

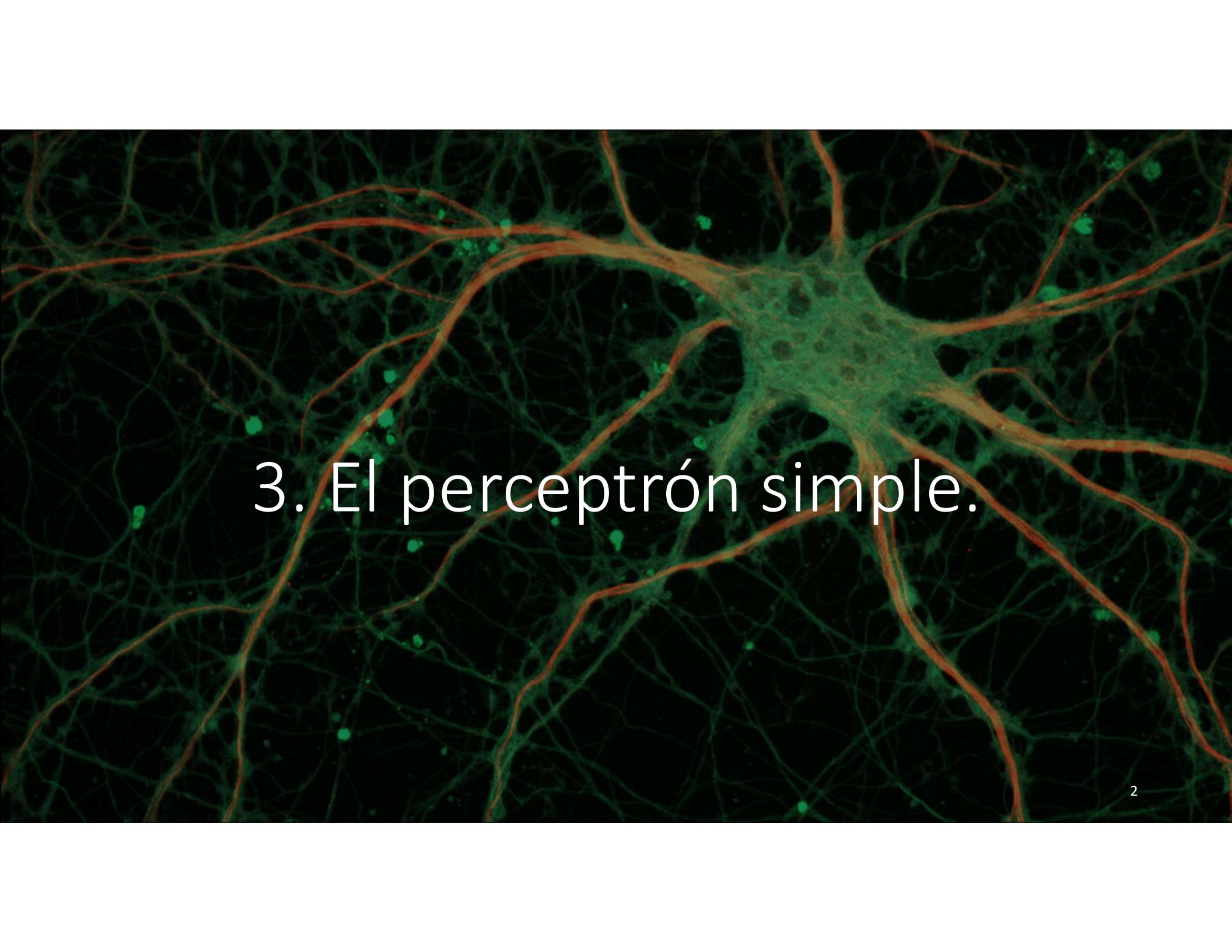


# Redes Neuronales (y Bayesianas)

## LDS1081

Juan Manuel Ahuactzin Larios  
juan.ahuactzin@udlap.mx

Todas las imágenes de este curso fueron obtenidas de Pexels, Pxhere y Microsoft Powerpoint:  
<https://www.pexels.com/> <https://pxhere.com/>

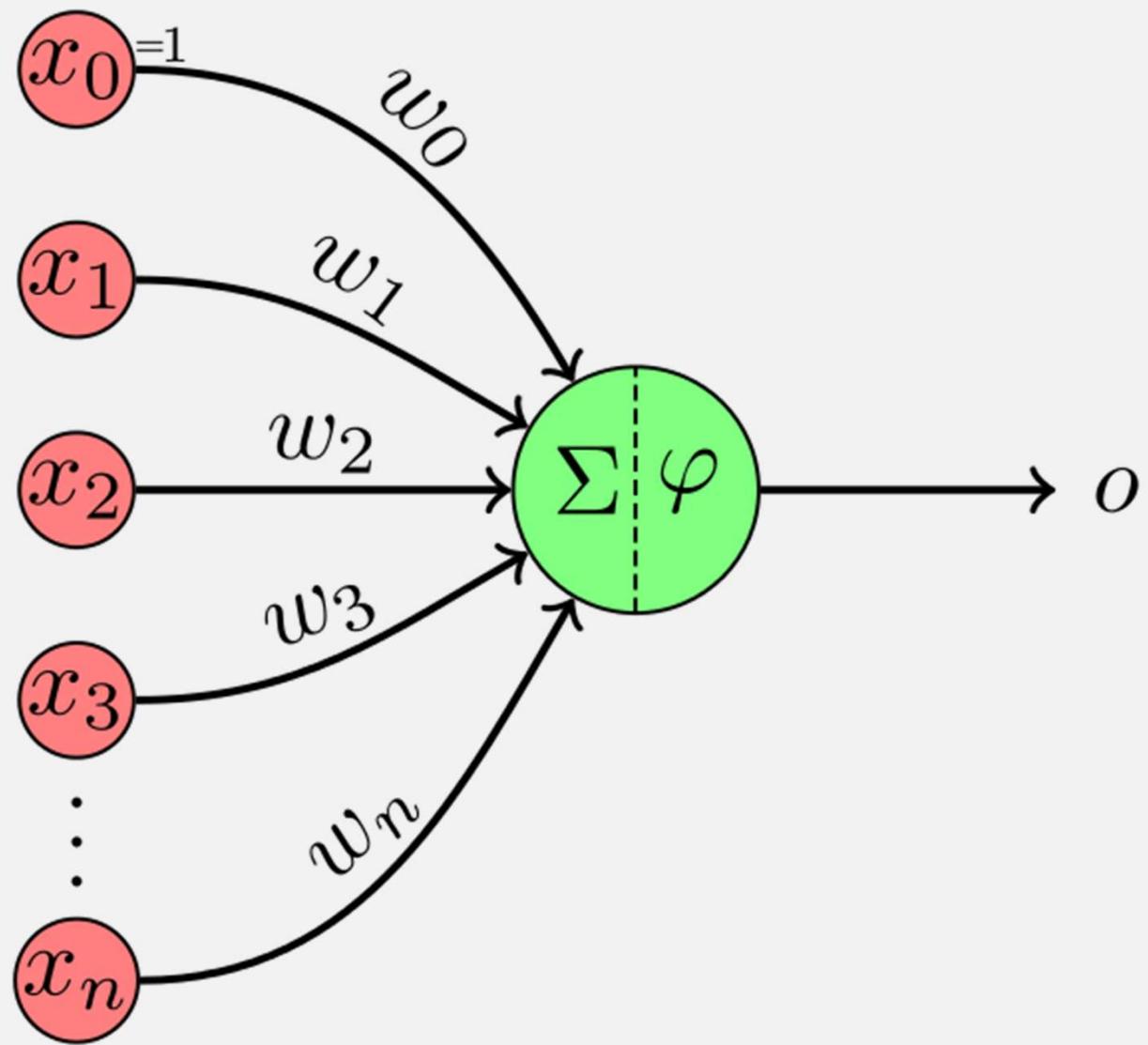


### 3. El perceptrón simple.

A close-up photograph of a person's hands holding a professional video camera, likely a Canon EOS model, with a large lens. The person is wearing a striped shirt. The background is blurred, showing some greenery and possibly a tripod or other equipment.

# INICIA GRABACIÓN DE VIDEO

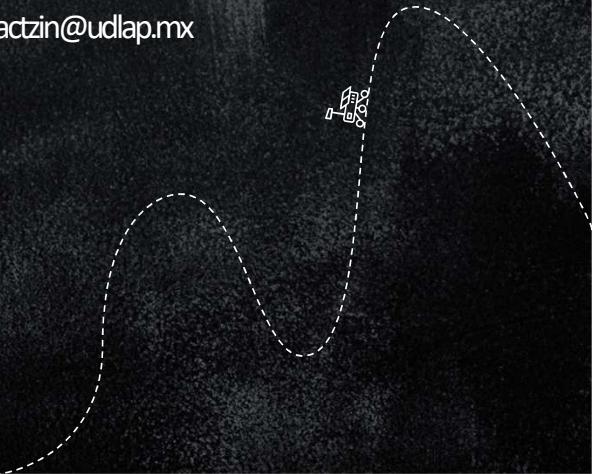
DR. JUAN MANUEL AHUACTZIN LARIOS



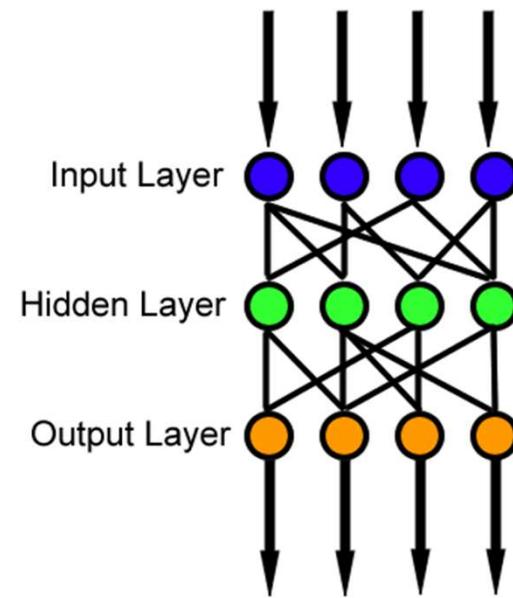
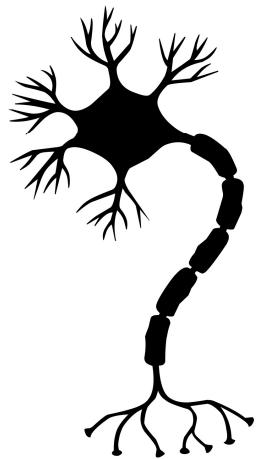
## 3.1 Estructura de un perceptrón.

Juan Manuel Ahuactzin Larios

[juan.ahuactzin@udlap.mx](mailto:juan.ahuactzin@udlap.mx)



# Principios



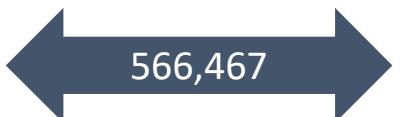


Caracor: 0.047 km/h



Blackbird: 3,529.6 km/h

1974: Procesador 4040



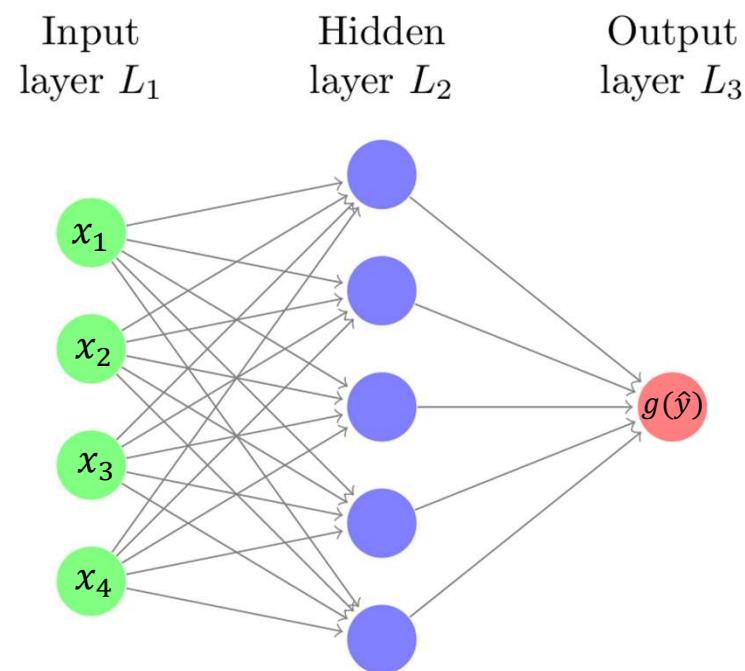
0.062 MIPS

2020: Procesador AMD Ryzen 5 3600X

35,121 MIPS

7.54 Veces aún más rápido que el blackbird

# Principios



# Principios

A partir de

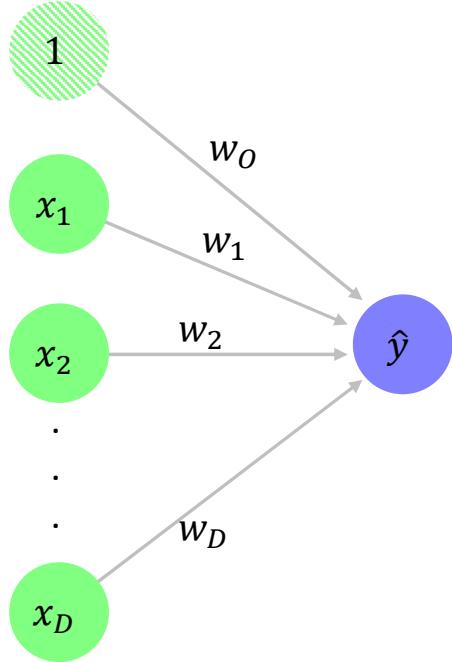
$$\hat{y}(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_D x_D$$

Geralizamos para todos los datos

$$\mathbf{y} = \mathbf{x}^T \mathbf{w} + \boldsymbol{\varepsilon}, \quad \hat{\mathbf{y}} = \mathbf{x}^T \mathbf{w}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,D} \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & \cdots & x_{N,D} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_N \end{bmatrix}, \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_N \end{bmatrix}$$

# Principios

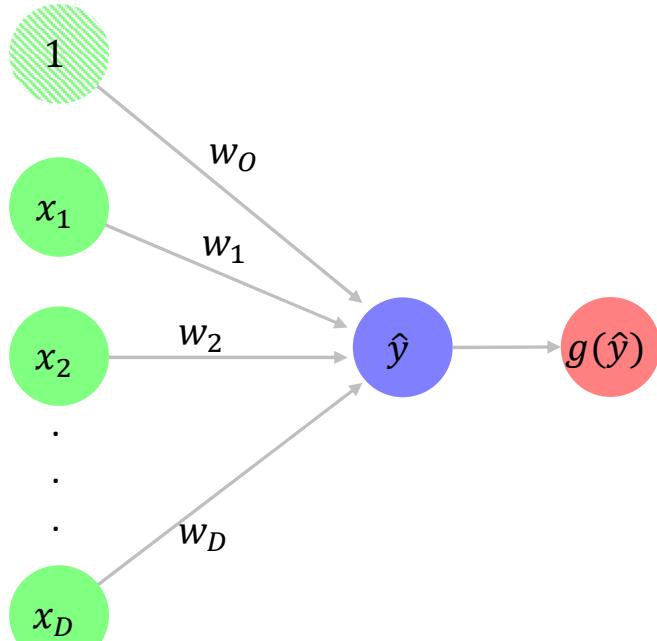


$$\hat{y}(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_D x_D$$

$$\mathbf{x} = [1, x_1, x_2, \dots, x_D], \quad \mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_N \end{bmatrix}$$

$$\hat{\mathbf{y}} = \mathbf{x}^T \mathbf{w}$$

# Principios



$$\hat{y}(\mathbf{x}, \mathbf{w}) = w_0 + w_1x_1 + \cdots + w_Dx_D$$

$$\mathbf{x} = [1, x_1, x_2, \dots, x_D], \quad \mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_N \end{bmatrix}$$

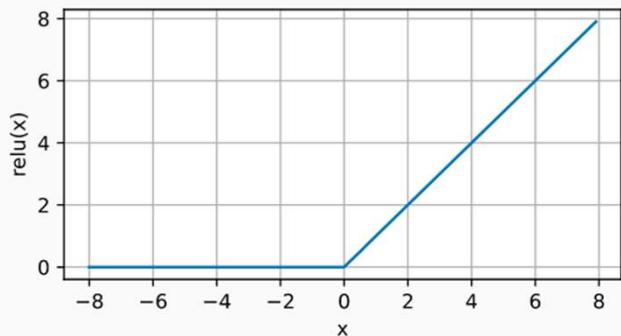
$$\hat{\mathbf{y}} = \mathbf{x}^T \mathbf{w}$$

$$g(\hat{y}) = g(\mathbf{x}^T \mathbf{w})$$

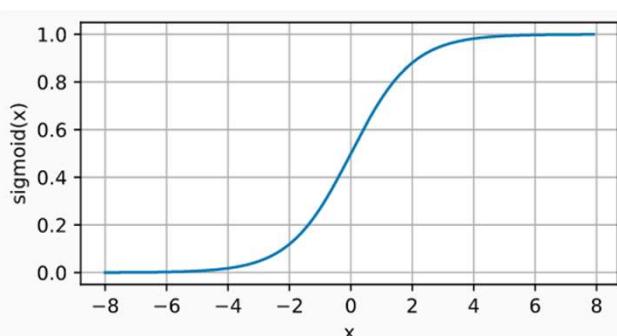
$g(\hat{y}) = \hat{y}$  (la función identidad) para problemas de regresión.

# Principios

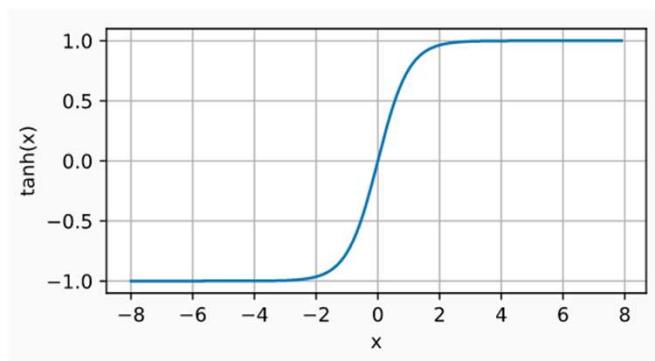
## Funciones de activación



$$\text{ReLU}(x) = \max(0, x)$$



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



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

# Principios

**Función de coste para regresión (*Loss function*)**

**Error cuadrático medio** (*Root Mean Square Error, RMSE*)

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}}$$

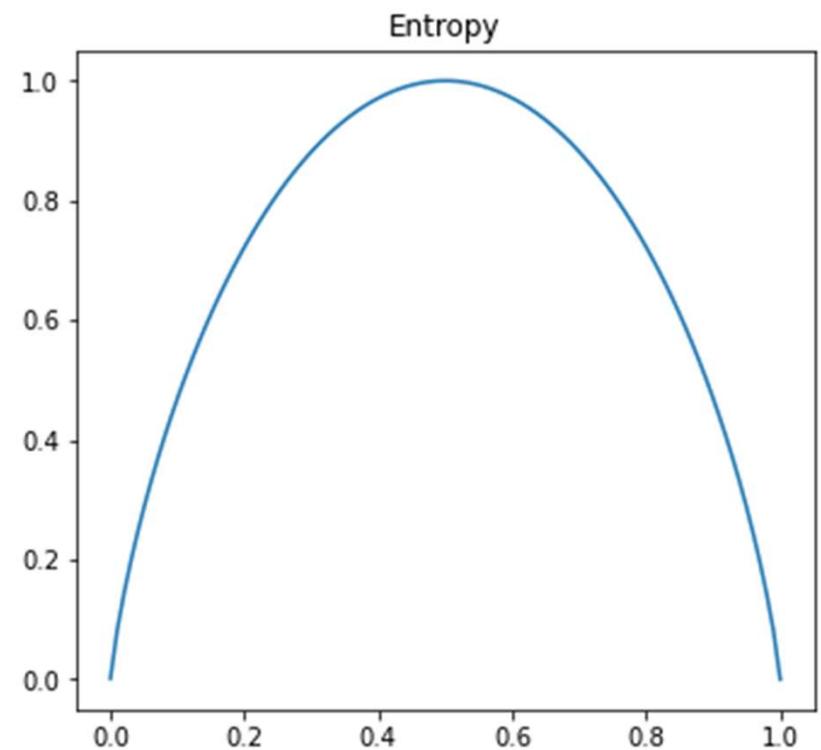
**Error medio absoluto** (*Mean absolute error, MAE* )

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

# Principios

## Entropía

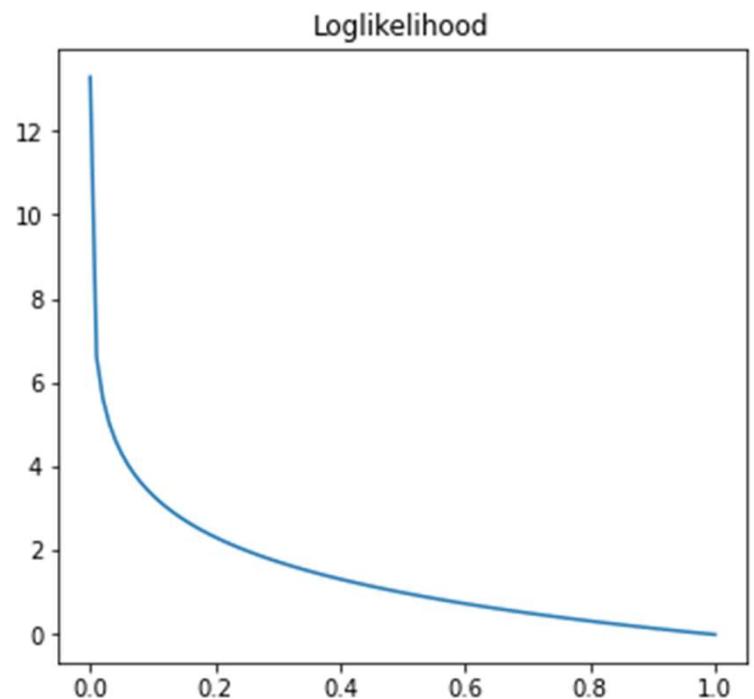
$$H(X) = - \sum_{i=1}^n P(x_i) \log(P(x_i))$$



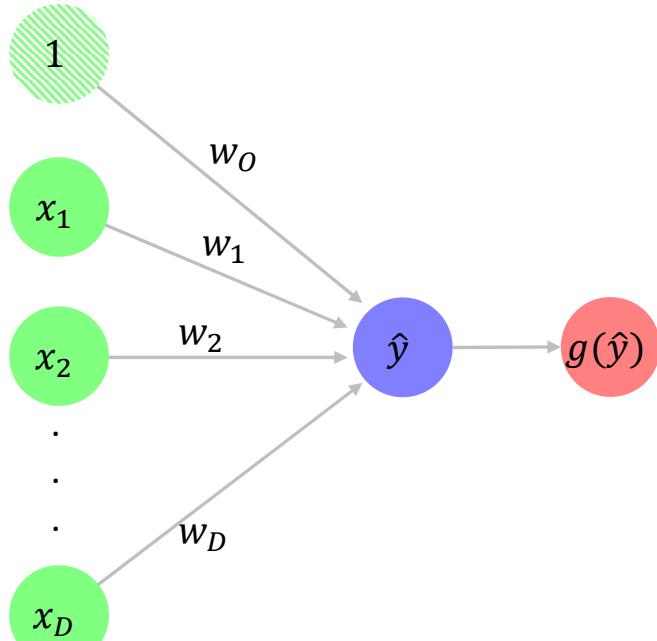
# Principios

**Función de coste para clasificación  
(Loss function)**

$$\begin{aligned}L_{log}(y, p) &= -\log(P(y|p)) \\&= -(y \log(p) + (1 - y) \log(1 - p))\end{aligned}$$



# Principios



$$\hat{y}(\mathbf{x}, \mathbf{w}) = w_0 + w_1x_1 + \cdots + w_Dx_D$$

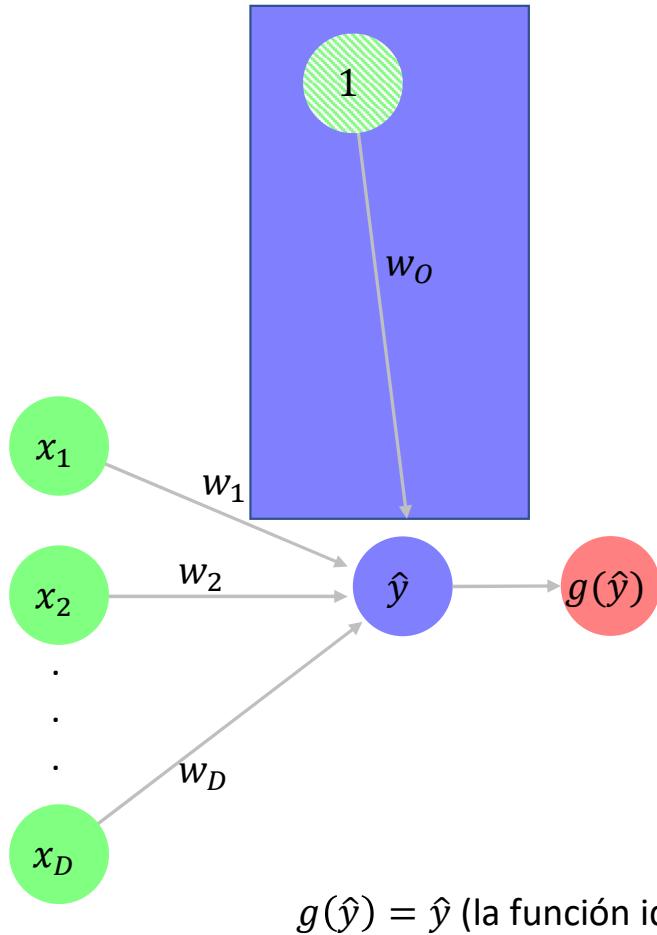
$$\mathbf{x} = [1, x_1, x_2, \dots, x_D], \quad \mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_N \end{bmatrix}$$

$$\hat{\mathbf{y}} = \mathbf{x}^T \mathbf{w}$$

$$g(\hat{y}) = g(\mathbf{x}^T \mathbf{w})$$

$g(\hat{y}) = \hat{y}$  (la función identidad) para problemas de regresión.

# Principios



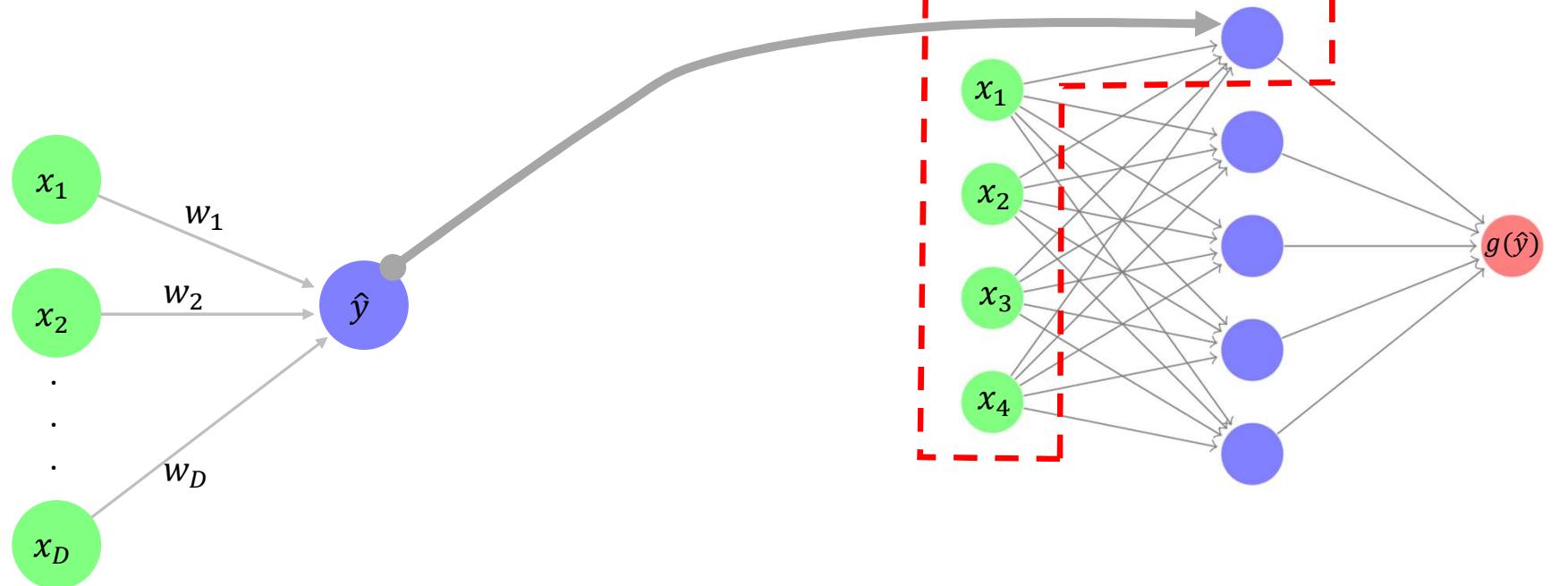
$$\hat{y}(\mathbf{x}, \mathbf{w}) = w_0 + w_1x_1 + \cdots + w_Dx_D$$

$$\mathbf{x} = [1, x_1, x_2, \dots, x_D], \quad \mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_N \end{bmatrix}$$

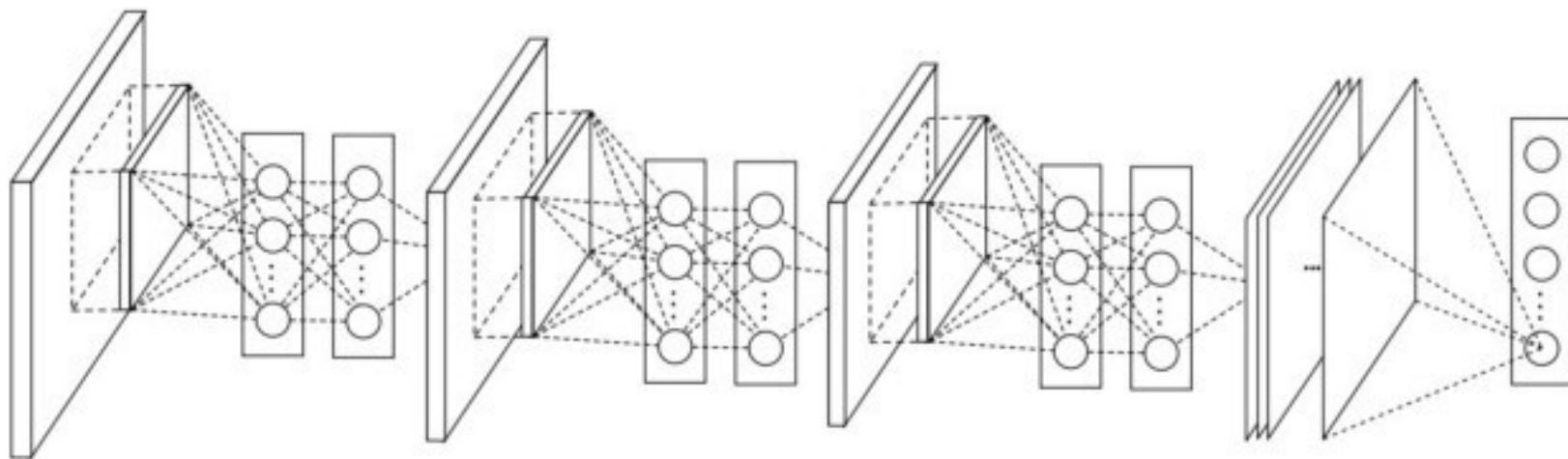
$$\hat{\mathbf{y}} = \mathbf{x}^T \mathbf{w}$$

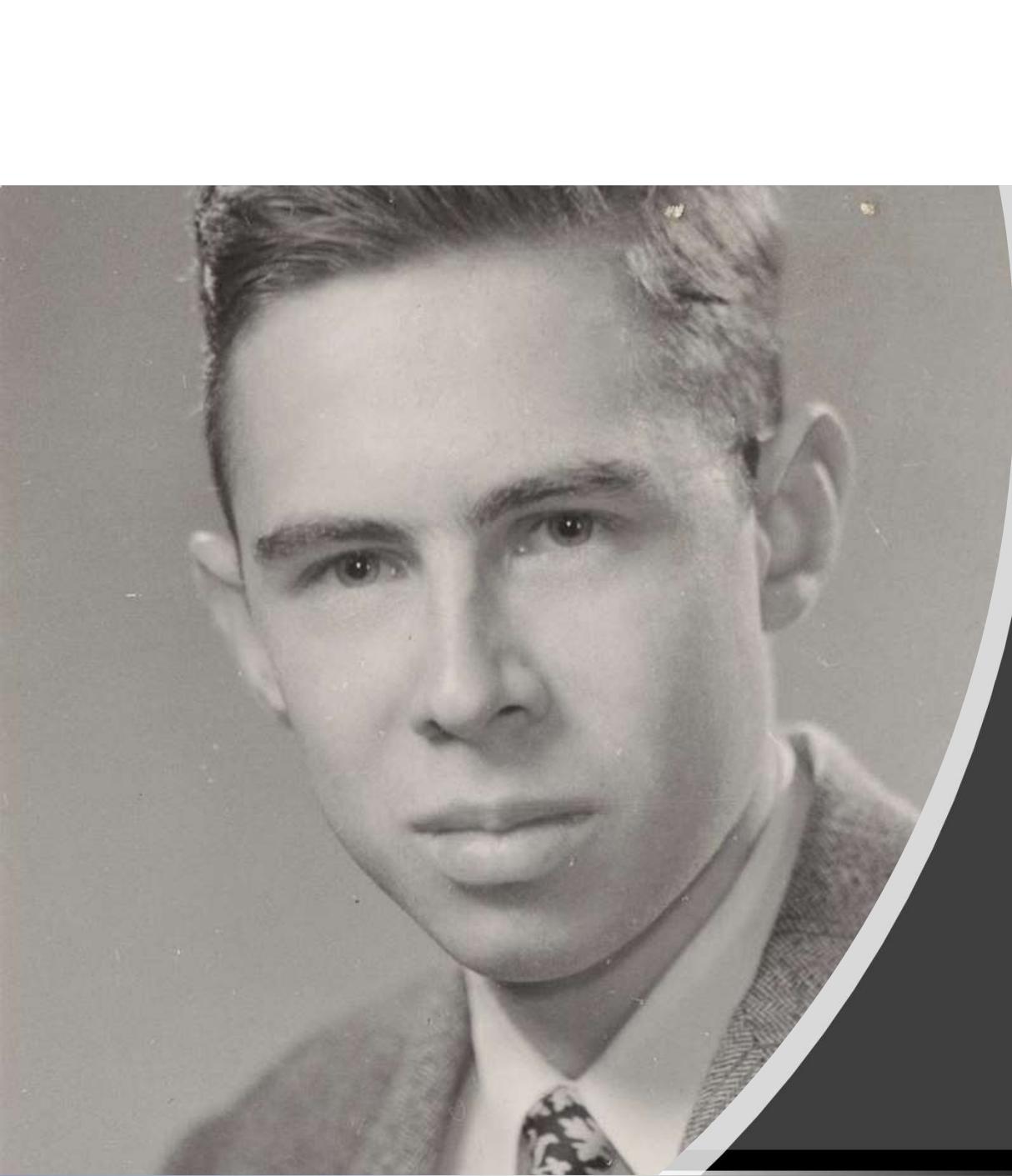
$$g(\hat{y}) = g(\mathbf{x}^T \mathbf{w})$$

# Principios



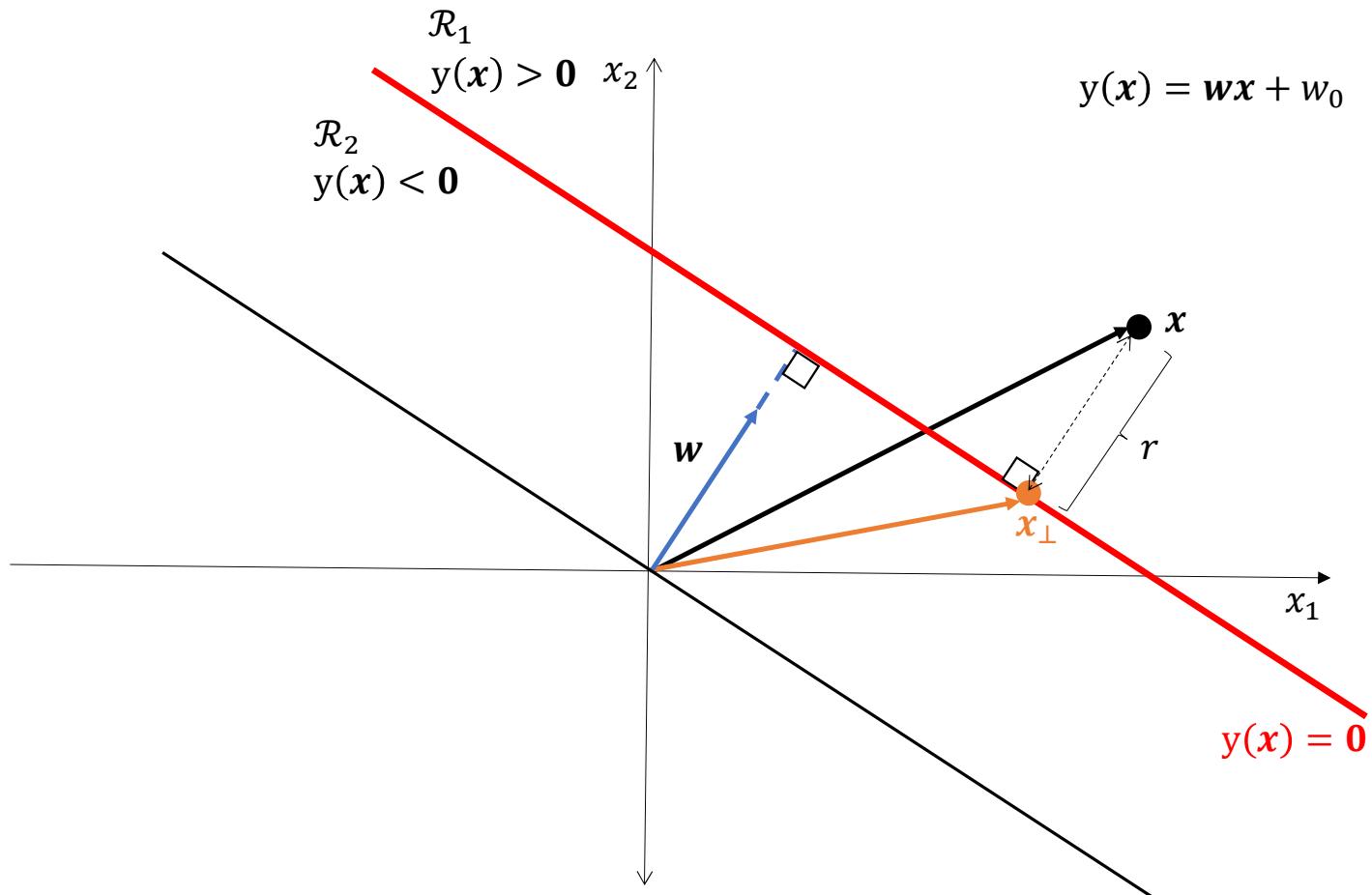
# Principios



A black and white portrait photograph of Frank Rosenblatt, a man with short, light-colored hair, wearing a suit jacket, white shirt, and patterned tie. He is looking slightly to his left with a neutral expression.

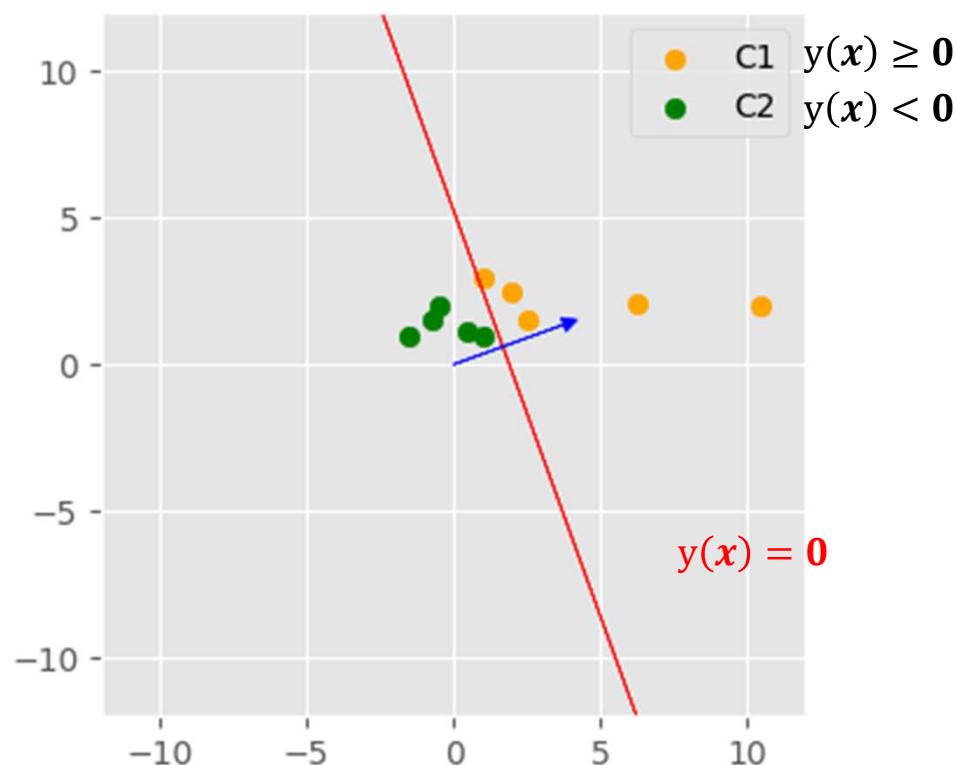
# Perceptrón de Rosenblatt (1962)

# Recordatorio



20

Iteration:21



## Las etiquetas

Para si un elemento  $x_i$  pertenece a la clase C1, entonces su etiqueta  $t_i = +1$  de otra forma, si pertenece a C2, tenemos que  $t_i = -1$

De esta forma, podemos definir una fusión de activación  $f(y)$  de la siguiente forma:

$$f(y) = \begin{cases} +1 & \text{si } y \geq 0 \\ -1 & \text{si } y < 0 \end{cases}$$

# Función de error

Para un vector de pesos  $w$  podríamos utilizar el número total de elementos mal clasificados sin embargo esto no es una buena idea ya que cambiar  $w$  al moverse usando el gradiente de la función de error no puede ser aplicado, en efecto, el gradiente será 0 casi en cualquier parte.

# Un elemento mal clasificado

Un elemento estará mal clasificado si  $t_i f(y(x_i)) < 0$

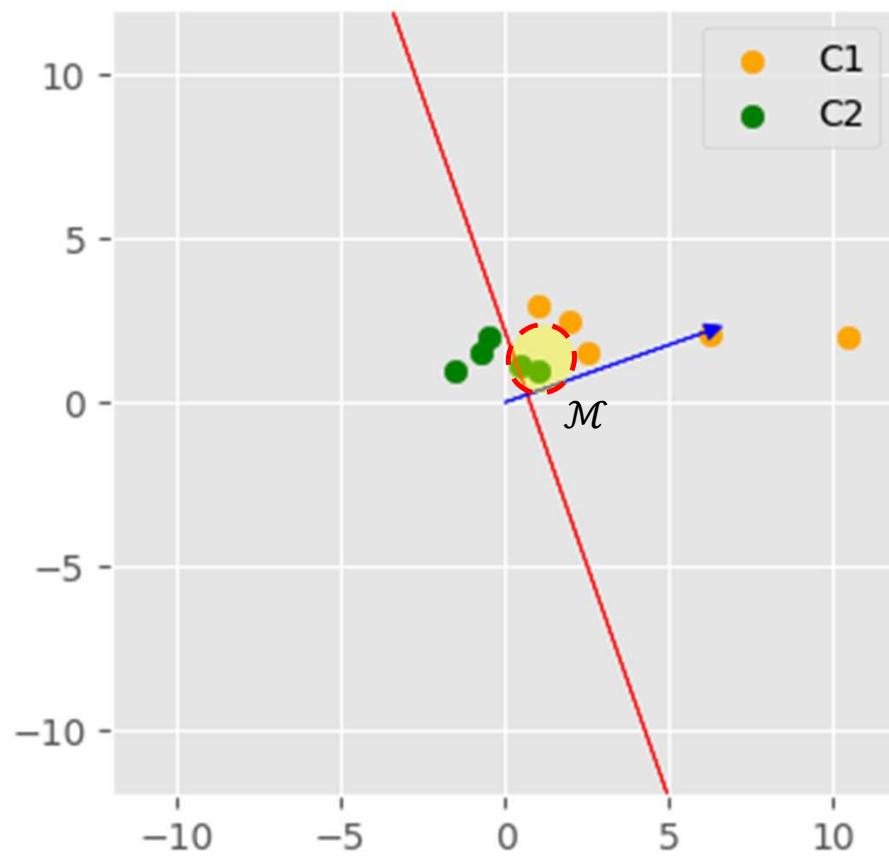
De esta forma el criterio de un perceptrón (*perceptron criterion*) está dado por:

$$E_p(\mathbf{w}) = - \sum_{i \in \mathcal{M}} t_i f(y(x_i))$$

Donde  $\mathcal{M}$  denota el conjunto de puntos mal clasificados.

# Un elemento mal clasificado

Iteration:16



## Alternativa a la función de error

Una alternativa a la función de error es el criterio del perceptrón es encontrar un vector  $\mathbf{w}$  tal que:

$$E_p(\mathbf{w}) = - \sum_{i \in \mathcal{M}} t_i y(x_i) = 0$$

Para hacerlo podemos utilizar un algoritmo de descenso de gradiente estocástico con esta función de error:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^k - \alpha \nabla E_p(\mathbf{w}_i)$$

# Función de error

Tenemos que  $y(\mathbf{x}) = \mathbf{w}\mathbf{x} + w_0$  por lo tanto

$$E_p(\mathbf{w}) = -\sum_{i \in \mathcal{M}} t_i y(x_i) = -\sum_{i \in \mathcal{M}} t_i (\mathbf{w}\mathbf{x} + w_0)$$

# Función de error

Si  $E_p(\mathbf{w}) = -\sum_{i \in \mathcal{M}} t_i(\mathbf{w}\mathbf{x} + w_0)$

Y expresamos

$$\mathbf{w}^k = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \text{ tenemos que:}$$

$$\mathbf{w}^{(k+1)} = \begin{bmatrix} w_0 - \frac{\delta E_p(\mathbf{w})}{\delta w_0} = w_0 + \alpha \frac{\delta \sum_{i \in \mathcal{M}} t_i(\mathbf{w}\mathbf{x} + w_0)}{\delta w_0} = w_0 + \alpha \sum_{i \in \mathcal{M}} t_i \\ w_1 - \frac{\delta E_p(\mathbf{w})}{\delta w_1} = w_1 + \alpha \frac{\delta \sum_{i \in \mathcal{M}} t_i(\mathbf{w}\mathbf{x} + w_0)}{\delta w_1} = w_1 + \alpha \sum_{i \in \mathcal{M}} t_i x_{i,1} \\ w_2 - \frac{\delta E_p(\mathbf{w})}{\delta w_2} = w_2 + \alpha \frac{\delta \sum_{i \in \mathcal{M}} t_i(\mathbf{w}\mathbf{x} + w_0)}{\delta w_2} = w_2 + \alpha \sum_{i \in \mathcal{M}} t_i x_{i,2} \end{bmatrix}$$

# Función de error

Esto quiere decir que actualizamos el vector de pesos  $\mathbf{w}$  de la siguiente manera:

$$w_0 = w_0 + \alpha \sum_{i \in \mathcal{M}} t_i$$

$$w_1 = w_1 + \alpha \sum_{i \in \mathcal{M}} t_i x_{i,1}$$

$$w_2 = w_2 + \alpha \sum_{i \in \mathcal{M}} t_i x_{i,2}$$

# El algoritmo

El algoritmo queda como sigue:

Para un número de épocas  $E$  hacemos:

Para cada entrada  $\mathbf{x}_i$  evaluamos  $y(\mathbf{x}_i)$  si  $\mathbf{x}_i$  está bien clasificado dejamos el vector de pesos  $\mathbf{w}$  como está. De otra forma lo actualizamos de la forma siguiente:

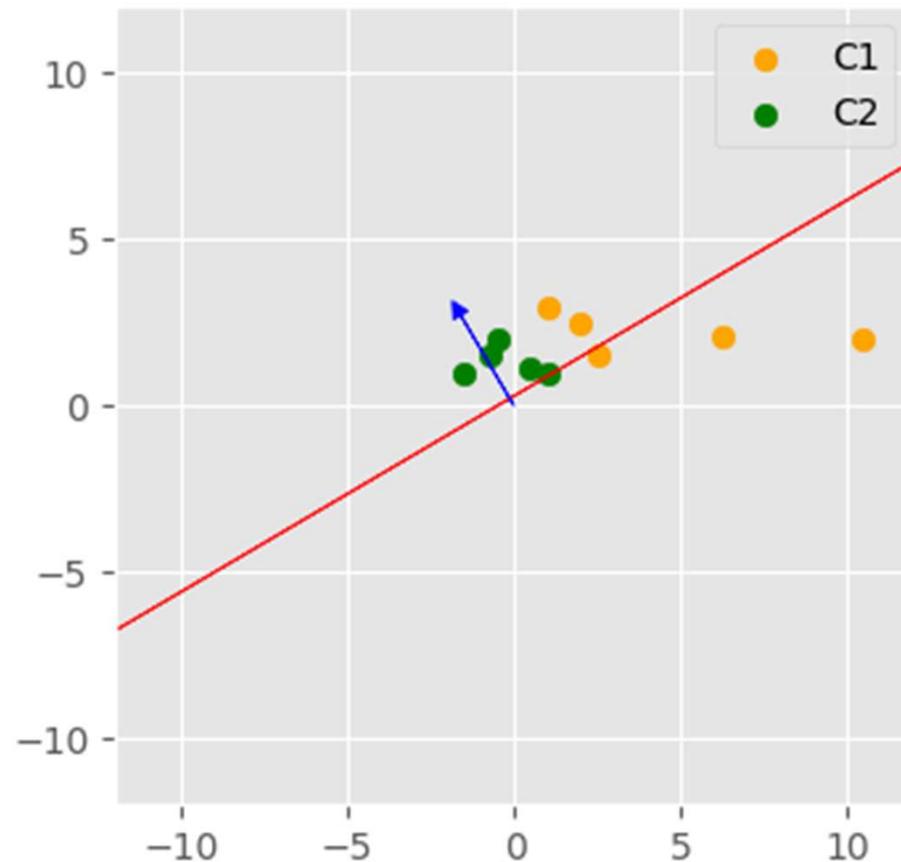
$$w_0 = w_0 + \alpha \sum_{i \in \mathcal{M}} t_i$$

$$w_1 = w_1 + \alpha \sum_{i \in \mathcal{M}} t_i x_{i,1}$$

$$w_2 = w_2 + \alpha \sum_{i \in \mathcal{M}} t_i x_{i,2}$$

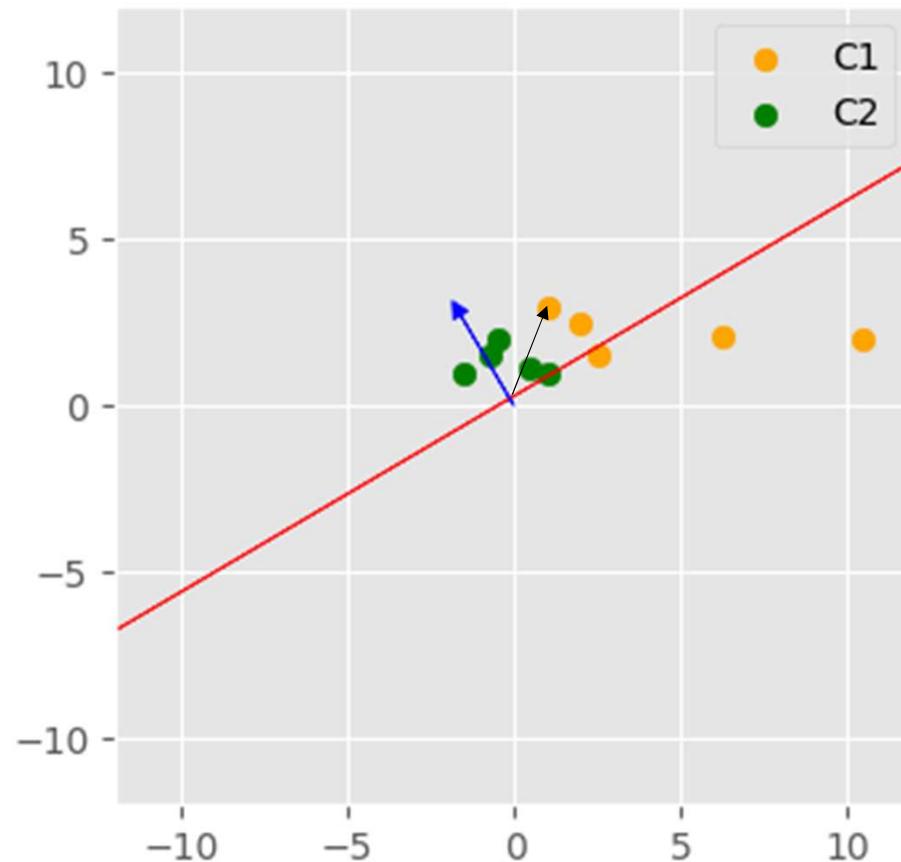
# El algoritmo

Iteration:5



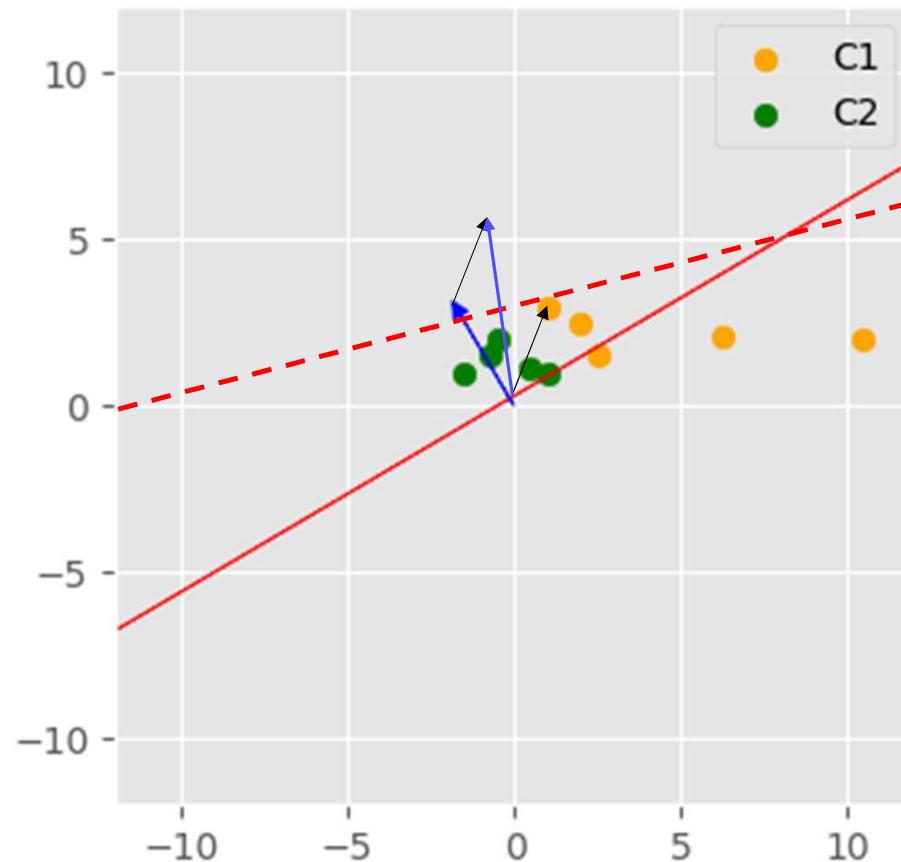
# El algoritmo

Iteration:5



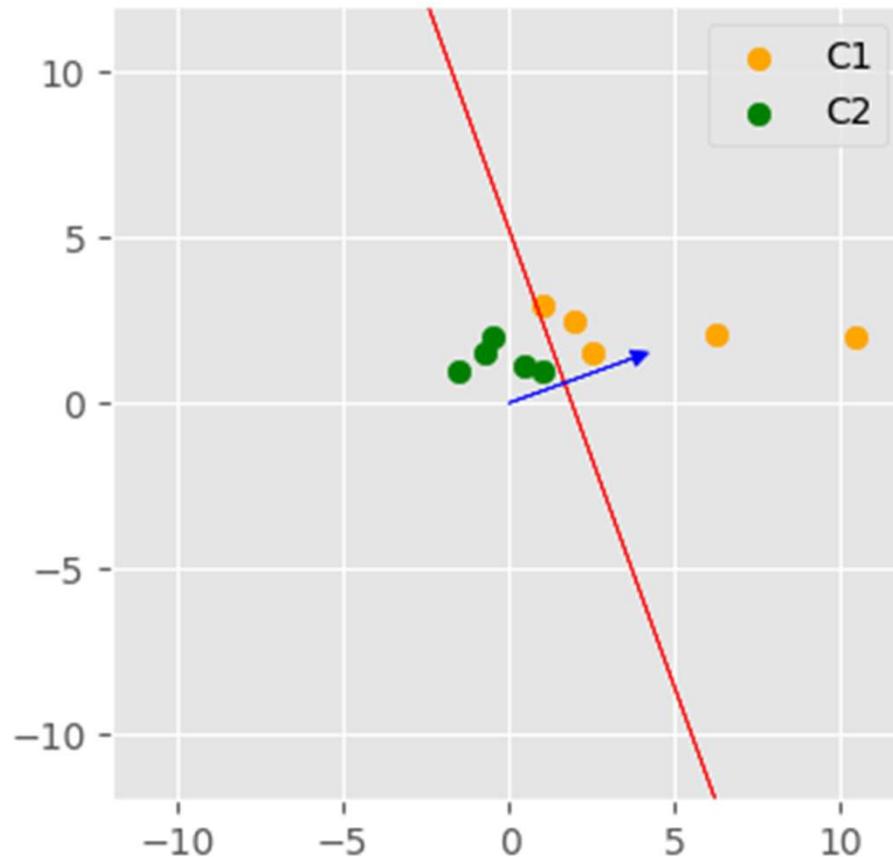
# El algoritmo

Iteration:5



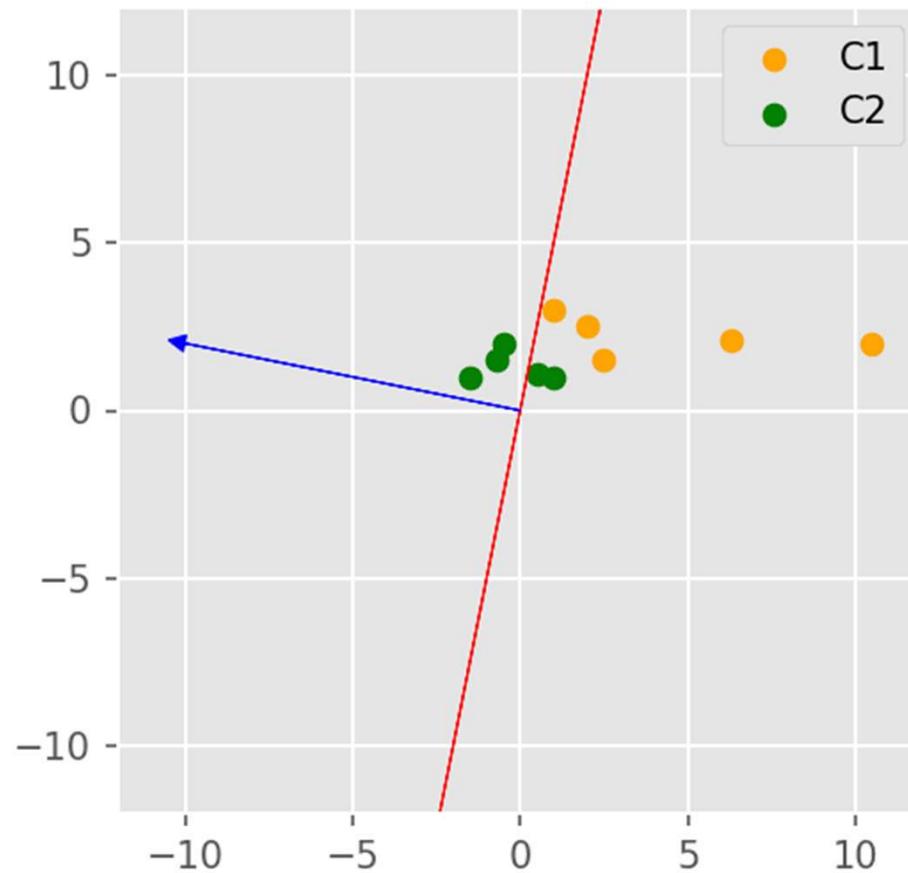
# El algoritmo

Iteration:21



# El algoritmo

Iteration:0



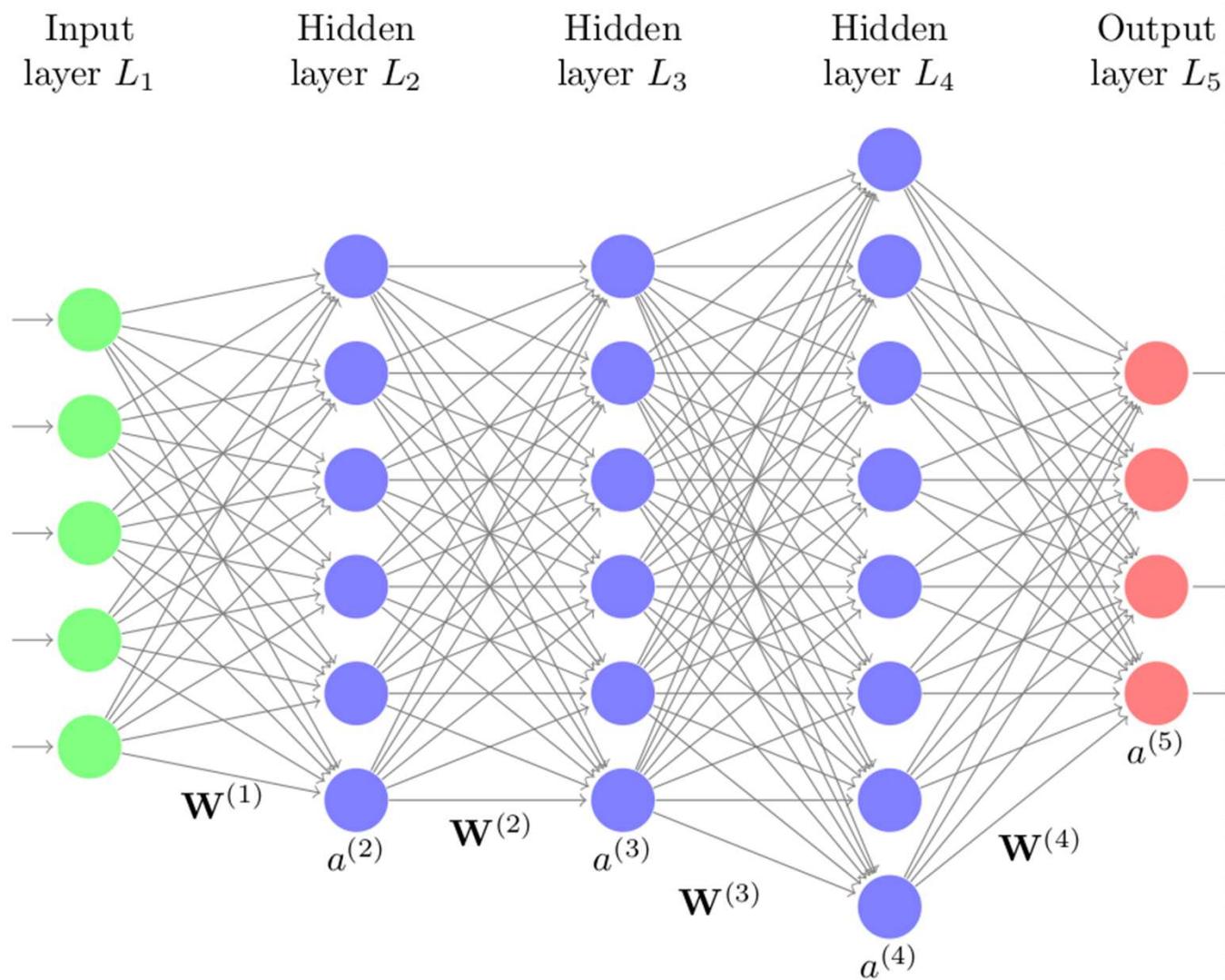
# Convergencia del algoritmo

Podría ser que al actualizar el vector de pesos un elemento que estaba bien clasificado pase a estar mal clasificado, por lo que no está garantizado reducir el total de la función de error en cada iteración. Sin embargo, el teorema de la convergencia del perceptrón (*perceptron convergence theorem*) que si existe una solución exacta esta será encontrada por el algoritmo en un número finito de pasos.

# Tipo de aprendizaje

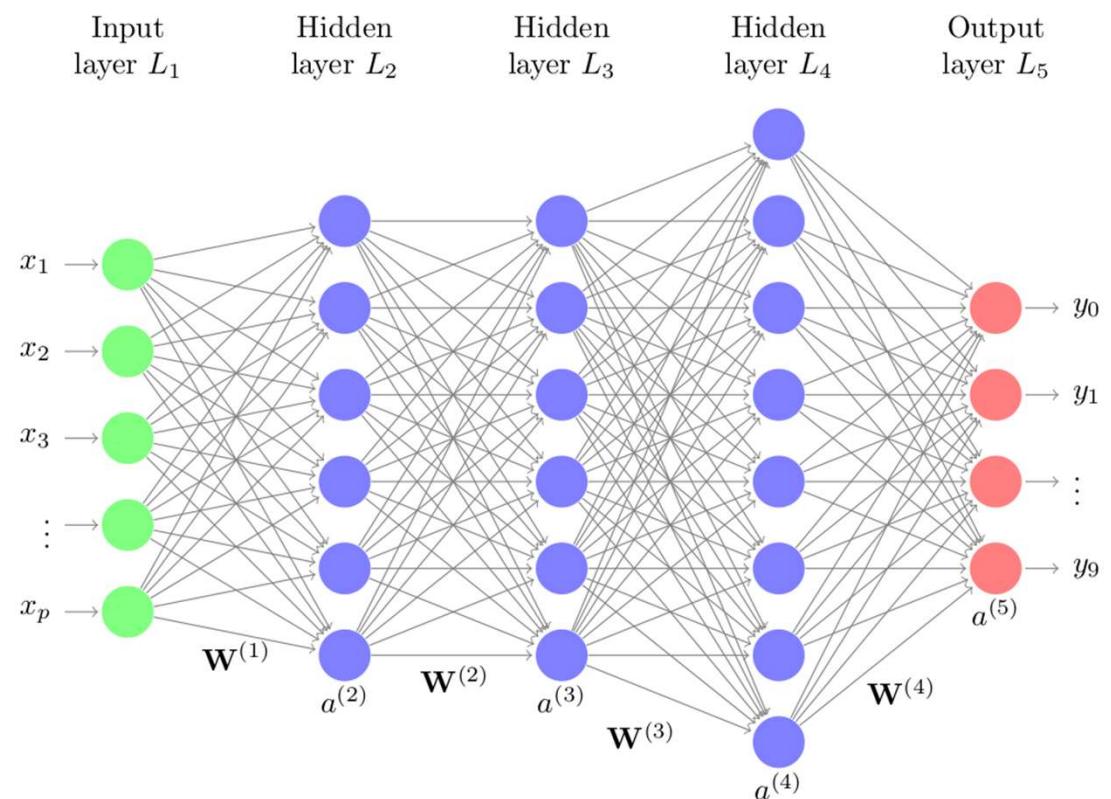
Dado que las actualizaciones se realizan para cada  $x_i$  de entrenamiento, este tipo de aprendizaje se denomina **aprendizaje en línea**. Si los errores se acumulan a lo largo de una **época** antes de actualizar los pesos, se denomina **aprendizaje por lotes o descenso de gradiente por lotes**.

Brownlee, J. (2021). How to Code a Neural Network with Backpropagation In Python (from scratch). (Visitado el 09/10/2022), <https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/>.



### 3.3 El perceptrón multicapa.

Generalizando



# Principios

## Entrenamiento

El objetivo es encontrar los valores correctos de cada capa de forma que las predicciones que se generen tengan el menor error posible.

$$\mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_N \end{bmatrix}$$

Identificar que predictores tienen mayor influencia.

# Principios

## Algoritmo general

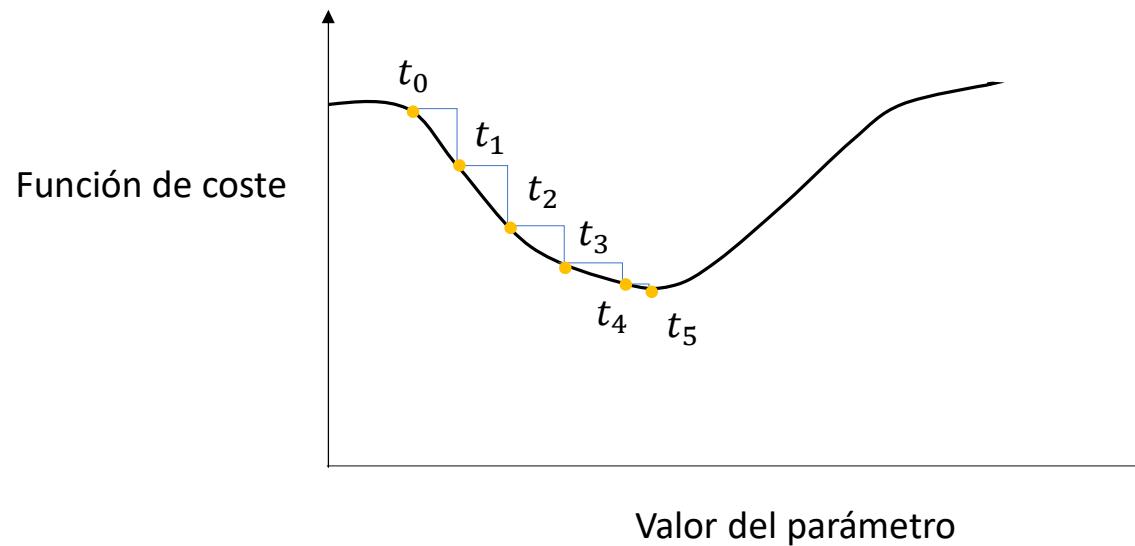
1. Iniciar con un lote  $X, y$  (*batch*) de los datos.
2. Iniciar cada  $W_i$  con valores aleatorios.
3. Para cada par  $(X, y)$  obtener  $\hat{y}$  y calcular el error, promediando el error de todas las observaciones (TMSE, MAE,  $L_{log}$ , etc).
4. Identificar cómo contribuye cada  $W_i$  en el error (gracias al gradiente).
5. Modificar ligeramente los pesos  $W_i = W_i + \Delta_i$  en la dirección que se reduce el error.
6. Si las condiciones de término no se cumplen, regresar al paso 3, de otra forma, terminar.

# Principios

## Algoritmo general

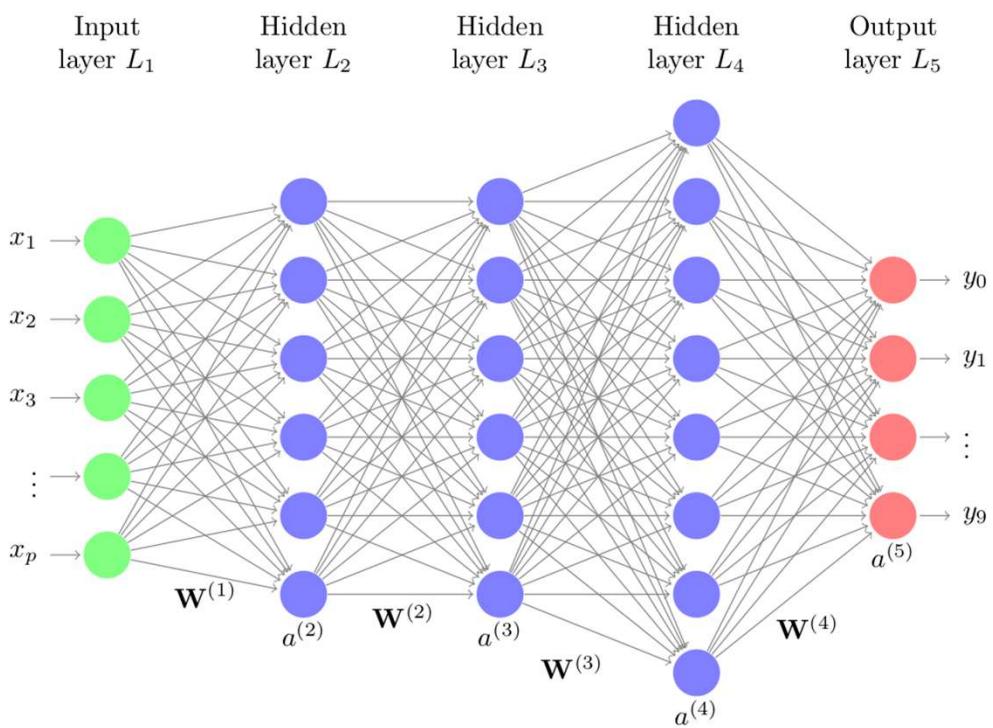
1. Iniciar con un lote  $X, y$  (*batch*) de los datos.
2. Iniciar cada  $W_i$  con valores aleatorios.
3. Para cada par  $(X, y)$  obtener  $\hat{y}$  y calcular el error, promediando el error de todas las observaciones (TMSE, MAE,  $L_{log}$ , etc).
4. Identificar cómo contribuye cada  $W_i$  en el error (gracias al gradiente).
5. Modificar ligeramente los pesos  $W_i = W_i + \Delta_i$  en la dirección que se reduce el error.
6. Si las condiciones de término no se cumplen, regresar al paso 3, de otra forma, terminar.

# Principios



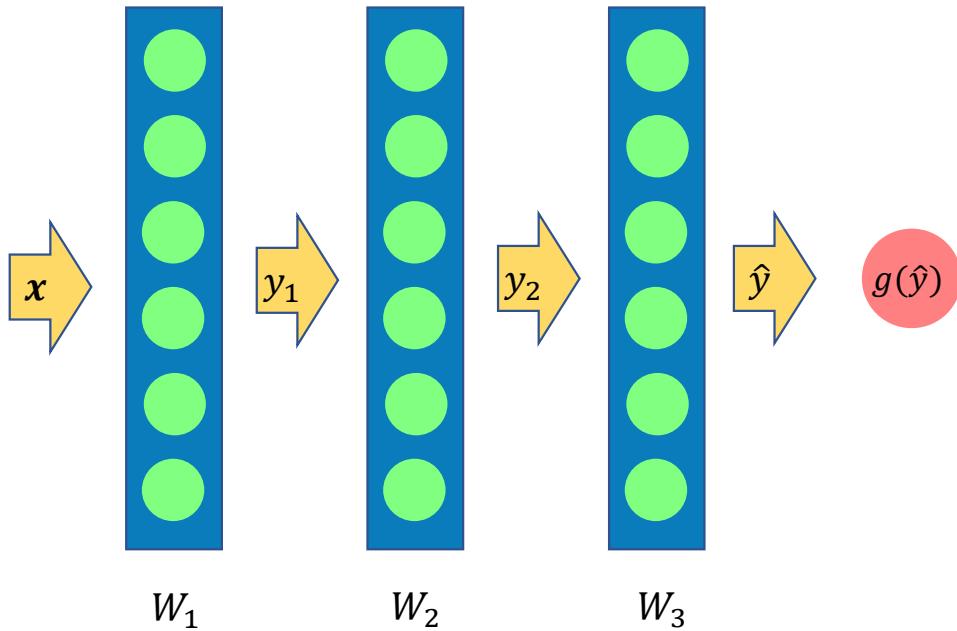
# Principios

## ***Backpropagation***



# Principios

*Backpropagation*



$$\hat{y} = y_2 W_3$$

$$y_2 = y_1 W_2$$

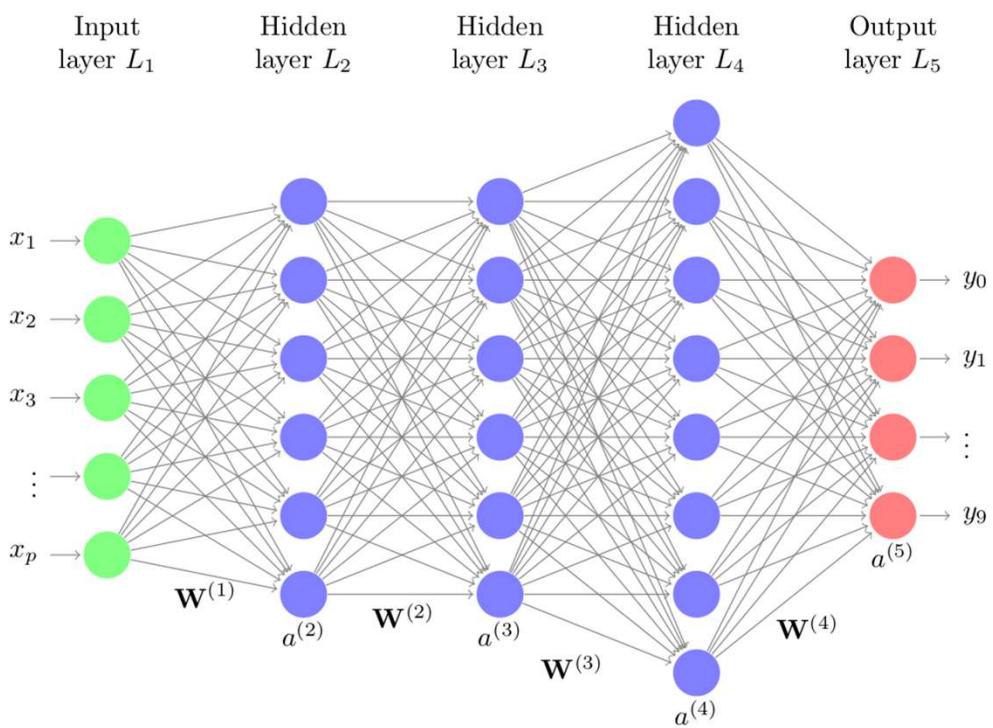
$$y_1 = x W_3$$

$$\hat{y} = x W_1 W_2 W_3$$

$$g(\hat{y}) = g(x W_1 W_2 W_3)$$

# Principios

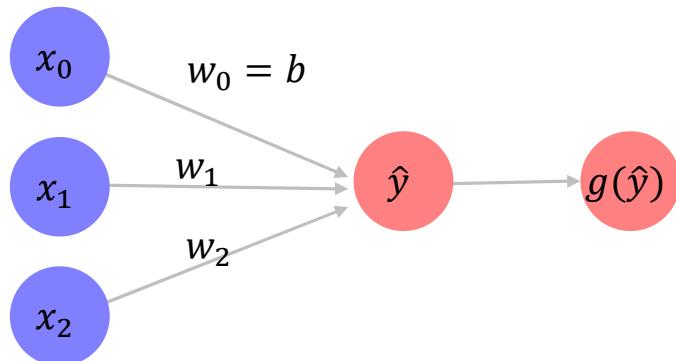
## ***Backpropagation***



# Principios

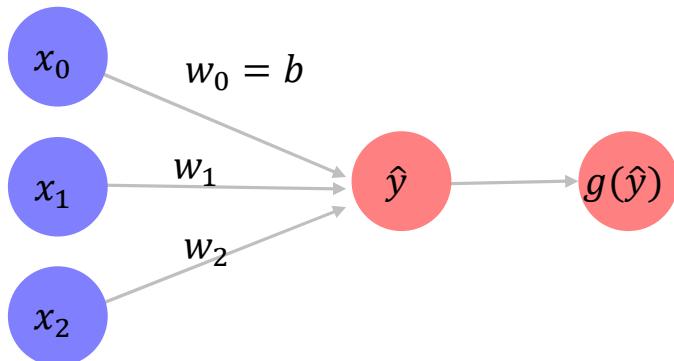
## ***Backpropagation***

*Veamos el caso para un perceptrón*



# Principios

## *Backpropagation*



$$y = 230$$

$$\hat{y} = x_1 w_1 + x_2 w_2 + w_0$$

$$g(\hat{y}) = (\hat{y} - y)^2 = (w_0 + x_1 w_1 + x_2 w_2 - y)^2$$

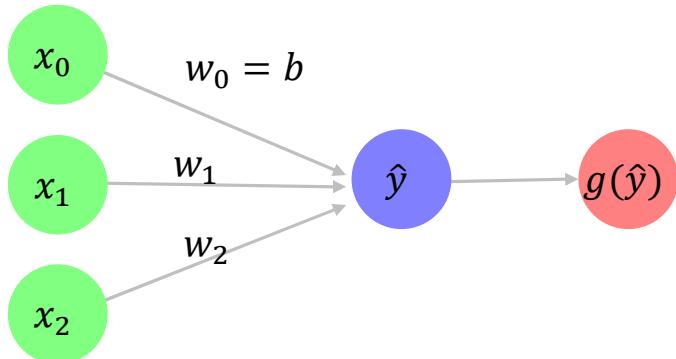
$$\frac{g'(\hat{y})}{dw_0} = 2(w_0 + x_1 w_1 + x_2 w_2 - y) = 2(\hat{y} - y)$$

$$\frac{g'(\hat{y})}{dw_1} = 2(w_0 + x_1 w_1 + x_2 w_2 - y)x_1 = 2(\hat{y} - y)x_1$$

$$\frac{g'(\hat{y})}{dw_2} = 2(w_0 + x_1 w_1 + x_2 w_2 - y)x_2 = 2(\hat{y} - y)x_2$$

# Principios

## *Backpropagation*



$$\hat{y} = 2w_1 + 7w_2 + w_0$$

$$y = 230$$

$$\hat{y} = x_1w_1 + x_2w_2 + w_0$$

$$g(\hat{y}) = (\hat{y} - y)^2 = (w_0 + x_1w_1 + x_2w_2 - y)^2$$

$$\frac{g'(\hat{y})}{dw_0} = 2(w_0 + x_1w_1 + x_2w_2 - y) = 2(\hat{y} - y)$$

$$\frac{g'(\hat{y})}{dw_1} = 2(w_0 + x_1w_1 + x_2w_2 - y)x_1 = 2(\hat{y} - y)x_1$$

$$\frac{g'(\hat{y})}{dw_2} = 2(w_0 + x_1w_1 + x_2w_2 - y)x_2 = 2(\hat{y} - y)x_2$$

$$w_0 = w_0 - \text{step } 2(\hat{y} - y)$$

$$w_1 = w_1 - \text{step } 2(\hat{y} - y) x_1$$

$$w_2 = w_2 - \text{step } 2(\hat{y} - y) x_2$$

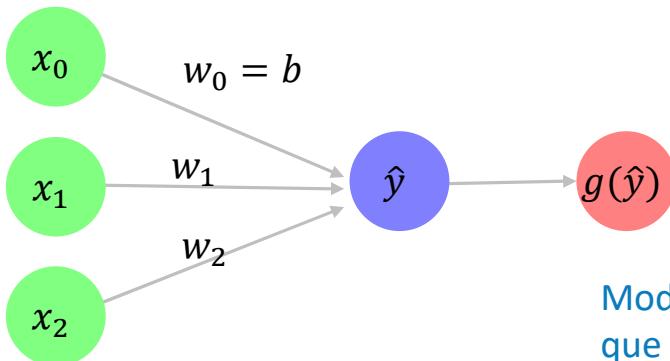
# Principios

## Algoritmo general

1. Iniciar con un lote  $X, y$  (*batch*) de los datos.
2. Iniciar cada  $W_i$  con valores aleatorios.
3. Para cada par  $(X, y)$  obtener  $\hat{y}$  y calcular el error, promediando el error de todas las observaciones (TMSE, MAE,  $L_{log}$ , etc).
4. Identificar cómo contribuye cada  $W_i$  en el error (gracias al gradiente).
5. Modificar ligeramente los pesos  $W_i = W_i + \Delta_i$  en la dirección que se reduce el error.
6. Si las condiciones de término no se cumplen, regresar al paso 3, de otra forma, terminar.

# Principios

## ***Backpropagation***



$$y = 230$$

$$\hat{y} = x_1 w_1 + x_2 w_2 + w_0$$

$$g(\hat{y}) = (\hat{y} - y)^2 = (w_0 + x_1 w_1 + x_2 w_2 - y)^2$$

Identificar cómo contribuye cada  $W_i$  en el error (gracias al gradiente).

$$\frac{g'(\hat{y})}{dw_0} = 2(w_0 + x_1 w_1 + x_2 w_2 - y) = 2(\hat{y} - y)$$

$$\frac{g'(\hat{y})}{dw_1} = 2(w_0 + x_1 w_1 + x_2 w_2 - y)x_1 = 2(\hat{y} - y)x_1$$

$$\frac{g'(\hat{y})}{dw_2} = 2(w_0 + x_1 w_1 + x_2 w_2 - y)x_2 = 2(\hat{y} - y)x_2$$

Modificar ligeramente los pesos  $W_i = W_i + \Delta_i$  en la dirección que se reduce el error.

$$w_0 = w_0 - \text{step } 2(\hat{y} - y)$$

$$w_1 = w_1 - \text{step } 2(\hat{y} - y) x_1$$

$$w_2 = w_2 - \text{step } 2(\hat{y} - y) x_2$$

# Ejemplo 45: Simulación Backpropagation en excel

# Principios

## *Backpropagation*

Ahora debemos optimizar para varias entradas  $X$

$$\mathbf{y} = \mathbf{X}^T \mathbf{W} + \boldsymbol{\varepsilon}, \quad \hat{\mathbf{y}} = \mathbf{x}^T \mathbf{W}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,D} \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & \cdots & x_{N,D} \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} w_0 \\ \vdots \\ w_N \end{bmatrix}, \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_N \end{bmatrix}$$

# Principios

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,D} \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & \cdots & x_{N,D} \end{bmatrix}$$

$$\hat{y}_i = x_{i,1}w_1 + x_{i,2}w_2 + \cdots + w_0$$

$$G(\hat{y}) = \sum_{i=1}^n g(\hat{y}_i) = \sum_{i=1}^n (\hat{y}_i - y_i)^2 = \sum_{i=1}^n (w_0 + x_{i,1}w_1 + x_{i,2}w_2 + \cdots + w_{i,D} - y_i)^2$$

$$\frac{g'(\hat{y}_i)}{dw_0} = \sum_{i=1}^n 2(w_0 + x_{i,1}w_1 + x_{i,2}w_2 + \cdots + w_{i,D} - y_i) = \sum_{i=1}^n 2(\hat{y}_i - y)$$

$$\frac{g'(\hat{y}_i)}{d\mathbf{w}_j} = \sum_{i=1}^n 2(w_0 + x_{i,1}w_1 + x_{i,2}w_2 + \cdots + w_{i,D} - y_i) \mathbf{x}_{i,j} = \sum_{i=1}^n 2(\hat{y}_i - y) \mathbf{x}_{i,j}$$

$$w_0 = w_0 - \text{step} \sum_{i=1}^n 2(\hat{y}_i - y)$$

$$\mathbf{w}_j = \mathbf{w}_j - \text{step} \sum_{i=1}^n 2(\hat{y}_i - y) \mathbf{x}_{i,j}$$

# Principios

## ***Backpropagation***

$$y_1 = g_1(x)$$

$$y_2 = g_2(g_1(x, W_1), W_2)$$

$$\hat{y} = g_3(g_2(g_1(x, W_1), W_2), W_3)$$

$$g(\hat{y}) = g(g_3(g_2(g_1(x, W_1), W_2), W_3))$$

## **Regla de la cadena**

Si tenemos que  $h(x) = f(g(x))$

Entonces  $h'(x) = f'(g(x))g'(x)$

## *Backpropagation* en una red multicapa

Varias funciones de error se expresan como una suma de términos, uno para cada dato del conjunto de entrenamiento.

$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w})$$

Consideremos el problema de evaluar  $\nabla E(\mathbf{w})$  para uno de esos términos en la función de error a usar en una **optimización en línea (secuencial)** o acumulada en los **métodos por lotes (batch)**.

# *Backpropagation* en una red multicapa

*Consideremos el caso simple del **modelo lineal** tenemos que*

$$y_k = \sum_i w_{ki} x_i$$

*Por lo tanto el error para una entrada toma la forma siguiente:*

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

# *Backpropagation* en una red multicapa

*El gradiente de*

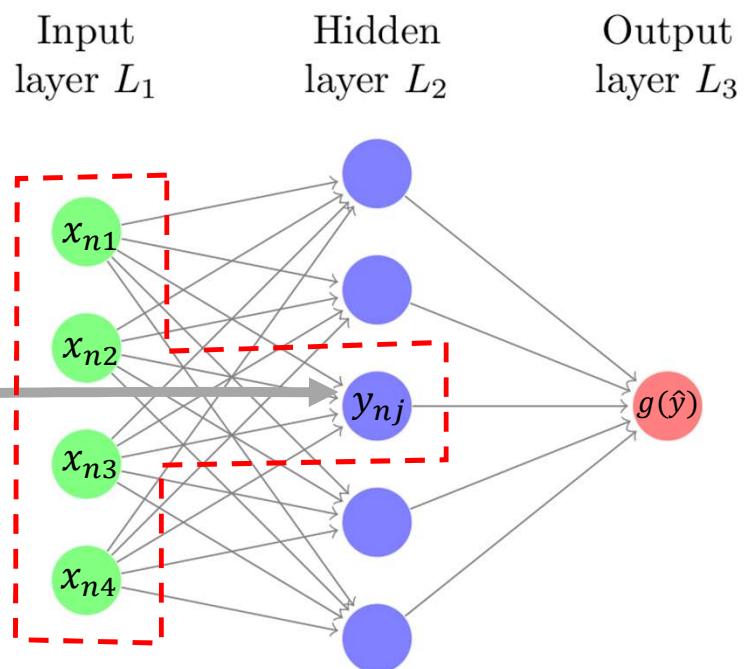
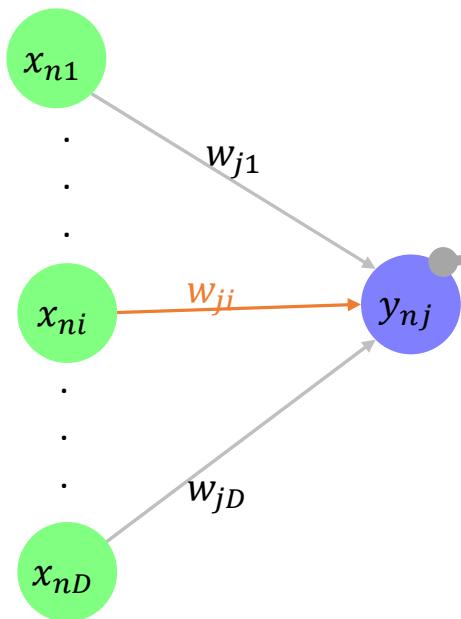
$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

*con respecto a un peso  $w_{ji}$  (peso del nodo  $i$  de la capa  $l$ , al nodo  $j$  de la capa  $l + 1$ ) está dado por:*

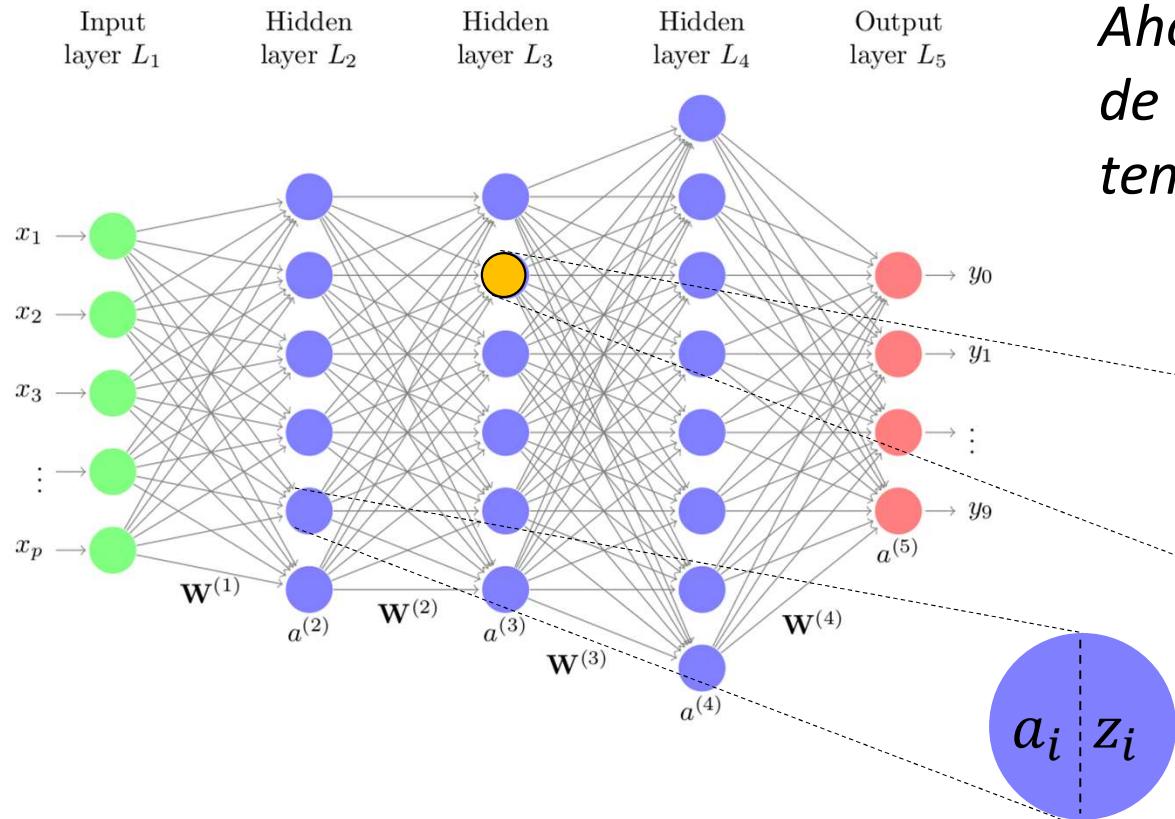
$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni}$$

# *Backpropagation en una red multicapa*

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni}$$



# *Backpropagation en una red multicapa*



*Ahora consideremos en **caso general** de una red neuronal multi capa tenemos que:*

$$a_j^{(l)} = \sum_i w_{ji} z_i \quad (1)$$

$$z = h(a)$$