Sistemas Inteligentes I

Tema 8. Redes Neuronales

José A. Montenegro Montes
monte@lcc.uma.es

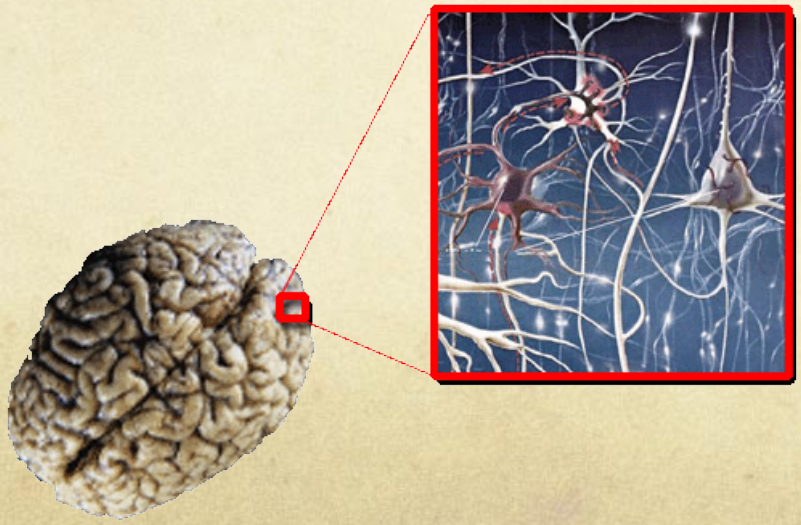
Resumen

- Introducción
- Neurona Artificial
- Perceptrón simple
- Perceptrón multicapa
- Ejemplos Aplicación
- Conclusiones
- Bibliografía

Introducción

Inspiración biológica: el cerebro

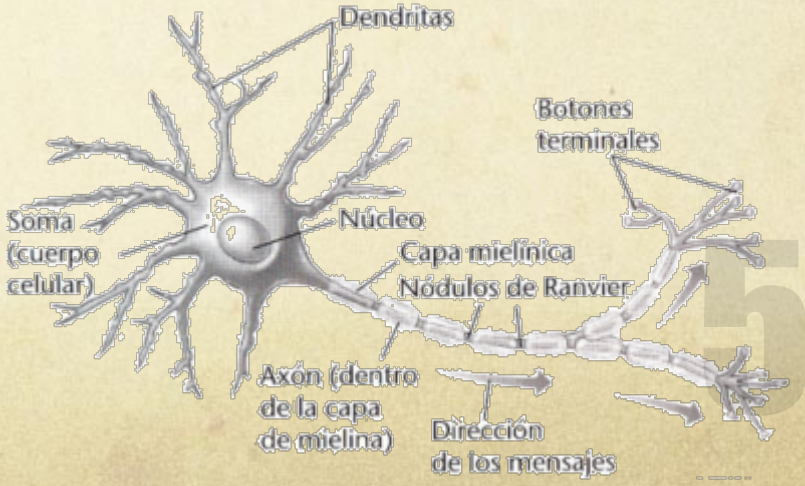
- 10^{11} neuronas
- +20 tipos neuronas
- 10^{14} sinapsis
- Tiempo de respuesta
  - 1-10 ms



- Órgano responsable de la cognición, las emociones, la memoria y el aprendizaje
- No se entiende el funcionamiento completo del cerebro
- Sólo algunas partes son esenciales para el procesamiento de la información

Comportamiento biológico

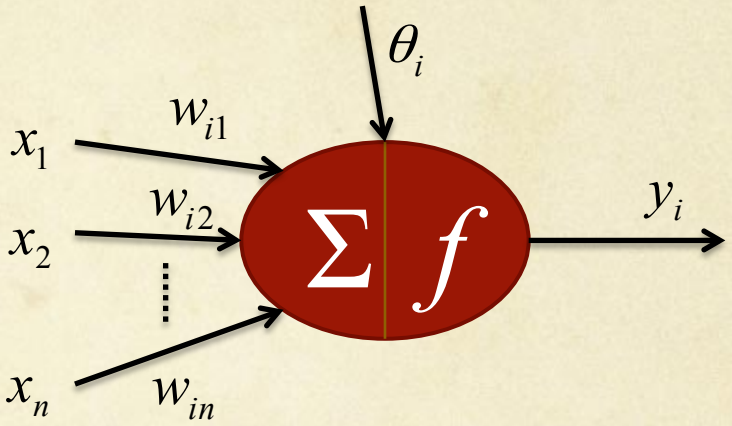
- Unidad de procesamiento elemental
- Las señales recibidas (a través de las dendritas) de otras neuronas mediante las sinapsis se combinan en el soma produciendo un potencial en el cuerpo celular
- Las señales emitidas (a través del axón) dependerán del potencial alcanzado



Neurona Artificial

Definición

- Pesos sinápticos
- Umbral
- Función de activación
- Estado



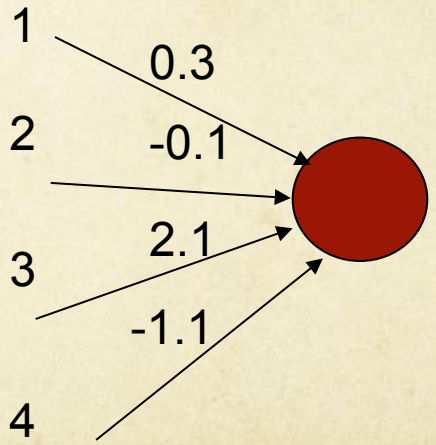
McCulloch & Pitts, 1943

$$y_i = f\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right)$$

Introducción Cálculo con Neuronas

- ¿Cómo generar las salidas?
- Una idea
  - *Sumar los pesos de entradas*

Entrada: (3, 1, 0, -2)
Procesamiento:
3(0.3) + 1(-0.1) + 0(2.1) + -1.1(-2)
= 0.9 + (-0.1) + 2.2
Salida: 3

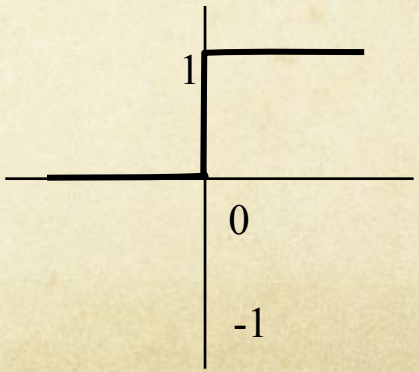


Función Activación

- Usualmente no utilizamos la suma de los pesos directamente
- Aplicaremos alguna función a la suma de los pesos antes de utilizar como salida
- Esta función es denominada Función de Activación

$$f(x) = \begin{cases} 1 \; si \; x \geq \theta \\ 0 \; si \; x < \theta \end{cases}$$

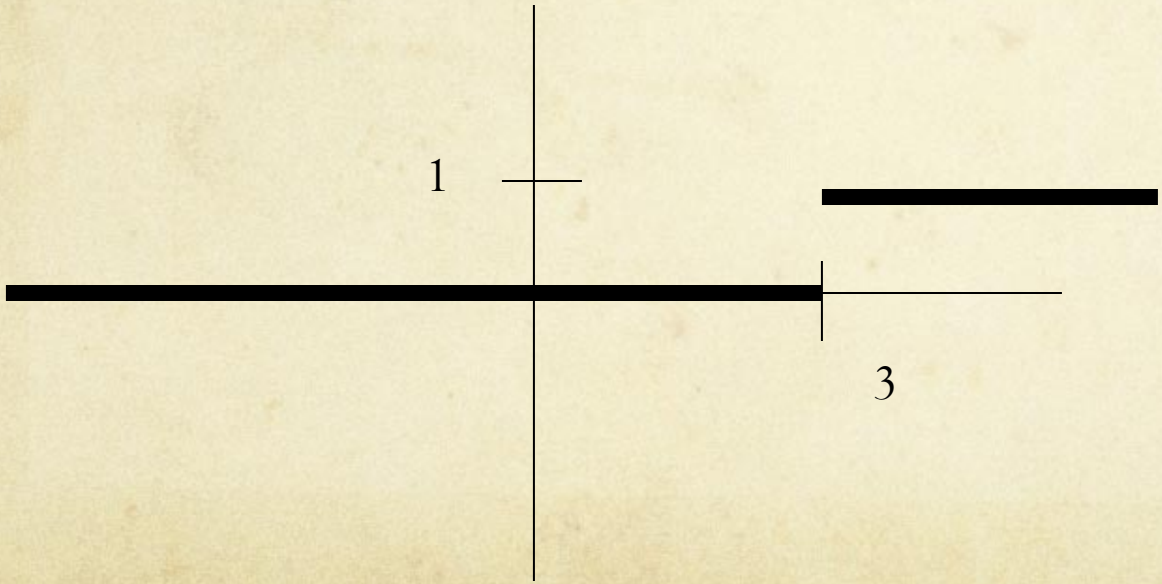
Umbral

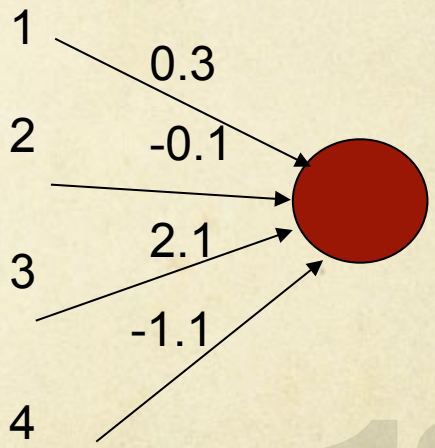


Función paso

Ejemplo Función Paso

- Sea $\Theta = 3$



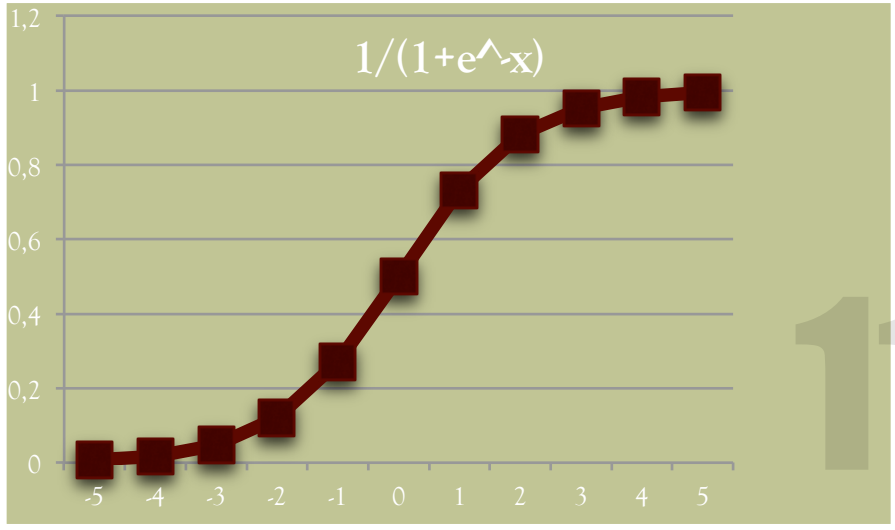


Salida después de utilizar la función activación: f(3)=1

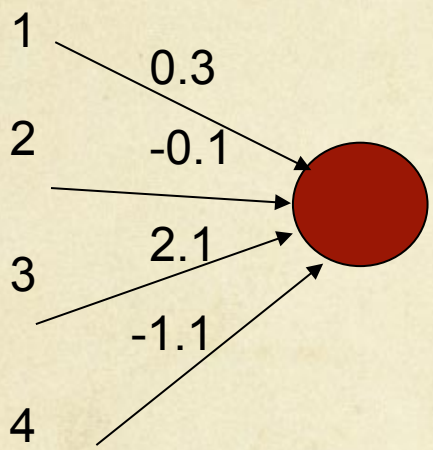
Otra Función Activación: Sigmoide

- Algunas redes neuronales necesitan que la función de activación sea continuamente derivable
- La función sigmoide (logística) es a menudo utilizada para aproximar la función paso

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



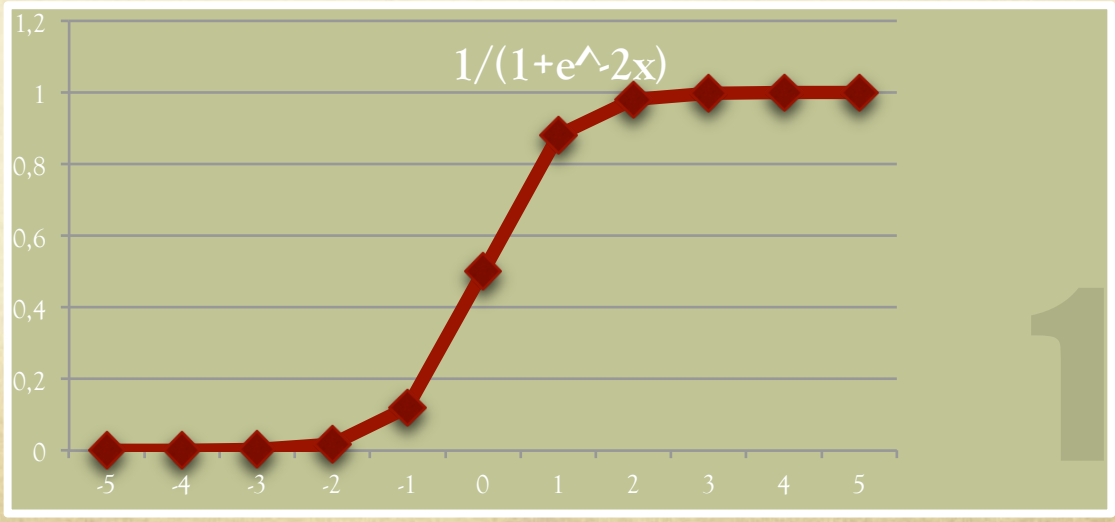
Ejemplo Sigmoide



Entrada: (10, 0, 0, 0) ¿Salida?

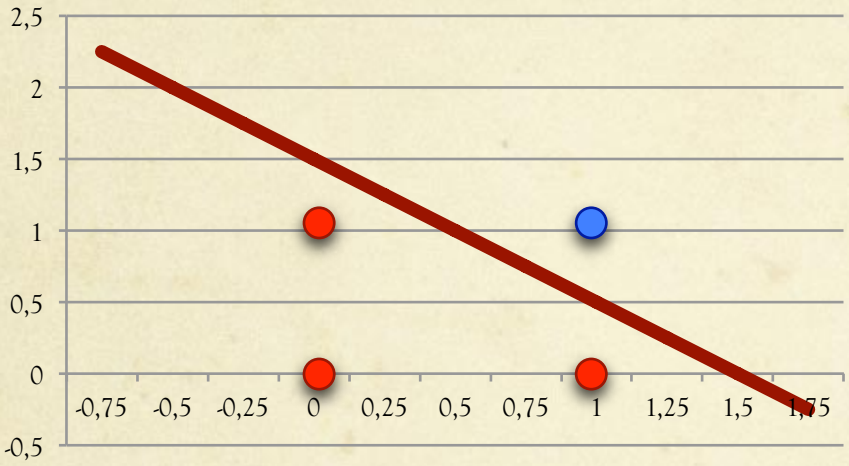
$$f(x) = \frac{1}{1+e^{-2x}}$$

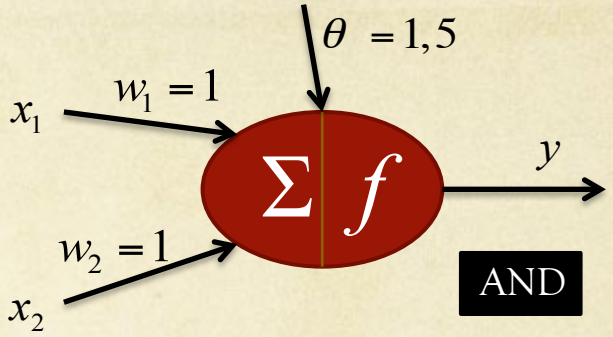
$$f(3) = \frac{1}{1+e^{-2x}} = .998$$



Ejemplos

- Función de activación: Paso





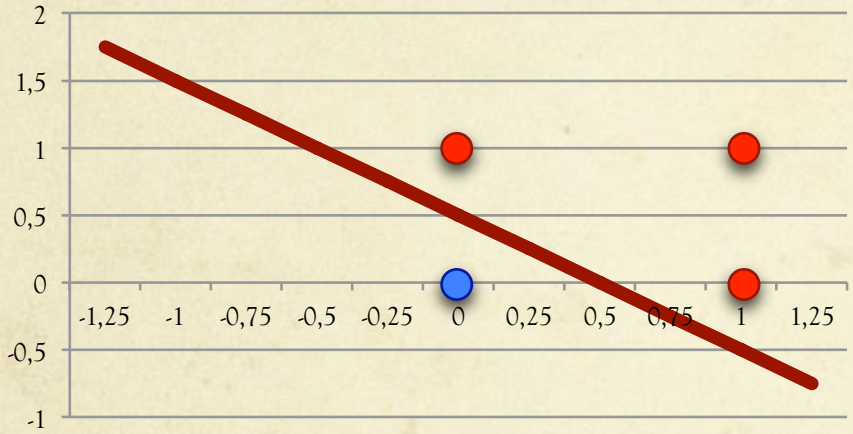
$$y = \text{sgn}\left(\sum_{j=1}^{n} w_j x_j - \theta\right)$$

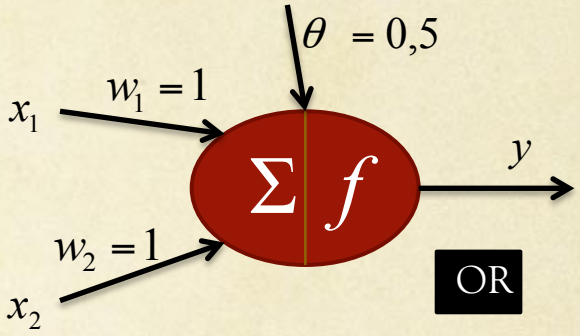
$$(\theta - W_1 x X_1) / W_2$$

x1	x2	y AND	y	e	W1	W2	x0	BIAS t
0	0	0	0	0	1	1	1	1,5
0	1	0	0	0				
1	0	0	0	0				
1	1	1	1	0				

Ejemplos

- Función de activación: Paso

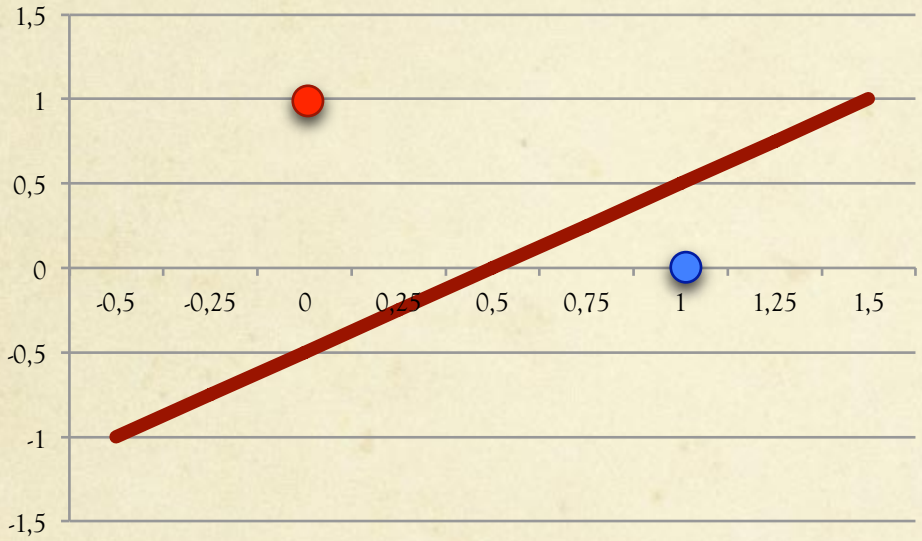


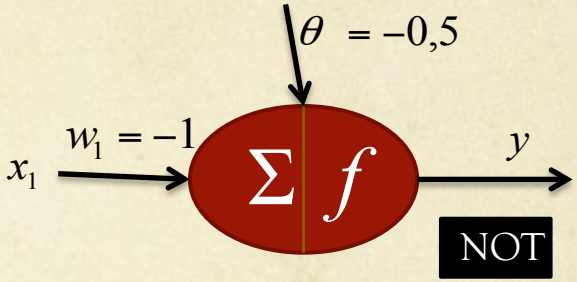


x1	x2	y OR	y	e	W1	W2	x0	BIAS t
0	0	0	0	0	1	1	1	0,5
0	1	1	1	0				
1	0	1	1	0				
1	1	1	1	0				

Ejemplos

- Función de activación: Paso





x1	y NOT	y	e	W1	x0	BIAS t
0	1	0	0	-1	1	-0,5
1	0	1	0			

Perceptrón Simple

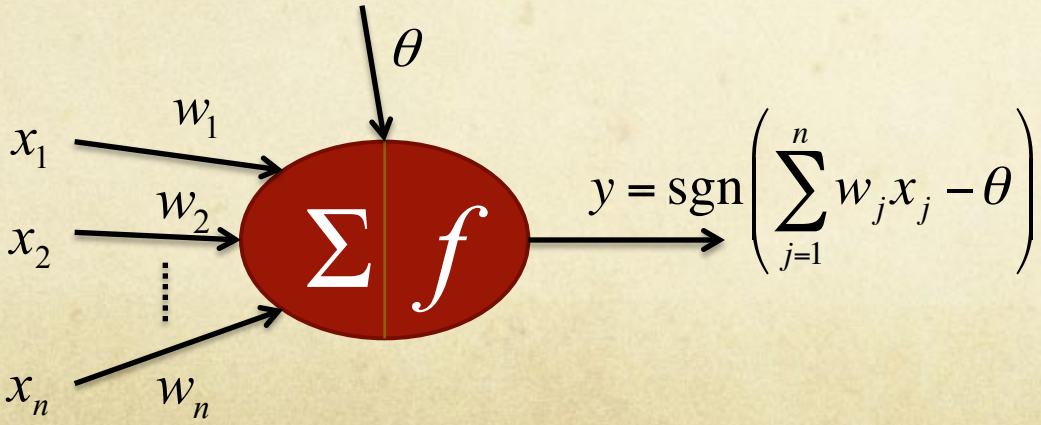
Perceptrón simple

- Frank Rosenblatt, 1958
- Aplicación al reconocimiento o clasificación
- Aprendizaje basado en la corrección del error

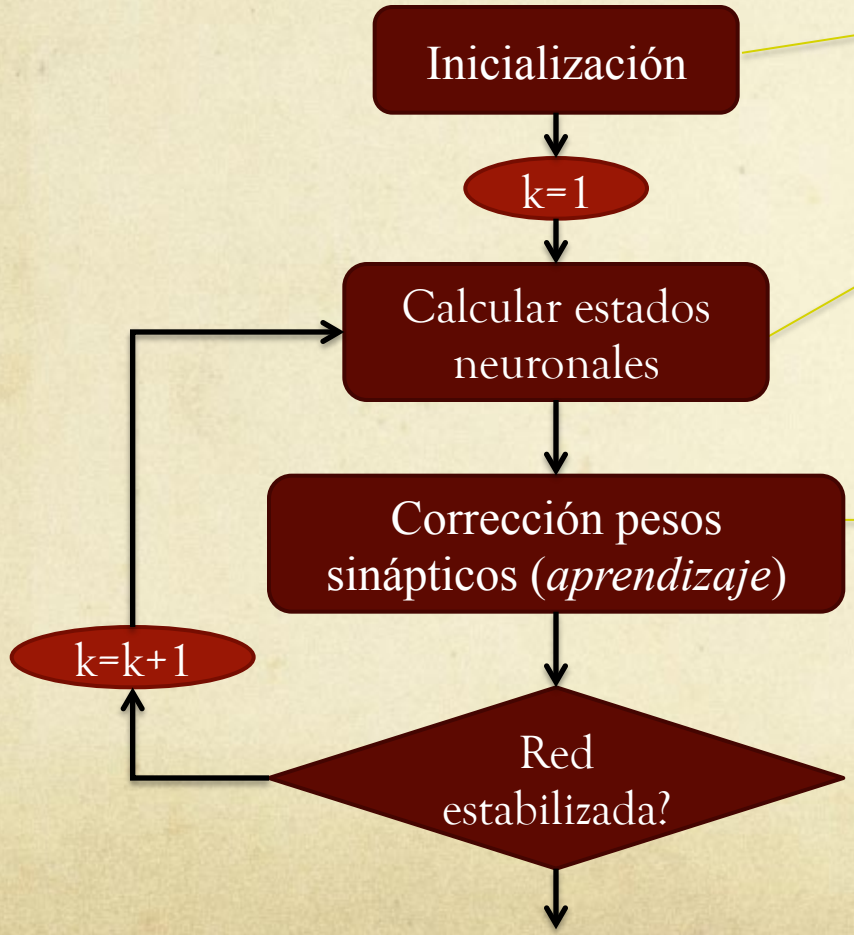
$$\Delta w_j(k) = \eta\left[z(k) - y(k)\right]x_j(k)$$

Error

Tasa de aprendizaje



Regla de aprendizaje



$$w_{ij} \in [-1,1]$$

$$y_i(k) = f\left(\sum_{j=1}^{n} w_{ij} x_j(k) - \theta_i\right)$$

$$\Delta w_{ij}(k) = \eta\left[z(k) - y_i(k)\right] x_j(k)$$

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}(k)$$

Regla de aprendizaje

OR

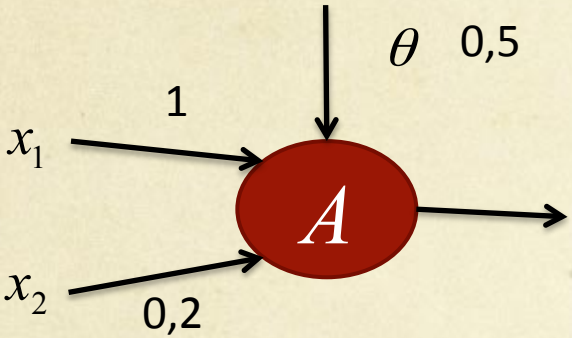
Época	x1	x2	y OR	y	e	w1	w2
1	0	0	0	0	0	0	-0,281227398
1	0	1	1	0	1	0	-0,055961068
1	1	0	1	0	1	0,22526633	-0,055961068
1	1	1	1	0	1	0,450532659	0,169305261
2	0	0	0	0	0	0,450532659	0,169305261
2	0	1	1	0	1	0,450532659	0,394571591
2	1	0	1	0	1	0,675798989	0,394571591
2	1	1	1	1	0	0,675798989	0,394571591
3	0	0	0	0	0	0,675798989	0,394571591
3	0	1	1	0	1	0,675798989	0,61983792
3	1	0	1	0	1	0,901065318	0,61983792
3	1	1	1	1	0	0,901065318	0,61983792
4	0	0	0	0	0	0,901065318	0,61983792
4	0	1	1	0	1	0,901065318	0,84510425
4	1	0	1	1	0	0,901065318	0,84510425
4	1	1	1	1	0	0,901065318	0,84510425
5	0	0	0	0	0	0,901065318	0,84510425
5	0	1	1	1	0	0,901065318	0,84510425
5	1	0	1	1	0	0,901065318	0,84510425
5	1	1	1	1	0	0,901065318	0,84510425

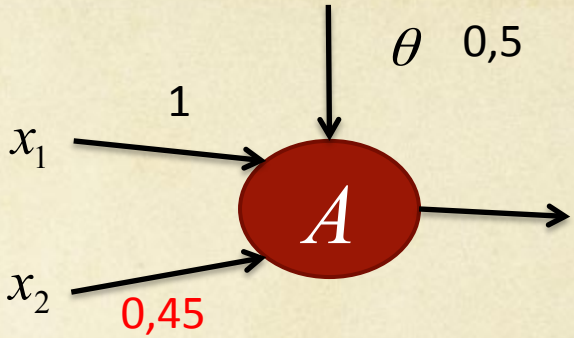
W1	W2	x0	BIAS t
0	-0,281227398	1	0,7

Tasa Aprendizaje

0,22526633

Regla de aprendizaje





Entrada Neurona A (0,0) = (0x1)+(0x0.2)-0.5=-0.5

Salida= sgn(-0.5)=0

Error = 0-0 =0. No se modifican los pesos

Entrada Neurona A (0,1) = (0x1)+(1x0.2)-0.5=-0.3

Salida= sgn(-0.3)=0

Error = 1-0=1. Se modifican los pesos

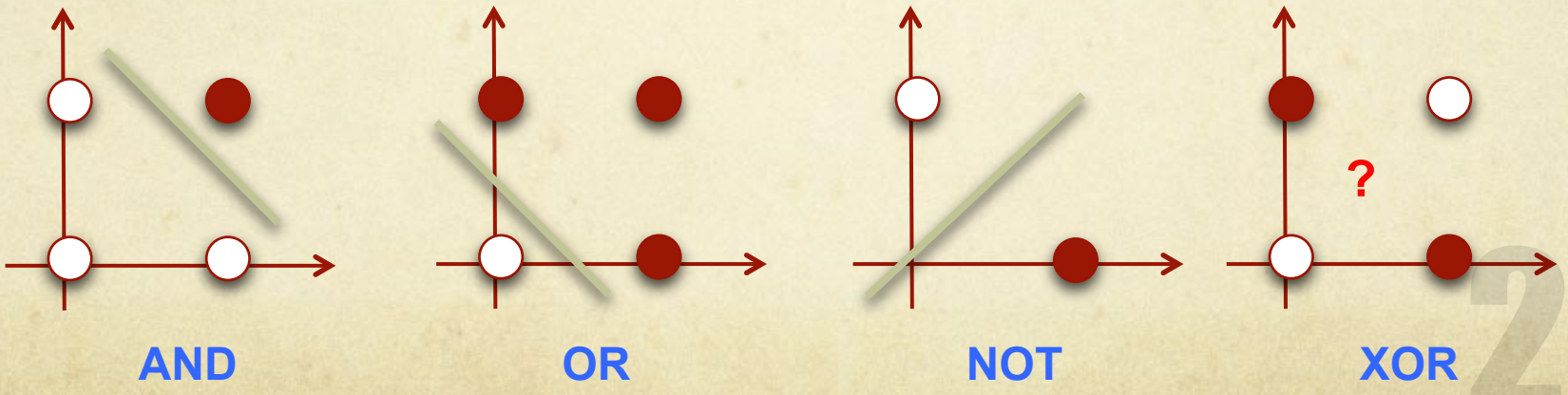
W1+=w1+(tasa*e*x1)=1+(0,25*1*0)=1

W2+=w2+(tasa*e*x2)=0,2+(0,25*1*1)=0,45

x1	x2	OR
0	0	0
0	1	1
1	0	1
1	1	1

Inconvenientes

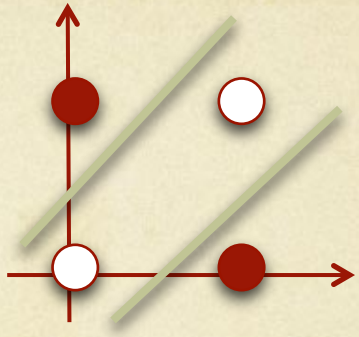
- Sólo clasifica conjuntos linealmente separables
- No es posible Implementación XOR
- Solución
  - Añadir capas

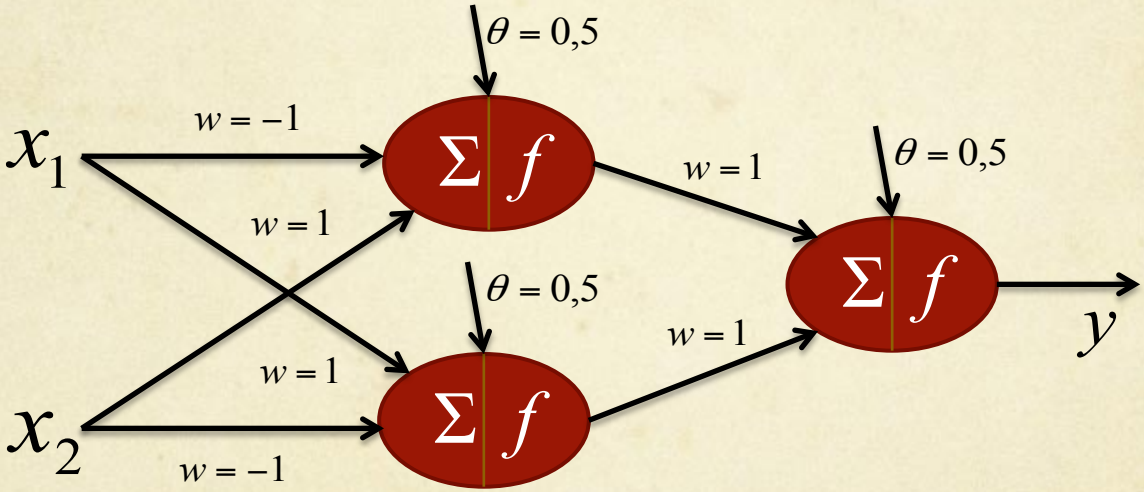


Perceptrón Multicapa

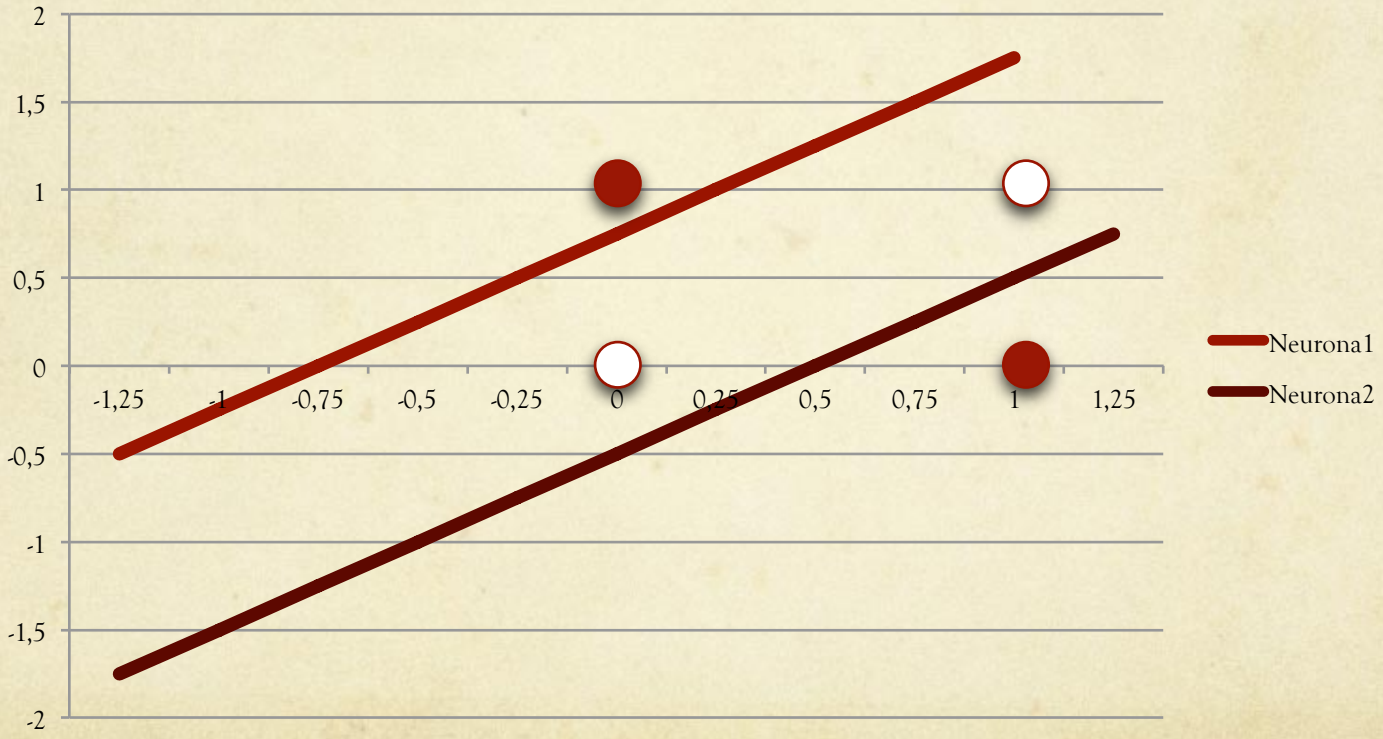
Perceptrón Multicapa

- Implementación XOR
  - Minsky y Papert, 1969

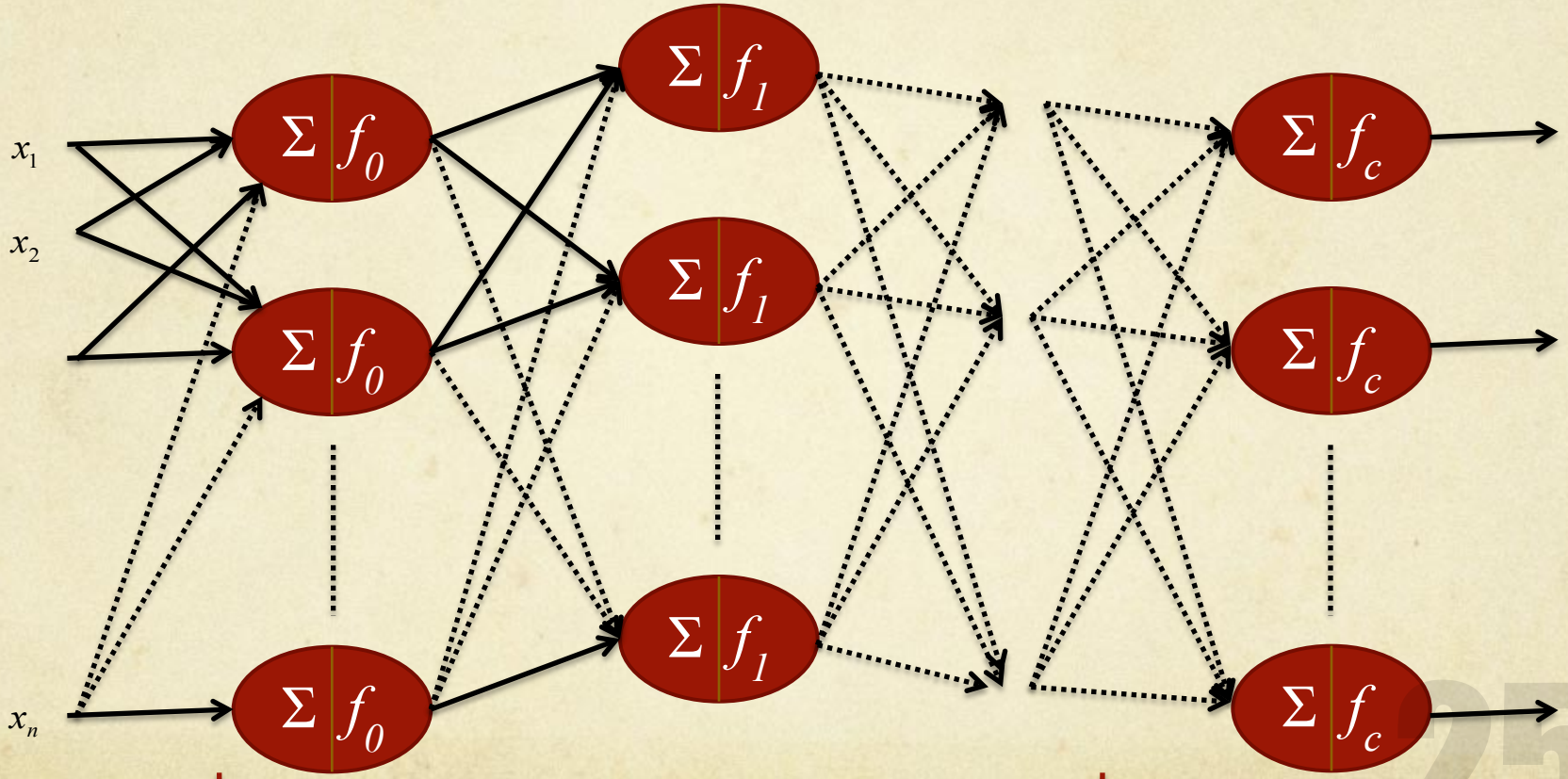




Perceptrón Multicapa



Topología


x_1
x_2
x_n
Σ f_0
Σ f_1
Σ f_c
Entradas
Capas ocultas
Salida

Regla de retro-propagación

- Utilizamos la función sigmoidal ya que es fácil derivar:

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

- Inicialmente los valores son calculados hacia delante.
- Posteriormente los errores son calculados desde capa salida hasta el inicio.

Regla de retro-propagación

- El error es calculado como antes: $e_k=d_k-y_k$
- Gradiente para las salidas:

$$\delta_k = \frac{dy_k}{dx_k} * e_k = y_k * (1 - y_k) * e_k$$

- Y para cada nodo j en las capas ocultas:

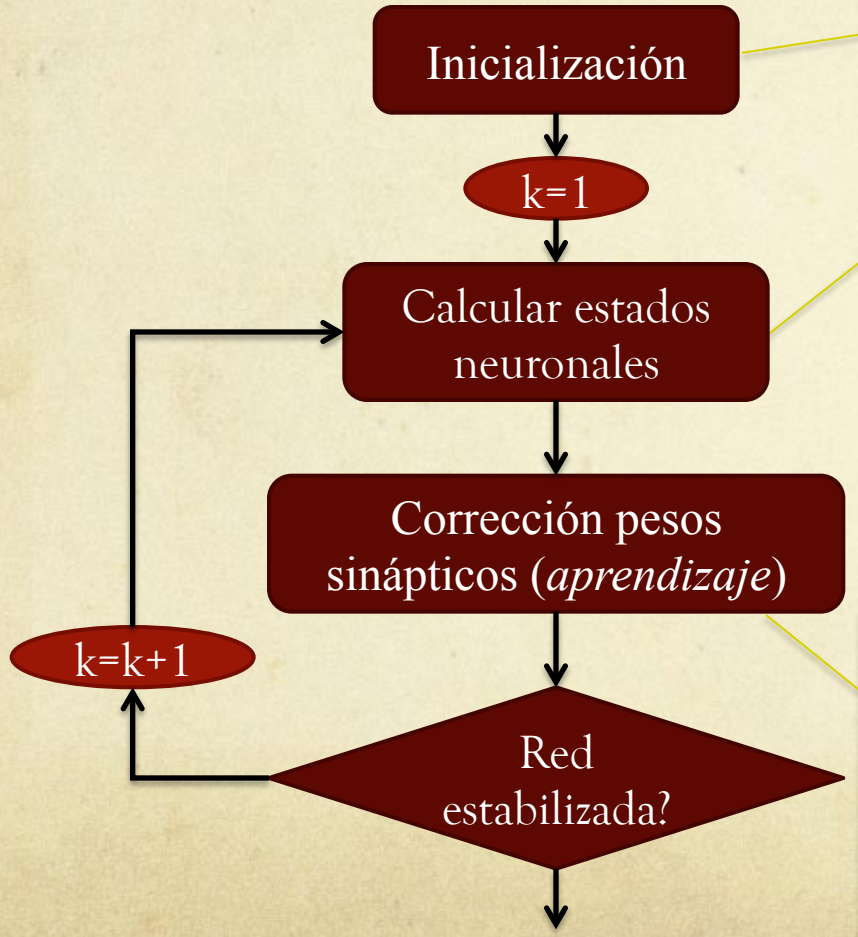
$$\delta_j = y_j * (1 - y_j) * \sum_{k=1}^{n} w_{ik}\delta_k$$

- Cada peso en la red es actualizado acorde la siguiente fórmula:

$$w_{ij} = w_{ij} + \alpha * x_i * \delta_j$$

$$w_{jk} = w_{jk} + \alpha * y_j * \delta_k$$

Regla de retro-propagación



$$w_{ij} \in [-1,1]$$

$$y_i(k) = f_c\left(\sum_{j=1}^{n} w_{c,ij}\, x_{c,j}(k) - \theta_{c,i}\right)$$

$$x_{c,i}(k) = f_{c-1}\left(\sum_{j=1}^{n} w_{c-1,ij}\, x_{c-1,j}(k) - \theta_{c-1,i}\right)$$

$$\Delta w_{c,ij}(k) = \eta\left[z(k) - y_i(k)\right] f_c'\left(h_{c,i}(k)\right) x_{c,j}(k)$$

$$\delta_{c-1,i}(k) = f_c'\left(h_{c,i}(k)\right) \sum_{j=1}^{n} w_{c,ij}(k)\,\delta_{c,j}(k)$$

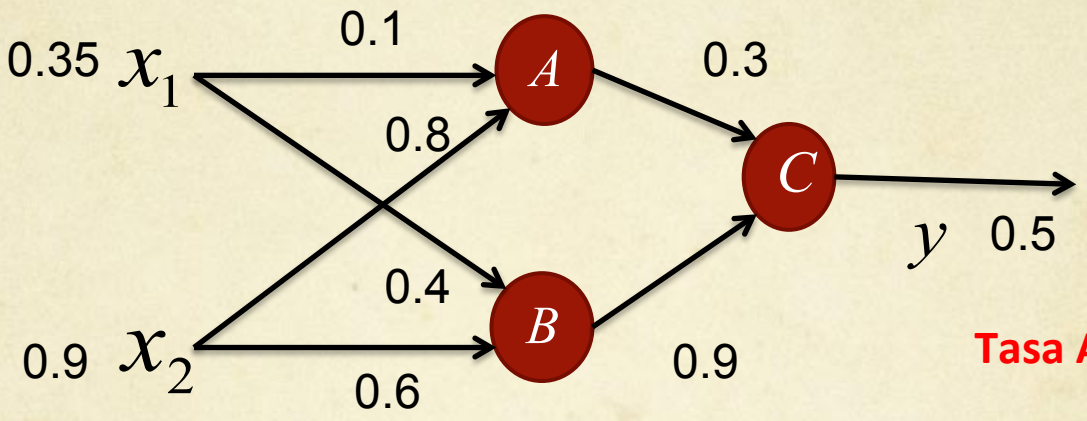
Regla de retro-propagación

function BACK-PROP-LEARNING($examples, network$) **returns** a neural network
  inputs: $examples$, a set of examples, each with input vector **x** and output vector **y**
    $network$, a multilayer network with L layers, weights $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in $network$ **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example (**x**, **y**) **in** $examples$ **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\ a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\ \Delta[j]$
      /* *Update every weight in network using deltas* */
      for each weight $w_{i,j}$ in $network$ **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return $network$

Figure 18.23 The back-propagation algorithm for learning in multilayer networks.

Ejemplo Regla de retro-propagación



Tasa Aprendizaje=1

Entrada Neurona A = (0.35x0.1)+(0.9x0.8)=0.755.

Salida= $1/(1+e^{-0.755})$=0.68.

Entrada Neurona B= (0.9x0.6)+(0.35x0.4) = 0.68.

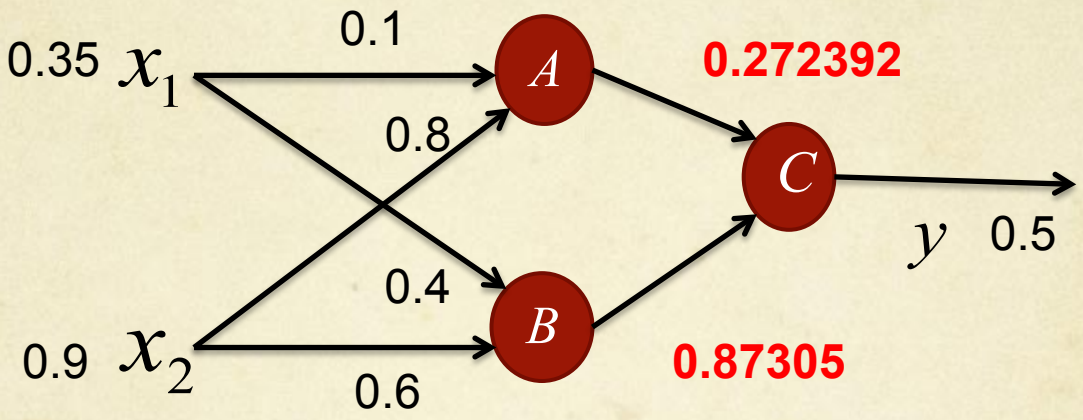
Salida= $1/(1+e^{-0.68})$=0.6637.

Entrada Neurona C = (0.3x0.68)+(0.9x0.6637) = 0.80133.

Salida= $1/(1+e^{-0.80133})$=0.69.

Error salida 0.5 – 0.69= -0.19

Ejemplo Regla de retro-propagación



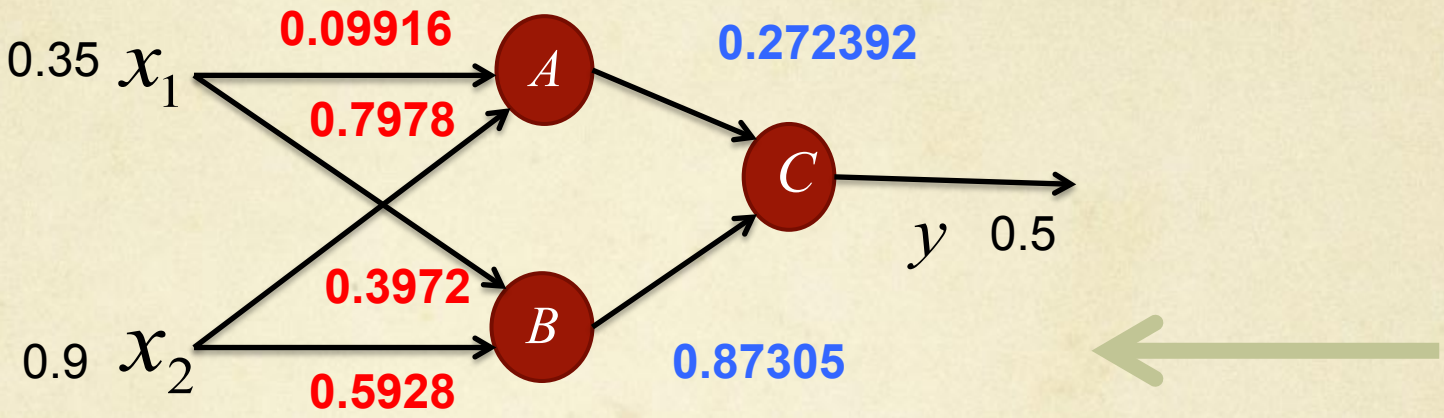
Gradiente Salida δ = out (1 - out) (t - out)= 0.69 (1-0.69) (0.5- 0.69) = -0.0406.

Nuevos pesos para la capa salida

w1 = w1+(δ x input) = 0.3 + (-0.0406 x 0.68) = 0.272392.

w2 = w2+(δ x input) = 0.9 + (-0.0406 x 0.6637) = 0.87305.

Ejemplo Regla de retro-propagación



Gradientes capas ocultas:

$\delta_1 = \delta$ x w_1 x out x (1-out) = -0.0406 x 0.272392 x **0.68 x 0.32** = -0.0024

$\delta_2 = \delta$ x w_2 x out x (1-out) = -0.0406 x 0.87305 x **0.6637 x** 0.3363 = -0.0079

Nuevos pesos:

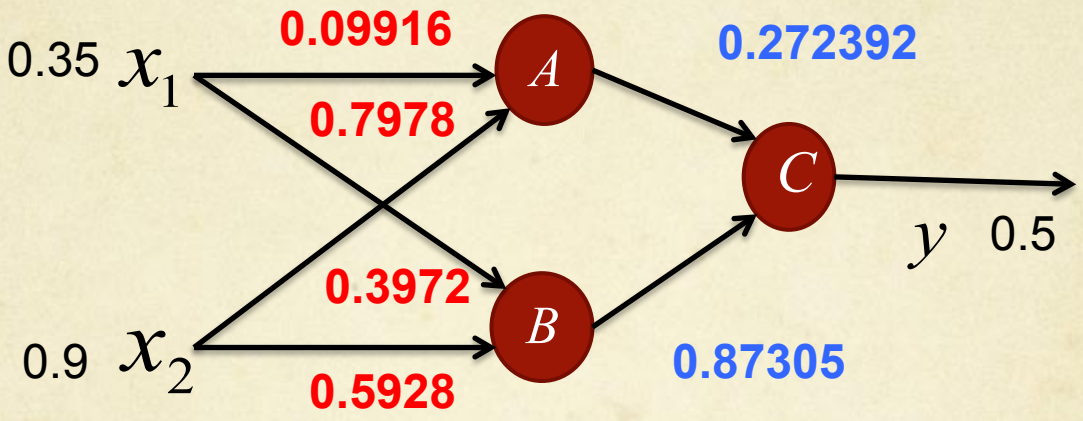
w_3 = 0.1 + (-0.0024 x 0.35) = 0.09916.

w_4 = 0.8 + (-0.0024 x 0.9) = 0.7978.

w_5 = 0.4 + (-0.0079 x 0.35) = 0.3972.

w_6 = 0.6 + (-0.0079 x 0.9) = 0.5928.

Ejemplo Regla de retro-propagación



Entrada Neurona A = (0.35x**0.09916**)+(0.9x**0.7978**)=0,75275.

Salida= 1/(1+e$^{-0,75275}$)=0,67977.

Entrada Neurona B= (0.9x**0.5928**)+(0.35x**0.3972**) = 0,67260.

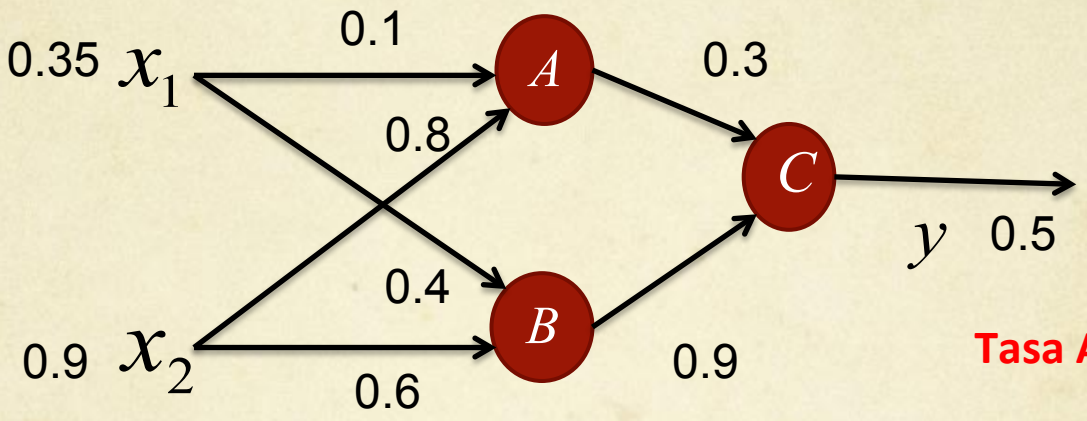
Salida= 1/(1+e$^{-0,67260}$)=0,66208.

Entrada Neurona C = (0.3x**0,67977**)+(0.9x**0,66208**) = 0,76312.

Salida= 1/(1+e$^{-0,76312}$)=0,682.

Error salida 0.5 – 0.68= -0.18 (Antes 0.69)

Ejemplo Regla de retro-propagación



Tasa Aprendizaje=1

Entrada Neurona A = (0.35x0.1)+(0.9x0.8)=0.755.

Salida= $1/(1+e^{-0.755})$=0.68.

Entrada Neurona B= (0.9x0.6)+(0.35x0.4) = 0.68.

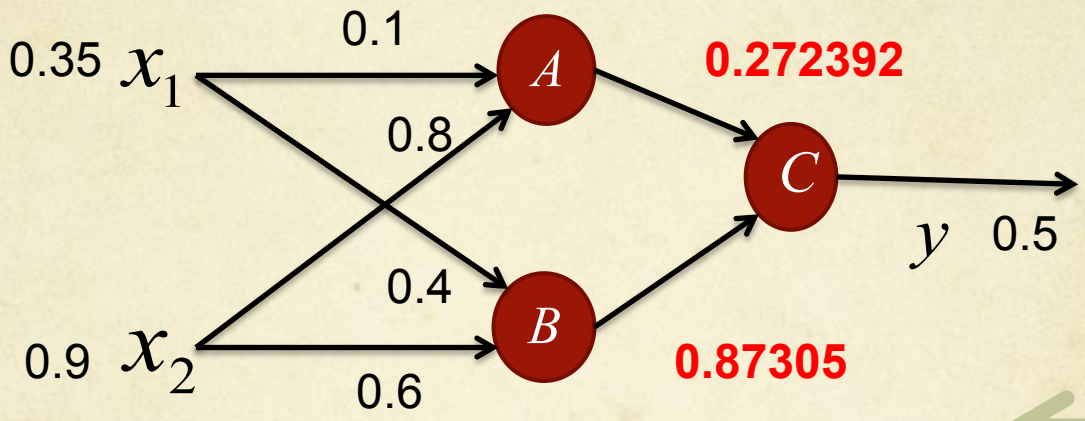
Salida= $1/(1+e^{-0.68})$=0.6637.

Entrada Neurona C = (0.3x0.68)+(0.9x0.6637) = 0.80133.

Salida= $1/(1+e^{-0.80133})$=0.69.

Error salida 0.5 – 0.69= -0.19

Ejemplo Regla de retro-propagación



Gradiente Salida δ = out (1 - out) (t - out)= 0.69 (1-0.69) (0.5- 0.69) = -0.0406.

$$\delta_k = \frac{dy_k}{dx_k} * e_k = y_k * (1 - y_k) * e_k$$

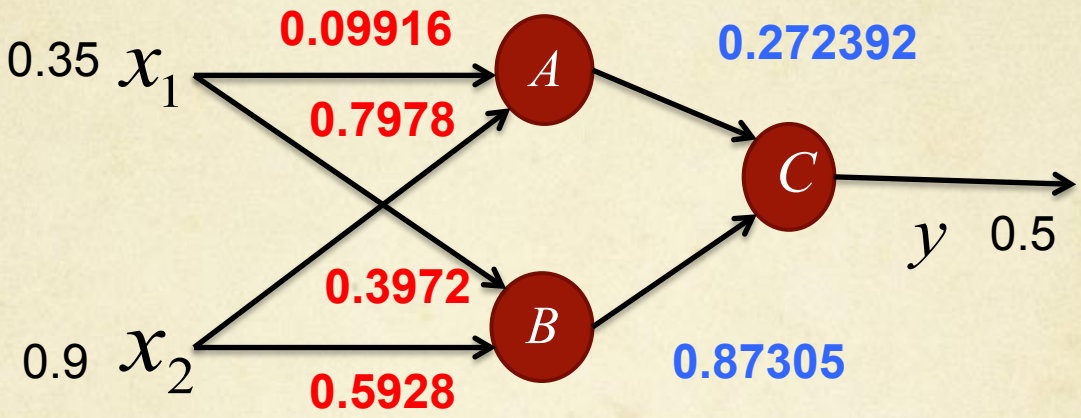
Nuevos pesos para la capa salida

w1 = w1+(δ x input) = 0.3 + (-0.0406 x 0.68) = 0.272392.

w2 = w2+(δ x input) = 0.9 + (-0.0406 x 0.6637) = 0.87305.

$$w_{jk} = w_{jk} + \alpha * y_j * \delta_k$$

Ejemplo Regla de retro-propagación



Gradientes capas ocultas:

δ_1 = out x (1-out) x δ x w1 = 0.68 x 0.32 x -0.0406 x 0.272392 = -0.0024

δ_2 = out x (1-out) x δ x w2 = 0.6637 x 0.3363 x -0.0406 x 0.87305 = -0.0079

Nuevos pesos:

w_3 = 0.1 + (0.35 x -0.0024) = 0.09916.

w_4 = 0.8 + (0.9 x -0.0024) = 0.7978.

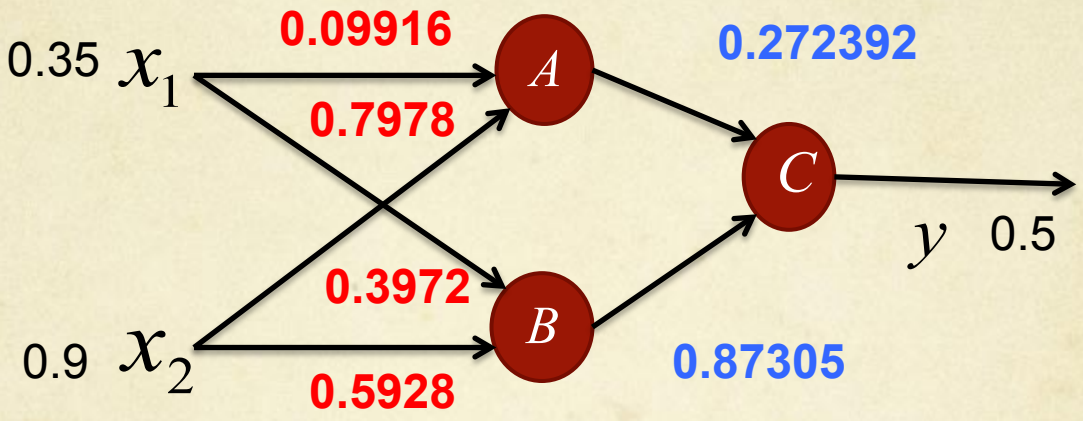
w_5 = 0.4 + (0.35 x -0.0079) = 0.3972.

w_6 = 0.6 + (0.9 x -0.0079) = 0.5928.

$$\delta_j = y_j * (1 - y_j) * \sum_{k=1}^{n} w_{ik}\delta_k$$

$$w_{ij} = w_{ij} + \alpha * x_i * \delta_j$$

Ejemplo Regla de retro-propagación



Entrada Neurona A = (0.35x**0.09916**)+(0.9x**0.7978**)=0,75275.

Salida= 1/(1+e$^{-0,75275}$)=0,67977.

Entrada Neurona B= (0.9x**0.5928**)+(0.35x**0.3972**) = 0,67260.

Salida= 1/(1+e$^{-0,67260}$)=0,66208.

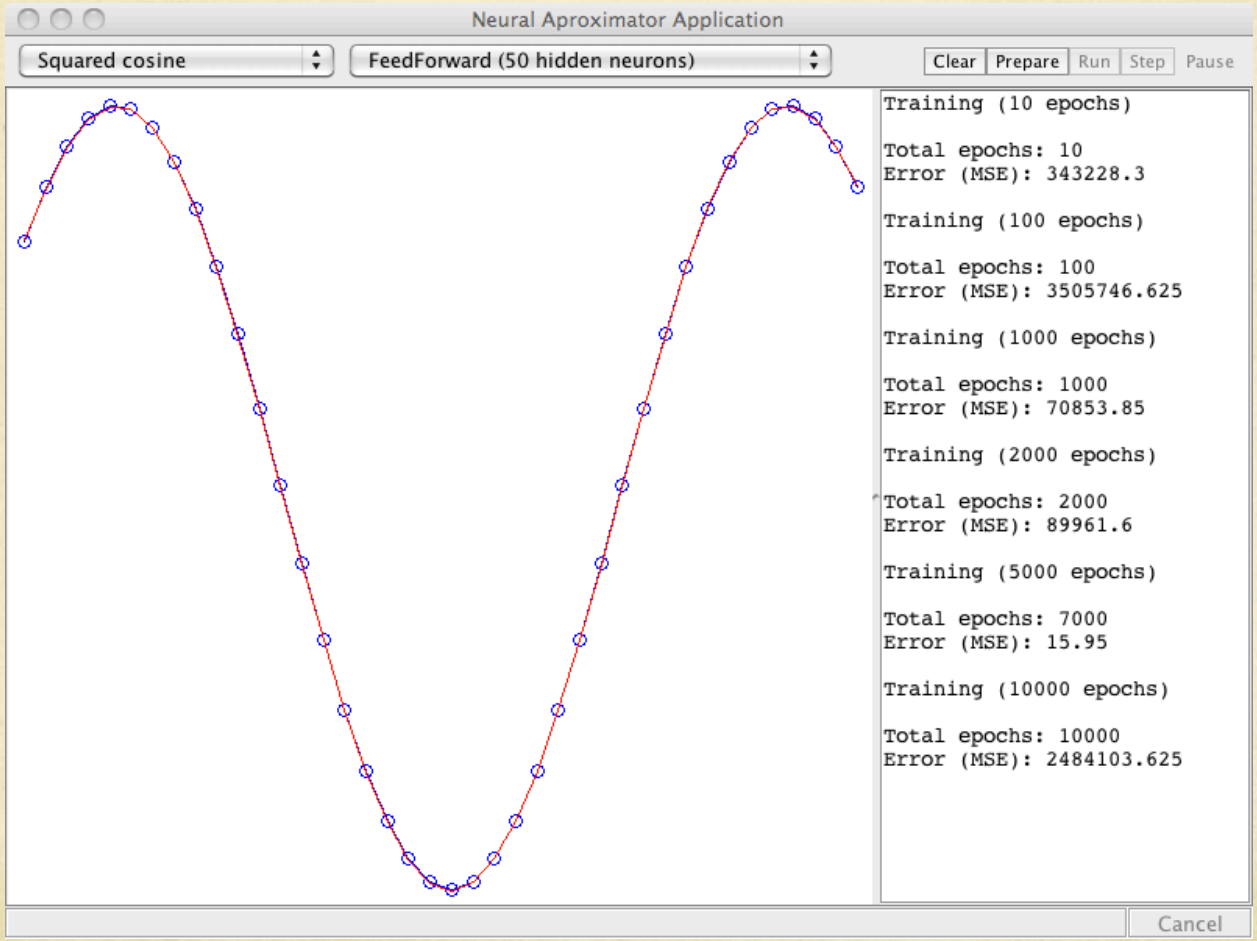
Entrada Neurona C = (0.3x**0,67977**)+(0.9x**0,66208**) = 0,76312.

Salida= 1/(1+e$^{-0,76312}$)=0,682.

Error salida 0.5 – 0.68= -0.18 (Antes 0.69)

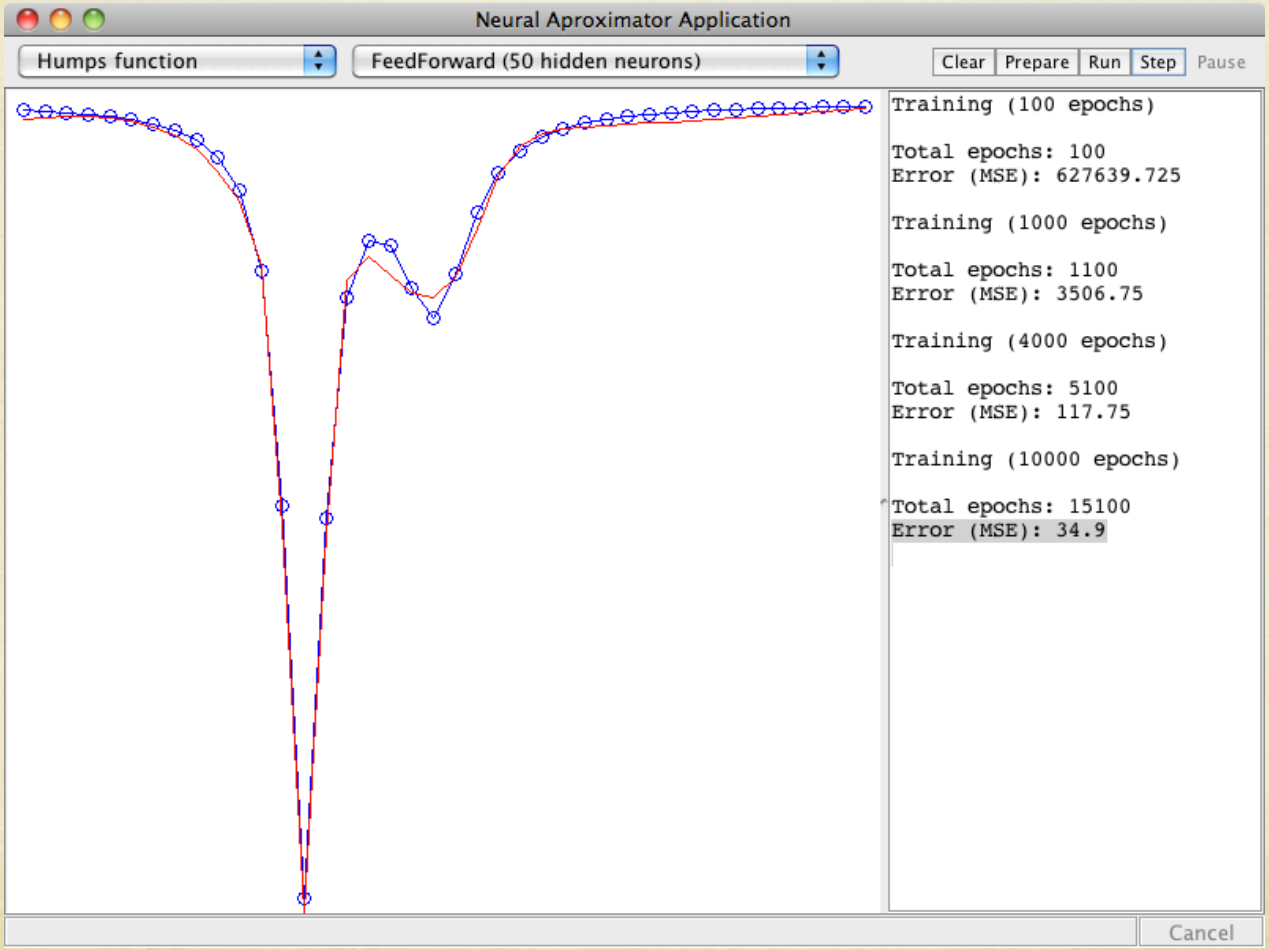
Ejemplos Aplicación

Ejemplo Aplicación



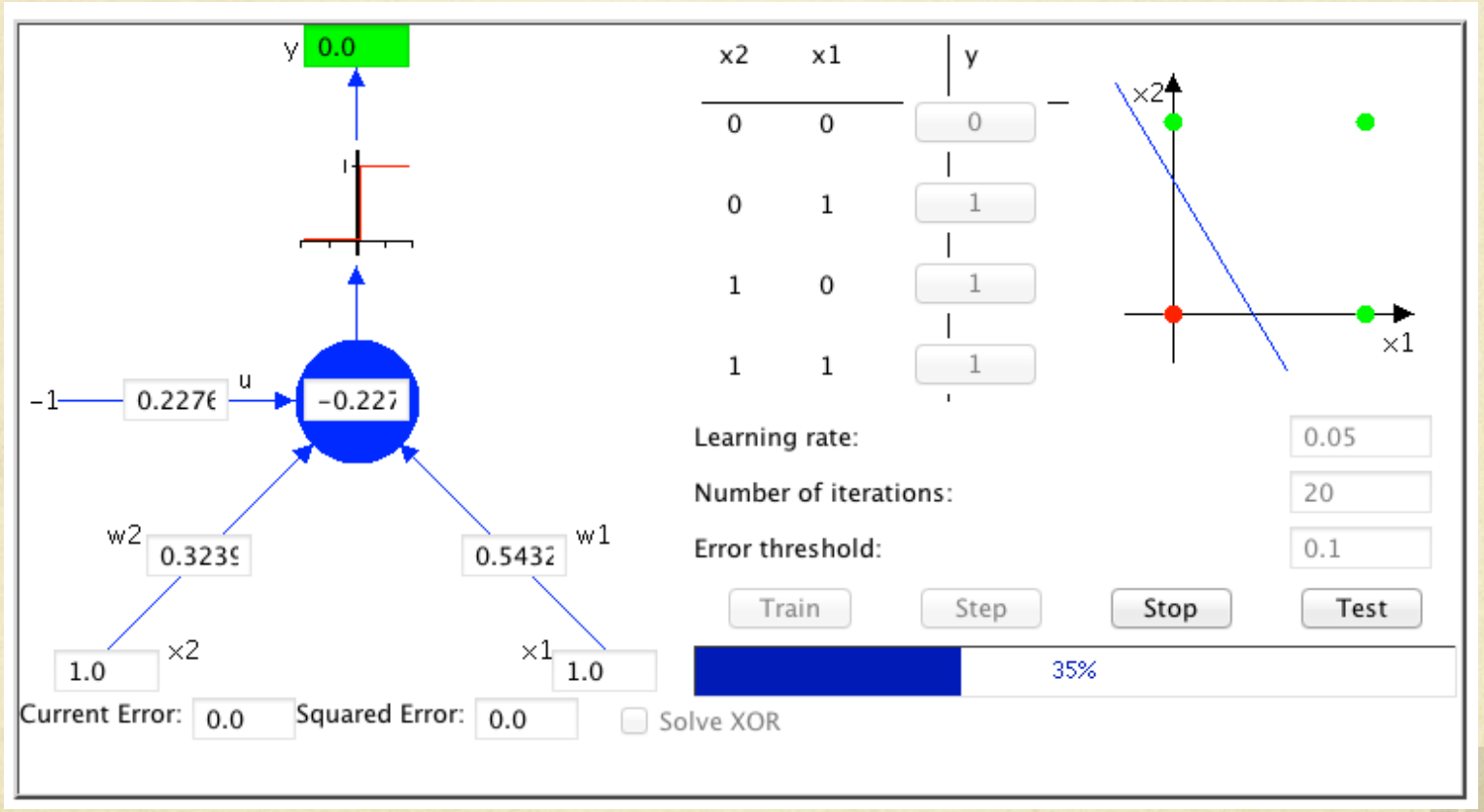

```
double y =  Math.cos(x)*Math.cos(x);
```

Ejemplo Aplicación

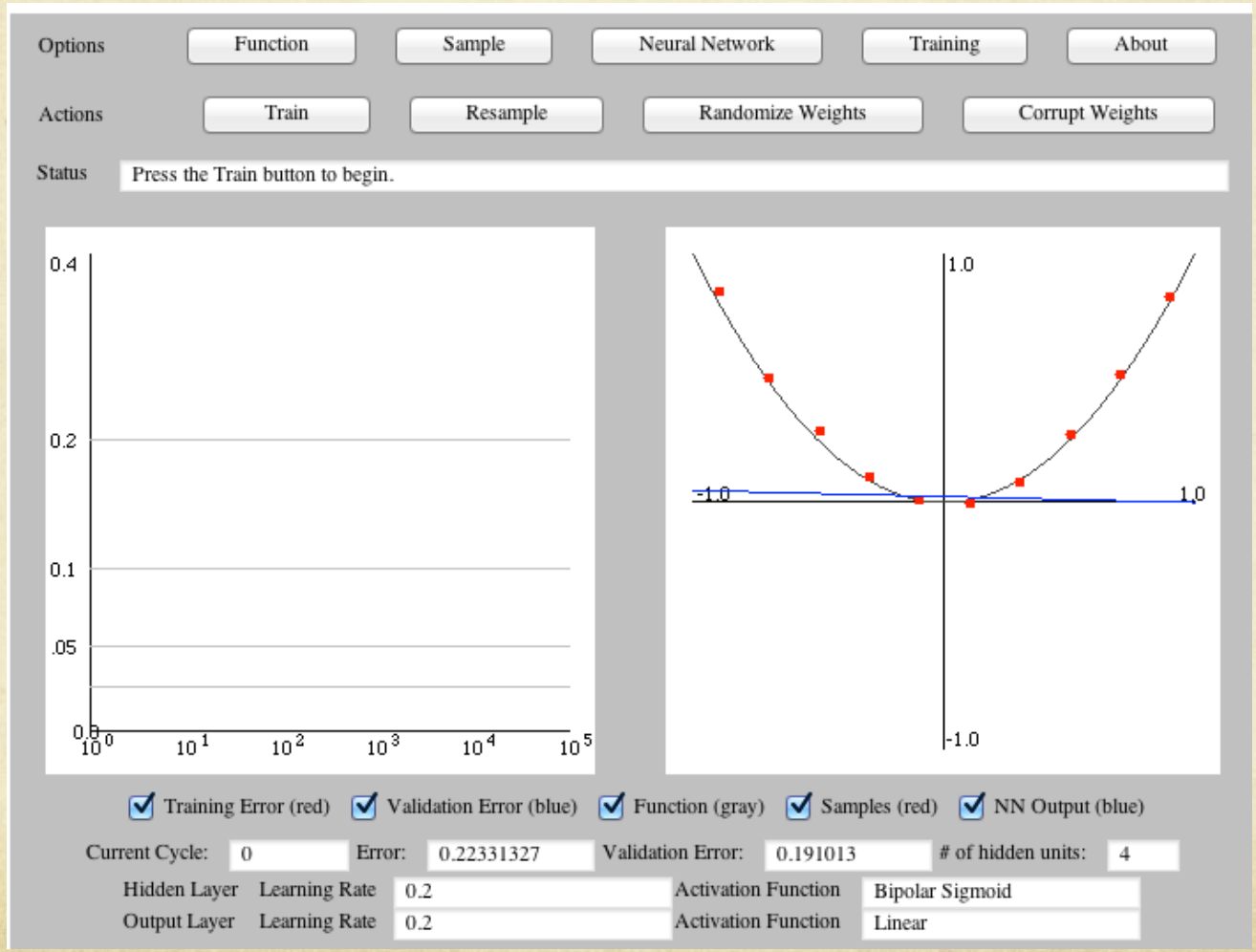


double y = 1.0/((x-.3)*(x-.3) + .01) + 1.0/((x-.9)*(x-.9) + .04) - 6.0;

Ejemplo Applet



Ejemplo Applet



Conclusiones

Conclusiones

- Las redes neuronales artificiales están inspiradas en el funcionamiento de las neuronas biológicas, pero no pretenden simularlas sino ser útiles para la resolución de problemas de ingeniería
- Pueden representar funciones no lineales complejas mediante una red de unidades sencillas
- El perceptrón multicapa puede resolver problemas insolubles para el perceptrón simple
- El algoritmo de retro-propagación busca valores de los pesos sinápticos que minimicen el error

Bibliografía

Bibliografía

- AIMA 3 Edición
- Artificial Neural Networks and Evolutionary Algorithms - An on-line book
- Artificial Intelligence Illuminated, *Ben Coppin*

Sistemas Inteligentes

José A. Montenegro Montes

monte@lcc.uma.es