REDES NEURONALES

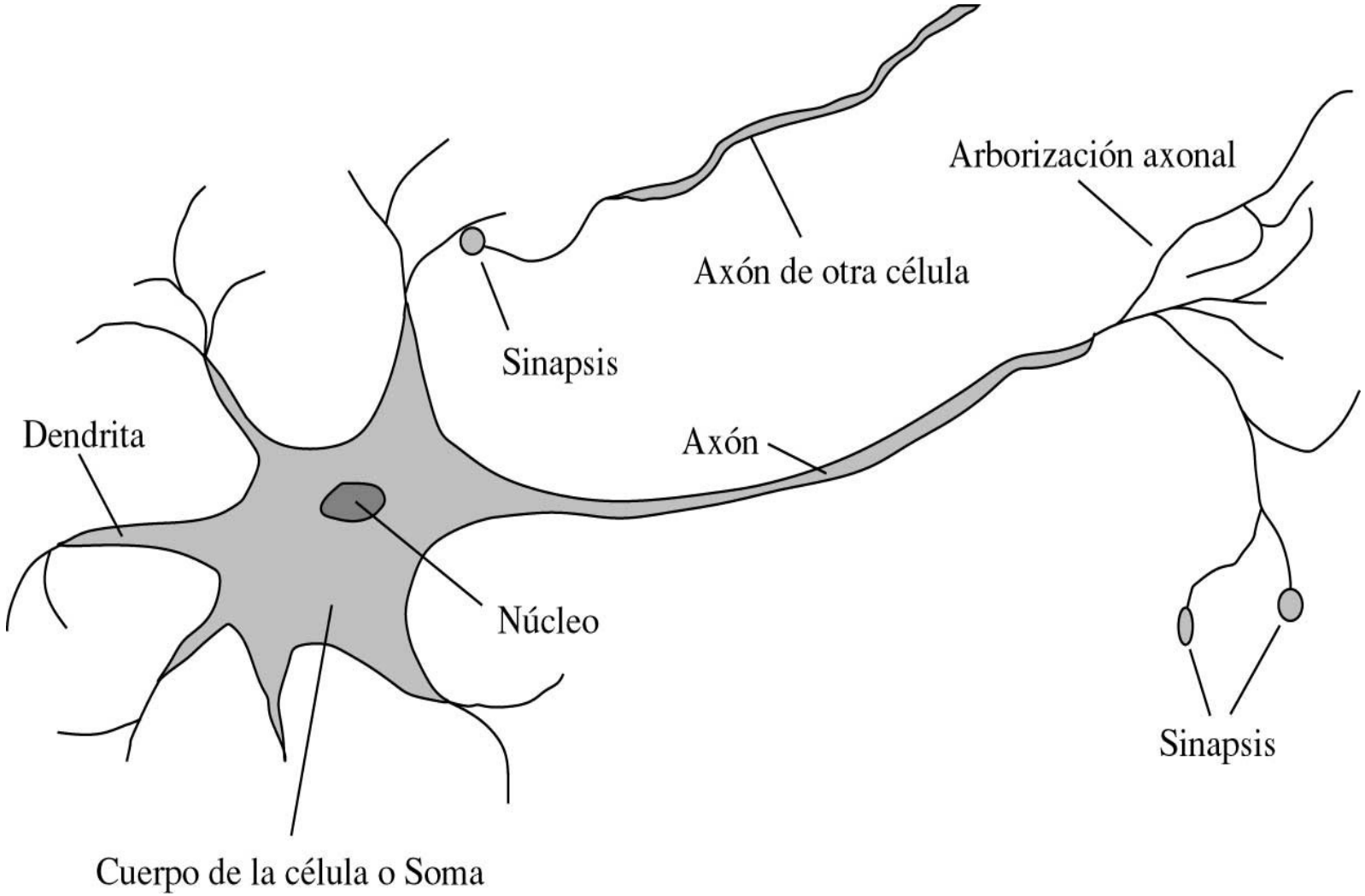
REPRESENTACIÓN

- Redes Neuronales Artificiales (ANN) se inspiran en las redes neuronales biológicas.

- Humanos
- Numero de neuronas $\sim 10^{10}$
- Conexiones por neurona $\sim 10^{4.5}$
- Reconocer una persona en ~1 seg.

- Neurociencia

NEURONA


Arborización axonal
Axón de otra célula
Sinapsis
Dendrita
Axón
Núcleo
Sinapsis
Cuerpo de la célula o Soma

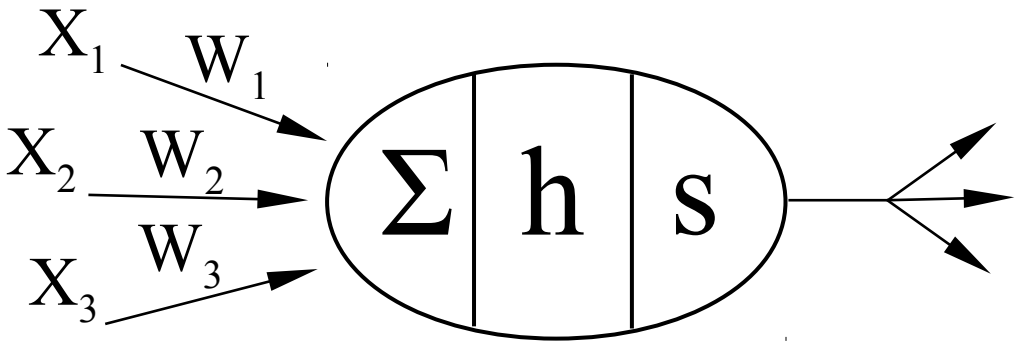
NEURONA

- **Dendritas:** reciben información desde el exterior o de otra dendrita
- **Soma** (cuerpo): suma y decide

  Se queda en reposo

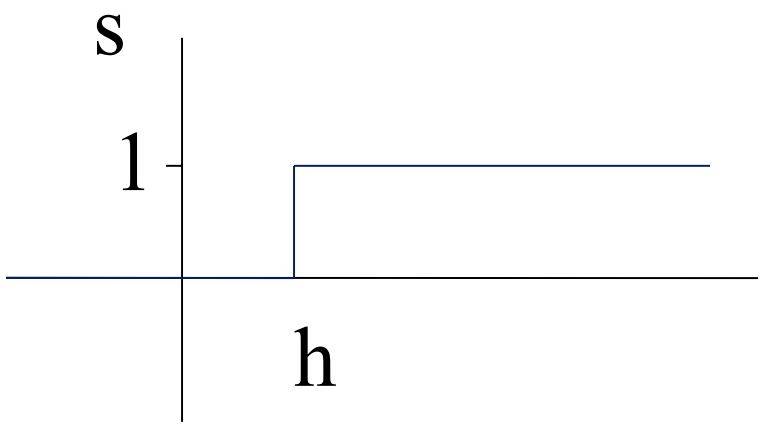
  Suma supera cierto umbral y transmite el pulso por el axón
- **Axón:** potencial de acción
- **Sinapsis:** conexión (química) de una neurona a otra
- Neurotransmisores inhibitorios o exitatorios
- **Aprendizaje:** modificación de las sinapsis o creación de sinapsis nuevas

NEURONA FORMAL

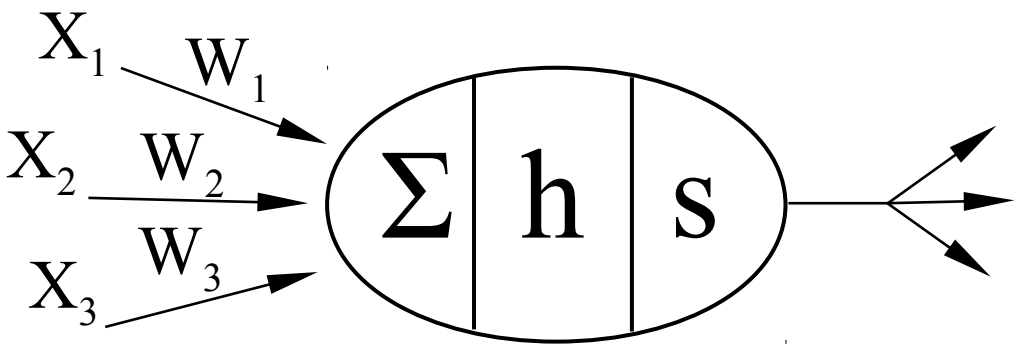


$$s \begin{cases} 0 & \Sigma < h \\ 1 & \Sigma > h \end{cases}$$

X = Entradas
W = Pesos
Σ = Función de sumatoria
H = Función de umbral
s = Salida



NEURONA FORMAL

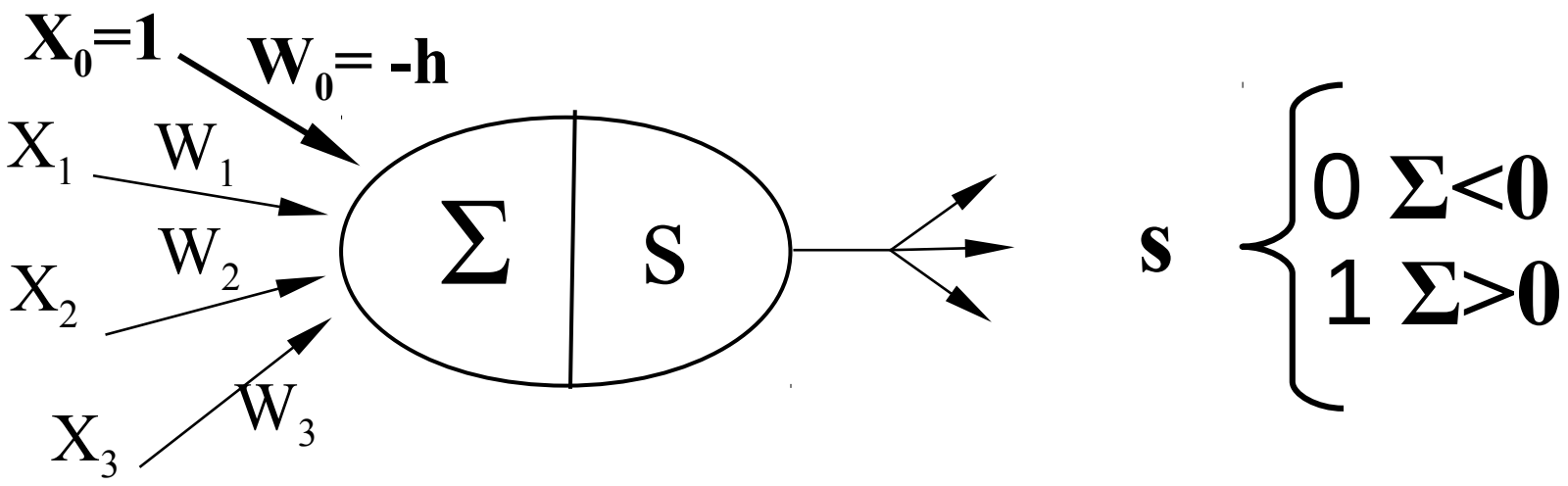


Función sumatoria = $w_1x_1+w_2x_2+w_3x_3$

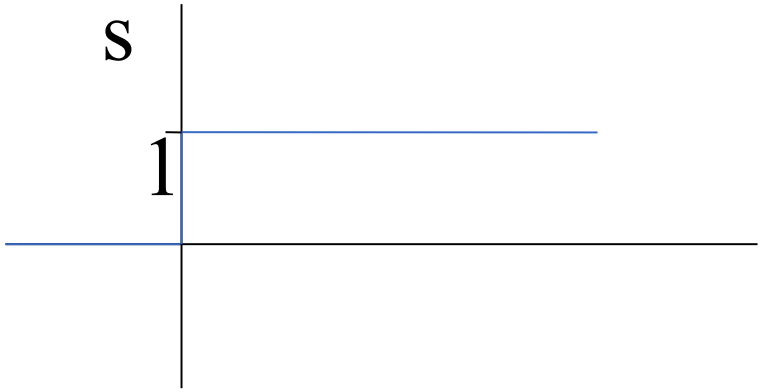
Para n entradas:

Función sumatoria = $\Sigma w_i x_i$ (i=1…n)

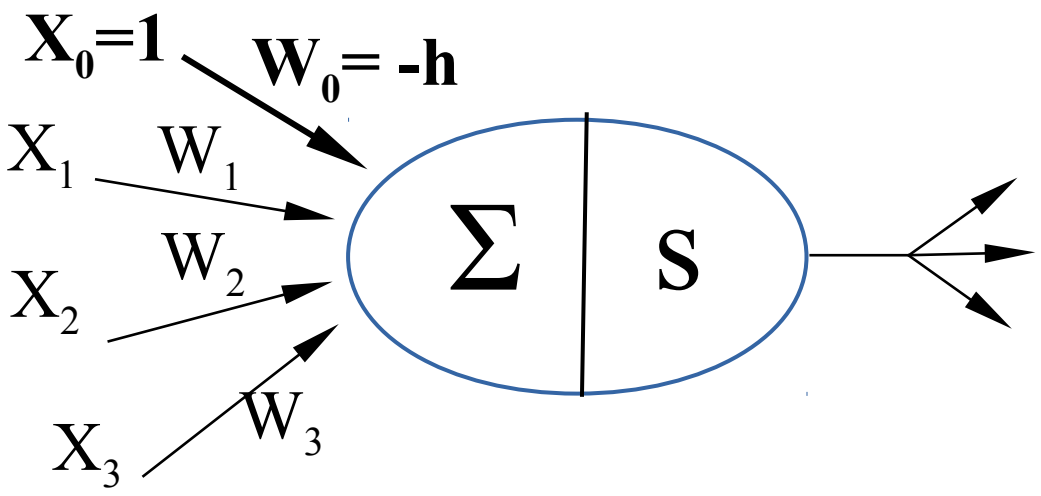
NEURONA FORMAL



Se elimina el **umbral h** insertando una entrada más a la neurona cuyo valor es siempre **positivo** y su **peso** es el valor **negativo de h.**



NEURONA FORMAL

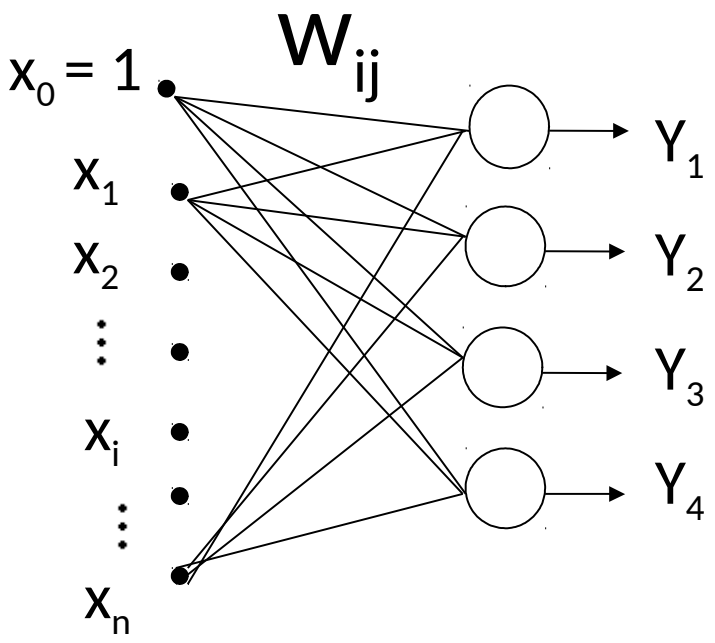


Función sumatoria = $\mathbf{w_0x_0}+w_1x_1+w_2x_2+w_3x_3$

Para n entradas:

Función sumatoria = $\Sigma w_i x_i$ (i=**0**…n)

PERCEPTRÓN



Es una red neuronal de **una capa** donde las entradas se conectan a todas las neuronas.

W_{ij} es el **peso** de la conexión de la **neurona i** a la **neurona j**

REGLA DE APRENDIZAJE

$$W_{ij}^{nuevo} = W_{ij}^{viejo} + C\,(t_j^{(l)} - y_j)\,a_i^{(l)}$$

C = coeficiente/tasa de aprendizaje < 1

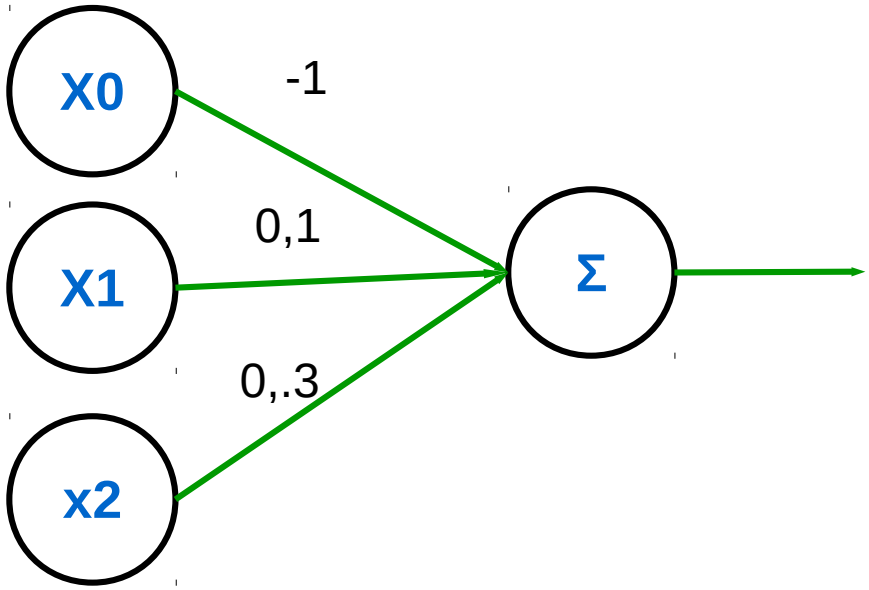
t_j = salida deseada

y_j = salida producida

a_i = entrada

Se itera calculando los pesos para cada conexión hasta que no exista error

APRENDER AND



Inicio con pesos aleatorios

X_0 siempre tiene como entrada 1

Al colocar X_0 si el valor es positivo entonces es verdadero.

Taza de aprendizaje r: 0,2

APRENDER AND

Se inicia recorriendo los ejemplos y para cada uno calcular:

Salida esperada z

Se calcula la suma **s** multiplicando las entradas con sus pesos

Suma s = x0*w0+x1*w1+x2*w2

Se obtiene la salida de la red

Salida de la red n = si s> t entonces 1 sino 0

Se calcula el error **e**

Error e = z – n

APRENDER AND

Se calcula la corrección a partir del error **e** y la tasa de aprendizaje **r**

Corrección d = r * e

Se calcula los nuevos pesos a partir de el peso anterior, el valor de entrada y la corrección **d**

nuevo w0 = w0 + x0*d
nuevo w1 = w1 + x1*d
nuevo w2 = w2 + x2*d

Repetir el proceso hasta que todos los ejemplos den la salida esperada.

PRIMERA ITERACIÓN

Entrada				Pesos			Salida		Error e	Corr d	Nuevos pesos		
x0	x1	x2	z	w0	w1	w2	s	n			w0	w1	w2
1	0	0	0	-1	0,1	0,3	-1	**0**	0	0	-1	0,1	0,3
1	0	1	0	-1	0,1	0,3	-0,7	**0**	0	0	-1	0,1	0,3
1	1	0	0	-1	0,1	0,3	-0,9	0	0	0	-1	0,1	0,3
1	1	1	1	-1	0,1	0,3	-0,6	**0**	**1**	**0,2**	-0,8	0,3	0,5

SEGUNDA ITERACIÓN

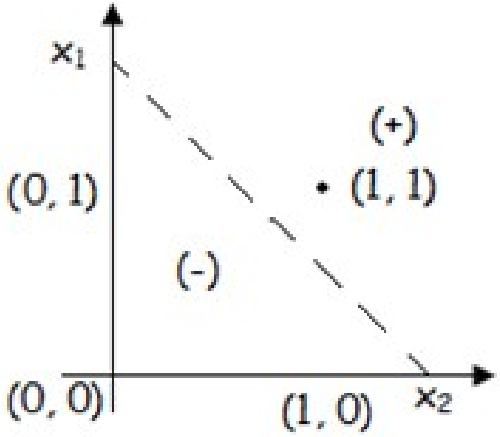
Entrada				Pesos			Salida		Error e	Corr d	Nuevos pesos		
x0	x1	x2	z	w0	w1	w2	s	n			w0	w1	w2
1	0	0	0	-0,8	0,3	0,5	-0,8	**0**	0	0	-0,8	0,3	0,5
1	0	1	0	-0,8	0,3	0,5	-0,3	**0**	0	0	-0,8	0,3	0,5
1	1	0	0	-0,8	0,3	0,5	-0,5	**0**	0	0	-0,8	0,3	0,5
1	1	1	1	-0,8	0,3	0,5	0	**1**	0	0	-0,8	0,3	0,5

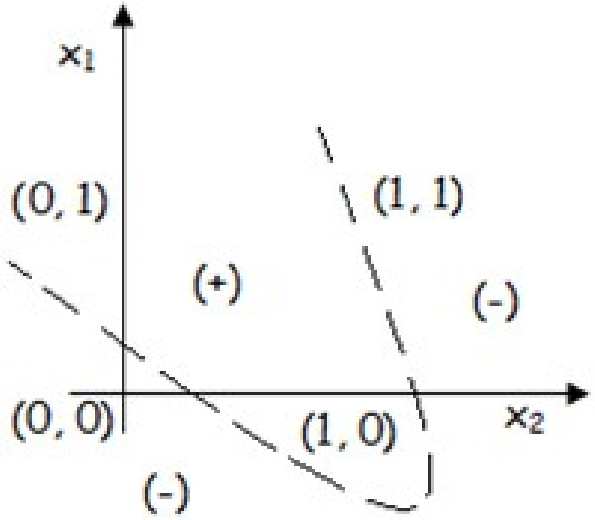
TEOREMA DE CONVERGENCIA

Si existe una solución para $\mathbf{a}^{(p)}$ -> $\mathbf{t}^{(p)}$ entonces puede encontrarse con el algoritmo de aprendizaje.

Para que exista esta solución el problema debe ser **linealmente separable.**

LINEALMENTE SEPARABLE





x_1	x_2	y
1	1	1
0	1	0
1	0	0
0	0	0

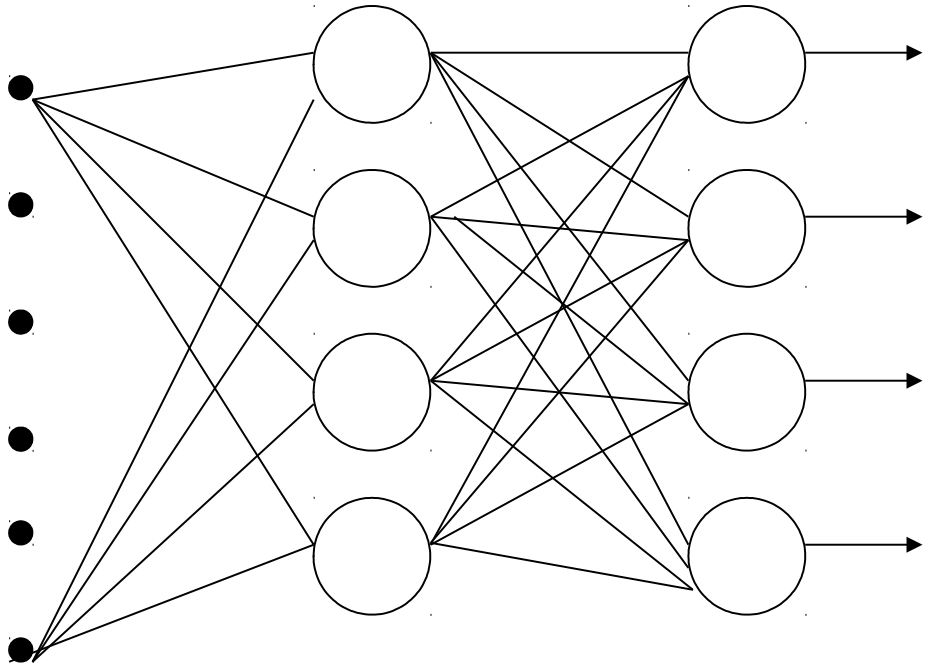
Linealmente separable

Linealmente no separable

x_1	x_2	z
1	1	0
0	1	1
1	0	1
0	0	0

REDES MULTICAPA

En las redes multicapa además de tener una capa de entrada y una de salida se pueden tener capas intermedias llamadas ocultas



BACKPROPAGATION

La backrpopagación o propagación hacia atrás es un método de cálculo de gradiente utilizado para entrenar redes neuronales **multicapa**.

Cuando se aplica un ejemplo de entrenamiento a esta red se calcula cada una de las **salidas**.

Se comparan estas salidas con las esperadas para obtener un **valor de error**. Este error se propaga hacia las capas anteriores calculando el “nivel de responsabilidad” de cada neurona.

ALGORITMO BACKPROPAGATION

1- Inicializar los pesos de manera aleatoria

Deben setearse los pesos para todas las conexiones entre neuronas incluyendo:

Las conexiones entre neuronas de capas ocultas

Las conexiones entre neuronas entre capas ocultas y capa

ALGORITMO BACKPROPAGATION

2- Realizar propagación hacia adelante

Presentar el primer ejemplo del conjunto de entrenamiento

Calcular la salida de cada neurona utilizando la función de activación seleccionada

$$f(x) = \frac{1}{1 + exp(-x)}.$$

BACKPROPAGATION

3- Calcular el error para cada neurona comenzando con la última capa y retrocediendo hasta la primera

Para las capas de salida:

$$\delta_{Mi} = Y_{Mi}(1 - Y_{Mi})(T_{pi} - Y_{Mi}),$$

Derivada de error Esperada - Salida

Para las capas ocultas:

$$\delta_{ji} = Y_{ji}(1 - Y_{ji}) \sum_{k=1}^{N_{j+1}} \delta_{(j+1)k} W_{(j+1)ki}.$$

Derivada de error Error de la neurona de salida –
Peso neurona actual a neurona de salida

BACKPROPAGATION

5- Aplicar los cambios a los pesos

$$W_{jik}^{+} = W_{jik} + \beta \delta_{ji} Y_{ji},$$

Nuevo peso es igual a viejo peso más la tasa de aprendizaje por el error calculado para la neurona y el valor de entrada

BACKPROPAGATION

6- Continuar con el siguiente ejemplo

7- Continuar el proceso hasta que el error sea aceptable

BIBLIOGRAFÍA

- Tom Mitchell. “Machine learning”. McGraw Hill, 1997
- Stuart Russell y Peter Norvig. “Inteligencia Artificial, Un Enfoque Moderno”. 2da edición. Pearson, 2003.