Dpmt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

# Tema 7: Redes Neuronales



1



Dpmt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

# Índice



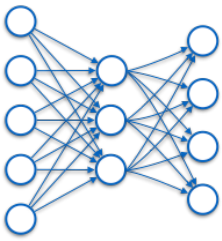


2

# Introducción

**Sistema computacional** inspirado en redes neurales biológicas

Conjunto de **neuronas artificiales** conectadas entre si



Son capaces de **aprender** presentándole una serie de ejemplos

- Los ejemplos deben estar *etiquetados* con la salida esperada
- Se van "*ajustando*" las conexiones entre neuronas







# Áreas de aplicación

Reconocimiento de imagen

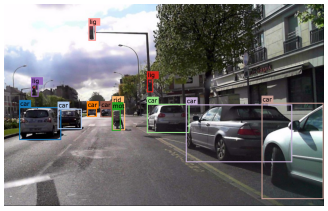
Reconocimiento del habla

Procesamiento del lenguaje natural

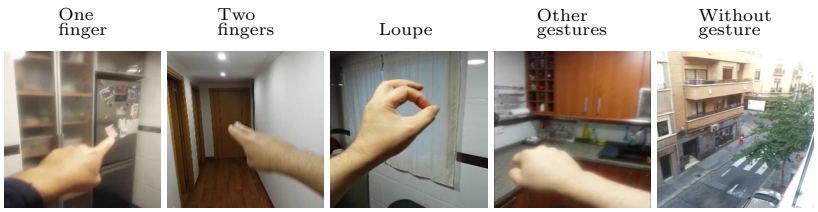
Conducción autónoma

Diagnosis médica

…



**Ejemplo**: Reconocimiento de gestos

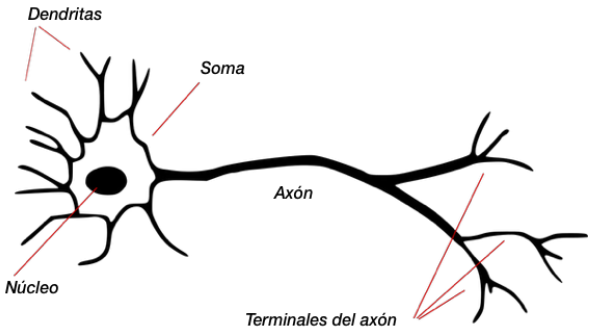


# Neuronas biológicas

**Dentritas**: Reciben entrada (potencial eléctrico) de otras neuronas

**Soma**: Integra las entradas. Es un dispositivo "*todo o nada*", se activa si recibe suficiente potencial de entrada.

**Axón**: Transporta la salida a otras neuronas. La comunicación entre el *axón* de una neurona y las *dendritas* de otra se denomina *sinapsis*.





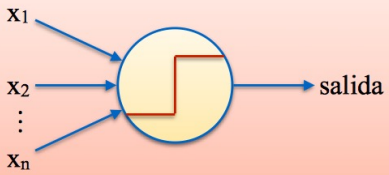




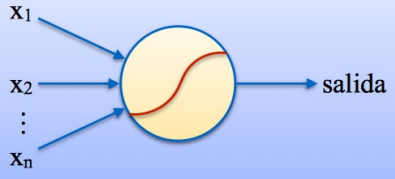



# Neuronas artificiales

## Perceptrón

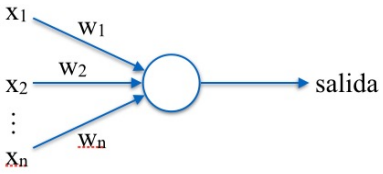



## Neurona *sigmoidea*



# Perceptrón

Neurona artificial que toma una serie de **entradas** $x$ y produce una ***salida***




El perceptrón **toma una decisión** (*salida*) ponderando ($w$) una serie de factores ($x$)

$$\text{salida} = \begin{cases} 0 & \text{si } \sum_j w_j x_j \leq \text{umbral} \\ 1 & \text{si } \sum_j w_j x_j > \text{umbral} \end{cases}$$







# Perceptrón: Notación matricial

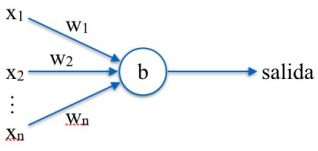
Podemos representar las entradas y pesos **como tuplas**

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

El ***bias*** ($b$) nos indica lo fácil que es que el perceptrón "se dispare" b = − umbral

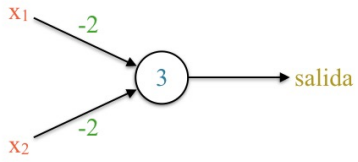
Podemos reescribir la **salida** como

$$\text{salida} = \begin{cases} 0 & \text{si } w \cdot x + b \leq 0 \\ 1 & \text{si } w \cdot x + b > 0 \end{cases}$$



# Ejemplo

Operación **NAND**




$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} -2 \\ -2 \end{bmatrix} + 3$$

| $x_1$ | $x_2$ | $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} -2 \\ -2 \end{bmatrix} + 3$ | Salida |
|---|---|---|---|
| 0 | 0 | 3 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | -1 | 0 |

. Podemos implementar cualquier operación lógica



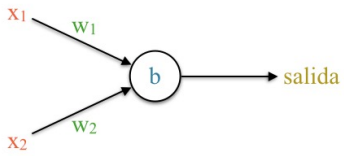


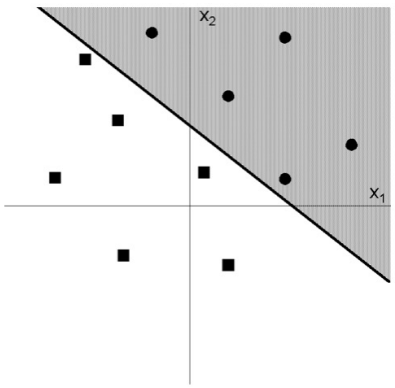




# Interpretación geométrica

- Podemos ver un **perceptrón de dos entradas**, como un clasificador de los puntos que quedan a cada lado de una recta




$$\text{salida} = \begin{cases} 0 & \text{si } w_1x_1 + w_2x_2 + b \leq 0 \\ 1 & \text{si } w_1x_1 + w_2x_2 + b > 0 \end{cases}$$

$$w_1x_1 + w_2x_2 + b = 0$$

Ecuación de la recta

# Interpretación geométrica

En el caso general de un perceptrón de $n$ entradas, los datos se clasifican mediante un **hiperplano** de $n$ dimensiones

$\mathbf{p} = (p_1, p_2, \ldots, p_n)$ — Punto del plano

$\mathbf{w} = (w_1, w_2, \ldots, w_n)$ — Vector normal al plano

$\mathbf{x} = (x_1, x_2, \ldots, x_n)$ — Vector de entrada

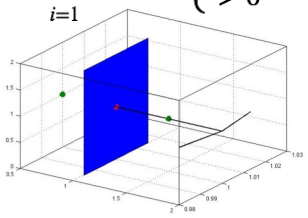
$$\mathbf{w} \cdot (\mathbf{x} - \mathbf{p}) = 0$$ Ecuación del hiperplano

$$\mathbf{w} \cdot \mathbf{x} - \mathbf{w} \cdot \mathbf{p} = 0$$

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \qquad b = -\mathbf{w} \cdot \mathbf{p}$$

Clasificador

$$\sum_{i=1}^{n} w_i x_i + b \begin{cases} < 0 \\ > 0 \end{cases}$$





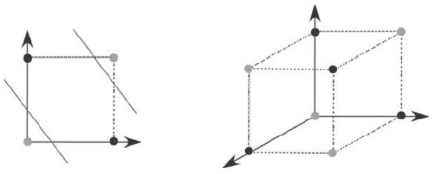




# No-separabilidad lineal

Existen situaciones donde **un único hiperplano** no puede separar los datos

Por ejemplo cuando la frontera de decisión es curva

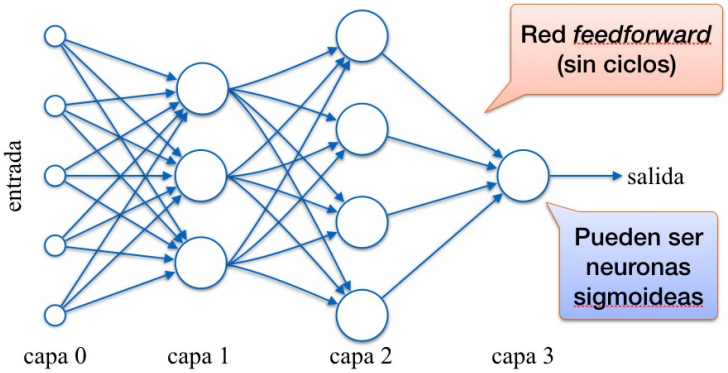


Podemos **combinar varios perceptrones** conectados entre si para implementar funciones más complejas

Dpmt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

# Percepción Multicapa (MLP)

- Organizaremos las neuronas en forma de **capas**
  - La salida de las neuronas de una capa será la entrada de las de la siguiente
  - A mayor **número de capas**, podrá tomar decisiones **más complejas**



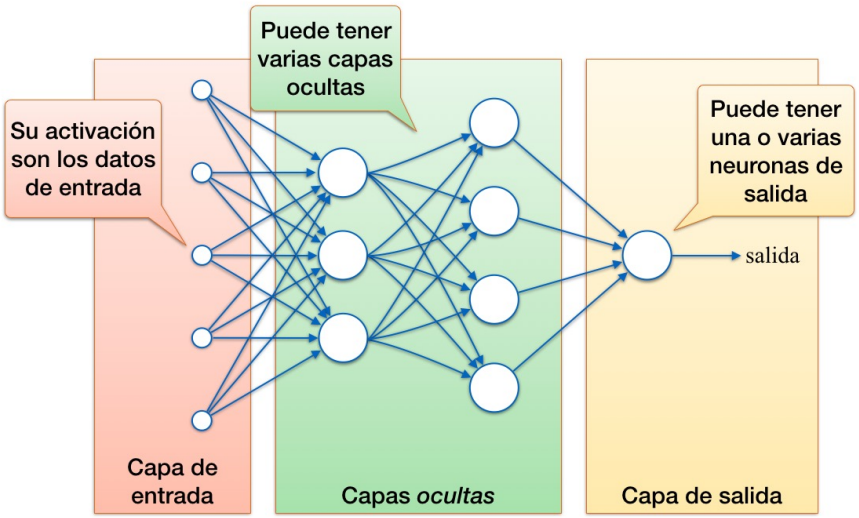







Dpmt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

# Arquitectura de la red



# Tipos de redes

- ***Shallow* Networks**
  - Sólo una capa oculta
- ***Deep* Networks**
  - Dos o más capas ocultas
  - Habitualmente se entrenan redes entre 5 y 10 capas
  - Se entrenan mediante **descenso por gradiente** y ***back propagation***
  - Nuevas técnicas han posibilitado el entrenamiento de estas redes más complejas



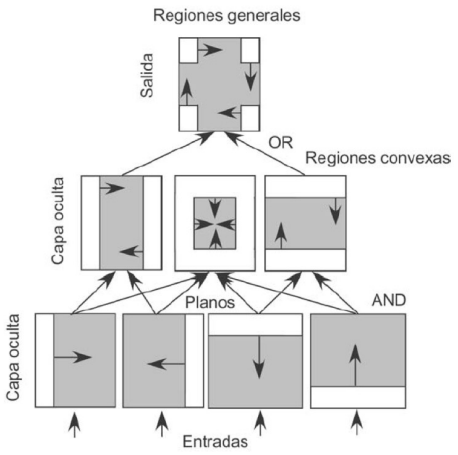




# Interpretación geométrica

Se demuestra que un perceptrón con dos capas puede **aproximar cualquier función**

Las capas ocultas nos permiten incorporar distintas **estrategias de separación**



# Diseño de la red

- El diseño de las capas de entrada y salida suele ser directo

- **Capa de entrada**
  - Tenemos en cuenta cómo podemos descomponer los datos que recibimos como diferentes neuronas de entrada

- **Capa de salida**
  - Tenemos en cuenta cómo podemos codificar el resultado que queremos obtener como salida

- El diseño de las **capas ocultas** no es trivial









# Ejemplo: MNIST

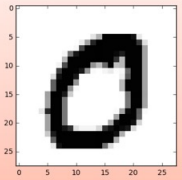
**Base de datos** MNIST (http://yann.lecun.com/exdb/mnist/)

Reconocimiento de **dígitos manuscritos del 0 al 9**

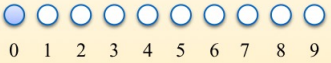
Dpmt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

# Ejemplo: MNIST

- **Capa de entrada**
  - Imágenes de 28x28 píxeles
  - Niveles de gris de [0, 1]
  - ***784 neuronas*** de entrada con valores [0, 1]



- **Capa de salida**
  - Número del 0 al 10
  - ***10 neuronas*** con valores {0, 1}




- **¿Cómo ajustamos los pesos?** **¡Puede ser entrenada!**







Dpmt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

# Entrenamiento

- No le decimos a la máquina **cómo** resolver un problema
  - La máquina **lo aprende** a partir de observar ejemplos
- Necesitamos entrenar la red con un **gran número de ejemplos** para ajustar los pesos que produzcan la salida deseada
- Dividiremos la base de datos en **dos conjuntos**:

- **Conjunto de entrenamiento (*training*)**
  - Ejemplos con los que ajustamos los pesos de la red

- **Conjunto de prueba (*test*)**
  - Ejemplos para validar los resultados se clasificación de la red

# Ejemplos de la base de datos

Para cada ejemplo tendremos una **entrada** $x$ y una **salida esperada** $y(x)$

Por ejemplo, en el caso del reconocimiento de dígitos la salida esperada para diferentes ejemplos podría ser:

| $x$ | $y(x)$ |
|---|---|
|  | $[1,0,0,0,0,0,0,0,0,0]^T$ |
|  | $[0,1,0,0,0,0,0,0,0,0]^T$ |
|  | $[0,0,0,0,0,1,0,0,0,0]^T$ |



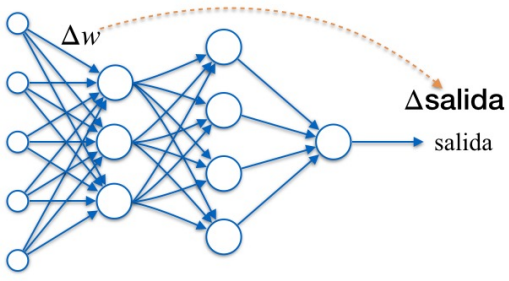


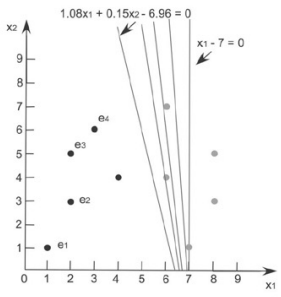




# ¿Cómo entrenamos la red?

- **Idea**: Supongamos que una pequeña modificación en un peso provoca una pequeña modificación de la salida



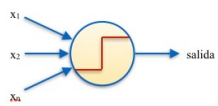




  - Podemos ir **modificando los pesos y biases** de forma que nos acerque a la salida deseada
- **Problema**: La salida del perceptron no se modifica poco a poco, es un escalón

# Neurona *sigmoidea*

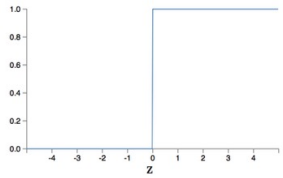
- **Definimos**: $z = w \cdot x + b$ — Entrada ponderada de la neurona

**Perceptrón**

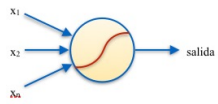



*Función de activación*

$$f(z) = \begin{cases} 0 & \text{si } z \le 0 \\ 1 & \text{si } z > 0 \end{cases}$$

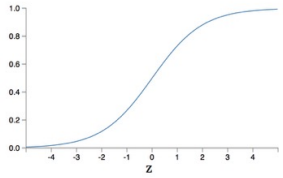



**Neurona *sigmoidea***




*Función de activación*

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$












# Propiedades de la neurona *sigmoidea*

Con valores de $z$ **de gran magnitud** equivale a la función escalón

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$z \ll 0 : \lim_{z \to -\infty} \sigma(z) = 0$$

$$z \gg 0 : \lim_{z \to \infty} \sigma(z) = 1$$

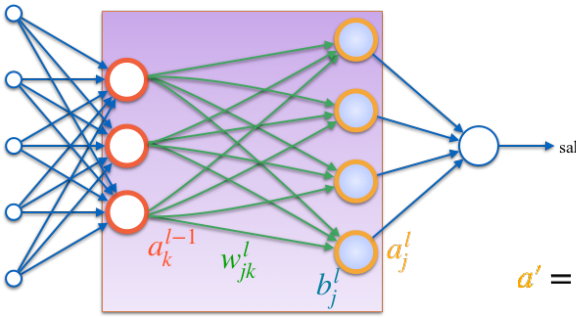
Podemos ver la función *sigmoidea* como un ***escalón suavizado***

**Pequeños cambios** de los pesos $\Delta w$ y del *bias* $\Delta b$ producen pequeños cambios en la salida

$$\Delta \text{salida} \approx \sum_j \frac{\partial \text{salida}}{\partial w_j} \Delta w_j + \frac{\partial \text{salida}}{\partial b} \Delta b$$

Dpmt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

# Cálculo de la activación en cada capa




$$a' = \sigma(Wa + b)$$

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \quad a' = \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \\ a'_4 \end{bmatrix}$$

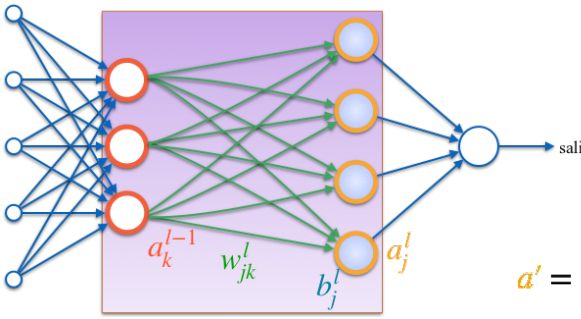






Dpmt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

# Cálculo de la activación en cada capa




$$a' = \sigma(Wa + b)$$

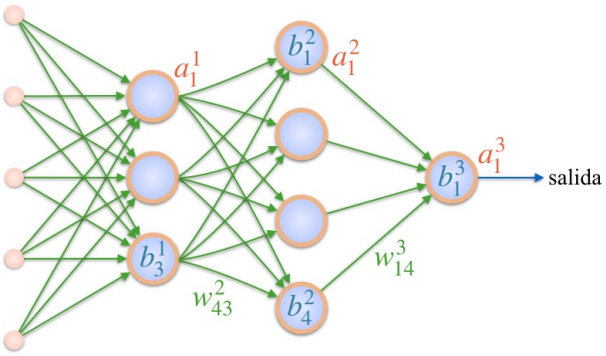
$$\begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \\ a'_4 \end{bmatrix} = \sigma\left( \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \right)$$

# Notación de la red

$w_{jk}^l$ — Peso entre la neurona $k$ de la capa $(l-1)$ y la neurona $j$ de la capa $l$

$b_j^l$ — *Bias* de la neurona $j$ de la capa $l$

$a_j^l$ — Activación de la neurona $j$ de la capa $l$












# Cálculo de la activación

**Entrada**:

Datos de entrada: $x$

Matrices de pesos de cada capa: $w = [W^1, W^2, \ldots, W^L]$

Tuplas de *biases* de cada capa: $b = [b^1, b^2, \ldots b^L]$

**Algoritmo FeedForward**

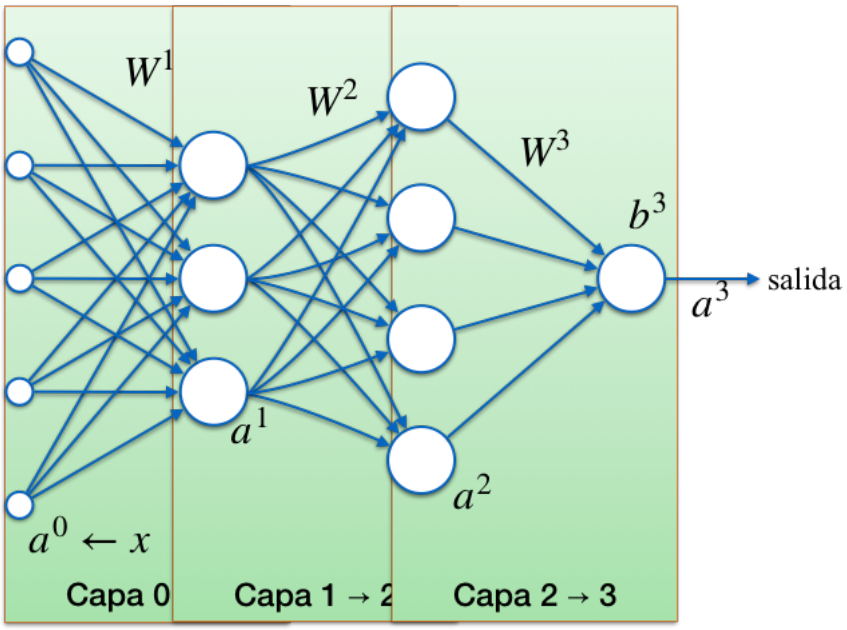
$a \leftarrow x$

**Para cada capa** $l \in [1,\ldots,L]$

$a \leftarrow \sigma(W^l a + b^l)$

**Devolver** $a$

# Cálculo de la activación










# Función de coste

Debemos buscar los **pesos** $w$ y **biases** $b$ de toda la red que produzcan las salidas deseadas.

Dado:

- $a$: **salida real** de la red para la entrada $x$, pesos $w$ y biases $b$
- $y(x)$: **salida esperada** de la red para la entrada $x$
- $n$: **Número total de ejemplos** de entrenamiento (*training*)

Podemos definir una **función de coste** $C$ que evalúe el error cuadrático medio (MSE) de clasificación de la red:

$$C(w,b) = \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

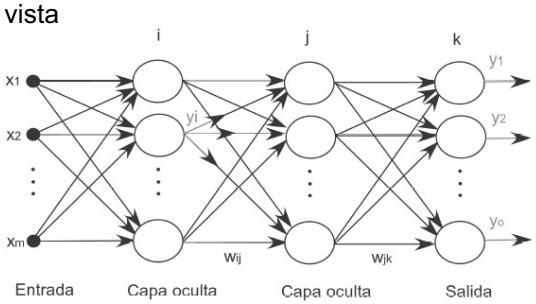
Debemos minimizar su valor

# *Backpropagation*: explicación heurística

- Supongamos que al clasificar un ejemplo una neurona de la última capa tiene una salida $\mathbf{y_k}$, siendo la deseada $\mathbf{d_k}$
- Dicha neurona es responsable de un error

$$\delta_k = (d_k - y_k) f'(net_k),$$

- La regla de actualización de los pesos de la última capa será similar a la regla delta ya vista







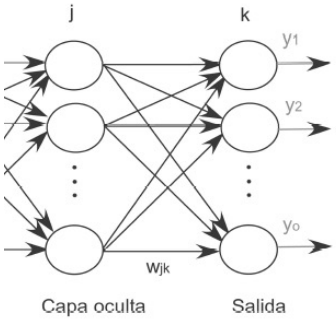


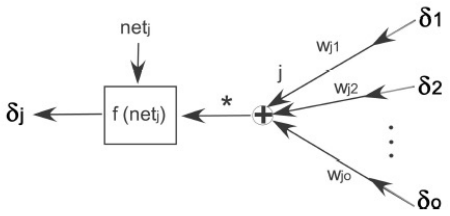



# Error (delta) en capas intermedias

- Una neurona de una capa intermedia contribuye en los δ de las de la capa siguiente
- Por tanto, para calcular su δ necesitamos estos

$$\delta_j = f'(net_j) \sum_k w_{jk} \delta_k$$



# *Backpropagation*: algoritmo

- Se aplica para cada ejemplo del conj. de entrenamiento. Se itera hasta que el error baje de un umbral
- Fases:
  - Hacia delante: cálculo de la salida de la red (los $y_k$). Cálculo de los δ en la última capa
  - Hacia atrás. Cálculo de los δ de la capa en función de los de la siguiente
  - Finalmente, actualización de los pesos de todas las capas

**Algoritmo** BACKPROPAGATION($red\ ejemplos, \eta$) {
  $\{w_{ij}\} \leftarrow$ INICIALIZAR;
  **Mientras** ¬ CONVERGENCIA($red$) **Hacer** {
    $e \leftarrow$ SELECCIONAREJEMPLO($ejemplos$);
    $\{y_k\} \leftarrow$ FORWARD($e$);
    $\{d_k\} \leftarrow$ DESEADAS($e$);
    **Para cada** $n_k \in$ CAPA($red, k$) **Hacer** {
      $\delta_k = (d_k - y_k) f'(net_k)$;
    }
    **Para** $j = k - 1$ **hasta** 1 **Hacer** {
      **Para** $n_j \in$ CAPA($red, j$) **Hacer** {
        $\delta_j = f'(net_j) \sum_{j+1} \delta_{k+1} w_{j(j+1)}$;
      }
    }
    **Para** $j = k$ **hasta** 1 **Hacer** {
      $w_{(j-1)j} = w_{(j-1)j} + \eta \delta_j y_{(j-1)}$;
    }
    $red \leftarrow$ ACTUALIZARRED($\{w_{ij}\}$);
  }
  **Devolver** $red$;
}









# Bibliografía

Escolano et al. Inteligencia Artificial. Thomson-Paraninfo 2003. Capítulo 4.

Mitchell, Machine Learning. McGraw Hill, Computer Science Series. 1997

Reed, Marks, Neural Smithing. MIT Press, CA Mass 1999

Neural Networks and Deep Learning. Libro digital:

http://neuralnetworksanddeeplearning.com/index.html

Neural Networks. Canal de YouTube 3blue1brown

https://youtu.be/aircAruvnKk