



Institución Universitaria

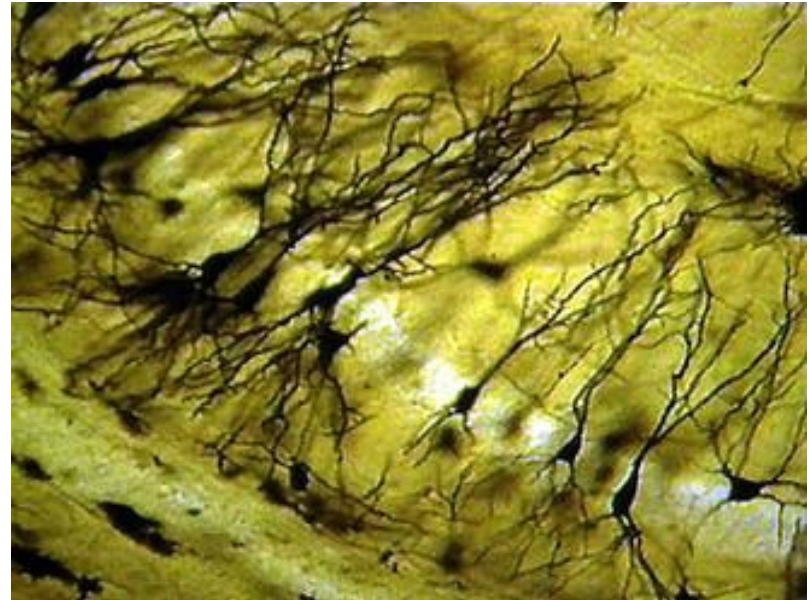
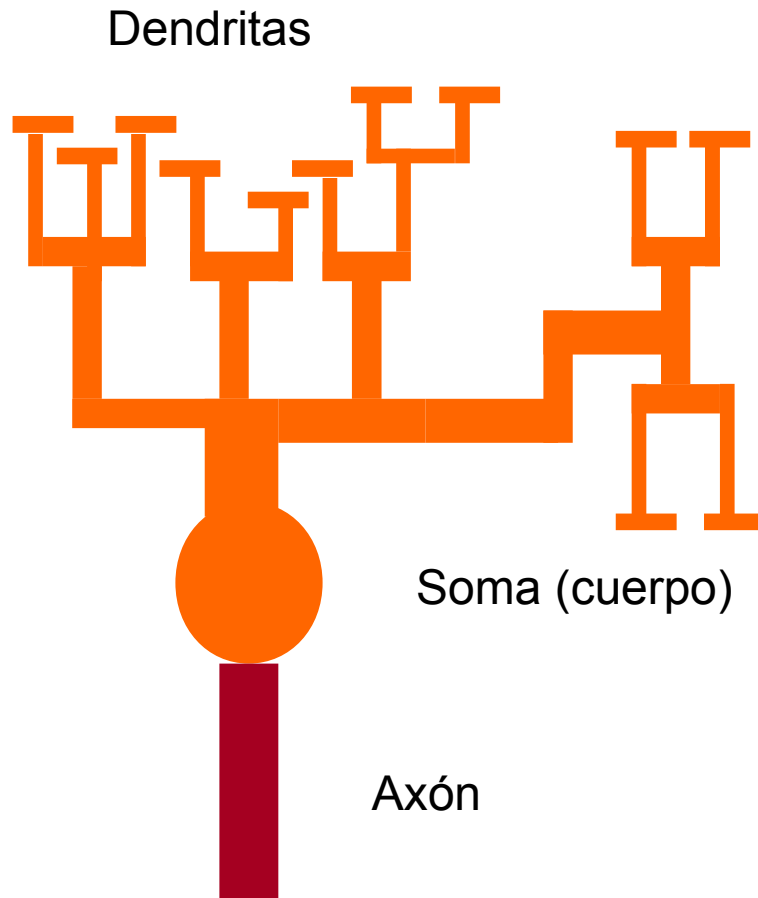
# Inteligencia Artificial

## 2.3. Perceptrón Simple.

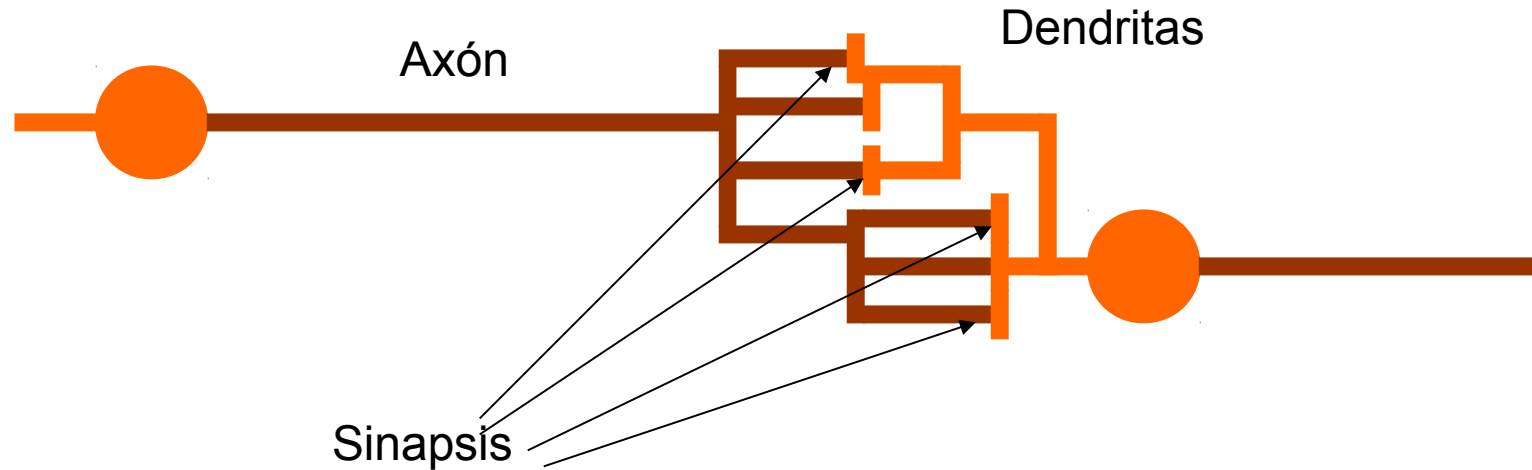
# Introducción: Inspiración biológica

- Los animales son capaces de adaptarse a los cambios de su entorno y utilizan su sistema nervioso para este propósito.
- Una simulación/emulación apropiada del sistema nervioso permitiría producir respuestas y comportamientos similares en sistemas artificiales.
- El sistema nervioso está compuesto de unidades relativamente pequeñas (neuronas), de forma tal que, copiar su comportamiento debería ser una aproximación a la solución del problema anterior.

# Introducción: Inspiración biológica



# Introducción: Inspiración biológica



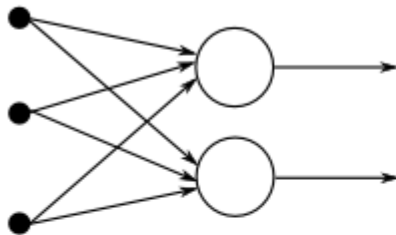
La transmisión de la información ocurre a través de las sinapsis.

# Introducción: Inspiración biológica

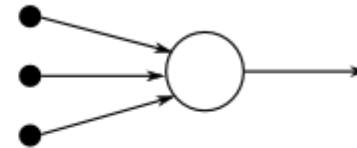
- Los picos que viajan a lo largo del axón activan la liberación de sustancias neurotransmisoras a la sinapsis.
- Los neurotransmisores causan la activación o inhibición de las dendritas.
- La integración de las señales de excitación y de inhibición pueden producir picos.
- La contribución de dichas señales depende de la fuerza de la conexión sináptica.

# Arquitectura/Tipos

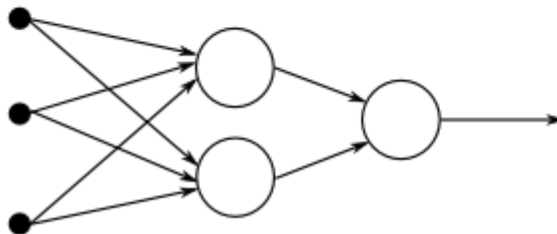
**Perceptron única capa:**  
clasificación lineal



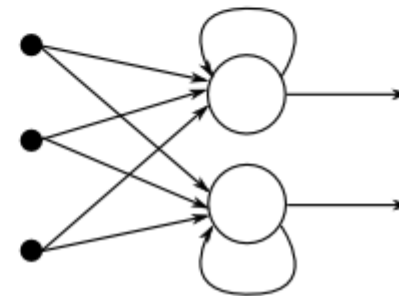
**Perceptron simple:**  
regresión / clasificación  
lineal



**Perceptron multicapa:**  
regresión / clasificación  
no lineal

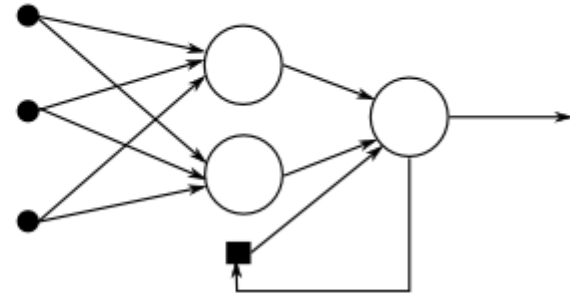
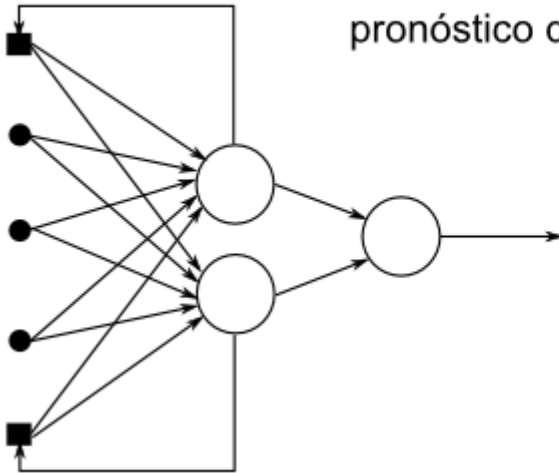


**Red competitiva:**  
clasificación no supervisada



# Arquitectura/Tipos

**Redes recurrentes:**  
pronóstico de series de tiempo





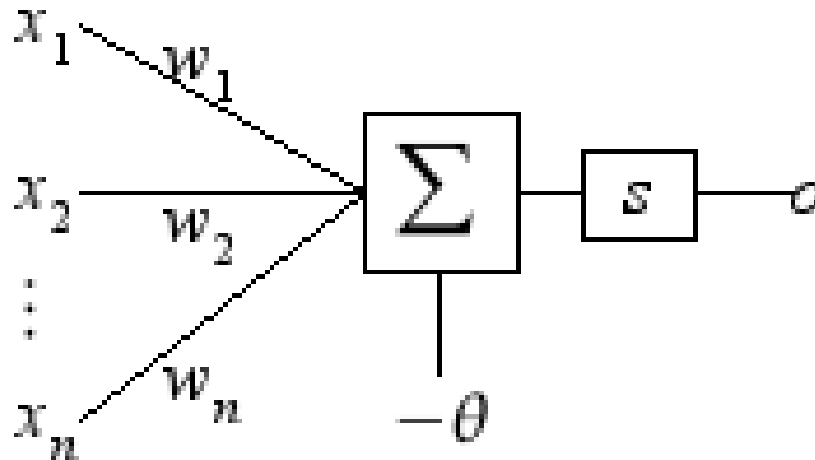
Institución Universitaria

# Perceptrón Simple



# Modelo del perceptrón

- En 1943 Warren McCulloch y Walter Pitts presentan una de las primeras aproximaciones a una neurona artificial. La característica principal de este modelo radicaba en que la sumatoria de las señales ponderadas de entrada eran comparadas con un umbral para determinar la salida de la neurona.



Cuando el valor de la sumatoria es mayor o igual que el umbral la salida de la neurona artificial es 1, de lo contrario 0.

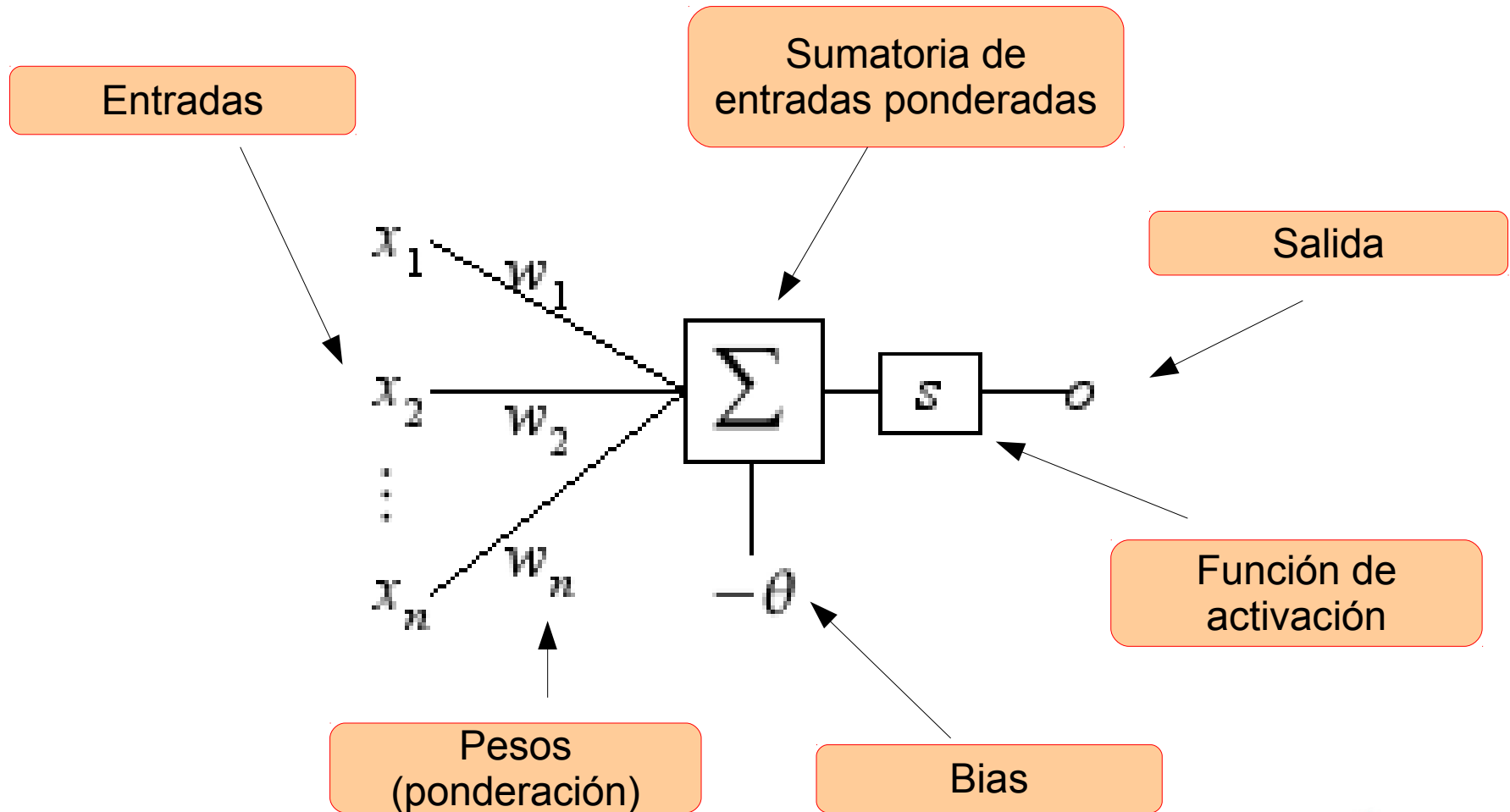
# Modelo del perceptrón

- A finales de los 50's, Frank Rosenblatt y otros investigadores desarrollaron una clase de redes neuronales llamadas perceptrones. Las neuronas en este modelo eran similares a las del modelo de McCulloch-Pitts. Sin embargo, la principal contribución de este nuevo modelo fue la introducción de una técnica (algoritmo) de aprendizaje automático para entrenar los perceptrones en problemas de reconocimiento de patrones.
- La idea principal del aprendizaje del perceptrón radica en utilizar el error de la clasificación para ir mejorando.
- Este puede aprender incluso utilizando valores de entrada randómicos.

# Modelo del perceptrón

- Este modelo presenta limitaciones que fueron expuestas en el trabajo de Marvin Minsky y Seymour Papert “*Perceptrons*”.
- Estas limitaciones fueron solucionadas mediante la introducción de perceptrones multicapa como se verá en clases posteriores.
- Para introducir el concepto de redes perceptrón primero trataremos el concepto operacional de perceptrones básicos (*Simple Perceptron*).

# Modelo del perceptrón



# Modelo del perceptrón: Clasificación-R.P.

- La función de activación generalmente corresponde a la función umbral o *hardlim*.
- Las entradas del perceptrón equivalen al vector de características del objeto de entrenamiento u objeto a clasificar.
- El Bias tiene relación con la distancia respecto al punto 0 del vector de clasificación.
- Los pesos se refieren a la “fuerza” de la característica en la clasificación.
- La salida del perceptrón se refiere a la clase del objeto.

# Modelo del perceptrón: Clasificación-R.P.

El modelo anterior puede ser expresado como:

$$a = f\left(\sum (W_p) + b\right)$$

Donde  $a$  es la salida de la neurona,  $f$  es la función de activación,  $W_p$  son las entradas ponderadas y  $b$  es el bias de la neurona. Desglozando un poco más tenemos:

$$a = f\left(x_1 * w_1 + x_2 * w_2 + \dots + x_n * w_n + b\right)$$



Institución Universitaria

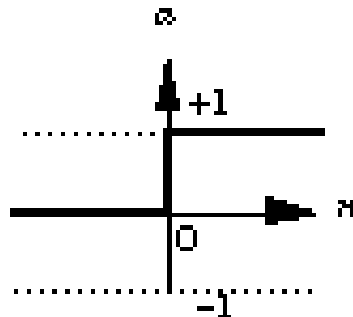
# Funciones de activación

# Funciones de activación

- La función de activación establece una “forma” de mostrar la salida del perceptrón.
- Equivale a un filtro de la salida del perceptrón.
- En problemas de clasificación generalmente se utiliza la función umbral o “*hardlim*”.
- En problemas de regresión se utiliza comúnmente la función sigmoidea.
- Sin embargo existen muchos tipos de funciones de activación.

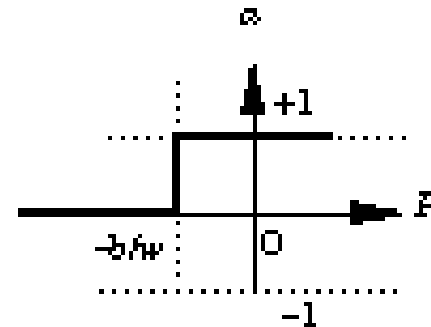


# Funciones de activación



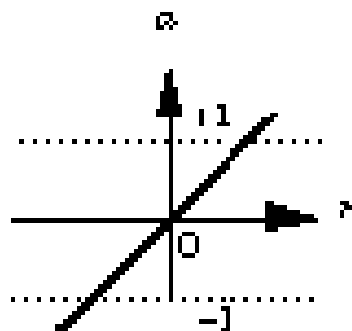
$$a = \text{hardlim}(x)$$

Hard Limit Transfer Function



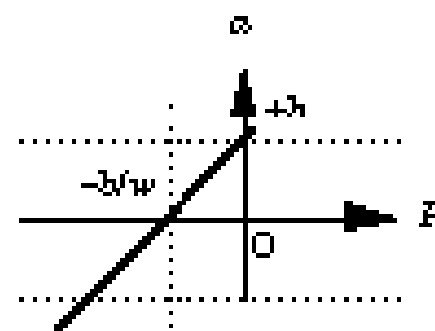
$$a = \text{hardlim}(wF + b)$$

Single-Input Hard Limit Neuron



$$a = \text{purelin}(x)$$

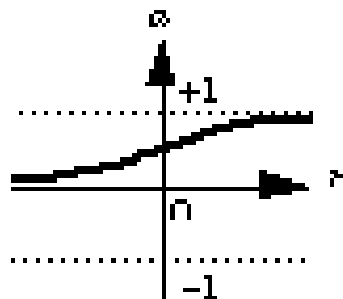
Linear Transfer Function



$$a = \text{purelin}(wF + b)$$

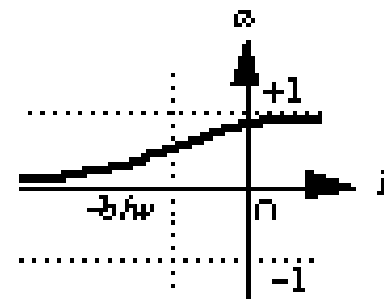
Single-Input Purelin Neuron

# Funciones de activación



$$a = \log \text{sig}(x)$$

Log-Sigmoid Transfer Function



$$a = \log \text{sig}(wF + b)$$

Single-Input Log-Sigmoid Neuron



Institución Universitaria

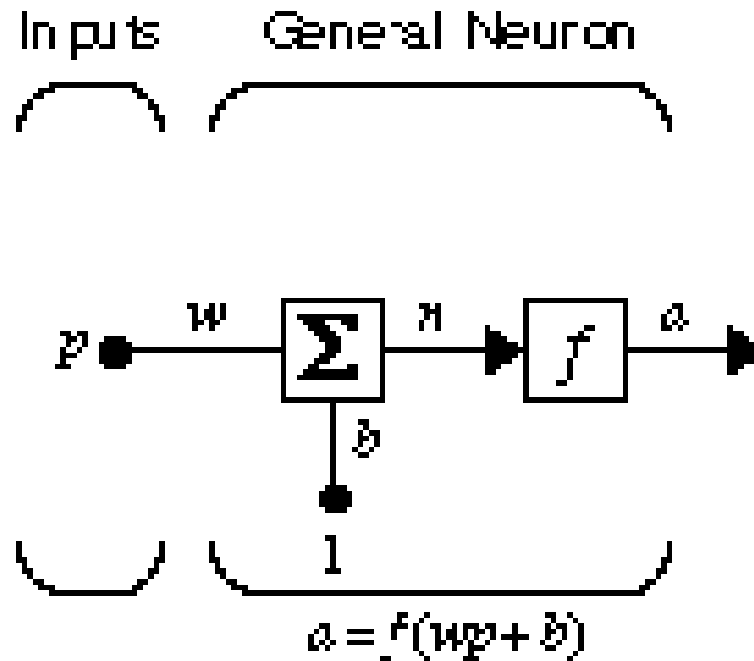
# Arquitectura

# Arquitectura

- La arquitectura del perceptrón se refiere a las múltiples formas que puede tomar dependiendo del problema a afrontar.
- La arquitectura depende tanto del número de entradas, como del número de salidas.

# Arquitectura

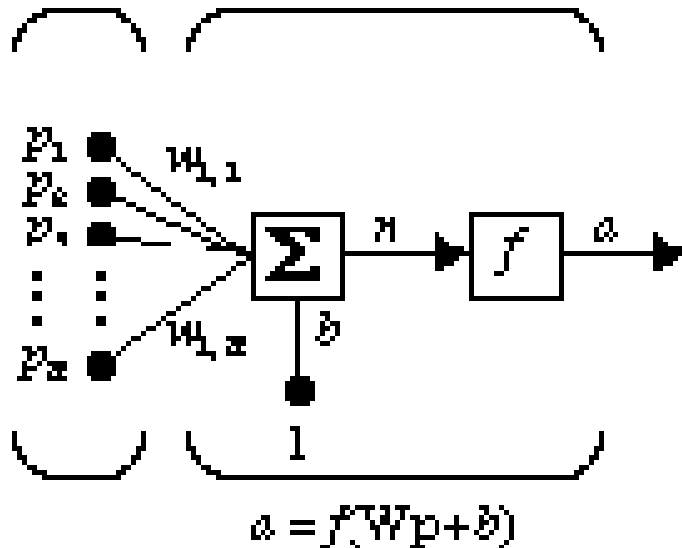
perceptrón con única entrada:



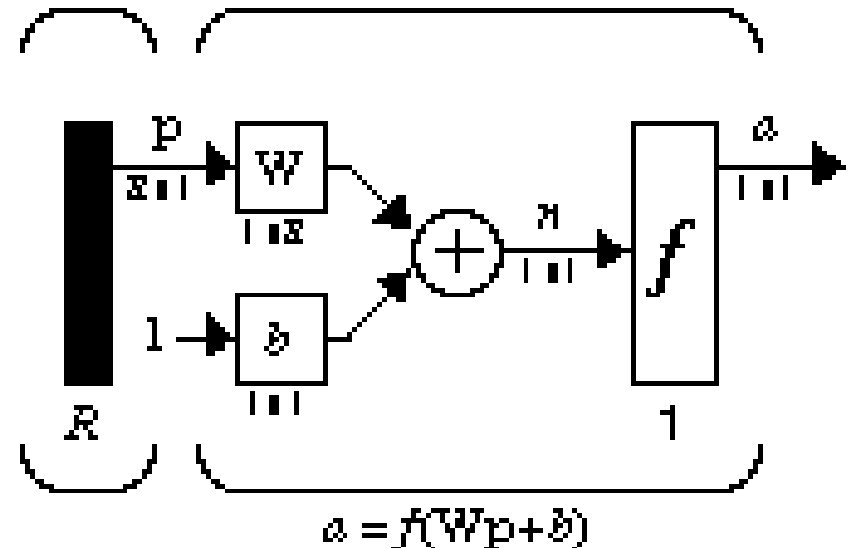
# Arquitectura

perceptrón con múltiple entrada:

Inputs Multiple-Input Neuron



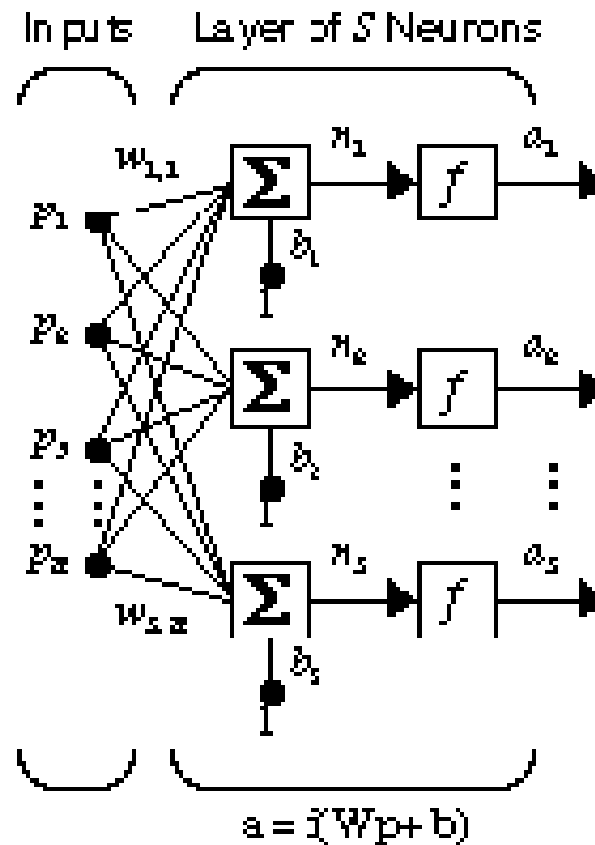
Input Multiple-Input Neuron



notación abreviada

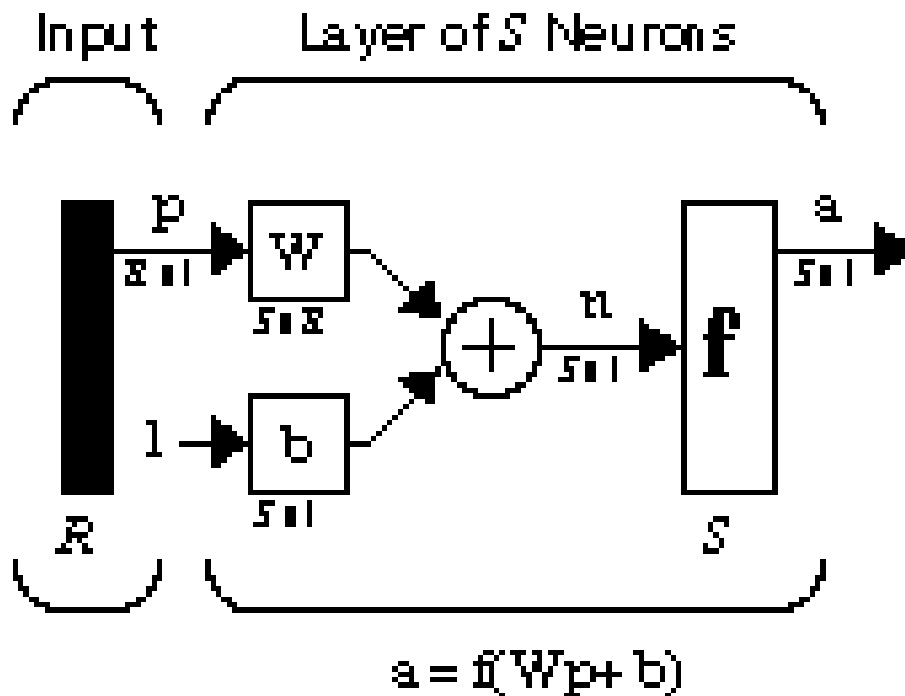
# Arquitectura

perceptrón con múltiple neuronas:



# Arquitectura

Notación matricial (álgebra lineal):



$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix}$$

$$p = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$





Institución Universitaria

# Entrenamiento: Regla de Aprendizaje Del Perceptrón Simple

# Regla de aprendizaje

- El aprendizaje de una neurona artificial perceptrón radica en encontrar el conjunto de pesos  $W$  que disminuya el error entre la clasificación real y la calculada.
- Para ello el aprendizaje propuesto por Rosenblatt radica en un método iterativo que utiliza el error de la salida para ir modificando los propios pesos de las entradas.
- Esto puede verse como encontrar el vector de pesos  $W$  cuyo vector ortogonal (perpendicular) separe de mejor forma las clases.

# Regla de aprendizaje

## medida de error

$$e = t - a$$

$t$  es la salida real o clase real y  $a$  es la clase calculada.

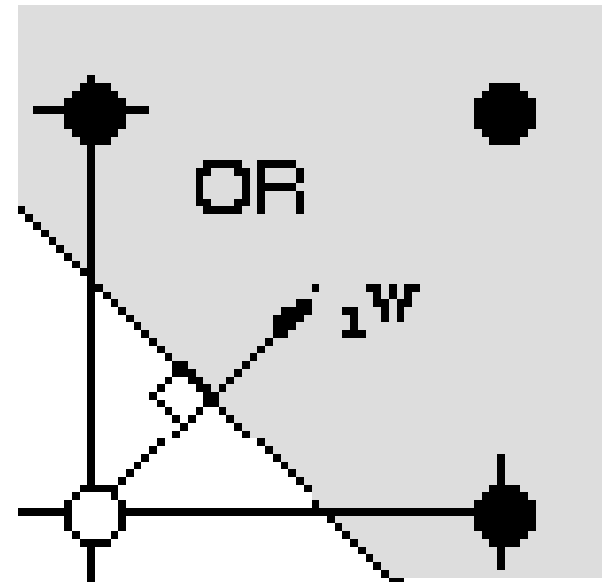
## Actualización pesos

$$W_{new} = W_{old} + ep'$$

$$W_{new} = W_{old} + (t - a) p'$$

## Actualización bias

$$b_{new} = b_{old} + e$$





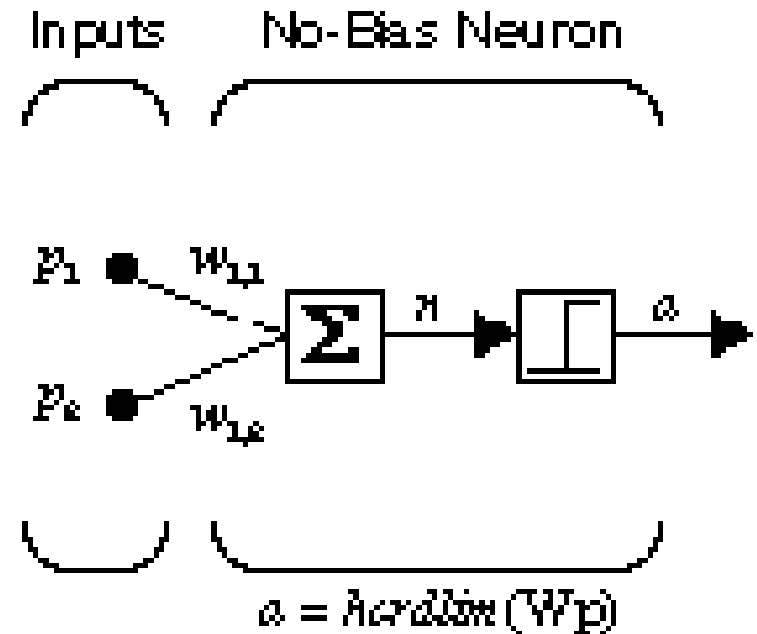
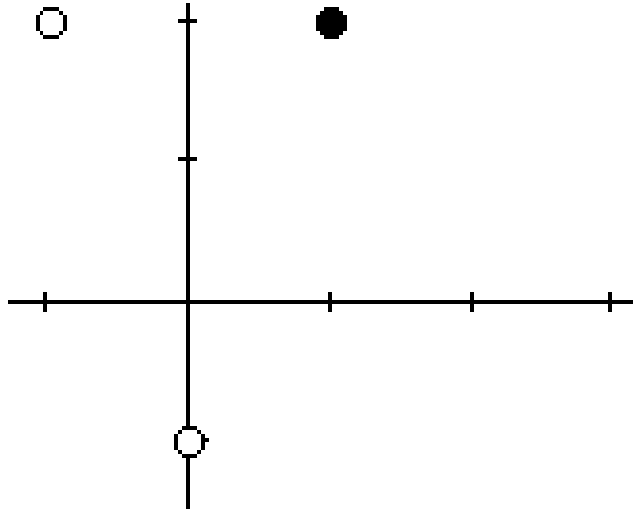
Institución Universitaria

## 3.3.1.6. Ejemplo

# Ejemplo

Supongamos los siguientes datos de entrenamiento y la arquitectura de perceptrón:

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, t_1 = 1 \right\} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_2 = 0 \right\} \quad \left\{ \mathbf{p}_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, t_3 = 0 \right\}$$



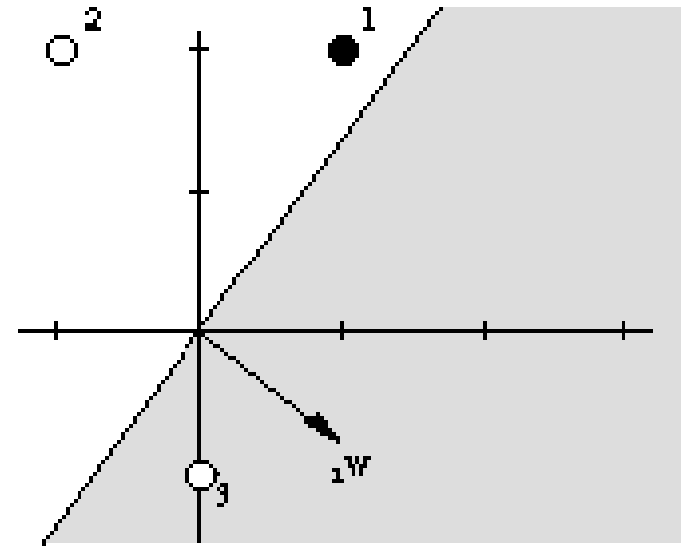
## Ejemplo

Supongamos el siguiente conjunto de pesos inicial randómico:

$${}_1\mathbf{w} = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix}$$

$$a = \text{hardlim}({}_1\mathbf{w}^T \mathbf{p}_1) = \text{hardlim}\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)$$

$$a = \text{hardlim}(-0.6) = 0$$

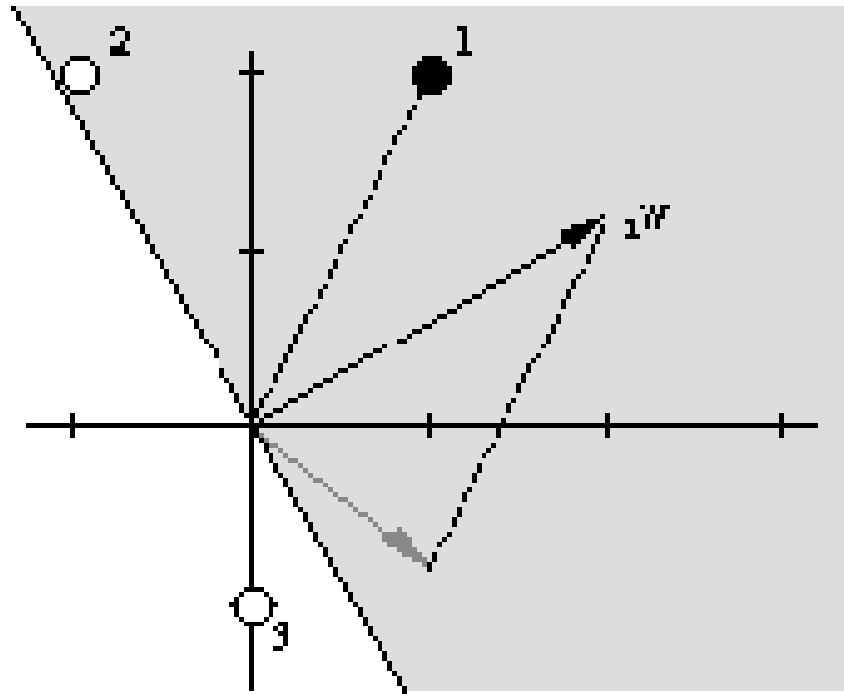


**Clasificación incorrecta!**

# Ejemplo

Modificamos los pesos:

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}_1 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix}$$



## Ejemplo

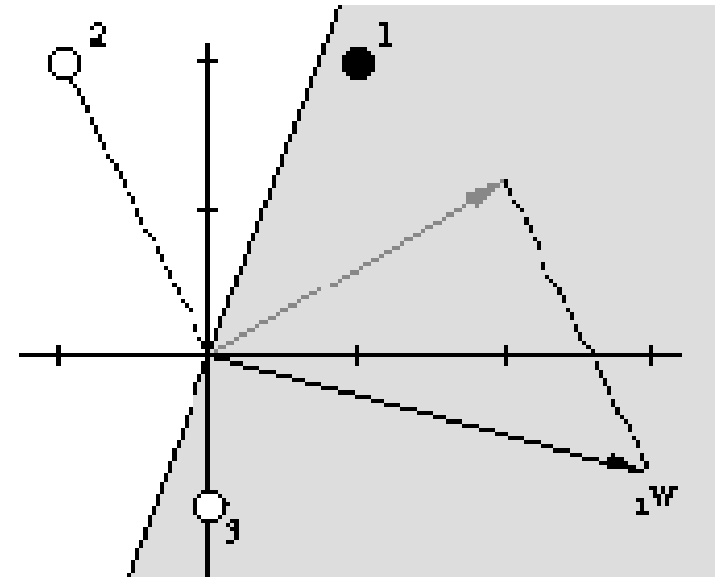
Para el segundo vector de características:

$$a = \text{hardlim}({}_1\mathbf{w}^T \mathbf{p}_2) = \text{hardlim}\left(\begin{bmatrix} 2.0 & 1.2 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix}\right)$$

$$a = \text{hardlim}(0.4) = 1 \quad \text{Clasificación incorrecta!}$$

Calculamos el nuevo W:

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_2 = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix} - \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix}$$





## Ejemplo

Para el segundo tercer de características:

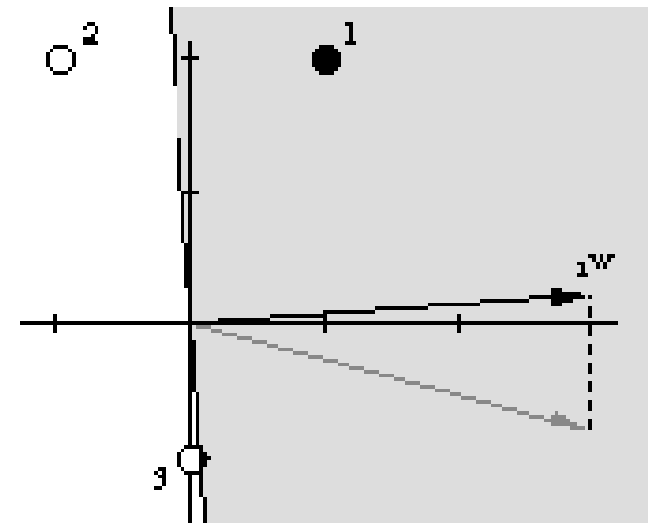
$$a = \text{hardlim}({}_1\mathbf{w}^T \mathbf{p}_3) = \text{hardlim}\left(\begin{bmatrix} 3.0 & -0.8 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$$

$$a = \text{hardlim}(0.8) = 1 \quad \text{Clasificación incorrecta!}$$

Modificamos los pesos:

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_3 = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix} - \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 3.0 \\ 0.2 \end{bmatrix}$$

Ahora nuestro clasificador funciona!



## Trabajemos en clase!

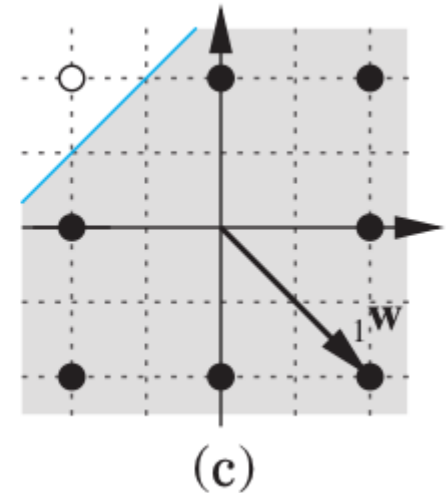
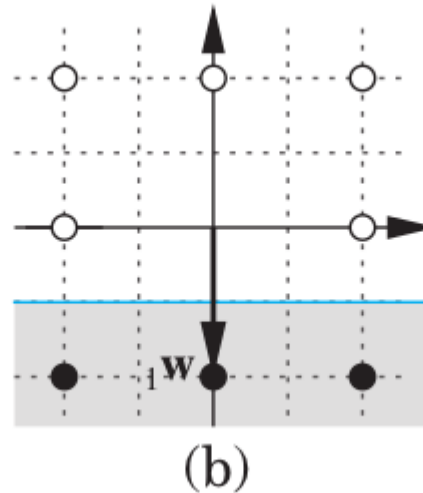
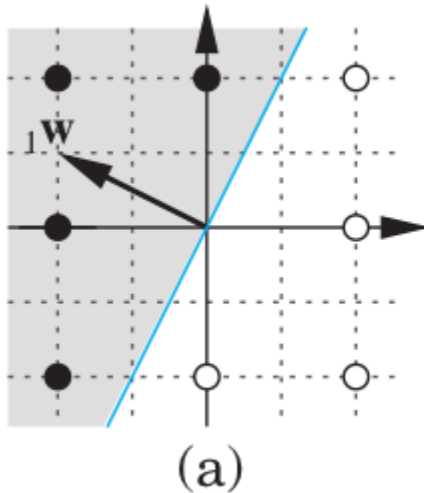
Dados los siguientes datos de entrenamiento, encontrar el conjunto de pesos que clasifica de forma correcta dichos datos:

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, t_1 = [1] \right\} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, t_2 = [0] \right\}$$

$$\mathbf{W} = [0.5 \ -1 \ -0.5] \quad b = 0.5$$

# Trabajemos en clase!

Dados los siguientes datos de entrenamiento, encontrar el conjunto de pesos que clasifica de forma correcta dichos datos:



$$\text{(a) } {}_1\mathbf{w}^T = \begin{bmatrix} -2 & 1 \end{bmatrix}, \quad \text{(b) } {}_1\mathbf{w}^T = \begin{bmatrix} 0 & -2 \end{bmatrix}, \quad \text{(c) } {}_1\mathbf{w}^T = \begin{bmatrix} 2 & -2 \end{bmatrix}.$$

# Trabajemos en clase!

Dados los siguientes datos de entrenamiento, encontrar el conjunto de pesos que clasifica de forma correcta dichos datos:

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} \left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} \left\{ \mathbf{p}_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$$
$$\left\{ \mathbf{p}_4 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \left\{ \mathbf{p}_5 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \mathbf{t}_5 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\} \left\{ \mathbf{p}_6 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}, \mathbf{t}_6 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}$$
$$\left\{ \mathbf{p}_7 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \mathbf{t}_7 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\} \left\{ \mathbf{p}_8 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}, \mathbf{t}_8 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}.$$

$$\mathbf{W}(0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{b}(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

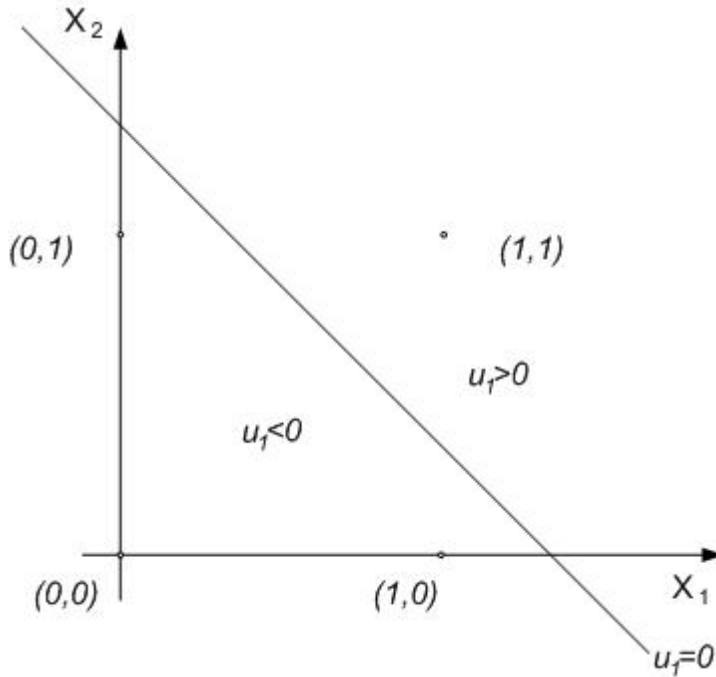


Institución Universitaria

## 2.4. MultiLayer Perceptron (Perceptrón Multicapa)

# Introducción

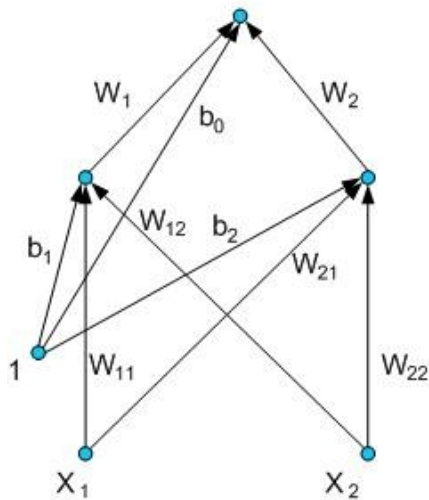
## El problema XOR



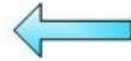
Como vimos anteriormente, una red de perceptrones entrelazados puede solucionar este problema.

# Introducción

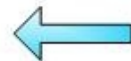
## Perceptrón Multicapa



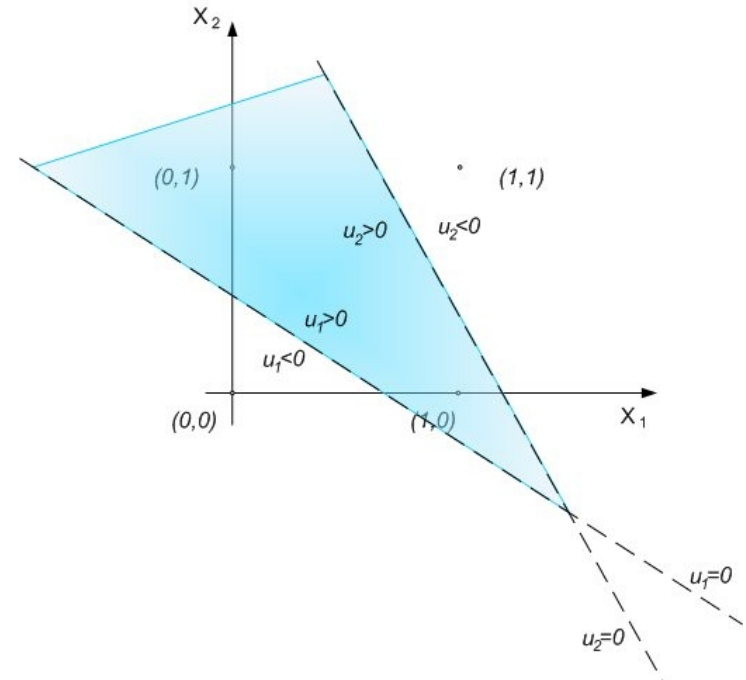
Output layer



Hidden layer



Input layer





Institución Universitaria

# Introducción

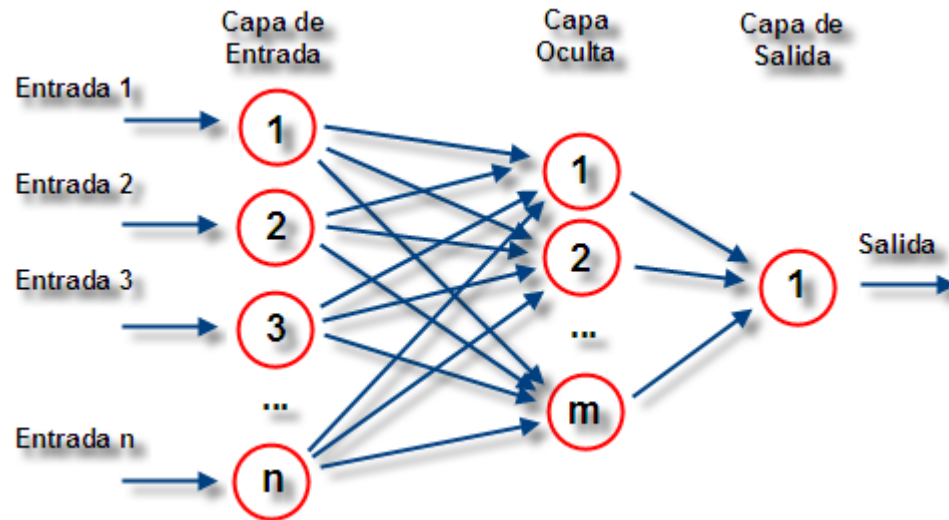
## Perceptrón Multicapa

- Es una arquitectura de red neuronal artificial formada por múltiples capas.
- Es la arquitectura más conocida.
- Es totalmente conectado.
- Tiene una **capa de entrada**, una o más **capas ocultas** y una **capa de salida**.
- Aplicación en reconocimiento de patrones, segmentación de imágenes e identificación de sistemas no lineales, entre otros.



# Introducción


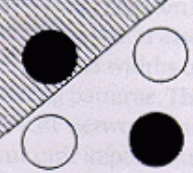



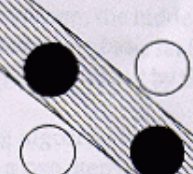

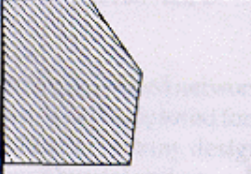

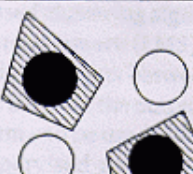

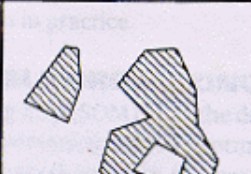
## Perceptrón Multicapa



Arquitectura

# Propiedades

## Particionamiento del espacio de características

Structure	Description of decision regions	Exclusive-OR problem	Classes with meshed regions	General region shapes
 Single layer	Half plane bounded by hyperplane			
 Two layer	Arbitrary (complexity limited by number of hidden units)			
 Three layer	Arbitrary (complexity limited by number of hidden units)			

# Entrenamiento

El entrenamiento de una red neuronal se puede entender como un proceso de **optimización matemática** de un conjunto de valores  $W$  (pesos) y  $b$  (bias), respecto a una medida de error.

Dos algoritmos muy conocidos para el entrenamiento de redes neuronales “perceptrón multicapa” son:

- **Propagación hacia atrás (backpropagation):** Basado en el descenso del gradiente.
- **Levenberg-Marquardt:** Basado en el método Gauss-Newton.

Lectura en clase: [http://es.wikipedia.org/wiki/Optimizaci3n\\_\(matem3tica\)](http://es.wikipedia.org/wiki/Optimizaci3n_(matem3tica))

# Propagación hacia atrás

- Algoritmo de aprendizaje supervisado.
- Arthur E. Bryson lo describe como un método dinámico de optimización en 1969.
- Paul Werbos, David Rumelhart, Geoffrey Hinton y Ronald Williams lo aplican en el contexto de las redes neuronales en 1974 con lo cual ganó un importante reconocimiento en la comunidad científica.
- Este aporte marcó el renacimiento de la investigación en las redes neuronales.
- Esta basado en el descenso del gradiente (regla delta generalizada).
- Utiliza el cálculo de derivadas parciales del error de la red con respecto a los parámetros de la misma.

# Propagación hacia atrás

El algoritmo puede verse de forma general como:

- Empezar con unos pesos sinápticos cualquiera (generalmente elegidos al azar).
- Introducir unos datos de entrada (en la capa de entradas) elegidos al azar entre los datos de entrada que se van a usar para el entrenamiento.
- Dejar que la red genere un vector de datos de salida (propagación hacia delante).
- Comparar la salida generada por la red con la salida deseada.
- La diferencia obtenida entre la salida generada y la deseada (denominada *error*) se usa para ajustar los pesos sinápticos de las neuronas de la capa de salidas.
- El error se *propaga hacia atrás* (back-propagation), hacia la capa de neuronas anterior, y se usa para ajustar los pesos sinápticos en esta capa.
- Se continua propagando el error hacia atrás y ajustando los pesos hasta que se alcance la capa de entradas.

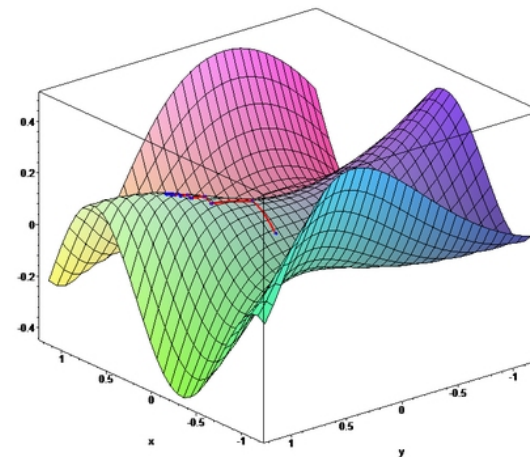
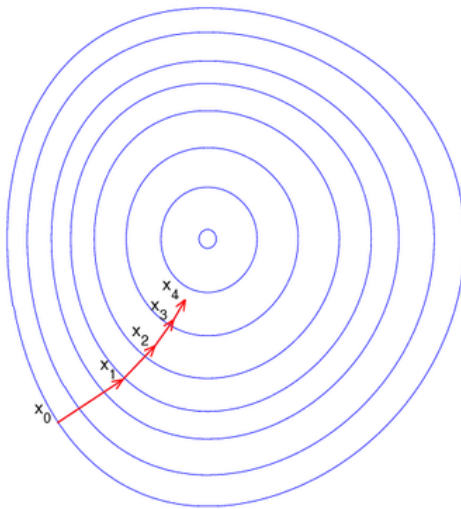
# Algunos conceptos

## Descenso del gradiente

Este método utiliza el concepto de derivada parcial para optimizar el valor de un parámetro respecto a un objetivo. Para nuestro caso, debemos optimizar los parámetros de la red neuronal, respecto al error de la salida.

$$x_{(n+1)} = x_{(n)} - \alpha_n * \partial f(x_n)$$

La derivada parcial de la función indica la pendiente de la misma en el punto  $x_n$ .





Institución Universitaria

# Algunos conceptos

Este método lo utilizaremos en el entrenamiento de nuestra red neuronal tipo perceptrón multicapa con el fin de ajustar los pesos de forma iterativa. Por otra parte, **necesitamos que las funciones de activación de las neuronas de la red sean derivables.**

## ¿Por qué propagación hacia atrás?

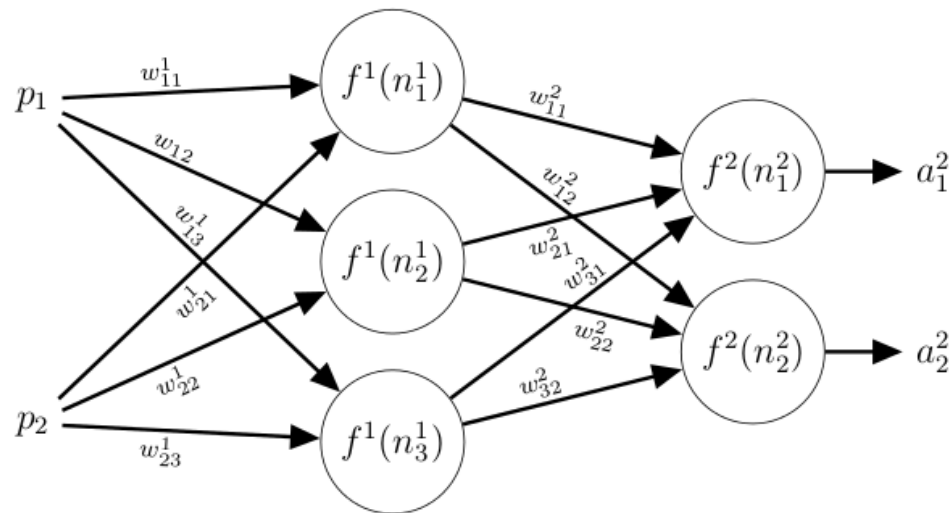
Debido a que no sabemos la relación directa entre el error generado en la capa de salida de nuestra red y los valores de los parámetros de las capas ocultas, necesitamos ir ajustando de forma iterativa dichos valores propagando el error desde la capa de salida hasta la capa de entrada.



# Propagación hacia atrás

## Notación

Antes de continuar con el método de entrenamiento backpropagation, necesitamos definir la notación que utilizaremos para describir nuestra red neuronal. Dada la siguiente red neuronal tenemos:





# Propagación hacia atrás

$W_n$  es una matriz que representa el conjunto de pesos de la capa  $n$ . Por lo que:

$$W_1 = \begin{matrix} & \begin{matrix} 1 & 1 & 1 \end{matrix} \\ \begin{matrix} 1 \\ 1 \\ 1 \end{matrix} & \begin{matrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{matrix} \end{matrix} \quad W_2 = \begin{matrix} & \begin{matrix} 2 & 2 \end{matrix} \\ \begin{matrix} 2 \\ 2 \\ 2 \end{matrix} & \begin{matrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{matrix} \end{matrix}$$

$p$  es un vector columna de entradas por lo que:  $p = \begin{matrix} p1 \\ p2 \end{matrix}$

$n_i$  es un vector columna de los valores de activación o entradas de las neuronas de la capa  $i$ . Por lo que:

$$n_1 = \begin{matrix} 1 \\ 1 \\ 1 \end{matrix} \quad n_2 = \begin{matrix} 2 \\ 2 \\ 2 \end{matrix}$$

# Propagación hacia atrás

$a_i$  es un vector columna de los valores de salida de la capa  $i$ . Por lo que:

$$\begin{array}{cc}
 \begin{array}{c} 1 \\ a_1 \\ 1 \\ a_1 = a_2 \\ 1 \\ a_3 \end{array} & 
 \begin{array}{c} 2 \\ a_1 \\ 2 \\ a_2 \end{array}
 \end{array}$$

# Propagación hacia atrás

## Actualización de los pesos y bias

$$W_{(m)} = W_m + \eta * a_{(m-1)} * (f'_m(n_m) \cdot \delta_m^T)$$

$$b_{(m)} = b_m + \eta * (f'_m(n_m) \cdot \delta_m^T)$$

Donde  $\eta$  es la tasa de aprendizaje o factor de aprendizaje,  $m = 0, 1, 2$  son los índices de las capa. Para cada capa se tiene: un vector de salida  $a_m$ , el vector de propagación del error  $\delta_m$ , el vector de activación  $n_m$  y la derivada de la función de activación  $f'_m()$ . La operación “.” se refiere a la multiplicación punto a punto entre dos vectores files como:

$$\begin{aligned} x \cdot y &= [x_1 \ x_2 \ x_3 \dots x_k] \cdot [y_1 \ y_2 \ y_3 \dots y_k] \\ &= [x_1 * y_1 \ x_2 * y_2 \ x_3 * y_3 \dots x_k * y_k] \end{aligned}$$

# Propagación hacia atrás

## Evaluación de la red neuronal

A partir de un vector de entrada  $p$  se resuelve la misma hasta la salida. Para la red anterior se calcula el vector de activación  $a_1$  de la capa oculta 1:

$$n_1 = (W_1^T p) = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} w_{11} * p_1 + w_{12} * p_2 + w_{13} * p_3 \\ w_{21} * p_1 + w_{22} * p_2 + w_{23} * p_3 \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \end{bmatrix}$$

$$a_1 = f_1(n_1) = \begin{bmatrix} f_1(n_1) \\ f_1(n_2) \end{bmatrix}$$

Con estas salidas de la capa procedemos a calcular la salida de la capa 2:

$$n_2 = (W_2^T a_1) = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} w_{11} * a_1 + w_{12} * a_2 + w_{13} * a_3 \\ w_{21} * a_1 + w_{22} * a_2 + w_{23} * a_3 \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \end{bmatrix}$$

$$a_2 = f_2(n_2) = \begin{bmatrix} f_2(n_1) \\ f_2(n_2) \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

# Propagación hacia atrás

## Evaluación de la red neuronal

Una vez se calcula la salida de la red, se procede a actualizar los pesos mediante la regla de actualización de pesos revisada anteriormente. Para ello realizamos de forma iterativa los siguientes pasos:

### Cálculo del error:

$$\delta_2 = t - a_2$$

### Propagación del error hacia atrás

$$\delta_1 = W_2 \delta_2 = \begin{bmatrix} w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} \delta_2 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} w_{21} * \delta_2 + w_{22} * \delta_2 \end{bmatrix} = \begin{bmatrix} \delta_2 \end{bmatrix}$$

# Propagación hacia atrás

**Actualización de los pesos W2:** Para ello se utiliza la ecuación 1. Sin embargo primero es necesario calcular la multiplicación punto a punto entre  $f'_m(n_m) \cdot \delta_m$

$$f'_2(n_2) \cdot \delta_2 = [f'_2(n_1) \quad f'_2(n_2)] \cdot [\delta_1 \quad \delta_2] = [f'_2(n_1) * \delta_1 \quad f'_2(n_2) * \delta_2]$$

Con el anterior resultado y reemplazando en la ecuación 1 tenemos:

$$W_2 = W_2 + \eta * a_1 * (f'_2(n_2) \cdot \delta_2)$$

$$W_2 = \begin{bmatrix} w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} + \eta \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} [f'_2(n_1) * \delta_1 \quad f'_2(n_2) * \delta_2]$$

$$W_2 = \begin{bmatrix} w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} + \eta \begin{bmatrix} f'_2(n_1) * \delta_1 * a_1 & f'_2(n_2) * \delta_2 * a_1 \\ f'_2(n_1) * \delta_1 * a_2 & f'_2(n_2) * \delta_2 * a_2 \\ f'_2(n_1) * \delta_1 * a_3 & f'_2(n_2) * \delta_2 * a_3 \end{bmatrix}$$

# Propagación hacia atrás

...Continuación:

$$W_2 = \begin{bmatrix} w_{11} + \eta * f'_2(n_1) * \delta_1 * a_1 & w_{12} + \eta * f'_2(n_2) * \delta_2 * a_1 \\ w_{21} + \eta * f'_2(n_1) * \delta_1 * a_2 & w_{22} + \eta * f'_2(n_2) * \delta_2 * a_2 \\ w_{31} + \eta * f'_2(n_1) * \delta_1 * a_3 & w_{32} + \eta * f'_2(n_2) * \delta_2 * a_3 \end{bmatrix}$$

**Actualización de los pesos W1:**

$$f'_1(n_1) \cdot \delta_1 = [f'_1(n_1) \quad f'_1(n_2) \quad f'_1(n_3)] \cdot [\delta_1 \quad \delta_2 \quad \delta_3] = [f'_1(n_1) * \delta_1 \quad f'_1(n_2) * \delta_2 \quad f'_1(n_3) * \delta_3]$$

$$W_1 = W_1 + \eta * a_0 * (f'_1(n_1) \cdot \delta_1)$$

$$W_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} + \eta \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} [f'_1(n_1) * \delta_1 \quad f'_1(n_2) * \delta_2 \quad f'_1(n_3) * \delta_3]$$

$$W_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} + \eta \begin{bmatrix} f'_1(n_1) * \delta_1 * p_1 & f'_1(n_2) * \delta_2 * p_1 & f'_1(n_3) * \delta_3 * p_1 \\ f'_1(n_1) * \delta_1 * p_2 & f'_1(n_2) * \delta_2 * p_2 & f'_1(n_3) * \delta_3 * p_2 \end{bmatrix}$$

# Propagación hacia atrás

...Continuación:

$$W_1 = \begin{bmatrix} w_{11} + \eta * f'_1(n_1) * \delta_1 * p_1 & w_{12} + \eta * f'_1(n_2) * \delta_2 * p_1 & w_{13} + \eta * f'_1(n_3) * \delta_3 * p_1 \\ w_{21} + \eta * f'_1(n_1) * \delta_1 * p_2 & w_{22} + \eta * f'_1(n_2) * \delta_2 * p_2 & w_{23} + \eta * f'_1(n_3) * \delta_3 * p_2 \end{bmatrix}$$



# Propagación hacia atrás

## Función de activación diferenciable

Como se mencionó anteriormente, la función de activación de las neuronas de la red deben ser diferenciables con el fin de poder aplicar Backpropagation. Por ejemplo, la función tangente sigmoidea:

$$f(n) = \frac{(e^n - e^{-n})}{(e^n + e^{-n})}$$

$$f'(n) = 1 - f(n)^2$$

# Propagación hacia atrás

## Ejercicio en clase:

Dada la entrada  $p = \begin{bmatrix} 1.0 \\ -2.0 \end{bmatrix}$ , la salida objetivo  $t = \begin{bmatrix} 0.5 \\ -1.0 \end{bmatrix}$  y los pesos  $W_1 = \begin{bmatrix} 0.3 & 0.5 & 0.7 \\ 0.2 & 0.3 & 0.4 \end{bmatrix}$

$$W_2 = \begin{bmatrix} 0.1 & 0.4 \\ 0.6 & 0.3 \\ 0.7 & 0.8 \end{bmatrix}$$

realizar un ciclo de backpropagation, con la estructura de red neuronal anterior.

# Bibliografía

1. Patrick Henry Winston, *Artificial Intelligence (3rd Edition)*. Addison-Wesley, 1992.
2. Tim Jones, *Artificial Intelligence: A systems approach*. Infinity Science Press LLC. 2008.
3. Artificial Intelligence: A Modern Approach. Stuart Russell, Peter Norvig. Prentice Hall. 2010.
4. Tim Jones, *AI Application Programming*. Charles River Media. 2003.
5. David M. Bourg et. al., *AI for Game Developers*. O'Reilly. 2004.