
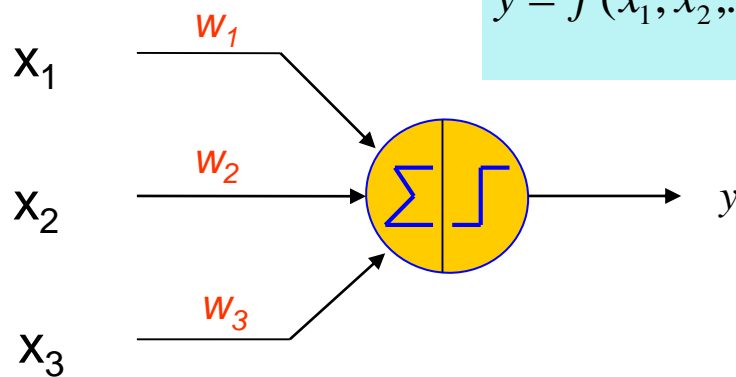
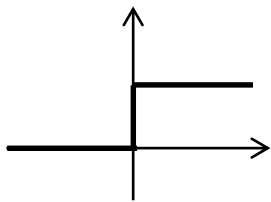


El Perceptrón

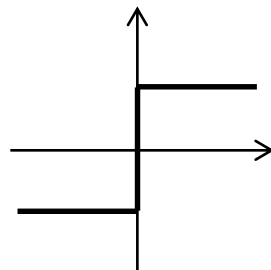
 1958 - El psicólogo **Frank Ronsenblant** desarrolló un modelo simple de neurona basado en el modelo de McCulloch y Pitts que utilizaba una regla de aprendizaje basada en la corrección del error: **Perceptrón**



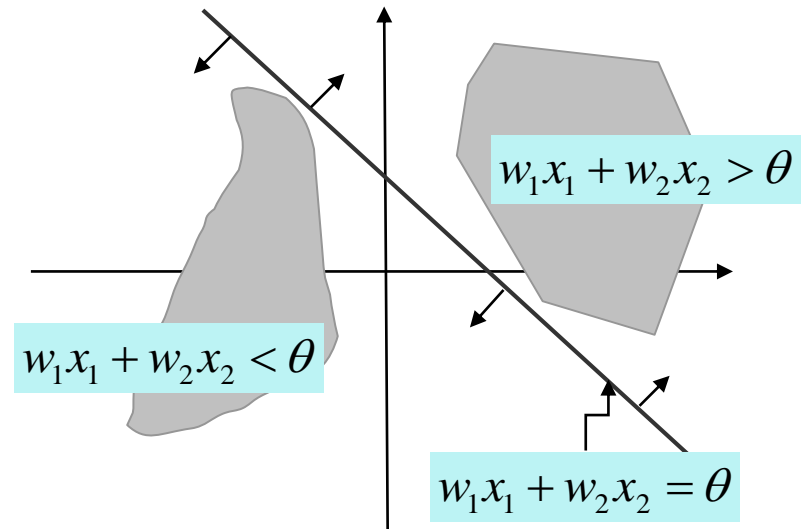
$$y = f(x_1, x_2, \dots, x_n) = \begin{cases} 1 & \text{si } w_1x_1 + w_2x_2 + \dots + w_nx_n \geq \theta \\ -1 & \text{si } w_1x_1 + w_2x_2 + \dots + w_nx_n < \theta \end{cases}$$



Función paso o *De Heaviside*
(Unidad de proceso binaria)



Función signo
(Bipolar)



Caso de perceptrón con 2 entradas

¿Qué se pretende con el Perceptrón?

Se dispone de la siguiente **información**:

- ❶ Conjunto de patrones $\{\mathbf{x}^k\}$, $k = 1, 2, \dots, p_1$, de la clase C_1 ($\mathbf{z}^k = 1$)
- ❶ Conjunto de patrones $\{\mathbf{x}^r\}$, $k = p_1 + 1, \dots, p$, de la clase C_2 ($\mathbf{z}^r = -1$)

►► Se pretende que el **perceptrón** asigne a cada entrada (patrón \mathbf{x}^k) la salida deseada \mathbf{z}^k siguiendo un proceso de corrección de error (**aprendizaje**) para determinar los **pesos sinápticos** apropiados

Regla de aprendizaje del Perceptrón:

$$\Delta w_j(k) = \eta(k) [z(k) - y(k)] x_j(k)$$

$$w_j(k+1) = \begin{cases} w_j(k) + 2\eta x_j(k) & \text{si } y(k) = -1 \text{ y } z(k) = 1, \\ w_j(k) & \text{si } y(k) = z(k) \\ w_j(k) - 2\eta x_j(k) & \text{si } y(k) = 1 \text{ y } z(k) = -1 \end{cases}$$

error

tasa de
aprendizaje

¿Cómo se modifica el sesgo θ ?

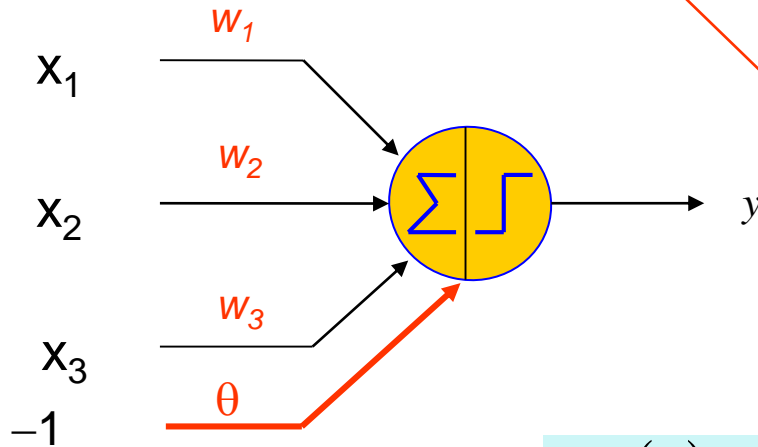
$$w_1x_1 + w_2x_2 + \dots + w_nx_n \geq \theta$$

\Leftrightarrow

$$w_1x_1 + w_2x_2 + \dots + w_nx_n + w_{n+1}x_{n+1} \geq 0$$

θ

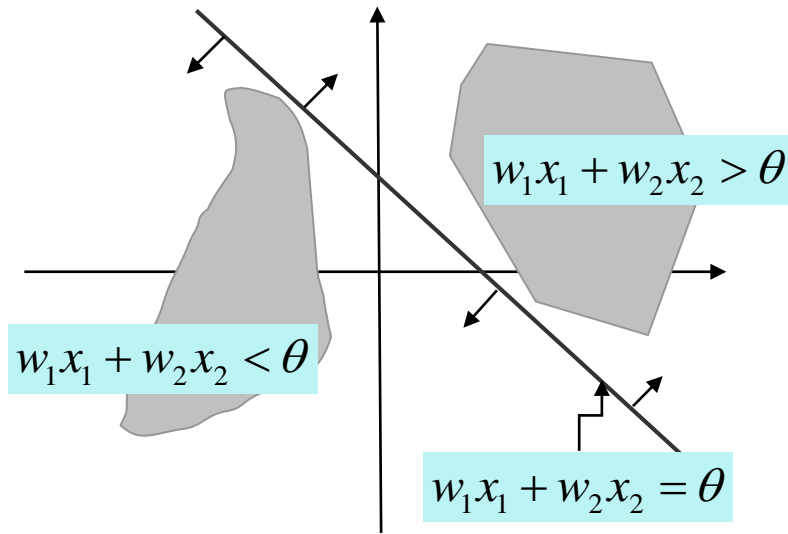
-1



$$w_1x_1 + w_2x_2 + \dots + w_nx_n + \theta(-1) \geq 0$$

$$\Delta\theta(k) = -\eta(k)[z(k) - y(k)]$$

El Perceptrón

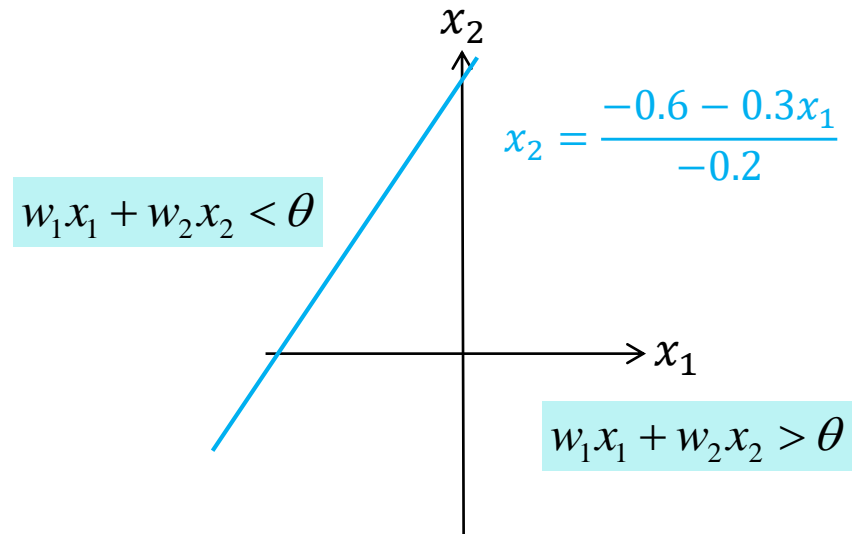
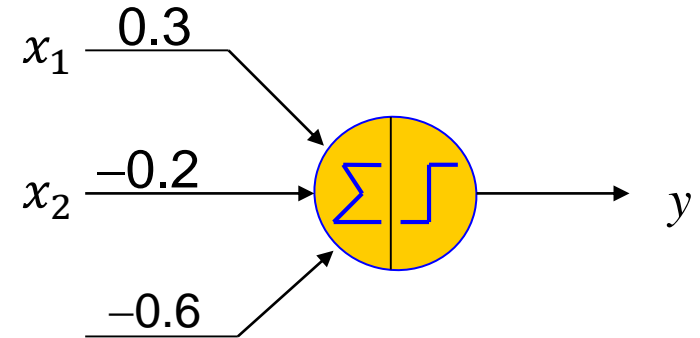


Caso de perceptrón con 2 entradas

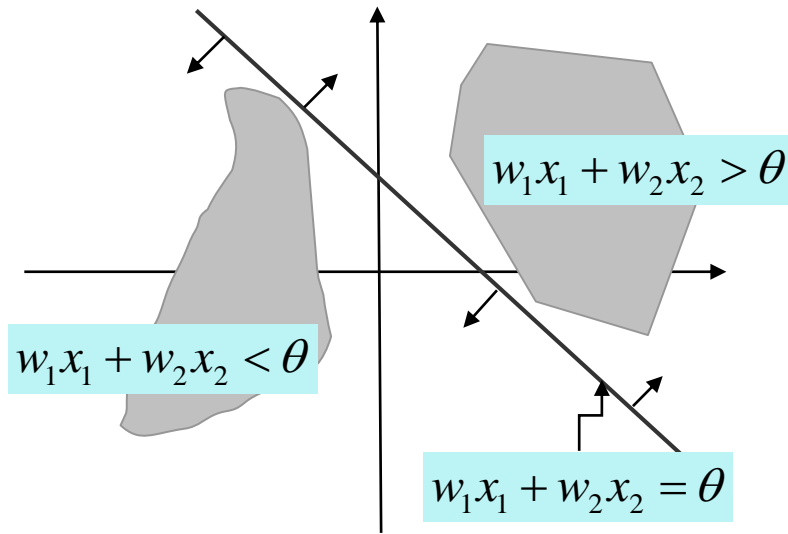
$$z = ax + b$$



$$x_2 = \frac{\theta - w_1x_1}{w_2}$$



El Perceptrón

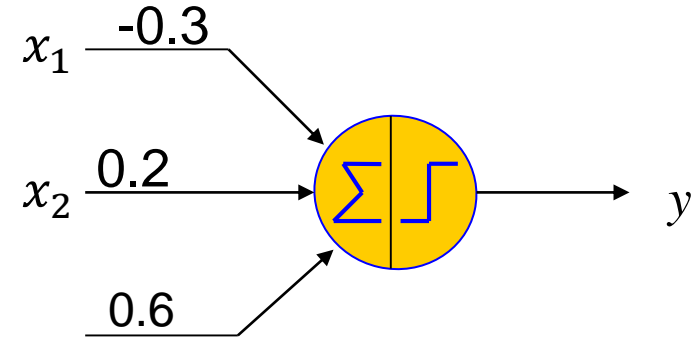


Caso de perceptrón con 2 entradas

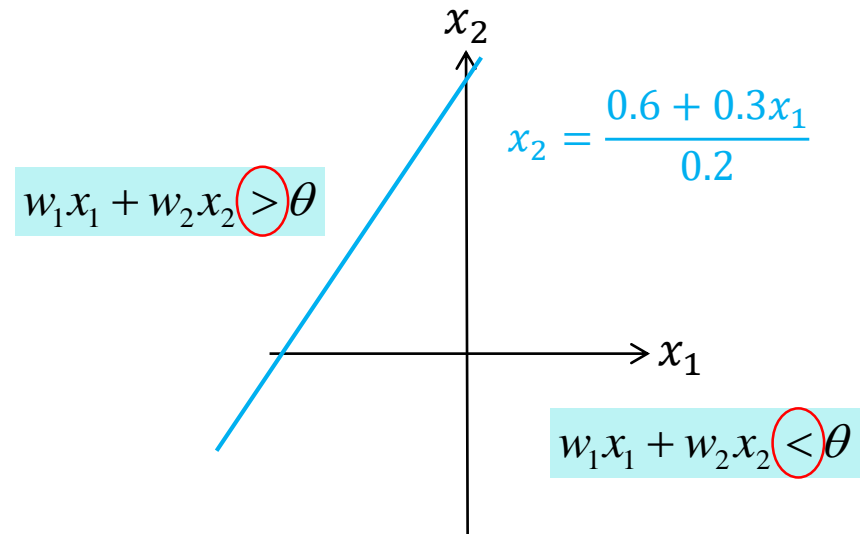
$$z = ax + b$$



$$x_2 = \frac{\theta - w_1x_1}{w_2}$$



Si invertimos el signo de los pesos...



Algoritmo del Perceptrón

Paso 0: Inicialización

Inicializar los pesos sinápticos con números aleatorios del intervalo $[-1,1]$.

Ir al paso 1 con $k=1$

Paso 1: (k-ésima iteración)

Calcular

$$y(k) = \text{sgn} \left(\sum_{j=1}^{n+1} w_j x_j(k) \right)$$

Paso 2: Corrección de los pesos sinápticos

Si $z(k) \neq y(k)$ modificar los pesos sinápticos según la expresión:

$$w_j(k+1) = w_j(k) + \eta [z_i(k) - y_i(k)] x_j(k), \quad j = 1, 2, \dots, n+1$$

Paso 3: Parada

Si no se han modificado los pesos en las últimas p iteraciones, es decir,

$$w_j(r) = w_j(k), \quad j = 1, 2, \dots, n+1, \quad r = k+1, \dots, k+p$$

parar. La red se ha estabilizado.

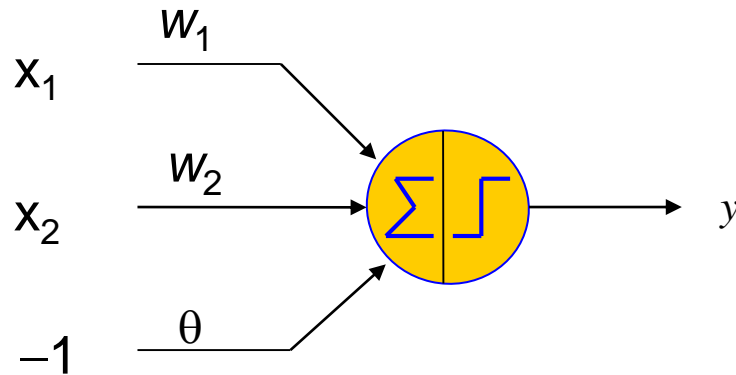
En otro caso, ir al **Paso 1** con $k=k+1$.

Ejemplo

😊 Diseña un perceptrón que implemente la función lógica AND

AND

<u>Entradas</u>	<u>Salidas</u>
(1, 1)	1
(1, -1)	-1
(-1, 1)	-1
(-1, -1)	-1

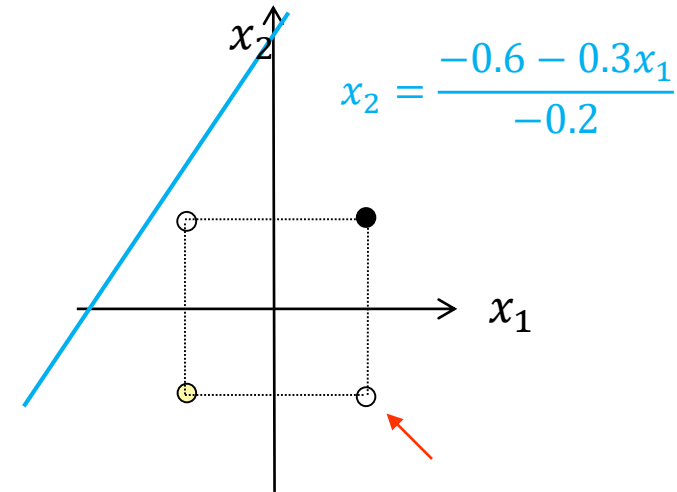
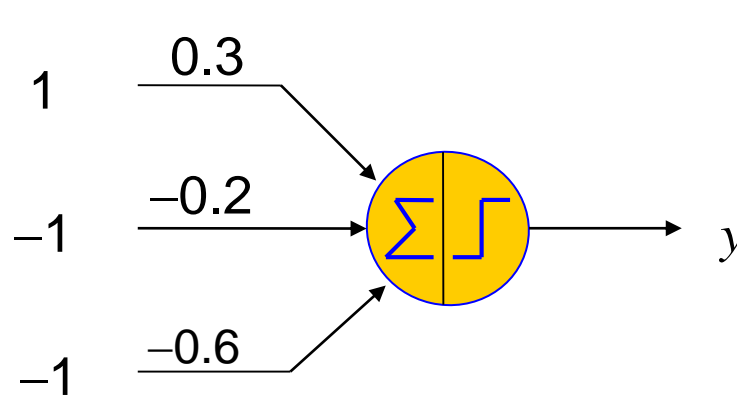


Paso 0: Inicialización aleatoria

$$w_1 = 0.3, \quad w_2 = -0.2, \quad \theta = -0.6,$$



Diseña un perceptrón que implemente la función lógica AND



Paso 1:

Patrón de entrada (1, -1):

$$h = 0.3(1) - 0.2(-1) - 0.6(-1) = 1.1$$

Paso 2: Corrección de los pesos sinápticos

$$w_1(1) = w_1(0) - 2\eta 1 = 0.3 - 1 = -0.7$$

$$w_2(1) = w_2(0) - 2\eta(-1) = -0.2 + 1 = 0.8$$

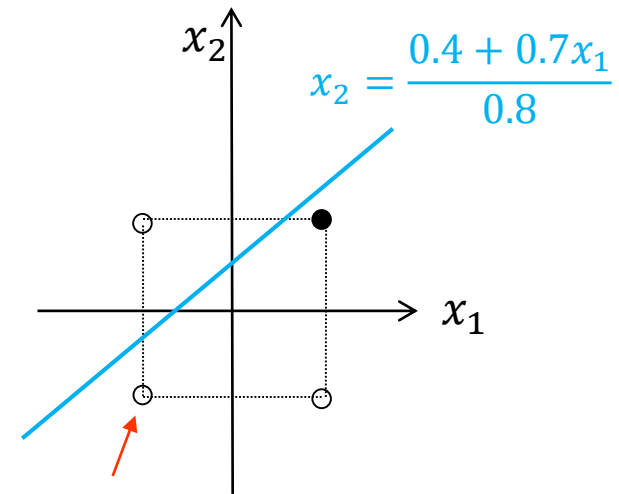
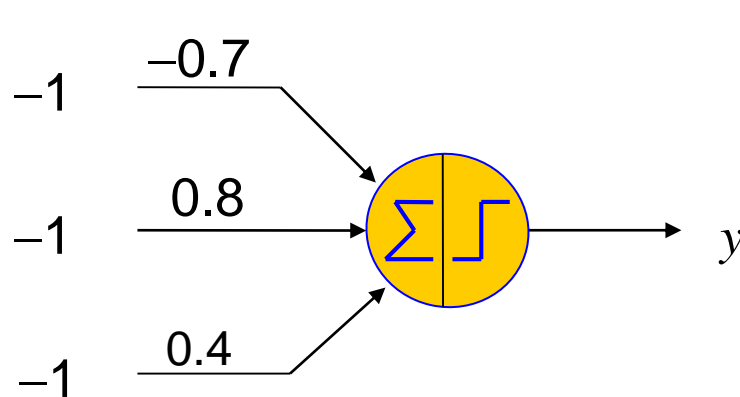
$$\theta(1) = \theta(0) - 2\eta(-1) = -0.6 + 1 = 0.4$$

$$y = 1$$

Elegimos $\eta=0.5$



Diseña un perceptrón que implemente la función lógica AND



Paso 1:

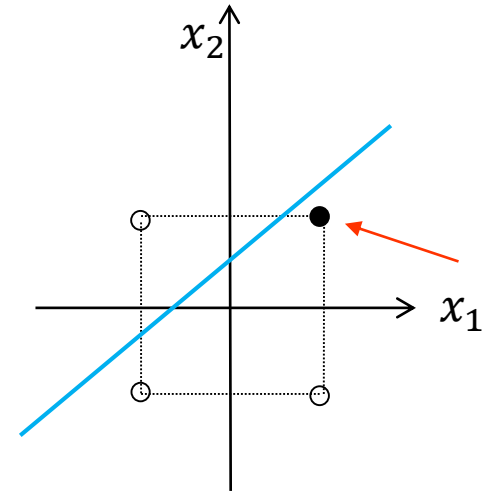
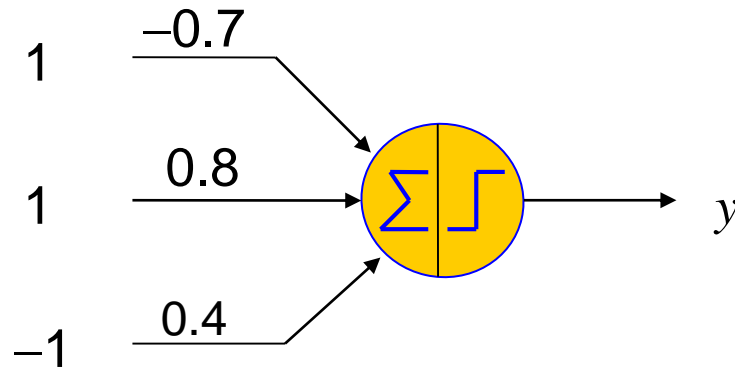
Patrón de entrada $(-1, -1)$: $h = -0.7(-1) + 0.8(-1) + 0.4(-1) = -0.5$

Como $y = -1$ y $z = -1$ la clasificación es **correcta**

$$y = -1$$



Diseña un perceptrón que implemente la función lógica AND



Paso 1:

Patrón de entrada (1,1):

$$h = -0.7(1) + 0.8(1) + 0.4(-1) = -0.3$$

Paso 2: Corrección de los pesos sinápticos

$$w_1(2) = w_1(1) + 2\eta(1) = -0.7 + 1 = 0.3$$

$$w_2(2) = w_2(1) + 2\eta(1) = 0.8 + 1 = 1.8$$

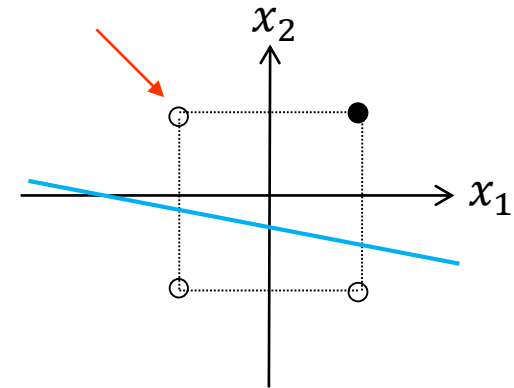
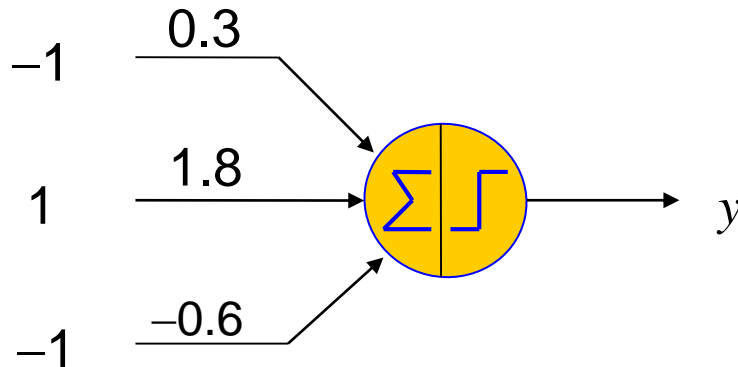
$$\theta(2) = \theta(1) + 2\eta(-1) = 0.4 - 1 = -0.6$$

Elegimos $\eta = 0.5$

$$y = -1$$



Diseña un perceptrón que implemente la función lógica AND



Paso 1:

Patrón de entrada $(-1, 1)$:

$$h = 0.3(-1) + 1.8(1) - 0.6(-1) = 2.1$$

Paso 2: Corrección de los pesos sinápticos

$$w_1(3) = w_1(2) - 2\eta(-1) = 0.3 + 1 = 1.3$$

$$w_2(3) = w_2(2) - 2\eta(1) = 1.8 - 1 = 0.8$$

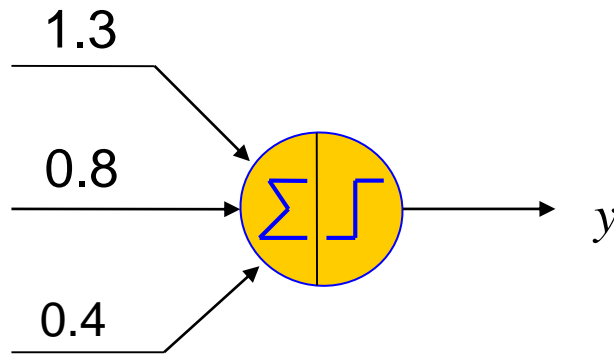
$$\theta(3) = \theta(2) - 2\eta(-1) = -0.6 + 1 = 0.4$$

$$y = 1$$

Elegimos $\eta = 0.5$



Diseña un perceptrón que implemente la función lógica AND



Ya hemos revisado todos los patrones de entrada

Patrón (1,1): $h = 1.3(1) + 0.8(1) + 0.4(-1) = 2.7$

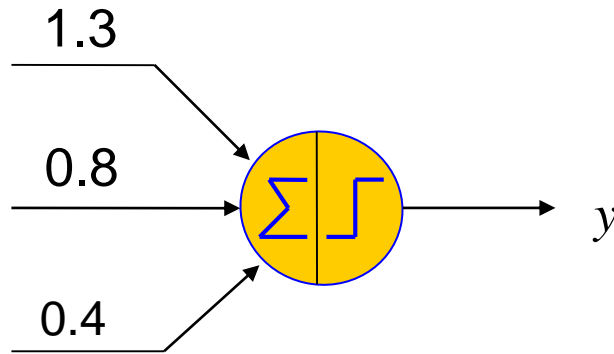
Patrón (1,-1): $h = 1.3(1) + 0.8(-1) + 0.4(-1) = 0.1$

Patrón (-1,-1): $h = 1.3(-1) + 0.8(-1) + 0.4(-1) = -2.5$

Patrón (-1,1): $h = 1.3(-1) + 0.8(1) + 0.4(-1) = -0.9$



Diseña un perceptrón que implemente la función lógica AND



Patrón (1,1): $h = 1.3(1) + 0.8(1) + 0.4(-1) = 2.7$



Patrón (1,-1): $h = 1.3(1) + 0.8(-1) + 0.4(-1) = 0.1$



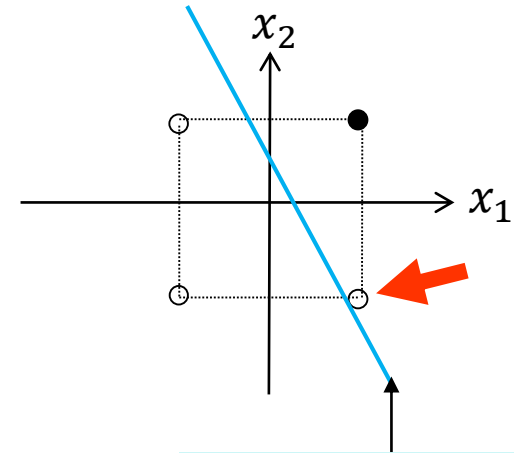
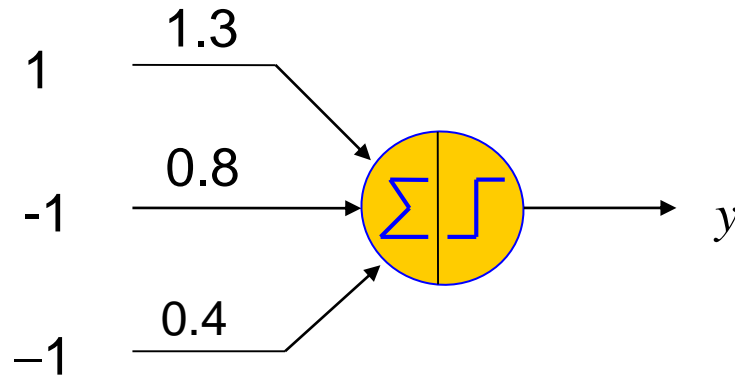
Patrón (-1,-1): $h = 1.3(-1) + 0.8(-1) + 0.4(-1) = -2.5$



Patrón (-1,1): $h = 1.3(-1) + 0.8(1) + 0.4(-1) = -0.9$




Diseña un perceptrón que implemente la función lógica AND



$$1.3x_1 + 0.8x_2 - 0.4 = 0$$

Patrón (1,1): $h = 1.3(1) + 0.8(1) + 0.4(-1) = 2.7$

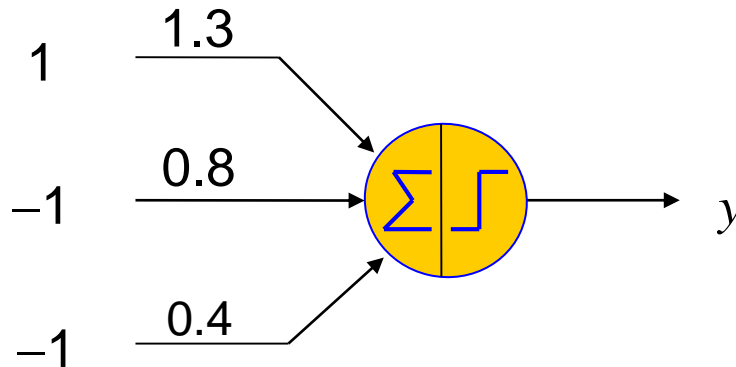
Patrón (1,-1): $h = 1.3(1) + 0.8(-1) + 0.4(-1) = 0.1$ 

Patrón (-1,-1): $h = 1.3(-1) + 0.8(-1) + 0.4(-1) = -2.5$

Patrón (-1,1): $h = 1.3(-1) + 0.8(1) + 0.4(-1) = -0.9$



Diseña un perceptrón que implemente la función lógica AND



Paso 1:

Patrón de entrada (1,-1):

$$h = 1.3(1) + 0.8(-1) + 0.4(-1) = 0.1$$

Paso 2: Corrección de los pesos sinápticos

$$w_1(3) = w_1(2) - 2\eta(1) = 1.3 - 1 = 0.7$$

$$w_2(3) = w_2(2) - 2\eta(-1) = 0.8 + 1 = 1.8$$

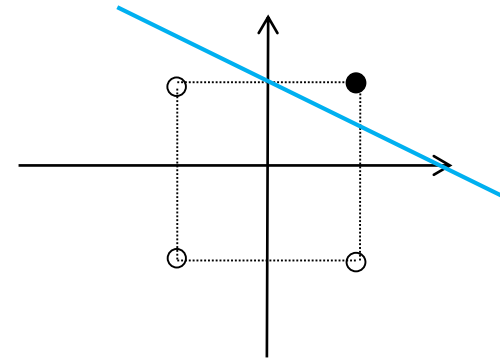
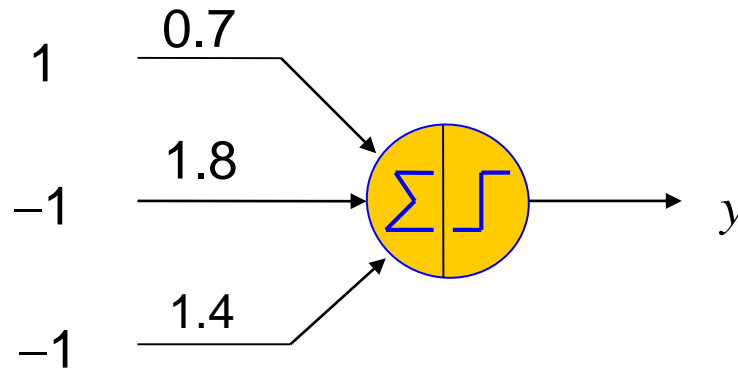
$$\theta(3) = \theta(2) - 2\eta(-1) = 0.4 + 1 = 1.4$$

$$y = 1$$

Elegimos $\eta = 0.5$



Diseña un perceptrón que implemente la función lógica AND



Paso 1:

Patrón de entrada (1,-1):

$$h = 0.7(1) + 1.8(-1) + 1.4(-1) = -2.5$$

Comprobar resto de muestras

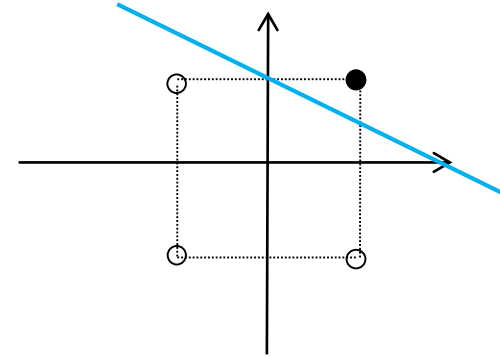
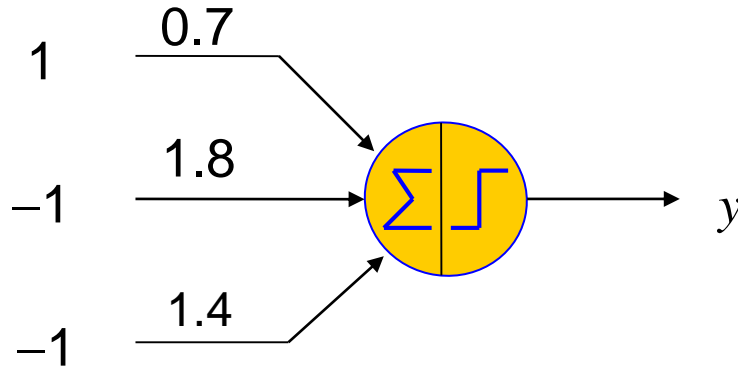
→FIN

$$y = -1$$

Elegimos $\eta = 0.5$



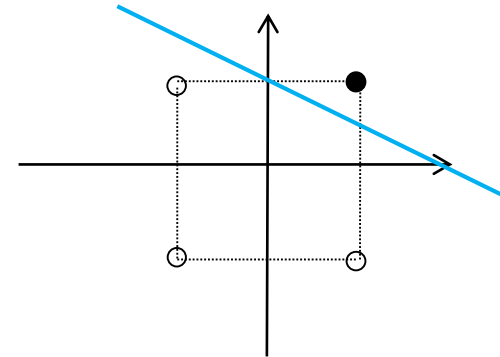
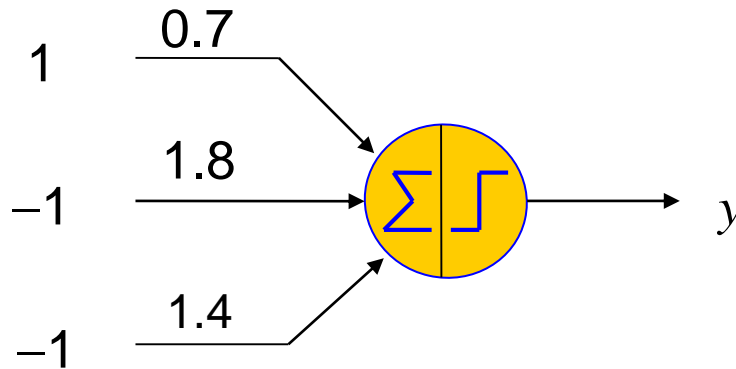
Diseña un perceptrón que implemente la función lógica AND



¿Cuándo parar realmente?



Diseña un perceptrón que implemente la función lógica AND



¿Cuándo parar realmente?

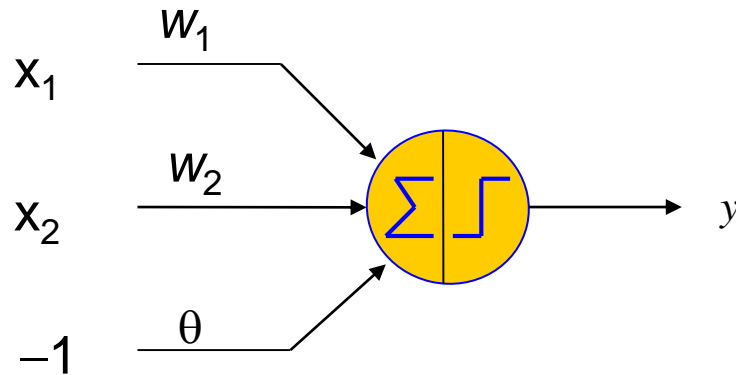
- ¿Cuando no haya error de clasificación?...
- ¿Cuando queramos?...

Ejemplo

☺ Diseña un perceptrón que implemente la función lógica XOR

XOR

<u>Entradas</u>	<u>Salidas</u>
(1, 1)	-1
(1, -1)	1
(-1, 1)	1
(-1, -1)	-1

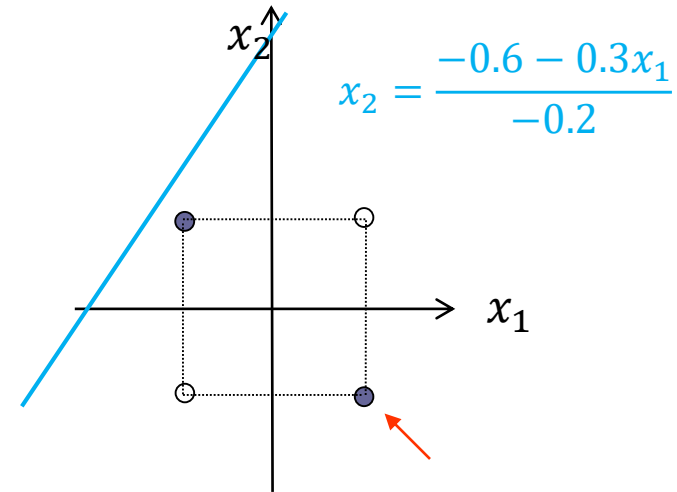
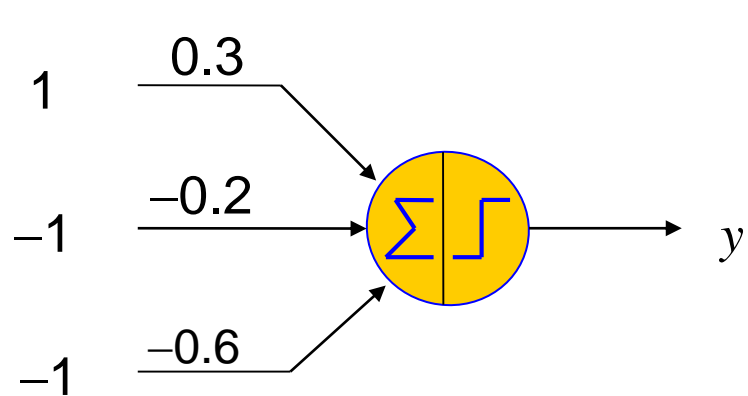


Paso 0: Inicialización aleatoria

$$w_1 = 0.3, \quad w_2 = -0.2, \quad \theta = -0.6,$$



Diseña un perceptrón que implemente la función lógica XOR



Paso 1:

Patrón de entrada (1,-1):

$$h = 0.3(1) - 0.2(-1) - 0.6(-1) = 1.1$$

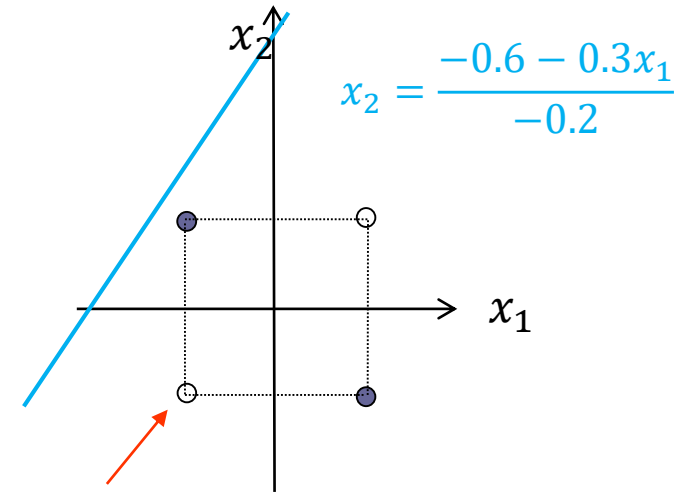
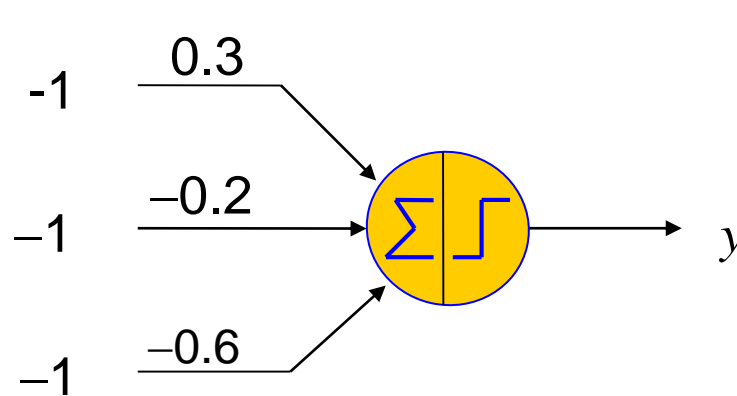
Elegimos $\eta=0.5$

$$y = 1$$

Como $y = 1$ y $z = 1$ la clasificación es **correcta**



Diseña un perceptrón que implemente la función lógica XOR



Paso 1:

Patrón de entrada $(-1, -1)$:

$$h = 0.3(-1) - 0.2(-1) - 0.6(-1) = 0.5$$

Paso 2: Corrección de los pesos sinápticos

$$w_1(2) = w_1(1) - 2\eta(-1) = 0.3 + 1 = 1.3$$

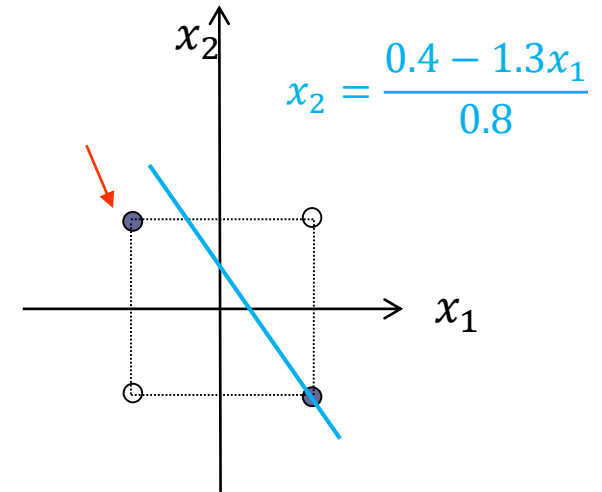
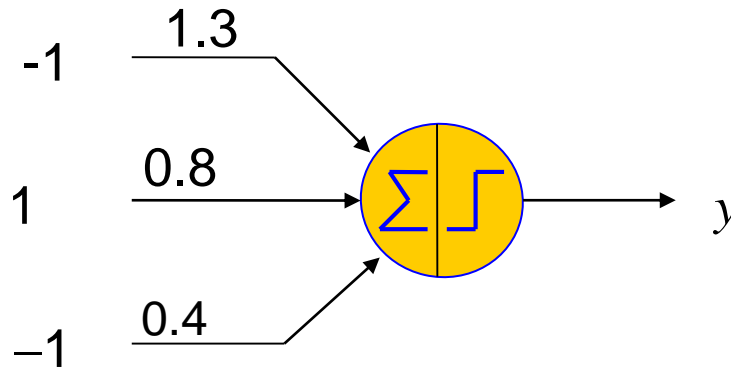
$$w_2(2) = w_2(1) - 2\eta(-1) = -0.2 + 1 = 0.8$$

$$\theta(2) = \theta(1) - 2\eta(-1) = -0.6 + 1 = 0.4$$

$$y = 1$$



Diseña un perceptrón que implemente la función lógica XOR



Paso 1:

Patrón de entrada $(-1, 1)$:

$$h = 1.3(-1) + 0.8(1) + 0.4(-1) = -0.9$$

Paso 2: Corrección de los pesos sinápticos

$$w_1(3) = w_1(2) + 2\eta(-1) = 1.3 - 1 = 0.3$$

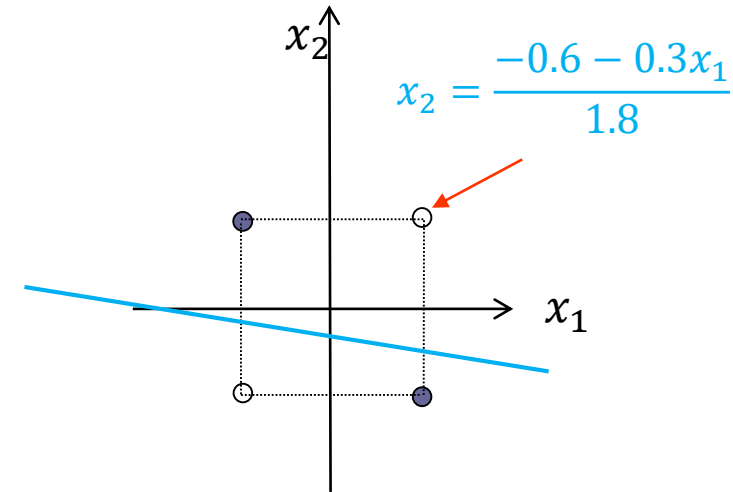
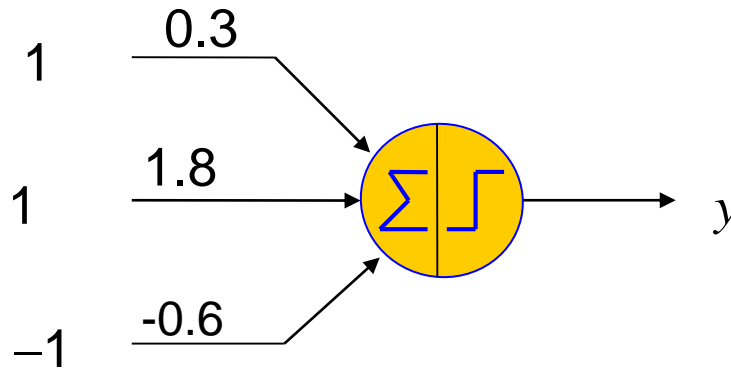
$$w_2(3) = w_2(2) + 2\eta(1) = 0.8 + 1 = 1.8$$

$$\theta(3) = \theta(2) + 2\eta(-1) = 0.4 - 1 = -0.6$$

$$y = -1$$



Diseña un perceptrón que implemente la función lógica XOR



Paso 1:

Patrón de entrada (1,1):

$$h = 0.3(1) + 1.8(1) - 0.6(-1) = 2.7$$

Paso 2: Corrección de los pesos sinápticos

$$w_1(4) = w_1(3) - 2\eta(1) = 0.3 - 1 = -0.7$$

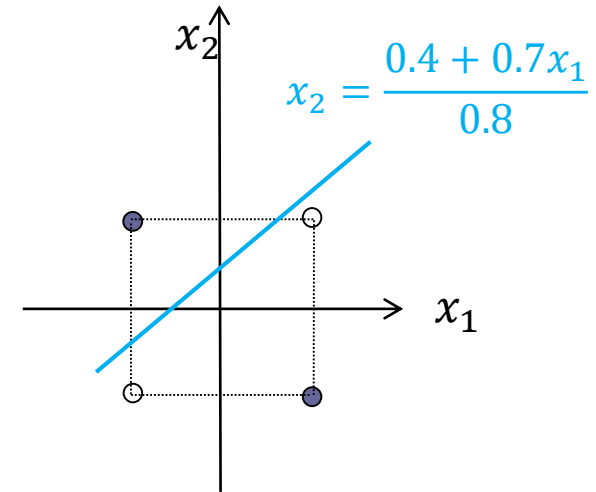
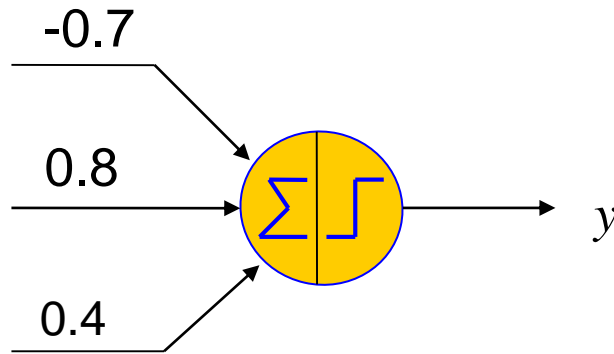
$$w_2(4) = w_2(3) - 2\eta(1) = 1.8 - 1 = 0.8$$

$$\theta(4) = \theta(3) - 2\eta(-1) = -0.6 + 1 = 0.4$$

$$y = 1$$



Diseña un perceptrón que implemente la función lógica AND



Patrón (1,1): $h = -0.7(1) + 0.8(1) + 0.4(-1) = -0.3$



Patrón (1,-1): $h = -0.7(1) + 0.8(-1) + 0.4(-1) = -1.9$



Patrón (-1,-1): $h = -0.7(-1) + 0.8(-1) + 0.4(-1) = -0.5$



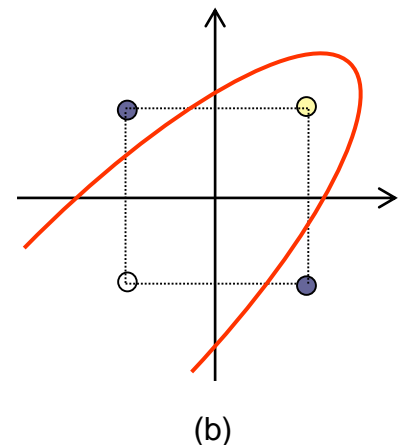
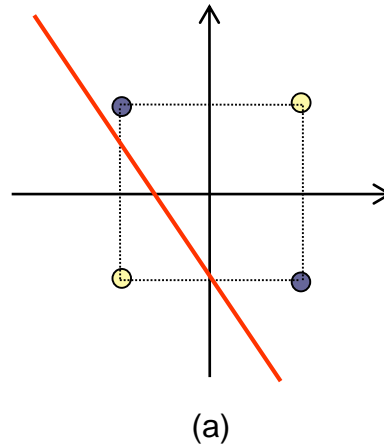
Patrón (-1,1): $h = -0.7(-1) + 0.8(1) + 0.4(-1) = 1.1$

El Perceptrón

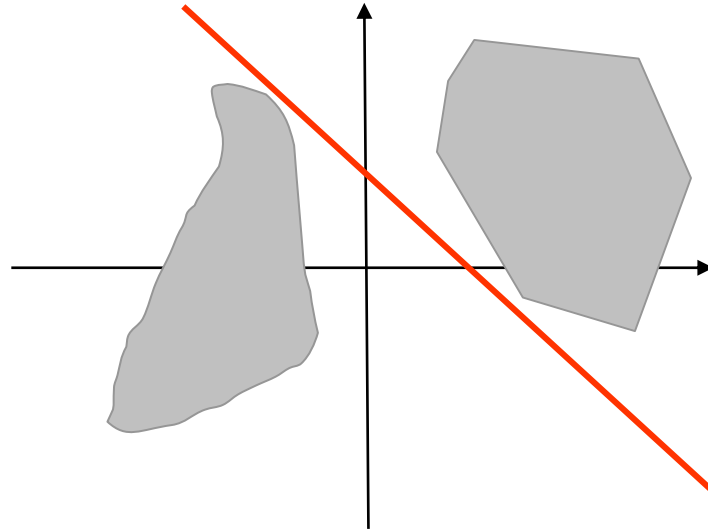
☹ ¿Dado un conjunto cualquiera de patrones de entrenamiento, puede el Perceptrón aprender a clasificarlos correctamente?

Problema XOR

<u>Entradas</u>	<u>Salidas</u>
(1, 1)	-1
(1, -1)	1
(-1, 1)	1
(-1, -1)	-1



Conjuntos separables linealmente



Teorema de convergencia del Perceptrón

Si el conjunto de patrones de entrenamiento con sus salidas deseadas,
 $\{\mathbf{x}^1, z^1\}, \{\mathbf{x}^2, z^2\}, \dots, \{\mathbf{x}^p, z^p\},$
es **linealmente separable** entonces el Perceptrón simple encuentra una
solución en un **número finito de iteraciones**

Demostración

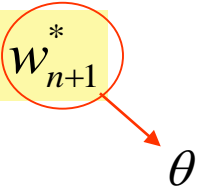
Como es **linealmente separable** entonces existen

$$\sum_{j=1}^n w_j^* x_j > w_{n+1}$$

si son de la clase C_1

$$\sum_{j=1}^n w_j^* x_j < w_{n+1}$$

si son de la clase C_2

$$w_1^*, w_2^*, \dots, w_{n+1}^*$$


θ

Demostración

Supongamos que la salida de la red $y(k)$ es diferente de la deseada $z(k)$

$$\sum_{j=1}^{n+1} (w_j(k+1) - w_j^*)^2 = \sum_{j=1}^{n+1} (w_j(k) + \eta[z(k) - y(k)]x_j(k) - w_j^*)^2$$

$$= \sum_{j=1}^{n+1} (w_j(k) - w_j^*)^2 + \eta^2 [z(k) - y(k)]^2 \sum_{j=1}^{n+1} x_j(k)^2 + 2\eta [z(k) - y(k)] \sum_{j=1}^{n+1} (w_j(k) - w_j^*) x_j(k)$$

Cuadrado del binomio
 $[\text{○} + \text{○}]^2$

Desarrollando productos
de la diferencia: $w_j(k) - w_j^*$

$$+ 2\eta [z(k) - y(k)] \left(\sum_{j=1}^{n+1} w_j(k) x_j(k) \right) - 2\eta [z(k) - y(k)] \sum_{j=1}^{n+1} w_j^* x_j(k)$$

$$|z(k) - y(k)| = 2$$

razonando por casos

razonando por casos

$$\begin{aligned} (-2)^2 &= 4 \\ (+2)^2 &= 4 \end{aligned}$$

$$- 2 \left| \sum_{j=1}^{n+1} w_j(k) x_j(k) \right|$$

$$2 \left| \sum_{j=1}^{n+1} w_j^* x_j(k) \right|$$

$$\leq \sum_{j=1}^{n+1} (w_j(k) - w_j^*)^2 + 4\eta^2 \sum_{j=1}^{n+1} x_j(k)^2 + 0 - 4\eta \left| \sum_{j=1}^{n+1} w_j^* x_j(k) \right|$$

Eliminamos arbitrariamente término negativo, obligamos que la igualdad sea una desigualdad \leq

Demostración

$$\sum_{j=1}^{n+1} (w_j(k+1) - w_j^*)^2 \leq \sum_{j=1}^{n+1} (w_j(k) - w_j^*)^2 + 4\eta^2 \sum_{j=1}^{n+1} x_j(k)^2 - 4\eta \left| \sum_{j=1}^{n+1} w_j^* x_j(k) \right|$$

$$\sum_{j=1}^{n+1} (w_j(k+1) - w_j^*)^2 \leq \sum_{j=1}^{n+1} (w_j(k) - w_j^*)^2 + 4\eta^2 L - 4\eta T$$

$$T = \min_{1 \leq k \leq p} \left\{ \sum_{j=1}^{n+1} w_j^* x_j(k) \right\}$$

$$L = \max_{1 \leq k \leq p} \left\{ \sum_{j=1}^{n+1} x_j(k)^2 \right\}$$

$$\sum_{j=1}^{n+1} (w_j(k+1) - w_j^*)^2 \leq \sum_{j=1}^{n+1} (w_j(k) - w_j^*)^2 + 4\eta(\eta L - T)$$

Si $\eta L - T < 0 \implies \overset{>0}{\sum_{j=1}^{n+1} (w_j(k+1) - w_j^*)^2} - \overset{>0}{\sum_{j=1}^{n+1} (w_j(k) - w_j^*)^2} \leq \overset{<0}{4\eta(\eta L - T)}$

$$\eta < \frac{T}{L}$$

$$\sum_{j=1}^{n+1} (w_j(k+1) - w_j^*)^2 < \sum_{j=1}^{n+1} (w_j(k) - w_j^*)^2$$


La distancia a los pesos óptimos de la siguiente iteración es menor estrictamente que la de la actual (mientras la salida sea diferente de la deseada)

Tasa de aprendizaje óptima

Se trata de elegir la tasa de aprendizaje η de manera que se produzca un mayor decrecimiento del error en cada iteración

Error cuadrático en la iteración $k+1$

$$E(\eta) = D(k+1) = D(k) + 4\eta^2 \sum_{j=1}^{n+1} x_j(k)^2 - 4\eta \left| \sum_{j=1}^{n+1} w_j(k) x_j(k) \right| - 4\eta \left| \sum_{j=1}^{n+1} w_j^* x_j(k) \right|$$


$$D(k+1) = \sum_{j=1}^{n+1} (w_j(k+1) - w_j^*)^2$$

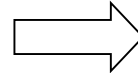
$$\frac{\partial E(\eta)}{\partial \eta} = 8\eta \sum_{j=1}^{n+1} x_j(k)^2 - 4 \left| \sum_{j=1}^{n+1} w_j(k) x_j(k) \right| - 4 \left| \sum_{j=1}^{n+1} w_j^* x_j(k) \right| = 0$$

$$\eta_{opt} = \frac{\left| \sum_{j=1}^{n+1} w_j(k) x_j(k) \right| + \left| \sum_{j=1}^{n+1} w_j^* x_j(k) \right|}{2 \sum_{j=1}^{n+1} x_j(k)^2}$$

Tasa de aprendizaje óptima

$$\eta_{opt} = \frac{\left| \sum_{j=1}^{n+1} w_j(k) x_j(k) \right| + \left| \sum_{j=1}^{n+1} w_j^* x_j(k) \right|}{2 \sum_{j=1}^{n+1} x_j(k)^2}$$

Aproximación, porque
no conocemos w_j^*



$$\tilde{\eta}_{opt} = \frac{\left| \sum_{j=1}^{n+1} w_j(k) x_j(k) \right|}{\sum_{j=1}^{n+1} x_j(k)^2}$$

$$-2 \left| \sum_{j=1}^{n+1} w_j(k) x_j(k) \right| = [z(k) - y(k)] \sum_{j=1}^{n+1} w_j(k) x_j(k)$$

$$\tilde{\eta}_{opt} = \frac{-[z(k) - y(k)] \sum_{j=1}^{n+1} w_j(k) x_j(k)}{2 \sum_{j=1}^{n+1} x_j(k)^2}$$

Regla de aprendizaje normalizada

Sustituyendo el parámetro η en la regla de aprendizaje:

$$w_j(k+1) = w_j(k) - \frac{[z(k) - y(k)] \sum_{j=1}^{n+1} w_j(k) x_j(k)}{2 \sum_{j=1}^{n+1} x_j(k)^2} [z(k) - y(k)] x_j(k)$$

=4

$$w_j(k+1) = w_j(k) - 2 \frac{\sum_{j=1}^{n+1} w_j(k) x_j(k)}{\sum_{j=1}^{n+1} x_j(k)^2} x_j(k)$$

Regla de aprendizaje normalizada

Partiendo de un vector de pesos normalizado, todos los pesos que se obtienen estarán normalizados:

$$\|\mathbf{w}(k+1)\|^2 = \sum_{j=1}^{n+1} w_j(k+1)^2$$

$$= \sum_{j=1}^{n+1} w_j(k)^2 + \sum_{j=1}^{n+1} x_j(k)^2 \left(2 \frac{\sum_{j=1}^{n+1} w_j(k) x_j(k)}{\sum_{j=1}^{n+1} x_j(k)^2} \right)^2 - 4 \frac{\sum_{j=1}^{n+1} w_j(k) x_j(k)}{\sum_{j=1}^{n+1} x_j(k)^2} \sum_{i=1}^{n+1} w_j(k) x_j(k)$$

$$= \sum_{j=1}^{n+1} w_j(k)^2 = 1$$

$$\|\mathbf{w}(k)\| = \sqrt{\sum_{j=1}^{n+1} w_j(k)^2} = 1$$

Interpretación geométrica de la regla de aprendizaje del Perceptrón

$$\mathbf{w}^T \mathbf{x} = \sum_{j=1}^{n+1} w_j x_j \geq 0 \quad \forall \mathbf{x} \in C_1$$

$$y = \text{sgn}(\mathbf{w}^T \mathbf{x}) = z, \quad \forall \mathbf{x} \in C_1 \cup C_2$$

$$\mathbf{w}^T \mathbf{x} = \sum_{j=1}^{n+1} w_j x_j < 0 \quad \forall \mathbf{x} \in C_2$$

$$\mathbf{w}(k+1) = \begin{cases} \mathbf{w}(k) + \mathbf{a}(k) & \text{si } (\mathbf{a}(k))^T \mathbf{w}(k) \leq 0 \\ \mathbf{w}(k) & \text{en otro caso} \end{cases}$$

$$\mathbf{a}(k) = \begin{cases} \mathbf{x}(k) & \text{si } z(k) = 1 \\ -\mathbf{x}(k) & \text{si } z(k) = -1 \end{cases}$$

Tasa aprendizaje 0.5

Se realizan las correcciones siempre y cuando se producen clasificaciones incorrectas, es decir,

$$(\mathbf{a}(k))^T \mathbf{w}(k) \leq 0$$

Interpretación geométrica de la regla de aprendizaje del Perceptrón

Se realizan las correcciones siempre y cuando se producen clasificaciones incorrectas, es decir,

$$(\mathbf{a}(k))^T \mathbf{w}(k) \leq 0$$

$$\begin{aligned}\mathbf{w}(k+1) &= \mathbf{w}(k) + \Delta(k) \\ &= \mathbf{w}(k) + \mathbf{a}(k)\end{aligned}$$

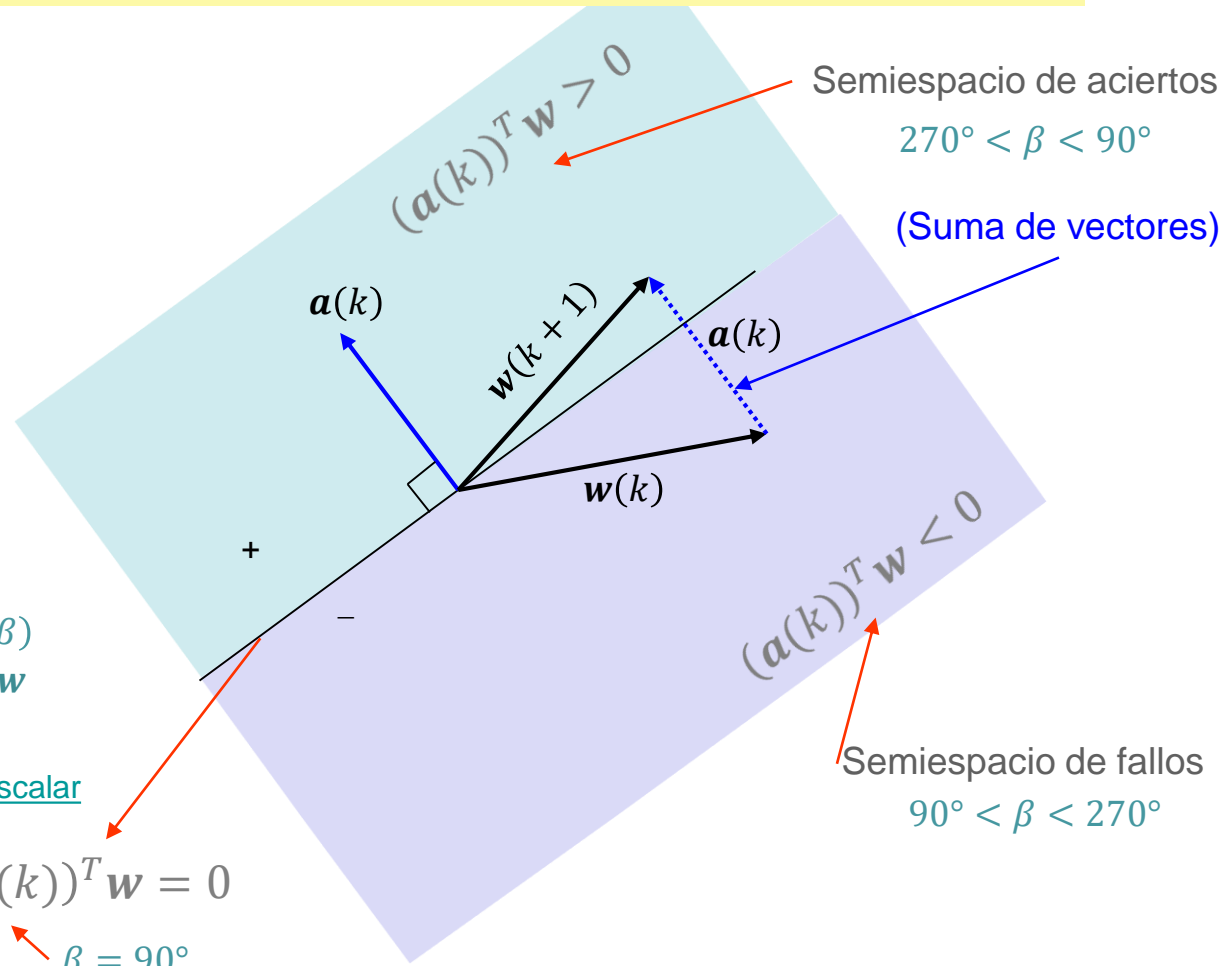
Tasa aprendizaje 0.5

Ojo: $(\mathbf{a}(k))^T \mathbf{w}(k)$
es un **producto escalar**, es decir:
 $(\mathbf{a}(k))^T \mathbf{w}(k) = \vec{a} \cdot \vec{w} = |\mathbf{a}| \cdot |\mathbf{w}| \cdot \cos(\beta)$
donde β es el ángulo que forman \mathbf{a} y \mathbf{w}

<https://www.fisicalab.com/apartado/producto-escalar>

$$(\mathbf{a}(k))^T \mathbf{w} = 0$$

$$\beta = 90^\circ$$



Deducción de la regla de aprendizaje por lotes

La regla de aprendizaje del Perceptrón intenta encontrar una solución \mathbf{w}^* para el siguiente sistema de desigualdades:

$$\mathbf{a}(k)^T \mathbf{w} > 0 \quad k=1,2,\dots,p$$

Función criterio a minimizar:

$$J(\mathbf{w}) = - \sum_{k \in I(\mathbf{w})} \mathbf{a}(k)^T \mathbf{w}$$

$I(\mathbf{w})$ es el conjunto de patrones clasificados incorrectamente utilizando el vector de pesos sinápticos \mathbf{w} (es decir, $\mathbf{a}(k)^T \mathbf{w} \leq 0$). Así, J nunca es negativo y si dicho conjunto es vacío entonces J alcanza su valor mínimo, $J = 0$.

Método del descenso del gradiente

$$\nabla J = \sum_{k \in I(\mathbf{w})} (-\mathbf{a}(k))$$

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) - \eta(k) \nabla J \\ &= \mathbf{w}(k) + \eta(k) \sum_{k \in I(\mathbf{w})} \mathbf{a}(k) \end{aligned}$$

Algoritmo de aprendizaje por lotes del Perceptrón

Paso 0: Inicialización

Inicializar los pesos sinápticos con números aleatorios del intervalo $[-1,1]$. Fijar un valor de parada s . Ir al paso 1 con $k=1$

Paso 1: (k -ésima iteración) Corrección de los pesos sinápticos

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k) \sum_{k \in I(\mathbf{w})} \mathbf{a}(k)$$

Obsérvese que si actualizamos un solo patrón, es la regla de aprendizaje individualizado

Paso 2: Parada

$$\text{Si } \left| \eta(k) \sum_{k \in I(\mathbf{w})} \mathbf{a}(k) \right| < s \quad \text{parar.}$$

En otro caso, ir al **Paso 1** con $k=k+1$.

Paso 1

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k) \mathbf{a}(k)$$

Una modificación: La Regla del Bolsillo

Consiste en tener en cuenta el número de iteraciones consecutivas del algoritmo de perceptrón en las cuales no se ha modificado el vector de pesos sinápticos (para cada uno de los vectores que va generando), es decir, tener en cuenta el número de patrones que se han clasificado correctamente con dicho vector hasta que se ha encontrado el primer patrón que clasifica incorrectamente. Se tiene “guardado en el bolsillo” la mejor solución explorada, es decir, el vector de pesos sinápticos generado que ha conseguido, hasta el momento, el mayor número de iteraciones sin ser modificado. Cuando se encuentra un nuevo vector de pesos sinápticos que consigue un mayor número de clasificaciones correctas consecutivas que el que hay en el bolsillo entonces el vector del bolsillo se reemplaza por este. La solución final viene dada por el vector de pesos sinápticos guardado en el bolsillo.

Una modificación: La Regla del Bolsillo

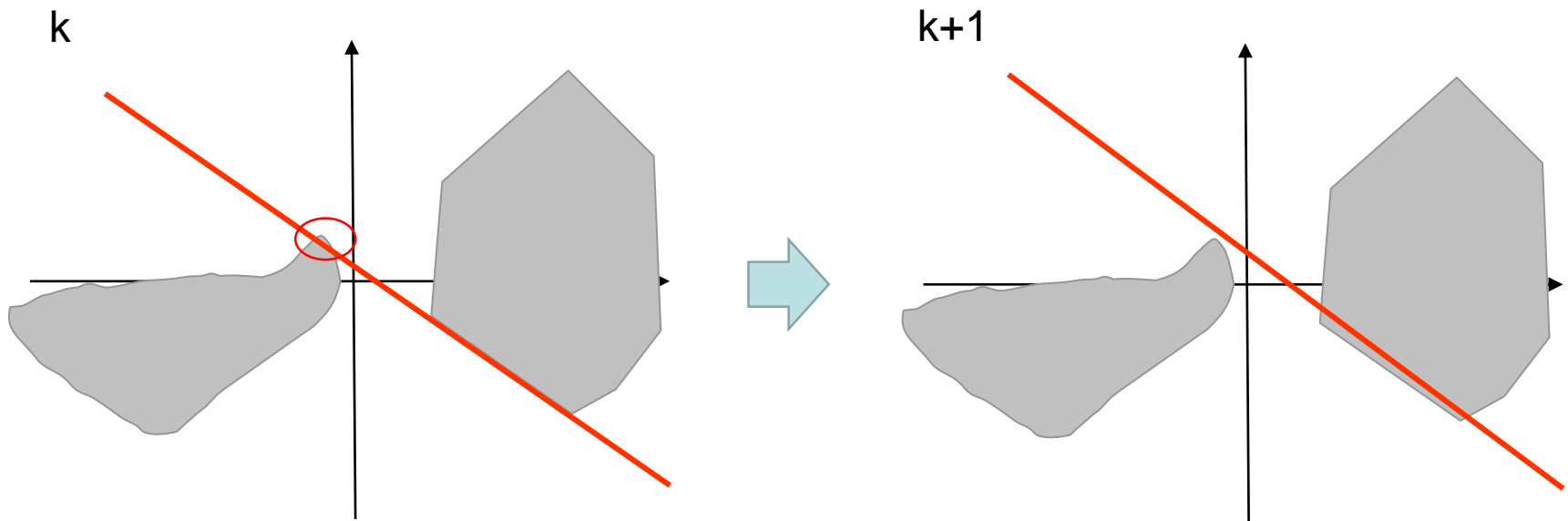
Forma de implementarlo:

- Al finalizar la época, calcular el número de aciertos del vector de pesos actual en el conjunto de entrenamiento. Si es mejor que el que teníamos en el bolsillo, se guarda el vector y el número de aciertos.
- Siempre supone un aumento del tiempo de cómputo respecto del algoritmo tradicional sin bolsillo, pues en este último no es necesario hacer una verificación del conjunto de entrenamiento al finalizar la época.
- Está garantizado que el vector del bolsillo es el mejor hasta el momento.
- Una forma de reducir el impacto sobre el tiempo de cómputo sería aplicar únicamente la regla del bolsillo en las últimas épocas.

Una modificación: La Regla del Bolsillo

Pero... ¿para qué sirve la regla del bolsillo?

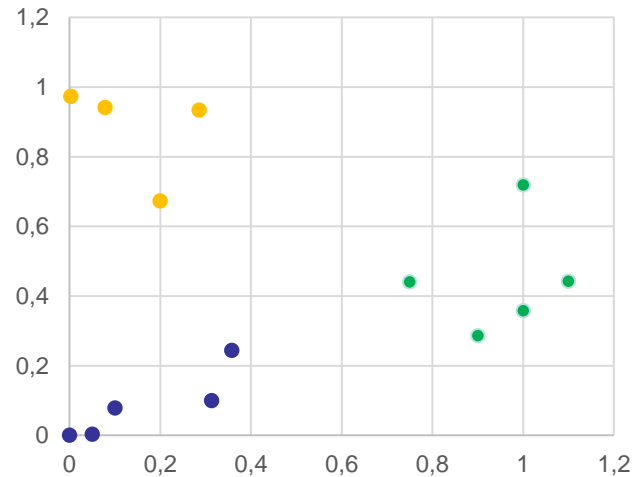
Imaginemos un caso extremo: tenemos que terminar de entrenar el perceptrón en la iteración $k+1$, y estamos en la iteración k . La situación es la de abajo:



¿Con qué vector de pesos te hubieras quedado? ¿con $w(k)$ ó con $w(k+1)$?

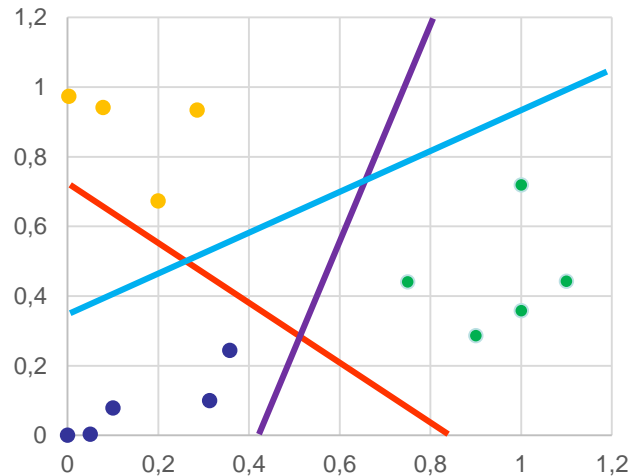
Redes de perceptrones simples

¿Es posible clasificar este conjunto de datos utilizando únicamente perceptrones simples?



Redes de perceptrones simples

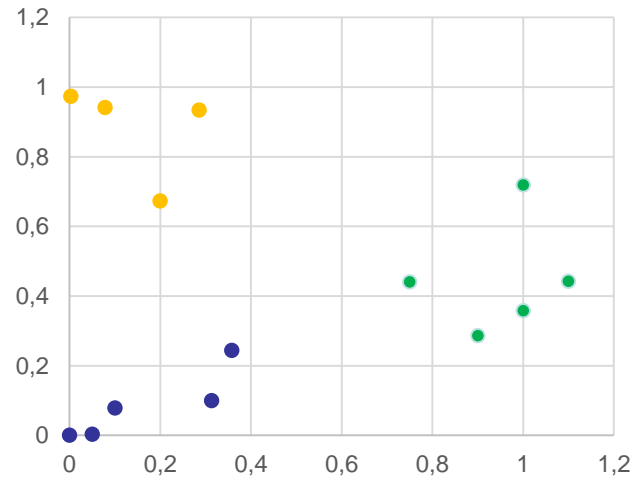
¿Es posible clasificar este conjunto de datos utilizando únicamente perceptrones simples?



Entrenar 3 perceptrones simples, cada uno de ellos especializado en diferenciar una clase de las otras dos. Una vez entrenados, en función del perceptrón que se active, sabremos a qué clase pertenece el patrón de entrada

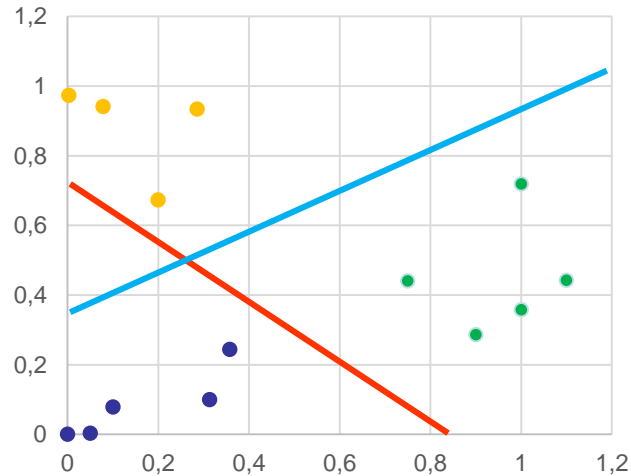
Redes de perceptrones simples

¿Y usando un máximo de dos perceptrones simples?



Redes de perceptrones simples

¿Y usando un máximo de dos perceptrones simples?



Entrenar 2 perceptrones simples, uno de ellos especializado en diferenciar una clase (naranja) de las otras dos y otro especializado en la clase azul. Una vez entrenados, en función del perceptrón que se active sabremos si es azul o naranja, si no se activa ninguno, será verde.

La ADALINA

La ADALINA (también llamada ADALINE), pues corresponde al acrónimo de **AD**aptive **L**inear **NE**uron) o neurona con adaptación lineal que fue introducida por Widrow en 1959. Esta neurona es similar al Perceptrón simple pero utiliza como función de transferencia la **función identidad** en lugar de la función signo. La salida de la ADALINA es simplemente una función lineal de las entradas (ponderadas con los pesos sinápticos):

$$y = \sum_{j=1}^N w_j x_j - \theta$$

$$y = \sum_{j=1}^{N+1} w_j x_j$$

$$\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^p\}$$

$$\{z^1, z^2, \dots, z^p\}$$

$$E = \frac{1}{2} \sum_{k=1}^p (z^k - y(k))^2 = \frac{1}{2} \sum_{k=1}^p \left(z^k - \sum_{j=1}^{N+1} w_j(k) x_j^k \right)^2$$

La ADALINA

Aprendizaje individualizado:

$$E = \frac{1}{2} \sum_{k=1}^p (z^k - y(k))^2 = \frac{1}{2} \sum_{k=1}^p \left(z^k - \sum_{j=1}^{N+1} w_j(k) x_j^k \right)^2$$

$$w_r(k+1) = w_r(k) + \Delta w_r(k)$$

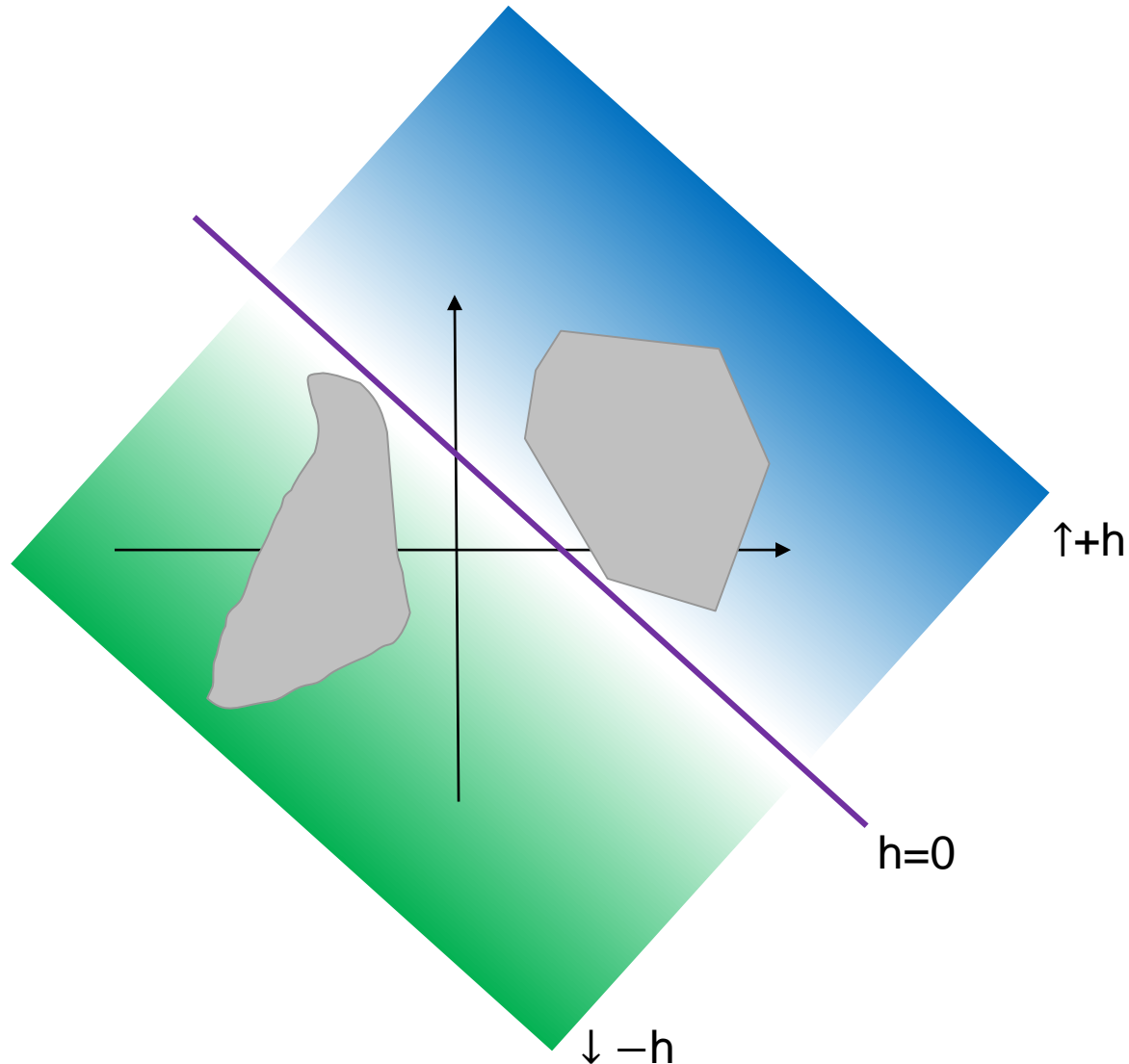
$$\Delta w_r(k) = -\eta \frac{\partial E}{\partial w_r(k)}$$

$$= \eta [z^k - y(k)] x_r^k$$

$$\frac{\partial V^n}{\partial x} = nV^{n-1} \cdot \frac{\partial V}{\partial x}$$

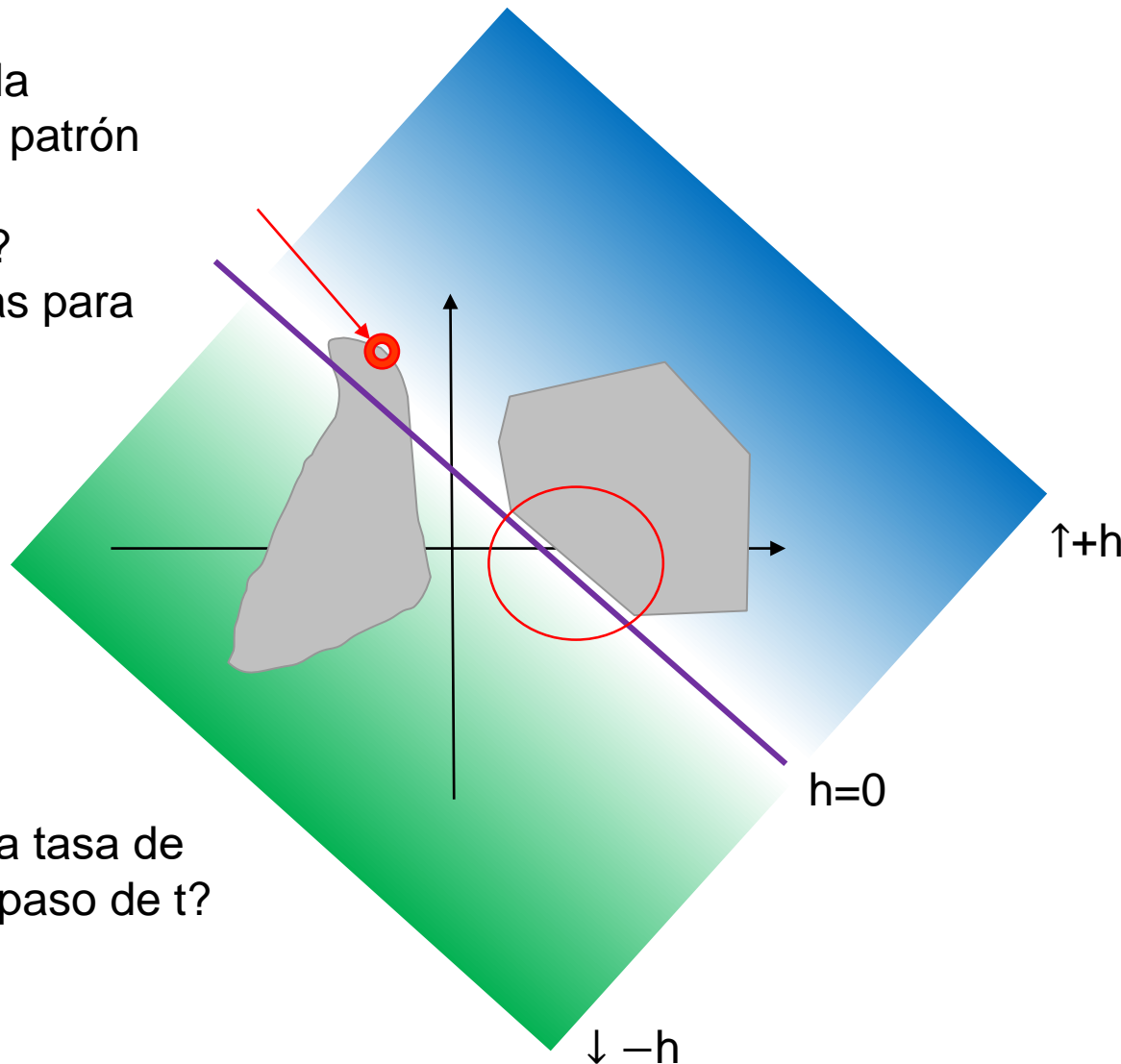
Interpretación gráfica salida de la ADALINA

Interpretación gráfica del potencial/salida de la Adalina



Tasa de aprendizaje no constante aplicada a la ADALINA

- ¿Y si estamos muy cerca de la solución y corregimos con un patrón lejano a $h=0$?
- ¿Cómo cambiarán los pesos?
- ¿Y si ya quedan pocas épocas para terminar?



...¿qué utilidad podría tener una tasa de aprendizaje decreciente con el paso de t ?

La ADALINA

Ejemplo función AND

$$y = \sum_{j=1}^{N+1} w_j x_j$$

$$w_r(k+1) = w_r(k) + \Delta w_r(k)$$

$$\Delta w_r(k) = -\eta \frac{\partial E}{\partial w_r(k)} = \eta [z^k - y(k)] x_r^k$$

→ Realiza 4 actualizaciones partiendo de $w_1 = 0, w_2 = 0, \theta = 0, \eta = 0.1$

k	x1	x2	-1	Z	Y	w1	w2	rho
						0	0	0
1	1	1	-1	1				
2	1	-1	-1	-1				
3	-1	1	-1	-1				
4	-1	-1	-1	-1				

La ADALINA

Ejemplo función AND

$$y = \sum_{j=1}^{N+1} w_j x_j$$

$$w_r(k+1) = w_r(k) + \Delta w_r(k)$$

$$\Delta w_r(k) = -\eta \frac{\partial E}{\partial w_r(k)} = \eta [z^k - y(k)] x_r^k$$

→ Calcula el Error Cuadrático Medio (ECM) que comete con la última configuración en los 4 patrones de entrenamiento

k	x1	x2	-1	Z	Y	w1	w2	rho
						0	0	0
1	1	1	-1	1		0,10	0,10	-0,10
2	1	-1	-1	-1		-0,01	0,21	0,01
3	-1	1	-1	-1		0,11	0,09	0,13
4	-1	-1	-1	-1		0,18	0,16	0,20

La ADALINA

Ejemplo función AND

$$y = \sum_{j=1}^{N+1} w_j x_j$$

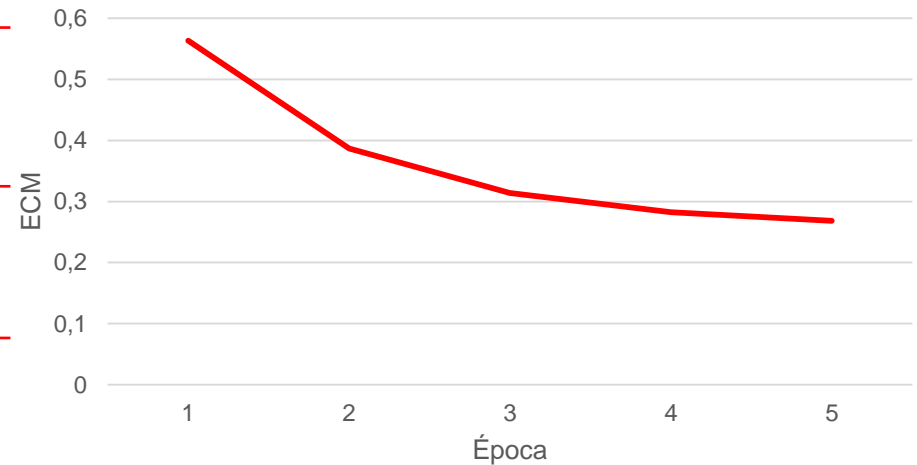
$$w_r(k+1) = w_r(k) + \Delta w_r(k)$$

t	x1	x2	-1	Z	Y	w1	w2	rho
						0	0	0
1	1	1	-1	1	0,00	0,10	0,10	-0,10
2	1	-1	-1	-1	0,10	-0,01	0,21	0,01
3	-1	1	-1	-1	0,21	0,11	0,09	0,13
4	-1	-1	-1	-1	-0,33	0,18	0,16	0,20
5	1	1	-1	1	0,14	0,26	0,24	0,11
6	1	-1	-1	-1	-0,09	0,17	0,33	0,20
7	-1	1	-1	-1	-0,04	0,27	0,24	0,30
8	-1	-1	-1	-1	-0,80	0,29	0,26	0,32
9	1	1	-1	1	0,23	0,37	0,33	0,24
10	1	-1	-1	-1	-0,21	0,29	0,41	0,32
11	-1	1	-1	-1	-0,19	0,37	0,33	0,40
12	-1	-1	-1	-1	-1,10	0,36	0,32	0,39
13	1	1	-1	1	0,29	0,43	0,39	0,32
14	1	-1	-1	-1	-0,28	0,36	0,47	0,39
15	-1	1	-1	-1	-0,28	0,43	0,39	0,46
16	-1	-1	-1	-1	-1,28	0,40	0,37	0,43
17	1	1	-1	1	0,33	0,47	0,43	0,37
18	1	-1	-1	-1	-0,33	0,40	0,50	0,43
19	-1	1	-1	-1	-0,34	0,47	0,43	0,50
20	-1	-1	-1	-1	-1,40	0,43	0,39	0,46

1ª época

ECM

AND



La ADALINA

Ejemplo función XOR

$$y = \sum_{j=1}^{N+1} w_j x_j$$

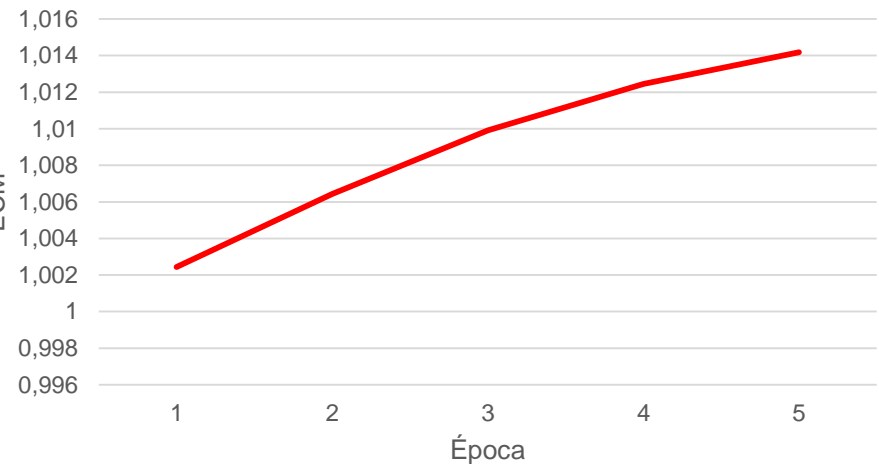
$$w_r(k+1) = w_r(k) + \Delta w_r(k)$$

t	x1	x2	-1	Z	Y	w1	w2	rho
						0	0	0
1	1	1	-1	-1	0,00	-0,10	-0,10	0,10
2	1	-1	-1	1	-0,10	0,01	-0,21	-0,01
3	-1	1	-1	1	-0,21	-0,11	-0,09	-0,13
4	-1	-1	-1	-1	0,33	0,02	0,04	0,00
5	1	1	-1	-1	0,06	-0,08	-0,06	0,11
6	1	-1	-1	1	-0,13	0,03	-0,18	0,00
7	-1	1	-1	1	-0,20	-0,09	-0,06	-0,12
8	-1	-1	-1	-1	0,27	0,04	0,07	0,00
9	1	1	-1	-1	0,10	-0,07	-0,04	0,11
10	1	-1	-1	1	-0,15	0,04	-0,15	0,00
11	-1	1	-1	1	-0,19	-0,08	-0,03	-0,12
12	-1	-1	-1	-1	0,23	0,04	0,09	0,00
13	1	1	-1	-1	0,13	-0,07	-0,02	0,12
14	1	-1	-1	1	-0,16	0,05	-0,14	0,00
15	-1	1	-1	1	-0,19	-0,07	-0,02	-0,12
16	-1	-1	-1	-1	0,21	0,05	0,10	0,00
17	1	1	-1	-1	0,15	-0,06	-0,01	0,12
18	1	-1	-1	1	-0,17	0,05	-0,13	0,00
19	-1	1	-1	1	-0,18	-0,07	-0,01	-0,12
20	-1	-1	-1	-1	0,20	0,05	0,11	0,00

1ª época

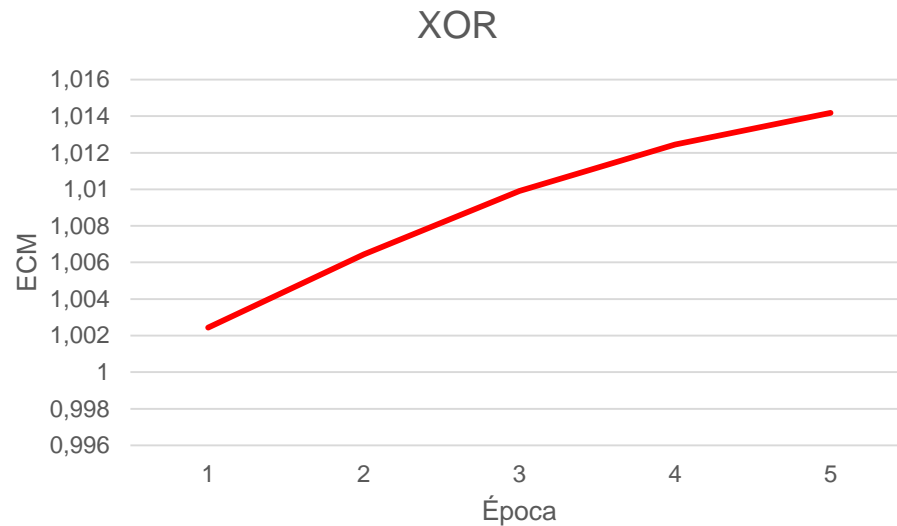
ECM

XOR

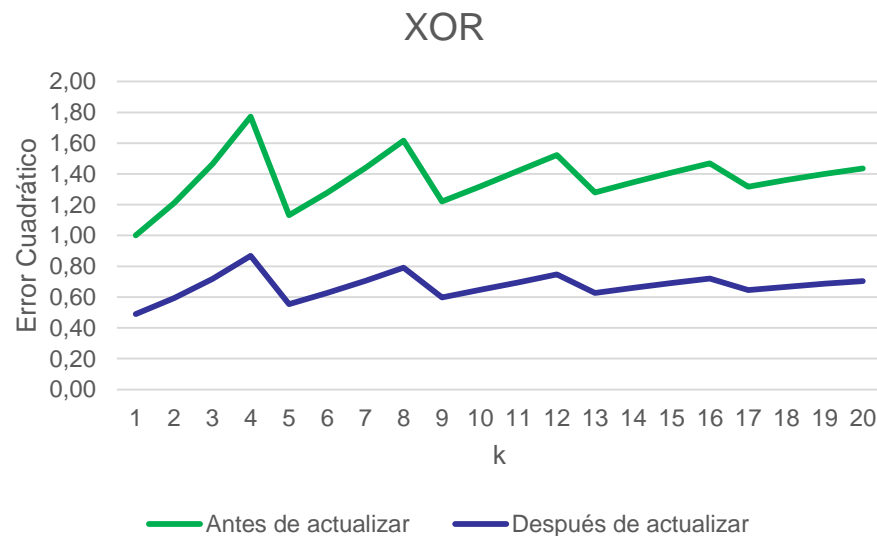


La ADALINA

Ejemplo función XOR



¿Funciona el método de actualización de pesos?



Calculado según la salida para un mismo patrón, "antes" y "después" de actualizar los pesos

La ADALINA

Aprendizaje por lotes:

$$E = \frac{1}{2p} \sum_{k=1}^p (z^k - y(k))^2 = \frac{1}{2p} \sum_{k=1}^p \left(z^k - \sum_{j=1}^{N+1} w_j x_j^k \right)^2$$

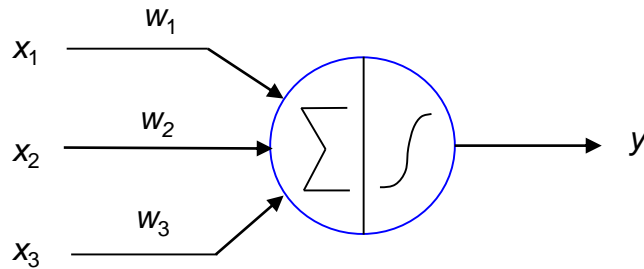
$$w_r(k+1) = w_r(k) + \Delta w_r(k)$$

$$\Delta w_r(k) = -\eta \frac{\partial E}{\partial w_r(k)}$$

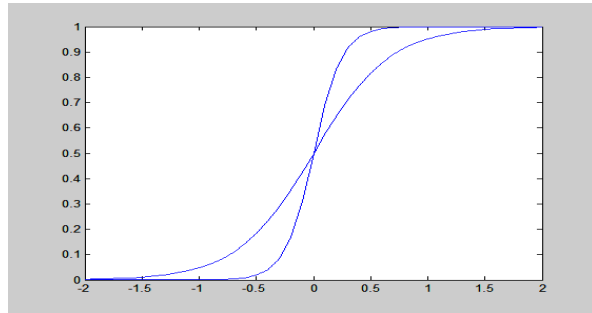
$$= \eta \frac{1}{p} \sum_{k=1}^p [z^k - y(k)] x_j^k$$

Neuronas con salida continua:

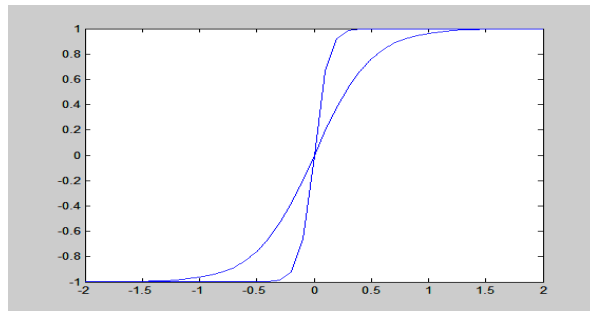
Regla de aprendizaje de Widrow-Hoff



$$y = g\left(\sum_{j=1}^N w_j x_j\right)$$



$$g(x) \equiv \frac{1}{1 + \exp(-2\beta x)}$$



$$g(x) = \tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}}$$

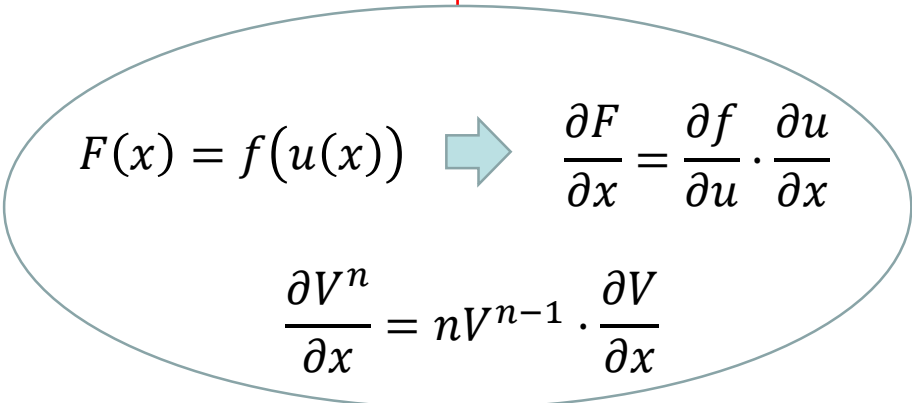
Neuronas con salida continua:

Regla de aprendizaje de Widrow-Hoff

$$E = \frac{1}{2} \sum_{k=1}^p (z^k - y(k))^2 = \frac{1}{2} \sum_{k=1}^p \left(z^k - g\left(\sum_{j=1}^{N+1} w_j(k) x_j^k\right) \right)^2$$

$$\Delta w_j(k) = -\eta \frac{\partial E}{\partial w_j(k)}$$

$$= \eta [z^k - y(k)] g'(h) x_j^k$$


$$F(x) = f(u(x)) \Rightarrow \frac{\partial F}{\partial x} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial x}$$

$$\frac{\partial V^n}{\partial x} = nV^{n-1} \cdot \frac{\partial V}{\partial x}$$

Neuronas con salida continua:

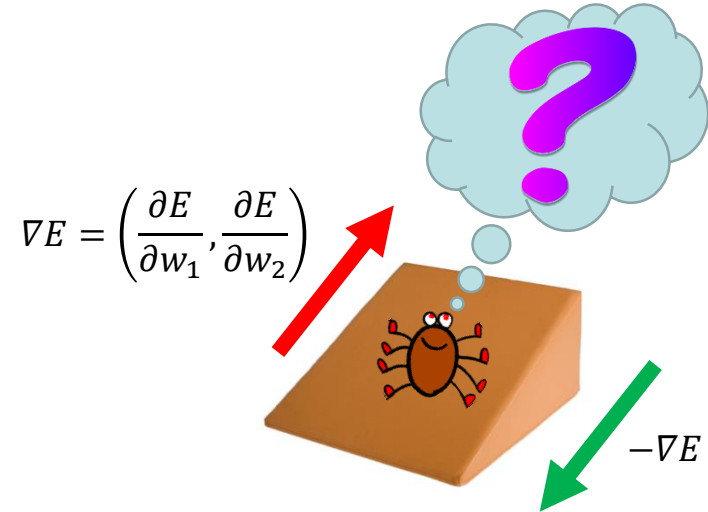
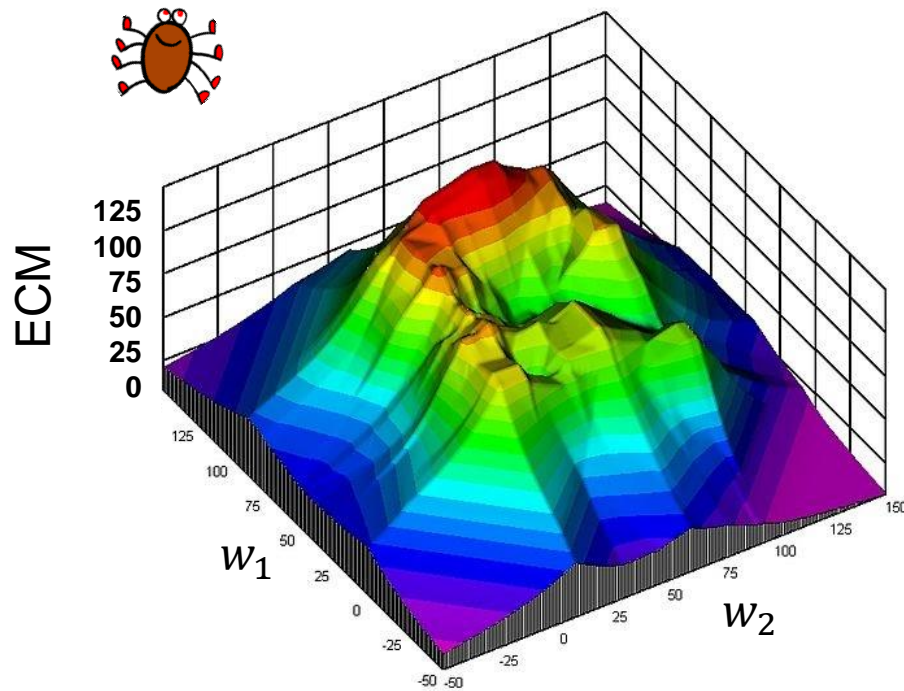
Regla de aprendizaje por lotes de Widrow-Hoff

$$E = \frac{1}{2p} \sum_{k=1}^p (z^k - y(k))^2 = \frac{1}{2p} \sum_{k=1}^p \left(z^k - g\left(\sum_{j=1}^{N+1} w_j x_j^k\right) \right)^2$$

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

$$= \eta \frac{1}{p} \sum_{k=1}^p [z^k - y(k)] g'(h) x_j^k$$

Interpretación del método del descenso por el gradiente



Interpretación del método del descenso por el gradiente

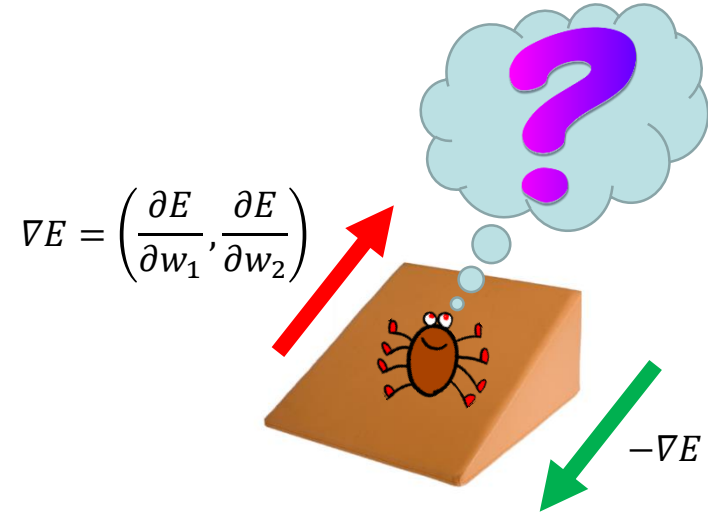
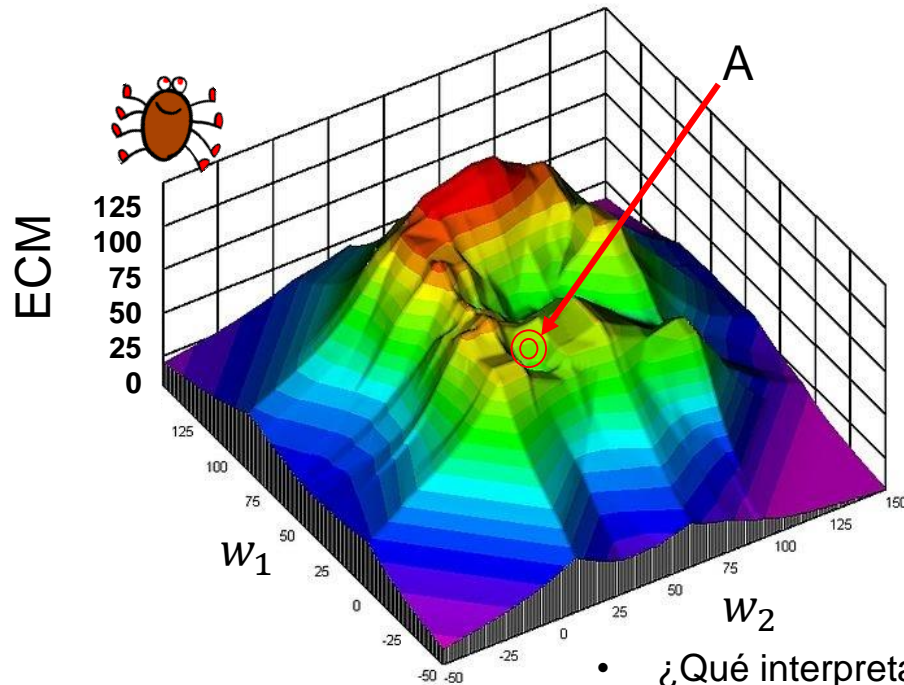
Supongamos que somos una pulga infinitesimal. La función de error (E) va a ser como un plano topográfico (con las alturas) de una región del mundo, nos podemos mover en w_1 y w_2 que son los pesos a optimizar y las variables de las que depende nuestra función de error (o plano topográfico). Queremos descender de altura lo más rápido que se nos ocurra, porque tenemos mal de altura, no podemos tomar decisiones complicadas... ¡somos una pulga!, aunque tenemos un don innato para derivar funciones.

Si calculamos el **gradiente en un punto**, nos da el **vector de máximo crecimiento** del plano topográfico (hacia donde crece más el terreno). Si somos una pulga de tamaño “infinitesimal”, para nosotros nuestro entorno en el que debemos decidir hacia donde mover la patita, será un plano inclinado, si crees que no lo es, es que no has reducido suficiente el tamaño de la pulga, sigue encogiéndola. (Nota: algo similar pasa cuando vemos que nuestro horizonte es plano, mientras que el planeta no lo es, es una cuestión de escala).

Por el cálculo y por sentido común, si queremos descender de altura (de error) en ese plano inclinado, debemos movernos en el **sentido contrario** al vector de máximo crecimiento. Es decir, desde donde está la pulga existirá un vector imaginario paralelo a la pendiente ascendente (gradiente), nos moveremos justo en el sentido contrario para así descender.

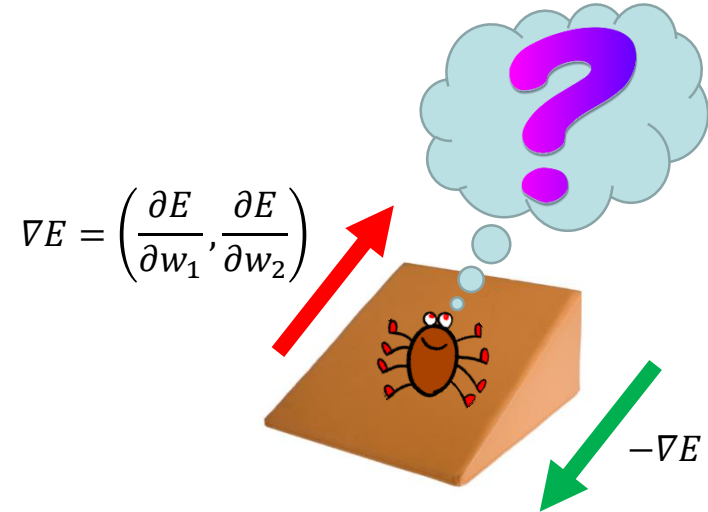
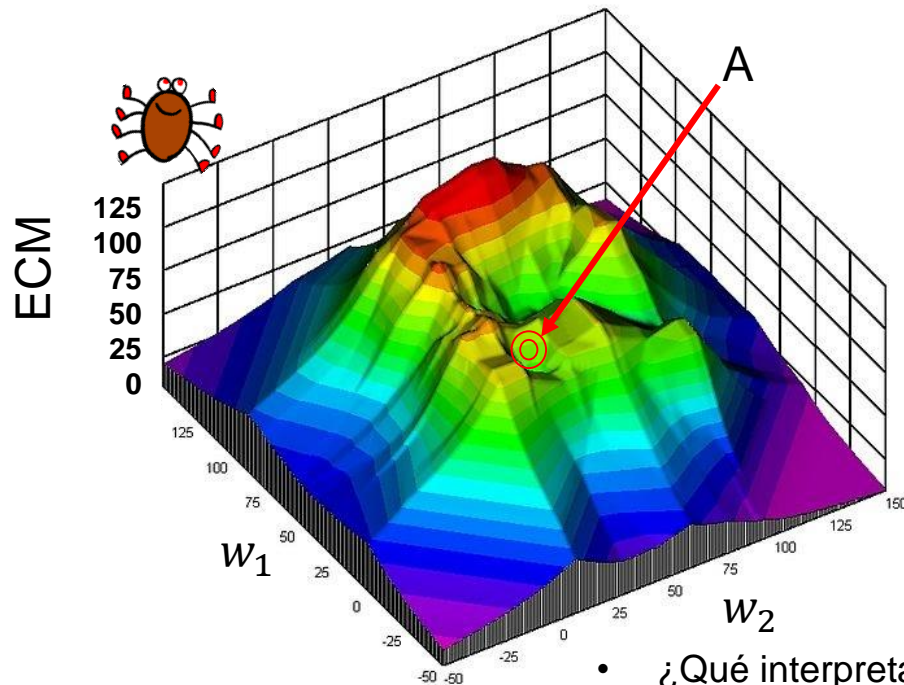
Esto no deja de ser un algoritmo ávido, es posible que al descender, caigamos en un mínimo local del error (un valle del plano) y ya no podamos salir, es decir, al iterar no se reduce el error. Quizá si hubiéramos escogido otro camino hubiéramos llegado a un mínimo más pequeño o incluso al error cero, mínimo global de la función E (nivel del mar).

Interpretación del método del descenso por el gradiente



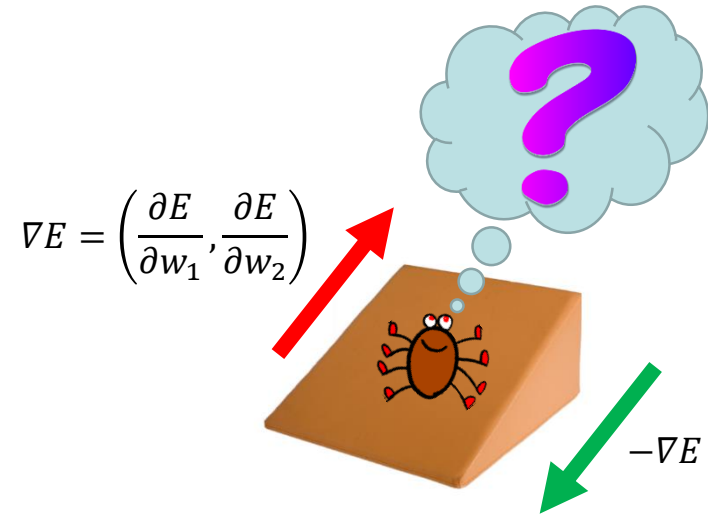
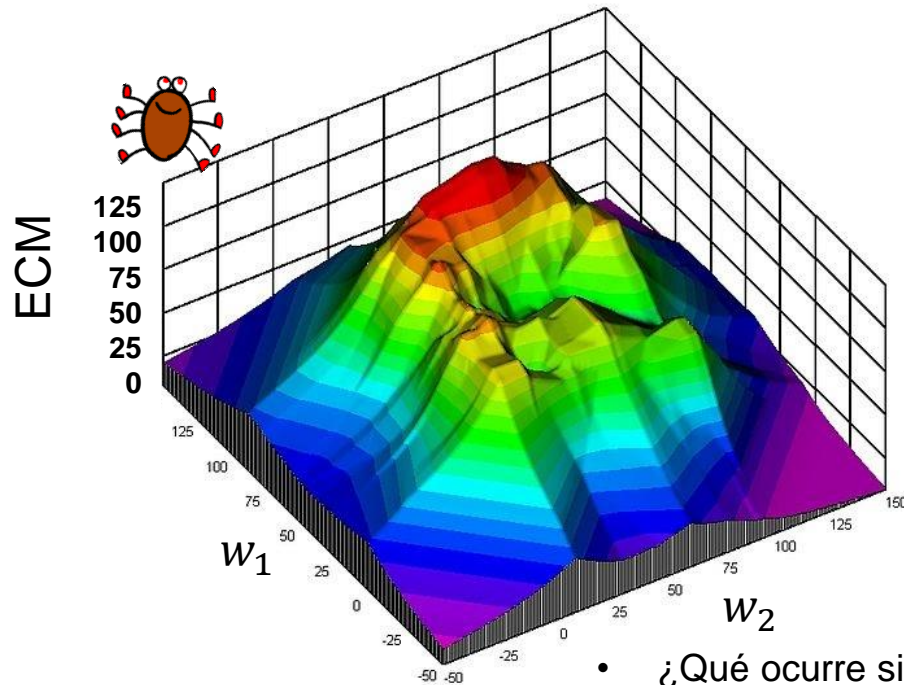
- ¿Qué interpretación tiene una inicialización de pesos aleatoria?
- ¿Qué ocurre si la pulga está en el punto A?
- ¿Por qué no tendría sentido bajar en vertical?

Interpretación del método del descenso por el gradiente



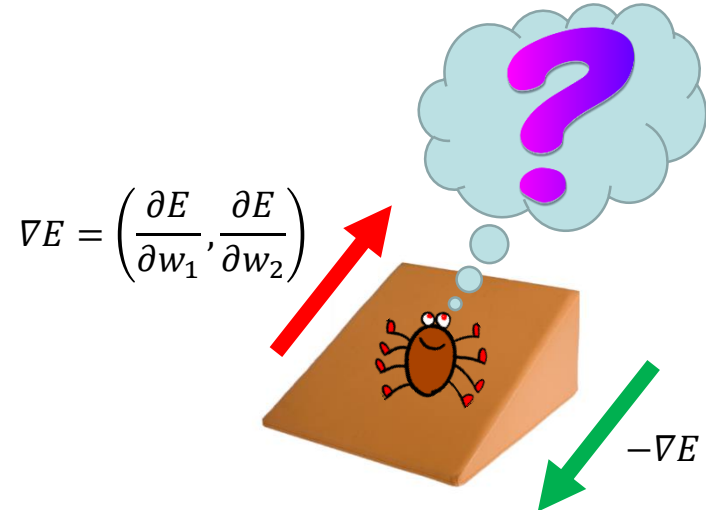
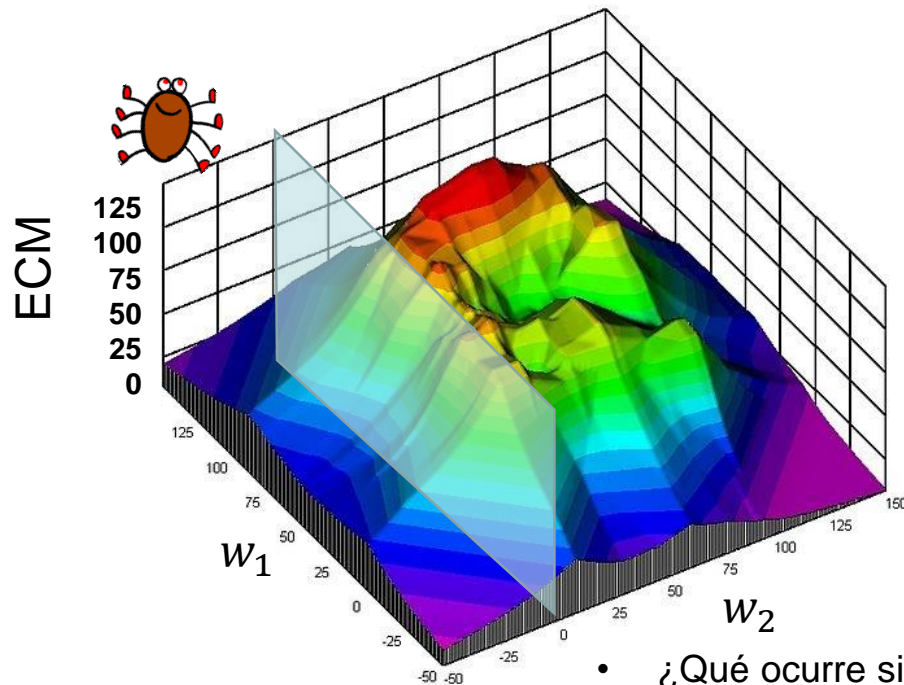
- ¿Qué interpretación tiene una inicialización de pesos aleatoria?
Que la pulga parte de un punto del mapa aleatorio, puede que caiga en un valle.
- ¿Qué ocurre si la pulga está en el punto A?
Que corremos riesgo de caer en un mínimo local de la función E .
- ¿Por qué no tendría sentido bajar en vertical?
Porque no olvidemos que a cada par w_1 y w_2 le corresponde un valor de E , sería como si nos saliéramos de la función que estamos intentando ajustar, la función E nos indica lo "bien o mal" que representamos la función a ajustar.

Interpretación del método del descenso por el gradiente



- ¿Qué ocurre si reducimos la entrada a una sola dimensión?, por ejemplo eliminamos w_2

Interpretación del método del descenso por el gradiente

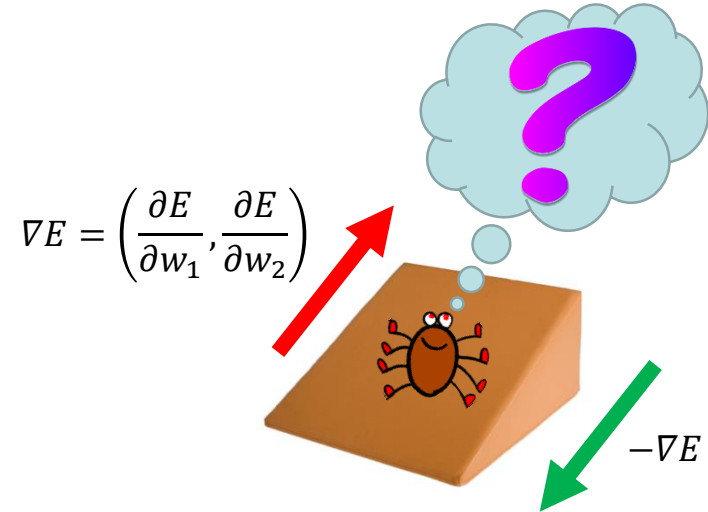
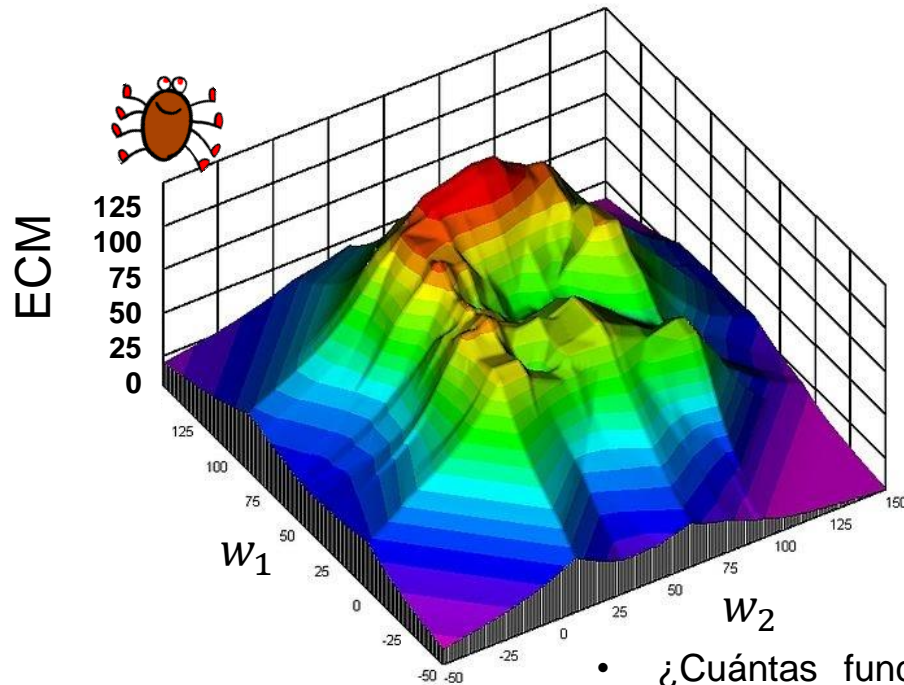


- ¿Qué ocurre si reducimos la entrada a una sola dimensión?, por ejemplo eliminamos w_2

Que nos moveríamos en un corte del terreno donde $w_2=0$. En este caso perderíamos algunas soluciones mejores y por lo tanto nos sería imposible llegar a ajustar tan bien la función objetivo. Lo bueno es que los cálculos serían más rápidos, sólo habría que ajustar w_1 .

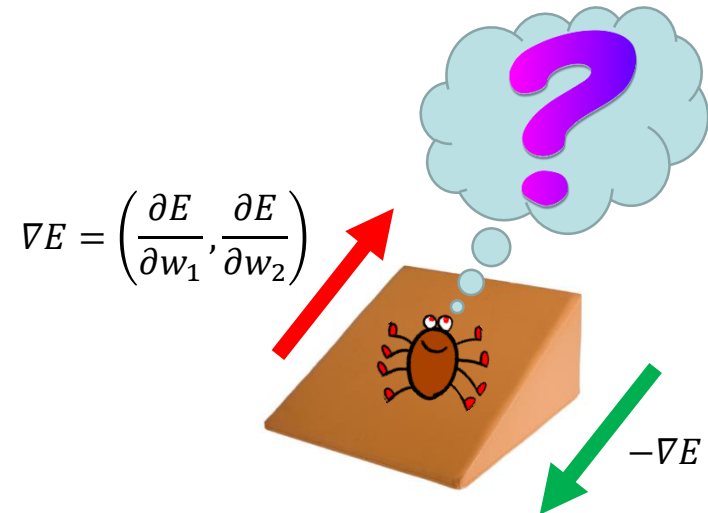
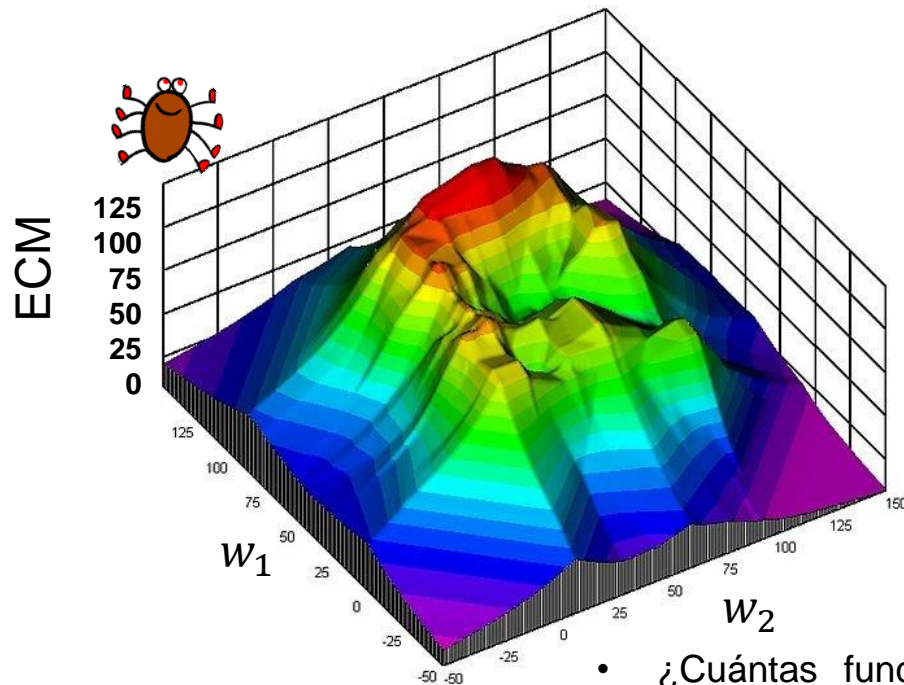
*Esto no es más que una aproximación muy simplificada del concepto de PCA o **Análisis de Componentes Principales**.*

Interpretación del método del descenso por el gradiente



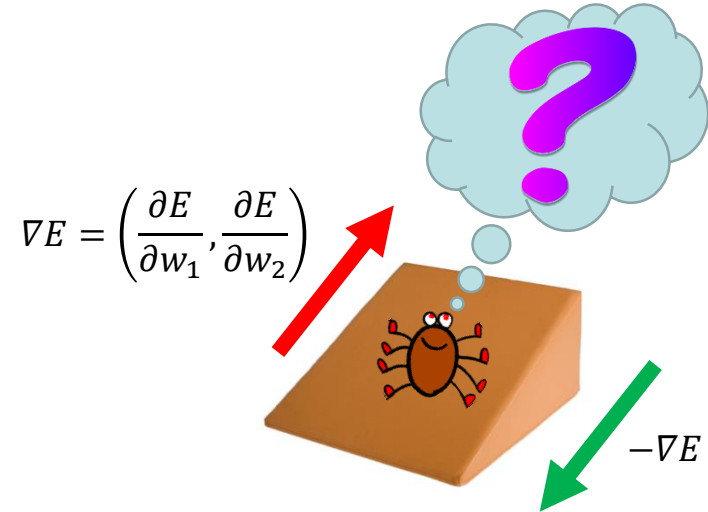
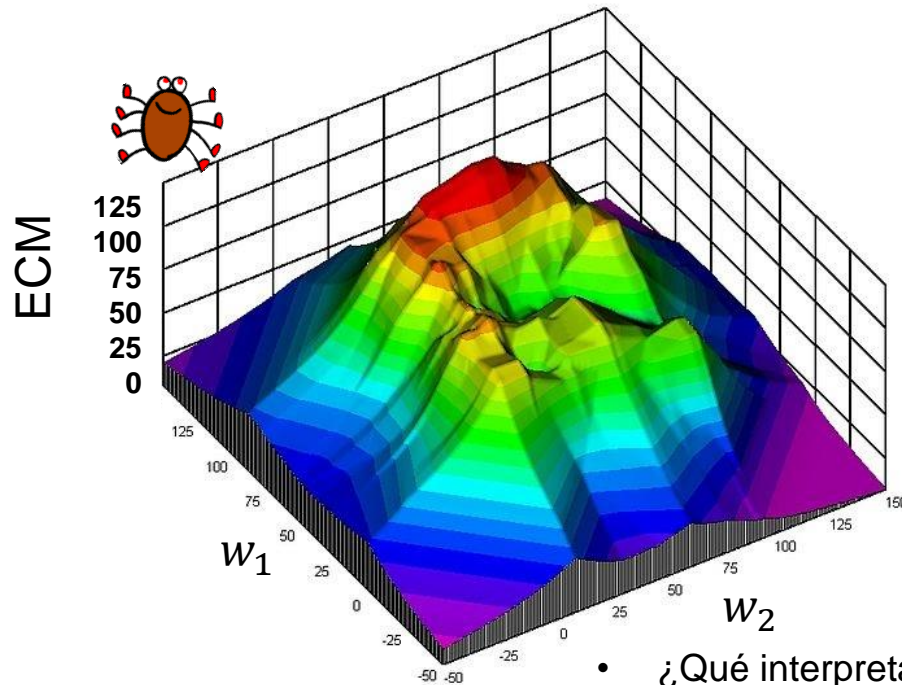
- ¿Cuántas funciones E o mapas topográficos existirían en el caso secuencial?
- Entonces si estamos en un mínimo local de E , ¿deberíamos dejar ya de iterar?

Interpretación del método del descenso por el gradiente



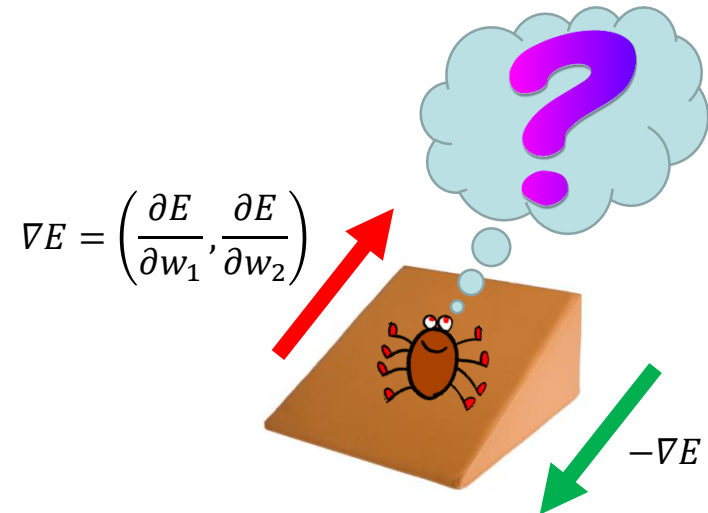
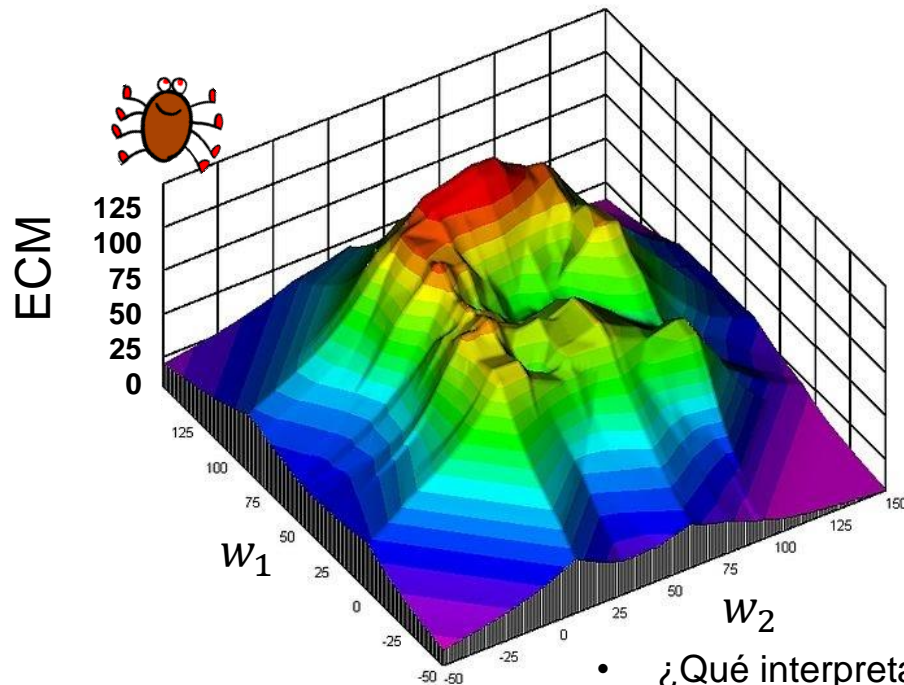
- ¿Cuántas funciones E o mapas topográficos existirían en el caso secuencial?
Uno por cada patrón de entrenamiento.
- Entonces si estamos en un mínimo local de E , ¿deberíamos dejar ya de iterar?
No, podría existir un movimiento en otro mapa E de otro patrón que haga mejorar el ajuste de la función. Incluso podría sacarnos de ese mínimo en el que habíamos caído.

Interpretación del método del descenso por el gradiente



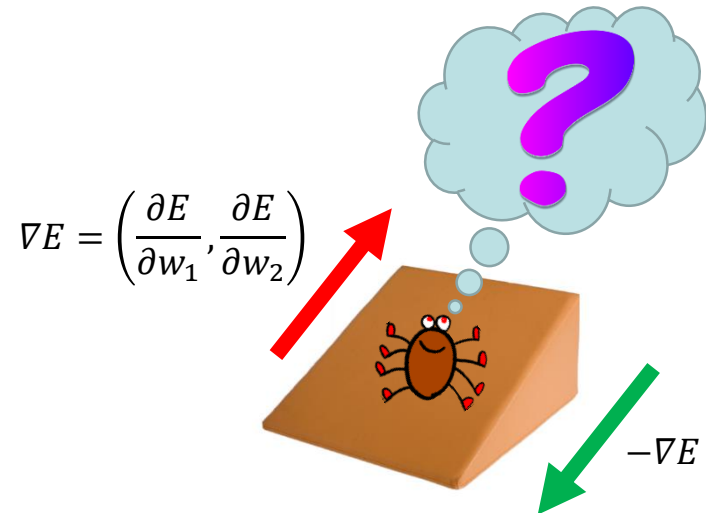
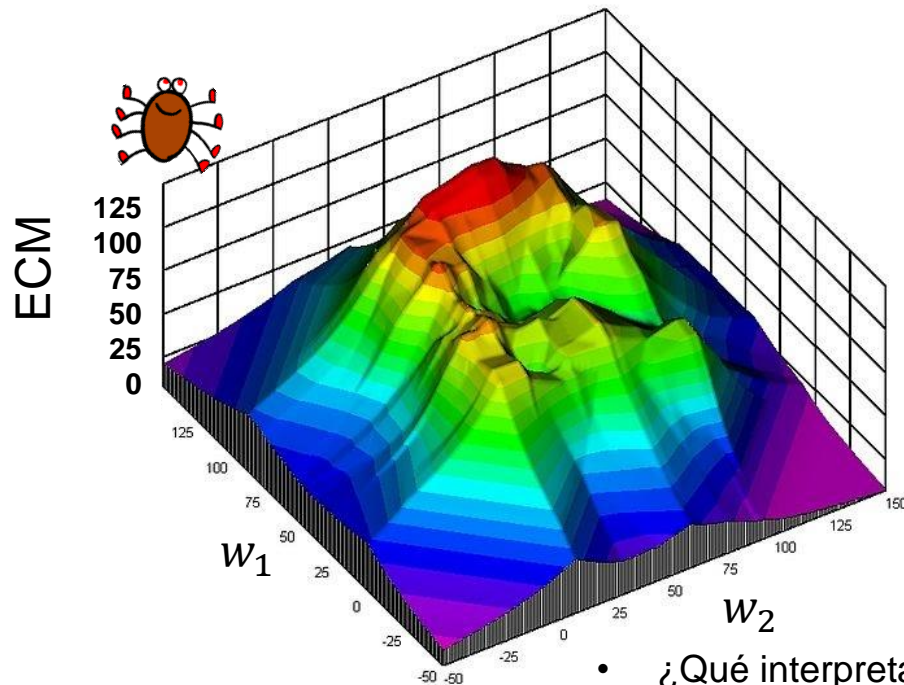
- ¿Qué interpretación gráfica tiene la actualización de pesos por lotes?
- ¿Por qué no calculamos el mapa completo y buscamos la mejor ruta?

Interpretación del método del descenso por el gradiente



- ¿Qué interpretación gráfica tiene la actualización de pesos por lotes?
Que calculamos los gradientes (sentido inverso) en cada mapa y nos movemos en la dirección media de todos esos movimientos.
- ¿Por qué no calculamos el mapa completo y buscamos la mejor ruta?
Porque tardaríamos demasiado tiempo en calcular todos los puntos, no deja de ser un algoritmo ávido. Es más, ¿deberíamos calcular un mapa completo para cada patrón!

Interpretación del método del descenso por el gradiente



- ¿Qué interpretación gráfica tiene la actualización de pesos por lotes?
Que calculamos los gradientes (sentido inverso) en cada mapa y nos movemos en la dirección media de todos esos movimientos.
- ¿Por qué no calculamos el mapa completo y buscamos la mejor ruta?
Porque tardaríamos demasiado tiempo en calcular todos los puntos, no deja de ser un algoritmo ávido. Es más, ¡deberíamos calcular un mapa completo para cada patrón!



¡LA PULGA MORIRÍA!