



# **INTRODUCCIÓN A LOS MODELOS COMPUTACIONALES:**

## **CUARTO CURSO DEL GRADO**

### **DE ING. INFORMÁTICA EN COMPUTACION**

# **Algoritmo de retropropagación del error**

**César Hervás-Martínez**  
**Grupo de Investigación AYRNA**

**Departamento de Informática y Análisis**  
**Númérico**  
**Universidad de Córdoba**  
**Campus de Rabanales. Edificio Einstein.**  
**Email: [chervas@uco.es](mailto:chervas@uco.es)**

**2018-2019**



# **INTRODUCCIÓN AL MODELADO MEDIANTE REDES NEURONALES**



# FUNCIONES DE BASE DE REDES NEURONALES



Asociadas a los pesos de capa de entrada a capa oculta

## □ Unidades Sigmoides

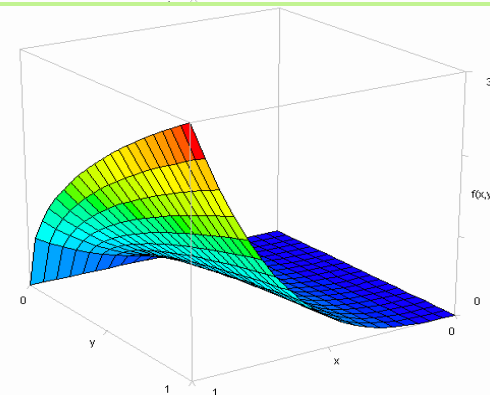
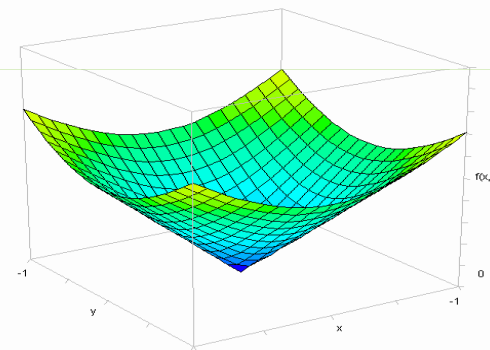
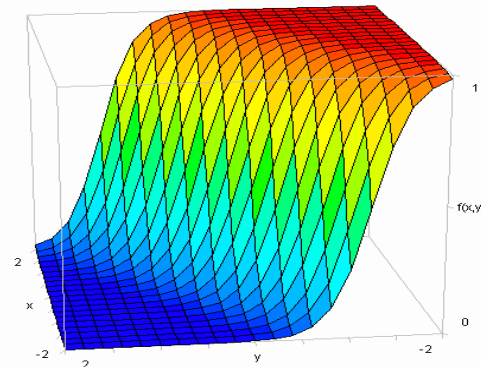
$$B_i(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp\left(-\sum_{i=1}^k (w_{j0} + w_{ji} \times x_i)\right)}$$

## □ Unidades de Base Radial

$$B_j(\mathbf{x}; (\mathbf{c}_j | r_j)) = \exp\left(-\frac{1}{2} \frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{r_j^2}\right)$$

## □ Unidades Producto

$$B_i(\mathbf{x}, \mathbf{w}) = \prod_{i=1}^k x_i^{w_{ji}}$$



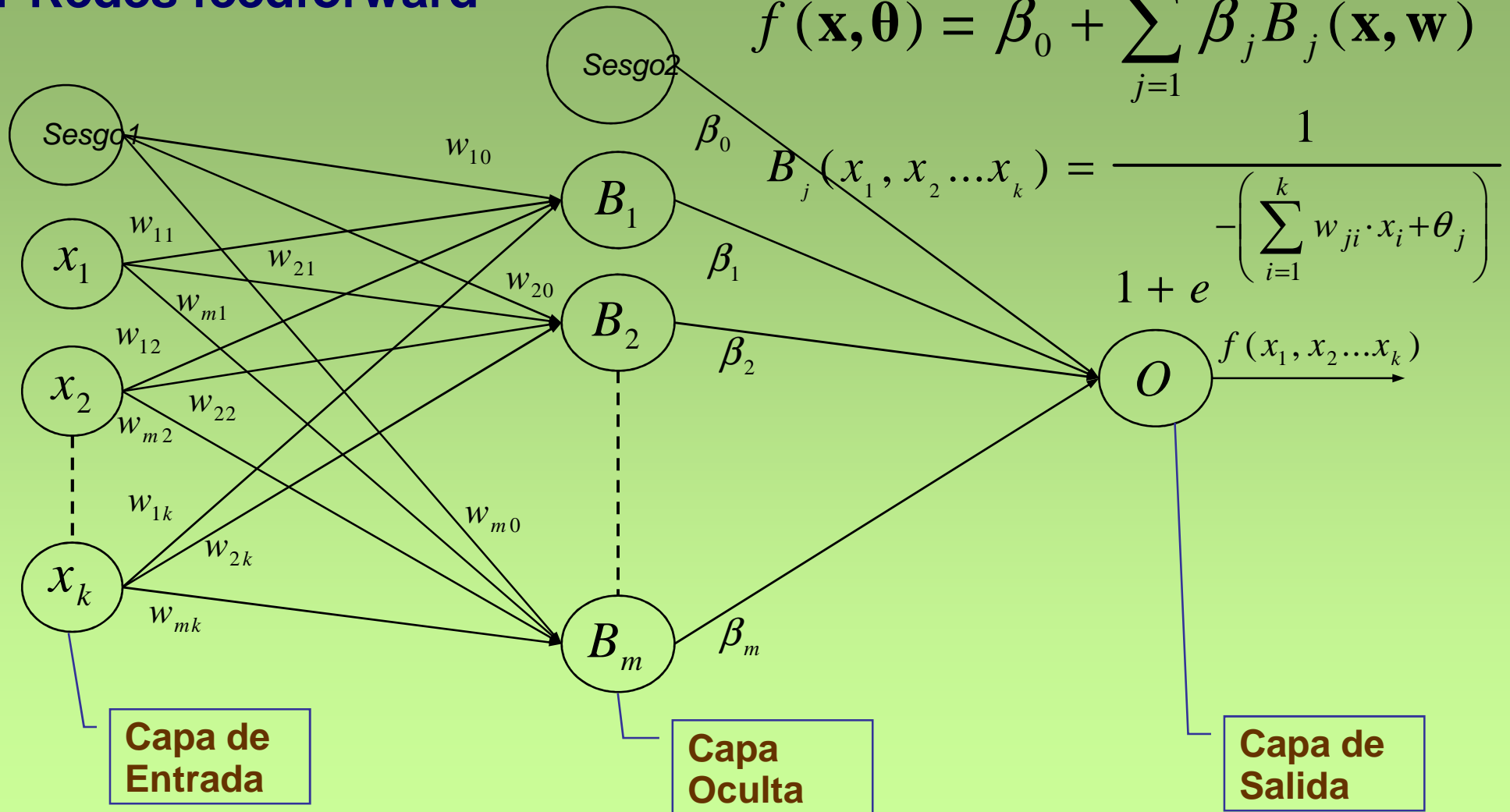


# ESTRUCTURA DE UNA RED NEURONAL

## ESTRUCTURA DE UN RED DE REGRESIÓN



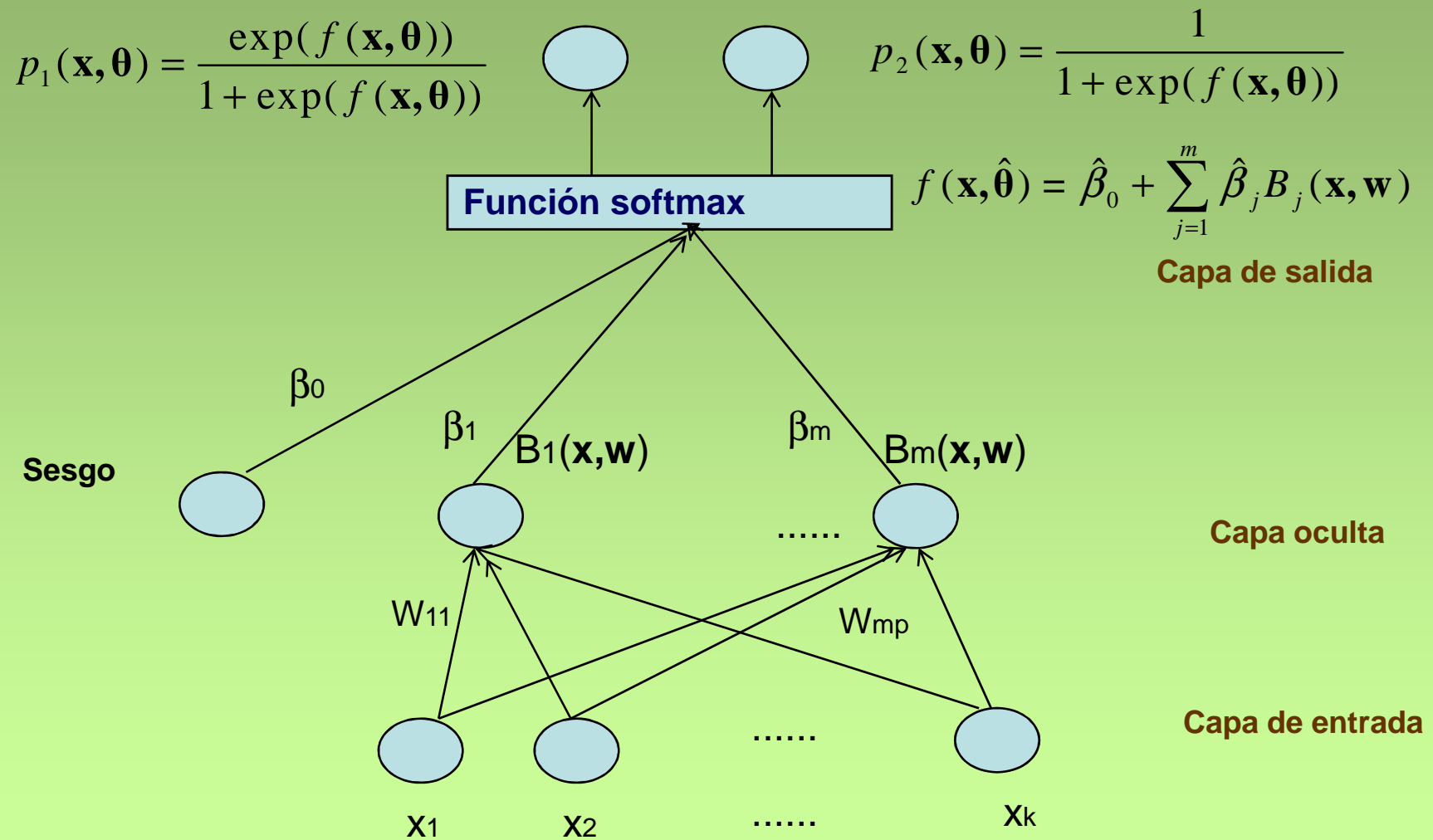
### Redes feedforward





# ESTRUCTURA DE UNA RED NEURONAL

## Representación de un modelo de red para clasificación binaria





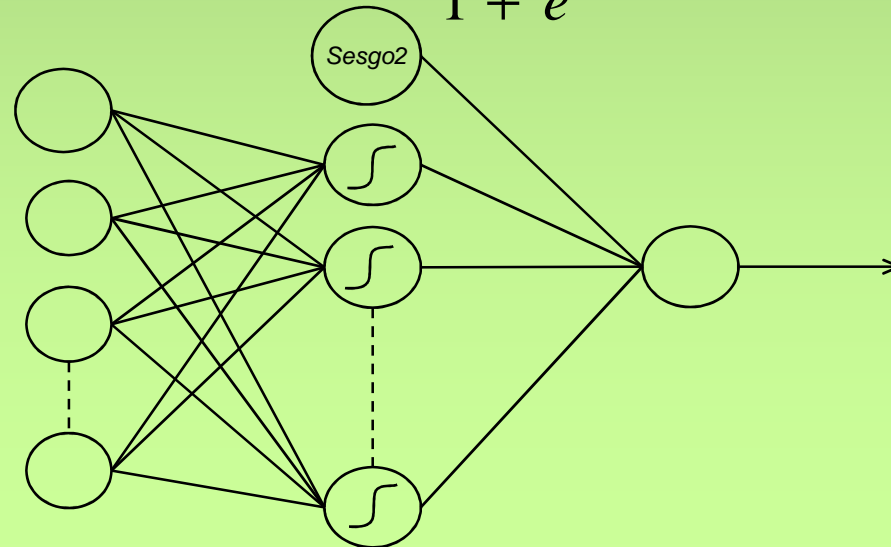
# Redes Neuronales Artificiales



□ Distintas unidades en capa oculta → Distintos modelos Redes Neuronales

## Unidades Sigmoides (MLP)

$$B_i(x_1, x_2 \dots x_k) = \frac{1}{1 + e^{-\left(\sum_{i=1}^k w_{ji} \cdot x_i + \theta_j\right)}}$$



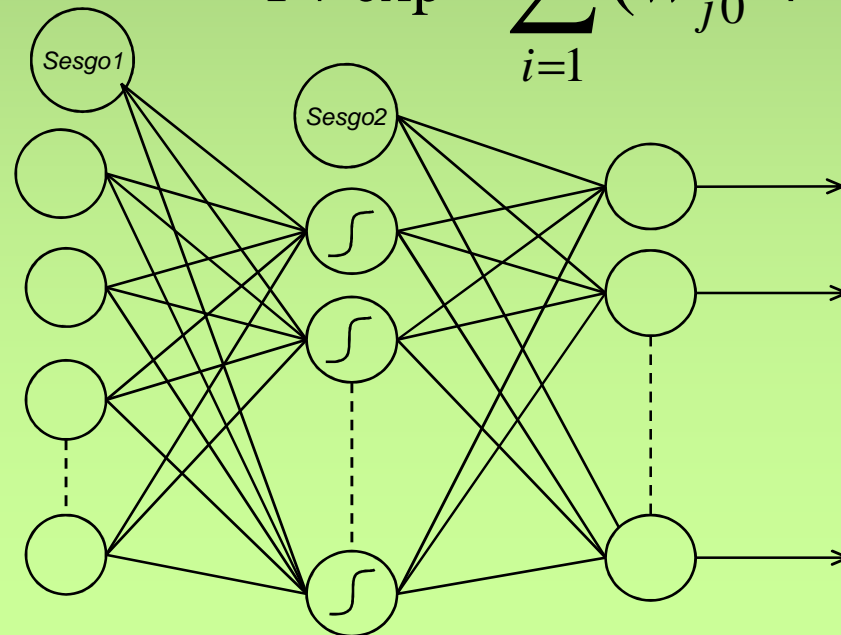


# Redes MLP en Clasificación (Unidades de base sigmoides)



- Distintas funciones de salida en capa oculta → Distintos modelos Redes Neuronales
- **Unidades Sigmoides en Clasificación Multiclase (US)**

$$f_j(x_1, x_2, \dots, x_k) = \frac{1}{1 + \exp - \sum_{i=1}^k (w_{j0} + w_{ji}x_i)}$$





# **APRENDIZAJE MEDIANTE ALGORITMOS DE GRADIENTE DESCENDENTE**





# REDES NEURONALES ARTIFICIALES MLP



## Arquitectura:

Una red *Feedforward* de al menos una capa oculta y con nodos ocultos no-lineales.

Una función de transferencia diferenciable, siendo la más común la **función sigmoide**

Aprendizaje: **supervisado, control del error, regla delta generalizada**

Algoritmo de retropropagación del error:

**Regla de adaptación de los pesos de la red (aproximación mediante un método de gradiente descendente)**



## REDES NEURONALES ARTIFICIALES MLP

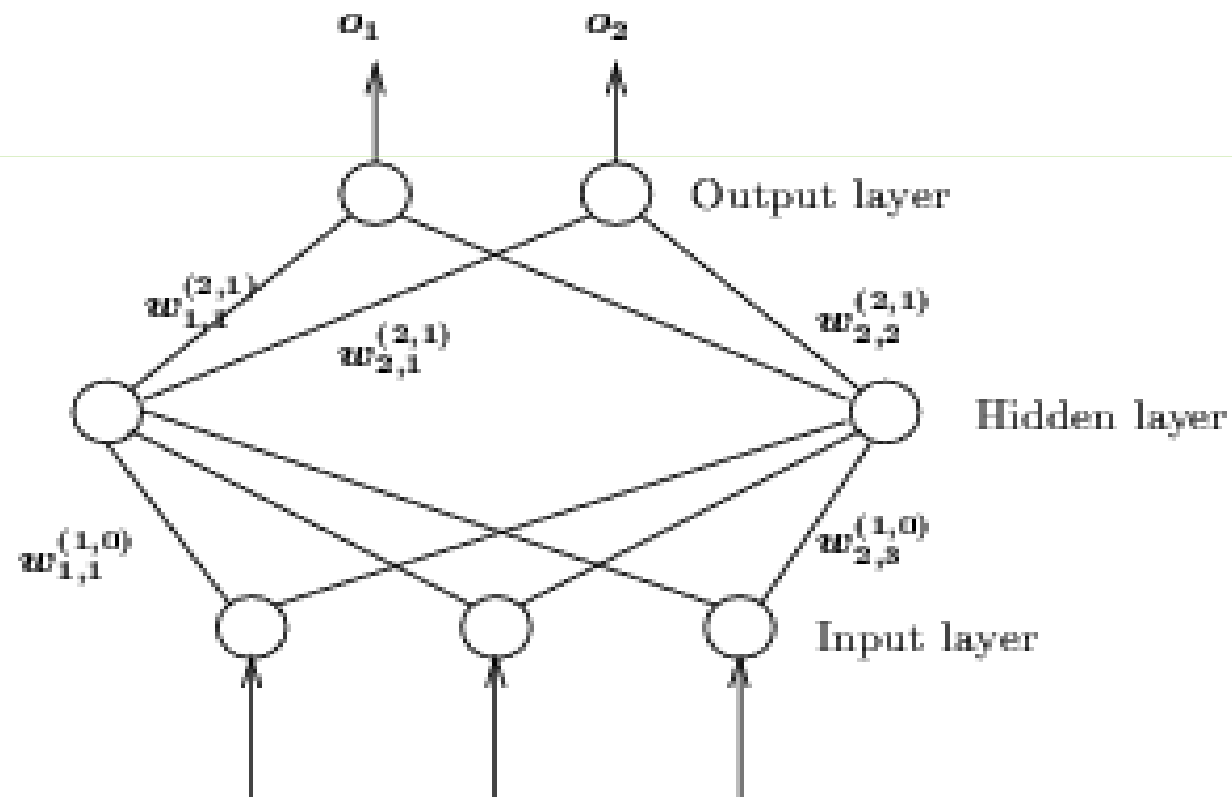


**Pesos: dos matrices de pesos:**

$w^{(1,0)}$  desde la capa de entrada (0) a la capa oculta (1)

$w^{(2,1)}$  desde la capa oculta (1) a la capa de salida (2)

$w_{2,1}^{(1,0)}$  peso desde el nodo 1 de la capa 0 al nodo 2 en la capa 1





# Algoritmo de retro-propagación del error



□ Notaciones: P patrones, K variables de salida, n variables de entrada

- Muestras de entrenamiento: pares de valores  $\{(x_p, d_p) \mid p = 1, \dots, P\}$  de forma tal que el aprendizaje es supervisado
- Patrón de entrada:  $\mathbf{x}_p = (x_{p,1}, \dots, x_{p,n})$
- Patrón de salida:  $\mathbf{o}_p = (o_{p,1}, \dots, o_{p,k})$
- Salida deseada:  $\mathbf{d}_p = (d_{p,1}, \dots, d_{p,k})$
- Error:  $l_{p,j} = o_{p,j} - d_{p,j}$  error de la salida j cuando se aplica el patrón  $\mathbf{x}_p$  a la red.

**Función objetivo:** Suma de errores al cuadrado  $= \sum_{p=1}^P \sum_{j=1}^K (l_{p,j})^2$

La minimización de este error dirige el entrenamiento (cambiando los pesos  $w^{(1,0)}$  y  $w^{(2,1)}$ )



# ALGORITMO DE RETROPROPAGACIÓN



## Unidad Sigmoides

- Como en el perceptrón, la primera parte de una unidad sigmoide computa una combinación lineal de sus entradas.

$$o' = \mathbf{W} \cdot \mathbf{x}$$

A continuación la **unidad sigmoide** computa su salida con la siguiente función.

$$o = \text{sgn}(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + \exp - (w_{10} + w_{11}x_1 + \dots + w_{1k}x_K)}$$

$$\text{o también } S(\mathbf{x}) = \frac{1}{1 + e^{-\sum_{j=0}^K w_{1j}x_j}}$$

- Esta segunda ecuación a menudo se la reconoce como una **función de alisamiento** puesto que transforma un dominio de entradas muy grande a un dominio de salida pequeño.



## Algoritmo de retro-propagación del error



Una función sigmoide tiene la propiedad de que su derivada se expresa con facilidad en términos de su salida, lo que hace la descripción del algoritmo de retropropagación del error mucho más compacta, al hacer uso de esta derivada.

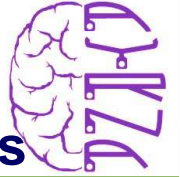
□ Derivada de la función sigmoide :

$$S(x) = \frac{1}{1 + e^{-x}}; \quad S'(x) = S(x)(1 - S(x))$$

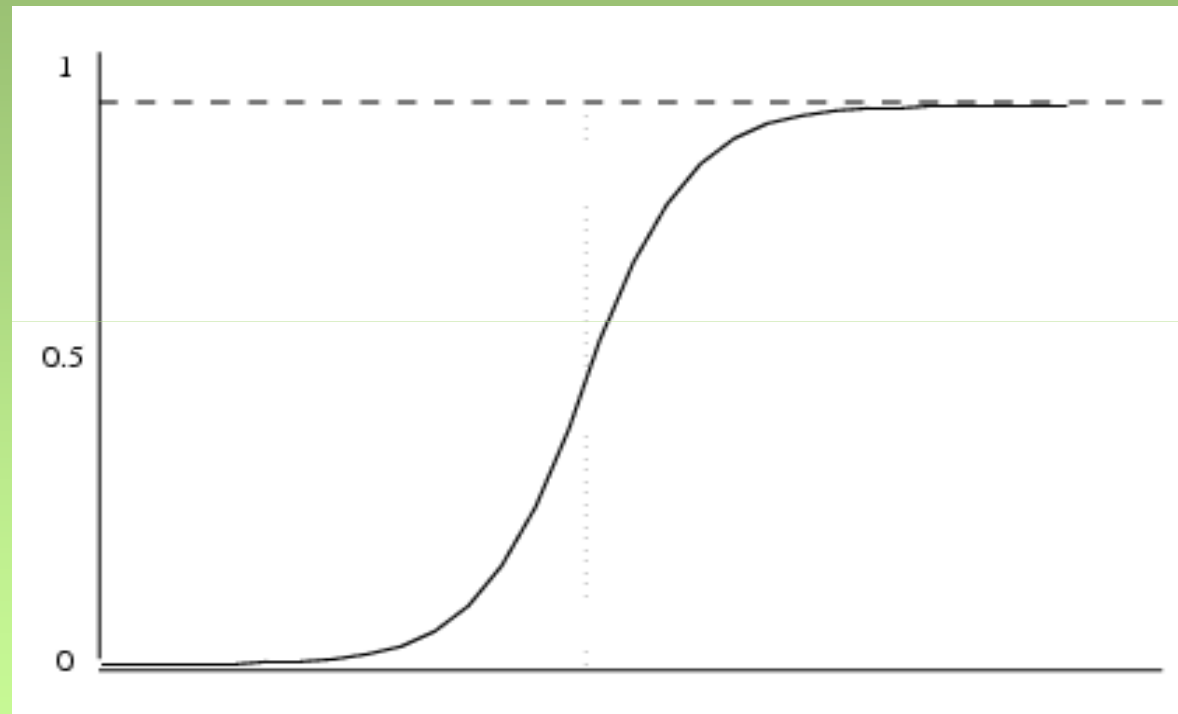
$$\begin{aligned} S'(x) &= -\frac{1}{(1 + e^{-x})^2} \cdot (1 + e^{-x})' = \\ &= -\frac{1}{(1 + e^{-x})^2} \cdot (-e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} = S(x)(1 - S(x)) \end{aligned}$$



## Algoritmo de retro-propagación del error



- Cuando  $|x|$  es suficientemente grande, se mueve hacia una de las dos regiones de saturación, comportándose como un umbral o función de tipo rampa.



Region de saturación

Region de saturación



# ALGORITMO DE RETROPROPAGACIÓN



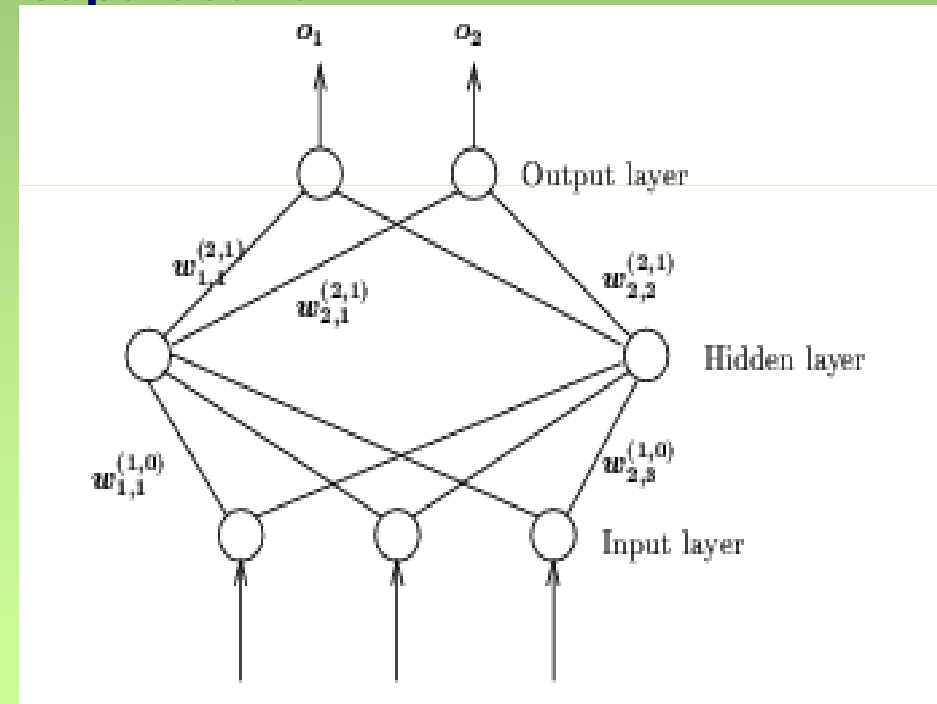
## Computación hacia adelante:

- Se aplica un vector  $x$  de entrada a los nodos de la capa de entrada
- Se calcula el vector de salida  $x^{(1)}$  de la capa oculta

$$x_j^{(1)} = S(\text{net}_j^{(1)}) = S\left(\sum_i w_{j,i}^{(1,0)} x_i\right)$$

- Se calcula el vector de salida  $o$  de la capa de salida

$$o_k = S(\text{net}_k^{(2)}) = S\left(\sum_j w_{k,j}^{(2,1)} x_j^{(1)}\right)$$



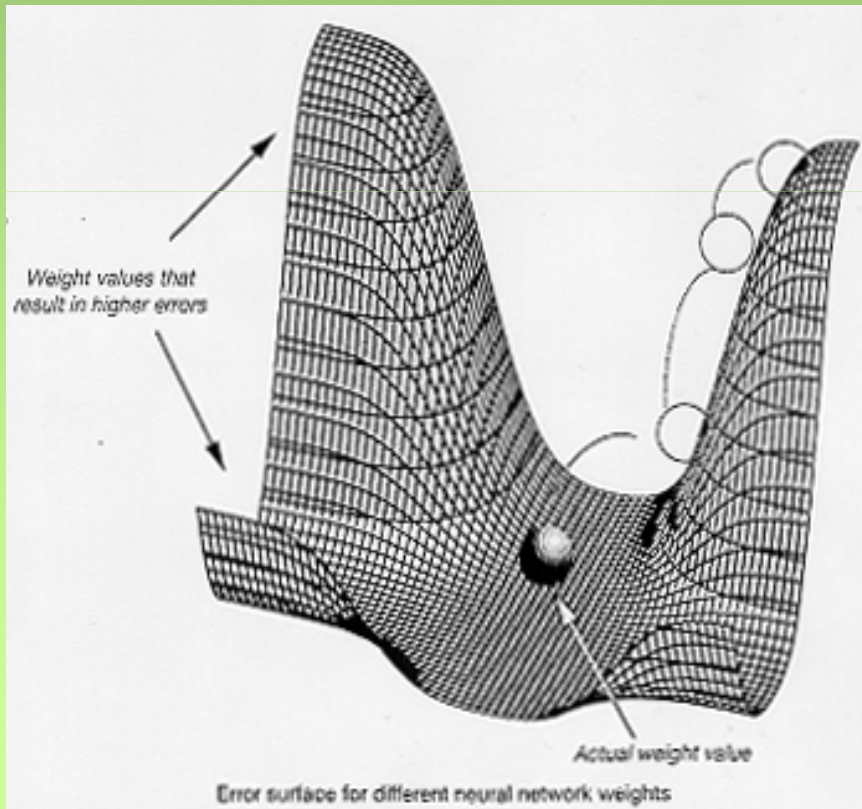
- Se dice que la red es una transformación de la entrada  $x$  a la salida  $o$



## ALGORITMO DE RETROPROPAGACIÓN DEL ERROR



**Back-propagation** es un procedimiento para ajustar los pesos de la red con el fin de minimizar el error de salida. La técnica empleada es un algoritmo de búsqueda local de **gradiente descendente**: se calcula la derivada de las funciones de transferencia y, para cada peso, se decide si añadir o restar una pequeña cantidad con el objetivo de reducir el error total.



Supongamos que podemos ajustar dos pesos ( $w_1$  y  $w_2$ ). La superficie de la izquierda es un ejemplo de como puede variar el error como resultado de la combinación de esos pesos diferentes.

Observamos que la dirección (positiva o negativa) y las etapas de las pendientes se pueden obtener utilizando derivadas parciales del error con respecto a  $w_1$  y  $w_2$

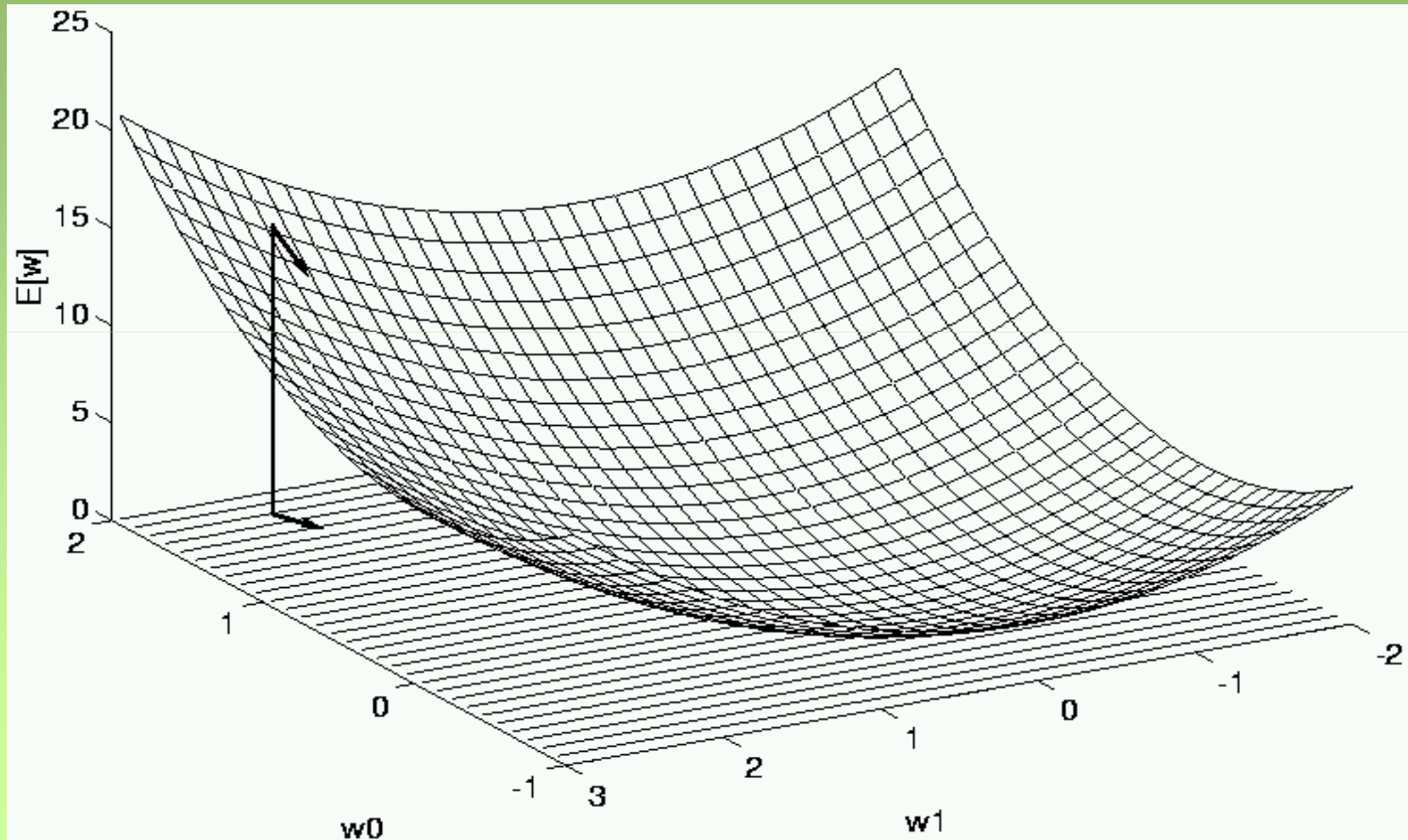




## GRADIENTE DESCENDENTE



Aquí mostramos otra gráfica para ver como el error de predicción puede cambiar cuando cambian los valores de los pesos:





# ALGORITMO DE RETROPROPAGACIÓN



Objetivo del aprendizaje:

**Minimizar la suma de errores al cuadrado**

$$\sum_{p=1}^P \sum_{j=1}^K (l_{p,j})^2$$

**para el número de patrones de  
entrenamiento P tanto como sea  
posible (a cero si fuese posible)**

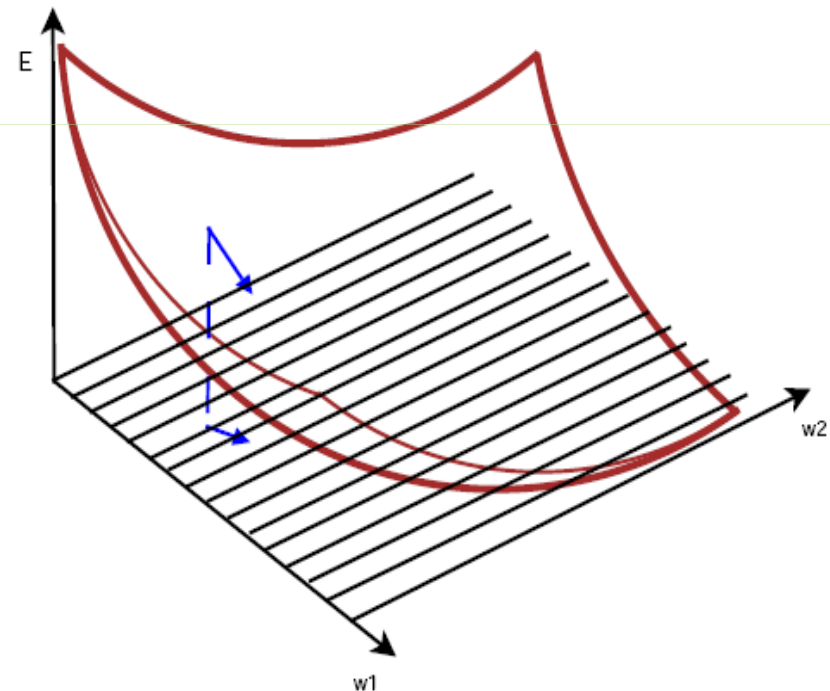


Figure 2: Hypothesis surface



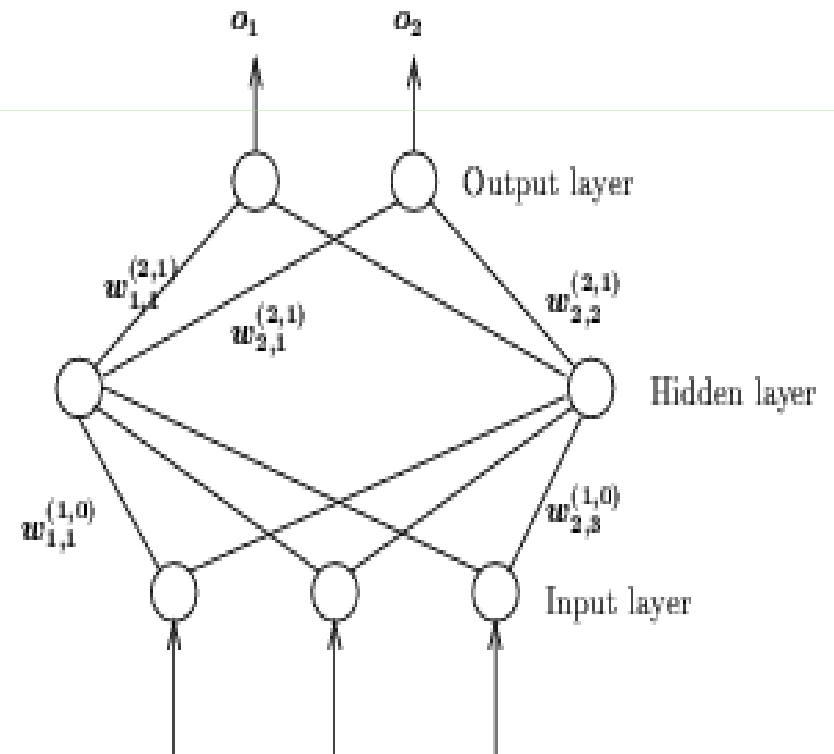
## ALGORITMO DE RETROPROPAGACIÓN: BP



□ Objetivo del algoritmo BP:

- Cambiar los pesos en  $w^{(2,1)}$  (de la capa oculta a la capa de salida):  
Utilizando la regla delta en una sola capa oculta de la red minimizando la suma de los errores cuadráticos

- La regla delta no se puede utilizar “*a priori*” para cambiar los pesos  $w^{(1,0)}$  (desde la capa de entrada a la capa oculta) porque no conocemos los valores deseados de los nodos de la capa oculta





## ALGORITMO DE RETROPROPAGACIÓN: BP



□ **Solucion:** Propagamos los errores de los nodos de la capa de salida hacia los nodos de la capa oculta impulsando los cambios de los pesos en  $w^{(1,0)}$  (utilizando de nuevo la regla delta), definiendo este procedimiento como aprendizaje por retropropagación del error), **BACKPROPAGATION (BP) learning**

- La clave está por tanto en como calcular los errores en los nodos de la capa oculta
- El error BP puede continuar retropropagandose si la red tiene más de una capa oculta
- Fué propuesto primero por Werbos (1974), y más adelante por Rumelhart, Hinton, and Williams (1986)



## ALGORITMO BP: Regla delta generalizada



Consideraremos un método de **aprendizaje secuencial**: para un patrón muestral dado  $(\mathbf{x}_p, \mathbf{d}_p)$ . La suma del error cuadrático cometido es

$$E = \sum_k (l_{p,k})^2$$

- Cambiando los pesos mediante un algoritmo de gradiente descendente
- Para los pesos en  $w^{(2,1)}_{k,j}$  :  $\Delta w^{(2,1)}_{k,j} \propto (-\partial E / \partial w^{(2,1)}_{k,j})$
- Para los pesos en  $w^{(1,0)}_{j,i}$  :  $\Delta w^{(1,0)}_{j,i} \propto (-\partial E / \partial w^{(1,0)}_{j,i})$
- Derivación mediante la regla de actualización de los pesos  $w^{(2,1)}_{k,j}$ , puesto que  $E$  es una función de  $l_k = d_k - o_k$ , y  $d_k - o_k$  es una función de  $net_k^{(2)}$ , y  $net_k^{(2)}$  es una función de  $w^{(2,1)}_{k,j}$ , si utilizamos la regla de la cadena, tenemos

$$\begin{aligned} \frac{\partial E}{\partial w^{(2,1)}_{k,j}} &= \frac{\partial E}{\partial (d_k - o_k)} \frac{\partial (d_k - o_k)}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial w^{(2,1)}_{k,j}} \\ &= -2(d_k - o_k) \mathcal{S}'(net_k^{(2)}) x_j^{(1)}. \end{aligned} \quad o_k = \mathcal{S}(net_k^{(2)}) = \mathcal{S}\left(\sum_j w^{(2,1)}_{k,j} x_j^{(1)}\right)$$



## ALGORITMO BP: Regla delta generalizada detallada



□ Consideraremos un método de aprendizaje secuencial: para un patrón muestral dado  $(\mathbf{x}_p, \mathbf{d}_p)$ . La suma del error cuadrático cometido es

$$E = \sum_k (l_{p,k})^2 = \sum_{k=1}^K (d_k - o_k)^2 = \sum_{k=1}^K (d_k - S(\sum_{j=1}^M w_{k,j}^{(2,1)} x_j^{(1)}))^2 =$$

$$\text{por tanto } -\frac{\partial E}{\partial w_{k,j}^{(2,1)}} = -\frac{\partial E}{\partial (d_k - o_k)} \frac{\partial (d_k - o_k)}{\partial o_k} \frac{\partial o_k}{\partial w_{k,j}^{(2,1)}} =$$

$$= -2(d_k - o_k)(-1) \frac{\partial S(\sum_{j=1}^M w_{k,j}^{(2,1)} x_j^{(1)})}{\partial w_{k,j}^{(2,1)}} = 2(d_k - o_k) S'(\sum_{j=1}^M w_{k,j}^{(2,1)} x_j^{(1)}) x_j^{(1)} =$$

$$= 2(d_k - o_k) S(\sum_{j=1}^M w_{k,j}^{(2,1)} x_j^{(1)}) (1 - S(\sum_{j=1}^M w_{k,j}^{(2,1)} x_j^{(1)})) x_j^{(1)}$$

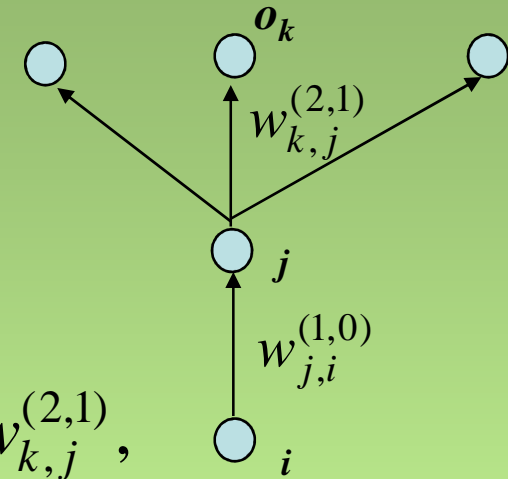
$$\begin{aligned} \frac{\partial E}{\partial w_{k,j}^{(2,1)}} &= \frac{\partial E}{\partial (d_k - o_k)} \frac{\partial (d_k - o_k)}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial w_{k,j}^{(2,1)}} \\ &= -2(d_k - o_k) \mathcal{S}'(net_k^{(2)}) x_j^{(1)}. \end{aligned}$$



## ALGORITMO BP: Regla delta generalizada



- Derivación mediante la regla de cambio para  $w_{j,i}^{(1,0)}$
- consideramos el nodo oculto  $j$ :
- el peso  $w_{j,i}^{(1,0)}$  influye en  $net_j^{(1)}$
- se envía  $S(net_j^{(1)})$  a todos los nodos de salida
- todos los  $K$  terminos en  $E$  son funciones de  $w_{j,i}^{(1,0)}$



$$E = \sum_k (d_k - o_k)^2, \quad o_k = S(net_k^{(2)}), \quad net_k^{(2)} = \sum_j x_j^{(1)} w_{k,j}^{(2,1)},$$

$$x_j^{(1)} = S(net_j^{(1)}), \quad net_j^{(1)} = \sum_i x_i w_{j,i}^{(1,0)}$$

Regla de la cadena

$$\frac{\partial E}{\partial w_{j,i}^{(1,0)}} = \sum_{k=1}^K \left\{ \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial x_j^{(1)}} \frac{\partial x_j^{(1)}}{\partial net_j^{(1)}} \frac{\partial net_j^{(1)}}{\partial w_{j,i}^{(1,0)}} \right\}$$

$$\frac{\partial E}{\partial w_{j,i}^{(1,0)}} = \sum_{k=1}^K \left\{ -2(d_k - o_k) \mathcal{S}'(net_k^{(2)}) w_{k,j}^{(2,1)} \mathcal{S}'(net_j^{(1)}) x_i \right\}$$



## ALGORITMO BP: Regla delta generalizada detallada



$$E = \sum_k (l_{p,k})^2 = \sum_{k=1}^K (d_k - o_k)^2 = \sum_{k=1}^K (d_k - S(\sum_{j=1}^M w_{k,j}^{(2,1)} x_j^{(1)}))^2 =$$

$$\text{por tanto } -\frac{\partial E}{\partial w_{i,j}^{(0,1)}} = \sum_{k=1}^K -\frac{\partial E}{\partial (d_k - o_k)} \frac{\partial (d_k - o_k)}{\partial o_k} \frac{\partial o_k}{\partial x_j^{(1)}} \frac{\partial x_j^{(1)}}{\partial net_j^{(1)}} \frac{\partial net_j^{(1)}}{\partial w_{i,j}^{(0,1)}} =$$

$$\left( \begin{array}{l} \text{pero } o_k = S(net_k^{(2)}), net_k^{(2)} = \sum_j x_j^{(1)} w_{k,j}^{(2,1)}, \\ x_j^{(1)} = S(net_j^{(1)}), net_j^{(1)} = \sum_i x_i w_{j,i}^{(1,0)} \end{array} \right) \text{ luego}$$

$$= \sum_{k=1}^K -2(d_k - o_k)(-1) \frac{\partial S(\sum_{j=1}^M w_{k,j}^{(2,1)} x_j^{(1)})}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial x_j^{(1)}} \frac{\partial x_j^{(1)}}{\partial net_j^{(1)}} \frac{\partial net_j^{(1)}}{\partial w_{j,i}^{(1,0)}} =$$

$$= \sum_{k=1}^K -2(d_k - o_k)(-1) S'(net_k^{(2)}) w_{k,j}^{(2,1)} S'(net_j^{(1)}) x_i$$

Regla de la  
cadena

$$\frac{\partial E}{\partial w_{j,i}^{(1,0)}} = \sum_{k=1}^K \left\{ -2(d_k - o_k) \frac{\partial E}{\partial o_k} \frac{\partial S(net_k^{(2)})}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial x_j^{(1)}} \frac{\partial x_j^{(1)}}{\partial net_j^{(1)}} \frac{\partial net_j^{(1)}}{\partial w_{j,i}^{(1,0)}} \right\}$$





## ALGORITMO DE RETROPROPAGACIÓN



- Entonces, el vector de pesos de la red se puede adaptar mediante:

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \nabla E(\mathbf{w})$$

- Donde  $\nabla E(\mathbf{w})$  es el gradiente de la función de error E:

$$\nabla E(\mathbf{w}) = \left( \frac{\partial E}{\partial w_0}, \dots, \frac{\partial E}{\partial w_m} \right)$$

Para cada valor k se puede adaptar el peso mediante:

$$w_k = w_k + \Delta w_k$$

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



## ALGORITMO BP: Regla delta generalizada



- Reglas del cambio:

- para los pesos de las capas (oculta-salida)  $w^{(2,1)}$  :

$$\frac{\partial E}{\partial w_{k,j}^{(2,1)}} = -2(d_k - o_k) \mathcal{S}'(net_k^{(2)}) x_j^{(1)}$$

$x_j^{(1)}$  = salida del valor del patrón  $x$  por el nodo  $j$  de la capa oculta

- $\Delta w_{k,j}^{(2,1)} = \eta \times \delta_k \times x_j^{(1)}$  donde  $\delta_k = (d_k - o_k) \mathcal{S}'(net_k^{(2)})$

- para los pesos de las capas (entrada-oculta)  $w^{(1,0)}$  :

$$\frac{\partial E}{\partial w_{j,i}^{(1,0)}} = \sum_{k=1}^K \left\{ -2(d_k - o_k) \mathcal{S}'(net_k^{(2)}) w_{k,j}^{(2,1)} \mathcal{S}'(net_j^{(1)}) x_i \right\}$$

- $\Delta w_{j,i}^{(1,0)} = \eta \times \mu_j \times x_i$   $\mu_j = (\sum_k \delta_k w_{k,j}^{(2,1)}) \mathcal{S}'(net_j^{(1)})$

donde Suma ponderada de errores de la capa de salida

## Algorithm Backpropagation;

Start with randomly chosen weights;

while MSE is unsatisfactory

and computational bounds are not exceeded, do

for each input pattern  $x_p$ ,  $1 \leq p \leq P$ ,

Compute hidden node inputs ( $net_{p,j}^{(1)}$ );

Compute hidden node outputs ( $x_{p,j}^{(1)}$ );

Compute inputs to the output nodes ( $net_{p,k}^{(2)}$ );

Compute the network outputs ( $o_{p,k}$ );

Modify outer layer weights:

$$\Delta w_{k,j}^{(2,1)} = \eta (d_{p,k} - o_{p,k}) \mathcal{S}'(net_{p,k}^{(2)}) x_{p,j}^{(1)}$$

Modify weights between input & hidden nodes:

$$\Delta w_{j,i}^{(1,0)} = \eta \sum_k \left( (d_{p,k} - o_{p,k}) \mathcal{S}'(net_{p,k}^{(2)}) w_{k,j}^{(2,1)} \right) \mathcal{S}'(net_{p,j}^{(1)}) x_{p,i}$$

end-for

end-while.

**Nota:** Si  $S$  es una función sigmoide,  
entonces  $S'(x) = S(x)(1 - S(x))$



Salida del valor del patrón p  
por el nodo j de la capa oculta



## ALGORITMO BP: Regla delta generalizada



Clasificación de patrones:

- Dos clases: 1 nodo de salida
- J clases: codificación “1 de J” representada como  $(0, \dots, 0, 1, 0, \dots, 0)$
- Con función sigmoide en la capa de salida, (no tenemos una función softmax) los nodos en la capa de salida nunca serán 1 o 0, sino  $1 - \varepsilon$  o  $\varepsilon$
- La reducción del error se muestra más lenta cuando nos movemos en las regiones de saturación (cuando  $\varepsilon$  es pequeño).
- Para conseguir un aprendizaje rápido, damos una determinada cota de error  $\varepsilon$ , y definimos un error
$$l_{p,k} = 0 \quad \text{si} \quad |d_{p,k} - o_{p,k}| \leq \varepsilon$$
- Cuando se clasifica un patrón  $x$  mediante una red entrenada con el algoritmo BP, la red lo clasifica en la  $k$ -ésima clase si  $l_k < l_l$  para todo  $l \neq k$



## FORTALEZAS DE UNA RED ENTRENADA MEDIANTE UN ALGORITMO BP



- ❑ Gran poder de representación
  - Cualquier función  $L_2$  puede ser representada mediante una red MLP entrenada mediante BP
  - Muchas de estas funciones se pueden aproximar mediante un algoritmo de aprendizaje BP (una aproximación de un algoritmo de gradiente descendente)
- ❑ Amplia aplicabilidad del aprendizaje BP
  - Sólo se necesita que tengamos un buen conjunto de patrones de entrenamiento
  - No se necesita conocimiento *a priori* substancial, o un conocimiento o entendimiento del propio dominio del problema (problemas mal estructurados)



## **FORTALEZAS DE UNA RED ENTRENADA MEDIANTE UN ALGORITMO BP**



### **Amplia aplicabilidad del aprendizaje BP**

- **Tolerancia a ruido y a datos perdidos en el conjunto de entrenamiento (leve degradación)**
- **Es fácil de implementar el fundamento del algoritmo**

### **Buen poder de generalización**

- **A menudo produce resultados precisos para patrones no situados en el conjunto de entrenamiento**



## DEBILIDADES DE UNA RED ENTRENADA MEDIANTE UN ALGORITMO BP



- ❑ El algoritmo a veces tarda mucho en converger

El entrenamiento para funciones complejas necesita a menudo cientos o miles de **épocas** (paso de todos los patrones de entrenamiento por el algoritmo)

- ❑ La red es esencialmente una caja negra

- Puede proporcionar una transformación o función no lineal entre los vectores de entrada y las clases de salida ( $x$ ,  $o$ ) pero no tenemos información de porque un patrón particular  $x$  es transformado a una clase particular de salida  $o$ .
- De esta forma no puede proporcionar, en general, una relación intuitiva causa/efecto que explique el resultado obtenido.



## DEBILIDADES DE UNA RED ENTRENADA MEDIANTE UN ALGORITMO BP



La red es esencialmente una caja negra

- Esto es debido a que los nodos ocultos y los pesos aprendidos no tienen una clara semántica.
  - Lo que podemos aprender son parámetros operacionales, no generales, del conocimiento abstracto de un dominio.
- A diferencia de otros métodos estadísticos, **no existe una teoría bien fundamentada que nos asegure la calidad del aprendizaje BP**
  - ¿Cual es el nivel de confianza que uno puede tener para una red entrenada mediante BP con el error final  $E$  (el cual puede estar o no cerca de cero)?
  - ¿Cual es el nivel de confianza de la salida  $o$  calculada a partir de una entrada  $x$  utilizando tal red?





## DEBILIDADES DE UNA RED ENTRENADA MEDIANTE UN ALGORITMO BP



- ❑ Problema con la aproximación de gradiente descendente.
- ❑ Solo garantiza que reduce el error total de entrenamiento a un mínimo local (E puede no quedar reducida a cero)
  - No puede escapar del estado de error del mínimo local.
  - No toda función que es representable puede ser aprendida.

Como es este problema de malo: Dependerá de la forma de la función de error, pues si presenta muchos valles y simas, fácilmente quedará atrapado el algoritmo en un mínimo local.



## DEBILIDADES DE UNA RED ENTRENADA MEDIANTE UN ALGORITMO BP



Posibles remedios:

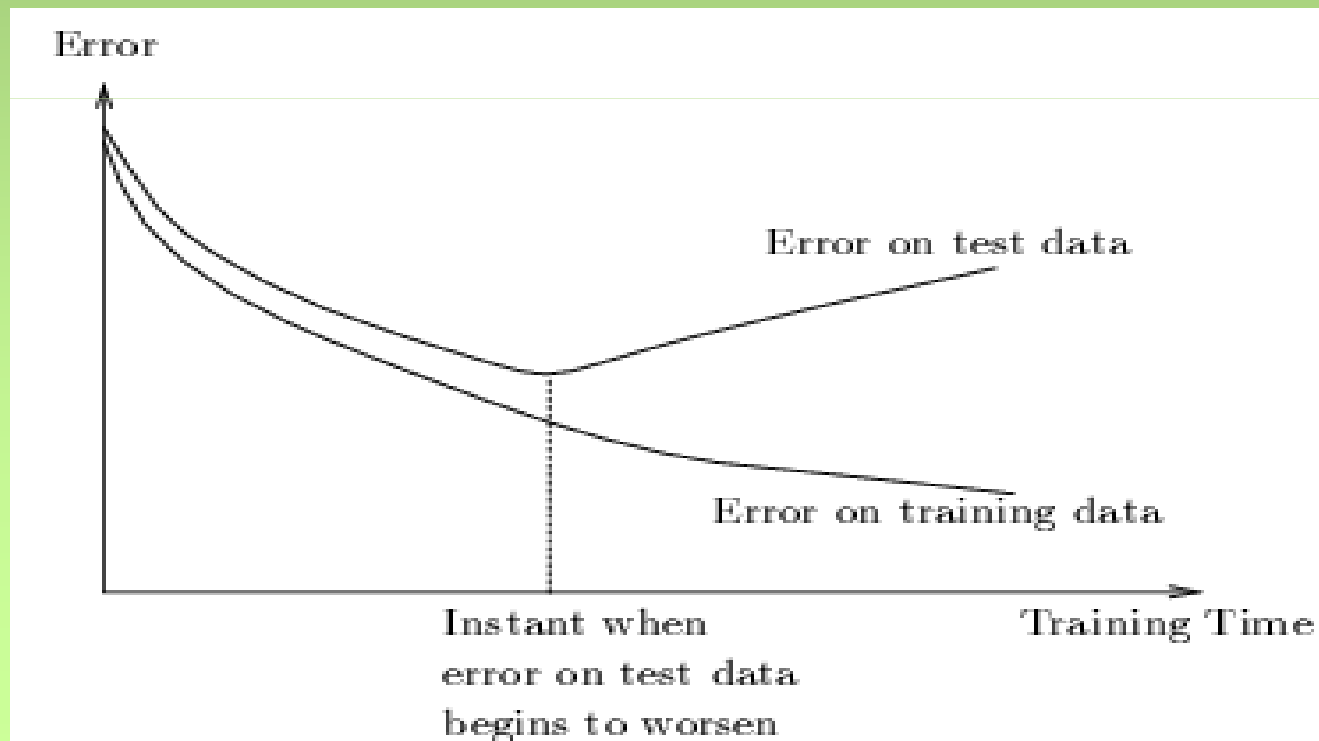
- Probar redes con diferentes números de capas y/o nodos ocultos (pudiendo dar lugar a diferentes funciones de error, unas mejores que otras)
- Probar diferentes pesos iniciales para el algoritmo BP (estos serán diferentes puntos de inicio sobre la superficie de error)
- Forzar una salida del mínimo local mediante por ejemplo una perturbación aleatoria (p.e., enfriamiento simulado)



## ANALISIS DE LA GENERALIZACIÓN



- ❑ La capacidad de generalización de la red no esta garantizada aun cuando el error de entrenamiento sea 0.
  - Problema de sobrefijación o sobreentrenamiento: la red entrenada fija o clasifica las muestras de entrenamiento perfectamente ( $E$  se reduce a 0) pero no da salidas adecuadas para patrones no pertenecientes al conjunto de entrenamiento





## ANALISIS DE LA GENERALIZACIÓN



- Posibles remedios:
  - Mas y mejores muestras
  - Utilizar redes de tamaño lo más pequeñas posibles
  - Utilizar una cota de error mayor (forzando una terminación más rápida, menos épocas)
  - Introducir ruido en las muestras
    - modificar  $(x_1, \dots, x_n)$  por  $(x_1 \alpha_1, \dots, x_n \alpha_n)$  donde  $\alpha_n$  son variables aleatorias que generen pequeños valores

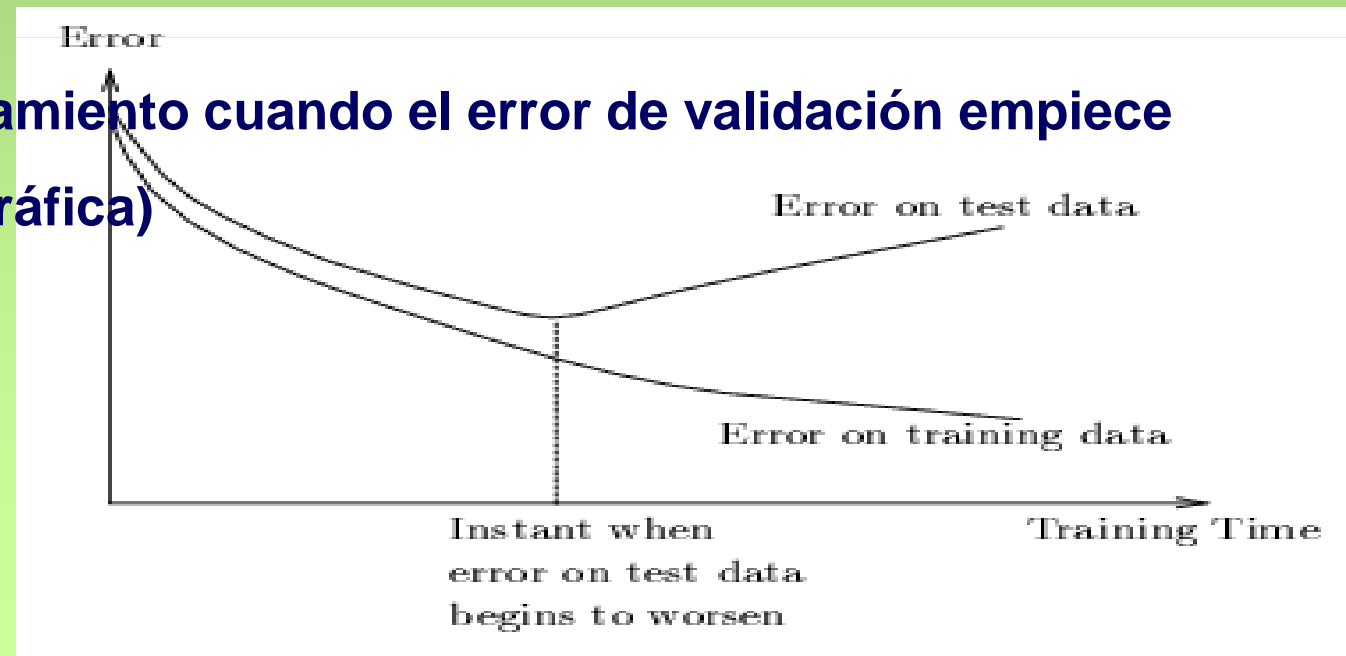


## ANALISIS DE LA GENERALIZACIÓN



### ■ Validación-cruzada en generalización

- Tomar aproximadamente un ( $\sim 10\%$ ) de las muestras de entrenamiento como datos de validación (no se utilizan para el cambio de pesos ni para generalizar)
- Chequear periódicamente los errores sobre este conjunto de validación
- Detener el entrenamiento cuando el error de validación empiece a aumentar (ver gráfica)





## Parálisis de la red con funciones de activación sigmoides



### ■ Regiones de saturación

Sea  $S(x) = 1 / (1 + e^{-x})$ ,

su derivada  $S'(x) = S(x)(1 - S(x)) \rightarrow 0$  cuando  $x \rightarrow \pm\infty$ .

Cuando  $x$  cae en una región de saturación,  $S(x)$  cambia fuertemente su valor independientemente de como de rápido aumente la magnitud de  $x$

- Una entrada a un nodo puede caer en una región de saturación cuando alguno de sus pesos de entrada se hagan muy grandes durante el aprendizaje. En consecuencia, los pesos dejan de cambiar sin importar los esfuerzos que se hagan.

$$\frac{\partial E}{\partial w_{k,j}^{(2,1)}} = -2(d_k - o_k)S'(net_k^{(2)})x_j^{(1)}$$

- Posibles remedios:

- Utilizar funciones de activación no saturadas
- Normalizar periódicamente todos los pesos

$$w_{k,j}^* = w_{k,j} / \|w_{.k}\|_2$$



## PARÁMETROS DEL ALGORITMO BP



- ❑ El aprendizaje (precisión, velocidad de convergencia, capacidad de generalización) **depende mucho de un conjunto de parámetros de aprendizaje**, entre otros son:
  - Pesos iniciales para el algoritmo BP, coeficiente de aprendizaje, n° de capas ocultas, n° de nodos en capa oculta, tipo de funciones de transferencia, etc...
  - La mayoría de ellos se determinan empíricamente (mediante experimentación).
  - Una de las técnicas más utilizada es la validación cruzada sobre el conjunto de entrenamiento utilizando un 5-fold y una estructura de tipo malla donde se van probando diferentes valores de cada uno de los parámetros del algoritmo.



## PARÁMETROS DEL ALGORITMO BP



- ❑ Una buena red entrenada mediante BP necesita algo más que el código del algoritmo. Es necesario determinar los parámetros del algoritmo para obtener un buen rendimiento en generalización.
- ❑ Aunque las deficiencias del algoritmo BP no pueden ser resueltas completamente, algunas de ellas se pueden resolver en la práctica.

### Pesos iniciales y sesgos

- De forma aleatoria,  $[-0.05, 0.05]$ ,  $[-0.1, 0.1]$ ,  $[-1, 1]$
- Normalizar los pesos de capa de entrada a capa oculta (Nguyen-Widrow)  $w^{(1,0)}$





## PARÁMETROS DEL ALGORITMO BP



### □ Pesos iniciales y sesgos

- Asignación aleatoria de los pesos iniciales para todos los pesos de capa oculta a capa de salida
- Para cada nodo de la capa oculta  $j$ , normalizar sus pesos de entrada mediante

$$w_{j,i}^{(1,0)*} = \beta \cdot w_{j,i}^{(1,0)} / \left\| w_j^{(1,0)} \right\|_2 \quad \text{donde } \beta = 0.7 \sqrt[n]{m}$$

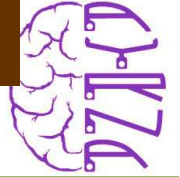
$m =$  n° de nodos ocultos,  $n =$  n° de nodos de entrada

$$\left\| w_j^{(1,0)*} \right\|_2 = \beta \text{ después de la normalización}$$

- Evitar sesgos en la inicialización de los pesos.



## DISEÑO DE EXPERIMENTOS Y TOPOLOGÍA DE LA RED



**Nº de muestras de entrenamiento:**

La calidad y cantidad de patrones o muestras de entrenamiento determina la calidad de los resultados de entrenamiento

Las muestras deben de representar a toda la población del espacio del problema

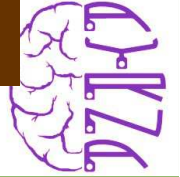
**Muestreo aleatorio**

**Muestreo proporcional (con conocimiento a priori del espacio del problema)**

**Nº de muestras de entrenamiento necesarias: No existe una teoría al respecto.**



## DISEÑO DE EXPERIMENTOS Y TOPOLOGÍA DE LA RED



**Regla empírica de Baum and Haussler (1989):**  $P = W/e$ , donde

**W:** n° total de pesos a ser entrenados (depende de la estructura de la red). **P:** n° de muestras de entrenamiento

**e:** porcentaje de error deseable de clasificación en entrenamiento.

Si la red puede ser entrenada para clasificar correctamente  $(1 - e/2)P$  de las **P** muestras de entrenamiento, entonces la precisión en clasificación de esta red es  $1 - e$  para patrones de entrada procedentes del mismo espacio muestral.

**Ejemplo:**  $W = 27$ ,  $e = 0.05$ ,  $P = 540$ . Si podemos entrenar la red para clasificar correctamente  $(1 - 0.05/2) \times 540 = 526$  de las muestras, la red trabajará correctamente el 95% de las veces con otros patrones de entrada.



# TOPOLOGÍA DE LA RED



- **Cuántas capas ocultas y cuántos nodos en capa oculta:**
  - Teóricamente con una sola capa oculta (posiblemente con muchos nodos en dicha capa) es suficiente para representar cualquier función  $L_2$
  - No existen resultados teóricos acerca del número de nodos en capa oculta, siendo este parámetro determinante en el aprendizaje de la red
  - **Regla empírica:**
    - $n = n^0$  de nodos de entrada;  $m = n^0$  de nodos en capa oculta
    - Para datos binarios:  $m = 2n$
    - Para datos reales:  $m \gg 2n$
  - En algunas aplicaciones, múltiples capas ocultas con menos nodos en cada capa pueden ser entrenadas más rápidamente con precisiones similares.



# **INTRODUCCIÓN A LOS MODELOS COMPUTACIONALES:**

## **CUARTO CURSO DEL GRADO**

### **DE ING. INFORMÁTICA EN COMPUTACION**

# **Algoritmo de retropropagación del error**

**GRACIAS POR SU ATENCIÓN**

**César Hervás-Martínez**  
**Grupo de Investigación AYRNA**

**Departamento de Informática y Análisis**  
**Númérico**  
**Universidad de Córdoba**  
**Campus de Rabanales. Edificio Einstein.**  
**Email: [chervas@uco.es](mailto:chervas@uco.es)**

**2018-2019**