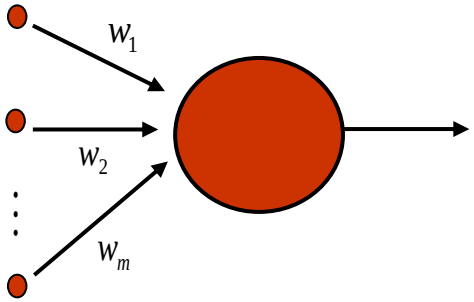
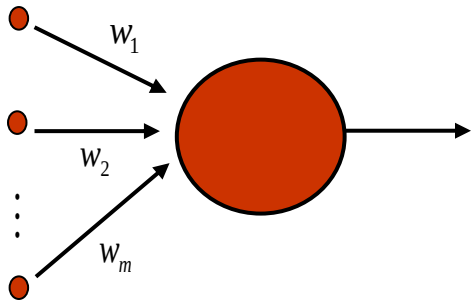
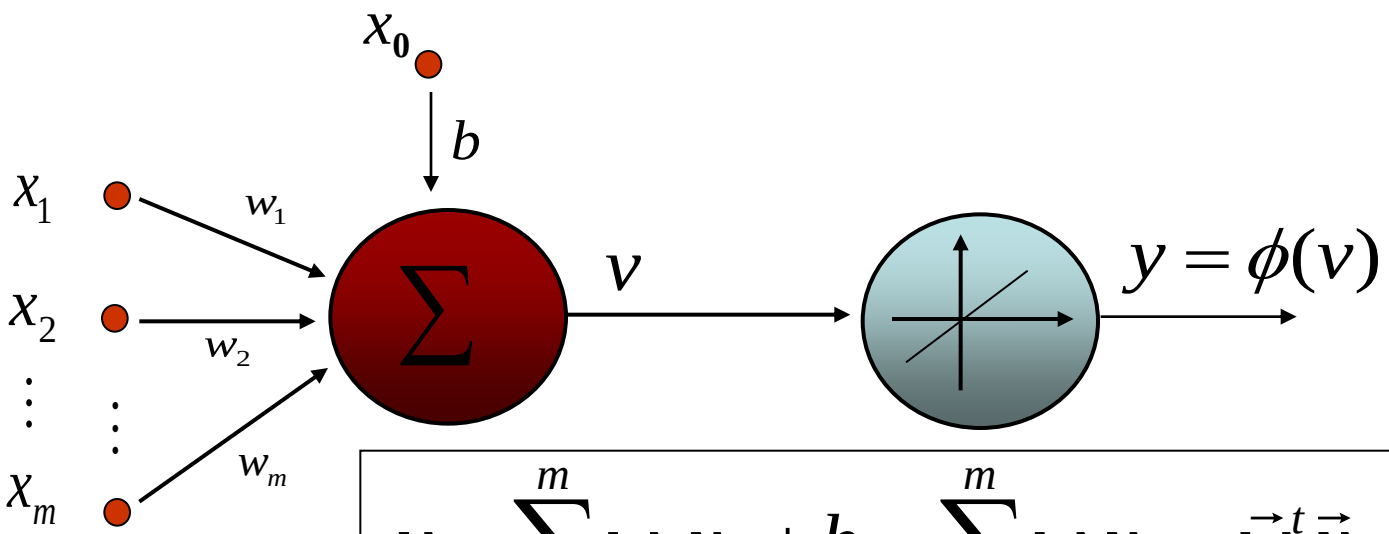
ALGUNOS MODELOS DE UNA NEURONA



ADALINE
(ADAptive Linear Element

Widrow y Hoff (1960)





$$v = \sum_{j=1}^{m} w_j x_j + b = \sum_{i=0}^{m} w_i x_i = \vec{w}^t \vec{x}$$

con $\vec{w} = \begin{bmatrix} b & w_1 & w_2 & \cdots & w_m \end{bmatrix}^t$

$\vec{x} = \begin{bmatrix} 1 & x_1 & x_2 & \cdots & x_m \end{bmatrix}^t$

$\Rightarrow \quad y = v$

Notación:

- D **es el conjunto de datos** $\{(x(k), d(k)), k = 1,,,,,, L\}$**, con** $x(k)$ **un vector**

Queremos minimizar el error cuadrático medio

$$E(\vec{w}) = \frac{1}{2} \sum_{x(k) \in D} (d(k) - y(k))^2$$

Esta función por definición es concava, diferenciable y tiene un mínimo.

De la estadística sabemos que la solución de mínimos cuadrado es:

$$W = \left(X^t X\right)^{-1} X^t d$$

Pseudo-inversa

con

$$X = \begin{pmatrix} x_1 & x_1 & \cdots & x_1 \\ x_2 & x_2 & \cdots & x_2 \\ \vdots & \vdots & \ddots & \vdots \\ x_N & x_N & \cdots & x_N \end{pmatrix}$$

Esta forma tiene un costo computacional muy alto y puede ser prohibitiva cuando tenemos tamaños de datos muy grandes.

Podemos tener como en el caso del perceptrón varios ADALINEs juntos y en este caso se le denomina MADALINE. Vamos a centrarnos en una sola neurona.

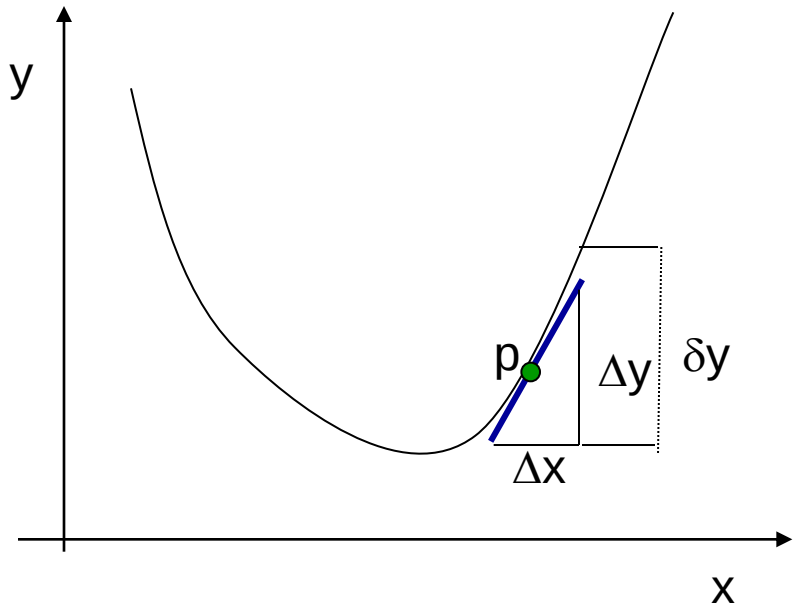
El Entrenamiento

La regla de actualización de pesos es la LMS (least mean square algorithm) que se basa en el método de descenso de gradiente para minimizar la diferencia entre la salida deseada y la actual (entrenamiento supervisado).

El método de descenso de gradiente forma parte del conjunto de métodos de optimización sin restricciones.

En el caso del perceptrón se requería que el problema fuera linealmente separable. Aqui buscamos un clasificador lineal que sea óptimo en algún sentido (minimizando una función de costos).

La idea es producir una secuencia de pesos W(n) tal que en cada iteración la función de costos disminuya, esto se logra con correciones al vector en dirección opuesta al gradiente (que resulta ser la de mayor descenso)



La pendiente de la función es el gradiente de la tangente a la curva en ese punto

P= Δy/Δx

en 1-D: Si inicio en a, para γ suficientemente pequeño, $b = a + \gamma y'(a)$ **es tal que**

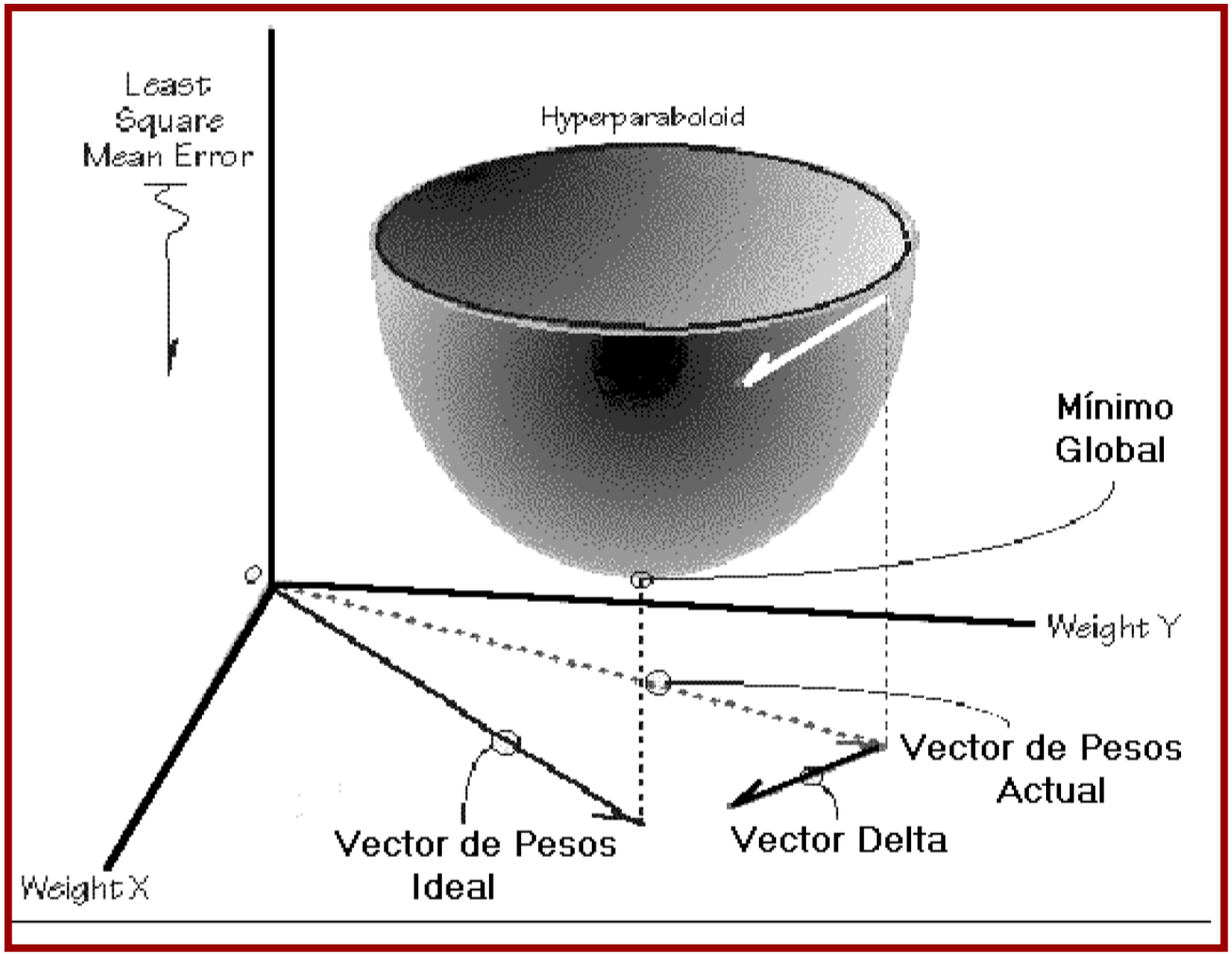
$$y(b) = y\big(a + \underbrace{\gamma\, y'(a)}_{h}\big) = y(a) + hy'(a) + O(h^2) = y(a) + \gamma\,(y'(a))^2 > y(a)$$

Hay que caminar en dirección opuesta!!

Dicho de otra manera: En 1D para encontrar un mínimo local ,

- **Si $f'(x) > 0$, me muevo hacia la izquierda,**
- **Si $f'(x) < 0$, me muevo hacia la derecha**
- **Si $f'(x) = 0$, me quedo quieta**

En espacios de dimensión >1, nos movemos en la dirección opuesta al gradiente.


Least
Square
Mean Error
Hyperparaboloid
Mínimo
Global
Weight Y
Vector de Pesos
Actual
Vector Delta
Vector de Pesos
Ideal
Weight X

El gradiente se calcula como:

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_m} \right]$$

En el algoritmo del descenso del gradiente tomamos un paso en dirección de máximo descenso que es menos el gradiente

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

En cada componente

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

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\left[\frac{1}{2}\sum_{x(k)\in D}(d(k)-y(k))^2\right]$$

$$= \frac{1}{2}\sum_{x(k)\in D}\frac{\partial}{\partial w_i}(d(k)-y(k))^2$$

$$= \frac{1}{2}\sum_{x(k)\in D}2(d(k)-y(k))\frac{\partial}{\partial w_i}(d(k)-y(k))$$

$$= \sum_{x(k)\in D}(d(k)-y(k))\frac{\partial}{\partial w_i}(d(k)-\vec{w}x(k))$$

$$= \sum_{x(k)\in D}-(d(k)-y(k))x_i(k)$$

Por lo tanto la regla de actualización es

$$\Delta w_i = \eta \sum_{x(k) \in D} (d(k) - y(k)) x_i(k)$$

El método del gradiente garantiza convergencia hacia 0 cuando los pesos sinápticos aproximan la relación de entrada y salida. La convergencia el asintótica, mientras que en el caso del perceptrón sabemos que converge en un número finito de pasos.

OJO!!!!

Hay dos tipos de actualizaciones distintas que inducen a tipos de convergencia diferentes: Actualización en lote y actualización contínua

Actualización por lotes (Batch update)

Repetir hasta satisfacer la condición de parada

- Buscar el gradiente del error $\nabla E_D(\vec{w})$
- $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D(\vec{w})$

Actualización continua (on-line update)

Repetir hasta satisfacer la condición de parada

- Para cada par (x,d) en D

** Calcule $\nabla E_d(\vec{w}) = (d(k) - y(k))x(k)$

** $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d(\vec{w})$

En actualización continua, usamos como función de error el error cometido en la n-ésima iteración, mientras que en el descenso de gradiente usamos la suma de los errores.

Algoritmo

1) Iniciar los pesos en valores aleatorios pequeños

2) Poner E=0

3) Para cada w_i, inicializar $\Delta w_i=0$

4) Para cada (x,d)en D

- $\delta = d - y(x)$
- $E = E + \delta^2$
- $\Delta w_i = \Delta w_i + \eta \delta x_i$, **para i=1….m**

5) Para cada w_i actualizar $w_i=w_i+\Delta w_i$

6) Si no se verifica la condición de parada, continuar al paso 2)

Este algoritmo actualiza los pesos al final de cada época en vez de continuamente con cada presentación de los estimulos, la convergencia está garantizada por la convergencia de descenso de gradiente.

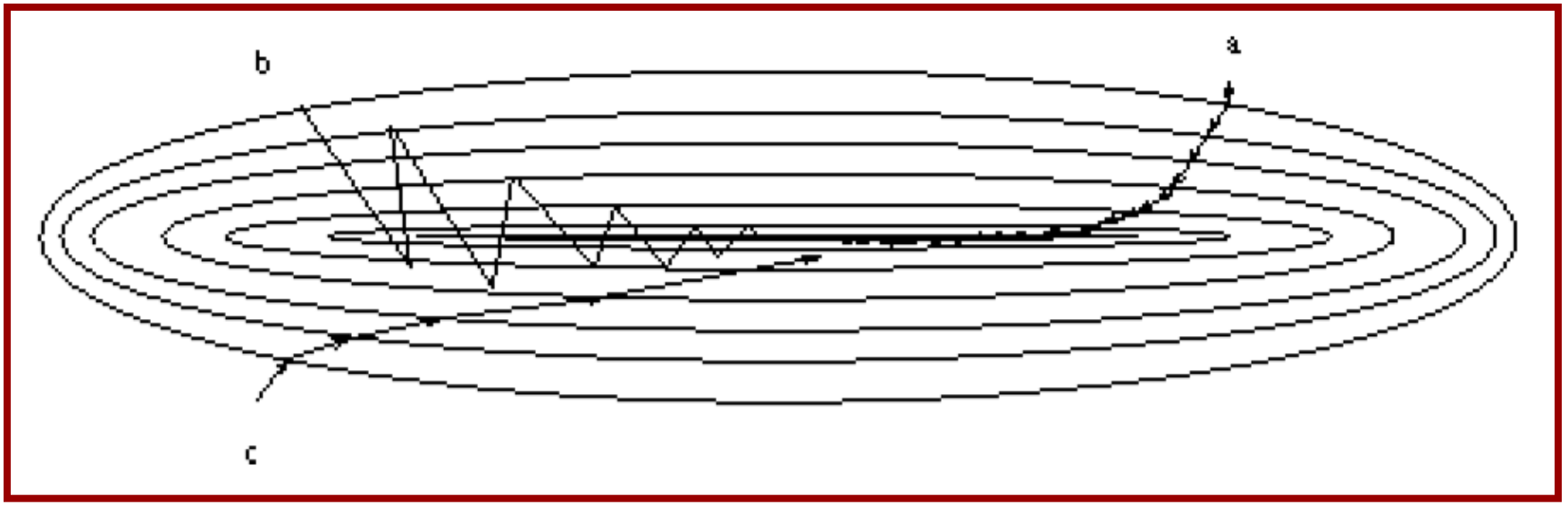
Sin embargo el LMS que se presenta en el Haykin es la actualización continua

1) Iniciar los pesos en valores aleatorios pequeños

2) Para cada (x,d)en D

- $\delta = d - y(x)$
- $w_i = w_i + \eta\delta x_i$, **para i=1….m**

propiedades	On-line	batch
convergencia	**Los pesos siguen una trayectoria "aleatoria" (stochastic gradient descent). A medida que aumenta el número de iteraciones los pesos se comportan como un movimiento Browniano.**	**Converge al mínimo lentamente (depende de η). Si η es suf. pequeño los pesos siguen una trayectoria suave.**
aproximaciones	Aproxima la actualización por lotes (batch) cuando η es pequeño	
evaluaciones		Mas eficiente



α) η **pequeña**

β) η **alta**

Se puede demostrar que LMS (contínua) converge si $0 < \eta < (2/\lambda_{max})$ donde λ_{max} es el mayor autovalor de la matriz de correlación para las entradas.

Un estimado esta dado por $0 < \eta < (2/tr(R_x))$, donde $tr(R_x)$ es la suma cuadrada de los estimulos.

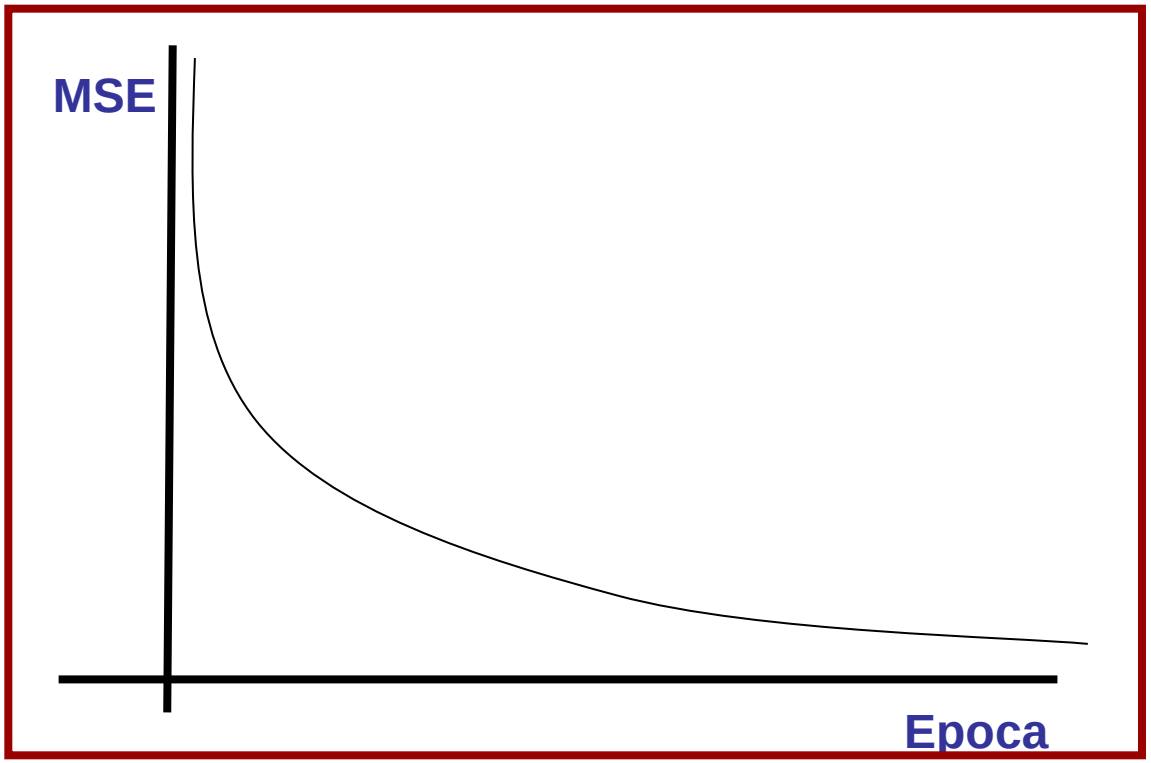
$$R_x = E[x * x^t]$$

A la actualización contínua se le puede mejorar su convergencia si se utilizan parámetros adaptativos

- $\eta = \eta_0$
- $\eta(n) = c/n$ (aproximación estocástica)

$$\eta(n) = \eta_0/(1 + n/\tau)$$

Si hemos elegido bien al parámetro de aprendizaje, en cada iteración vamos a acercarnos más hacia un mínimo (en nuestro caso global), por lo que el error en cada época debería ser menor que la anterior.



FAQ:

Cómo es la curva de aprendizaje del perceptrón?

Porqué no es monótona si la regla de actualización es la misma?

1) Clasificación. Recordar que un ADALINE busca un hiperplano. En esta aplicación es el que separa. El error cometido no será 0, por lo que típicamente la condición de parada debe cambiar.

1) Diferencia entre errores en épocas sucesivas sea pequeña
2) Número máximo de épocas.

2) Regresión lineal. Se busca un modelo de la forma:

$$Y = a + b_1x_1 + b_2x_2 + b_3 x_3 + ...+ b_n x_n$$

3) Modelos autorregresivos. Se busca un modelo de la forma:

$$Y = a + b_1x(n) + b_2x(n-1) + b_3 x(n-2) + ...+ b_k x(n-k+1)$$

4) Aproximación polinómica. Se busca un modelo de la forma:

$$Y = a + b_1x + b_2x^2 + b_3 x^3 + ...+ b_k x^k$$

1. **Definición de red neural, vinculación con cerebro humano.**
2. **Paradigmas del aprendizaje.**
3. **Definimos una red neural artificial .**
4. **Modelos de una neurona:**
    1. **Perceptrón**
        1. **Neurona no–lineal.**
        2. **Clasificación, data linealmente separables.**
        3. **Usa LMS (actualización contínua) como regla de aprendizaje.**
        4. **Convergencia garantizada para cualquier valor de la tasa de aprendizaje.**
    2. **Adaline**
        1. **Dispositivo lineal.**
        2. **Usa LMS como regla de aprendizaje.**
        3. **Convergencia garantizada según estilo de iteración, sensible a cambios en la tasa de aprendizaje.**
5. **Vimos ejemplos de declaración de cada dispositivo en Matlab.**

Hay problemas difíciles de resolver con una sola neurona, o con una capa de neuronas y hace falta recurrir a arquitecturas más complejas para resolverlas.

Vamos a ver redes artificiales con más de una capa de neuronas, y la forma de aprendizaje:

retropropagación (Backpropagation)