = \frac{2}{1+e^{-2x}} - 1$$

$$f'(x) = 1 - f(x)^2$$

desde Python

```
import numpy as np
import grafica as gr
from matplotlib import pyplot as plt

x = np.array(range(-30,30,1))/10.0
y = gr.evaluar('tansig', x)
plt.plot(x,y,'-')
plt.axis([-2, 2, -2, 2])
plt.show()
```



Ejemplo

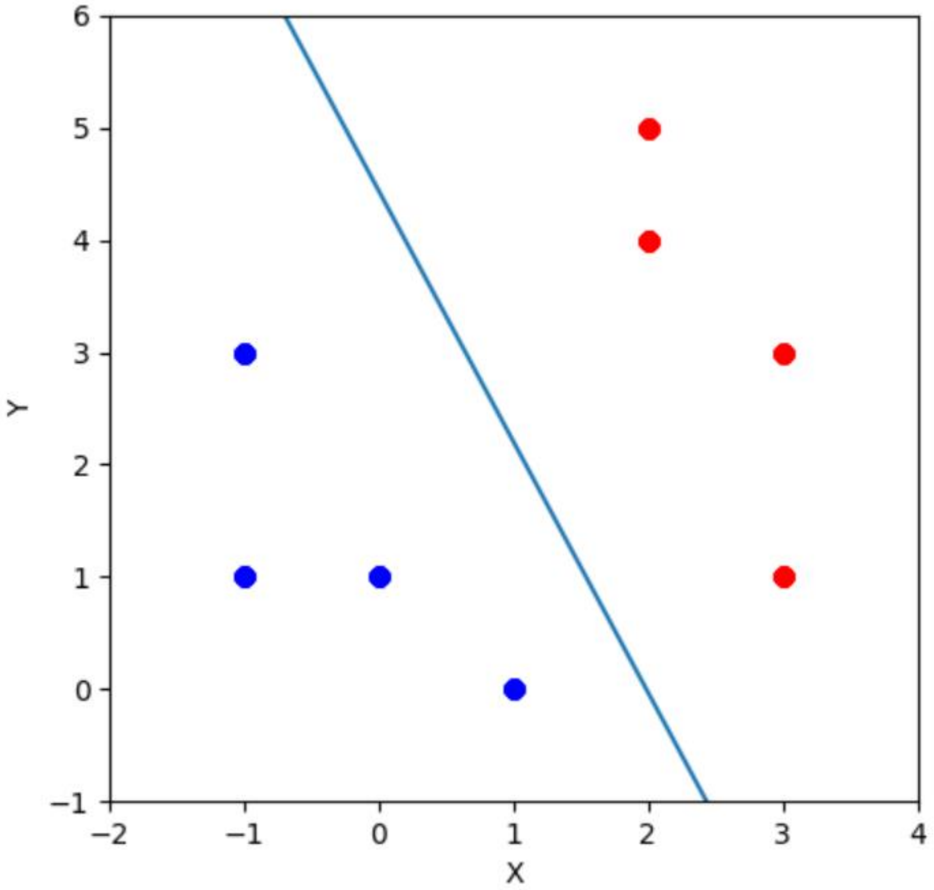
- Dados los siguientes conjuntos de puntos del plano

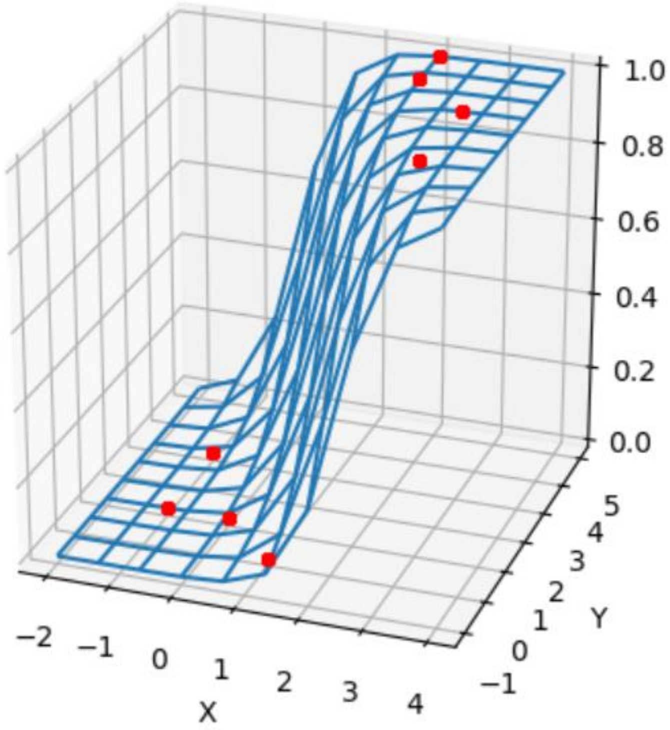
  A = {(2,2), (1,0), (0,1), (-1,1)}

  B = {(3,1), (3,3), (2,4), (2,5)}

  - Utilice una **neurona no lineal** para clasificarlos
  - Representar gráficamente la solución propuesta.

A = {(-1,3), (1,0), (0,1), (-1,1)}
B = {(3,1), (3,3), (2,4), (2,5)}





neuronaNoLineal.py

Entrenamiento de una neurona no lineal

- Seleccionar el valor de α
- Inicializar los pesos W y b con valores random
- Mientras (la variación del ECM sea mayor a la cota prefijada)
  - Para cada ejemplo
    - Ingresar el ejemplo a la red.
    - Calcular el error $\varepsilon = (y - \hat{y})$ y $\frac{\partial \varepsilon^2}{\partial w} = (y - \hat{y})^2$
    - Actualizar los pesos de la red

$$w_i = w_i - \alpha \frac{\partial \varepsilon^2}{\partial w_i}$$

¿Cómo sería la derivada del error si la neurona no es lineal?

$$\frac{\partial \varepsilon^2}{\partial w} = \left[\frac{\partial (y-\hat{y})^2}{\partial w_0}; \quad \ldots; \quad \frac{\partial (y-\hat{y})^2}{\partial w_n} \right]$$

$$\frac{\partial \varepsilon^2}{\partial w_j} = -2(y-\hat{y}) \frac{\partial f}{\partial (neta)} \frac{\partial (neta)}{\partial w_j}$$

f'

$$\frac{\partial (neta)}{\partial w_j} = \frac{\partial (\sum_{i=0}^{n} w_i x_i)}{\partial w_j} = x_j$$

$$\boxed{\frac{\partial \varepsilon^2}{\partial w_j} = -2(y-\hat{y}) f'(neta) x_j}$$

Entrenamiento de una neurona no lineal

- Seleccionar el valor de α
- Inicializar los pesos W y b con valores random
- Mientras (la variación del ECM sea mayor a la cota prefijada)
  - Para cada ejemplo
    - Ingresar el ejemplo a la red.
    - Calcular $\frac{\partial \varepsilon^2}{\partial w_i} = -2 * \varepsilon * f'(neta) * x_i$
    - Actualizar los pesos de la red

$$w_i = w_i - \alpha \frac{\partial \varepsilon}{\partial w_i} = w_i + 2\alpha * \varepsilon * f'(neta) * x_i$$

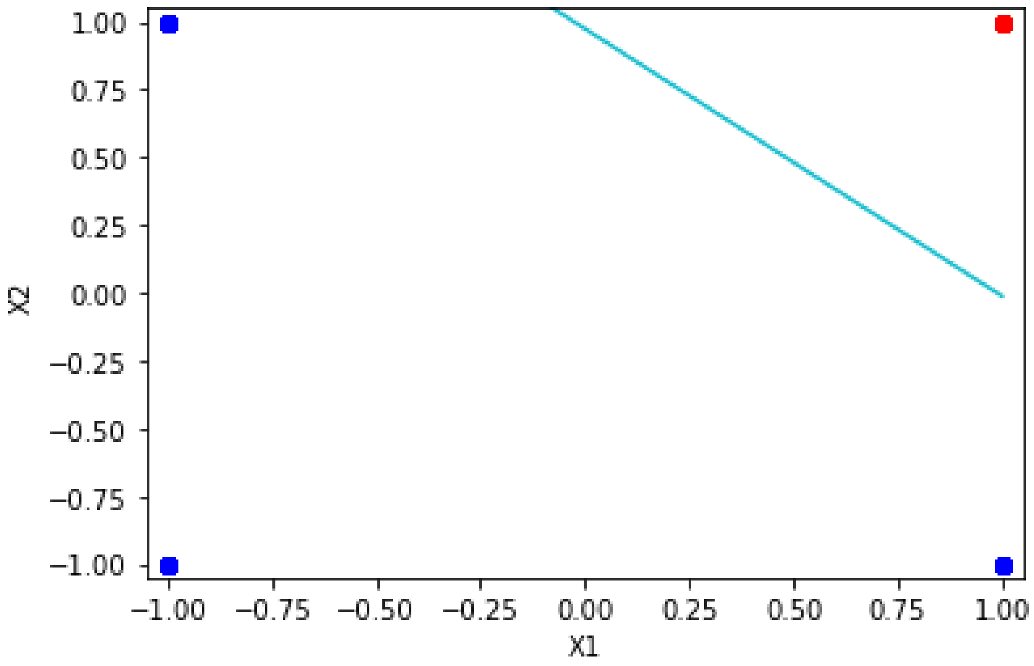
XOR

NeuronaGral_AND_enClase.ipynb

- Utilice una neurona no lineal con salida sigmoide para resolver el problema del XOR

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

$$f'^{(x)} = f(x) * (1 - f(x))$$



ClassNeuronaGral.py

nn = NeuronaGradiente(alpha=0.01, n_iter=50, cotaE=10E-07, FUN='sigmoid',

random_state=None, draw=0, title=['X1','X2'])

- Parámetros de entrada
  - **alpha**: valor en el intervalo (0, 1] que representa la velocidad de aprendizaje.
  - **n_iter**: máxima cantidad de iteraciones a realizar.
  - **cotaE**: termina si la diferencia entre dos errores consecutivos es menor que este valor.
  - **FUN**: función de activación – 'sigmoid', 'tanh', 'purelin'.
  - **random_state**: None si los pesos se inicializan en forma aleatoria, un valor entero para fijar la semilla
  - **draw**: valor distinto de 0 si se desea ver el gráfico y 0 si no. Sólo si es 2D.
  - **title**: lista con los nombres de los ejes para el gráfico. Se usa sólo si **draw** no es cero.

ClassNeuronaGral.py

nn = NeuronaLineal(alpha=0.01, n_iter=50)

nn.fit(X, T)

- Parámetros de entrada
  - **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.
  - **T** : arreglo de N elementos siendo N la cantidad de ejemplos
- Retorna
  - **w_** : arreglo de M elementos siendo M la cantidad de atributos de entrada
  - **b_** : valor numérico continuo correspondiente al bias.
  - **errors_**: errores cometidos en cada iteración.

ClassNeuronaGral.py

Y = nn.predict(X)

- Parámetros de entrada
  - **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.

- Retorna: un arreglo con el resultado de aplicar la neurona general entrenada previamente con fit() a la matriz de ejemplos X.
  - **Y** : arreglo de N elementos siendo N la cantidad de ejemplos

NeuronaGral_AND_RN.ipynb

```
import numpy as np
from ClassNeuronaGral import NeuronaGradiente

# Ejemplos de entrada de la función AND
X = np.array([[0,0], [0,1],[1,0],[1,1]])
X = 2*X-1
T = np.array([0,0,0,1])

ppn = NeuronaGradiente(alpha=0.1, n_iter=50, cotaE=10e-07, FUN='sigmoid’,
                       random_state=None, draw=1, title=['x1', 'x2'])
ppn.fit(X,T)

#-- % de aciertos ---
Y = (ppn.predict(X)>0.5)*1
print("Y = ", Y)
print("T = ", T)
aciertos = sum(Y == T)
print("aciertos = %d   (%.2f%%)" % (aciertos, 100*aciertos/X.shape[0]))
```

Ejemplo

- Sobre una cinta transportadora circulan naranjas y melones. Se busca obtener un clasificador de frutas que facilite su almacenamiento. Para cada fruta se conoce su diámetro, en centímetros y su intensidad de color naranja, medida entre 0 y 255.

- Utilice la información del archivo **FrutasTrain.csv** para entrenar una **neurona no lineal** capaz de reconocer los dos tipos de fruta.

- Compare la manera de obtener la función discriminante de la neurona no lineal con respecto al perceptrón.
  - **NeuronaGral_FRUTAS_RN.ipynb**
  - **Perceptron_FRUTAS_RN.ipynb**

Funciones de costo

- Error cuadrático medio

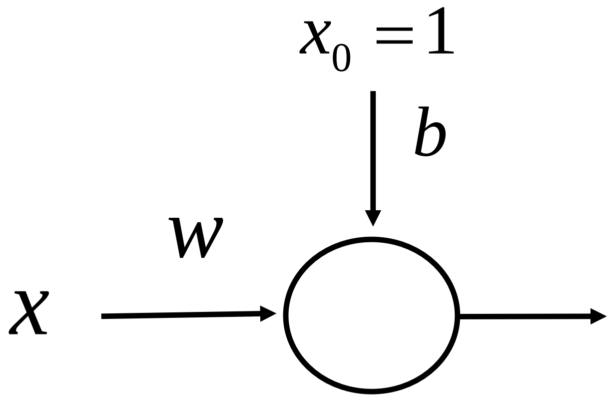
$$C = \frac{1}{n}\sum_{n}(y - \hat{y})^2 = \frac{1}{n}\sum_{n}(y - f(neta))^2$$

- Entropía cruzada binaria

$$C = -\frac{1}{n}\sum_{n}[y \ln \hat{y} + (1 - y)\ln(1 - \hat{y})]$$

Ejemplo

- Entrene una neurona no lineal con función de salida sigmoide entre 0 y 1 utilizando como función de costo el error cuadrático medio para que reciba un 1 y responda 0



$$y = \frac{1}{1 + e^{-(wx+b)}}$$

$x = 1$ (entrada)
$y = 0$ (salida esperada)

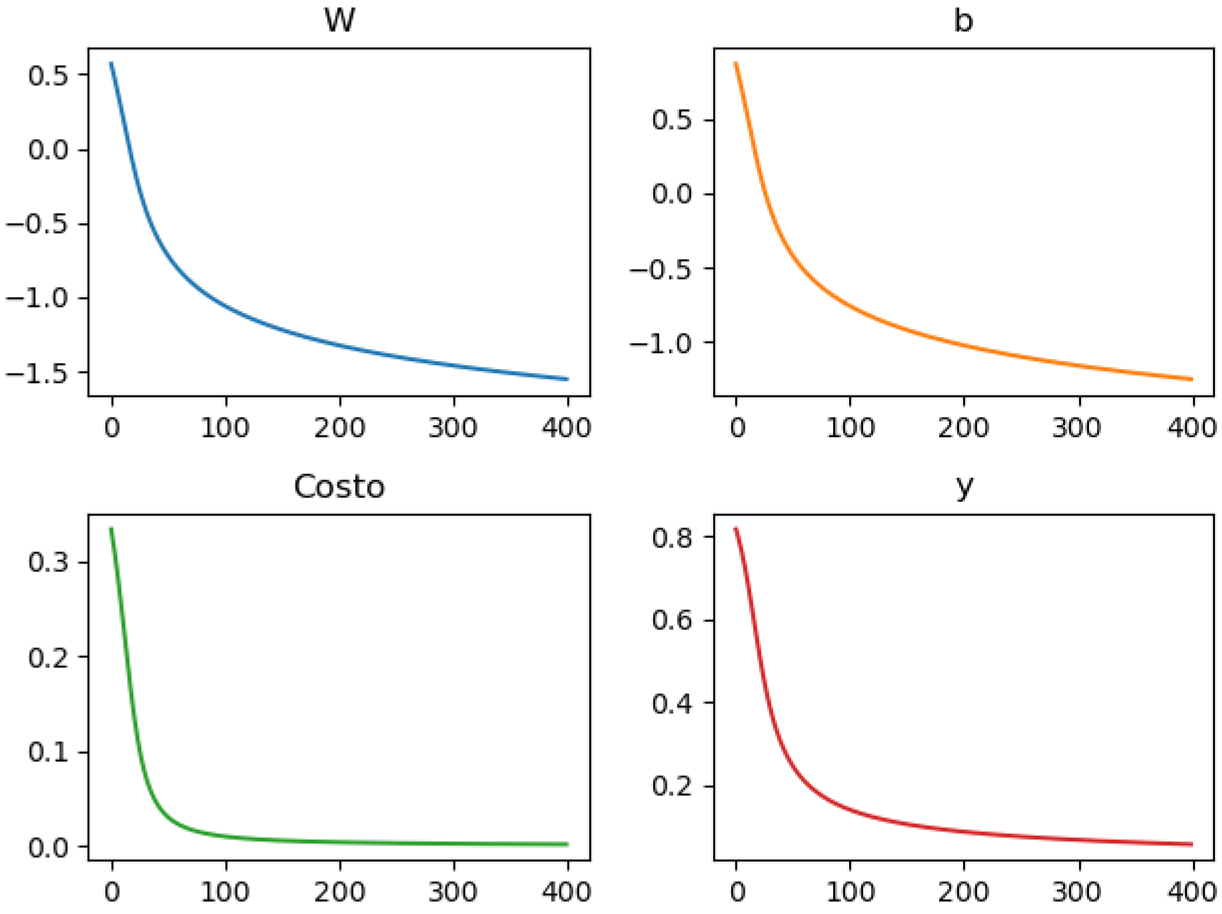
Función de costo

$$C = \frac{(y - \hat{y})^2}{2}$$

Ejemplo

NeuronaNoLineal_1Ej.ipynb

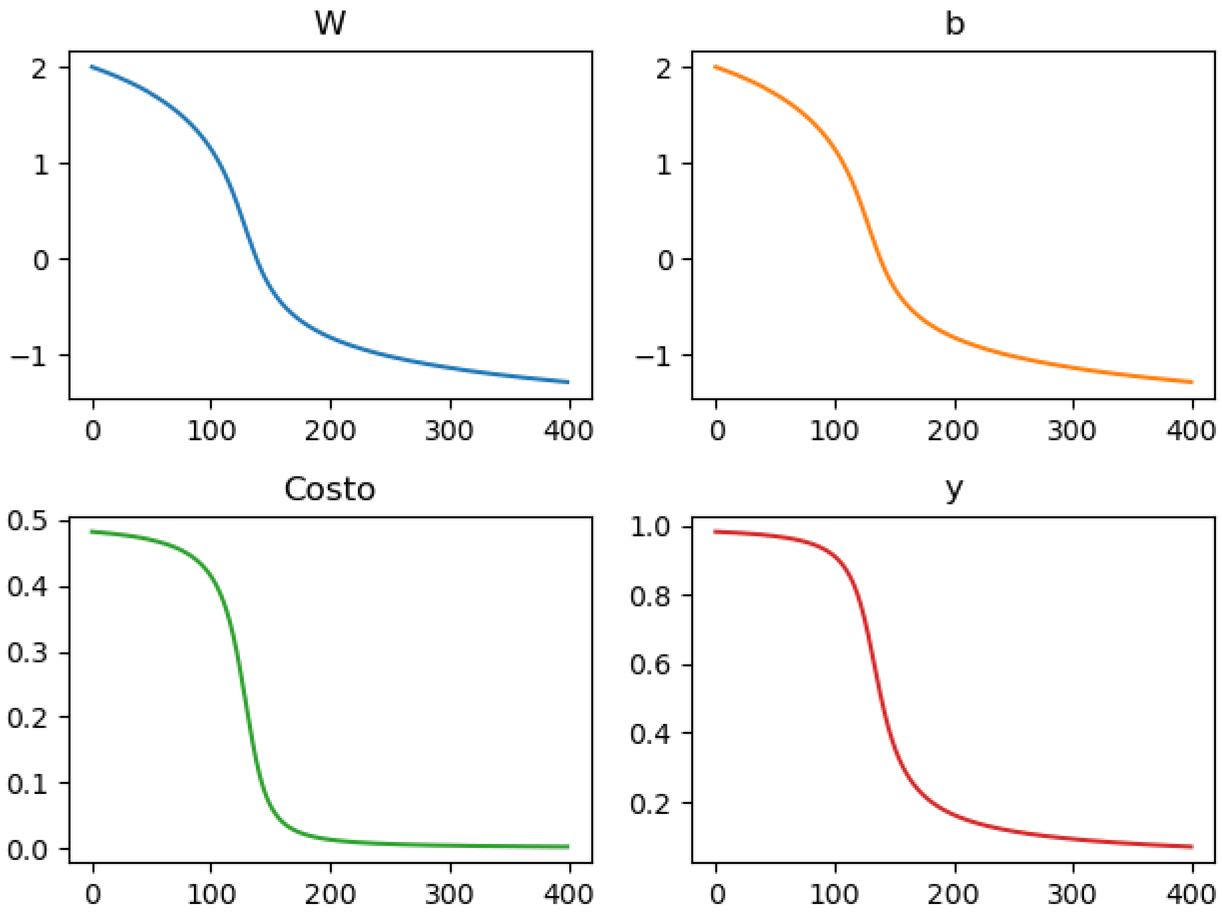
- alfa=0.25 e inicio en W=0.6 y b=0.9



Ejemplo

NeuronaNoLineal_1Ej.ipynb

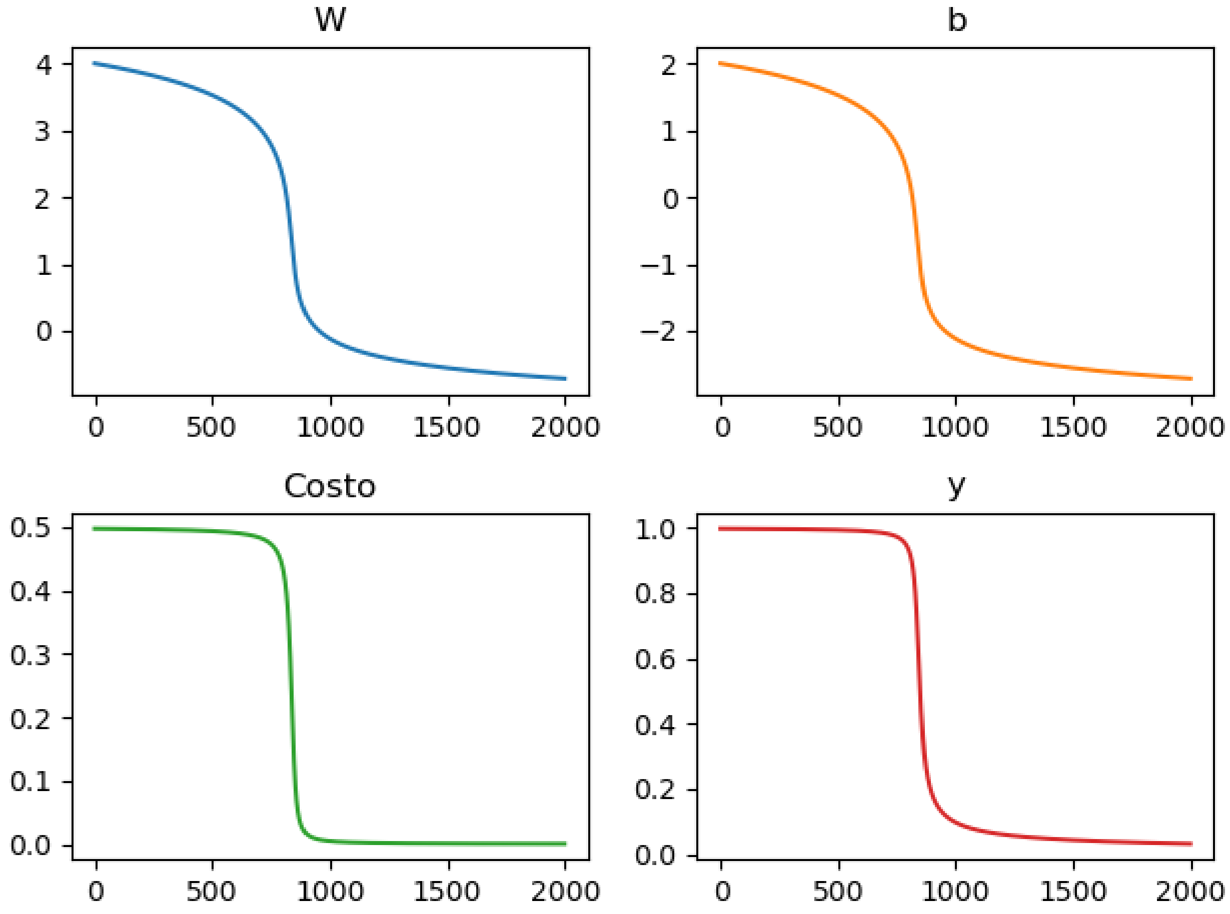
- alfa=0.25 e inicio en W=2 y b=2



Ejemplo

NeuronaNoLineal_1Ej.ipynb

- alfa=0.25 e inicio en W=4 y b=2



Entropía cruzada binaria

- Es una función de costo que puede usarse con neuronas no lineales con función sigmoide entre 0 y 1

$$C = -\frac{1}{n}\sum_{n}[\ y \ln \hat{y} + (1-y)\ \ln(1-\hat{y})\ ]$$

donde

- y es el valor binario esperado
- $\hat{y} = 1/(1 + e^{-\sum x_i w_i})$ es la salida de la neurona

- Ver que es una función de costo
  - C > 0
  - C tiende a 0 (cero) a medida que la neurona aprende la salida deseada.

Derivada de la entropía cruzada binaria

$$C = -\frac{1}{n}\sum_{n}[y \ln \hat{y} + (1-y)\ln(1-\hat{y})]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_{n}\left(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right)\frac{\partial\, \hat{y}}{\partial w_j}$$

$f(neta)$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_{n}\left(\frac{y}{f(neta)} - \frac{1-y}{1-f(neta)}\right)\frac{\partial f(neta)}{\partial w_j}$$

Derivada de la entropía cruzada binaria

$$C = -\frac{1}{n}\sum_{n}[y \ln \hat{y} + (1-y)\ \ln(1-\hat{y})]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_{n}\left(\frac{y}{f(neta)} - \frac{1-y}{1-f(neta)}\right)\frac{\partial f(neta)}{\partial w_j}$$

$$\frac{y-f(neta)}{f(neta)(1-f(neta))}$$

$$f'(neta)x_j$$

Derivada de la entropía cruzada binaria

$$C = -\frac{1}{n}\sum_{n}[y \ln \hat{y} + (1-y)\ln(1-\hat{y})]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_{n}\left(\frac{y - f(neta)}{f(neta)\big(1 - f(neta)\big)}\right) f'(neta)\, x_j$$

Si $f(neta) = \frac{1}{1+e^{-neta}}$, $f'(neta) = f(neta)\big(1 - f(neta)\big)$

Derivada de la entropía cruzada binaria

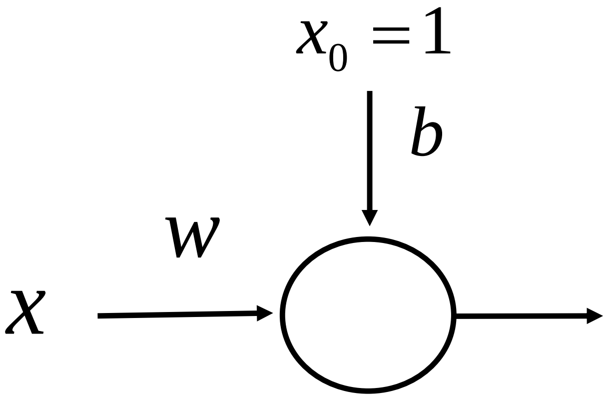
$$C = -\frac{1}{n}\sum_{n}[y \ln \hat{y} + (1-y) \ln(1-\hat{y})]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_{n}\left(\frac{y - f(neta)}{f(neta)\big(1 - f(neta)\big)}\right) f'(neta) x_j$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_{n}\big(y - f(neta)\big)\ x_j = -\frac{1}{n}\sum_{n}(y - \hat{y})\ x_j$$

Ejemplo

- Entrene una neurona no lineal con función de salida sigmoide entre 0 y 1 utilizando como función de costo el error cuadrático medio para que reciba un 1 y responda 0



$$y = \frac{1}{1+e^{-(wx+b)}}$$

$x = 1$ (entrada)

$y = 0$ (salida esperada)

Función de costo

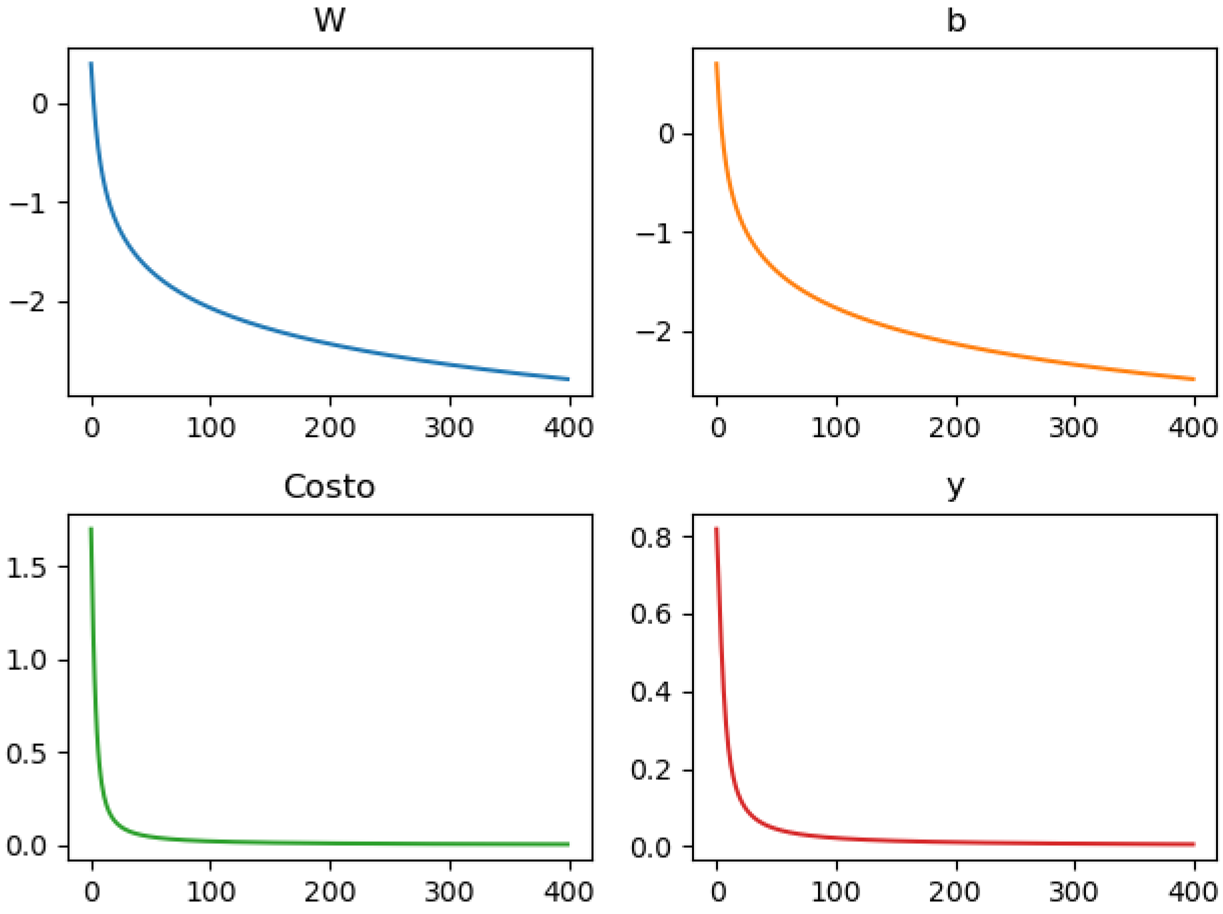
$$C = -(y \ln \hat{y} + (1-y) \ln(1-\hat{y}))$$

$$\frac{\partial C}{\partial w} = -(y-\hat{y})\; x$$

Ejemplo

NeuronaNoLineal_1Ej_EC.ipynb

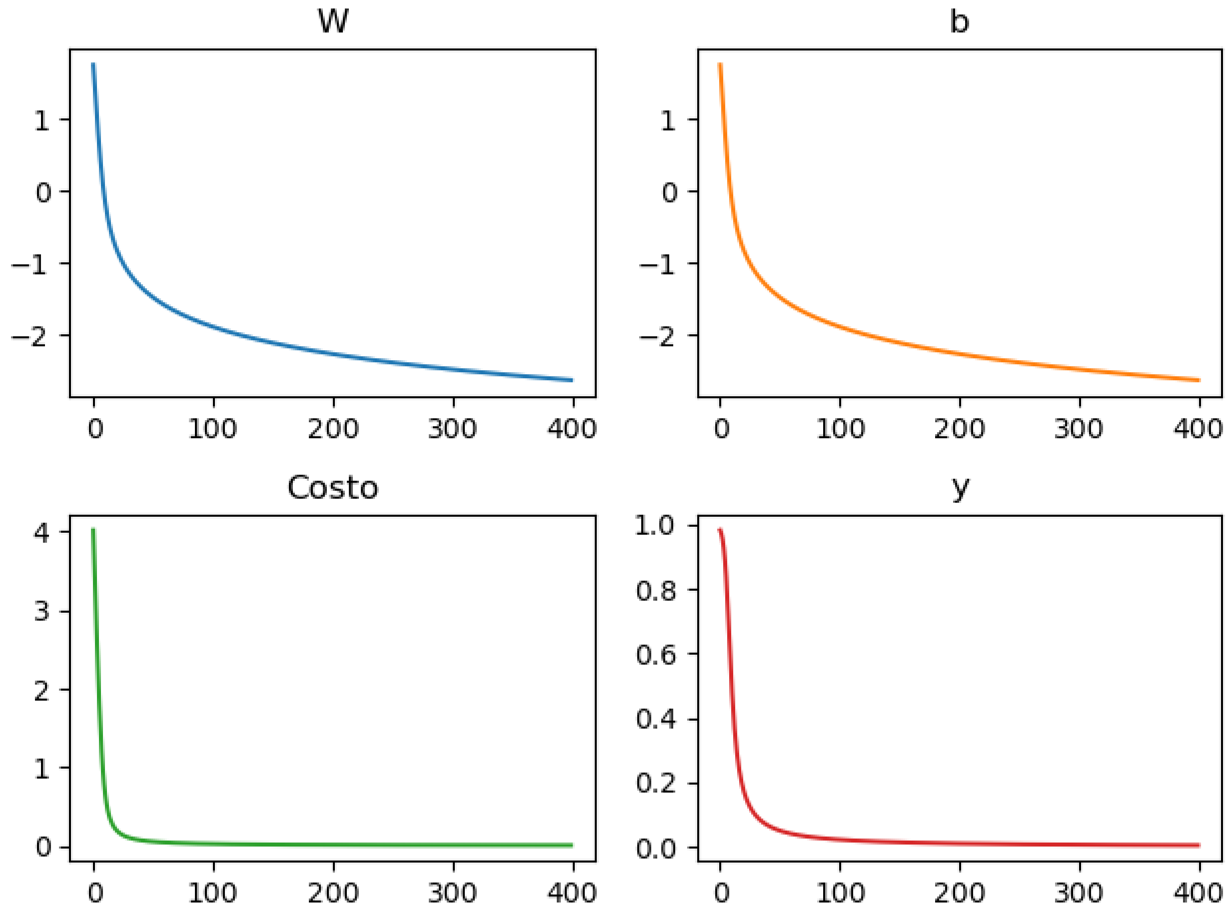
- alfa=0.25 e inicio en W=0.6 y b=0.9



Ejemplo

NeuronaNoLineal_1Ej_EC.ipynb

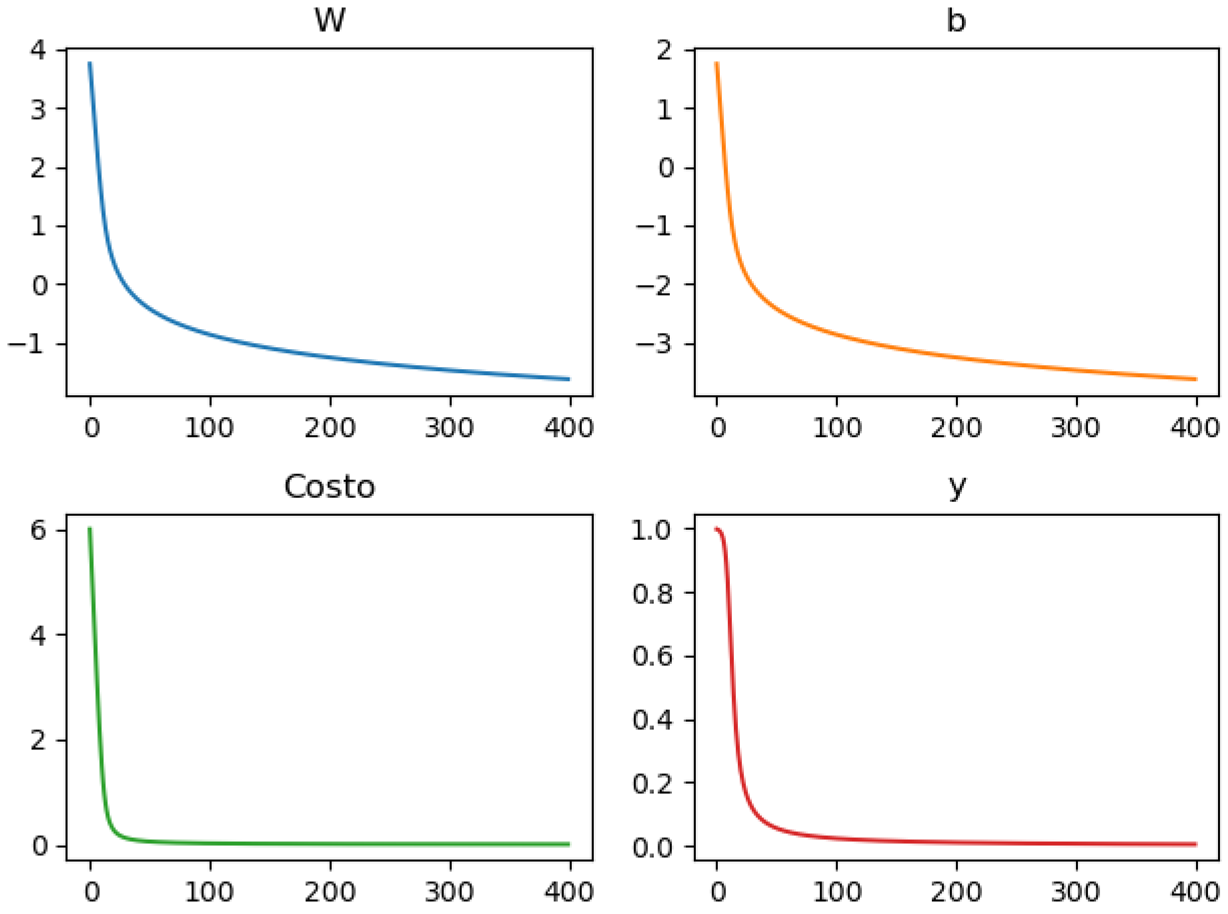
- alfa=0.25 e inicio en W=2 y b=2



Ejemplo

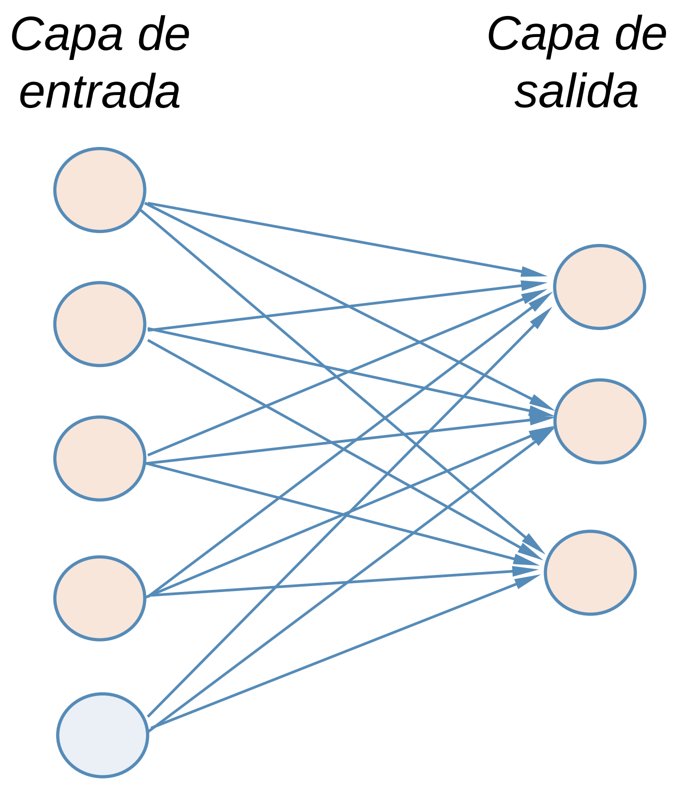
NeuronaNoLineal_1Ej_EC.ipynb

- alfa=0.25 e inicio en W=4 y b=2



Clasificación con más de 2 clases

- Pueden utilizarse varias neuronas no lineales para resolver un problema de clasificación con más de 2 clases.
- Cada neurona de la capa de salida buscará responder por un valor de clase distinto.
- El error de la capa será la suma de los errores de las neuronas que la forman.



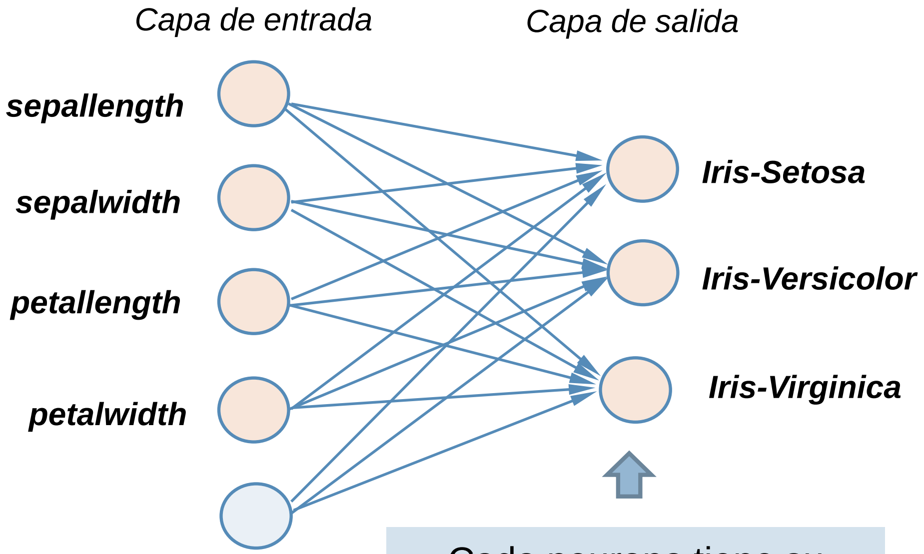
Clasificación de flores de Iris

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]]
```



Cada neurona tiene su propio vector de pesos. Luego **W es una matriz** y **b es un vector**

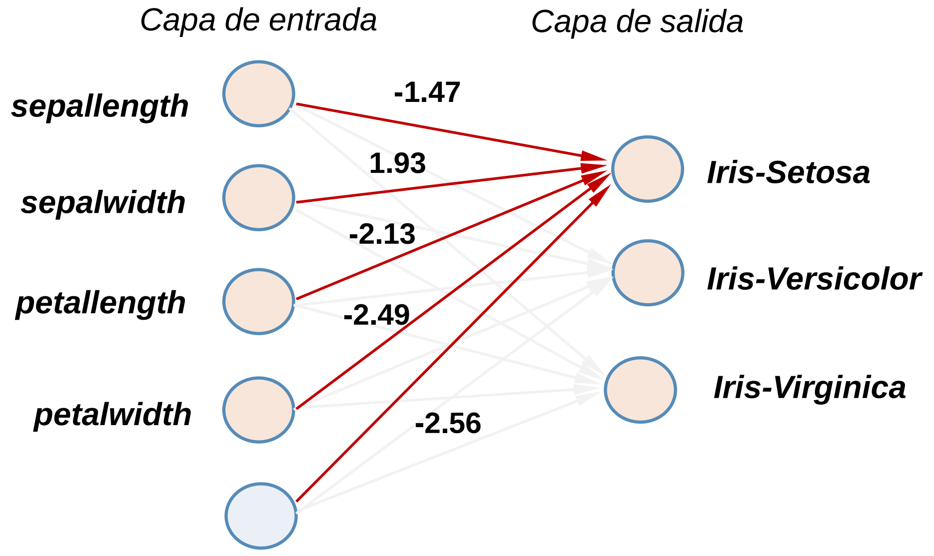
Clasificación de flores de Iris

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]]
```



Para obtener el resultado de la red debe calcularse

$$f\left(W * x^{T} + b\right)$$

siendo f la función de activación

```
[[-1.47, 1.93,-2.13,-2.49],
 [ 0.79,-1.38, 3.36,-3.57],
 [-1.57,-0.99, 6.17, 3.92]]
```

W

```
[[-2.56],
 [-0.35],
 [-7.03]]
```

b

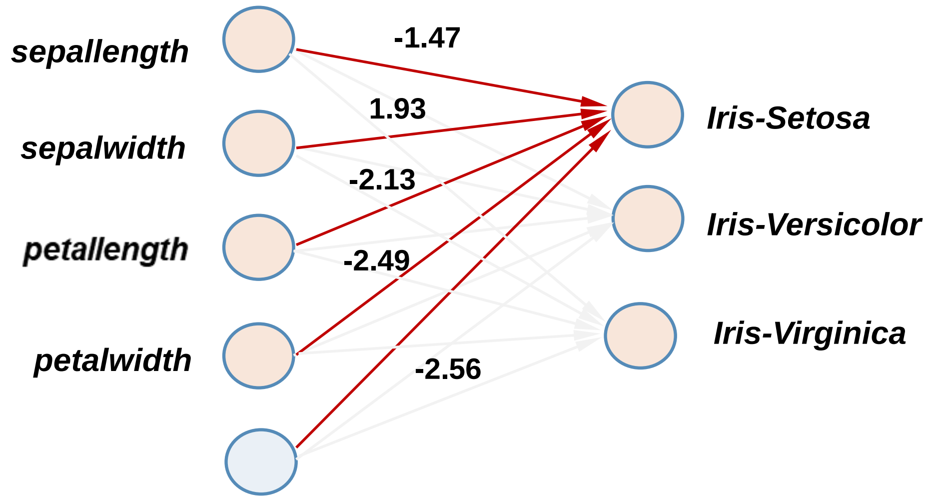
Clasificación de flores de Iris

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]]
```



W

```
[[-1.47, 1.93,-2.13,-2.49],
 [ 0.79,-1.38, 3.36,-3.57],
 [-1.57,-0.99, 6.17, 3.92]]
```

*

x^T

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

+

b

```
[[-2.56],
 [-0.35],
 [-7.03]]
```

=

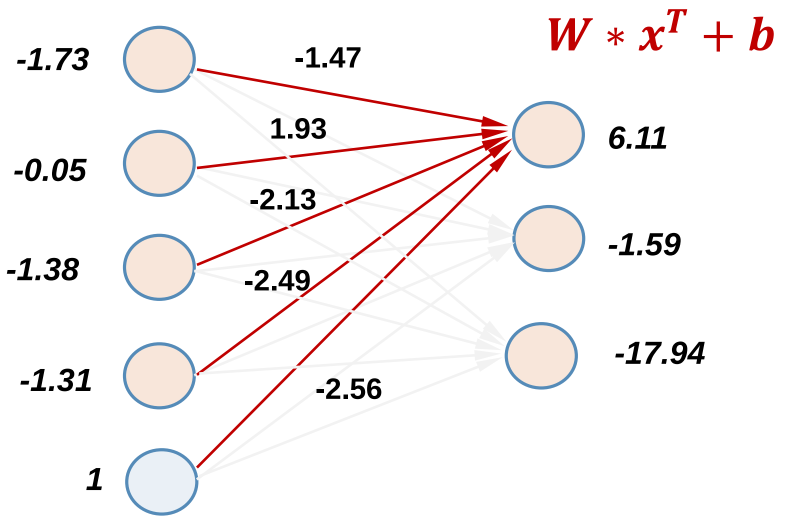
Clasificación de flores de Iris

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]]
```



W * x^T + **b** = $W * x^T + b$

```
[[-1.47, 1.93,-2.13,-2.49],
 [ 0.79,-1.38, 3.36,-3.57],
 [-1.57,-0.99, 6.17, 3.92]]
```

*

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

\+

```
[[-2.56],
 [-0.35],
 [-7.03]]
```

=

```
[[  6.11],
 [ -1.59],
 [-17.94]]
```

Clasificación de flores de Iris

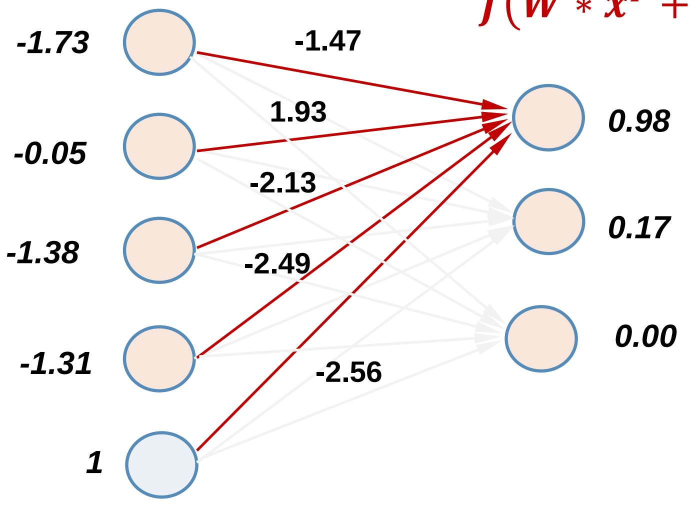
X

```
[[-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]]
```

$f(W * x^T + b)$



W

```
[[-1.47, 1.93,-2.13,-2.49],
 [ 0.79,-1.38, 3.36,-3.57],
 [-1.57,-0.99, 6.17, 3.92]]
```

*

x^T

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

\+

b

```
[[-2.56],
 [-0.35],
 [-7.03]]
```

=

$W * x^T + b$

```
[[  6.11],
 [ -1.59],
 [-17.94]]
```

$f(W * x^T + b)$ =

```
[[ 1/(1+exp( -6.11))],
 [ 1/(1+exp( -1.59))],
 [ 1/(1+exp(-17.94))]]
```

=

```
[[ 0.98],
 [ 0.17],
 [ 0.00]]
```

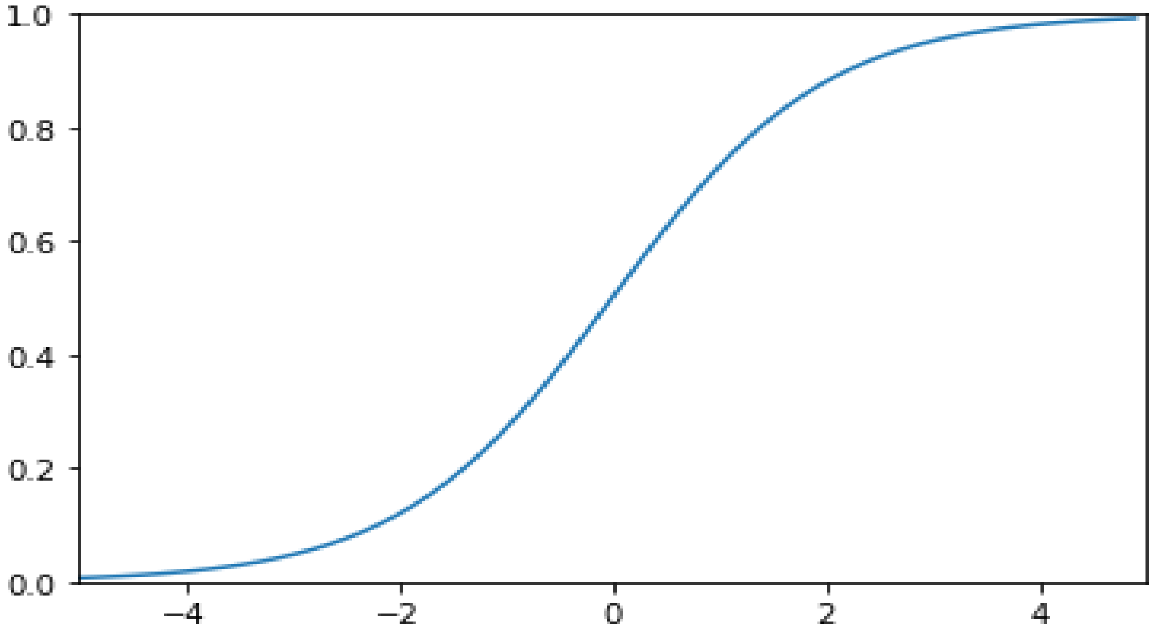
Se interpreta como

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

Función sigmoid()

X	$f(X)$
-5.00	0.01
-4.00	0.02
-3.00	0.05
-2.00	0.12
-1.39	0.20
-1.00	0.27
0.00	0.50
1.00	0.73
1.39	0.80
2.00	0.88
3.00	0.95
4.00	0.98
5.00	0.99

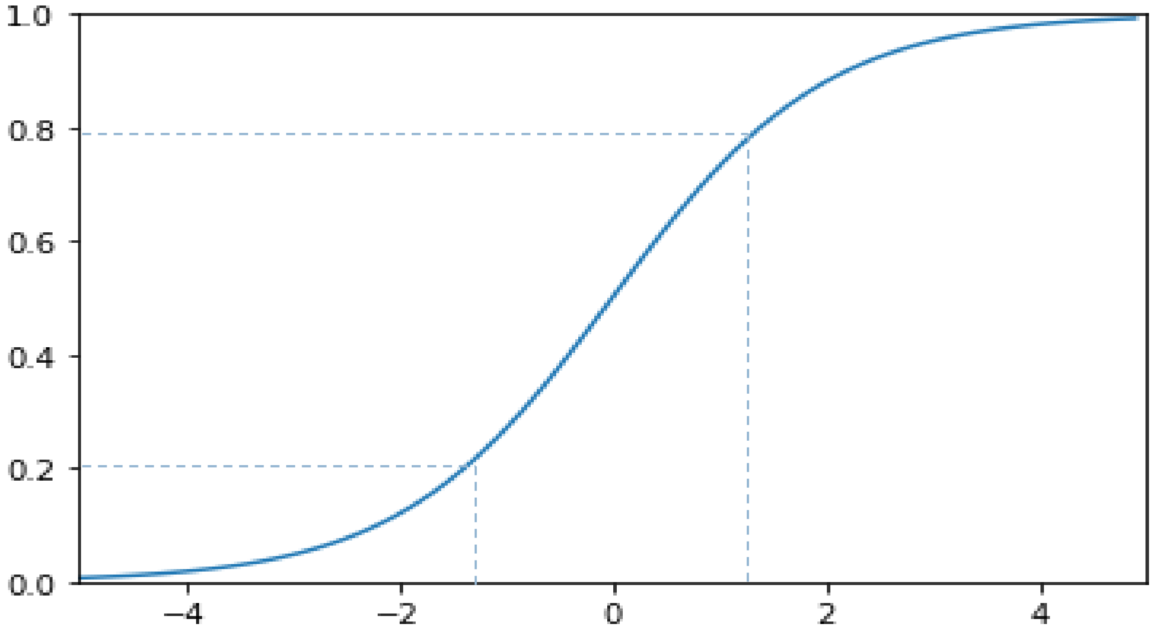
$$f(x) = \frac{1}{1 + \exp(-x)}$$



Función sigmoid()

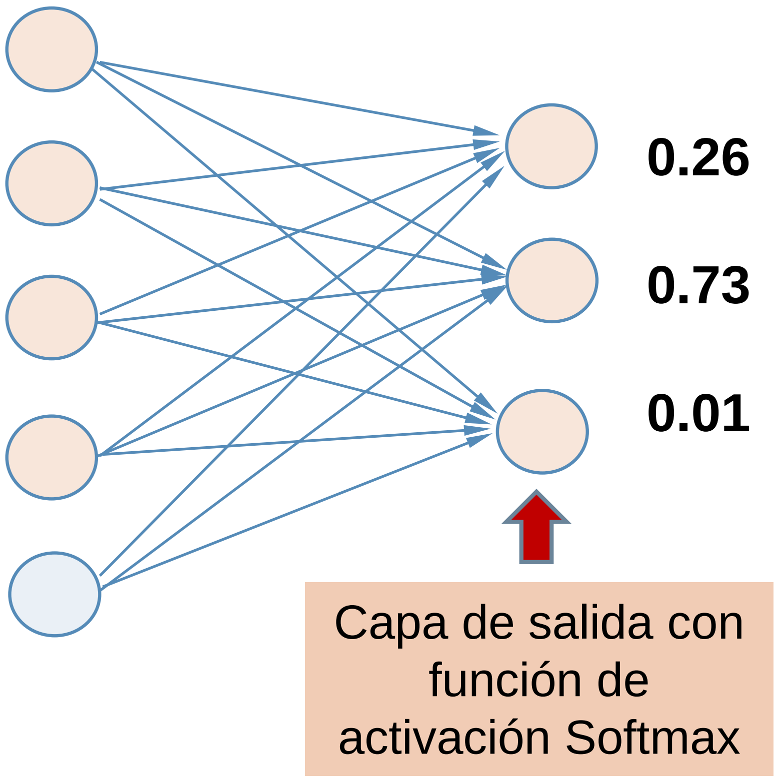
X	$f(X)$
-5.00	0.01
-4.00	0.02
-3.00	0.05
-2.00	0.12
-1.39	**0.20**
-1.00	0.27
0.00	0.50
1.00	0.73
1.39	**0.80**
2.00	0.88
3.00	0.95
4.00	0.98
5.00	0.99

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



Función Softmax

- Se utiliza como función de activación en la última capa para normalizar la salida de la red a una distribución de probabilidad.

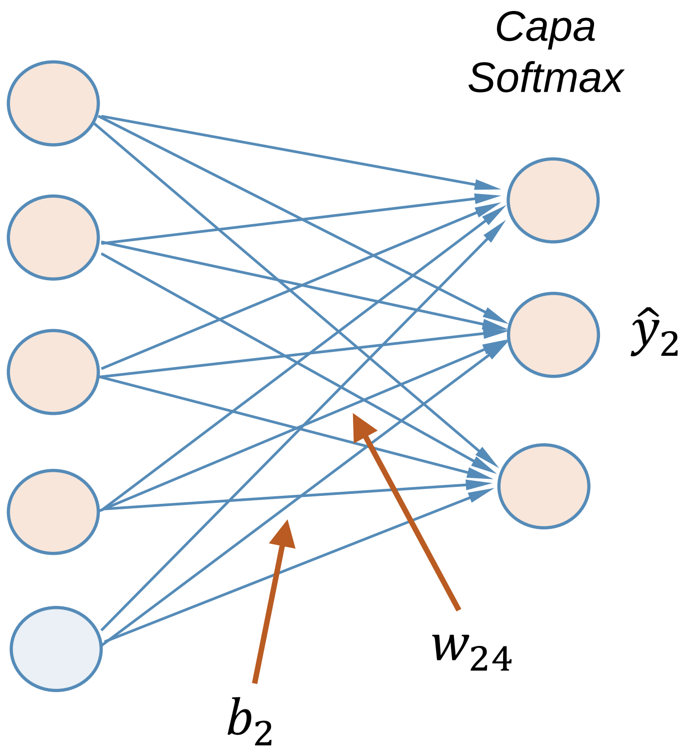


Capa softmax

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

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

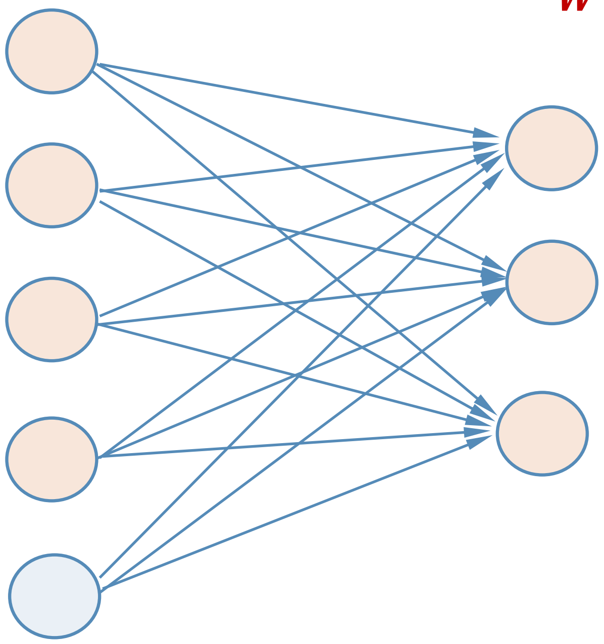
- La salida de la capa es una distribución de probabilidad
  - $\hat{y}_j > 0 \quad j = 1..k$
  - $\sum_j \hat{y}_j = 1$



Ver que el incremento en algún $\hat{y}_j$ producirá disminuciones en el resto

Función Softmax

- Ejemplo



$W * x^T + b$

5

4

2

Función sigmoide →

$f(W * x^T + b)$

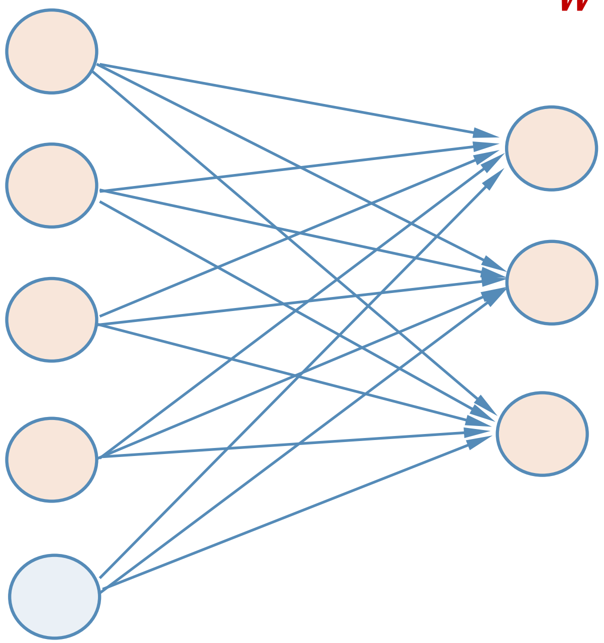
0.99

0.98

0.88

Función Softmax

- Ejemplo



$W * x^T + b$

5

4

2

Función softmax

$f(W * x^T + b)$

0.71

0.26

0.04

Función softmax

Entrada

Propagación hacia adelante →

Salidas de la propagación hacia adelante

cat	dog	horse
5	4	2
4	2	8
4	4	1

Función sigmoide →

Salidas de la capa con función sigmoide

cat	dog	horse
0.99	0.98	0.88
0.98	0.88	0.99
0.98	0.98	0.73

Función softmax

Neta	sigmoid	exp(neta)	softmax
5	0,99	148,41	0,71
4	0,98	54,60	0,26
2	0,88	7,39	0,04

Entrada

Propagación hacia adelante →

Salidas de la propagación hacia adelante

cat	dog	horse
5	4	2
4	2	8
4	4	1

Función Softmax →

Salidas de la capa Softmax

cat	dog	horse
0.71	0.26	0.04
0.02	0.00	0.98
0.49	0.49	0.02

Capa Softmax

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

- **Función de costo: Negative Log-Likelihood (NLL)**

$$C = -\sum_k y_k \ln \hat{y}_k$$

donde $\boldsymbol{y}$ es un vector binario que vale 1 sólo en la posición correspondiente al valor de clase esperado. Luego

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

s es la neurona correspondiente al valor de clase esperado

Capa Softmax

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

- **Función de costo: Negative Log-Likelihood (NLL)**

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

y es la neurona correspondiente al valor de clase esperado

- Derivada de la función NLL

$$\frac{\partial C}{\partial w_{jk}} = -\left(y_j - \hat{y}_j\right) x_j \qquad \frac{\partial C}{\partial b_j} = -\left(y_j - \hat{y}_j\right)$$

Coincide con la derivada de la entropía cruzada binaria

Capa softmax

RNMulticlase_IRIS_RN.ipynb

- **Función de costo: Negative Log-Likelihood (NLL)**

Entrada	Salida Softmax: cat	dog	horse	Loss, L(a): NLL
	**0.71**	0.26	0.04	0.34
	0.02	0.00	**0.98**	0.02
	0.49	**0.49**	0.02	0.71

$-\log(\hat{y}_s)$ *en la clase correcta* →

La clase correcta está pintada de rojo

Total = 1.07

- Sólo se evalúa en la neurona correspondiente a la salida esperada
- Cuando más cerca está de 1 menor será el error.
- A menor valor de la neurona softmax correspondiente a la clase correcta, mayor error.

ClassRNMulticlase.py

rn = RNMulticlase(alpha=0.01, n_iter=50, cotaE=10e-07, FUN='sigmoid', COSTO='ECM', random_state=None)

Parámetros de entrada

- **alpha**: valor en el intervalo (0, 1] que representa la velocidad de aprendizaje.
- **n_iter**: máxima cantidad de iteraciones a realizar.
- **cotaE**: termina si la diferencia entre dos errores consecutivos es menor que este valor.
- **FUN:** función de activación – 'sigmoid', 'tanh', 'softmax'.
- **COSTO**: función de costo – 'ECM', 'EC_binaria', 'EC'
- **random_state**: None si los pesos se inicializan en forma aleatoria, un valor entero para fijar la semilla

ClassRNMulticlase.py

rn = RNMulticlase(alpha=0.01, n_iter=50, cotaE=10e-07, FUN='sigmoid', COSTO='ECM'

rn.fit(X, T)

- Parámetros de entrada
  - **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.
  - **T** : arreglo de N elementos siendo N la cantidad de ejemplos
- Retorna
  - **w_** : arreglo de M elementos siendo M la cantidad de atributos de entrada
  - **b_** : valor numérico continuo correspondiente al bias.
  - **errors_**: errores cometidos en cada iteración.
  - **accuracy_**: precisión de la clasificación en cada iteración

ClassRNMulticlase.py

Y = nn.predict(X)

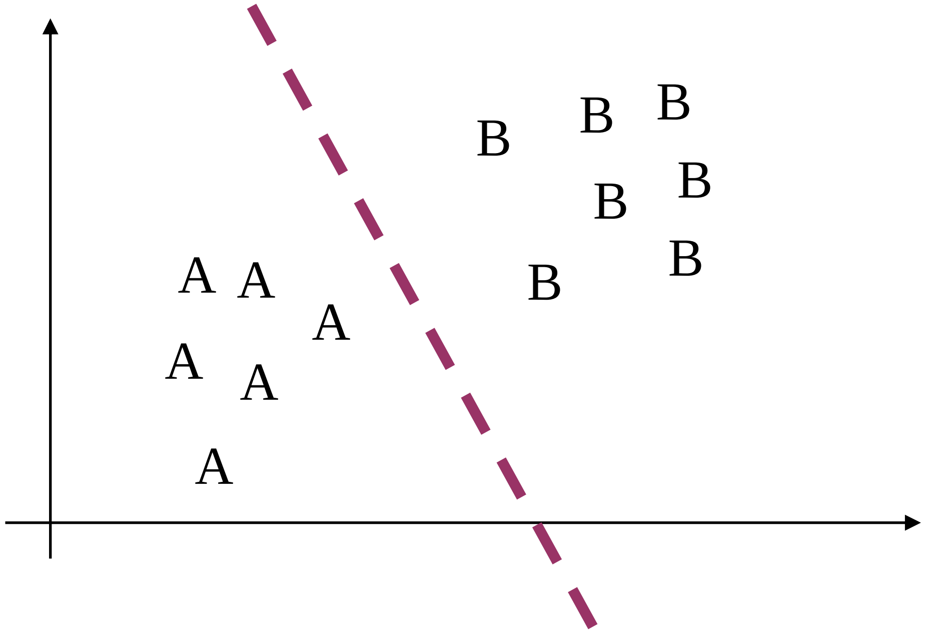
- Parámetros de entrada
  - **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.

- Retorna: un arreglo con el resultado de aplicar la neurona general entrenada previamente con fit() a la matriz de ejemplos X.
  - **Y** : vector de N elementos siendo N la cantidad de ejemplos con el número de clase.

Ver el siguiente código

RNMulticlase_IRIS_RN.ipynb

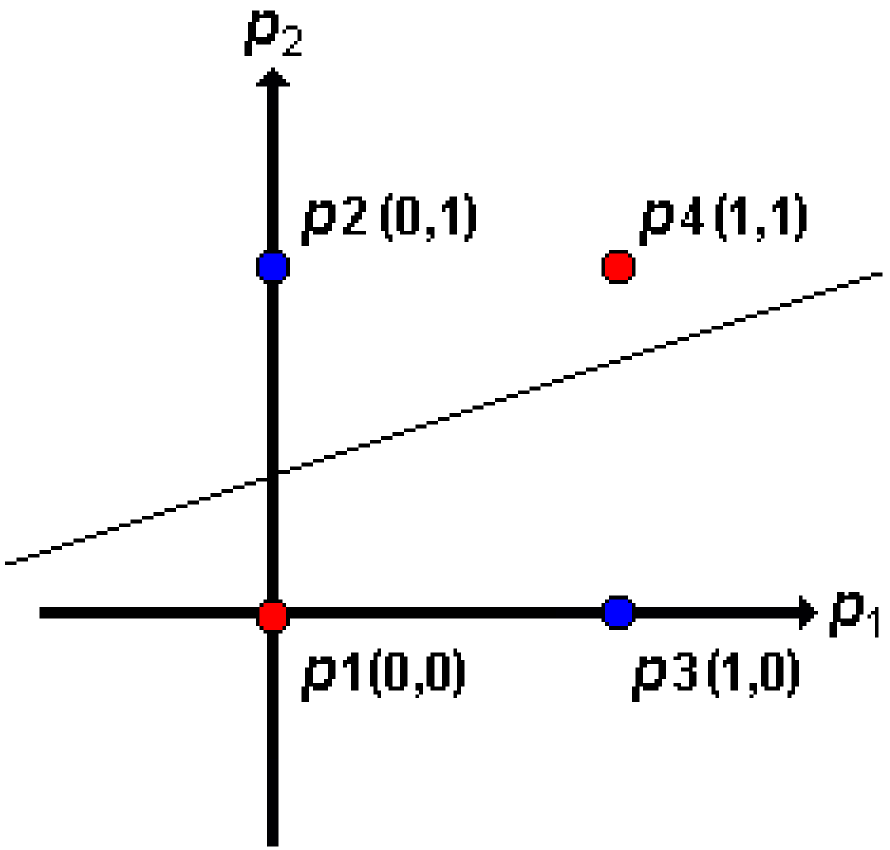
¿Puede utilizarse una neurona lineal para clasificar dos conjuntos de ejemplos?



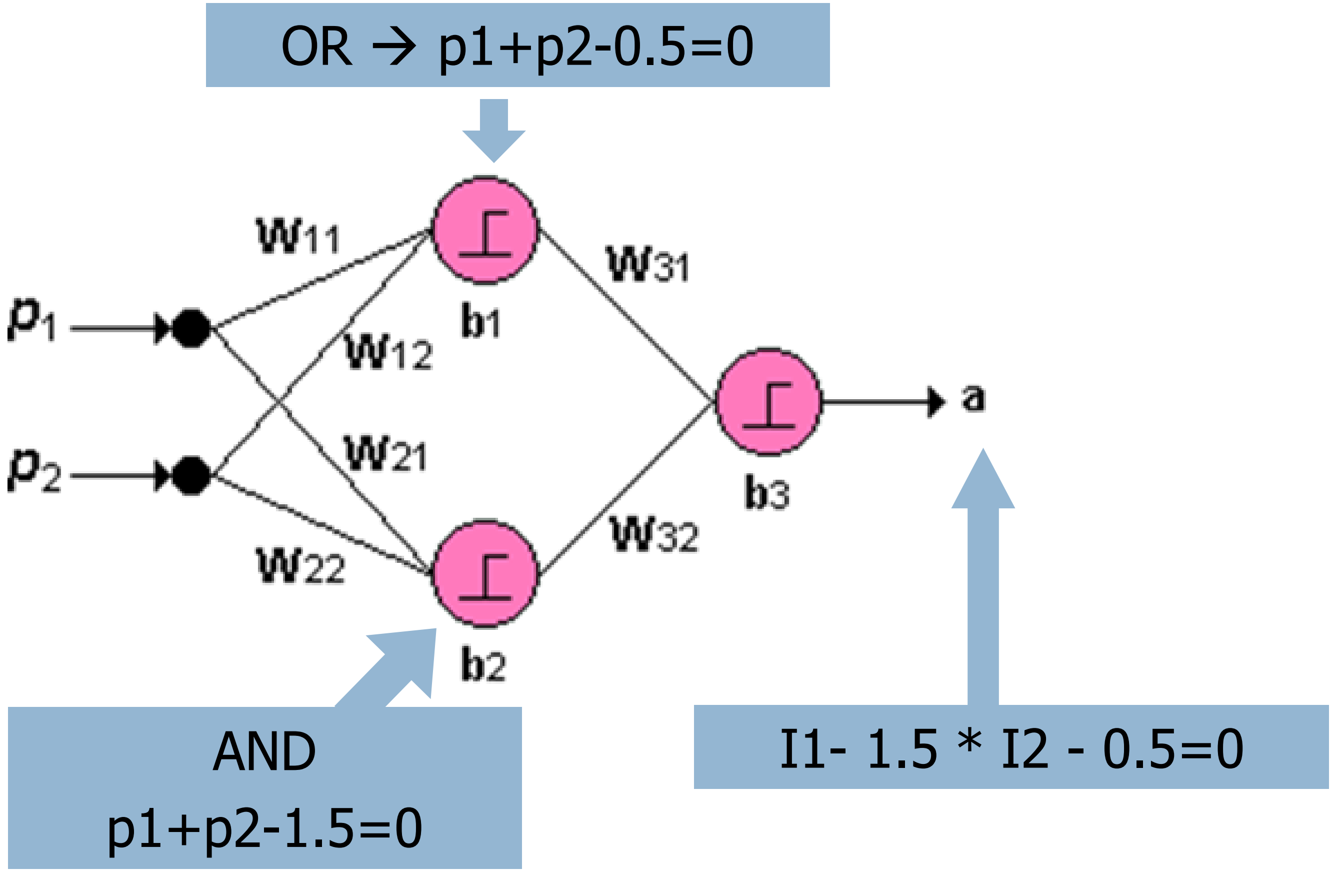
- Limitaciones?
- Resolución utilizando una neurona no lineal.

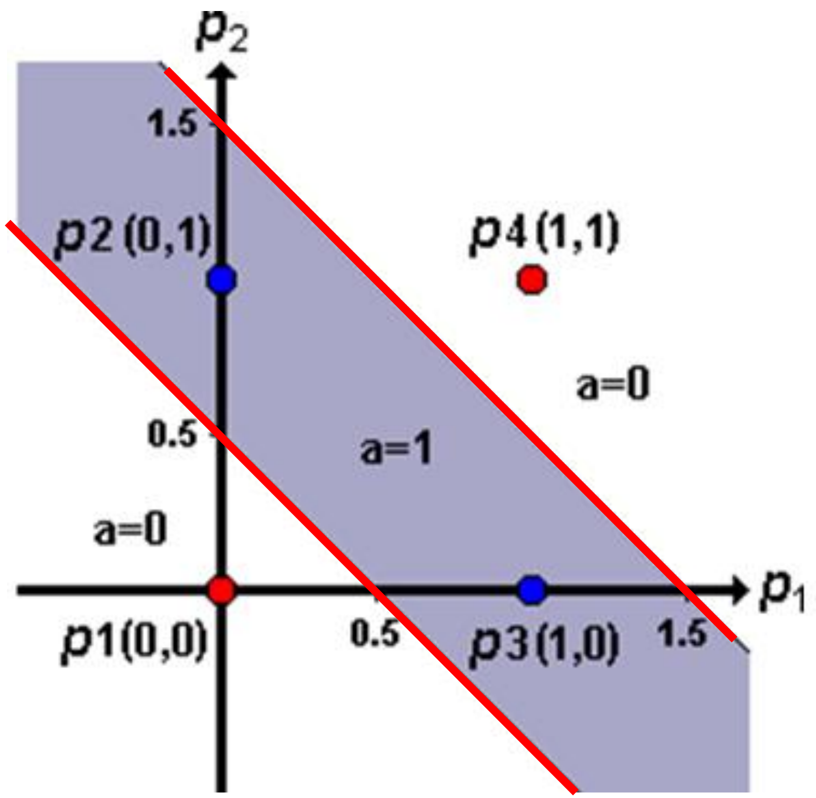
Redes multicapa

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



XOR



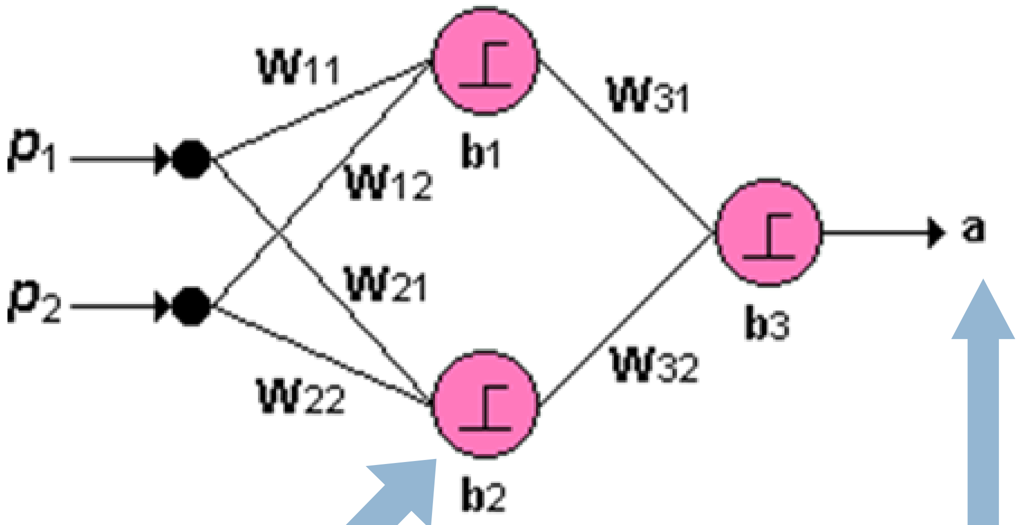


$w_{11}=1 \ \ w_{12}=1 \ \ b_1=-0.5$; $w_{21}=1 \ \ w_{22}=1 \ \ b_2=-1.5$; $w_{31}=1 \ \ w_{32}=-1.5 \ \ b_3=-0.5$

XOR

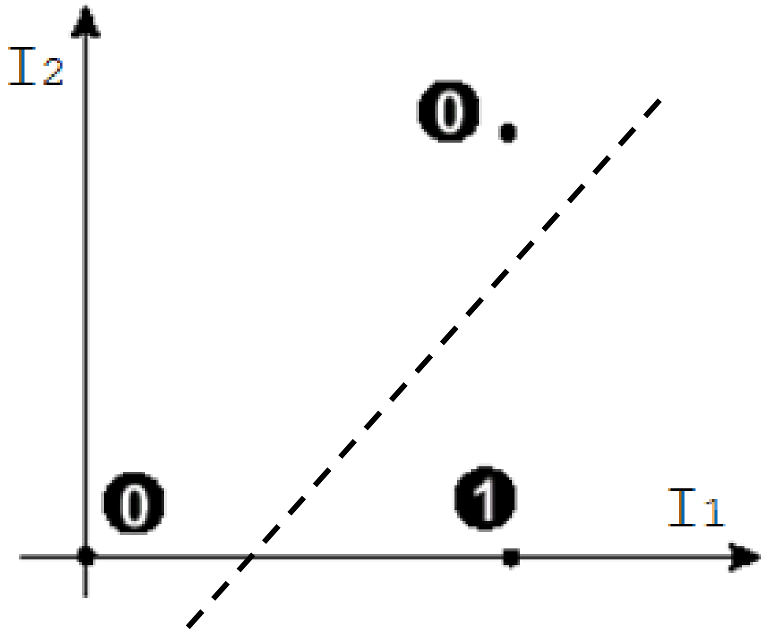
p1	p2	I1 (or)	I2 (AND)	a
1	0	1	0	1
1	1	1	1	0
0	0	0	0	0
0	1	1	0	1

OR → p1+p2-0.5=0

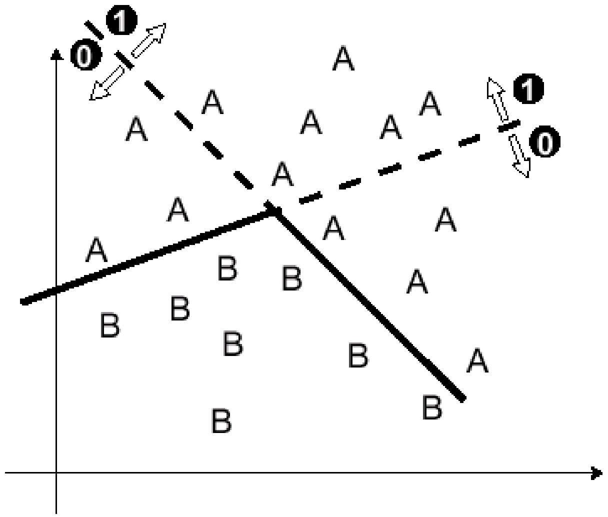


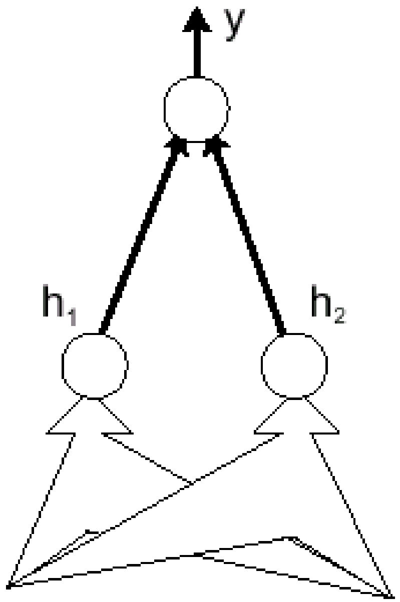
AND
p1+p2-1.5=0

I1- 1.5 * I2 - 0.5=0



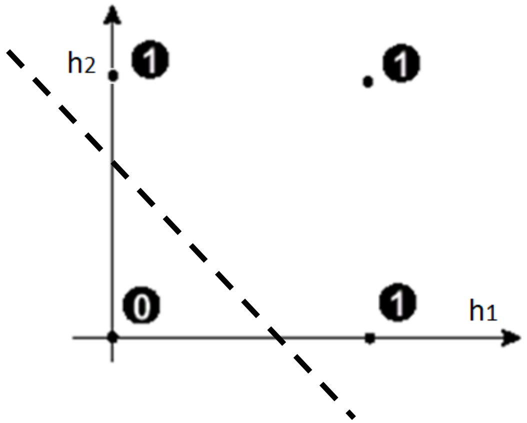
Problema no separable linealmente





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]