



Sistemas de Visión Artificial

Tema 5. Técnicas de reconocimiento (1ª parte)

Autores: Sira Palazuelos, Luis M. Bergasa , Manuel Mazo,
M. Ángel García, Marisol Escudero, J. Manuel Miguel
Departamento de Electrónica. Universidad de Alcalá.



1. Introducción a las técnicas de reconocimiento de objetos
2. Clasificadores lineales: redes neuronales



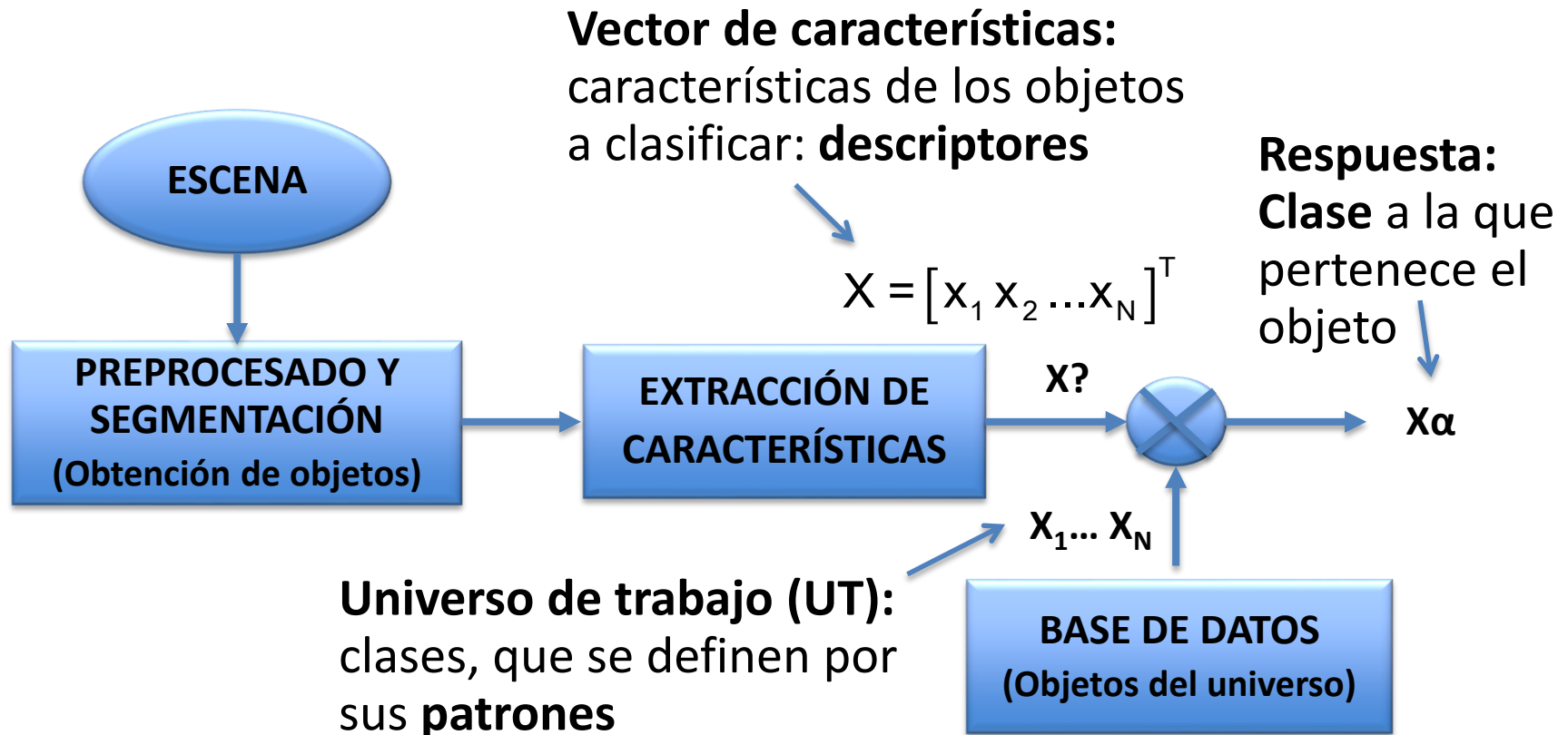
1. Introducción a las técnicas de reconocimiento de objetos



- El **reconocimiento de objetos** es la última etapa de los sistemas de visión artificial.
- **Reconocer** objetos consiste en determinar qué objetos están presentes en una imagen a partir de:
 - las **características** que hayamos obtenido de la escena, que describen cuantitativamente las distintas regiones en las que la hayamos segmentado, y
 - los **objetos** que sabemos que pueden aparecer en ella.
- Para ello se **extraen las características** de los distintos objetos de la imagen y se **clasifican automáticamente**, es decir, cada objeto se asocia a una **clase** basándose en grado de **semejanza** entre su vector de características (X?) y los vectores de los **patrones** previamente definidos.

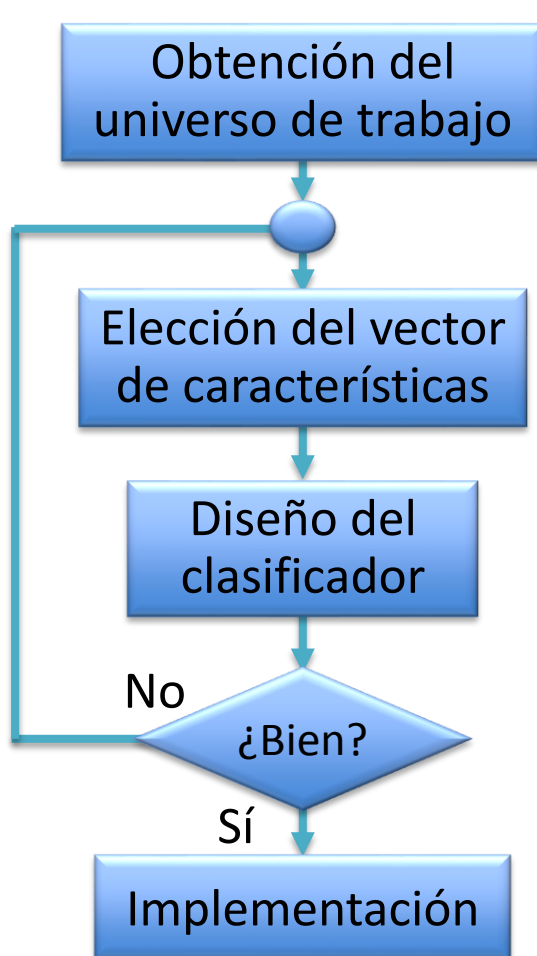
Reconocimiento de objetos

- Diagrama de bloques de un **sistema de reconocimiento** de formas/objetos cuando se está **utilizando**:

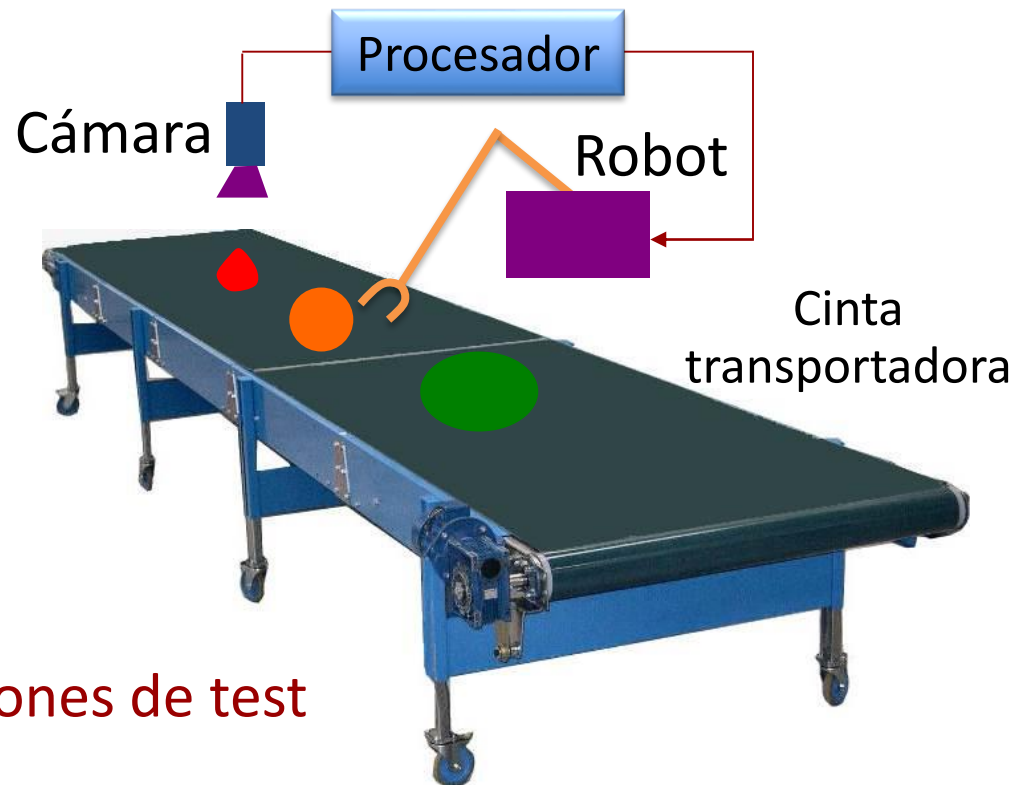


Diseño de un sistema de reconocimiento

□ Diseño de un sistema de reconocimiento de objetos



Patrones de entrenamiento



Patrones de test

1. Obtención del universo de trabajo

- ☐ UT=(sandías, naranjas, fresas)

2. Elección del vector de características X , que debe posibilitar que las clases sean **separables**. No hay reglas exactas para la selección de características. Es un paso crítico.

3. Las características elegidas deben ser:

- ☐ **Discriminantes**: que tengan valores lo más **similares** posible dentro de cada clase y lo más **diferentes** posible entre objetos pertenecientes a distintas clases.
- ☐ **Independientes**, es decir, que su información no sea redundante.
- ☐ Calculadas en **tiempo real**.
- ☐ Obtenidas con sensores **económicos**.
- ☐ **Invariantes a giros, traslaciones y homotecias (zoom)**.



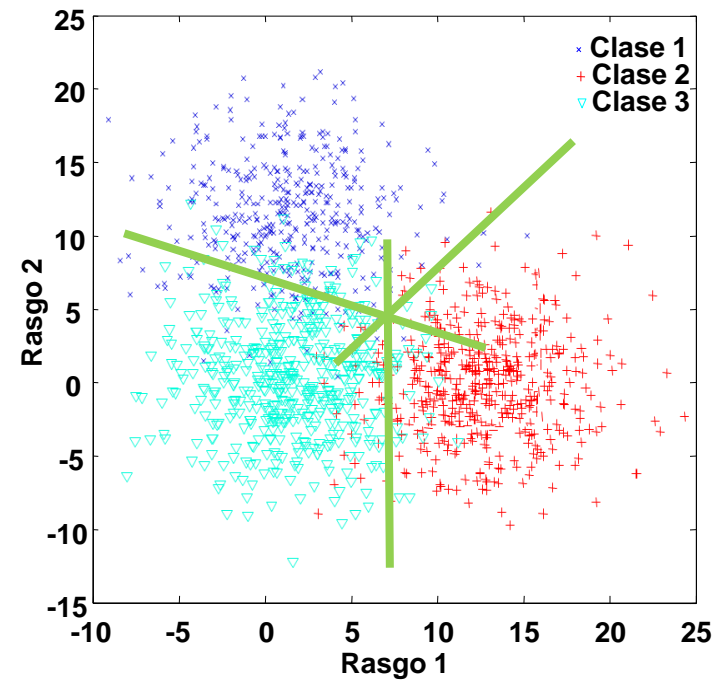
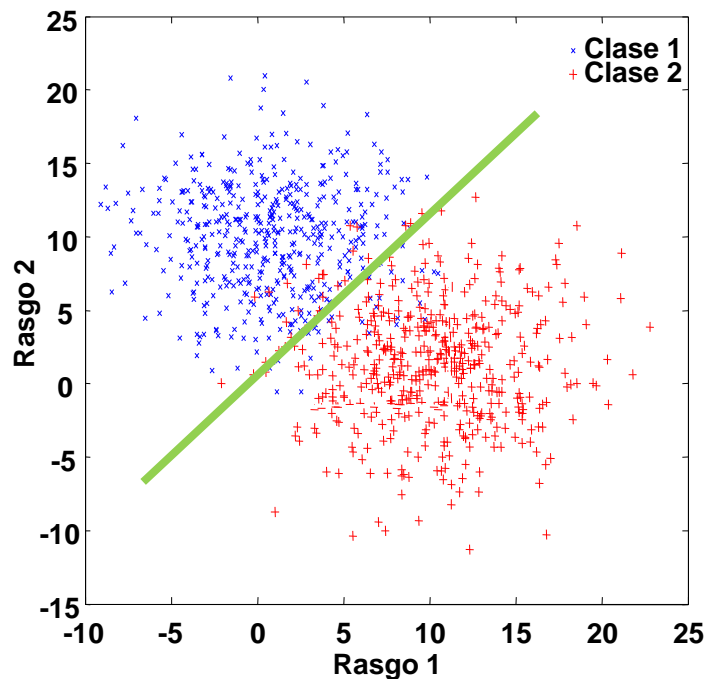


- ❑ Posibles **características** de los objetos:
 - ❑ **Características geométricas:** área, altura y ancho, perímetro, elipse, circularidad, momentos invariantes, descriptores de Fourier,...
 - ❑ **Características cromáticas:** color promedio, gradiente promedio del borde, contraste, momentos invariantes con información de color, textura,...
 - ❑ **Descriptores topológicos:** número de huecos, número de componentes conectados, número de Euler (diferencia entre los anteriores)...
- ❑ **Clasificación de las características** según su dispersión respecto a la media:
 - ❑ **Determinísticas.** Ej: tamaño de monedas.
 - ❑ **Aleatorias.** Ej: tamaño de fresas.

3. Diseño del clasificador:

- ❑ Los **clasificadores** separan los datos en las distintas clases o **regiones de decisión**. Hay distintas formas de hacerlo. El resultado dependerá del algoritmo elegido, de las características seleccionadas, de la separabilidad de los datos, etc.
- ❑ los clasificadores pueden expresarse mediante un conjunto G de C **funciones discriminantes** $G=(g_1, g_2, \dots, g_C)$ y la correspondiente **regla de clasificación**, por ejemplo, $\operatorname{argmax}(g_c(X))$ para $1 \leq c \leq C$.
- ❑ El diseño del clasificador implica la elección (y entrenamiento si es necesario) de la(s) **funcion(es) de distribución o funcion(es) discriminante(s) y de la regla de clasificación** que permiten **discriminar de forma inequívoca** entre las clases del UT (dadas unas características adecuadas).
- ❑ Suelen incluirse también una **clase de rechazo** para los objetos que no encajan en ninguna de las clases conocidas.
- ❑ El **error de clasificación** nos da una medida de cómo funciona el clasificador (problemas de **falsos positivos y falsos negativos**).

- Hay varios tipos de clasificadores, cada uno de los cuales tiene su **frontera de decisión**.
- Los **clasificadores lineales** separan las zonas por medio de líneas rectas. Problema: encontrar las líneas que separan las clases de forma óptima.

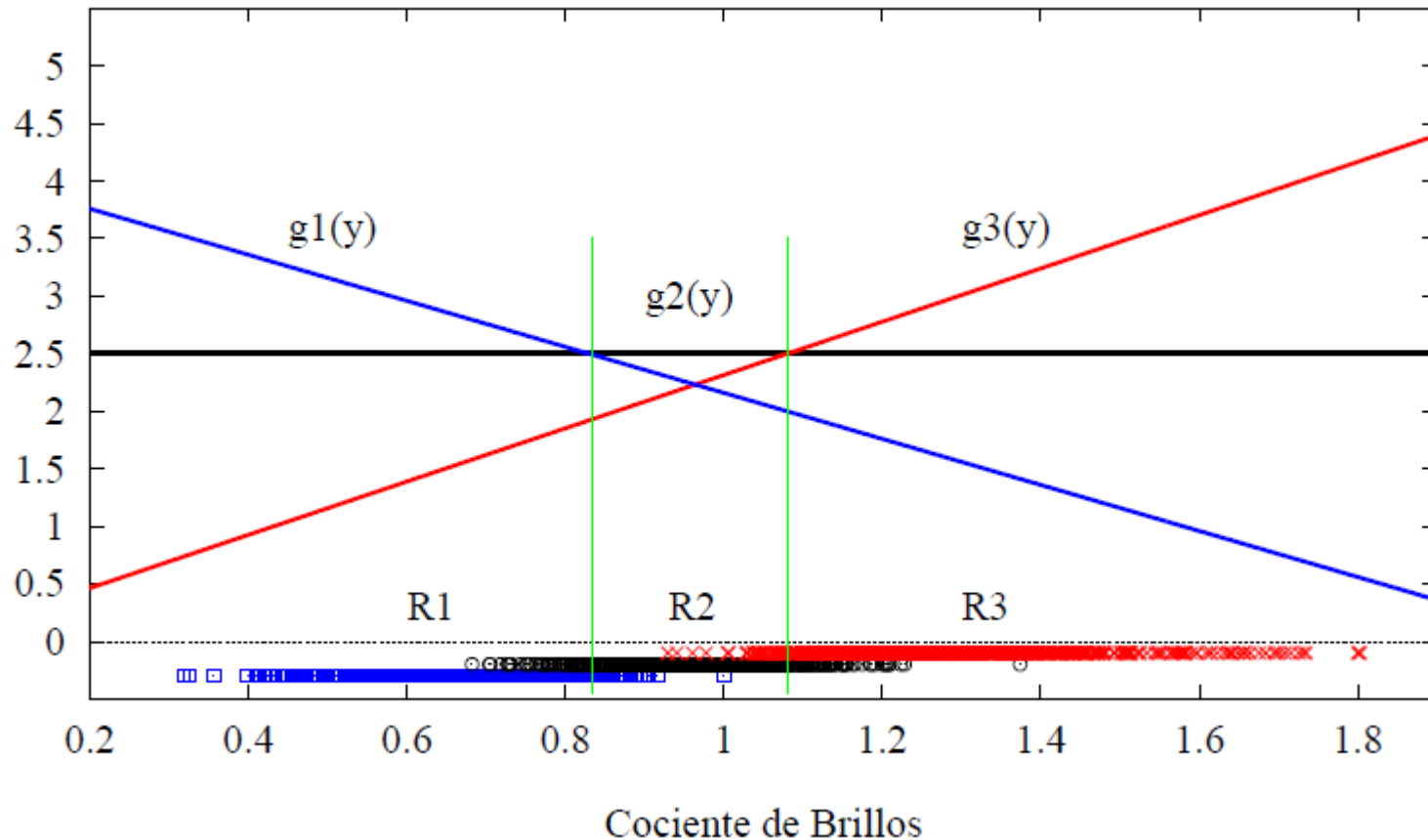


Modificada de <http://web.iti.upv.es/~evidal/students/app/tema5/t5app2p.pdf>



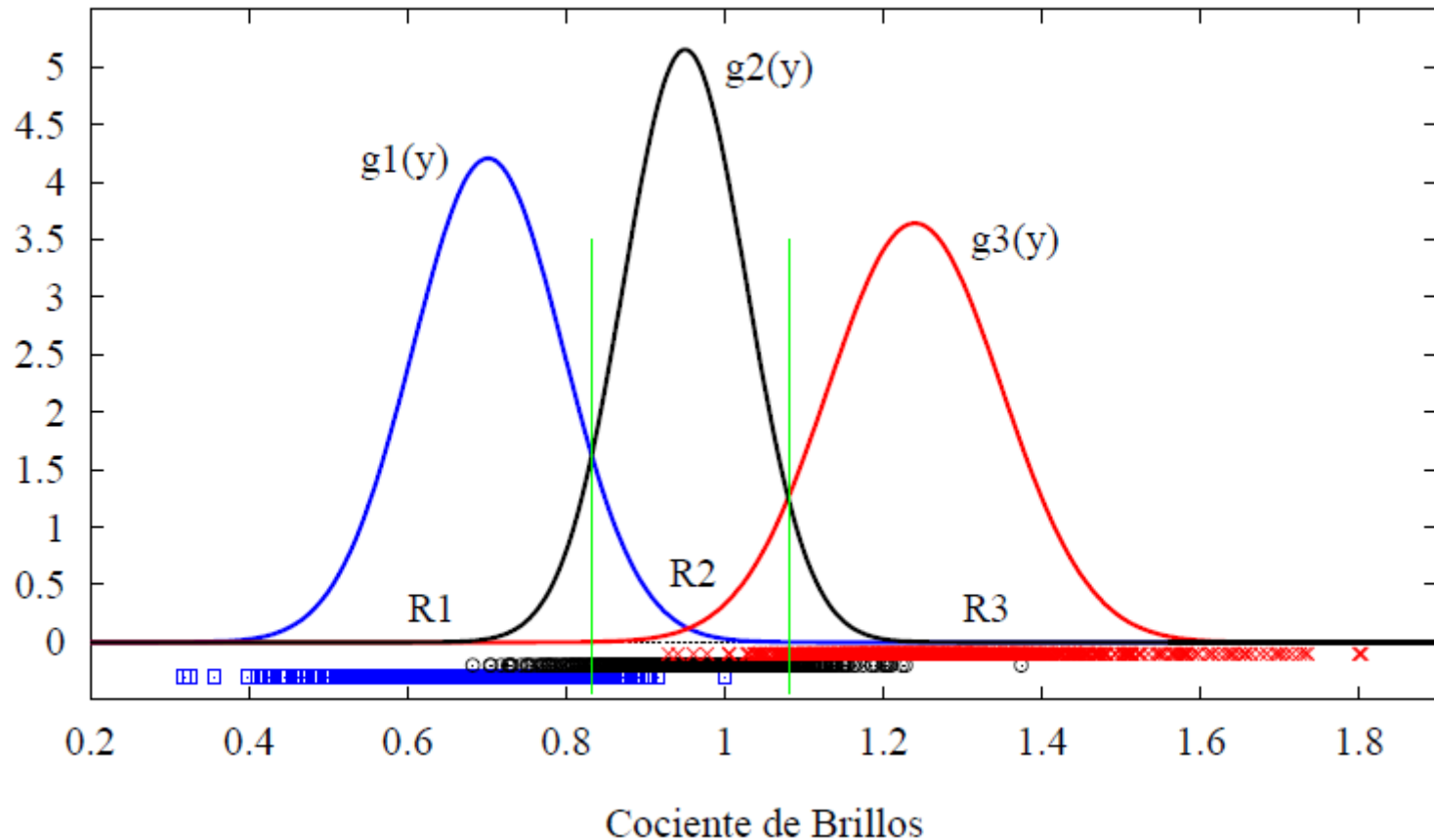
- Ejemplos: en las siguientes transparencias podemos ver la representación del valor de **funciones discriminantes** de distintos tipos en función del valor de las características (1 característica en las gráficas bidimensionales y 2 características en el caso de las representaciones tridimensionales).
- La **clase seleccionada** para cada valor de las características será aquella para la cual su función discriminante obtengan un resultado **mayor**.
- Las **fronteras de decisión** serán los puntos (valores de características) para los que las funciones discriminantes proporcionen el **mismo** resultado.

- Fronteras de decisión y funciones discriminantes (lineales) con 1 característica.



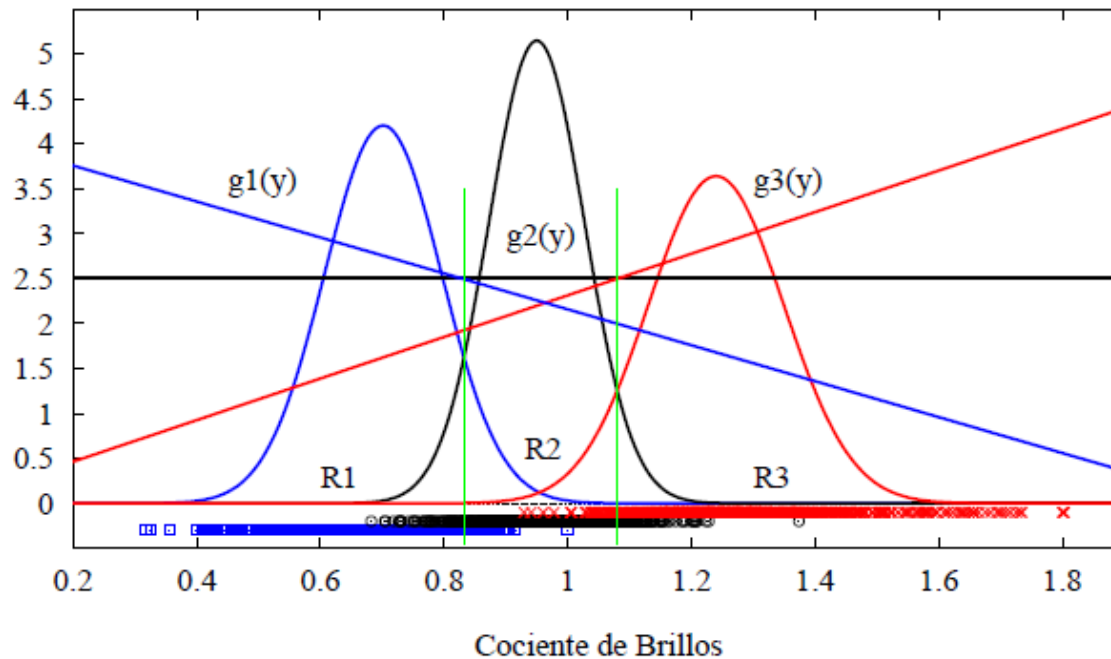
<http://web.iti.upv.es/~evidal/students/app/tema5/t5app2p.pdf>

- Fronteras de decisión y funciones discriminantes (gaussianas) con 1 característica.



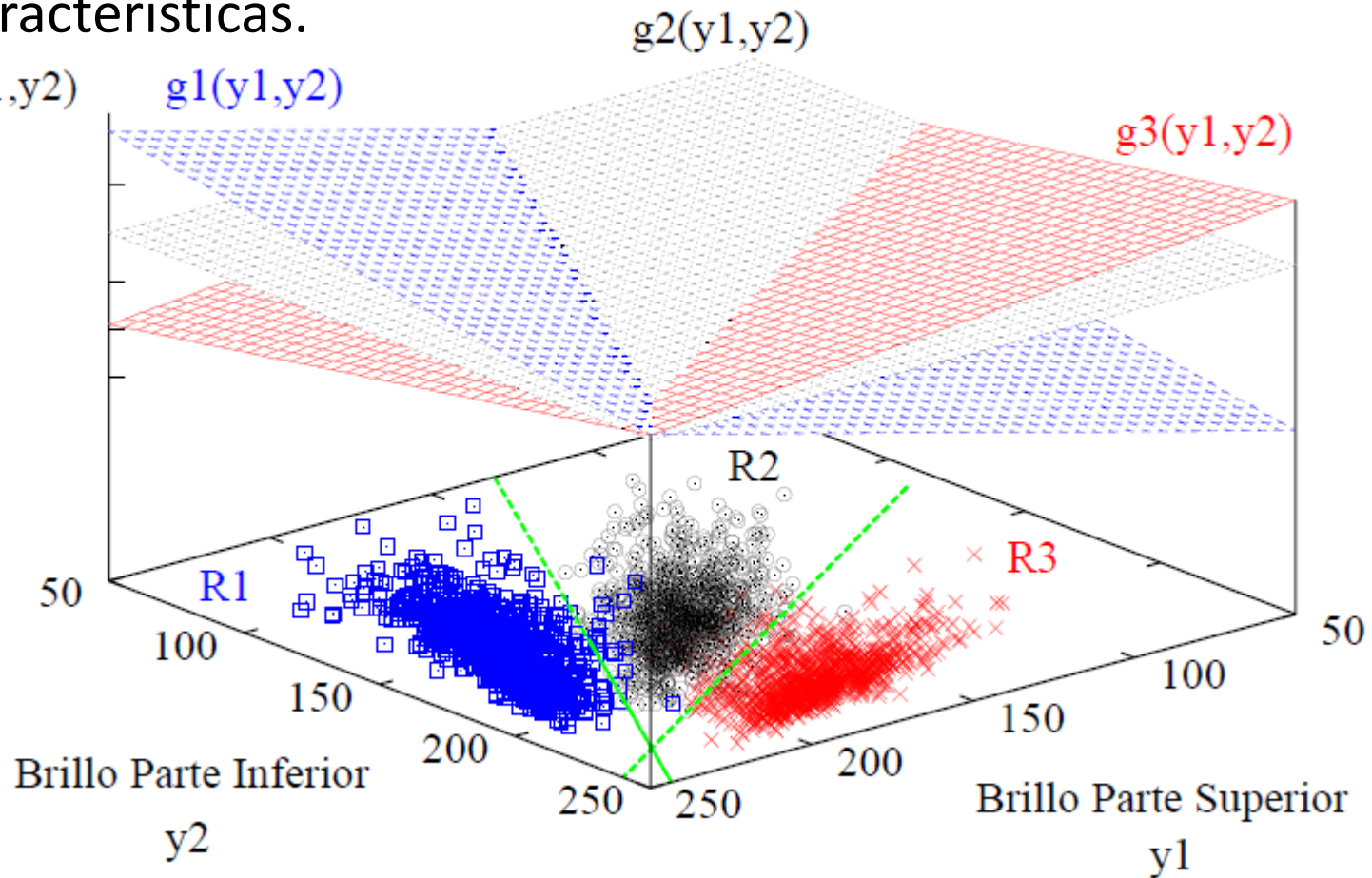
<http://web.iti.upv.es/~evidal/students/app/tema5/t5app2p.pdf>

- En general, dos clasificadores diferentes proporcionarán clasificaciones diferentes.
- Dos clasificadores son **equivalentes** si inducen las mismas **fronteras de decisión**. En la gráfica se puede observar que los clasificadores lineal y gaussiano de los ejemplos anteriores son equivalentes.



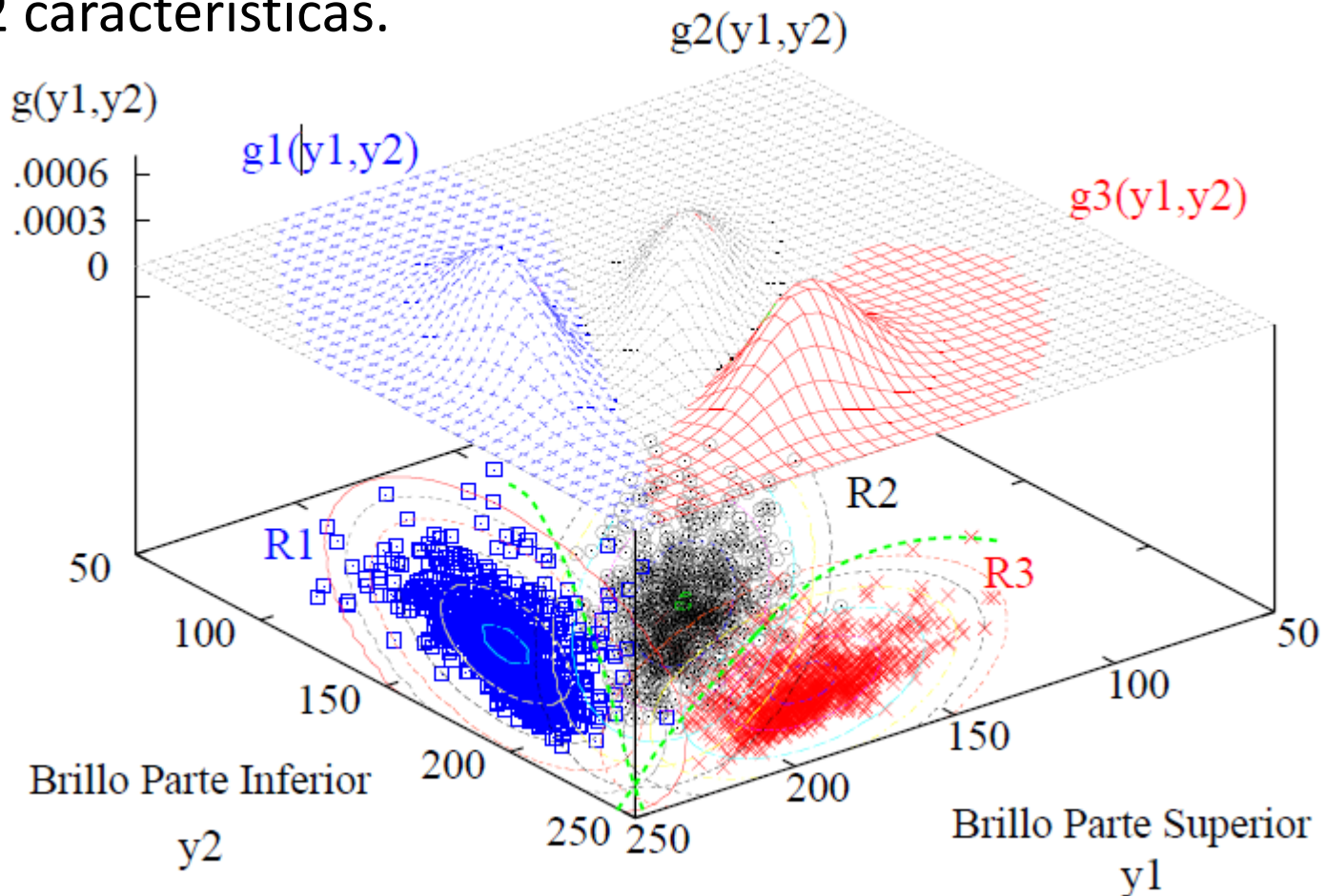
<http://web.iti.upv.es/~evidal/students/app/tema5/t5app2p.pdf>

- Fronteras de decisión y funciones discriminantes (lineales) para 2 características.



<http://web.iti.upv.es/~evidal/students/app/tema5/t5app2p.pdf>

- Fronteras de decisión y funciones discriminantes (gaussianas)) para 2 características.



<http://web.iti.upv.es/~evidal/students/app/tema5/t5app2p.pdf>

- ❑ **Tipos de clasificación**, según el conocimiento a priori de los objetos del universo de trabajo:
 - ❑ **Clasificación supervisada.**
 - ❑ **Clasificación no supervisada.**
- ❑ La **clasificación es supervisada** si existe a priori un conjunto de objetos ya clasificados en un conjunto determinado de clases y se puede utilizar esa información para entrenar el sistema.
- ❑ El proceso de **clasificación supervisada** consta de 2 etapas:
 - ❑ **Entrenamiento:** en la que a partir de los objetos ya clasificados (conjunto de entrenamiento) se obtienen una o varias reglas de decisión adaptadas a ese conjunto (que lo clasifican bien).
 - ❑ **Clasificación:** fase en la que a partir de las reglas de decisión se clasifican nuevos objetos cuya clase en principio se desconoce.

- La **clasificación** es **no supervisada** si no se conoce a priori el conjunto y número de clases en el que hay que separar los objetos.
- Es un proceso más complejo que en el caso de la clasificación supervisada.
- El **sistema encuentra automáticamente el proceso, sin conocimiento previo**, y localiza un conjunto de funciones discriminantes óptimo que permite agrupar los objetos con rasgos afines en diferentes clases.



2. Clasificadores lineales: redes neuronales

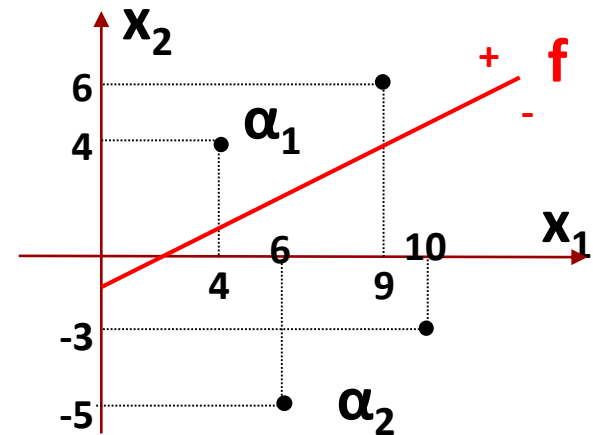
- El **clasificador lineal** separa las zonas utilizando una combinación lineal (**función discriminante lineal, FDL**) de las propiedades (componentes del vector de características, X):
$$f(x) = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n, \quad w_i = \text{peso de la caract. } i$$
- Cada **función discriminante (FD)** permite distinguir 2 regiones:

Si $f(X) > 0$ signo + Clase 1

Si $f(X) < 0$ signo - Clase 2

Si $f(X) = 0$ frontera de decisión

$$P.ej.: f(X) = x_2 - \frac{x_1}{2} + 1$$



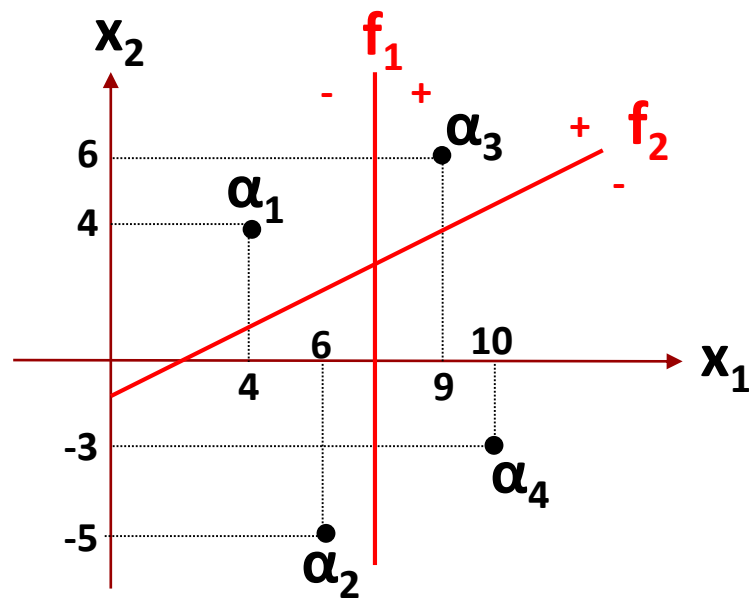
- Si tenemos 2 características la **frontera de decisión** (lugar de los puntos que podrían pertenecer a 2 clases) será una línea. Si tenemos 3 características, un plano, y si tenemos más, un hiperplano.

- Si tenemos que separar **varias regiones** utilizaremos **varias FDLs** y la **regla de clasificación** que indicará la clase dependiendo del resultado de aplicar cada FDL a las propiedades.
- En el peor de los casos necesitaremos una FDL por cada par de clases: en total $N(N-1)/2$.
- Ejemplo:

$$f_1(X) = x_1 - 7$$

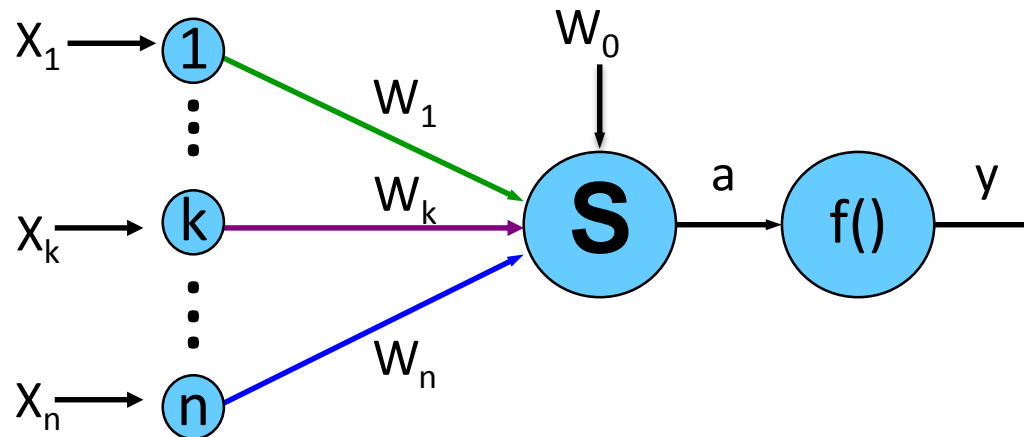
$$f_2(X) = x_2 - \frac{x_1}{2} + 1$$

	f1	f2
α_1	"-"	"+"
α_2	"-"	"-"
α_3	"+"	"+"
α_4	"+"	"-"



Redes neuronales

- Una forma práctica de hacer un **clasificador por regiones** es mediante **redes neuronales**.
- Las **redes neuronales (RNs)** están formadas por un conjunto de **neuronas básicas**, que son elementos de cálculo no lineales con varias entradas y una salida, cuya estructura se muestra a continuación:



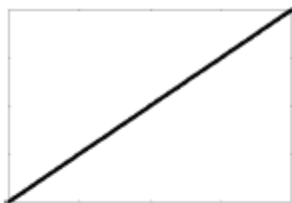


- Las **entradas** de la red neuronal serán las características de los objetos, x_1, \dots, x_n (con n =número de características).
- Cada una de las entradas tiene un **peso** asociado (w_1, \dots, w_n), que multiplica su valor, más un **offset o polarización** $w_0 \rightarrow$ la entrada de la neurona (también llamada **estado de activación de la neurona**) es: $a = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + \dots + w_0$
- Los **conocimiento** de las RNs se encuentra en los **pesos** de la red, que son **constantes, pueden tomar cualquier valor** (positivo o negativo) y se ajustan durante el **período de entrenamiento**, donde aprenden a clasificar los **patrones de entrenamiento**.
- Las entradas se ponderan y se suman al llegar a la neurona, y ésta, según su **función de transferencia** (a veces llamada **función de activación**), dará a la salida un valor determinado:

$$y = f\left(\sum_{i=1}^N w_i x_i + w_0\right) = f(W^T X)$$

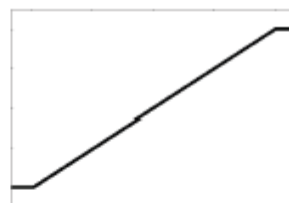
□ Las **funciones de activación** típicas son:

Lineal



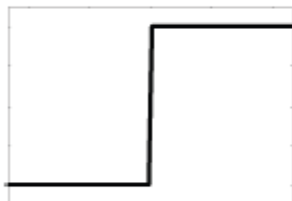
$$y = x$$

A tramos



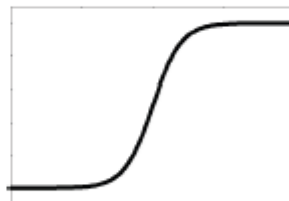
$$y = \begin{cases} -1 & x < -l \\ x & -l \leq x \leq l \\ 1 & x > l \end{cases}$$

Signo



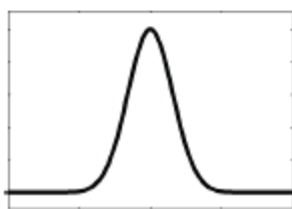
$$y = \text{sgn}(x)$$

Sigmoide



$$y = \frac{1}{1 + e^{-x}}$$

Gaussiana



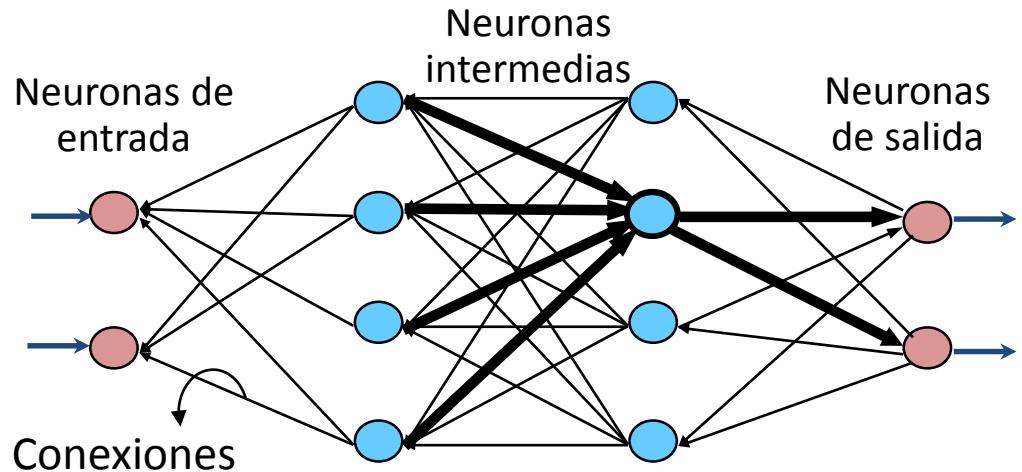
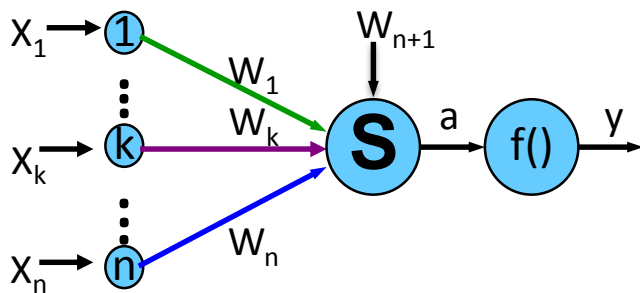
$$y = Ae^{-Bx}$$

<http://www.aic.uniovi.es/ssii/SSII-T9-RedesNeuronales.pdf>



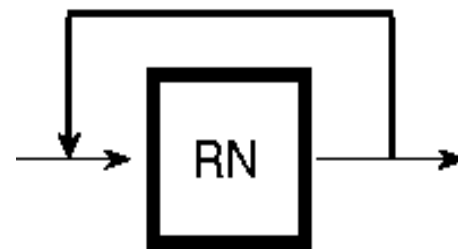
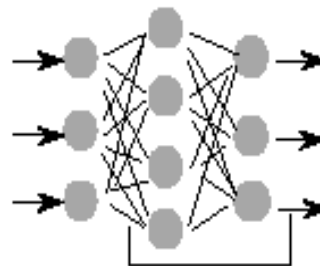
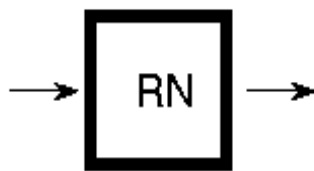
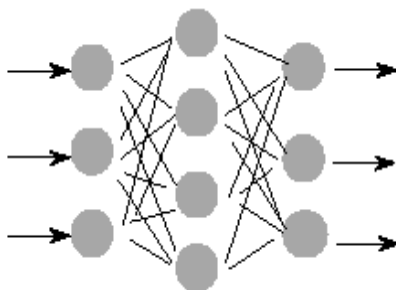
- La uso de una red neuronal tiene dos fases:
 - En primer lugar se realiza el **entrenamiento** de la misma, con lo que se consigue **ajustar los pesos** w_i a sus valores óptimos para un determinado conjunto de entrenamiento. Al final de esta fase (si el proceso **converge**) la red **clasificará, correctamente** o con **una tasa de error** aceptable, el conjunto de entrenamiento.
 - A partir de este momento ya se puede pasar a la fase de **clasificación**, en la cual cuando se presenta a la red una entrada igual o parecida a alguna que recibió en la fase de entrenamiento, la red proporcionará una respuesta que dependerá del entrenamiento.

- Las redes neuronales se pueden **clasificar** atendiendo a diversos criterios (I):
 - **Tipo de entradas:** entradas **continuas** y entradas **discretas**.
 - **Arquitectura:** Las neuronas normalmente se conectan entre sí formando unidades jerárquicas llamadas **capas**. Según el número de éstas, podemos clasificarlas en redes de **una sola capa** (perceptrón, Kohonen), o **redes multicapa** (*backpropagation*).



http://www.wiphala.net/courses/intelligent_systems/OS08/2006-I/class/class_10_neural_networks.ppt

- Las redes neuronales se pueden **clasificar** atendiendo a diversos criterios (II):
 - **Sentido de avance de la información:** redes *feed-forward* donde la información siempre se transmite desde las entradas hacia las salidas, sin que exista ningún bucle de realimentación, y redes *feed-back* donde existen bucles de realimentación y la salida depende de las entradas actuales y de los estados anteriores de la red.

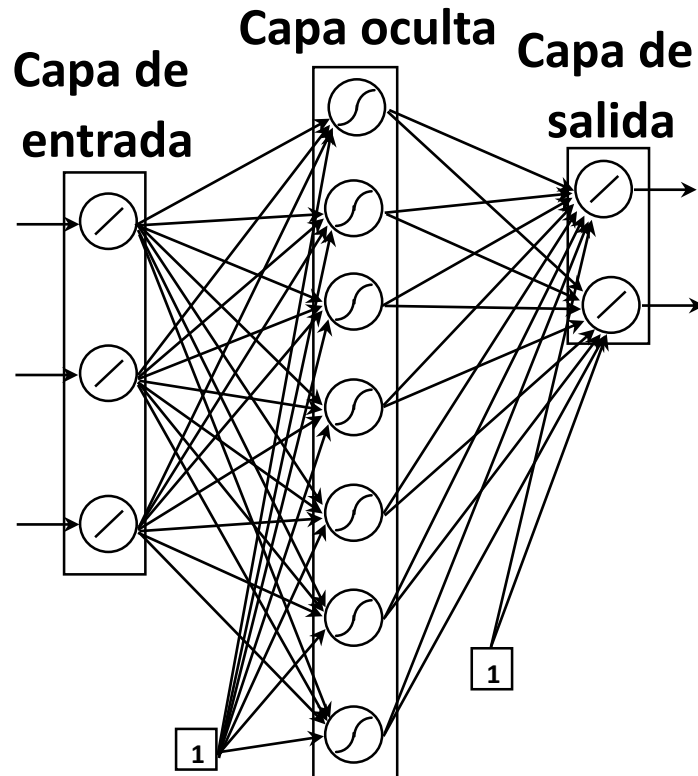


Modelo de una red *feed-forward*

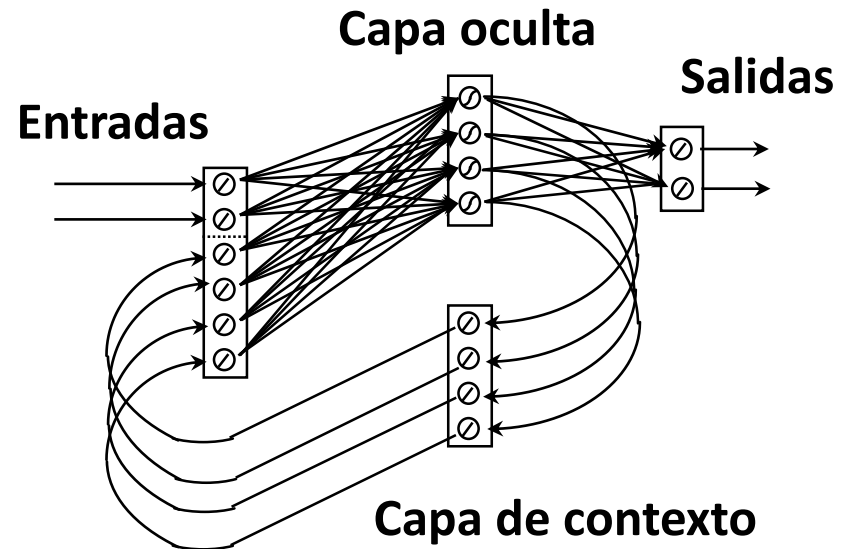
Modelo de una red *feed-back*

http://www.wiphala.net/courses/intelligent_systems/OS08/2006-I/class/class_10_neural_networks.ppt

□ Ejemplos:



Red feedforward
(Perceptrón multicapa)



Red feedback
(Red de Elman)

www.isa.cie.uva.es/~maria/diagnosis-redes.ppt



- Las redes neuronales se pueden **clasificar** atendiendo a diversos criterios (III):
 - Según el **modo de aprendizaje**: según la información disponible en la etapa de aprendizaje, este se pueden dividir en:
 - **Entrenamiento supervisado**: a la red se le proporciona un conjunto de entrenamiento, donde se conocen las entradas y la salida correspondiente a dicha combinación de las entradas. Esta información se utiliza para ajustar los pesos de forma iterativa.
 - **Entrenamiento no supervisado**: es la red quien se organiza de tal forma que es capaz de distinguir diferentes clases de datos de entrada, realizando una separación o *cluster*, sin conocer a priori las clases existentes, ni saber si la clasificación es correcta o no: la red descubre las relaciones presentes en los ejemplos y se “autoorganiza”.

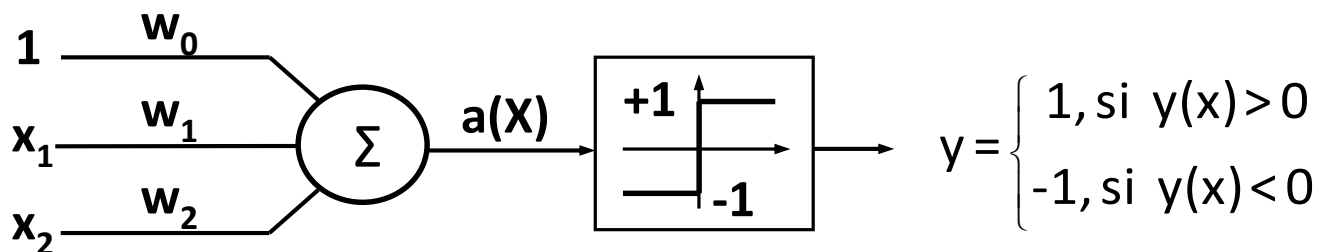
Redes neuronales

El perceptrón

- El **perceptrón** es un modelo de red **supervisada** de **una sola capa**. Es la red neuronal más sencilla.
- En su forma más simple (perceptrón para dos clases de patrones), aprende una **función de decisión lineal** que divide las entradas en dos conjuntos separados por un hiperplano:

$$a = w_0 + w_1x_1 + w_2x_2 + \dots = \sum_{i=1}^N w_i x_i + w_0 = W^T X$$

$$y = f(a) = \{ 1 \text{ si } a > 0, -1 \text{ si } a < 0 \} \text{ (signo, sgn)}$$



- Es un clasificador que funciona correctamente cuando se trata de identificar clases que sean **linealmente separables**: puede implementar una función AND o NAND, pero no una XOR.

Redes neuronales

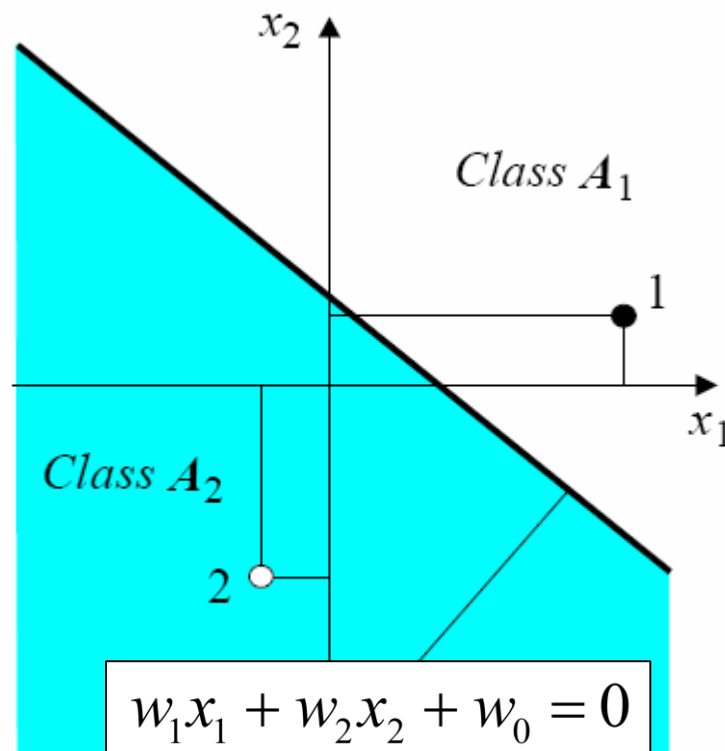
El perceptrón

□ Perceptrón de dos entradas

- La **frontera de decisión** esta determinada por:

$$w_1x_1 + w_2x_2 + w_0 = 0$$

$$w^T x = 0$$



Adaptado de <http://es.scribd.com/doc/82482616/02-redes>

Redes neuronales

El perceptrón

□ Frontera de decisión del perceptrón de dos entradas

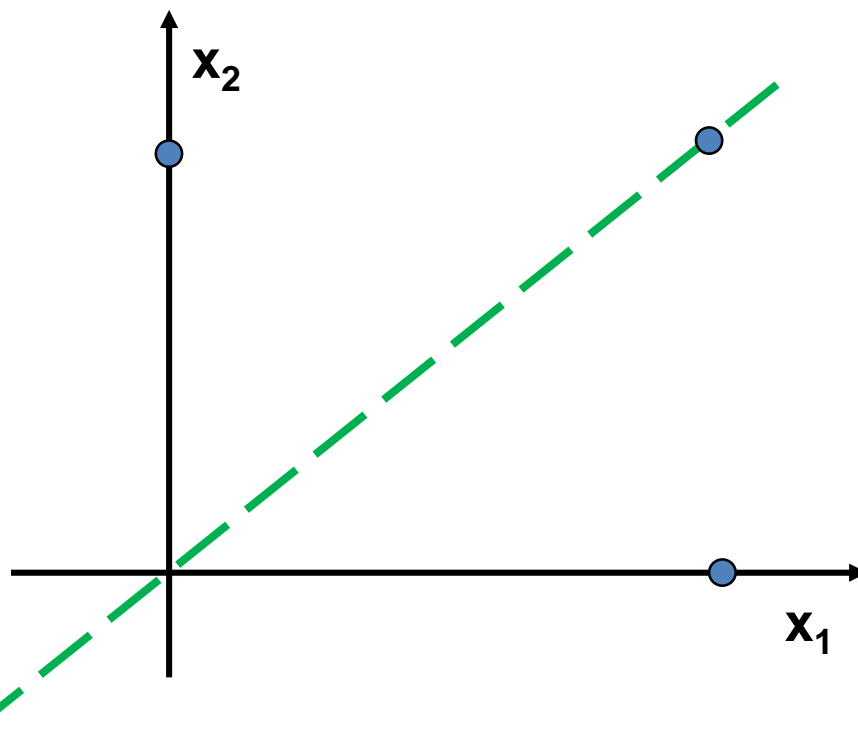
$$w = (w_1, w_2, w_0) = (1, -1, 0)$$

$$w^T \cdot x = 0$$

$$1 \cdot x_1 - 1 \cdot x_2 + 0 \cdot 1 = 0$$

$$x_1 - x_2 = 0 \rightarrow x_1 = x_2$$

**Esta es la ecuación
para la frontera de
decision**

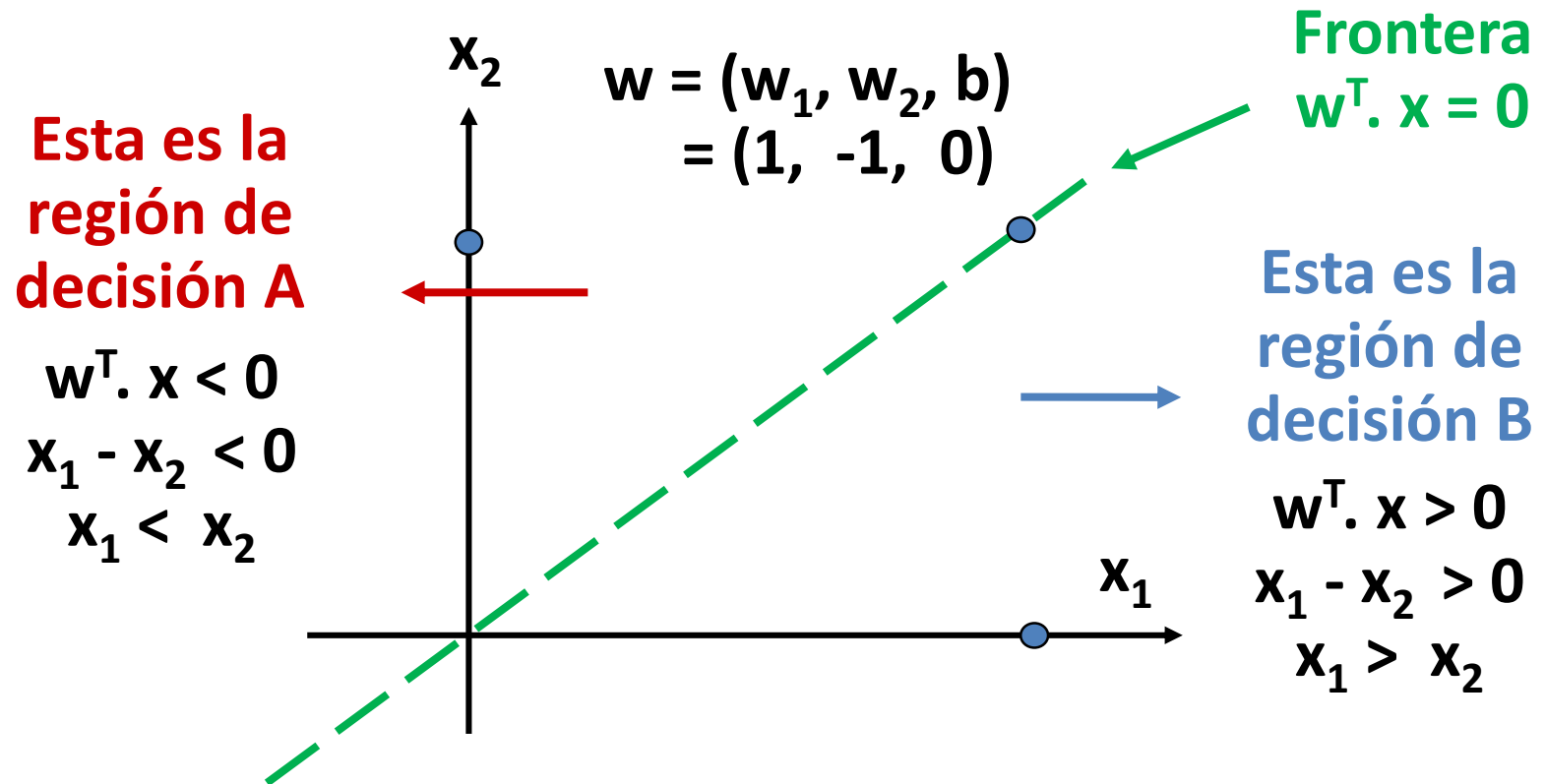


Adaptado de <http://es.scribd.com/doc/82482616/02-redes>

Redes neuronales

El perceptrón

□ Frontera de decisión del perceptrón de dos entradas

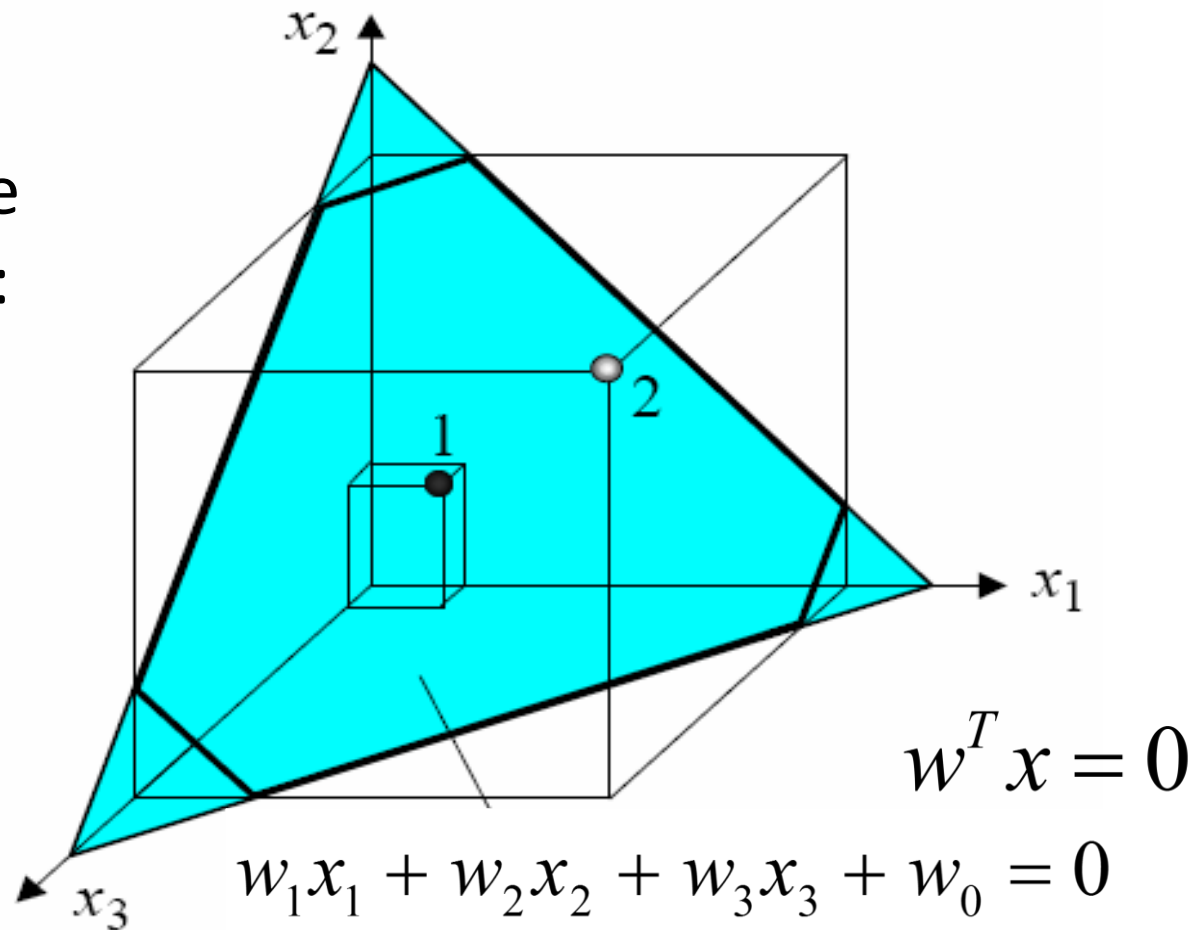


Adaptado de <http://es.scribd.com/doc/82482616/02-redes>

Redes neuronales

El perceptrón

- **Frontera de decisión del perceptrón de tres entradas: hiperplano**



Adaptado de <http://es.scribd.com/doc/82482616/02-redes>



□ Aprendizaje del perceptrón (I)

□ Es **aprendizaje supervisado**.

□ Etapas del aprendizaje del perceptrón (I):

1. Obtenemos el **conjunto de entrenamiento**, que tiene un listado valores de entrada (conjunto de pares (x_{i1}, x_{i2})) y la salida deseada para cada uno (y_{di}). Por ejemplo, para una red de 2 entradas:

$$\{(x_{11}, x_{12}), y_{d1}\}, \{(x_{21}, x_{22}), y_{d2}\}, \dots, \{(x_{M1}, x_{M2}), y_{dM}\}$$

2. Se **inician** los pesos y el offset con valores **aleatorios** (w_1, w_2, w_0).
3. Se introduce en la red una **entrada** (x_{i1}, x_{i2}) del conjunto de entrenamiento y se **calcula la salida** (y) que proporcionaría la red con los valores de los pesos disponibles en ese momento.
4. Se **compara** la salida obtenida (y) con la indicada por los datos de entrenamiento (y_{di}) para las entradas utilizadas.



□ Aprendizaje del perceptrón (II)

□ Etapas del aprendizaje del perceptrón (II):

5. Si el **error** es **cero no** se modifican los pesos. Los pesos se **modifican** en caso de error:
 - Si salida = -1, pero debería ser 1
 - Si salida = 1, pero debería ser -1
6. Se repite este proceso desde el paso 3 con todos los patrones de entrenamiento: **época**.
7. Las épocas se repiten hasta que se alcance el **criterio de terminación**. En el caso de conjuntos de entrenamiento linealmente separables, la **condición de terminación puede ser que se clasifiquen correctamente todos los ejemplos**.

□ Aprendizaje del perceptrón (II)

- ¿Cómo se **modifican los pesos** cuando la salida ha sido errónea?
- El **error** ante cada entrada es la diferencia entre la salida deseada y la salida obtenida:

$$\varepsilon(t) = (y_d - y) = (y_d - W^t \cdot X)$$

- Puesto que el error depende de los pesos actuales, estos han de ser variados para que el error disminuya.
- **Regla de aprendizaje del perceptrón:**

$$W(t+1) = W(t) + \eta \cdot \varepsilon \cdot X$$

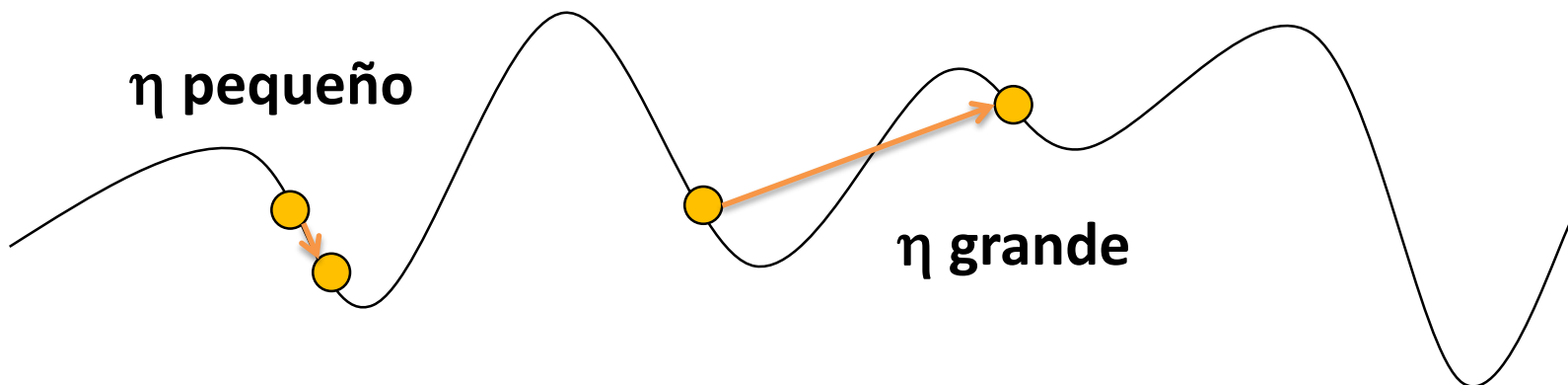
Tasa, factor o velocidad
de aprendizaje

Error = $\varepsilon = y_d - y$

□ Aprendizaje del perceptrón (III)

□ La tasa de aprendizaje, η :

- Debe tener un valor positivo y pequeño.
- Si el valor es demasiado grande puede haber problemas de convergencia.
- Si es demasiado pequeño, aprenderá muy lentamente (necesitará más iteraciones).
- Puede ser decreciente con el tiempo.



www.wiphala.net/courses/intelligent_systems/ICSI271/2009-II/class/class_42_perceptron_net.ppt

□ Aprendizaje del perceptrón (IV)

$$\left. \begin{array}{l} W(t+1) = W(t) + \eta \cdot \varepsilon \cdot X \\ \text{Error} = \varepsilon = y_d - y \end{array} \right\} \longrightarrow \Delta W = \eta \cdot \varepsilon \cdot X \longrightarrow \Delta W = \eta \cdot (y_d - y) \cdot X$$

- ¿Cómo se modifica el offset (w_0)? Suponiendo que su componente x vale 1:

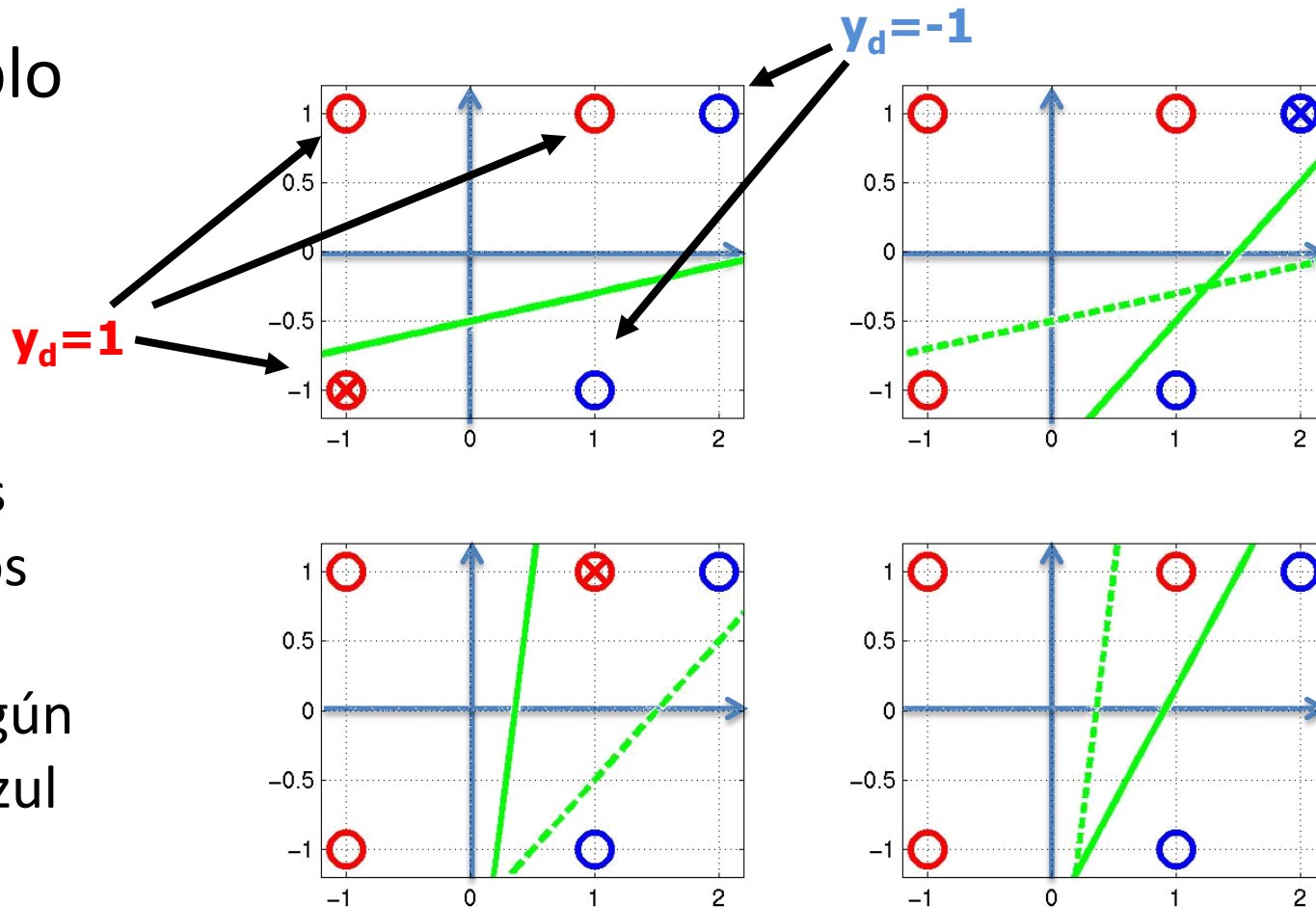
$$\Delta w_0 = \eta \cdot (y_d - y)$$

- En el conjunto de entrenamiento **no** deben incluirse datos con **componentes nulas**, ya que, aunque la salida sea errónea, no se pueden utilizar para modificar los pesos $\rightarrow \Delta W$ será 0 para las componentes nulas.

Redes neuronales

El perceptrón

□ Ejemplo



Deseamos
agrupar los
datos en
clases, según
su color azul
o rojo

Adaptado de <http://es.scribd.com/doc/82482616/02-redes>

Redes neuronales

El perceptrón

□ Ejemplo.

Iniciada

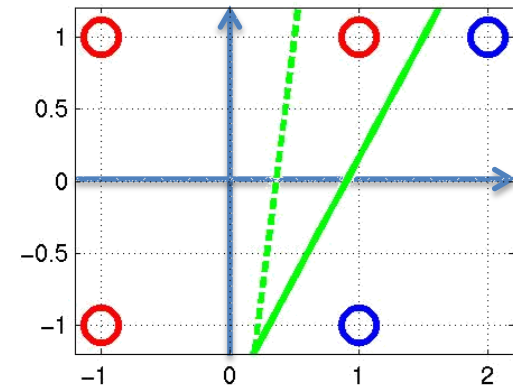
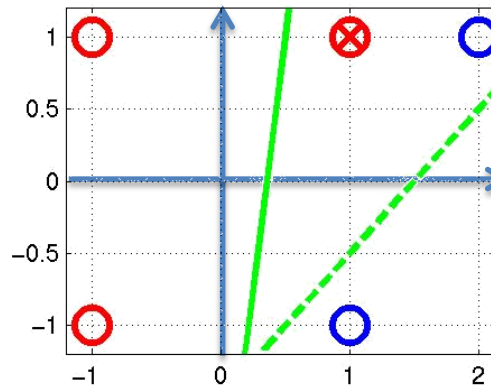
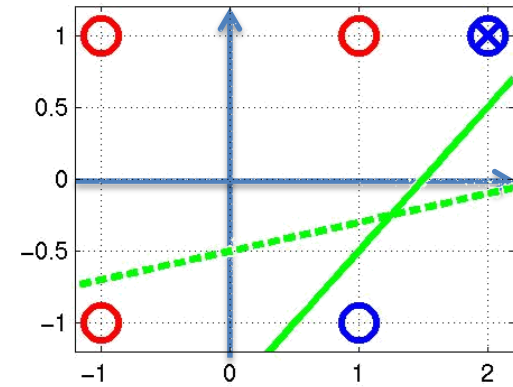
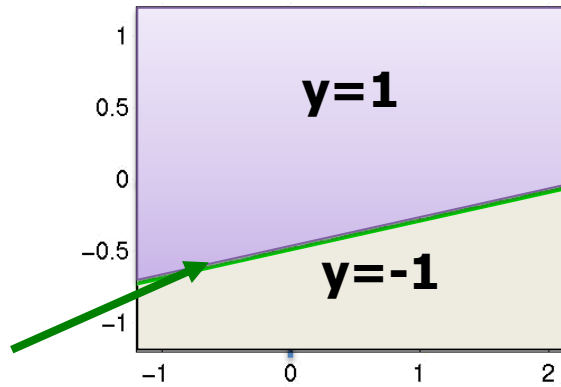
aleatoriamente a:

$$w = [0.25 \ -0.1 \ 0.5] =$$

$$= [w_0, w_1, w_2]$$

$$\rightarrow 0.25 - 0.1x_1 + 0.5x_2 = 0$$

$$x_2 = 0.2x_1 - 0.5$$



Adaptado de <http://es.scribd.com/doc/82482616/02-redes>



Redes neuronales

El perceptrón

□ Ej.

$$w = [0.25 \ -0.1 \ 0.5]$$

$$x_2 = 0.2x_1 - 0.5$$

$$(x, y_d) = ([-1, -1], 1)$$

$$y = \text{sgn}(0.25 - 0.1x_1 + 0.5x_2)$$

$$y = \text{sgn}(0.25 + 0.1 - 0.5)$$

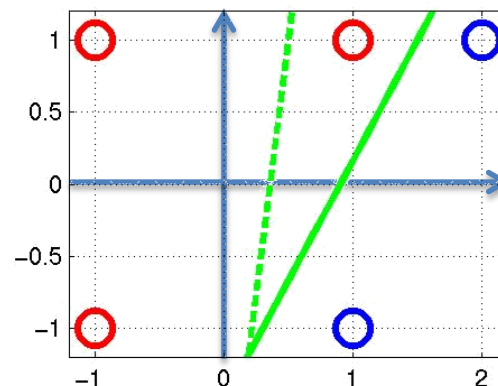
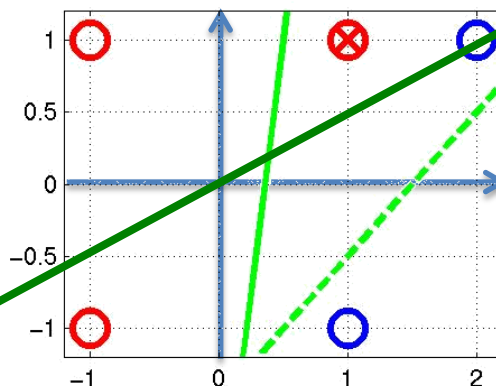
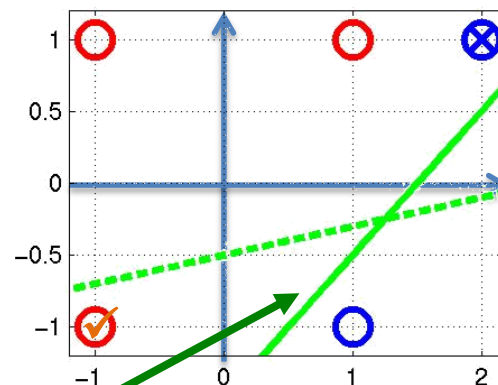
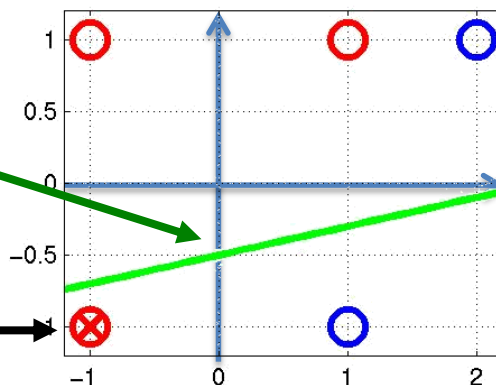
$= -1$

$$\eta = 0.1 \quad \varepsilon = 2$$

$$\Delta w = [0.2 \ -0.2 \ -0.2]$$

$$w = [0.45 \ -0.3 \ 0.3]$$

$$x_2 = x_1 - 1.5$$



Adaptado de <http://es.scribd.com/doc/82482616/02-redes>



Redes neuronales

El perceptrón

□ Ej.

$$w = [0.45 \ -0.3 \ 0.3]$$

$$x_2 = x_1 - 1.5$$



$$(x, y) = ([2, 1], -1)$$

$$y = \text{sgn}(0.45 - 0.6 + 0.3)$$

-1

$$\eta = 0.1 \quad \varepsilon = -2$$

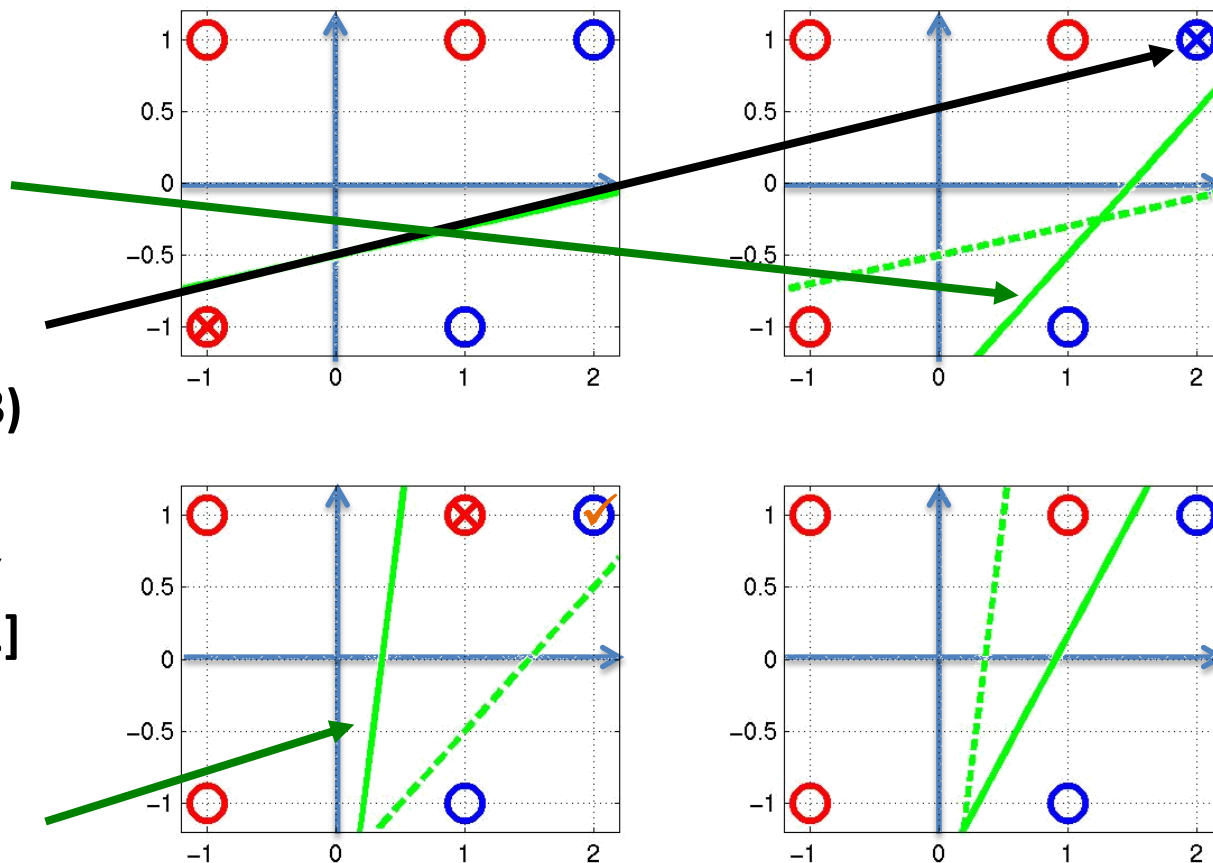


$$\Delta w = [-0.2 \ -0.4 \ -0.2]$$



$$w = [0.25 \ -0.7 \ 0.1]$$

$$x_2 = 7 \cdot x_1 - 2.5$$



Adaptado de <http://es.scribd.com/doc/82482616/02-redes>



Redes neuronales

El perceptrón

□ Ej.

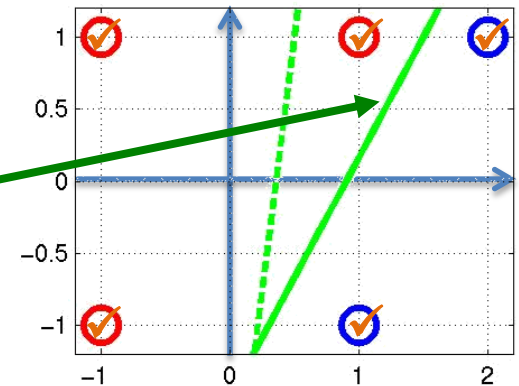
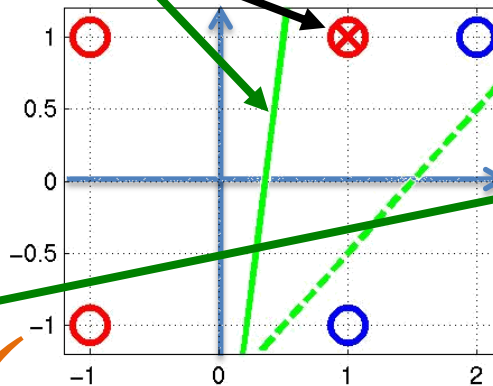
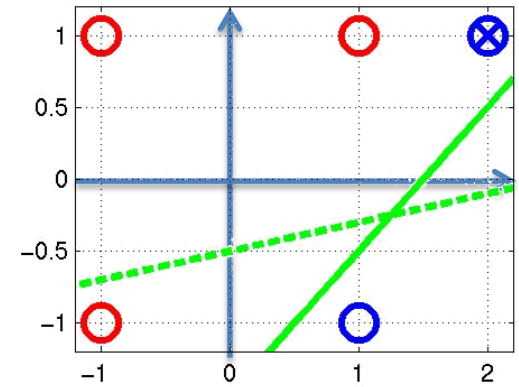
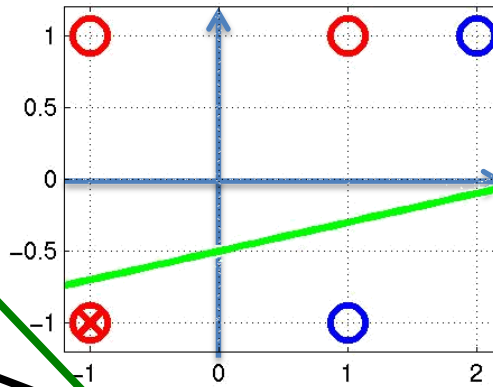
$$\mathbf{w} = [0.25 \ -0.7 \ 0.1]$$
$$x_2 = 7 \cdot x_1 - 2.5$$

$$(x, y) = ([1, 1], 1)$$
$$y = \text{sgn}(0.25 - 0.7 + 0.1)$$
$$= -1$$

$$\eta = 0.1 \quad \varepsilon = 2$$

$$\Delta \mathbf{w} = [0.2 \ 0.2 \ 0.2]$$

$$\mathbf{w} = [0.45 \ -0.5 \ 0.3]$$
$$x_2 = (5/3) \cdot x_1 - 1.5$$



Adaptado de <http://es.scribd.com/doc/82482616/02-redes>

Redes neuronales

El perceptrón

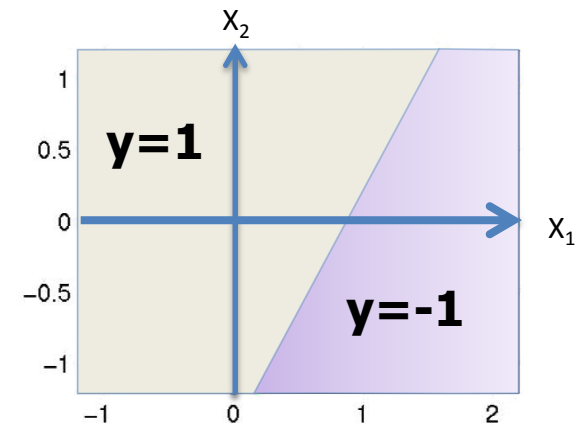
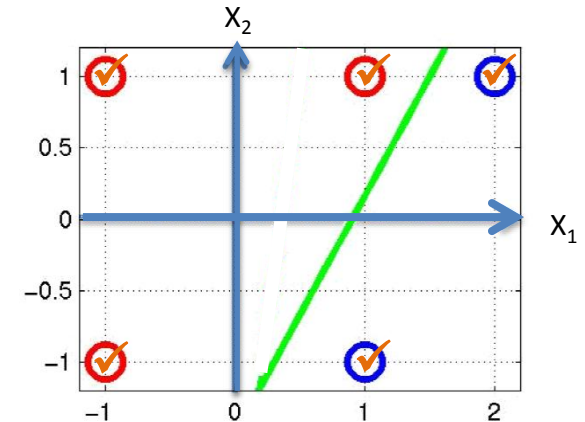
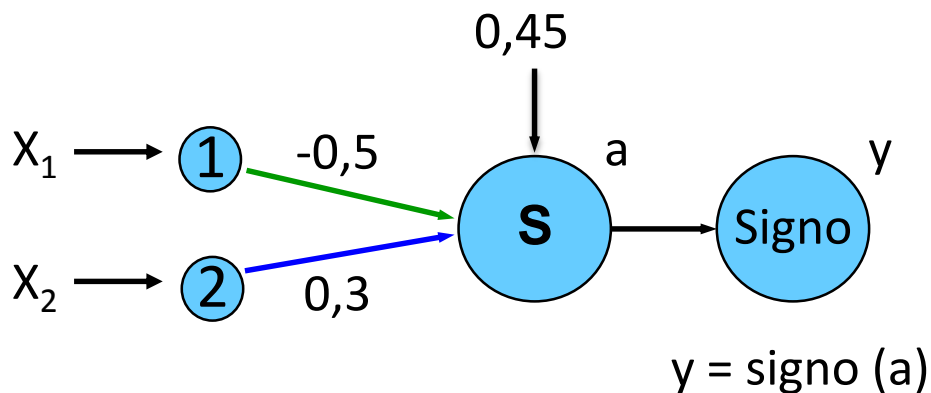
- Es decir, la red entrenada, capaz de separar adecuadamente los dos conjuntos, es la siguiente:

$$W = [w_0, w_1, w_2] = [0.45 \ -0.5 \ 0.3]$$

$$\text{Recta: } x_2 = (5/3) \cdot x_1 - 1.5$$

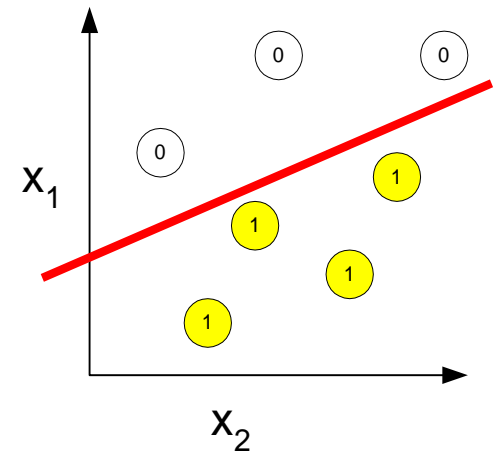
Estado de activación de la neurona:

$$a = 0,45 - 0,5 X_1 + 0,3 X_2$$



□ Teorema de la convergencia del perceptrón

- Se puede demostrar que el algoritmo de aprendizaje siempre encontrará los pesos que clasifiquen las entradas, si tales pesos existen.
- Minsky & Papert demostraron que tales pesos existen si y solamente si el problema es linealmente separable.
- Es decir, si dos clases son **linealmente separables**, y la **tasa/velocidad de entrenamiento suficientemente pequeña** el algoritmo delta entrenará la red para **separarlas perfectamente (sin errores)** en un número **finito** de iteraciones.



Adaptado de <http://es.scribd.com/doc/82482616/02-redes>

Redes neuronales

El perceptrón

□ ¿Como entrenar al perceptrón para reconocer este 3?

□ Perceptrón de 64 entradas y 1 salida

□ Asignamos:

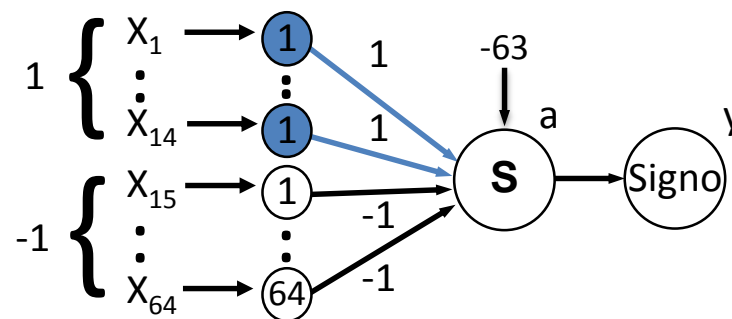
□ -1 a los pesos de las entrada de valor -1 .

□ $+1$ a los pesos de las entradas de valor $+1$.

□ Polarización = -63 .

□ La salida del perceptrón será 1 cuando se le presente un tres “perfecto”, y -1 para cualquier otro patrón.

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	+1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	+1	-1	-1
-1	-1	-1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	+1	-1	-1
-1	-1	-1	-1	-1	+1	-1	-1
-1	-1	+1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1



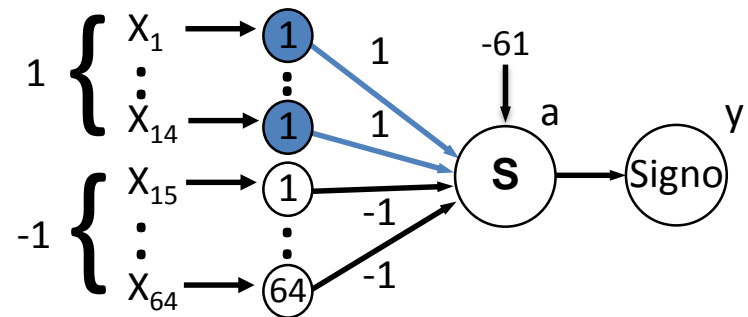
Adaptado de <http://es.scribd.com/doc/82482616/02-redes>

Redes neuronales

El perceptrón

- ¿Que pasa si el 3 a ser reconocido es ligeramente diferente?
- El 3 con un bit **corrupto** genera resultado -1 en la red anterior.
- Si polarización = -61 , este 3 y todos los patrones con un bit corrupto también serán reconocidos.
- Un solo ejemplo de patrón corrupto en el entrenamiento hace que el sistema sea capaz de **generalizar**.

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	+1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	+1	-1	-1
-1	-1	-1	+1	+1	+1	-1	-1
-1	+1	-1	-1	-1	+1	-1	-1
-1	-1	-1	-1	-1	+1	-1	-1
-1	-1	+1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1



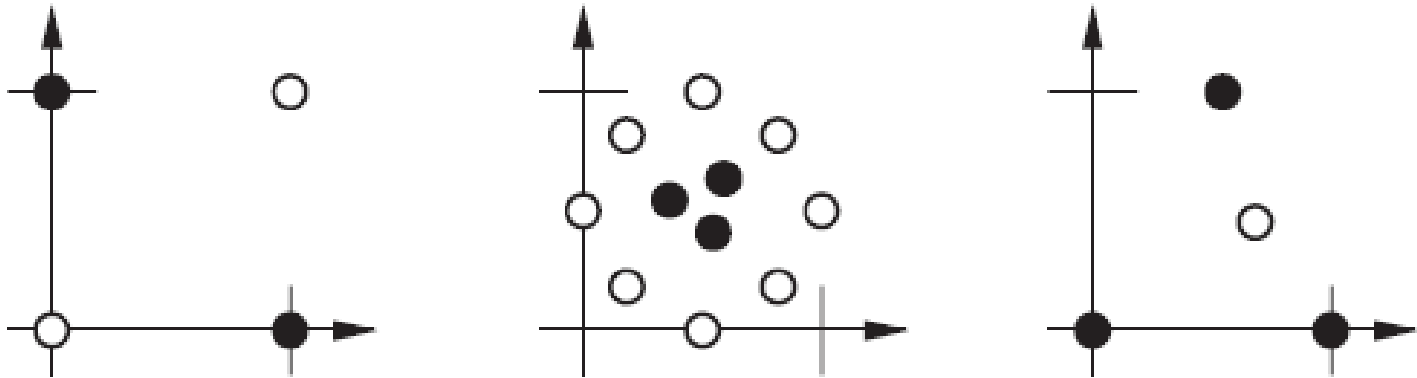
Adaptado de <http://es.scribd.com/doc/82482616/02-redes>

Redes neuronales

El perceptrón

□ Limitaciones del perceptrón de 1 capa:

- Se garantiza la **convergencia** de la regla de aprendizaje del perceptrón a una solución en un numero finito de pasos, siempre que **exista** una solución: que las dos clases sean **linealmente separables** (las clases deben poder separarse con una línea recta entre ellas).
- Problemas linealmente no separables:



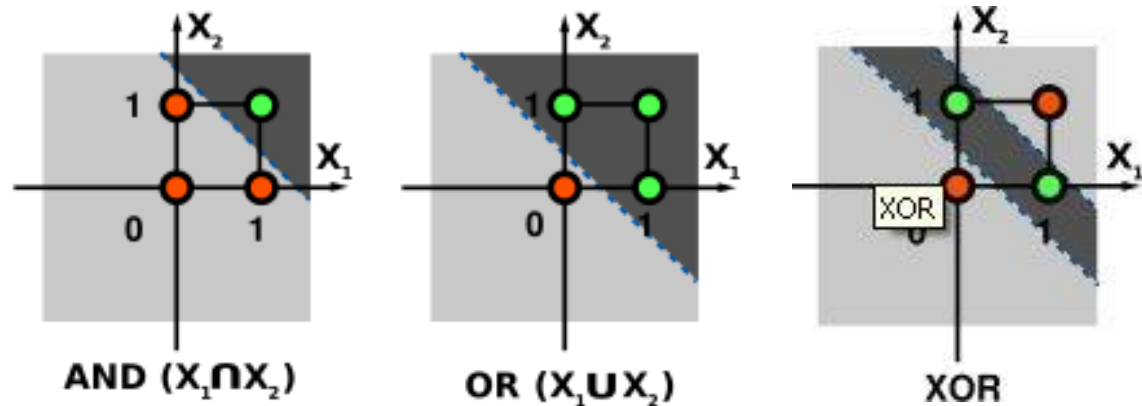
Adaptado de <http://es.scribd.com/doc/82482616/02-redes>

Redes neuronales

El perceptrón

□ Problema de la XOR

- Podemos entrenar un perceptrón de una capa para implementar la función AND, u OR, pero no para la XOR (ni muchas otras!) ya que **no se puede separar con una recta**.

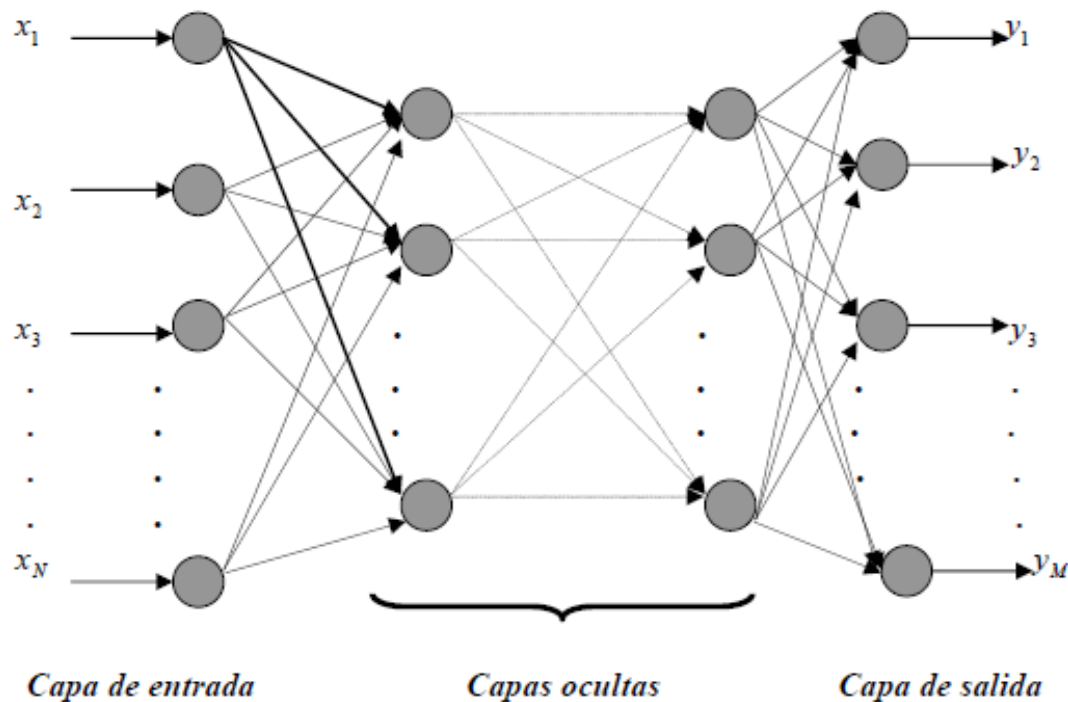


- El problema de la XOR puede resolverse con 2 rectas: utilizaremos dos neuronas, una por cada recta, y la salida de ambas la conectaremos a una tercera: **red multicapa**.

www.wiphala.net/courses/intelligent_systems/ICSI271/2009-II/class/class_42_perceptron_net.ppt

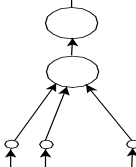
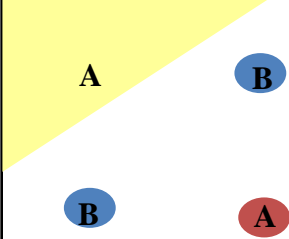
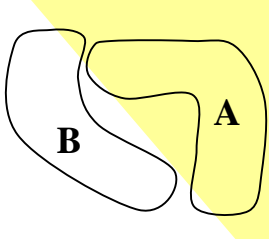

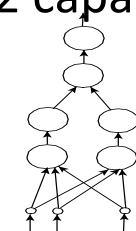
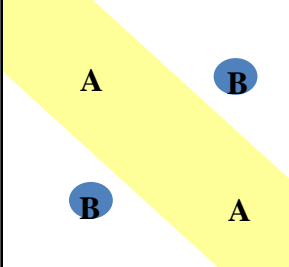
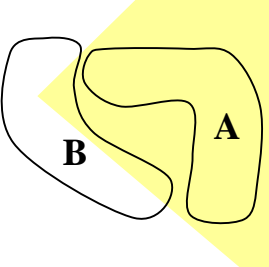
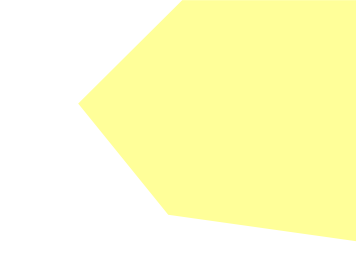
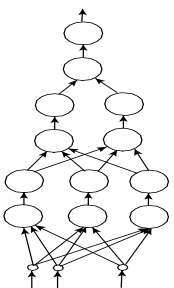
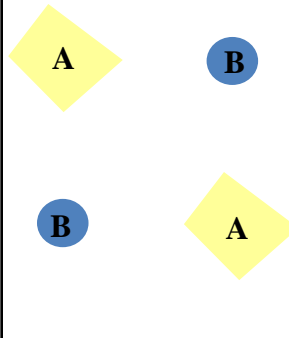
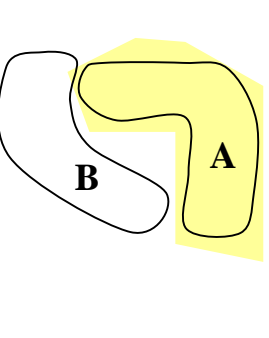
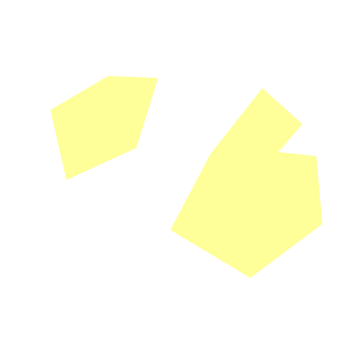
□ Redes neuronales multicapa

- Para implementar funciones de decisión que reconozcan **patrones multiclase, con independencia de que las clases sean o no linealmente separables**, se utilizan arquitecturas con **numerosas neuronas**. Ej: Perceptrón multicapa:



□ Redes neuronales multicapa

- En la siguiente transparencia se han representado los tipos de **regiones de decisión** que se pueden formar con redes neuronales progresivas (*feed-forward*) monocapa y multicapa, todas con tres entradas y una salida:
 - Las redes **monocapa** separan clases **linealmente separables**.
 - Las redes de **dos capas** separan **regiones cerradas o convexas**.
 - Las redes de **tres capas con funciones de activación no lineales** pueden implementar **funciones de decisión de complejidad arbitraria**. **Teorema de Kolmogorov**: cualquier función continua con n entradas y m salidas puede ser implementada exactamente por una red neuronal de tres capas sin retroalimentación que tenga:
 - una capa de entrada de n elementos que únicamente copian las entradas a la siguiente capa,
 - $(2n + 1)$ elementos de procesamiento en la capa intermedia, y
 - m elementos de procesamiento en la capa de salida.
- En ocasiones se utilizan redes con **más de tres capas** por la dificultad para implementar con 3 capas ciertas funciones.

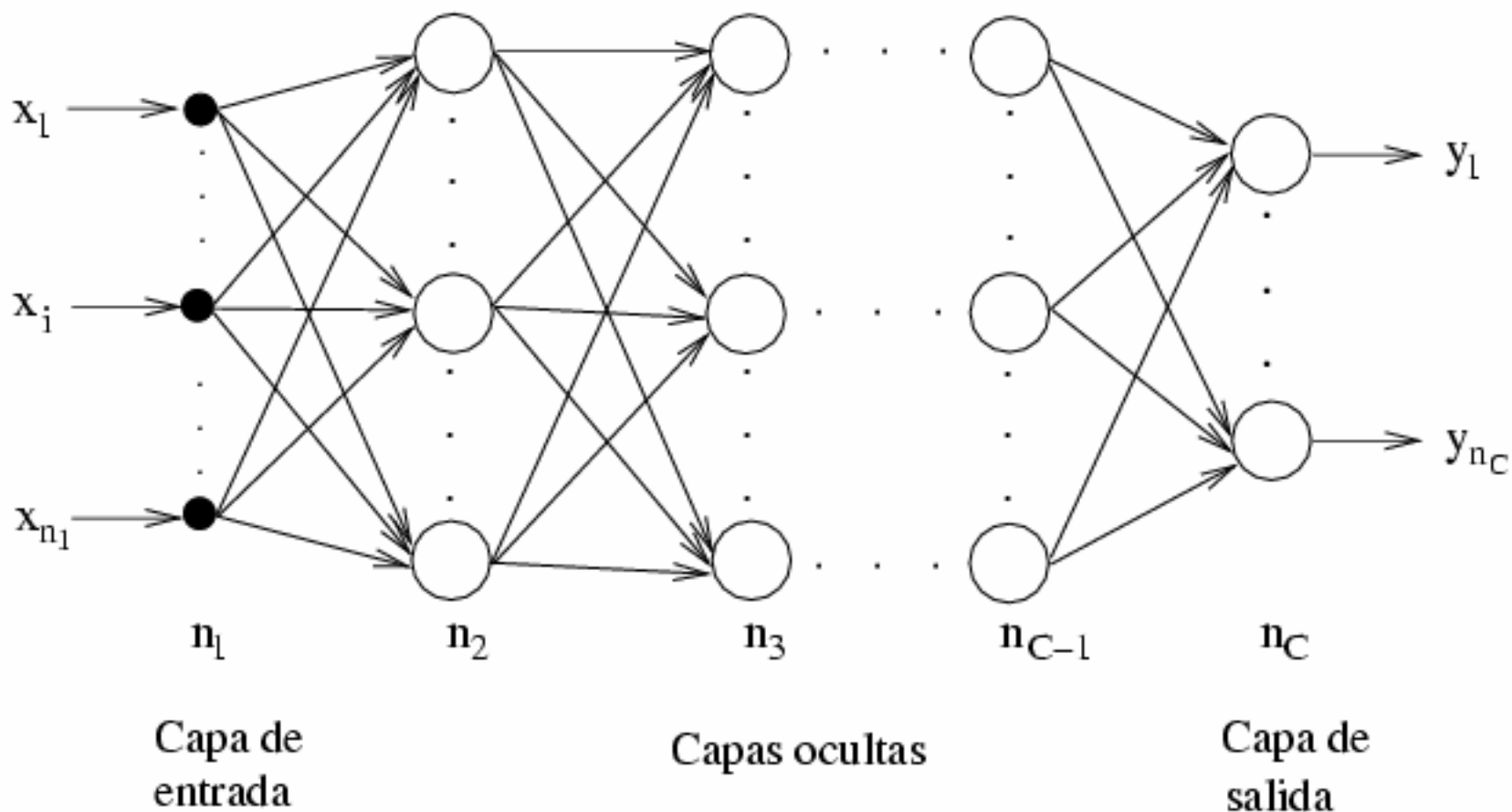
Estructura	Regiones de decisión	Problema de la XOR	Clases con regiones mezcladas	Formas de regiones más generales
<p>1 capa</p> 	Medio plano limitado por un hiperplano			
<p>2 capas</p> 	Regiones cerradas o convexas			
<p>3 capas</p> 	Clasificador universal. Complejidad arbitraria limitada por el # de neuronas			



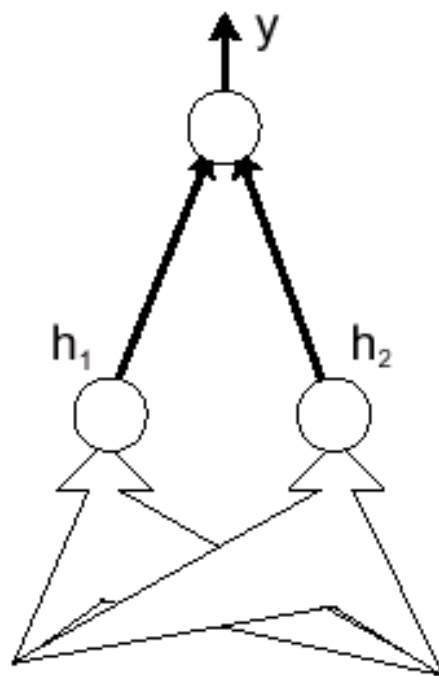
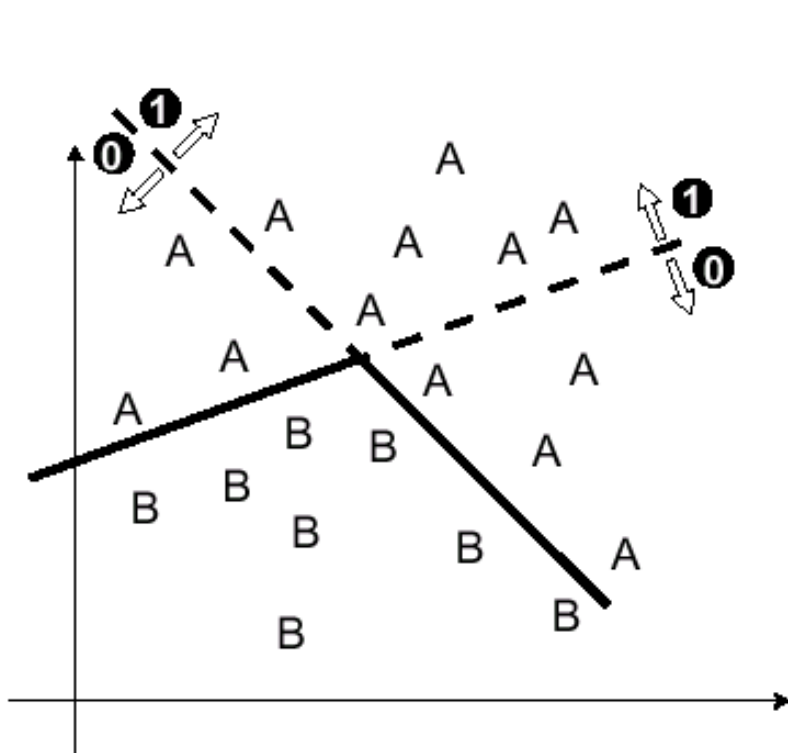
□ El perceptrón multicapa. Arquitectura

- Está formado por una **capa de entrada**, una o más **capas intermedias**, y una **capa de salida**.
- El número de neuronas de la **capa de entrada** es igual a la **dimensión del vector de características**. Solo se encarga de recibir las señales de entrada y propagarlas a la siguiente capa.
- El número de neuronas de la **capa de salida** es igual al **número de clases de patrones que es capaz de reconocer la red neuronal**. Proporciona al exterior la respuesta de la red.
- Las **capas ocultas** realizan un procesamiento no lineal de los datos recibidos. No existen reglas conocidas para determinar el número de **nodos de las capas ocultas**, con lo que su número se escoge arbitrariamente y se refina haciendo pruebas. Habitualmente es suficiente con una sola capa oculta con pocas unidades.
- Son redes **feedforward**: alimentadas hacia adelante.
- La **salida de cada neurona** de una determinada capa generalmente se conecta las entradas de **todas** las neuronas de la **siguiente capa**.

□ El perceptrón multicapa. Arquitectura



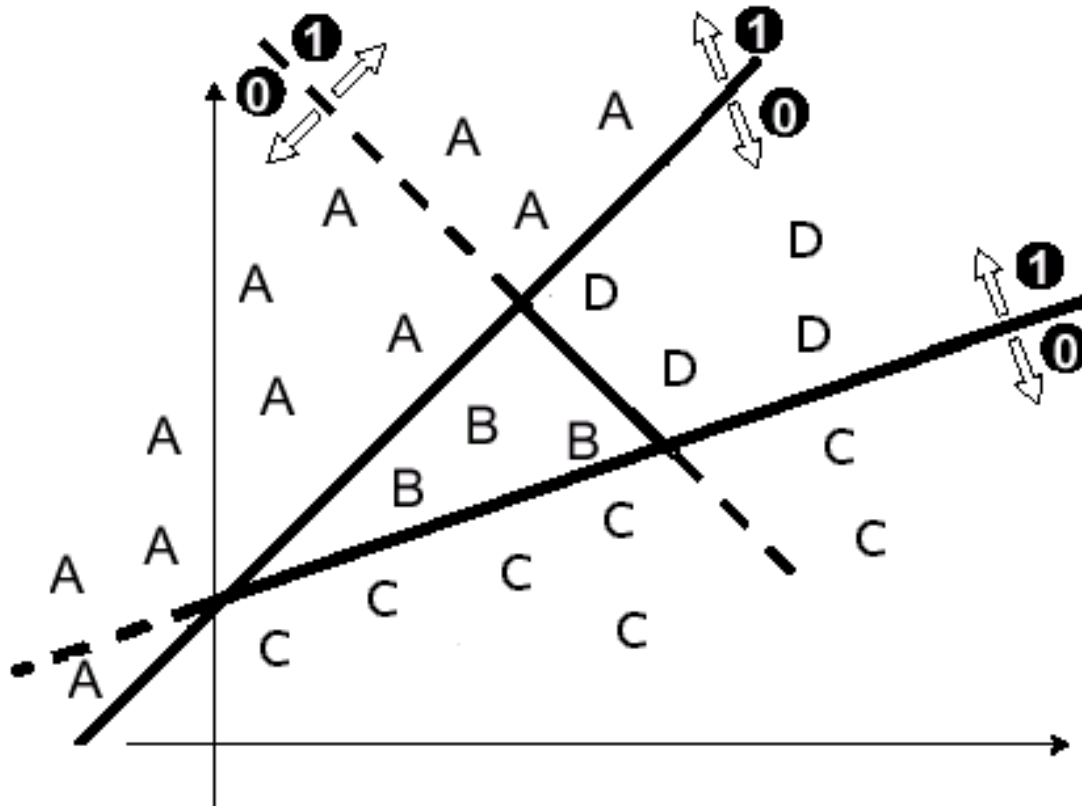
- El **perceptrón multicapa** permite integrar aprendizaje entre varias capas. Por ejemplo, si son 2 capas:



h_1	h_2	y
0	0	0
0	1	1
1	0	1
1	1	1

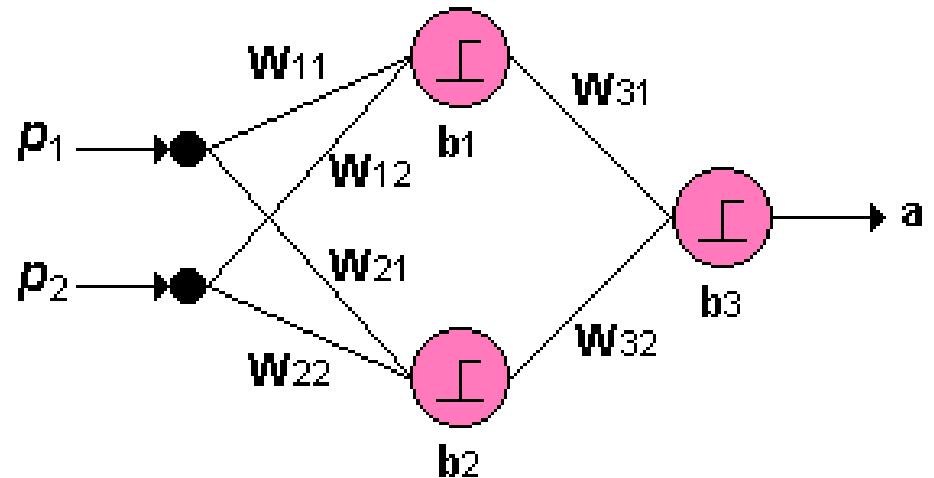
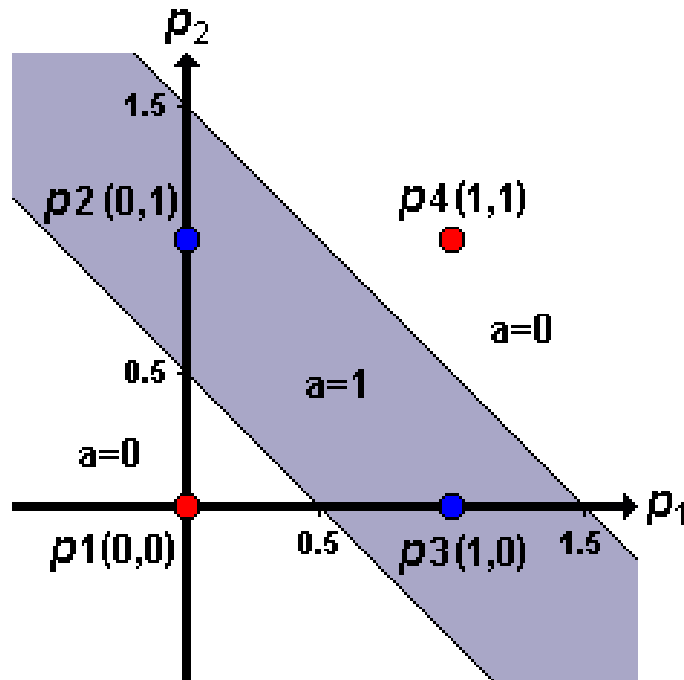
} $A \leftrightarrow 1$

□ Podemos seguir aumentando la potencia:



www.wiphala.net/courses/intelligent_systems/ICSI271/2009-II/class/class_42_perceptron_net.ppt

- El perceptrón multicapa es capaz de resolver el problema del XOR.



$$w_{11} = w_{12} = w_{21} = w_{22} = w_{31} = 1$$

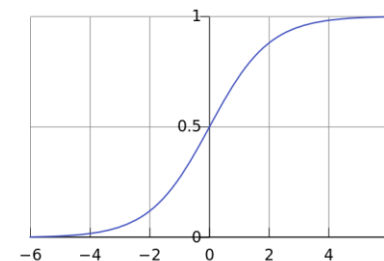
$$w_{32} = -1.5 \quad b_1 = b_3 = 0.5 \quad b_2 = 1.5$$

□ Algoritmo *backpropagation* (I)

- El entrenamiento las redes neuronales multicapa utilizando el **algoritmo *backpropagation* o de retropropagación o de propagación hacia atrás de errores** consta de los siguientes pasos:

1. Decidimos la **estructura** de la red (número de capas y de neuronas en cada capa): L capas, n neuronas en la capa de entrada (capa 1), m en la de salida (capa L).
2. Elegimos una **función de activación** de las neuronas **diferenciable**. Habitualmente de tipo sigmoideo, (es decir, con forma de S), como, p. ej. la función logística:

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



3. Inicializamos los pesos y las polarizaciones w_{ij} de forma **aleatoria** y con **valores pequeños** (-0.5, 0.5).

□ Algoritmo *backpropagation* (II)

4. Generamos los **datos de entrenamiento**: conjunto de tuplas entradas - salidas deseadas, por ejemplo, si son 2 entradas, 2 salidas y M tuplas de entrenamiento:

$$\{(x_{11}, x_{12}), (y_{d11}, y_{d12})\}, \{(x_{21}, x_{22}), (y_{d21}, y_{d22})\}, \dots, \\ \{(x_{M1}, x_{M2}), (y_{dM1}, y_{dM2})\}$$

5. Elegimos un dato de entrenamiento, p. ej. r: $\{(x_{r1}, x_{r2}), (y_{dr1}, y_{dr2})\}$.

6. Calculamos las **salidas de la red** para (x_{r1}, x_{r2}) con los pesos disponibles **propagando los valores desde las neuronas de entrada hacia adelante**, es decir, desde $l=1$ hasta L:

- in_i = entrada que recibe una unidad i
- a_i = salida de la unidad i
- Si i es una neurona de **entrada ($l=1$)** $\rightarrow a_i = x_{ri}$
- En una neurona i de la **capa l (con $l \neq 1$)**:

El sumatorio se realiza sobre todas las unidades j de la capa l - 1

$$in_i = \sum_j w_{ji} a_j, \quad a_i = g(in_i)$$

□ Algoritmo *backpropagation* (III)

7. Calculamos la diferencia entre las salidas de la red para el dato x_r (las salidas de las neuronas i de la **capa de salida**) obtenidas con los pesos actuales (a_i) y las salidas deseadas (y_{dri}), con lo que obtenemos el **vector de error** con el error de cada neurona de salida para dicho dato.
8. **Ajustamos los pesos** de la red de forma que se **minimice el error**. En las **capas de salida**:

$$w(t+1)_{ji} = w_{ji}(t) + \eta \cdot a_j \cdot \Delta_i \quad \text{con} \quad \Delta_i = g'(in_i)(y_{dri} - a_i)$$

Tasa, factor o velocidad
de aprendizaje

Error modificado en la
unidad i

Pero, si la neurona no pertenece a la capa de salida no sabemos cual es el valor de salida esperado → no se puede utilizar la misma fórmula.

□ Algoritmo *backpropagation* (IV)

8. (cont.) ¿Cómo actualizamos los pesos de las conexiones de las **capas ocultas**? Vamos hacia atrás calculando el error Δ_j de cada unidad de la capa oculta $l-1$ a partir de error de las unidades de la capa l con las que está conectada j .

$$w(t+1)_{kj} = w_{kj}(t) + \eta \cdot a_k \cdot \Delta_j \quad \text{con} \quad \Delta_j = g'(in_j) \sum_i w_{ji} \Delta_i$$

Es decir, se considera que cada unidad j es “responsable” del error que tiene cada una de las unidades a las que envía su salida, contribuyendo de forma proporcional a su peso.

Para calcular la modificación de los pesos, se calcula el error en la etapa de salida y se propaga la modificación hacia atrás: **retropropagación**.

9. Repetimos los pasos anteriores para cada par de entrenamiento (**época**) e **iteramos** hasta que el error sea aceptable.

□ Algoritmo *backpropagation* con unidades sigmoides

- Habitualmente las neuronas en este tipo de redes tienen **función de activación sigmoide**, donde:

$$g(x) = \frac{1}{1 + e^{-x}} \rightarrow g'(x) = g(x)(1 - g(x))$$

- En estos casos, al aplicarlo a las neuronas, se cumple que:

$$g'(in_i) = g(in_i)(1 - g(in_i)) = a_i(1 - a_i)$$

Es decir, el valor de la derivada depende solo del valor a su salida, no a su entrada.

- Así, el cálculo de errores modificados queda:

- Para la capa de salida, $\Delta_i(t+1) = a_i(1 - a_i)(y_i - a_i)$

- Para las capas ocultas, $\Delta_j = a_i(1 - a_i) \sum_i w_{ji} \Delta_i$

- Esto significa que no necesitamos almacenar los in_i en el paso de cálculo de las salidas para usarlos en el de retropropagación.

□ **Momentum en el algoritmo *backpropagation***

- Retropropagación es un método de descenso por el gradiente y por tanto existe el problema de los **mínimos locales**.
- Una variante muy común en el algoritmo de retropropagación consiste en introducir un **sumando adicional en la actualización de pesos** que hace que en cada actualización de pesos se tenga también en cuenta la actualización realizada en la iteración anterior.
- Entonces, en la iteración n-ésima, los pesos se actualizan los según la siguiente fórmula:

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)} \quad \text{donde} \quad \Delta w_{ji}^{(t)} = \eta a_j \Delta_i + \alpha \Delta w_{ji}^{(t-1)}$$

$0 < \alpha \leq 1$ es una constante denominada **momentum**.

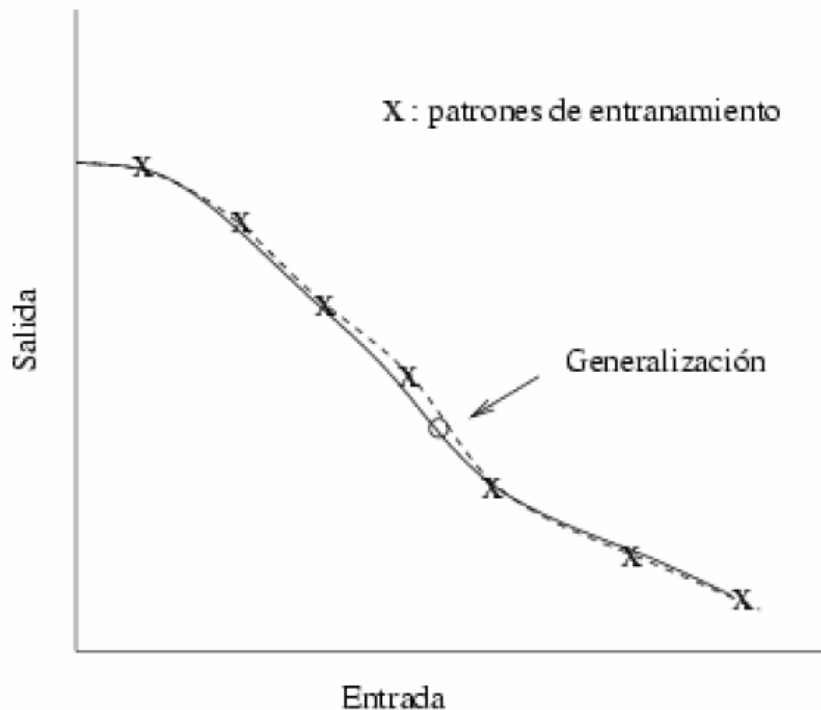
- La técnica del *momentum* puede ser eficaz a veces para **escapar de “pequeños mínimos locales”**, donde una versión sin momentum se estancaría. También tiene efecto **estabilizador** si hay oscilaciones de signo.



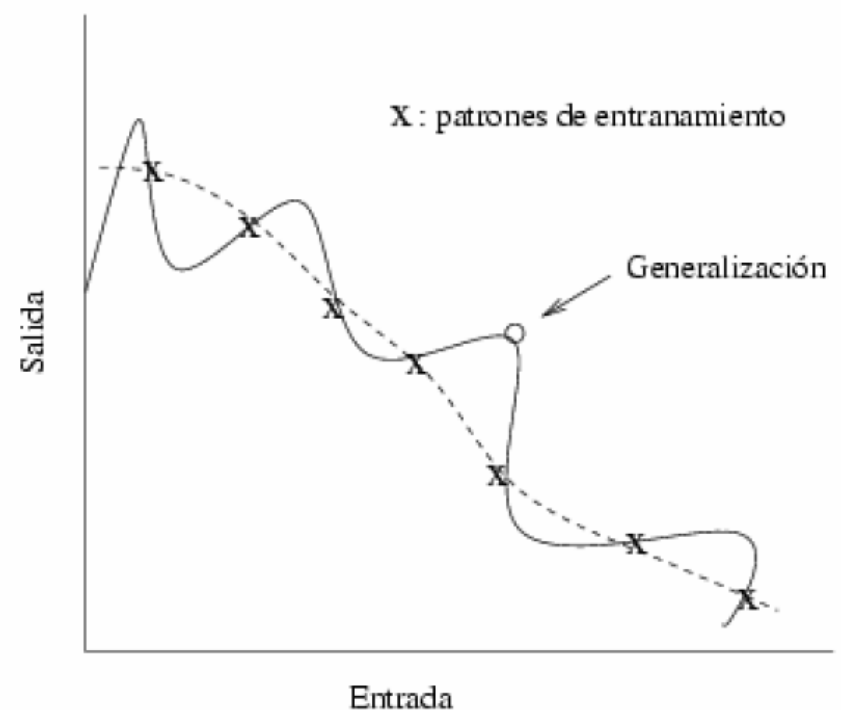
- El algoritmo de backpropagation recorre varias veces (**iteraciones**) el conjunto de entrenamiento: ¿Cuántas?
 - Si son demasiado pocas, la red no entrenará correctamente.
 - Si son demasiadas se **sobreentrenará**: se “aprenderá de memoria” los datos de entrenamiento y no será **capaz de generalizar**.
- Se pueden usar diferentes **criterios de parada** en el algoritmo de retropropagación. Por ejemplo:
 - **Número de iteraciones prefijadas**: no sabremos si se ha entrenado correctamente.
 - Cuando el **error sobre el conjunto de entrenamiento está por debajo de una cota prefijada**. En este caso, se corre el riesgo de **sobreajuste o sobreentrenamiento**.
 - Usar un **conjunto de prueba independiente** para **validar** el error y comprobar el error en dicho conjunto, de forma que se pueda valorar la **capacidad de generalización**.

Redes neuronales multicapa

- Ejemplo de redes con buena y mala capacidad de generalización



Buena capacidad de generalización



Mala capacidad de generalización

<http://eva.evannai.inf.uc3m.es/et/docencia/rn-inf/documentacion/Tema3-MLP.pdf>



- Para estudiar la **capacidad de generalización** se utiliza el **método de validación cruzada** (*cross-validation*). Este método consiste en dividir **los datos de entrenamiento** en dos partes:
 - **Conjunto de entrenamiento (80%)**: se utiliza para determinar los parámetros de la red (pesos).
 - **Conjunto de validación (10%)**: se utiliza para estimar el **error de generalización**, es decir, la tasa de clasificación incorrecta del clasificador con datos diferentes a los utilizados en el proceso de entrenamiento. Se usa en la etapa de entrenamiento para ajustar el modelo, p. ej. para decidir el número iteraciones a realizar sobre los datos de entrenamiento (para ajustar los pesos al máximo sin perder capacidad de generalización).
- Además, en la **etapa de evaluación** se utiliza el **conjunto de test (10%)**. Es un conjunto independiente de los anteriores que se utiliza para evaluar la red: el porcentaje de acierto que se obtenga en este conjunto será el que se proporcione como resultado de la clasificación.



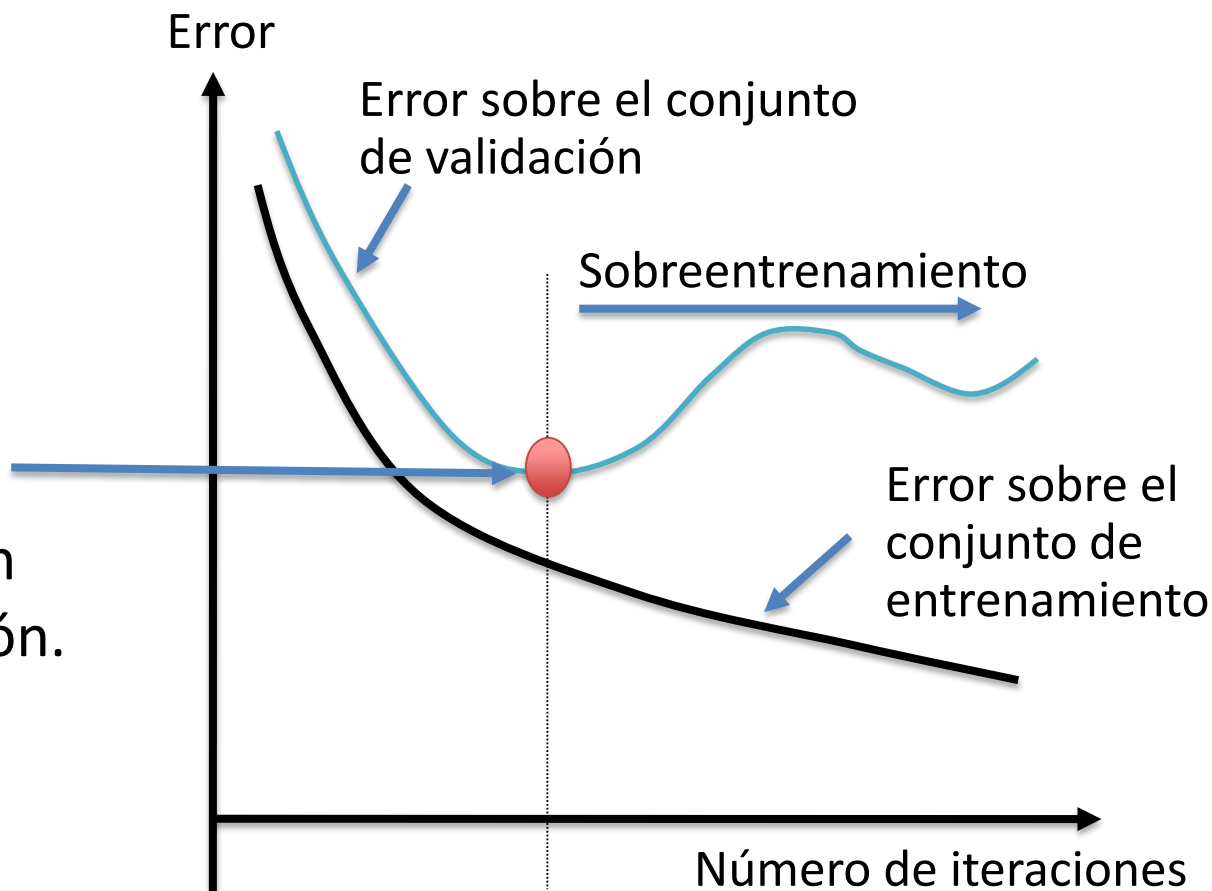
- ❑ **Objetivo final** en la etapa de entrenamiento: que el clasificador consiga un **error de generalización pequeño**.
- ❑ A veces, es necesario exigir **menor ajuste** a los datos de entrenamiento para obtener **mejor generalización**.
- ❑ Para poder hacer esto es necesario ver la **evolución** en la etapa de entrenamiento tanto el **error** que comete la red sobre el **conjunto de entrenamiento**, como el **error de generalización** (el error que comete sobre el conjunto de validación): cada cierto número de ciclos de entrenamiento se pasa el conjunto de validación por la red (sin ajustar pesos) y se mide su error, para **evaluar su capacidad de generalización**.



- Habitualmente el **error sobre el conjunto de entrenamiento decrece monótonamente** durante la fase de entrenamiento, ya que la red va aprendiendo, al adaptar los pesos a dichos datos.
- El **error sobre el conjunto de validación** decrece hasta un punto a partir del cual crece, lo que indica que a partir de ese punto la red se está **sobreentrenando**: se está ajustando con demasiada exactitud a los datos de entrenamiento y está perdiendo la capacidad de generalización. Suele ocurrir cuando:
 - la red tiene muchos parámetros (muchos grados de libertad), o
 - se entrena durante demasiadas iteraciones.
- **Solución:**
 - red con menos capas,
 - redes con menos neuronas (en las capas ocultas), o
 - menos los ciclos de entrenamiento.

□ Posible evolución del error de generalización

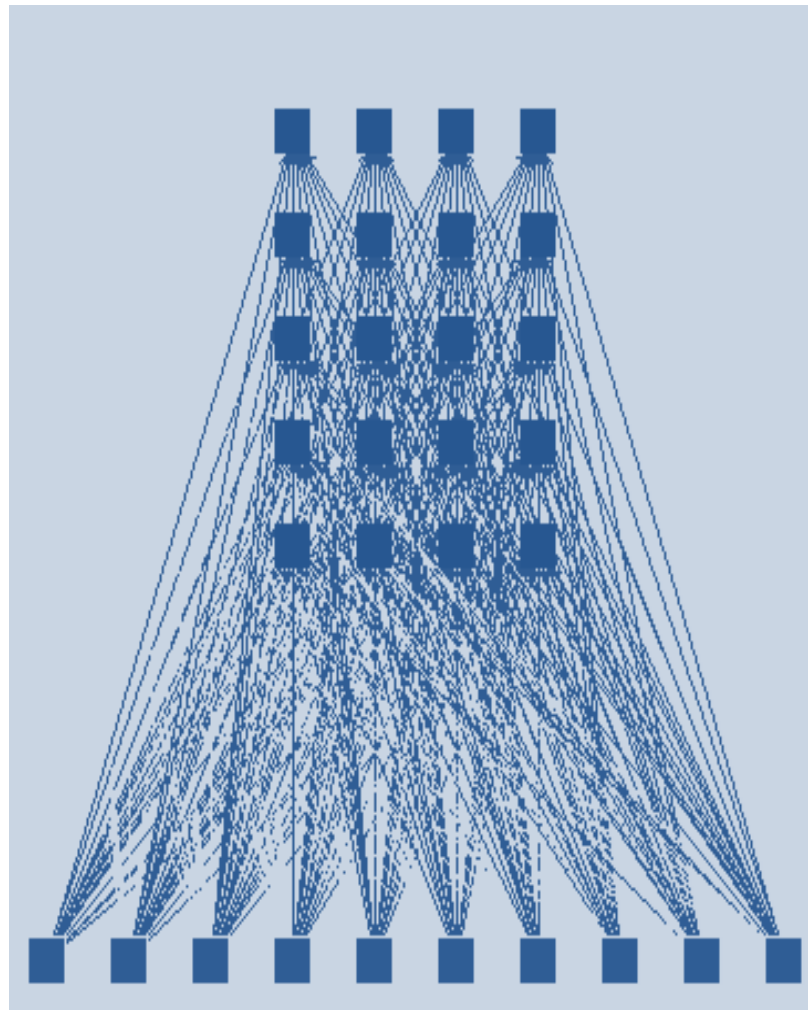
El proceso de entrenamiento debe finalizar cuando se alcance el primer mínimo de la función del error de validación.





- ❑ Los **mapas autoorganizados** o **SOM** (Self-Organizing Map), también llamados **redes de Kohonen** utilizan algoritmos de **aprendizaje no supervisado**: durante el proceso de aprendizaje deben descubrir por sí mismos regularidades o categorías sin refuerzo ni conocimiento del error → se deben **autoorganizar** solamente en función de los datos de entrenamiento.
- ❑ Sus neuronas están **distribuidas de forma regular** en una **rejilla** de, normalmente, dos dimensiones.
- ❑ Las SOM de 2 dimensiones pueden ser de varios tipos: (**hexagonales, rectangulares, ...**), dependiendo de si cada neurona tiene 6, 4 u 8 vecinas.
- ❑ Son redes neuronales **competitivas** (solo una neurona y sus vecinas aprenderán de un determinado dato de entrenamiento): en el **entrenamiento** de la red, los datos se comparan con **el vector de pesos de cada neurona**. La neurona que presenta menor diferencia entre su vector de peso y el datos de entrada es la **neurona ganadora** y se modificarán sus pesos y los de sus vecinas.

- Un **mapa autoorganizado** está compuesto por 2 capas de neuronas:
 - **Capa de entrada:** N neuronas, 1 por cada componente de los datos de entrada. Se encarga de recibir y transmitir a la capa de salida de información de entrada.
 - **Capa de salida:** M neuronas. Procesa la información y forma el mapa de rasgos. Se suelen organizar en forma de mapa bidimensional.



<http://www.gc.ssr.upm.es/inves/neural/ann2/unsupmod/competle/kohonen.htm>

- ❑ Los **mapas autoorganizados** pueden ser descritos formalmente como un mapeado no-lineal, ordenado, suave de los datos de entrada altamente dimensionales hacia los elementos de un *array* regular de baja dimensión.
- ❑ El algoritmo de los mapas autoorganizados consiste en un procedimiento **iterativo** capaz de representar la estructura topológica del espacio de entrada (discreto o continuo) por medio de un conjunto discreto de **prototipos de vectores de peso asociados a neuronas de la red**. Las SOM mapean los **patrones de entradas vecinos en neuronas vecinas**.
- ❑ Cuando una nueva señal x llega, todas las neuronas **compiten** para representarla. La unidad que mejor se ajusta (*Best Matching Unit*) es la neurona que gana la competencia y en conjunto con sus vecinas del *grid* aprenden la señal. Las neuronas vecinas gradualmente se especializarán para representar señales de entradas similares y las representaciones se organizarán y ordenarán en el *array*.

□ Algoritmo de aprendizaje de los mapas autoorganizados (I):

1. Inicialización de los pesos w_{ij}

- con valores pequeños aleatorios, o
- con los pesos de algunas de las entradas.

2. Elegir aleatoriamente un **patrón de entrenamiento** x .

3. Cada neurona calcula la **similitud** entre su vector de pesos w_{ij} y el vector de entrada x_k , usando la distancia Euclídea:

$$d(\mathbf{x}, \mathbf{w}_i) = \|\mathbf{x} - \mathbf{w}_i\| = \sqrt{(x_1 - w_{i1})^2 + \dots + (x_N - w_{iN})^2}$$

4. La **neurona ganadora**, será la de menor distancia al vector de entrada:

$$d(\mathbf{x}, \mathbf{w}_r) \leq d(\mathbf{x}, \mathbf{w}_k), \quad k = 1, 2, \dots, M$$

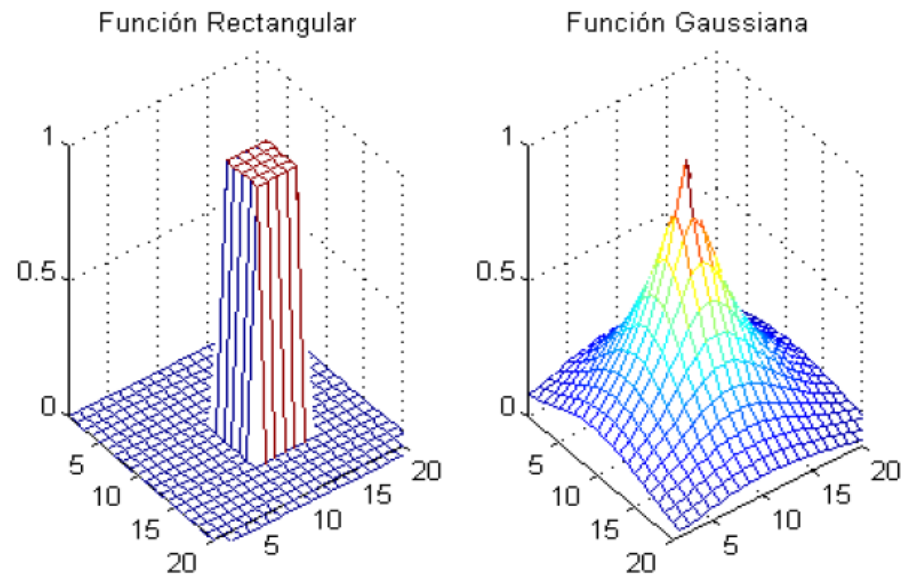
□ Algoritmo de aprendizaje de los mapas autoorganizados (II):

5. La **unidad ganadora y sus vecinas** se adaptan para representar la entrada a través de la **modificación de sus vectores de referencia** hacia la entrada actual.

$$w_{ij}(n+1) = w_{ij}(n) + \eta(n) \cdot h(n) \cdot (x_j - w_{ij}(t))$$

donde:

- $\eta(n)$ es la **velocidad de aprendizaje**,
 - $h(n)$ es la **función de vecindad** (con un pico de amplitud en la neurona vencedora).
 - Tanto $\eta(n)$ como $h(n)$ **decrecen** con el número de iteraciones.
6. Se vuelve al paso 2 hasta que el entrenamiento termina.



Funciones de vecindad más extendidas

□ Algoritmo de aprendizaje competitivo individualizado:

1. Inicialización de los pesos w_{ij} con valores pequeños aleatorios. $K=1$.
2. Elegir aleatoriamente un patrón de entrenamiento x .
3. Cada neurona calcula la similitud entre su vector de pesos w_{ij} y el vector de entrada x_k , usando la distancia Euclídea:

$$d(\mathbf{x}, \mathbf{w}_i) = \|\mathbf{x} - \mathbf{w}_i\| = \sqrt{(x_1 - w_{i1})^2 + \dots + (x_N - w_{iN})^2}$$

4. La neurona ganadora, r , será la de menor distancia:

$$d(\mathbf{x}, \mathbf{w}_r) \leq d(\mathbf{x}, \mathbf{w}_k), \quad k = 1, 2, \dots, M$$

5. Actualización de los pesos de la neurona ganadora:

$$\mathbf{w}_r(k+1) = \mathbf{w}_r(k) + \eta(k)[\mathbf{x}(k) - \mathbf{w}_r(k)]$$

Las demás neuronas **no** actualizan su peso.

6. Calcular la nueva tasa de aprendizaje según: $\eta(k) = \eta_0 (1 - \frac{k}{T})$
7. Si $k=T$ parar: hemos encontrado los vectores sinápticos.

En caso contrario, poner $k=k+1$ y volver al paso 2.