

INTELIGENCIA ARTIFICIAL (1INF24)



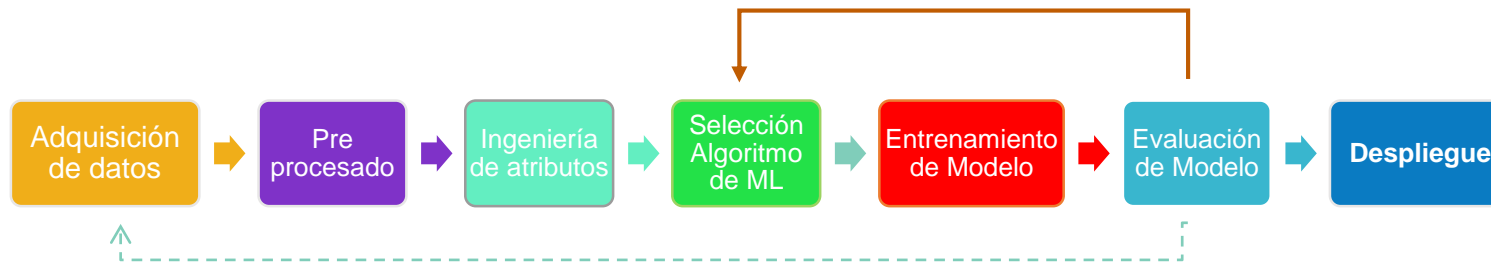
Unidad 2: Fundamentos de *Machine Learning* y redes neuronales artificiales

Tema 5: *Introducción a las Redes Neuronales Artificiales*

Dr. Edwin Villanueva Talavera

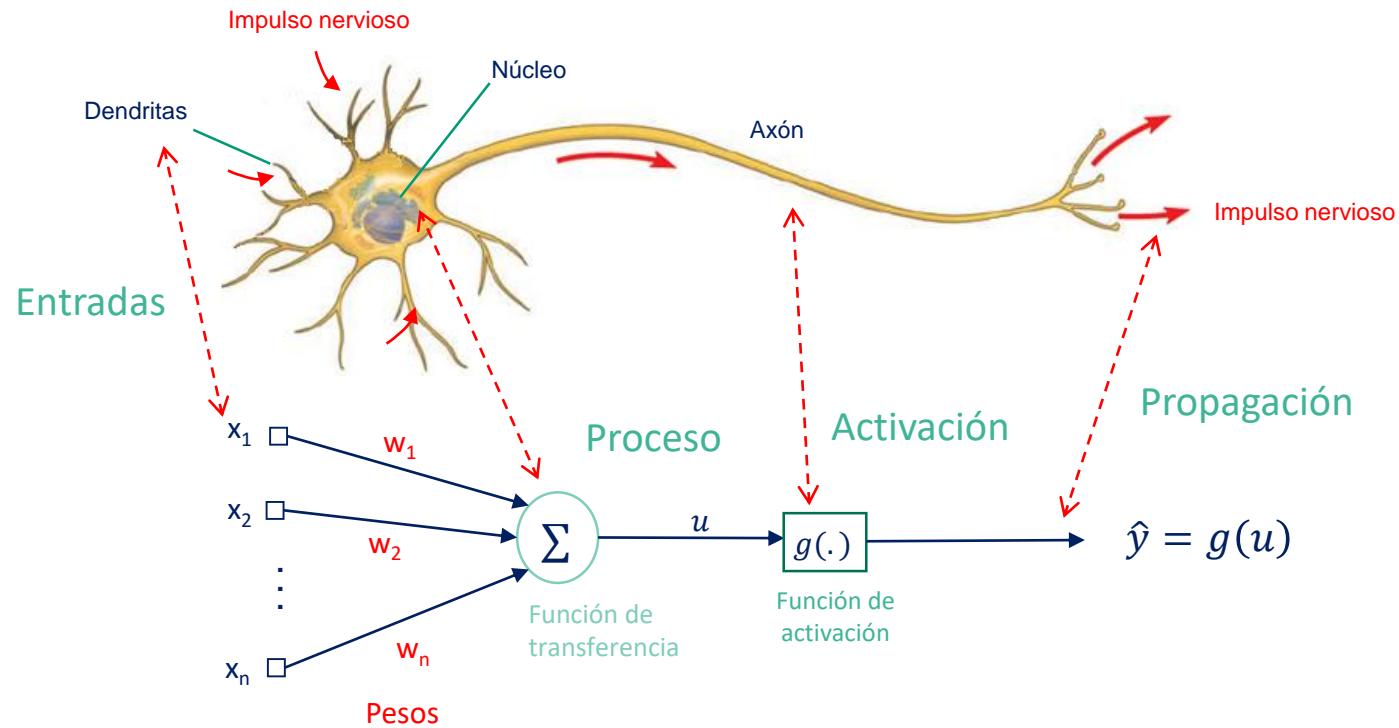
Contenido

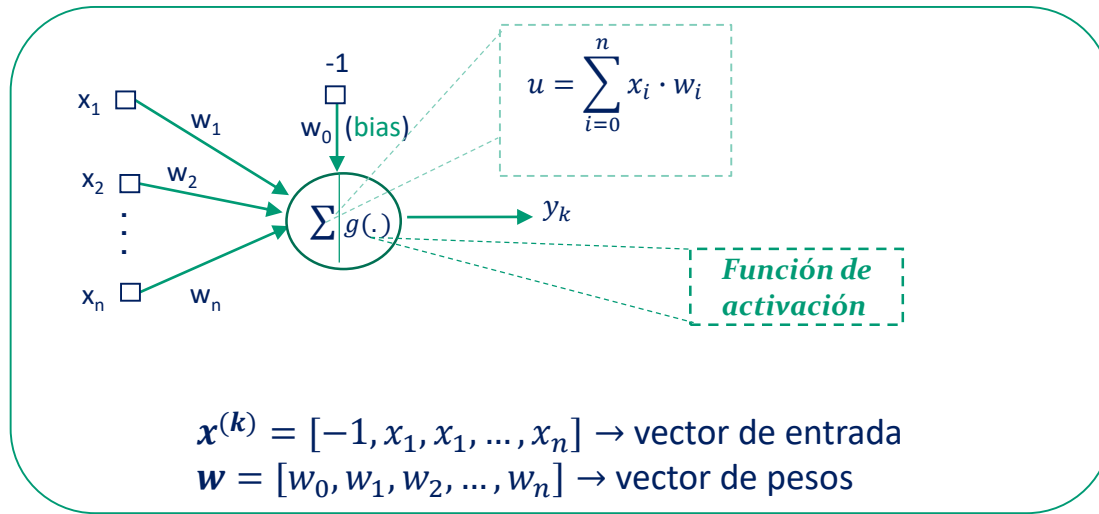
- La neurona biológica
- La neurona artificial
- Redes Multilayer Perceptron (MLP)
- Entrenamiento de MLP
- Aplicación en MLP en clasificación de imágenes



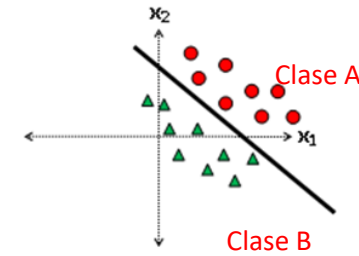
Neurona artificial

- **Perceptron:** McCullouch y Pitts (1943) propusieron un modelo computacional inspirado en las redes neuronales biológicas.

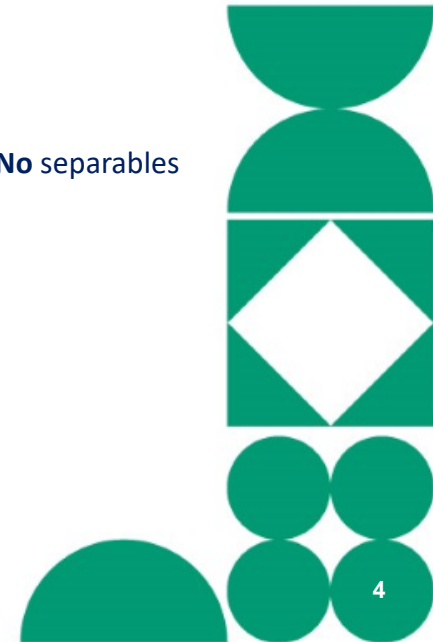
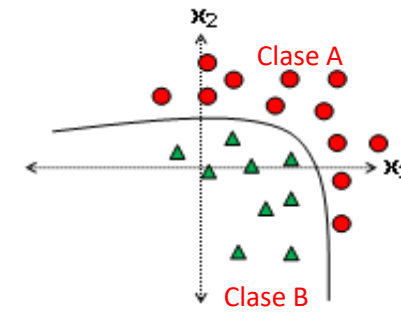




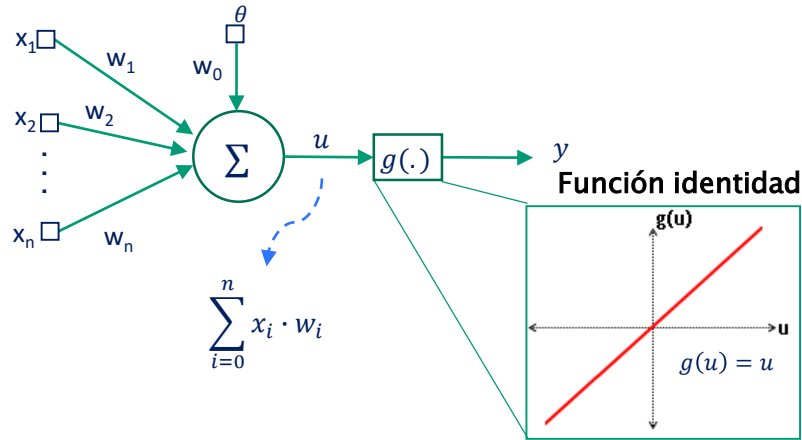
Puede separar clases linealmente separables



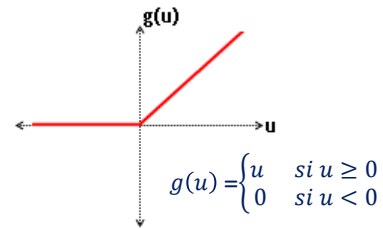
No es capaz de separar clases linealmente **No** separables



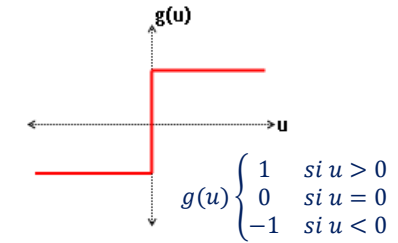
Funciones de activación populares



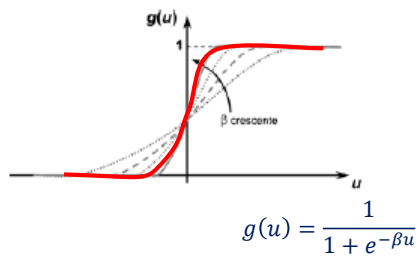
Función RELU



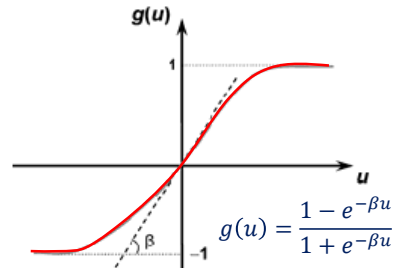
Función signo



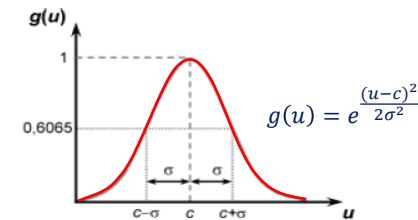
Función sigmoidea



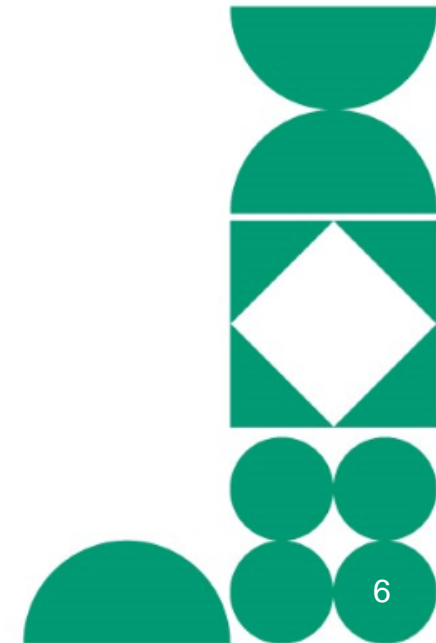
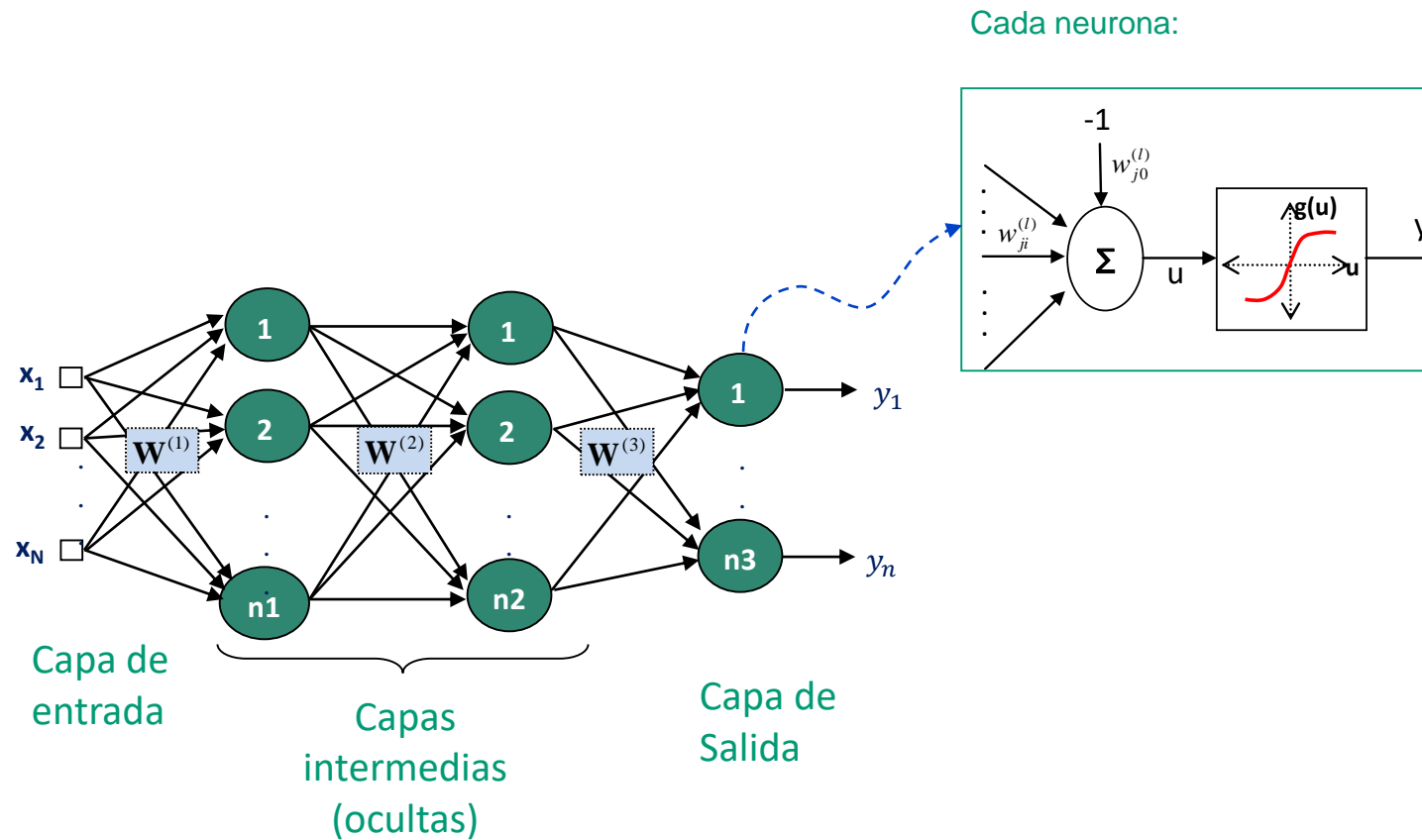
Función tangente hiperbólica



Función gaussiana



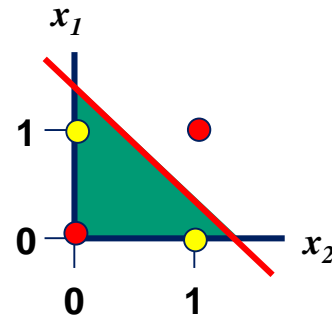
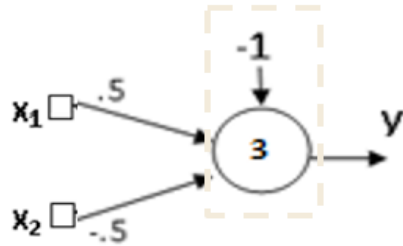
Redes Multilayer Perceptron (MLP)



Redes Multilayer Perceptron (MLP)

XOR con MLPs:

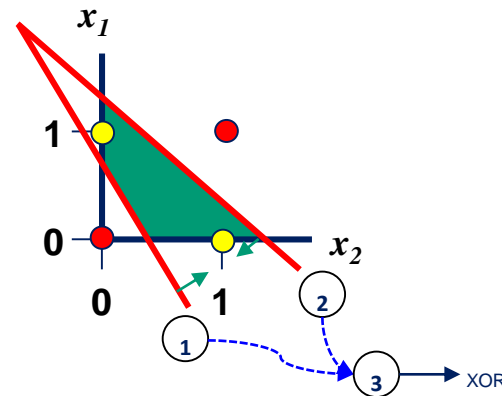
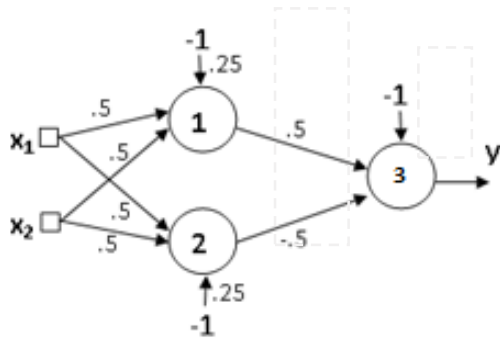
1 Capa



XOR

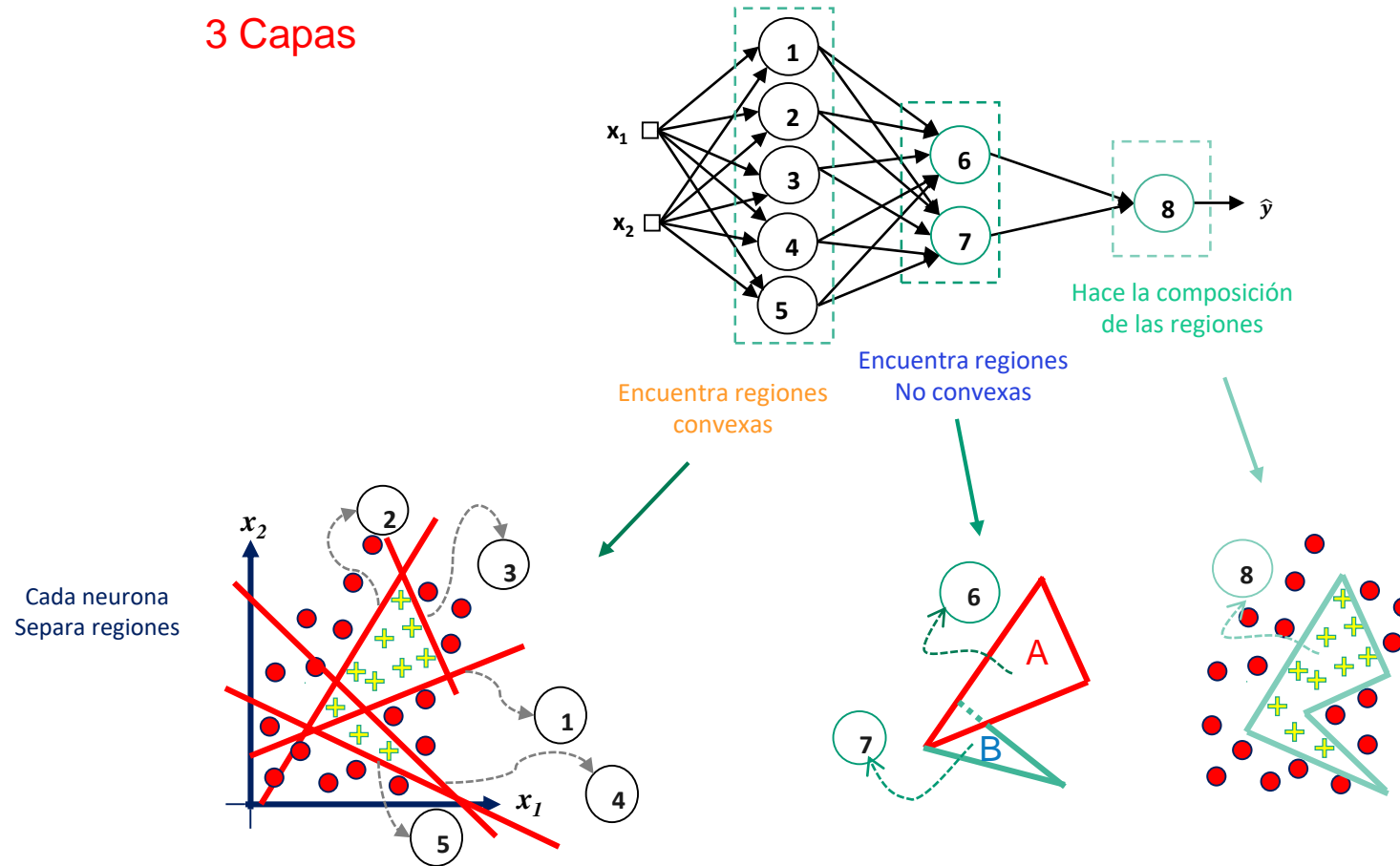
x_1	x_2	y
1	1	0
0	1	1
1	0	1
0	0	0

2 Capas



Redes Multilayer Perceptron (MLP)

3 Capas



Redes Multilayer Perceptron (MLP)

Entrenamiento

El proceso de entrenamiento de MLP se realiza mediante el algoritmo **BackPropagation**, también conocido como **regla delta generalizada**.

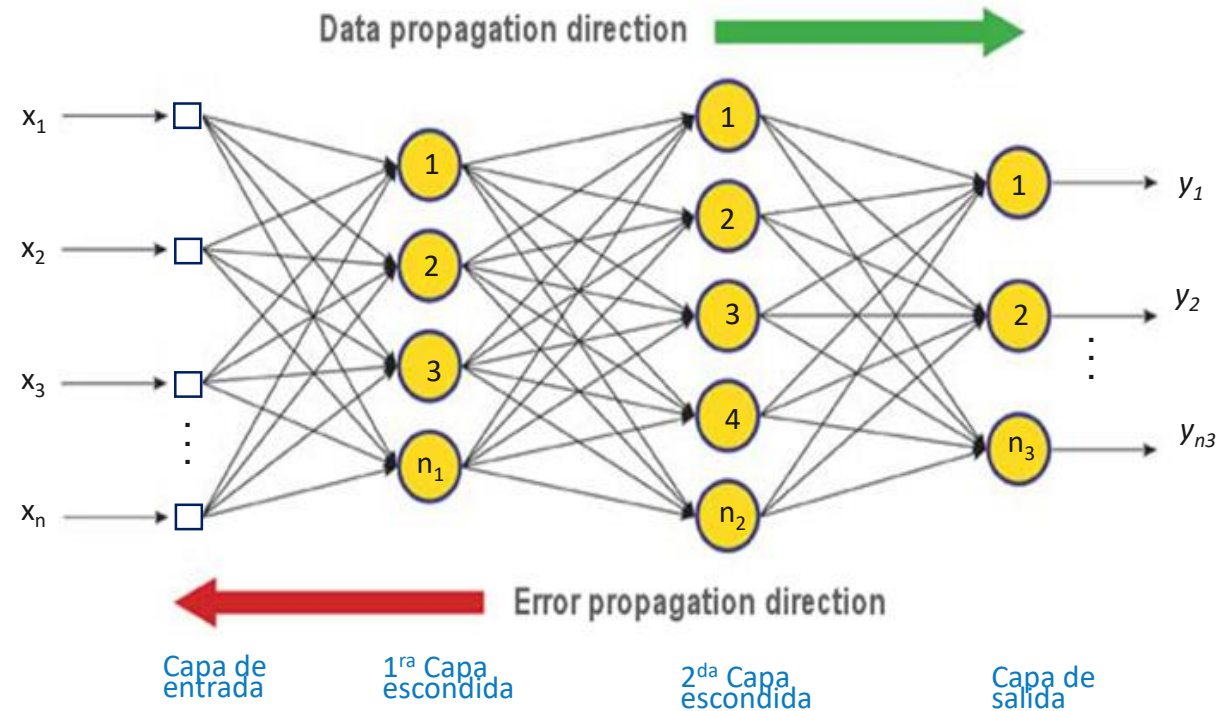
- El proceso se lleva a cabo mediante sucesivas aplicaciones de dos fases muy específicas.

I. Fase Forward

Destinado a ajustar la matriz de pesos sinápticos hacia la capa neural de salida.

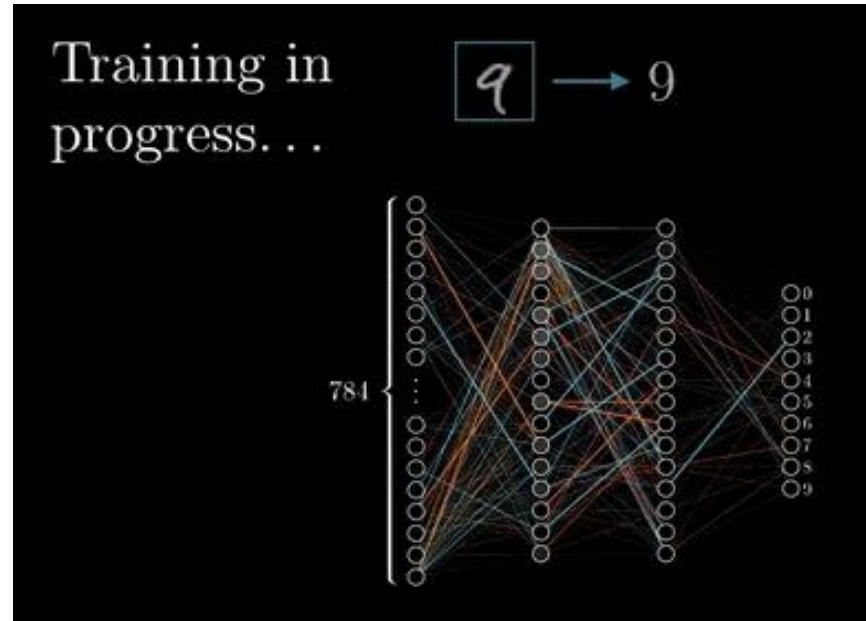
II. Fase Backward

Destinado a ajustar las matrices de pesos asociados a las capas intermedias.



Redes Multilayer Perceptron (MLP)

- Algoritmo de entrenamiento BackPropagation



I. Fase Forward



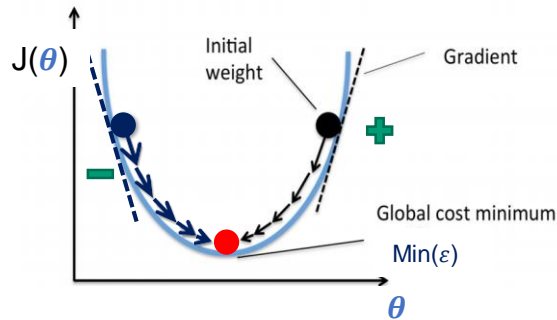
II. Fase Backward



Redes Multilayer Perceptron (MLP)

Entrenamiento backpropagation: caso de un solo parámetro

1. Iniciar con valores aleatorios el vector de θ^T

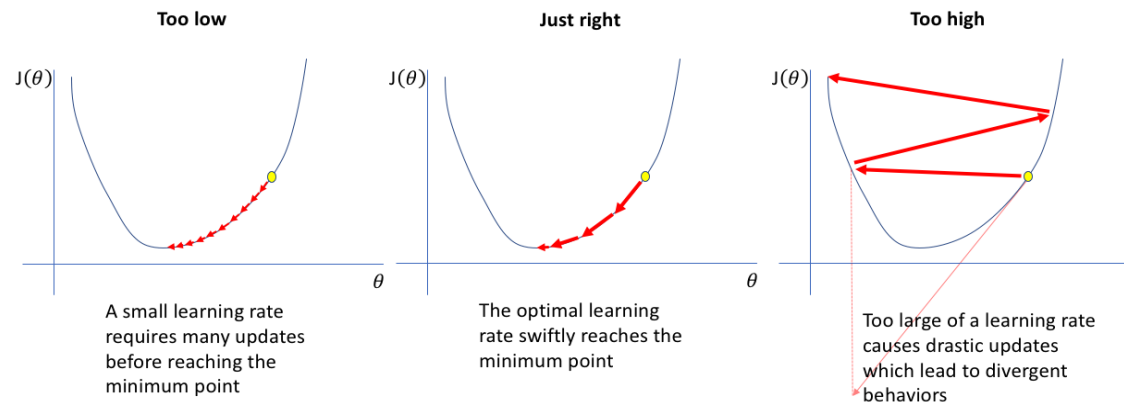


2. En cada iteración actualiza los valores de θ usando la siguiente ecuación:

$$\theta^{(t)} = \theta^{(t-1)} - \eta \frac{\partial J(\theta)}{\partial \theta}$$

gradiente: define cuan rápido cambia la función de costo J con respecto al cambio de θ

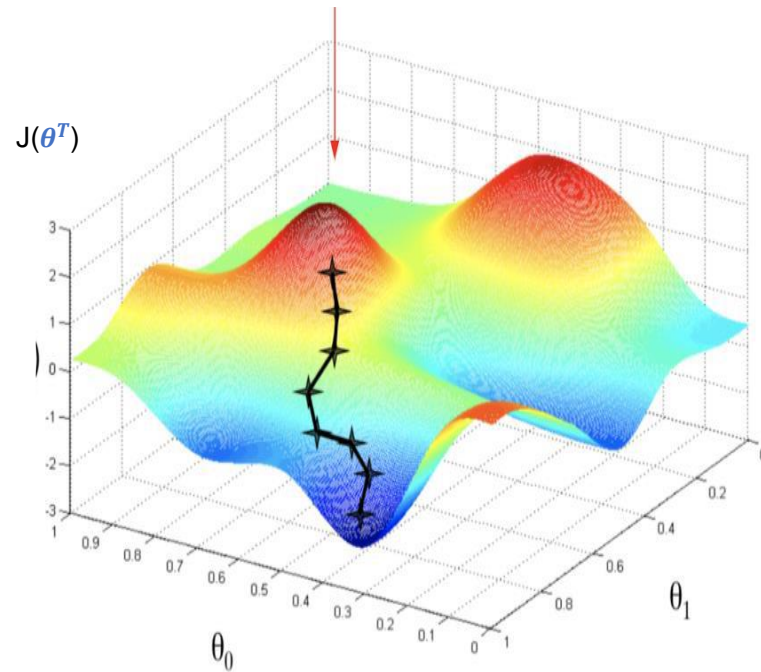
Tasa de aprendizaje: define la cantidad de iteraciones requerida para que el algoritmo encuentre el mínimo de la función. La tasa siempre es un valor menor a 1.



Redes Multilayer Perceptron (MLP)

Entrenamiento backpropagation: caso de dos parámetros

1. Iniciar con valores aleatorios el vector de $\theta^T = [\theta_0, \theta_1]^T$

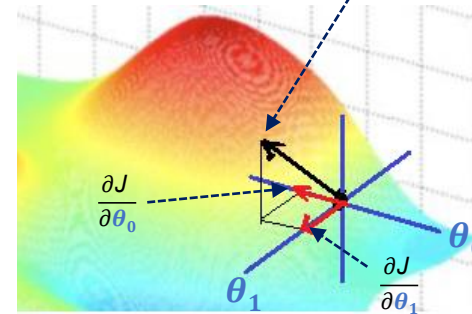


2. En cada iteración actualiza los valores de θ^T usando la siguiente ecuación:

$$\theta^{T(t)} = \theta^{T(t-1)} - \eta \nabla J(\theta^T)$$

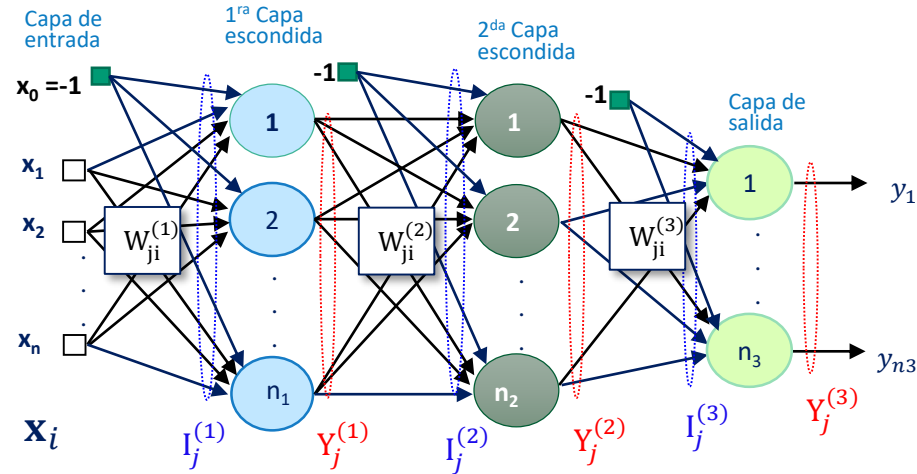
Vector gradiente:

$$\nabla J = \left[\frac{\partial J}{\partial \theta_0}, \frac{\partial J}{\partial \theta_1} \right]$$

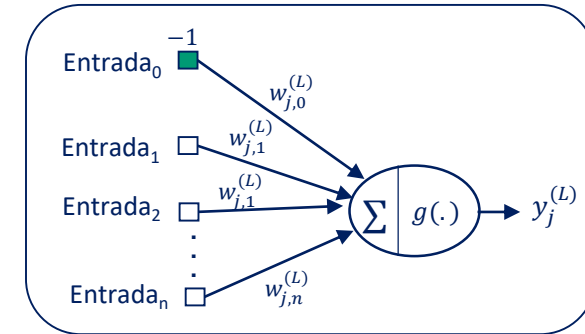


Algoritmo *Backpropagation*

- Notación de variables y parámetros



Cada neurona tiene la siguiente configuración



$$W_{ji}^{(L)}$$

Peso de conexión de la neurona i de capa $L-1$ hacia neurona j de capa L

$I_j^{(L)}$: entrada ponderada de neurona j en capa L

$$I_j^{(1)} = \sum_{i=1}^n W_{ji}^{(1)} \cdot x_i^{(1)}$$

$$I_j^{(2)} = \sum_{i=1}^n W_{ji}^{(2)} \cdot Y_j^{(2)}$$

$$I_j^{(3)} = \sum_{i=1}^n W_{ji}^{(3)} \cdot Y_j^{(3)}$$

$Y_j^{(L)}$: salida de neurona j de capa L

$$Y_j^{(1)} = g(I_j^{(1)})$$

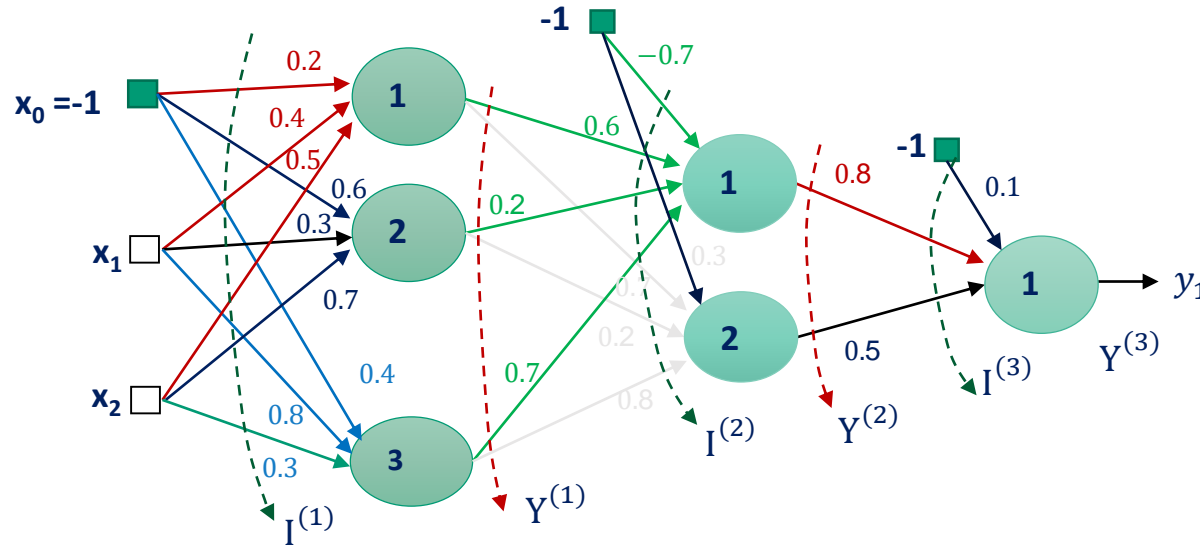
$$Y_j^{(2)} = g(I_j^{(2)})$$

$$Y_j^{(3)} = g(I_j^{(3)})$$

$g(\cdot)$: Función de activación

Algoritmo *Backpropagation*

Ejemplo:



Matrices de pesos:

$$W_{ji}^{(1)} = \begin{bmatrix} 0.2 & 0.4 & 0.5 \\ 0.3 & 0.6 & 0.7 \\ 0.4 & 0.8 & 0.3 \end{bmatrix}$$

$$W_{ji}^{(2)} = \begin{bmatrix} -0.7 & 0.6 & 0.2 & 0.7 \\ 0.3 & 0.7 & 0.2 & 0.8 \end{bmatrix}$$

$$W_{ji}^{(3)} = [0.1 \quad 0.8 \quad 0.5]$$

$I_j^{(L)} \rightarrow$ entradas ponderadas

$$I_j^{(L)} = \sum_{i=1}^N W_{ji}^{(L)} \cdot x_i$$

$Y_j^{(L)} \rightarrow$ salidas de cada capa

$$Y_j^{(L)} = g(I_j^{(L)})$$

$g()$: tangente hiperbólica

Cálculo de $I_j^{(L)}$ para $x_1 = 0.3$ y $x_2 = 0.7$

$$I_j^{(1)} = \begin{bmatrix} W_{10}^{(1)} \cdot x_0 + W_{11}^{(1)} \cdot x_1 + W_{12}^{(1)} \cdot x_2 \\ W_{20}^{(1)} \cdot x_0 + W_{21}^{(1)} \cdot x_1 + W_{22}^{(1)} \cdot x_2 \\ W_{30}^{(1)} \cdot x_0 + W_{31}^{(1)} \cdot x_1 + W_{32}^{(1)} \cdot x_2 \end{bmatrix} = \begin{bmatrix} 0.2 \cdot (-1) + 0.4 \cdot 0.3 + 0.5 \cdot 0.7 \\ 0.3 \cdot (-1) + 0.6 \cdot 0.3 + 0.7 \cdot 0.7 \\ 0.4 \cdot (-1) + 0.8 \cdot 0.3 + 0.3 \cdot 0.7 \end{bmatrix} = \begin{bmatrix} 0.27 \\ 0.37 \\ 0.05 \end{bmatrix}$$

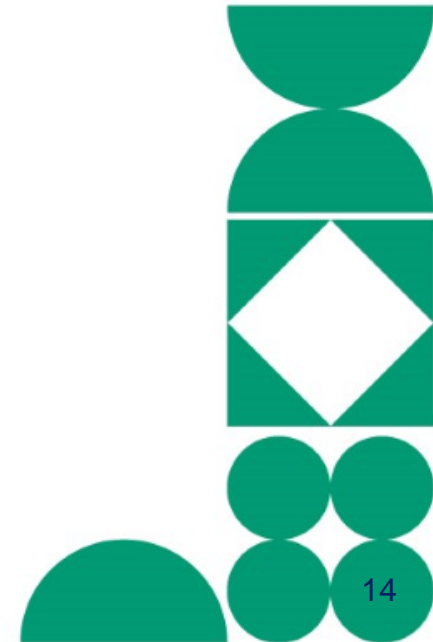
Cálculo de $Y_j^{(L)}$

$$Y_j^{(1)} = \begin{bmatrix} Y_1^{(1)} \\ Y_2^{(1)} \\ Y_3^{(1)} \end{bmatrix} = \begin{bmatrix} g(I_1^{(1)}) \\ g(I_2^{(1)}) \\ g(I_3^{(1)}) \end{bmatrix} = \begin{bmatrix} \tanh(0.27) \\ \tanh(0.37) \\ \tanh(0.05) \end{bmatrix} = \begin{bmatrix} 0.26 \\ 0.35 \\ 0.05 \end{bmatrix} \xrightarrow{Y_0^{(1)}=-1} Y_j^{(1)} = \begin{bmatrix} Y_0^{(1)} \\ Y_1^{(1)} \\ Y_2^{(1)} \\ Y_3^{(1)} \end{bmatrix} = \begin{bmatrix} -1 \\ 0.26 \\ 0.35 \\ 0.05 \end{bmatrix} \rightarrow I_j^{(2)} = \begin{bmatrix} 0.96 \\ 0.59 \end{bmatrix}$$

$$Y_j^{(2)} = \begin{bmatrix} Y_1^{(2)} \\ Y_2^{(2)} \end{bmatrix} = \begin{bmatrix} g(I_1^{(2)}) \\ g(I_2^{(2)}) \end{bmatrix} = \begin{bmatrix} \tanh(0.96) \\ \tanh(0.59) \end{bmatrix} = \begin{bmatrix} 0.74 \\ 0.53 \end{bmatrix} \xrightarrow{Y_0^{(2)}=-1} Y_j^{(2)} = \begin{bmatrix} Y_0^{(2)} \\ Y_1^{(2)} \\ Y_2^{(2)} \end{bmatrix} = \begin{bmatrix} -1 \\ 0.74 \\ 0.53 \end{bmatrix} \rightarrow I_j^{(3)} = [0.76]$$

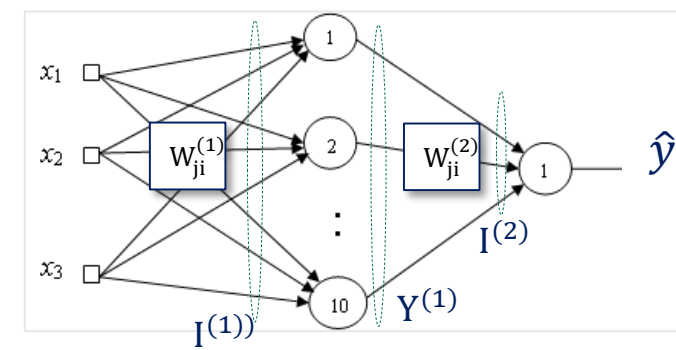
salida de la red

$$Y_j^{(3)} = [Y_1^{(3)}] = [g(I_1^{(3)})] = [\tanh(0.76)] = [0.64]$$



Algoritmo *Backpropagation*

□ Paso Forward



Inicializar los pesos $W_{ji}^{(l)}$ de la red con valores aleatorios

Definir parámetros de aprendizaje (ej. $\eta = 0.01$) y tasa de error (ej. $\varepsilon = 10^{-6}$)

Definir función de activación (ej. función logística)

Repetir una cantidad de veces (épocas)

Para cada par entrada-salida $\{\mathbf{x}^{(k)}, y^{(k)}\}$

Calcular la salida $\hat{y}^{(k)}$ para la entrada $\mathbf{x}^{(k)} \rightarrow$ Paso Forward

Datos de entrada

$$\begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ -1 & -0.6 & 0.1 & 4.0 \\ -1 & -1.4 & 0.8 & 4.4 \\ \vdots & \vdots & \vdots & \vdots \\ -1 & 0.6 & 0.2 & 5.8 \end{bmatrix} \mathbf{x}^{(k)}$$

targets

$$\begin{bmatrix} y \\ 1 \\ -1 \\ \vdots \\ 1 \end{bmatrix} y^{(k)}$$

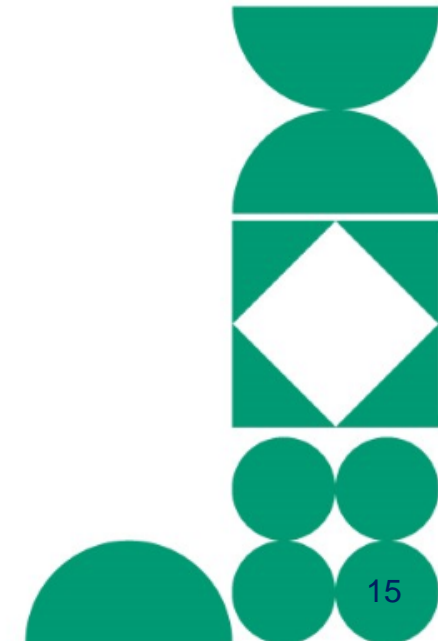
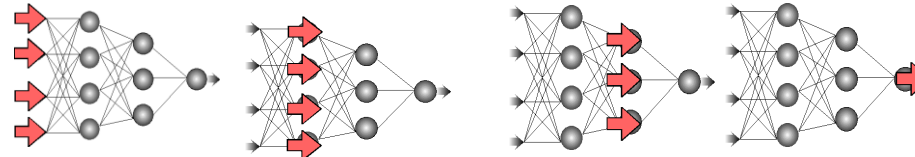
$I^{(1)} = W_{ji}^{(1)} \cdot \mathbf{x}^{(k)} \rightarrow$ Salida del combinador lineal de capa 1

$Y^{(1)} = g(I^{(1)}) \rightarrow$ Salida de la Capa 1

$Y^{(1)} = [-1, Y^{(1)}] \rightarrow$ concatena -1 (para el bias de la capa 2)

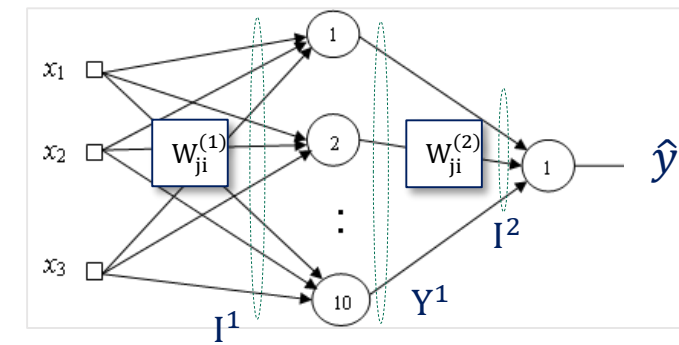
$I^{(2)} = W_{ji}^{(2)} \cdot Y^{(1)} \rightarrow$ Salida del combinador lineal de capa 2

$\hat{y}^{(k)} = g(I^{(2)}) \rightarrow$ Salida hacia la Capa 2 con beta=1



Algoritmo *Backpropagation*

□ Paso backward



Repetir una cantidad de veces (épocas)

Para cada par entrada-salida $\{\mathbf{x}^{(k)}, y^{(k)}\}$

Calcular la salida $\hat{y}^{(k)}$ para la entrada $\mathbf{x}^{(k)} \rightarrow$ Paso Forward

Actualizar $W_{ji}^{(L)}$ para cada capa $L \rightarrow$ Paso Backward

Gradiente local de
neurona j de capa 2

$$\text{Delta}2_j = (y_j^{(k)} - \hat{y}_j^{(k)}) \cdot g'(I_j^{(2)}) \quad \# \ g'(): \text{derivada de la función de activación}$$

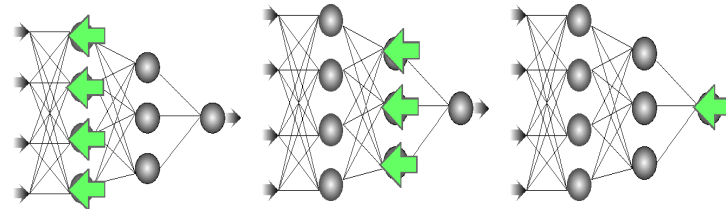
$$W_{ji}^{(2)} = W_{ji}^{(2)} + \eta \cdot \text{Delta}2_j \cdot Y_i^{(1)} \quad \# \text{ Reajuste de pesos de capa 2}$$

Gradiente local de
neurona j de capa 1

$$\text{Delta}1_j = g'(I_j^{(1)}) \cdot \sum_{k=1}^{n2} (\text{Delta}2_k \cdot W_{kj}^{(2)}) \quad \# \ n2: \text{nro neuronas en capa 2}$$

$$W_{ji}^{(1)} = W_{ji}^{(1)} + \eta \cdot \text{Delta}1_j \cdot x_i^{(k)} \rightarrow \text{Reajuste de pesos de capa 1}$$

end

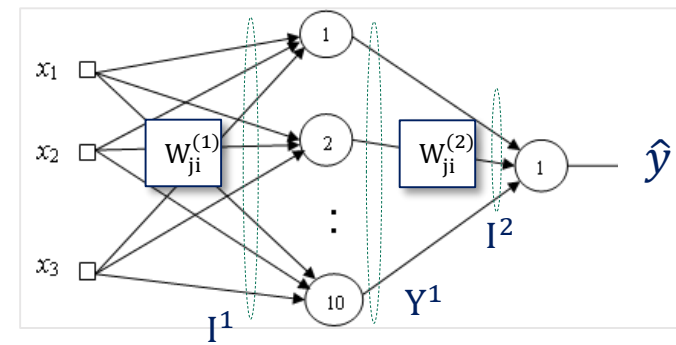


Esto es porque se asume que la función de costo J es el error cuadrático medio $\text{MSE} = (y^{(k)} - \hat{y}^{(k)})^2$

En general este termino es $\frac{\partial J}{\partial \hat{y}^{(k)}}$ evaluado en $\hat{y}^{(k)}$



Algoritmo *Backpropagation*



□ Inferencia

Presentar un vector de datos \mathbf{x} a clasificar

Con $W_{ji}^{(2)}$ y $W_{ji}^{(1)}$ ajustadas en la fase de entrenamiento

Ejecutar las siguientes instrucciones:

 Obtener I^1 y Y^1

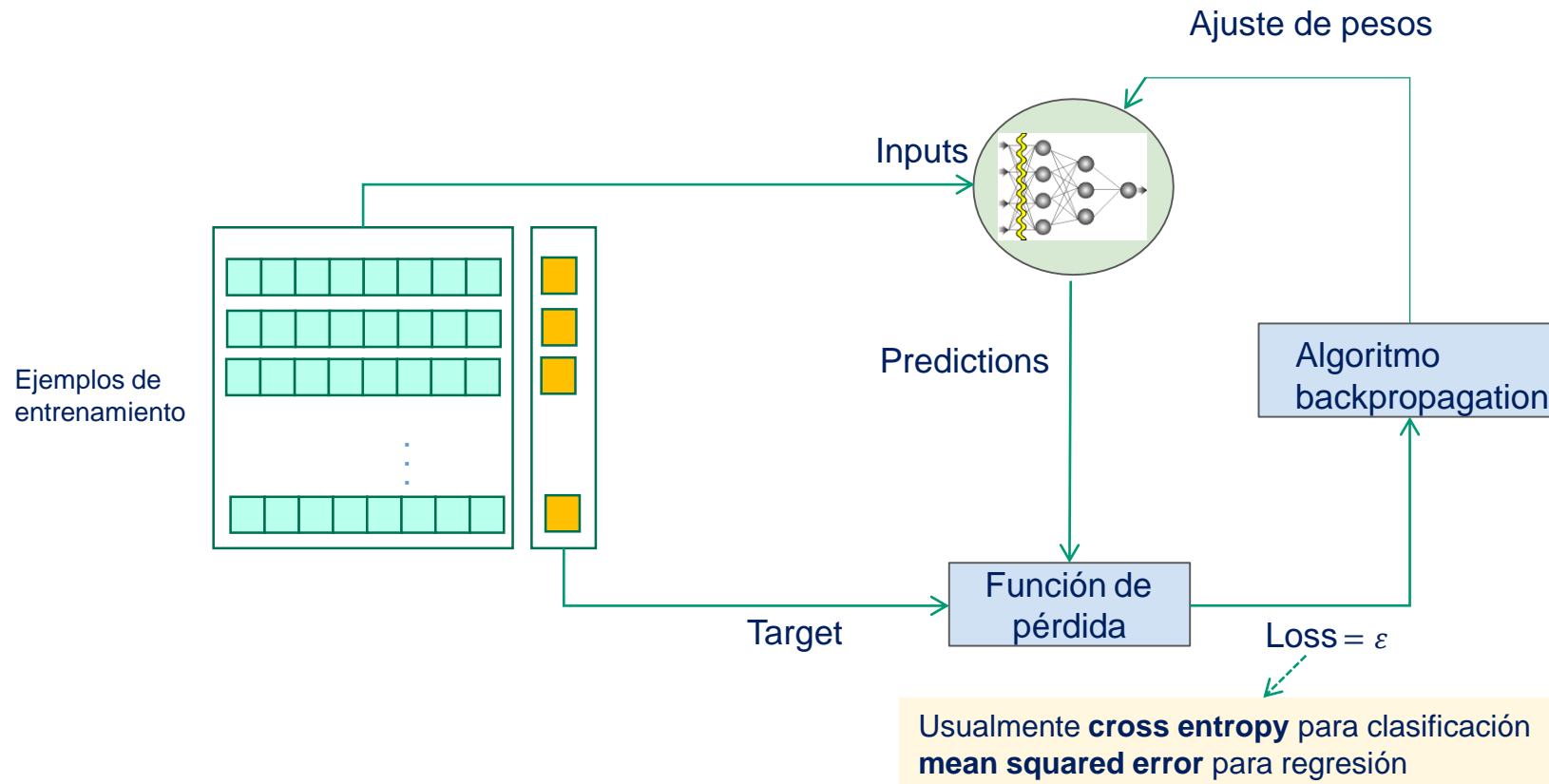
 Obtener I^2 y \hat{y}

end



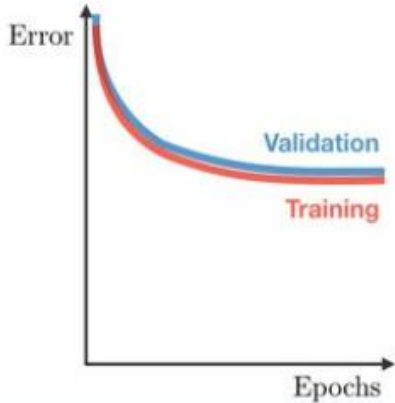
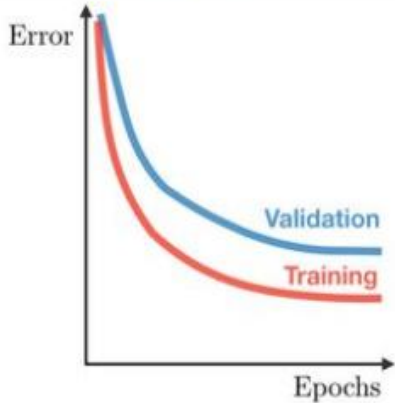
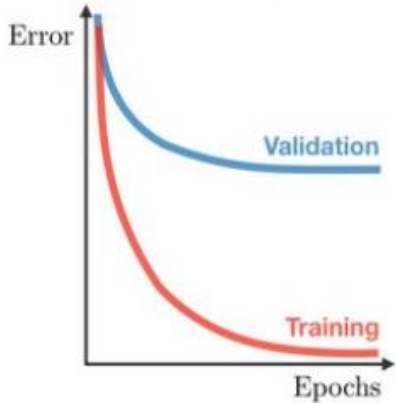
Redes Multilayer Perceptron (MLP)

Entrenamiento



Diagnóstico del aprendizaje

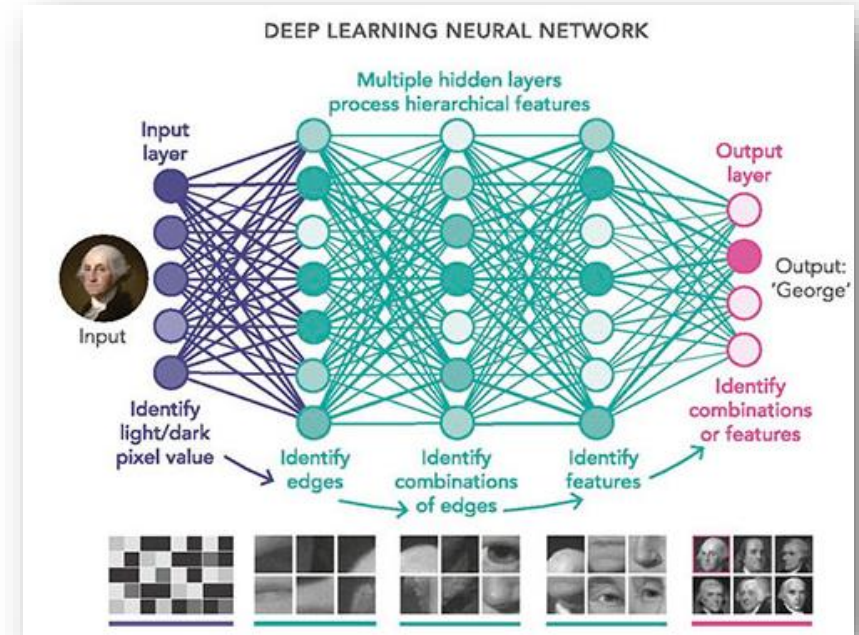
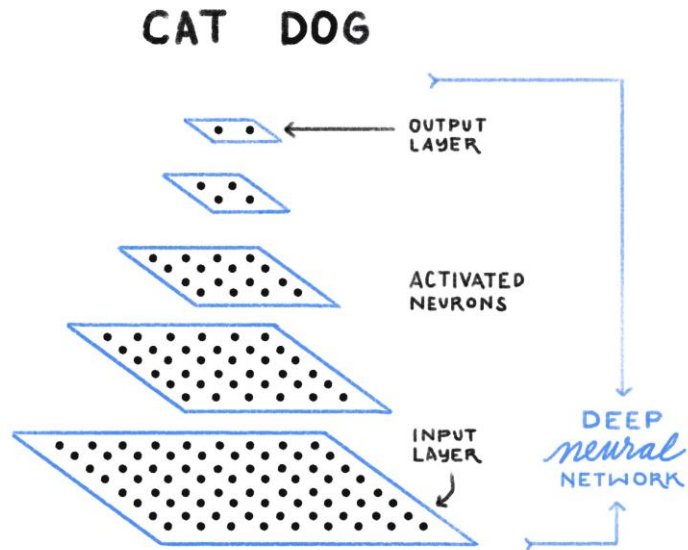
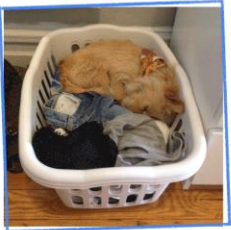
Curvas de aprendizaje en train y test

	Underfitting	Just right	Overfitting
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none"> • Complexify model • Add more features • Train longer 		<ul style="list-style-type: none"> • Perform regularization • Get more data



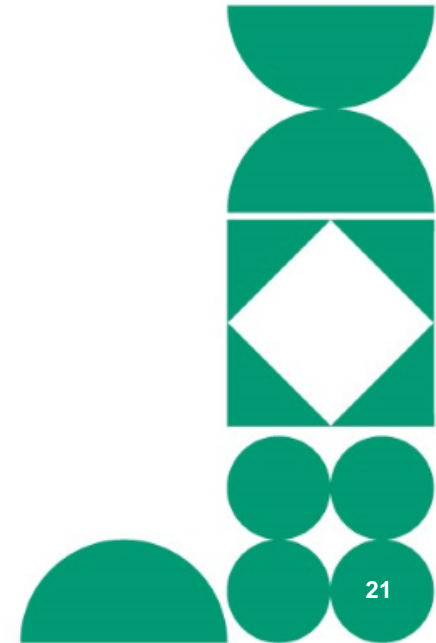
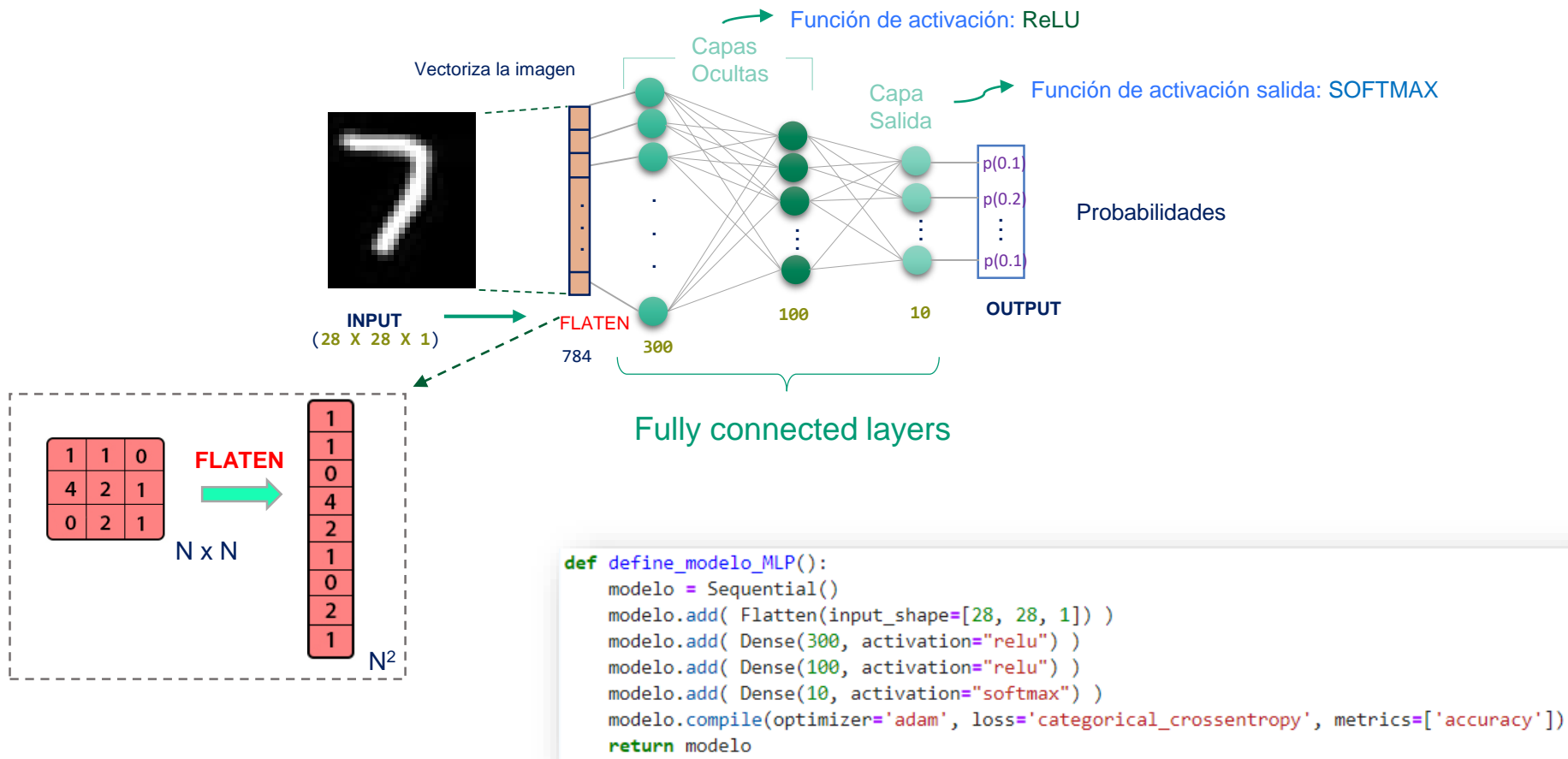
Deep Learning

IS THIS A
CAT or DOG?



Aplicación de MLP en clasificación de imágenes

Caso: identificación de dígitos escritos a mano



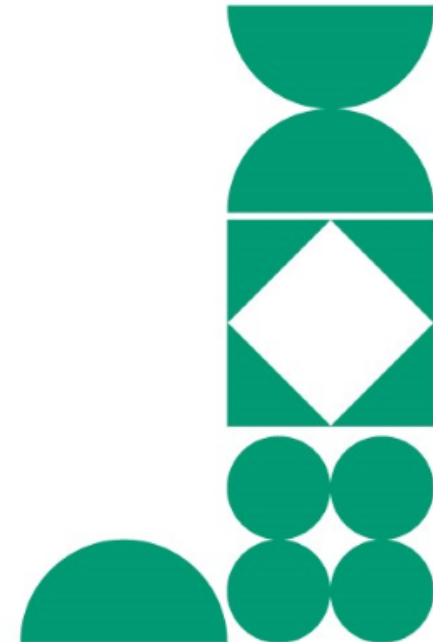
Bibliografía

- ❖ J. Watt and R. Borhani and A. Katsaggelos (2020). Machine Learning Refined: Foundations, Algorithms, and Applications. 2nd Edition. Cambridge: Cambridge University Press.
- ❖ C. Bishop (2006). Pattern Recognition and Machine Learning. Springer, New York.
- ❖ S. Raschka & V. Mirjalili (2019). *Python Machine Learning*. Third Edition. California: O'Reilly Media.

Sugerencia de links interesantes:

DERIVADAS - Clase Completa: Explicación Desde Cero
https://www.youtube.com/watch?v=_6-zwdrqD3U

DERIVADAS: Las Famosas Reglas EXPLICADAS
<https://www.youtube.com/watch?v=O6PeN5SJxzk>



¡Gracias!

