

Inteligencia Artificial
Primer Semestre 2025



SVM

Máquinas de Soporte Vectorial

Contenido

En el vasto mundo del aprendizaje automático, donde los modelos se enfrentan al desafío de clasificar datos complejos o realizar predicciones precisas, las Máquinas de Vectores de Soporte (SVM) se erigen como una herramienta elegante y poderosa. Este algoritmo no solo busca trazar la línea que mejor separa los datos, sino que lo hace maximizando el margen entre las clases, garantizando un enfoque robusto y eficiente.

1 Introducción

¿Por qué necesitamos SVM?
Comparación con regresión logística.

2 Fundamentos

Concepto de hiperplano de separación.
Márgenes y soporte vectorial.
Función de costo y optimización en SVM.

3 Clasificación para modelos linealmente no separables

Kernel Trick y Transformación de Espacios.

SVM

Un Support Vector Machine (SVM) es un modelo de aprendizaje supervisado que se utiliza para resolver problemas de clasificación y regresión. En particular, en clasificación, SVM busca una hiperplano que separe las clases de manera óptima.

1 Eficiencia en Espacios de Alta Dimensión

Funciona bien con datos donde el número de características es mayor que el número de muestras.

3 Robustez ante Sobreajuste

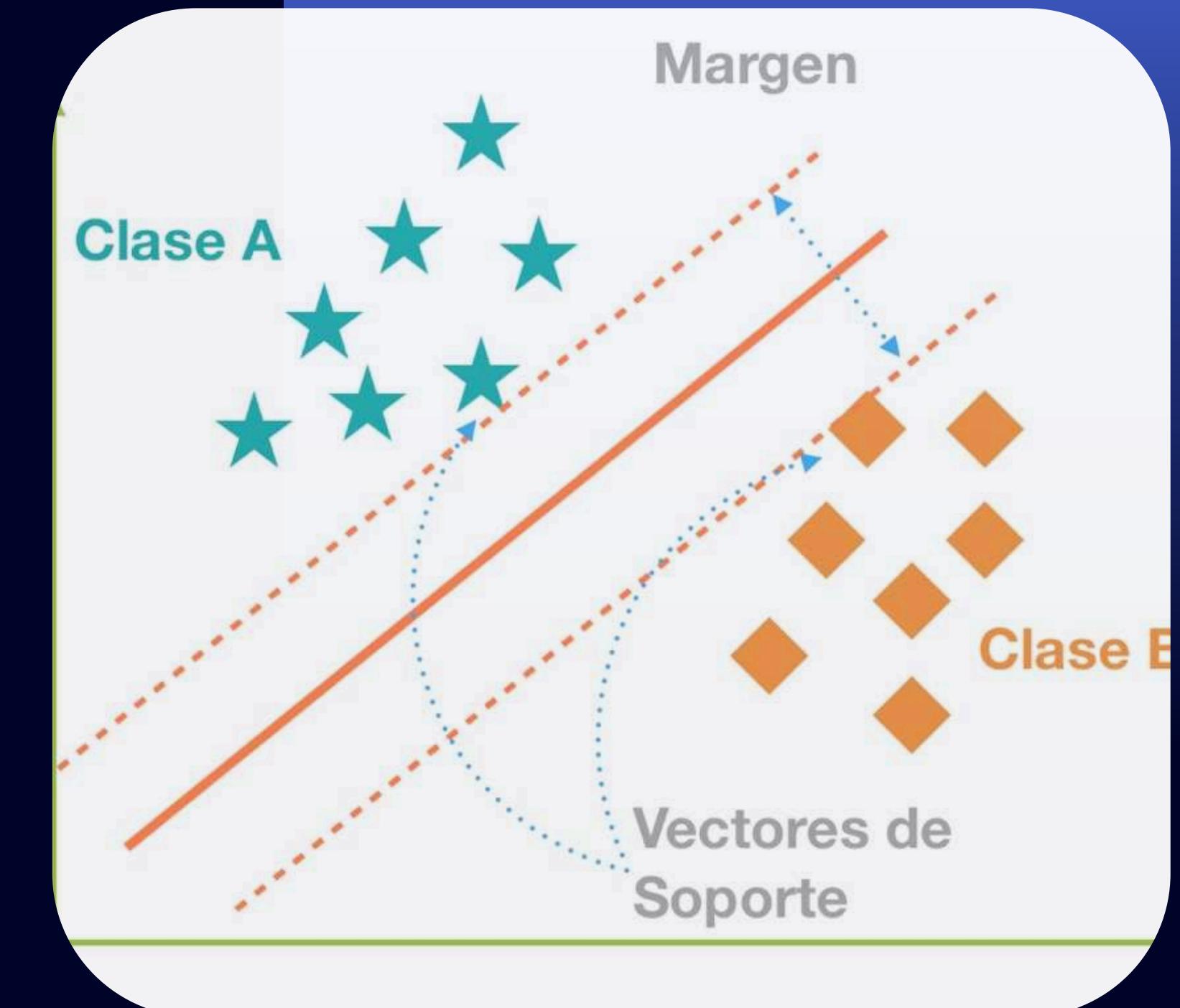
Con un buen ajuste de hiperparámetros (C y γ), evita el sobreajuste en datasets pequeños o ruidosos.

2 Uso de Kernels para Datos No Lineales

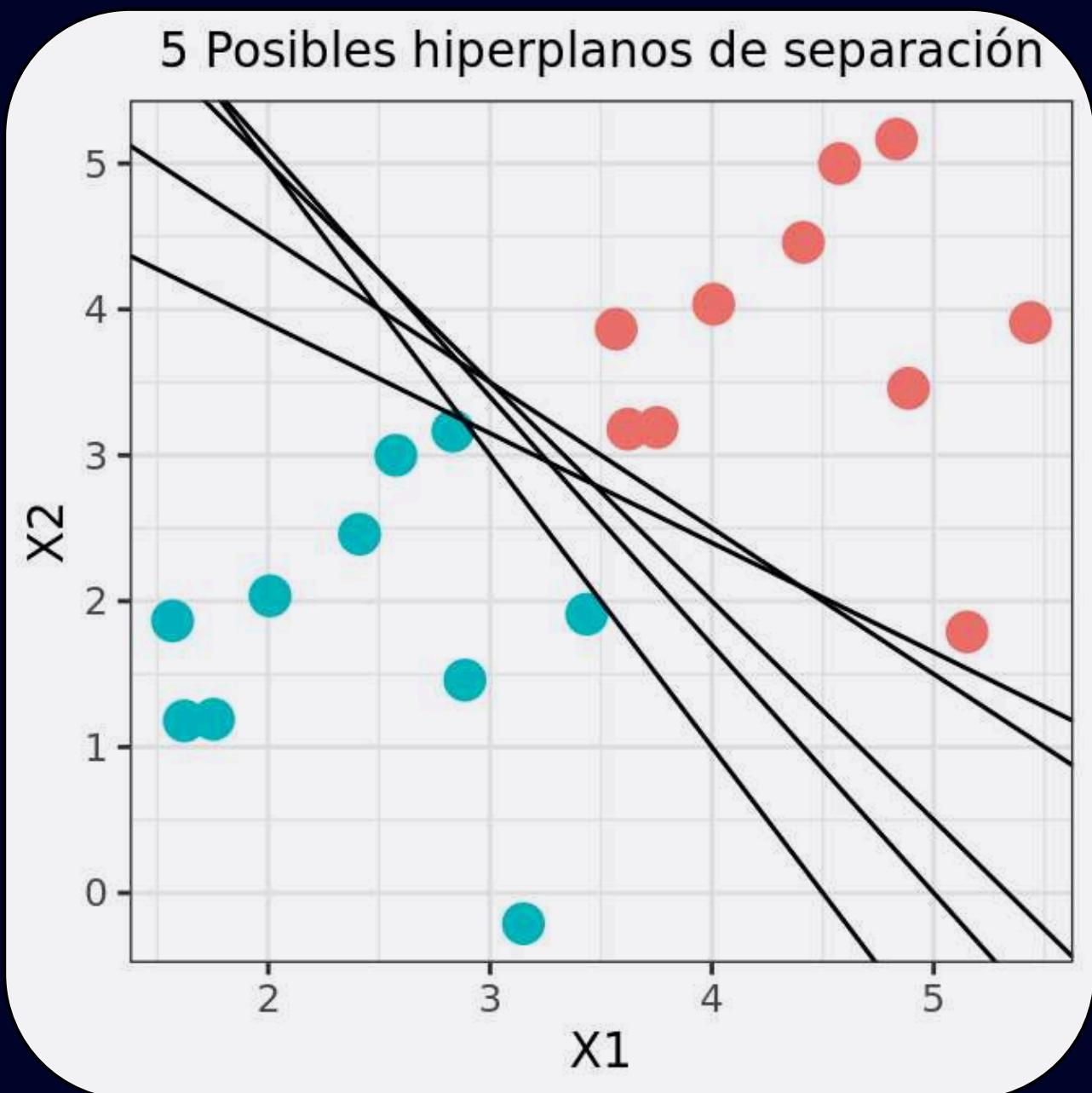
Puede transformar datos no linealmente separables en espacios de mayor dimensión para hacerlos separables.

4 Buena Generalización

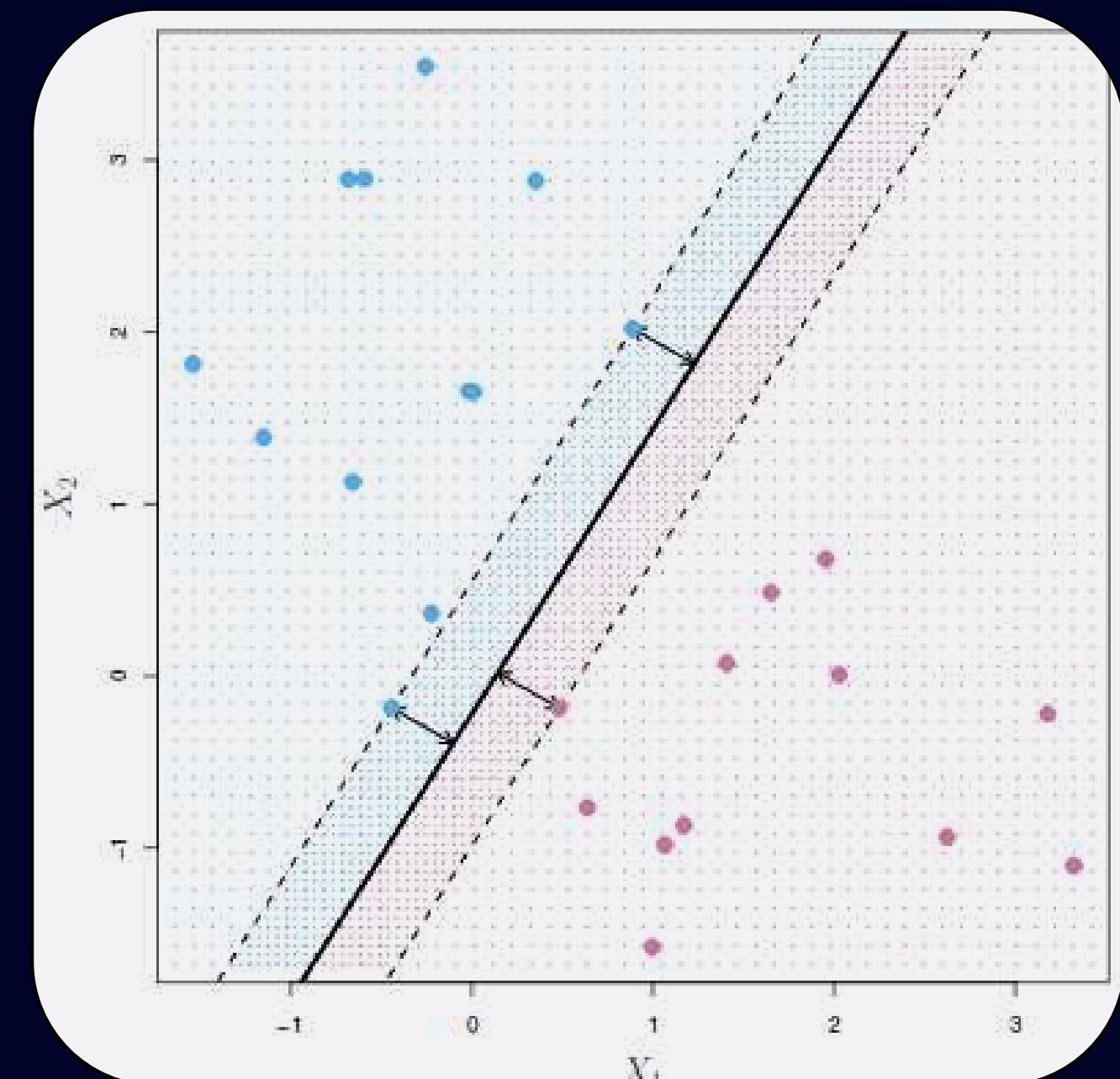
Maximiza el margen de separación entre clases, lo que mejora su capacidad de generalización en nuevos datos.



Regresión vs SVM



Una regresión Lineal encuentra muchos hiperplanos de separación



Un SVM encuentra el hiperplano de separación óptimo

Característica	SVM	Regresión Logística
Tipo de Modelo	Clasificador basado en márgenes	Clasificador basado en probabilidad
Función de Decisión	Encuentra el hiperplano con el mayor margen entre clases.	Modela la probabilidad de pertenencia a una clase.
Manejo de No Linealidad	Puede usar trucos de kernel para transformar datos no lineales.	Solo funciona bien en problemas linealmente separables (a menos que usemos transformaciones de características).
Robustez ante Outliers	Más resistente gracias a la maximización del margen y a las slack variables.	Más sensible a outliers, ya que optimiza la verosimilitud.
Velocidad de Entrenamiento	Más lento en grandes volúmenes de datos, especialmente con kernels complejos.	Generalmente más rápido y eficiente en grandes datasets.
Interpretabilidad	Difícil de interpretar cuando se usan kernels.	Fácil de interpretar, ya que los coeficientes representan la importancia de cada variable.
Uso en Alta Dimensión	Funciona bien con datos de muchas dimensiones (ejemplo: texto, imágenes).	Puede sufrir en dimensiones altas si no se aplica regularización adecuada.

.Py

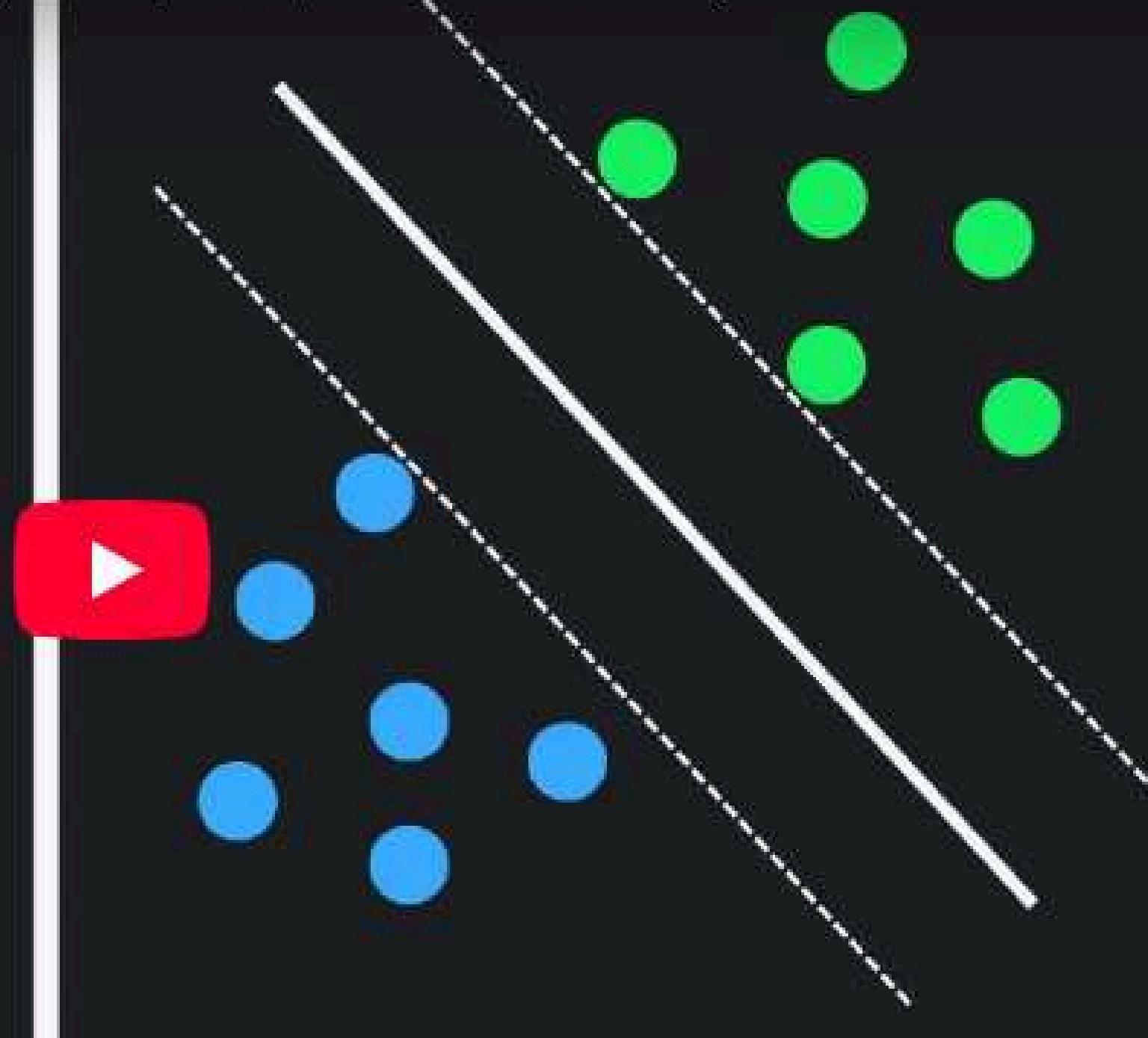
Aprende SVM (Support Vector Machines) con Python | Machine Learning 101



Share

SVM

Support Vector Machines



Watch on YouTube

◆ ¿Cuándo usar SVM?

- Cuando los datos no son linealmente separables y queremos usar kernels.
- Cuando hay pocos datos y muchas características (ejemplo: bioinformática, texto).

◆ ¿Cuándo usar Regresión Logística?

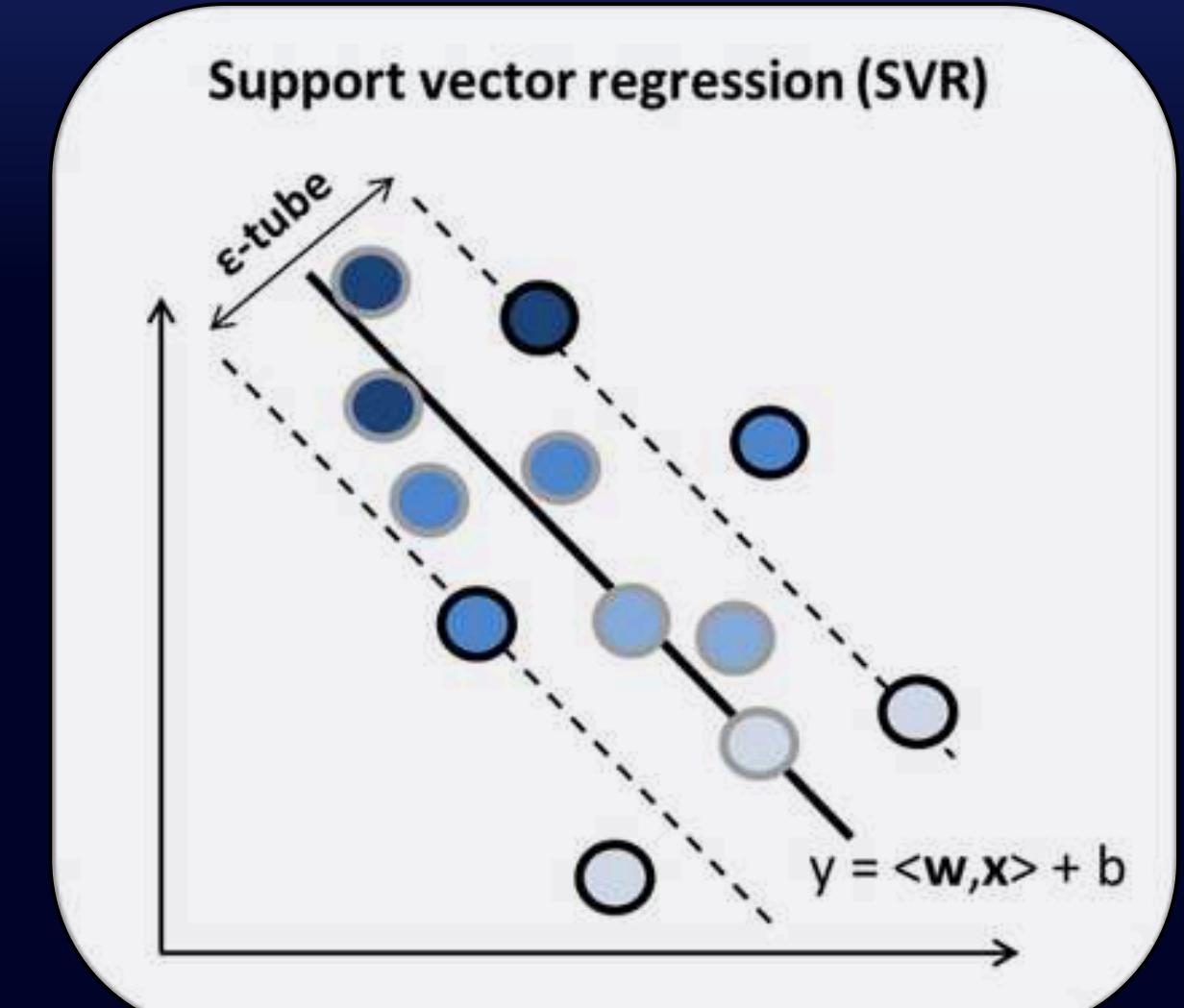
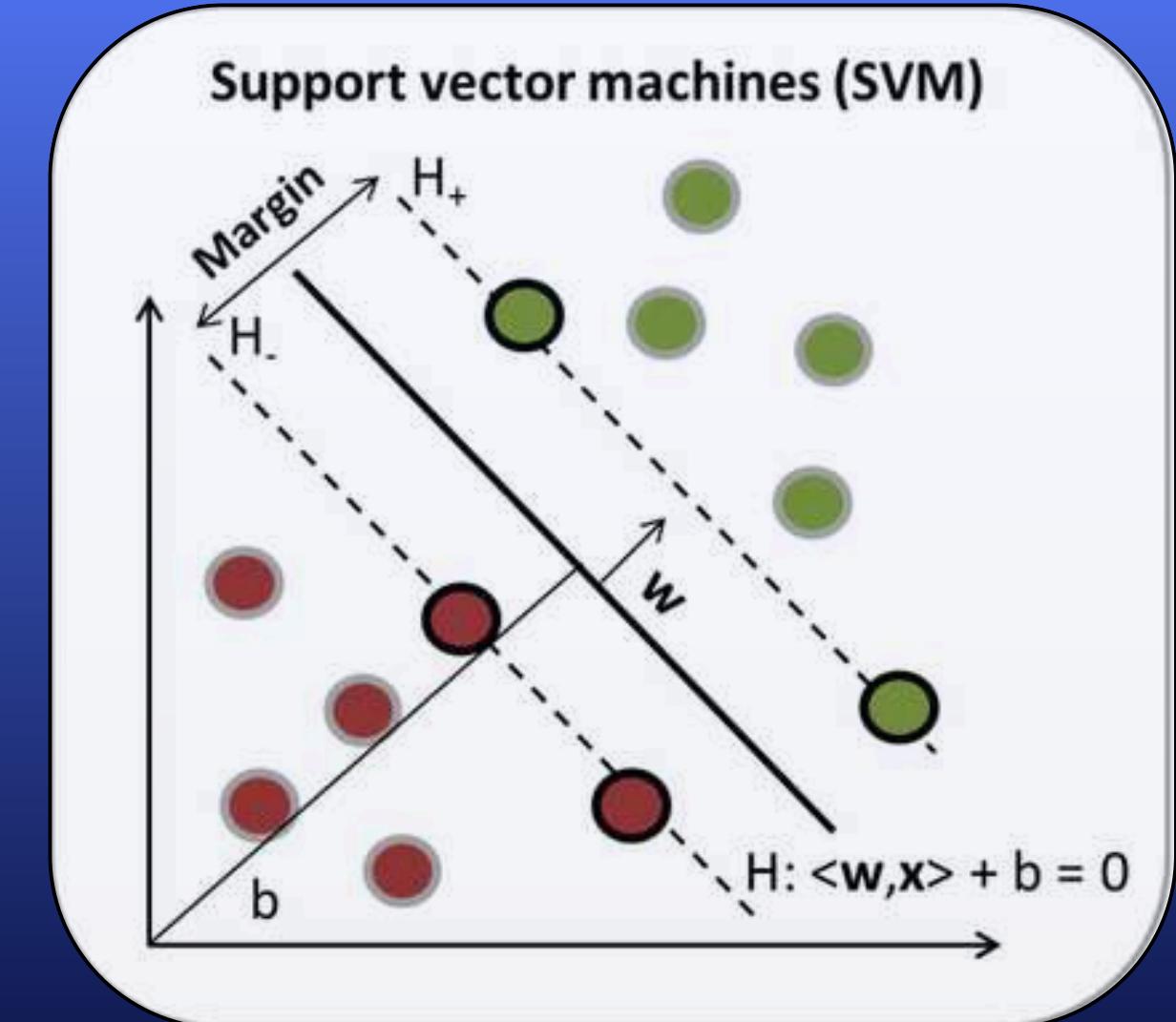
- Cuando queremos una interpretación clara de los coeficientes.
- Cuando los datos son linealmente separables o queremos una probabilidad de clasificación.

Si el problema es lineal y queremos rapidez e interpretabilidad, usamos Regresión Logística.

Si el problema es más complejo, con patrones no lineales, SVM puede ser la mejor opción gracias a los kernels.

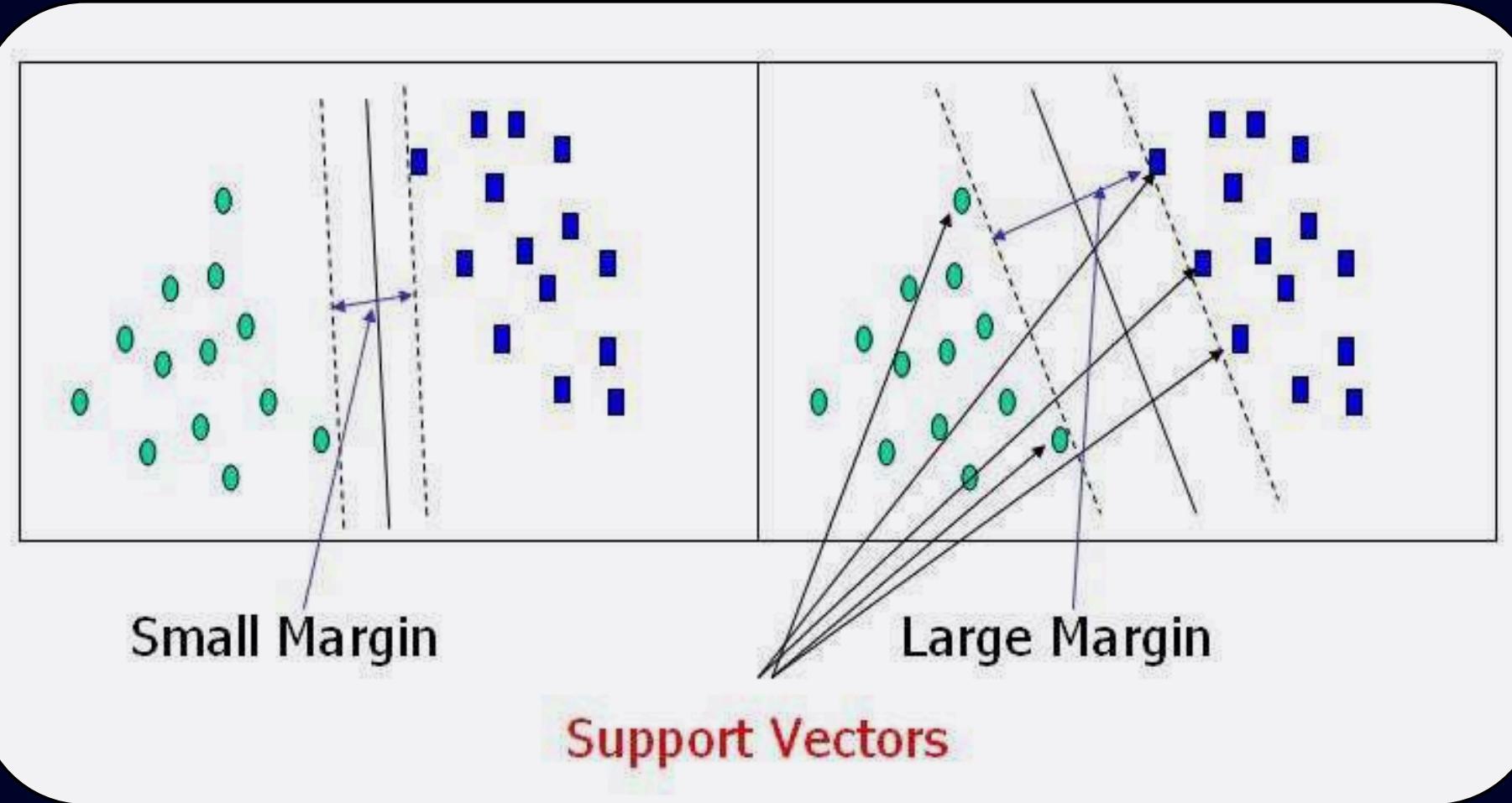


Característica	SVM (Support Vector Machine)	SVR (Support Vector Regression)
Propósito	Clasificación (asigna etiquetas a clases)	Regresión (predice valores continuos)
Función Objetivo	Encuentra el hiperplano óptimo con máximo margen para separar clases.	Encuentra un hiperplano óptimo que minimice el error dentro de un margen de tolerancia (ϵ \epsilon).
Manejo de Datos No Lineales	Usa kernels para transformar datos no lineales en un espacio de mayor dimensión.	También usa kernels para modelar relaciones no lineales en regresión.
Concepto Clave	Maximización del margen entre clases.	Minimización del error dentro de una tolerancia ϵ \epsilon (tubo de regresión).
Ejemplo de Uso	Detección de spam, reconocimiento facial, clasificación de imágenes.	Predicción de precios de casas, análisis financiero, series de tiempo.



Máxima Marginalidad en SVM

El margen es la distancia entre el hiperplano óptimo y los puntos más cercanos de cada clase. En SVM, buscamos maximizar este margen para mejorar la capacidad de generalización del modelo.



1 Soporte Vectorial

Los vectores de soporte son los puntos de datos más cercanos al hiperplano de decisión.

2 Hiperplano Óptimo

Esta condición garantiza que los vectores de soporte estén en los bordes del margen y que los datos estén correctamente clasificados dentro del margen máximo.

3 Función de costo

Encuentra un hiperplano óptimo que maximice el margen entre clases y minimice los errores de clasificación.

SVM Conceptos



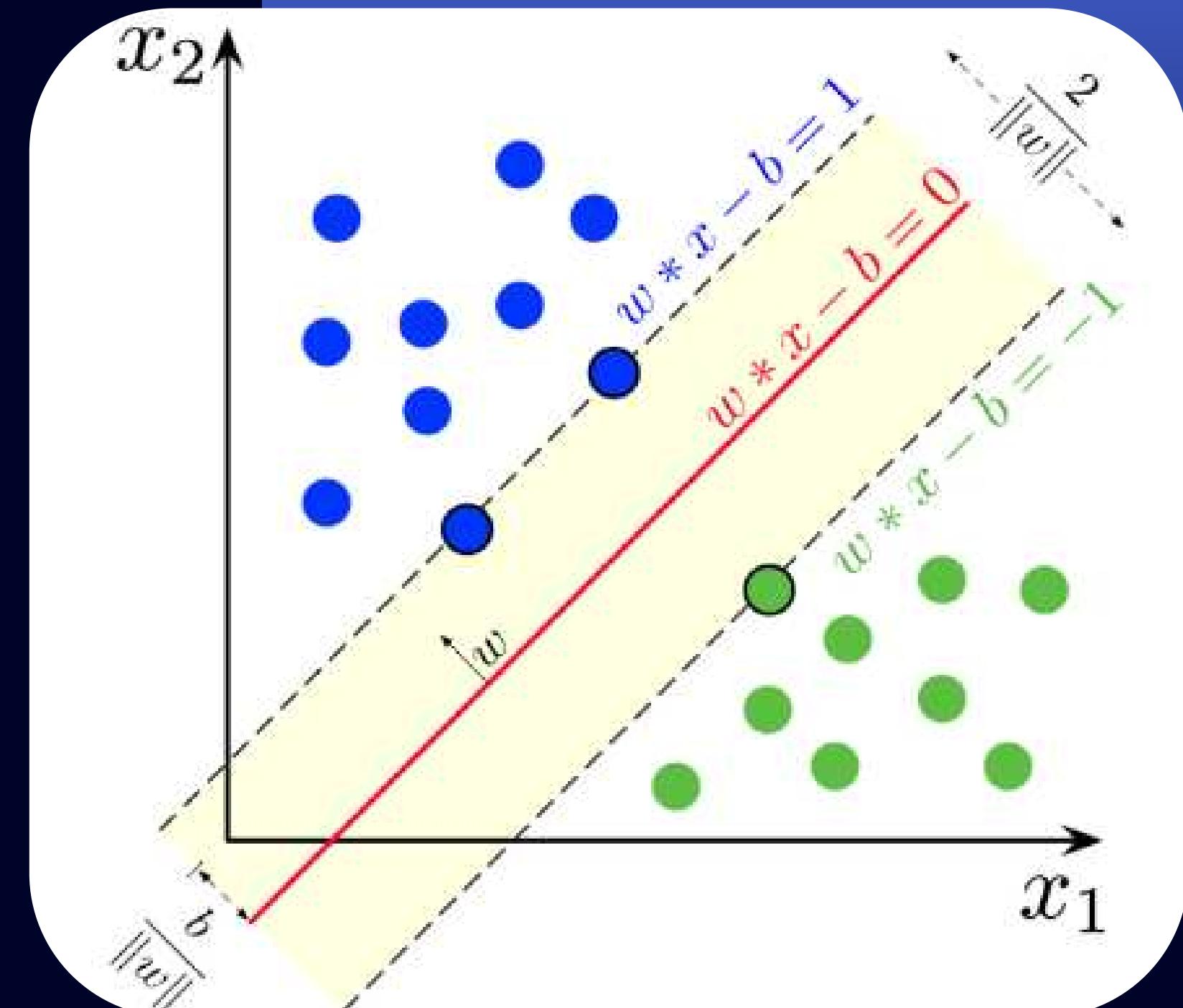
Hiperplano

Un hiperplano en un espacio de n dimensiones es una ecuación de la forma:

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

El vector \mathbf{x} representa un punto en el espacio de entrada, es simplemente el conjunto de datos de entrada, con n dimensiones:

$$\mathbf{x} = (x_1, x_2, \dots, x_n)$$



norma euclídea

$$\|(x_1, \dots, x_n)\| = \sqrt{x_1^2 + \dots + x_n^2}$$

SVM Conceptos



Hiperplano

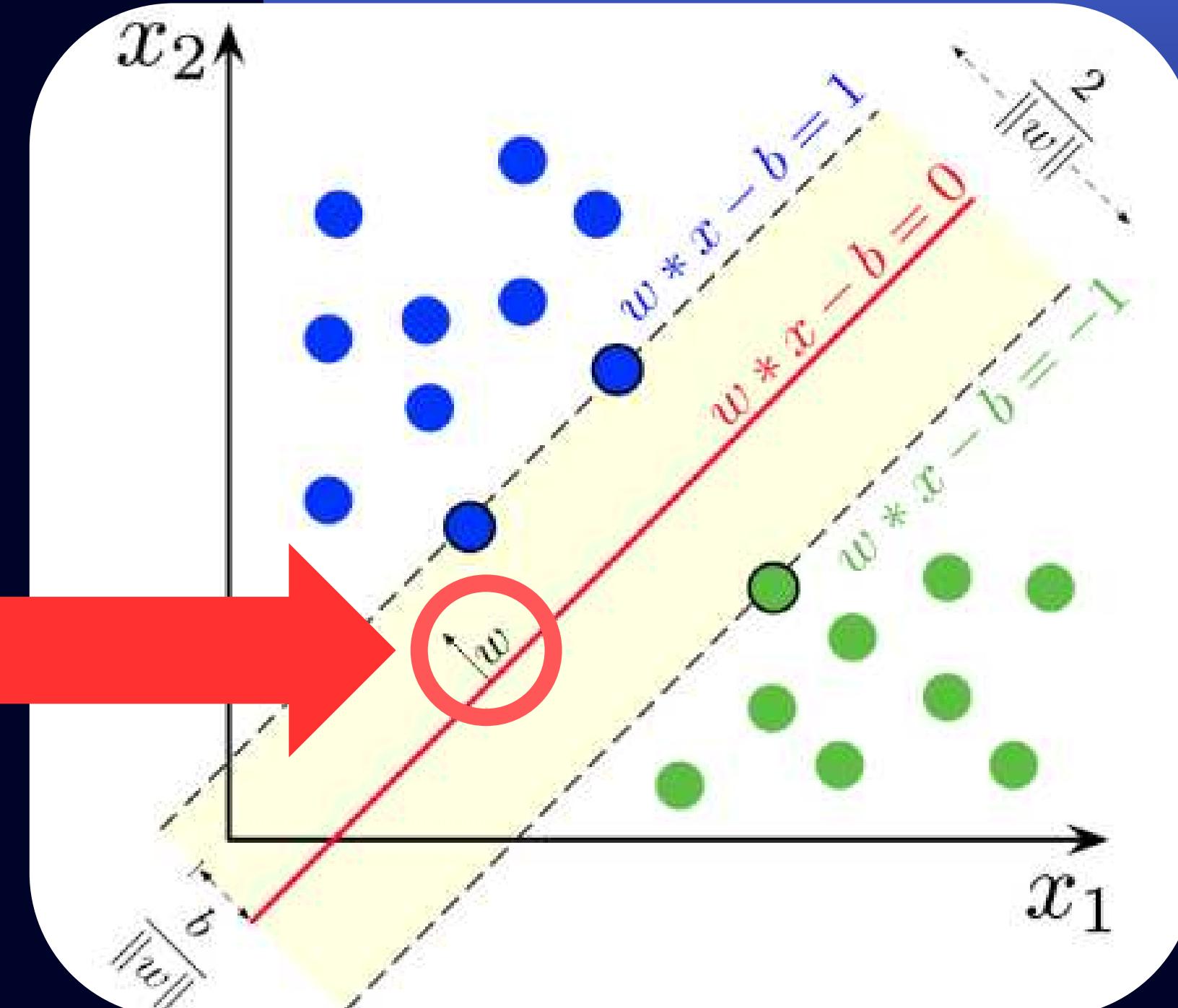
El vector de pesos \mathbf{w} es un vector normal (perpendicular) al hiperplano de separación. Define la dirección y orientación del hiperplano en el espacio.

$$\mathbf{w} = (w_1, w_2, \dots, w_n)$$

Se obtiene resolviendo un problema de optimización en SVM, busca minimizar la norma $\|\mathbf{w}\|$ (norma del vector) mientras mantiene las restricciones de clasificación.

Se usan métodos matemáticos como:

- Multiplicadores de Lagrange
- Método del gradiente



una norma en un espacio vectorial es un operador que permite definir una noción de "longitud" o "tamaño" de cualquier vector.

SVM Conceptos



Norma $\|w\|$

Si tienes un vector de pesos $w=(w_1, w_2, \dots, w_n)$, su norma Euclidiana se calcula así:

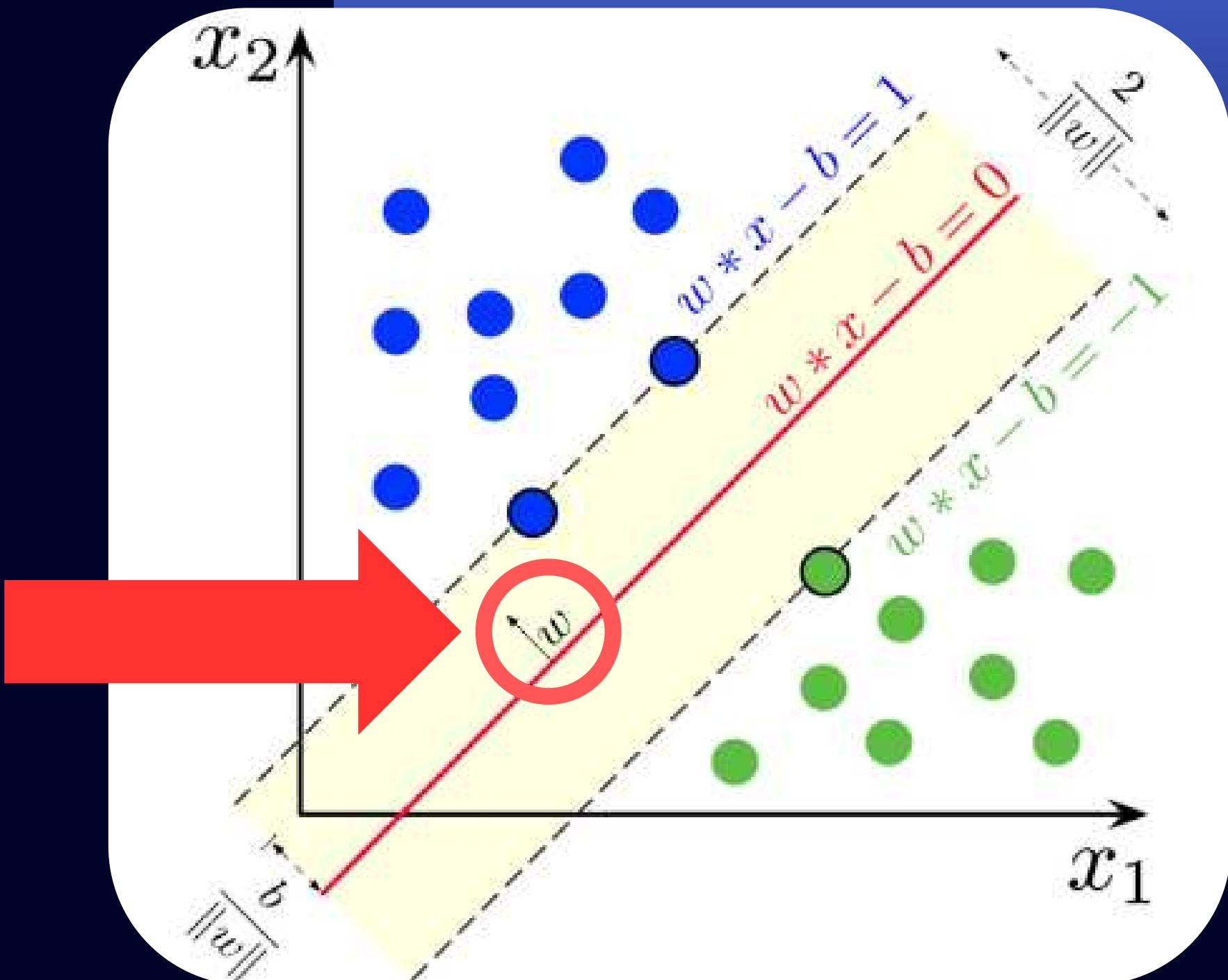
$$\|w\| = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$$

Supongamos que $w=(3,4)$

$$\|w\| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

norma euclíadiana

$$\|(x_1, \dots, x_n)\| = \sqrt{x_1^2 + \dots + x_n^2}$$



una norma en un espacio vectorial es un operador que permite definir una noción de "longitud" o "tamaño" de cualquier vector.

.PY

Gradiente Descendente desde cero con Python | Deep Learning 101

Gradiente Descendente

el algoritmo para entrenar una IA

Share



Watch on YouTube

SVM Conceptos



Hiperplano

El objetivo es encontrar el hiperplano que maximiza el margen.

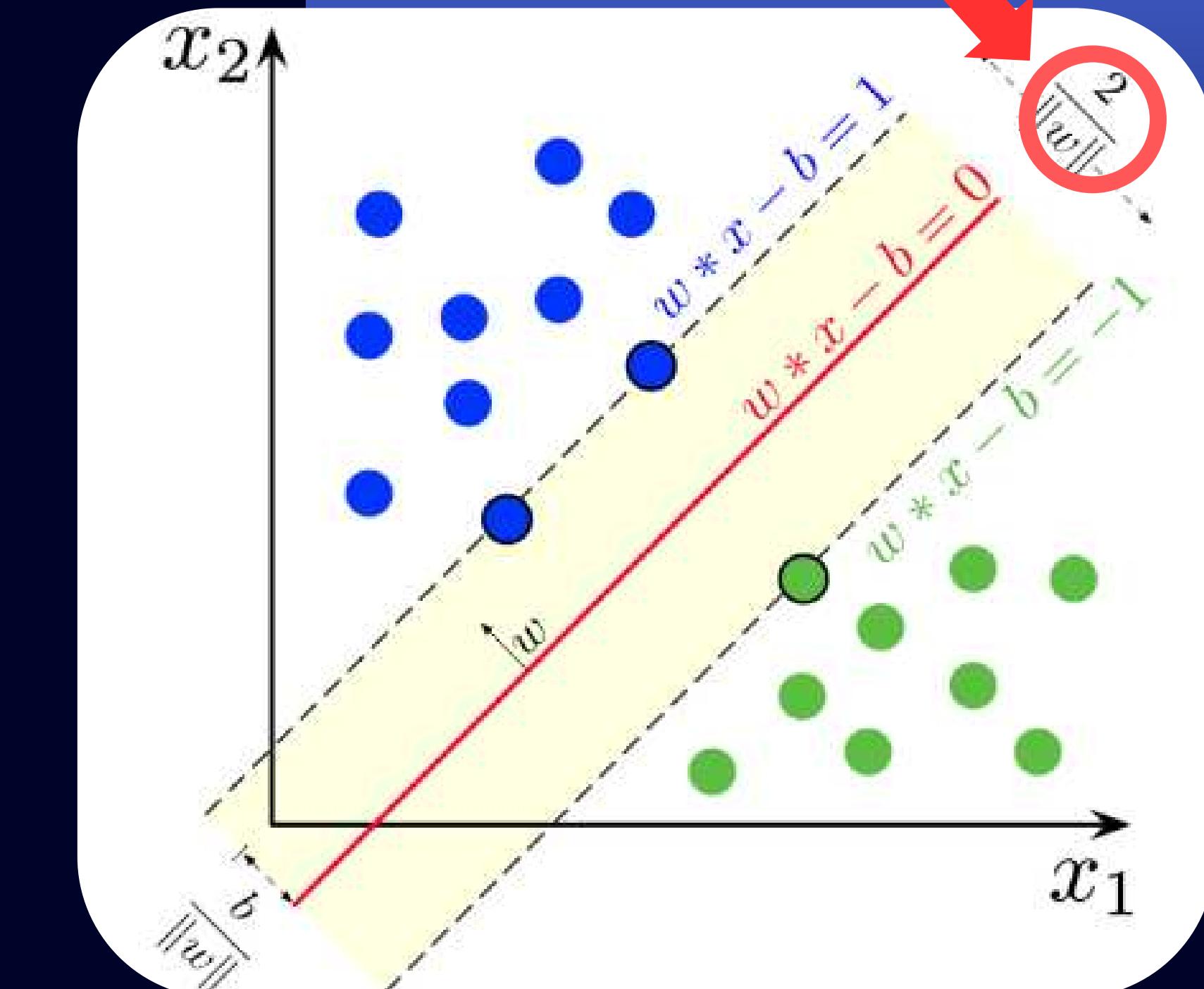
Para esto, se resuelve el siguiente problema de optimización:

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

Esto se hace porque:

- minimizar $\frac{\|w\|^2}{2}$ equivale a maximizar el margen
- Usar $\|w\|^2$ en lugar de $\|w\|$ facilita la optimización, ya que se evita la raíz cuadrada

maximizar el margen



SVM Conceptos



Restricciones

Los puntos más cercanos al hiperplano (vectores soporte) deben cumplir:

- $\mathbf{w} \cdot \mathbf{x}_+ + b = +1$ (para clase +1)
- $\mathbf{w} \cdot \mathbf{x}_- + b = -1$ (para clase -1)

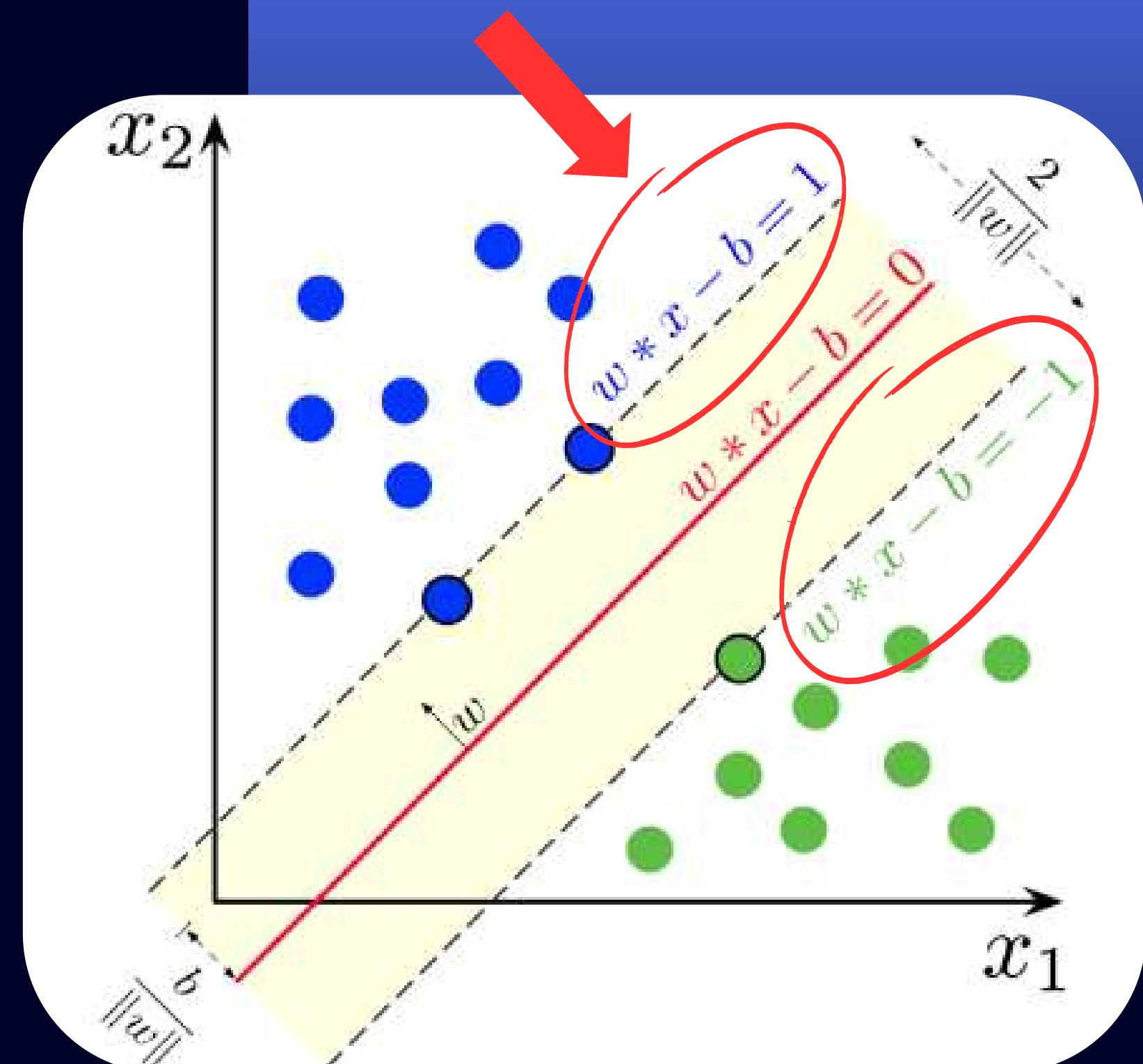
Queremos que todos los puntos cumplan:

- Si $y_i = +1$: $\mathbf{w} \cdot \mathbf{x}_i + b \geq +1$
- Si $y_i = -1$: $\mathbf{w} \cdot \mathbf{x}_i + b \leq -1$

Esto se puede compactar en una sola restricción:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i$$

Restricciones

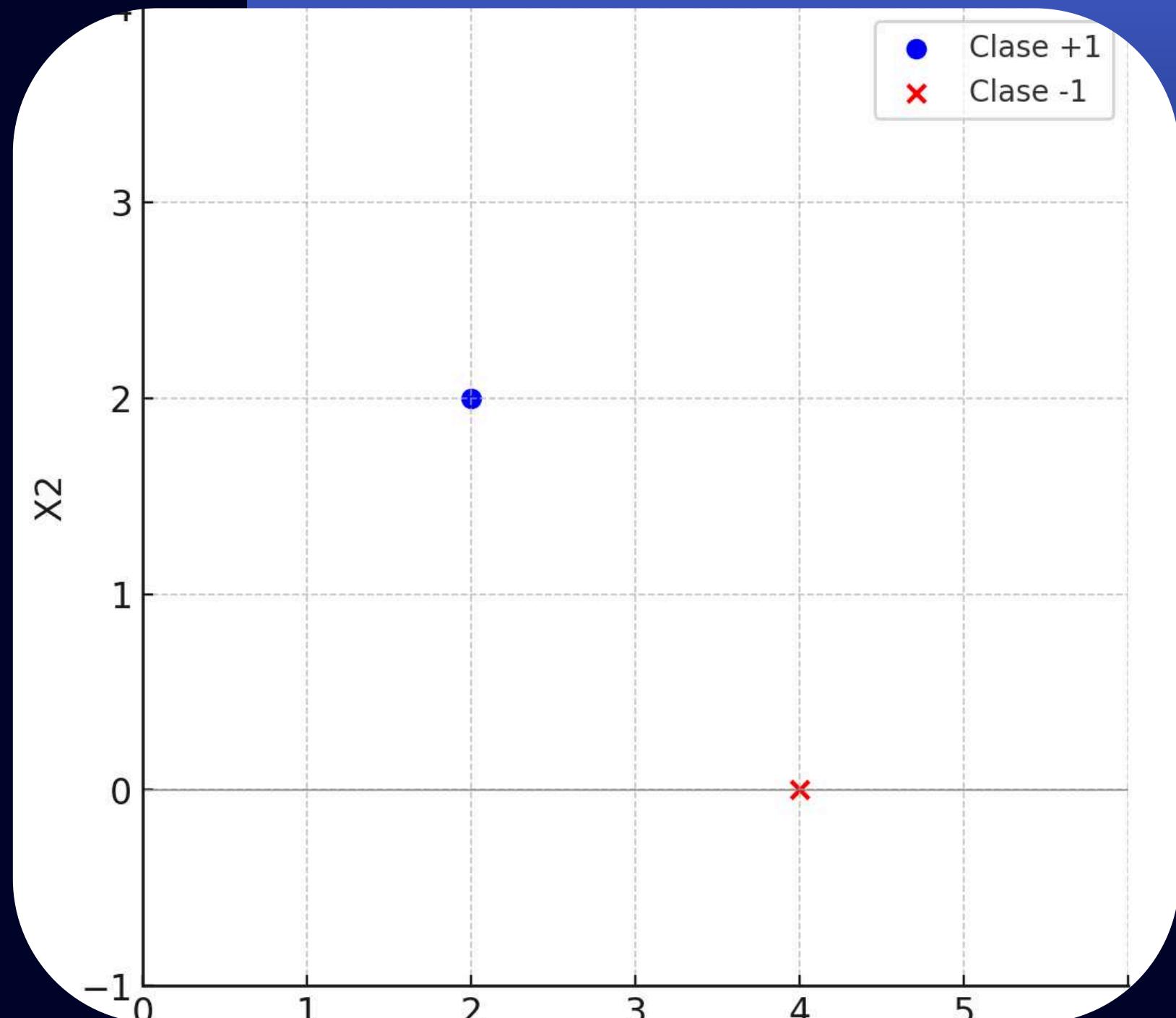


SVM Conceptos

- Ejemplo Demostrativo - Escogiendo puntos para Vectores de Soporte

X_1	X_2	Y
2	2	1
4	0	-1

Entonces, tomando en consideracion la ecuacion: $y_i(\underline{w}x_i + b) \geq 1$, que representa la restriccción de optimización que asegura que todos los puntos de entrenamiento queden más allá o exactamente sobre sus líneas de soporte correspondientes



SVM Conceptos



- Ejemplo Demostrativo - Plantear las restricciones de optimización

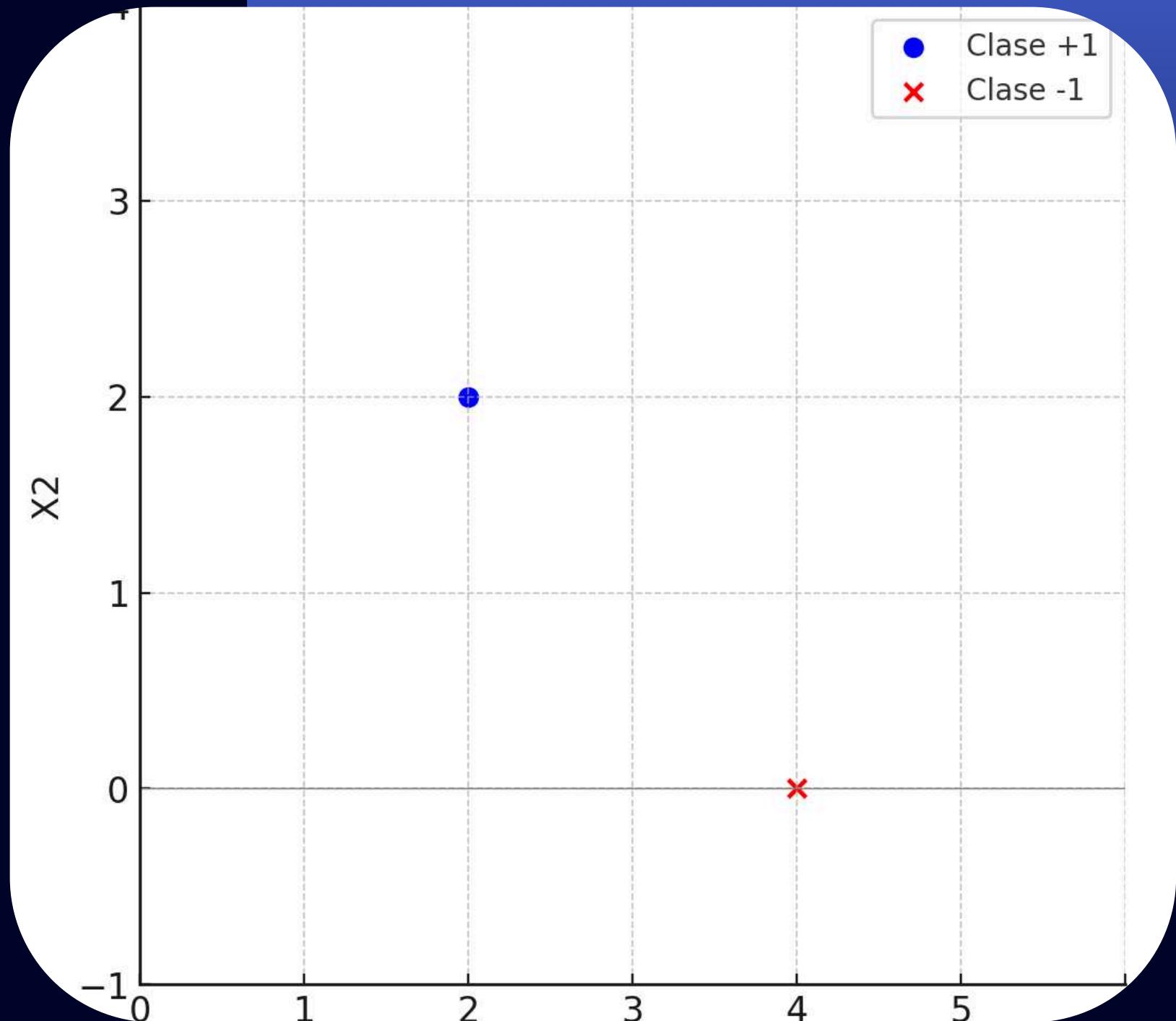
Si $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ entonces usando los puntos para vectores de soporte, definimos las restricciones:

Para $x_1=2, x_2=2, y=+1$

- $(2w_1 + 2w_2 + b) \geq 1$

Para $x_1=4, x_2=0, y=-1$

- $-(4w_1 + 0w_2 + b) \geq 1 \quad 4w_1 + b \leq -1$



SVM Conceptos



- Ejemplo Demostrativo - Resolver el sistema de ecuaciones

Dadas las restricciones anteriores, los vectores de soporte son los que cumplen con la igualdad:

- $2w_1 + 2w_2 + b = 1$
- $4w_1 + b = -1$

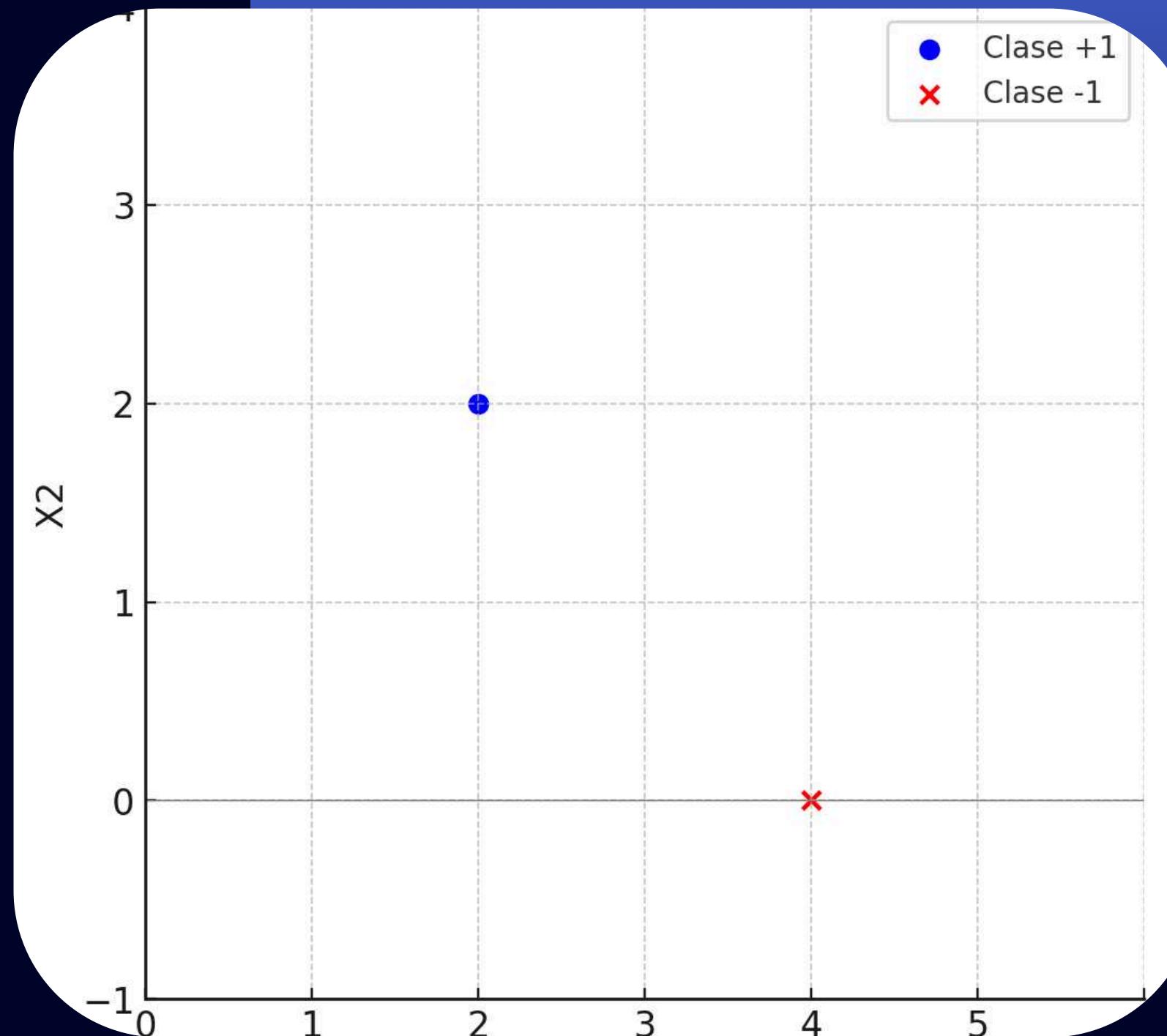
Estas ecuaciones son un sistema de 2 ecuaciones con 3 incógnitas (w_1, w_2, b)

De la segunda ecuación:

- $b = -1 - 4w_1$

Sustituimos en la primera:

$$2w_1 + 2w_2 + (-1 - 4w_1) = 1$$



SVM Conceptos



Ejemplo Demostrativo - Resolver el sistema de ecuaciones

$$2w_1 + 2w_2 + (-1 - 4w_1) = 1$$

Entonces

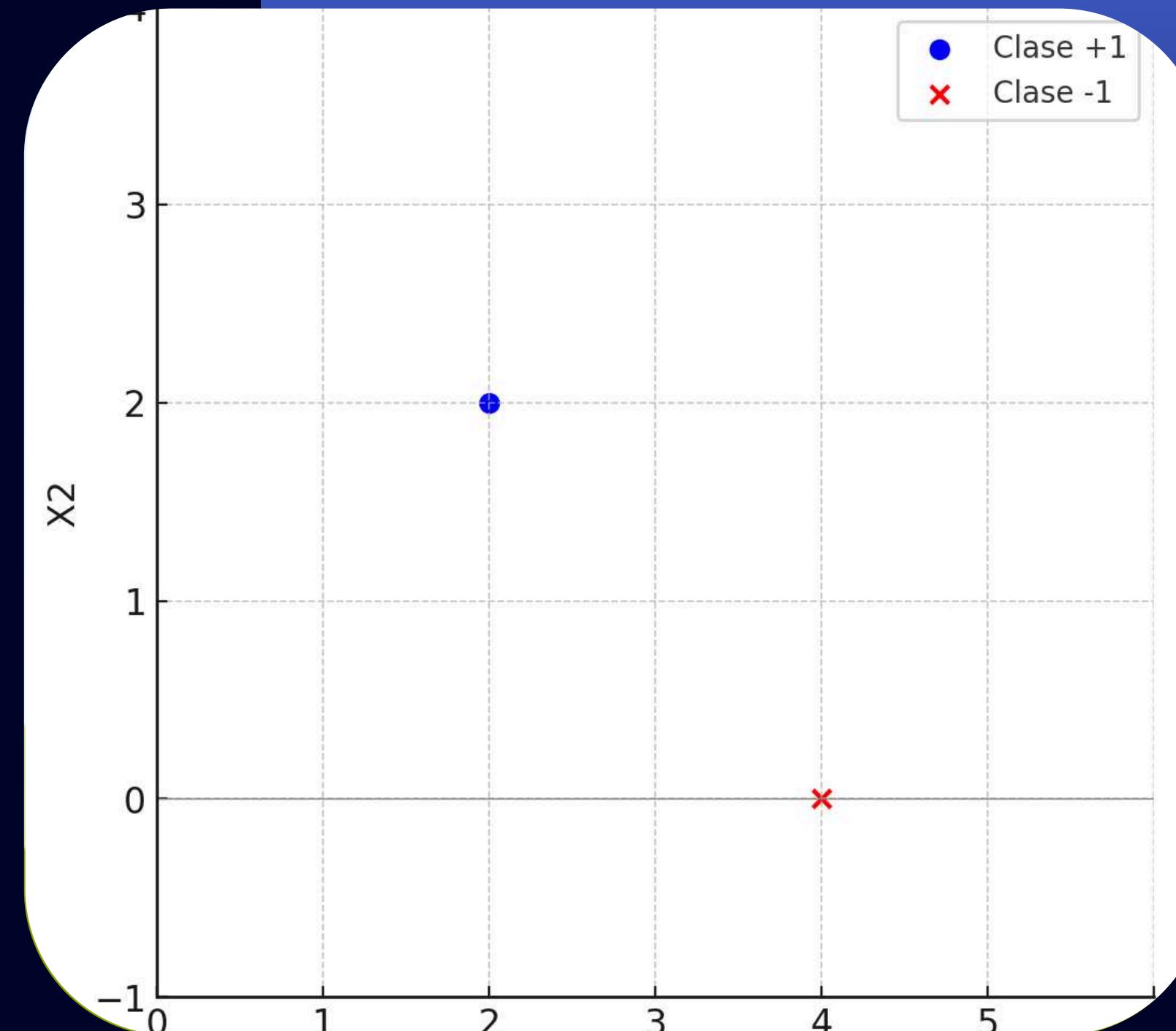
$$\begin{aligned} -2w_1 + 2w_2 - 1 &= 1 \\ -2w_1 + 2w_2 &= 2 \quad -w_1 + w_2 &= 1 \quad w_2 = w_1 + 1 \end{aligned}$$

Expresamos la norma:

Queremos minimizar $\|\mathbf{w}\|^2 = w_1^2 + w_2^2$

Reemplazamos w_2 en $\|\mathbf{w}\|^2 = w_1^2 + (w_1 + 1)^2$

Expandimos: $w_1^2 + (w_1^2 + 2w_1 + 1) \quad 2w_1^2 + 2w_1 + 1$



SVM Conceptos

Ejemplo Demostrativo - Optimizar

Obtuvimos $\|\mathbf{w}\|^2$: $2w_1^2 + 2w_1 + 1$

Para minimizar esta función cuadrática, necesitamos derivar respecto a w_1 :

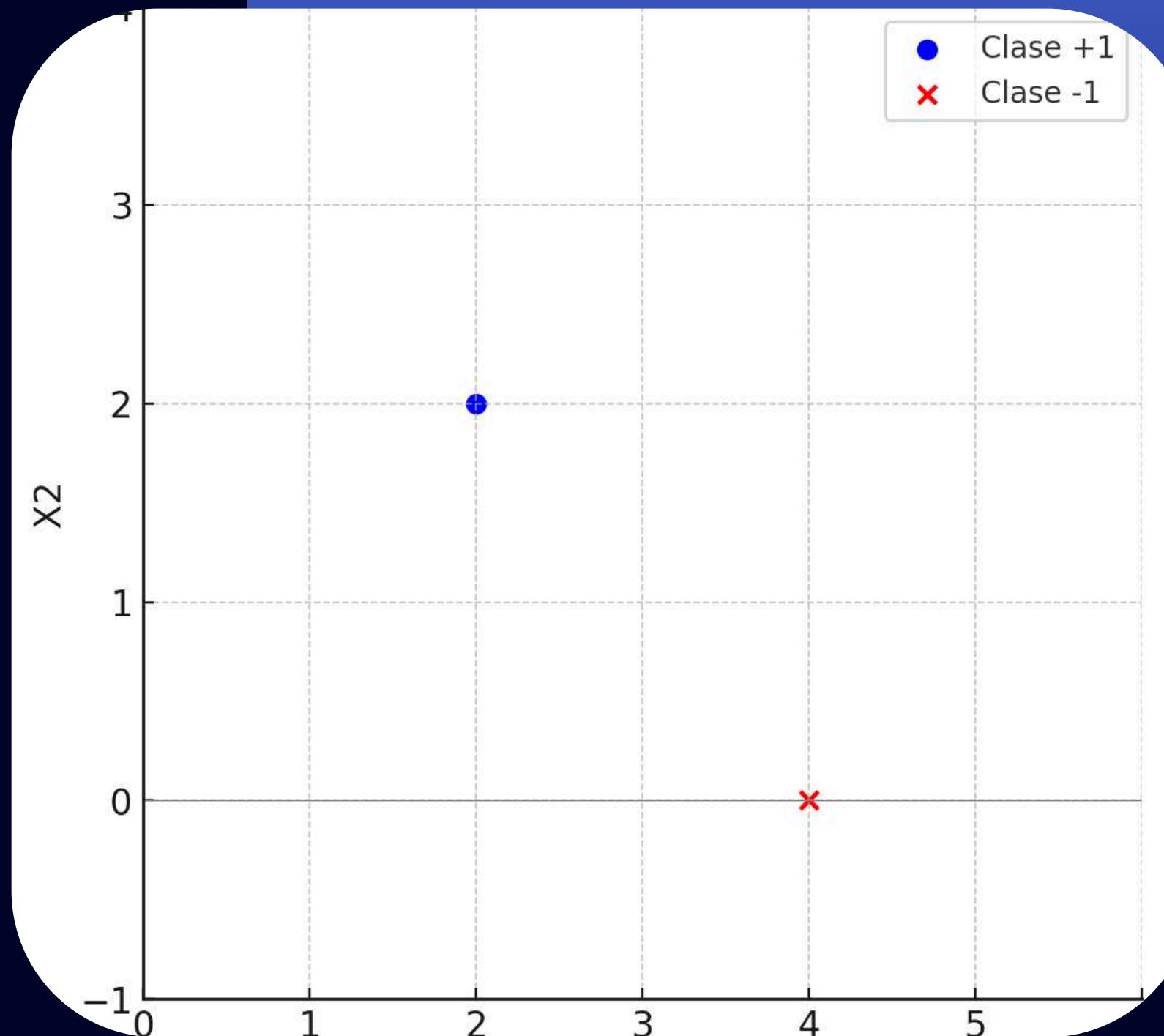
$$\frac{d}{dw_1} (2w_1^2 + 2w_1 + 1) = 4w_1 + 2$$

e igualamos a cero, para encontrar el mínimo de w_1 :

- $4w_1 + 2 = 0 \quad w_1 = -1/2$

Dado w_1 , calculamos w_2 :

$$w_2 = w_1 + 1 = (-1/2) + 1 = 1/2$$



SVM Conceptos



Sesgo

El sesgo b es un escalar que ajusta la posición del hiperplano.

Si $b=0$, el hiperplano pasa por el origen.

Si $b \neq 0$, el hiperplano se desplaza.

Se calcula después de obtener el vector de pesos w

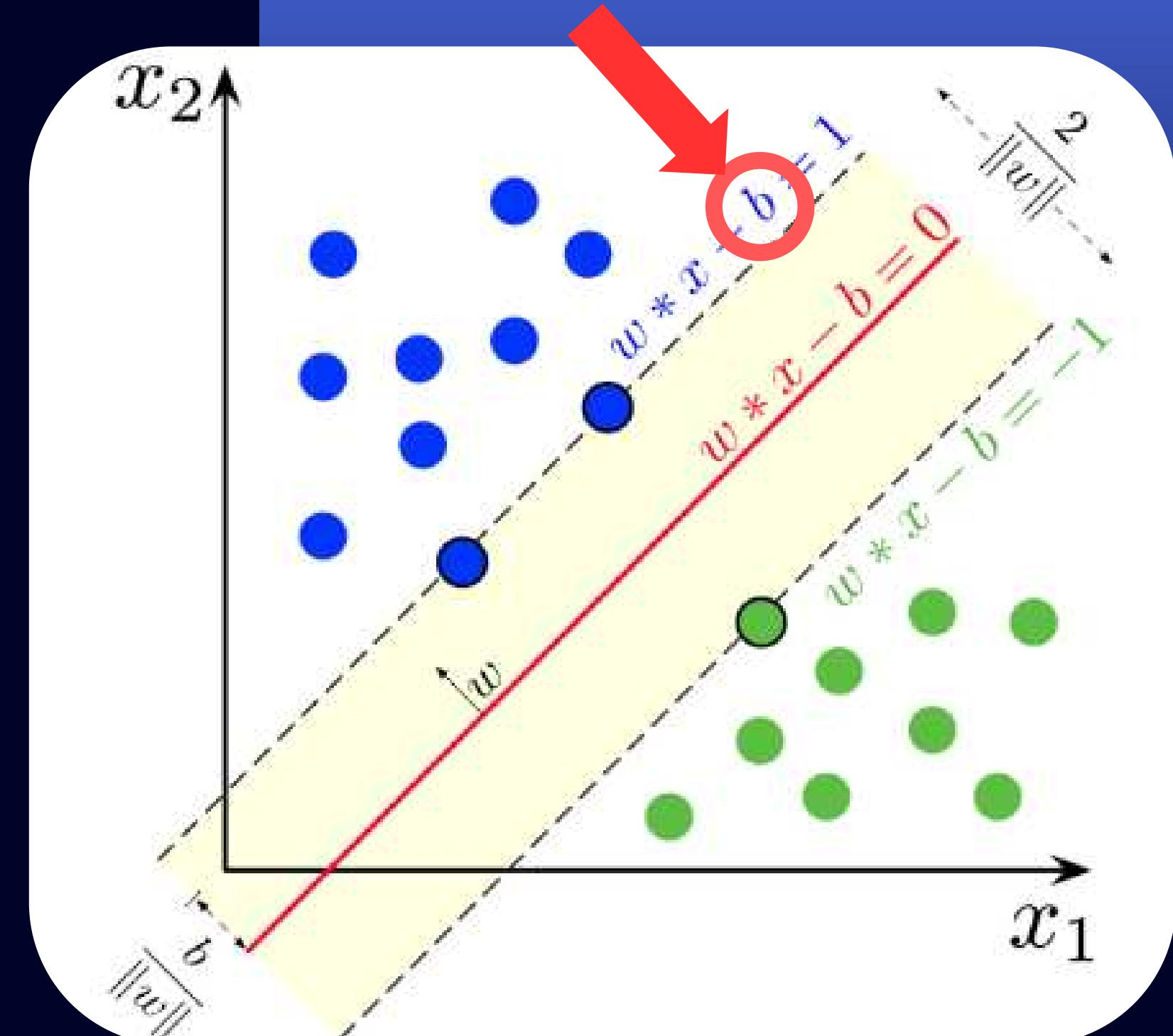
Para calcular b :

1. Se obtiene el vector de pesos w
2. Se elige un vector de soporte X_i
3. Se usa la fórmula:

$$a. b = Y_i - w \cdot X_i$$

donde Y_i es la etiqueta de la clase

Sesgo o Bias



SVM Conceptos

Ejemplo Demostrativo - Determinar el sesgo

Finalmente, sustituimos en b:

$$b = -1 - 4w_1 = -1 - 4(-1/2) = -1 + 2 = \underline{1}$$

Entonces la ecuación del hiperplano es:

$$\underline{w_1x_1 + w_2x_2 + b = 0}$$

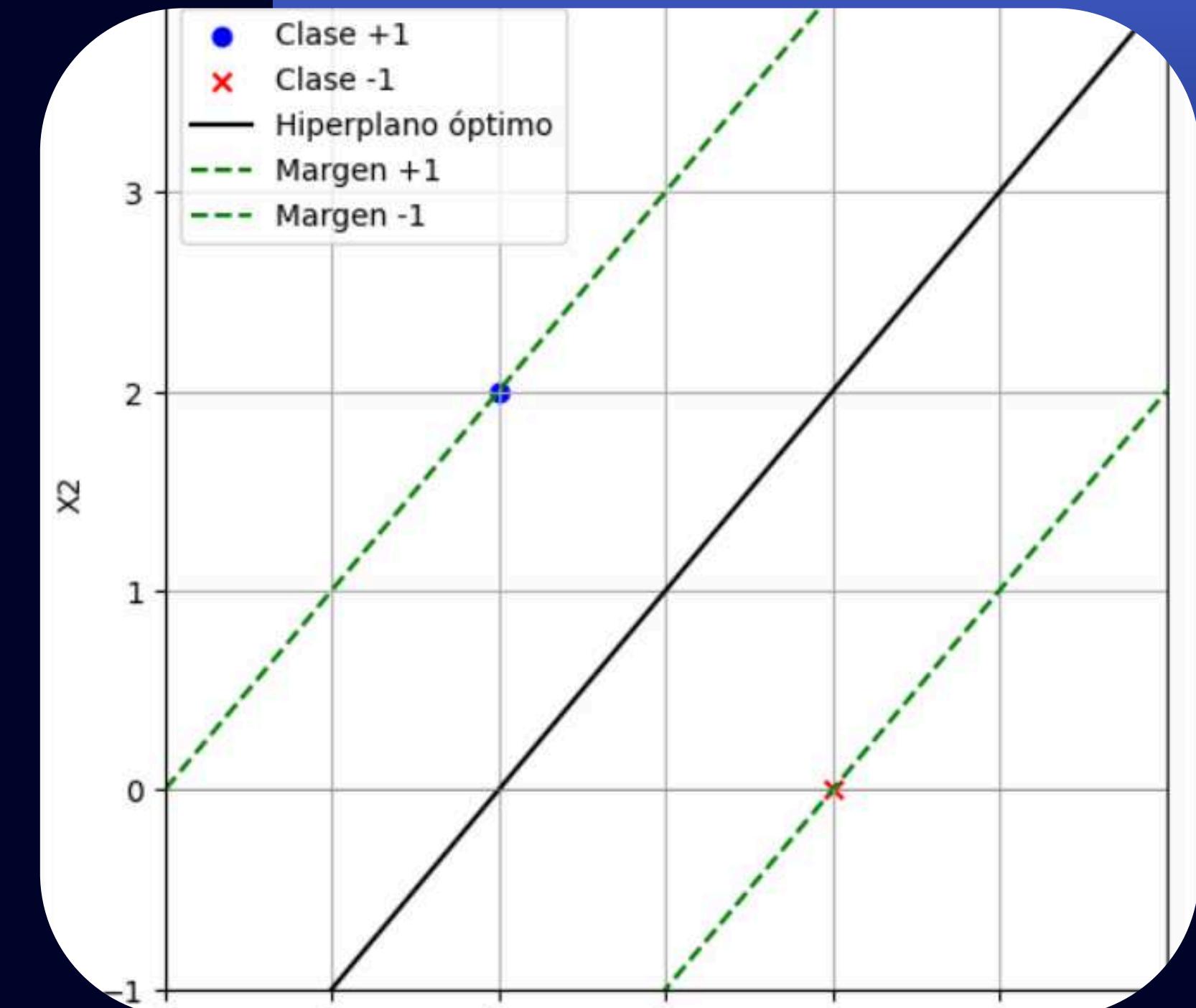
$$\underline{(-1/2)x_1 + (1/2)x_2 + 1 = 0}$$

La norma de w es:

$$\|w\| = \sqrt{\left(-\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2} = \sqrt{\frac{1}{4} + \frac{1}{4}} = \sqrt{\frac{1}{2}} = \frac{1}{\sqrt{2}}$$

El ancho del margen es:

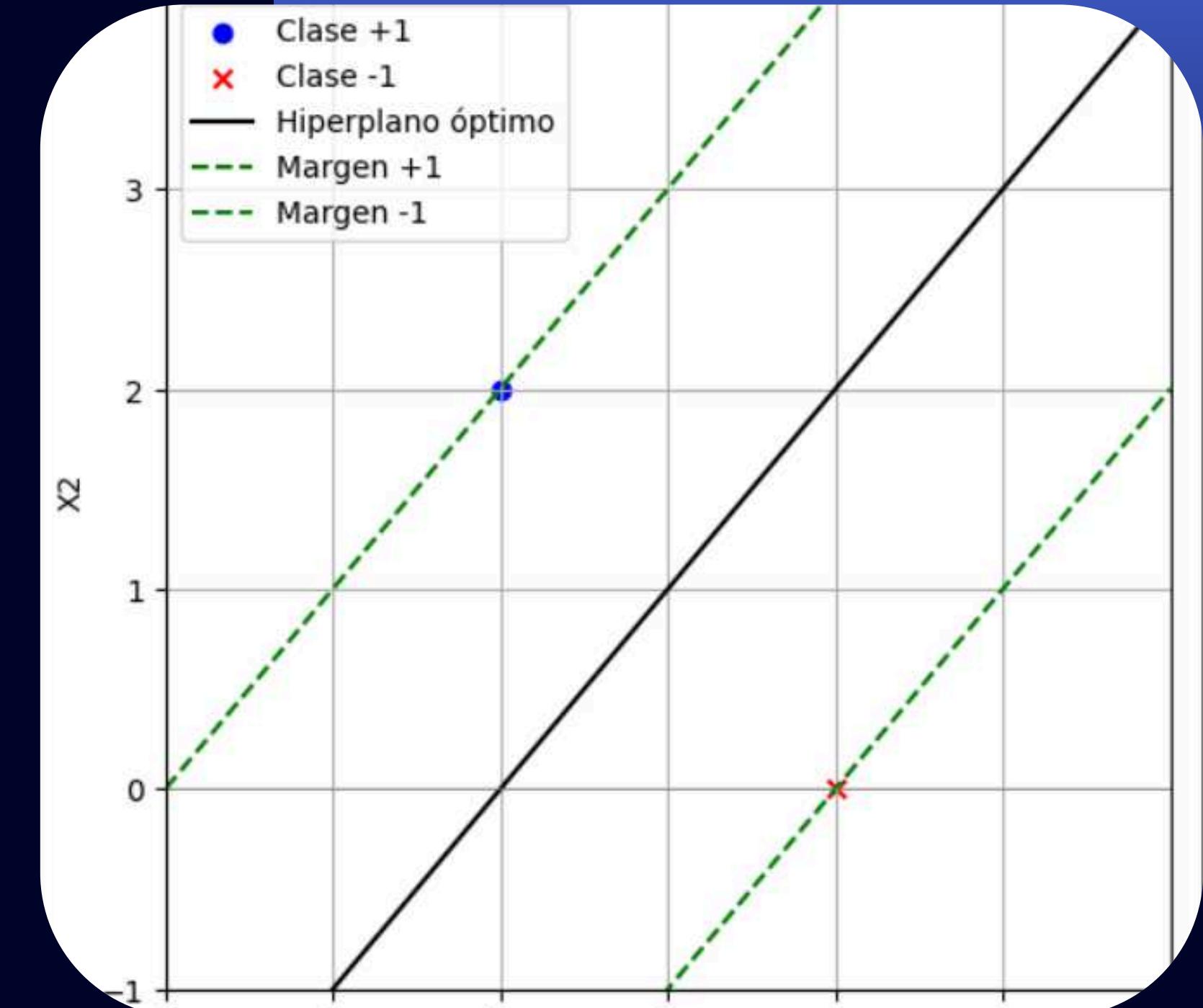
$$\text{Margen} = \frac{2}{\|w\|} = 2\sqrt{2}$$



SVM Conceptos

Resumen

Variable	Valor
w_1	$-1/2$
w_2	$1/2$
b	1
$\ w\ $	$\frac{1}{\sqrt{2}}$
Margen	$2\sqrt{2}$



Algoritmo SVM como Clasificador



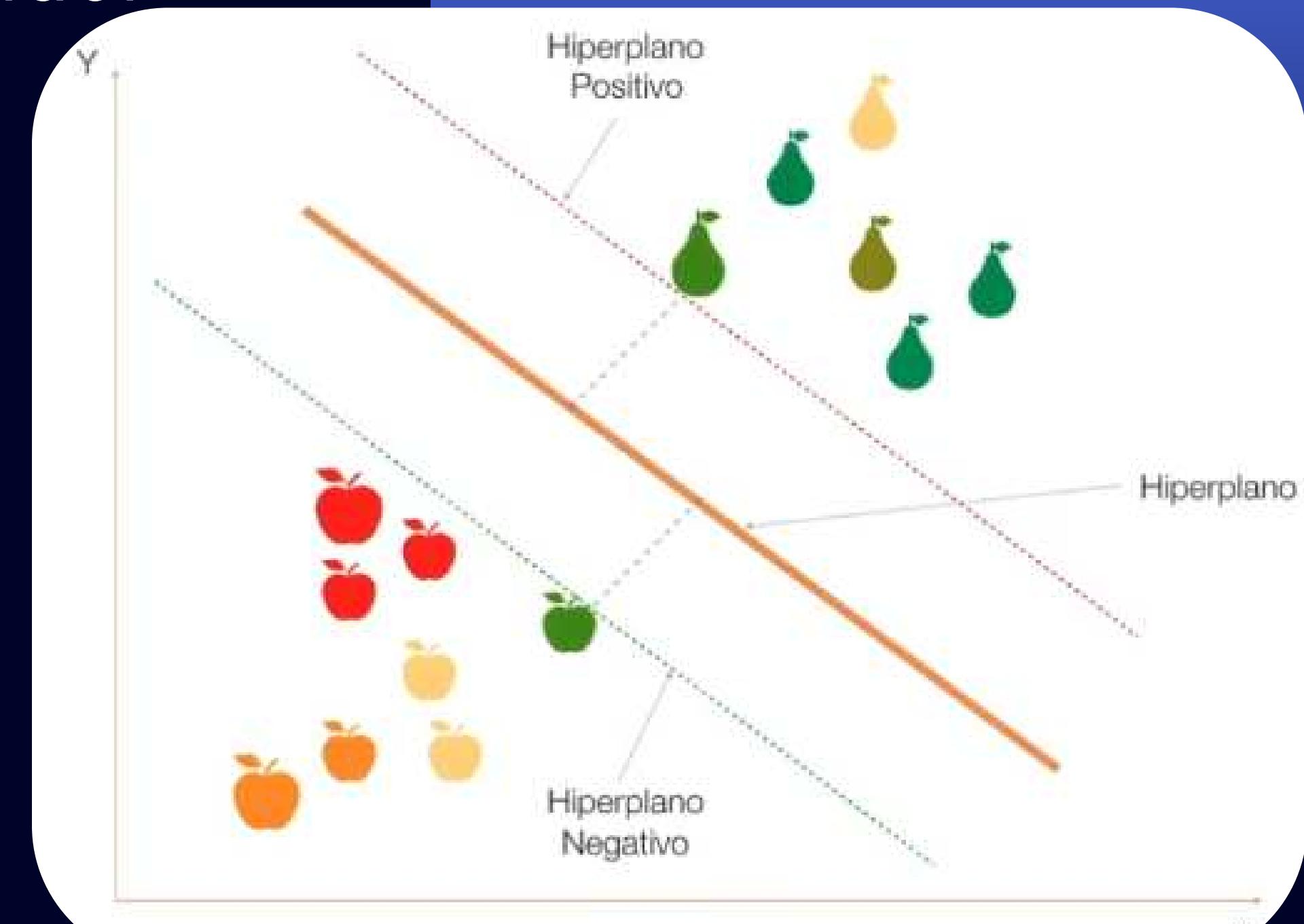
Clasificar Nuevos Puntos

Una vez entrenado el modelo, se puede clasificar un nuevo punto de datos (x_1, x_2, \dots, x_n) usando la función de decisión:

$$\text{Clase} = \text{sign}(w_1x_1 + w_2x_2 + b)$$

donde la función **sign** tiene la siguiente lógica:

- Si el argumento es negativo, la función devuelve -1.
- Si el argumento es cero, la función devuelve 0.
- Si el argumento es positivo, la función devuelve 1.



Algoritmo SVM como Clasificador

Prediccion, usando el ejemplo anterior

Dado $x_1=1$ y $x_2=3$, predecir Y,

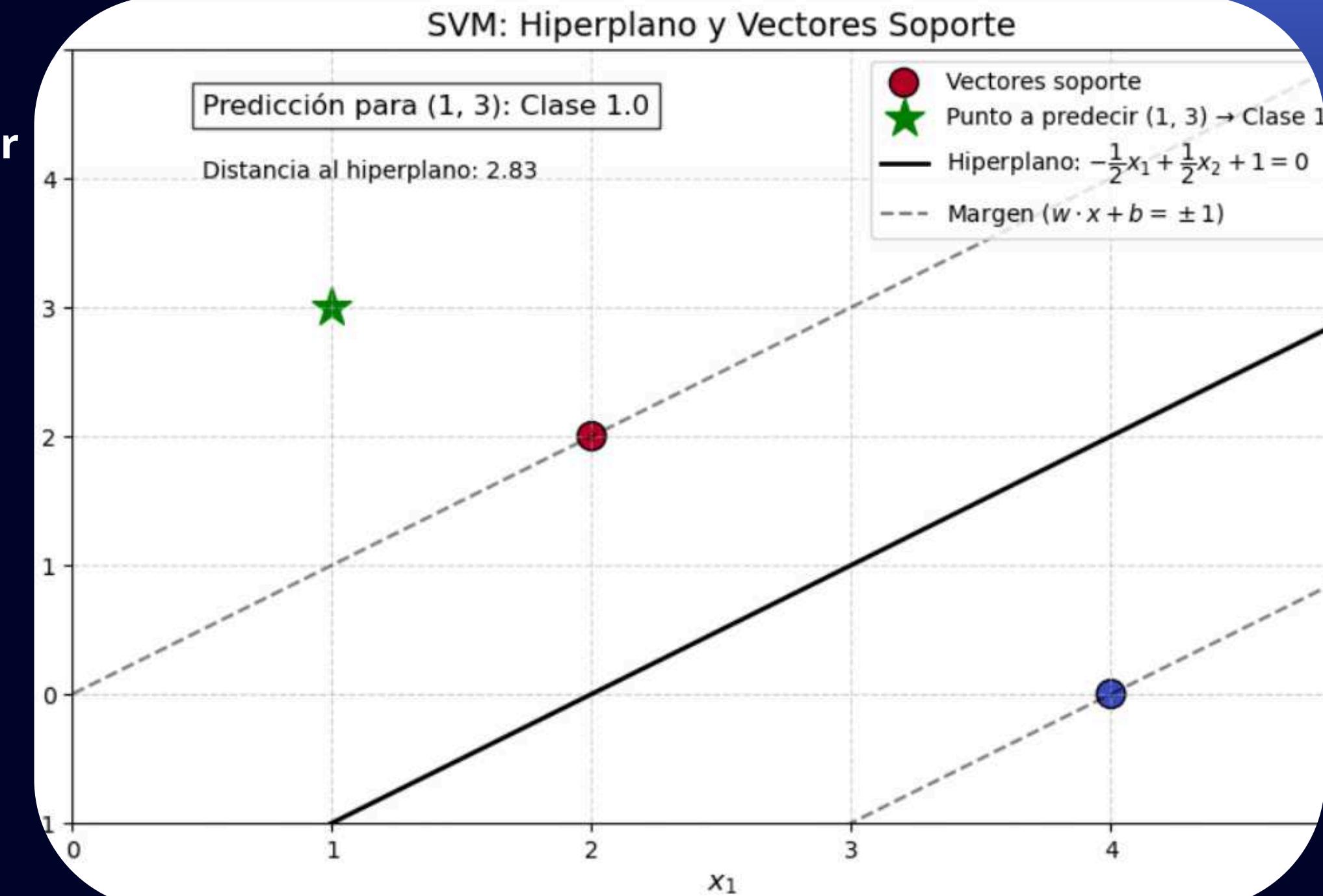
Sabemos que: $(-\frac{1}{2})x_1 + (\frac{1}{2})x_2 + 1$

Entonces: $(-0.5)(1) + (0.5)(3) + 1 = 2$

Dado:

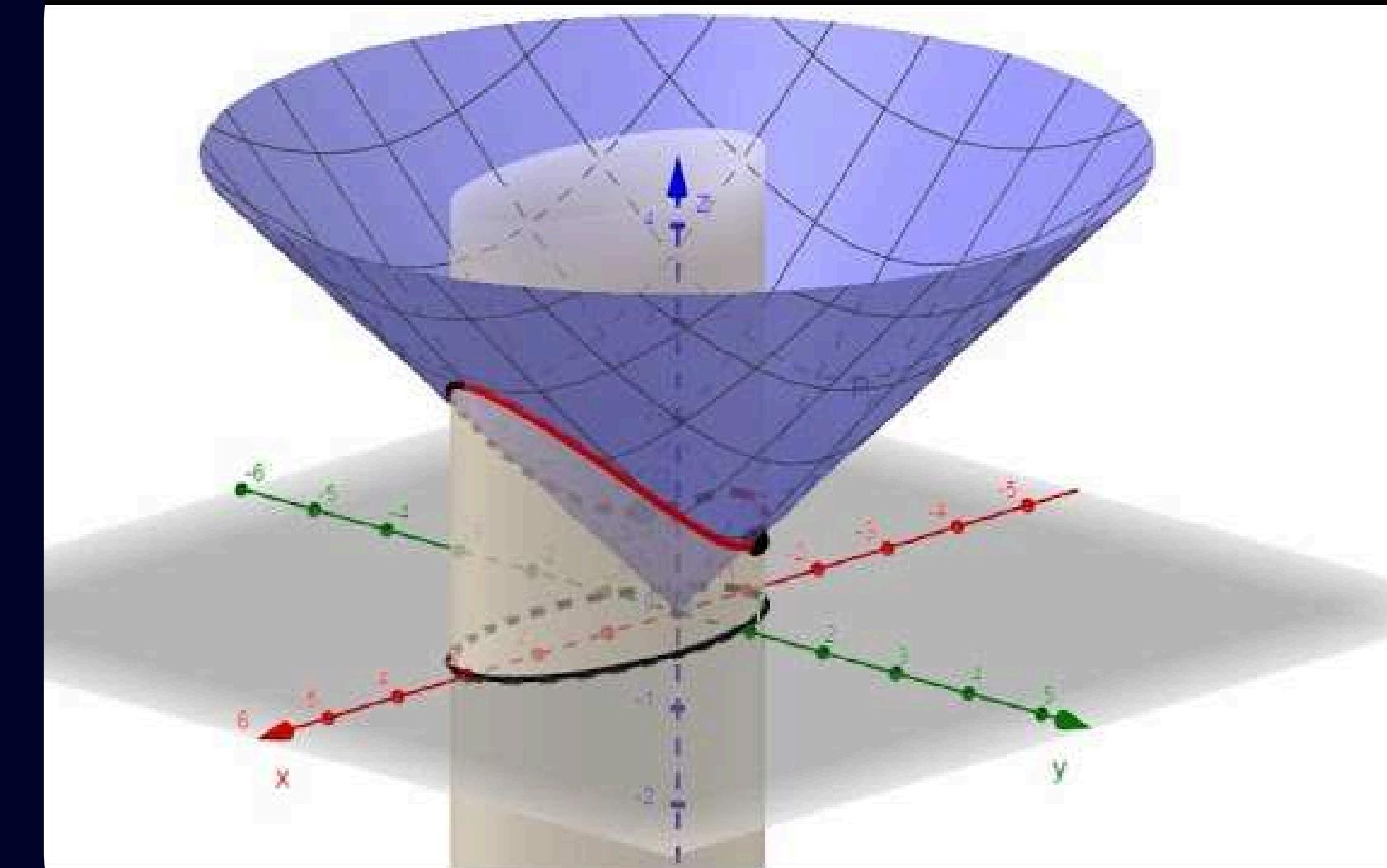
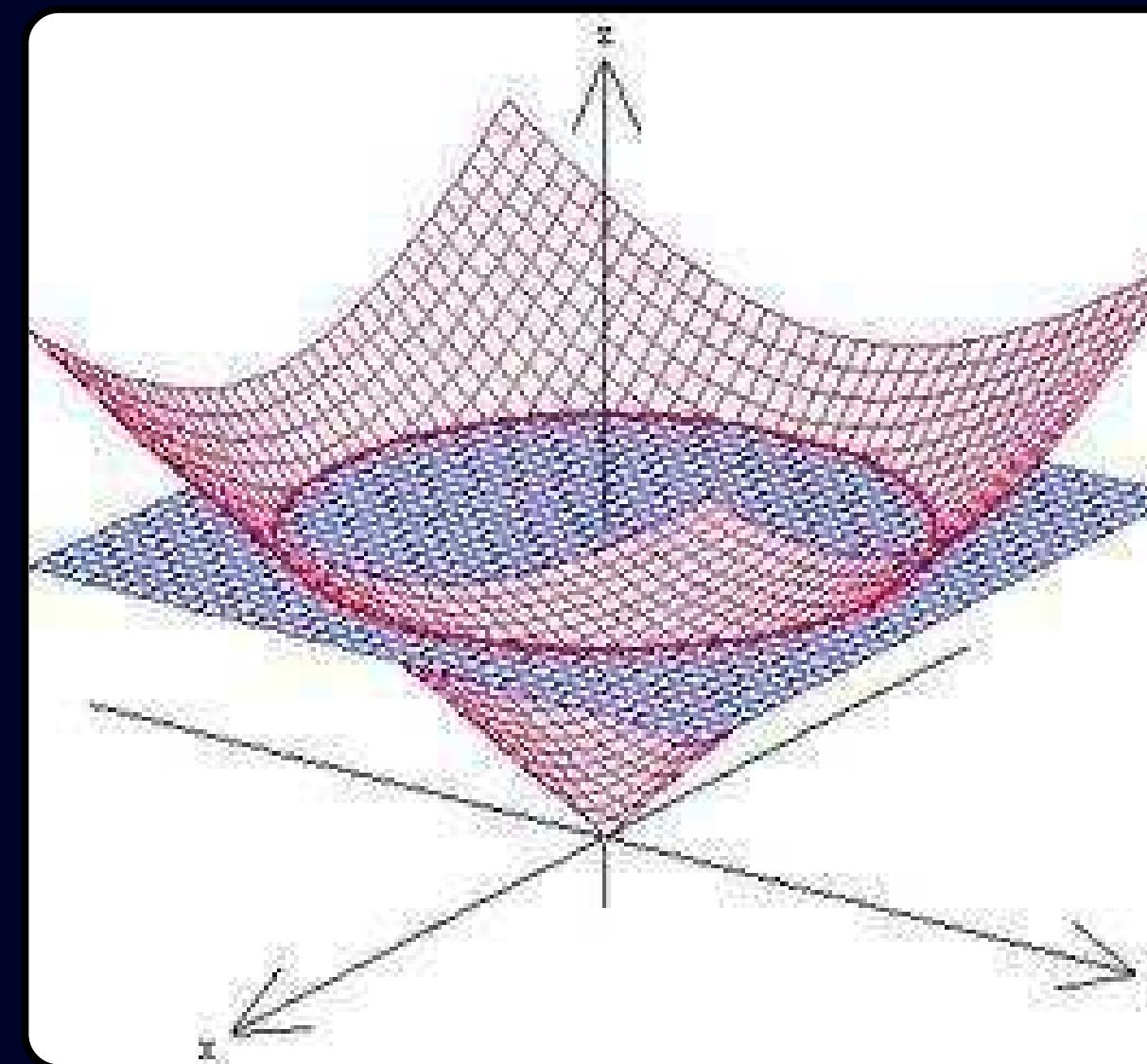
$$\hat{y} = \begin{cases} +1, & \text{si } f(\mathbf{x}) > 0, \\ -1, & \text{si } f(\mathbf{x}) < 0. \end{cases}$$

$f(x)=2>0$ entonces "y" predicha es = +1



Algoritmo SVM como Clasificador

- Esto es factible cuando tenemos un sistema de dos variables (2 dimensiones), para sistemas superiores se debe utilizar multiplicadores de Lagrange.



Demostracion cuando tenemos Xi Parametros

- Cuando tenemos X_i y Y_i

$$x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}$$

Nuevo problema de optimizacion, removiendo restricciones, introduciendo un multiplicador de Lagrange ($\alpha_i \geq 0$) para cada restriccción

$$y_i(w \cdot x_i + b) - 1 \geq 0$$

Entonces:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i [y_i(w \cdot x_i + b) - 1]$$

donde N, es el numero de vectores de soporte.

Demostracion cuando tenemos Xi Parametros



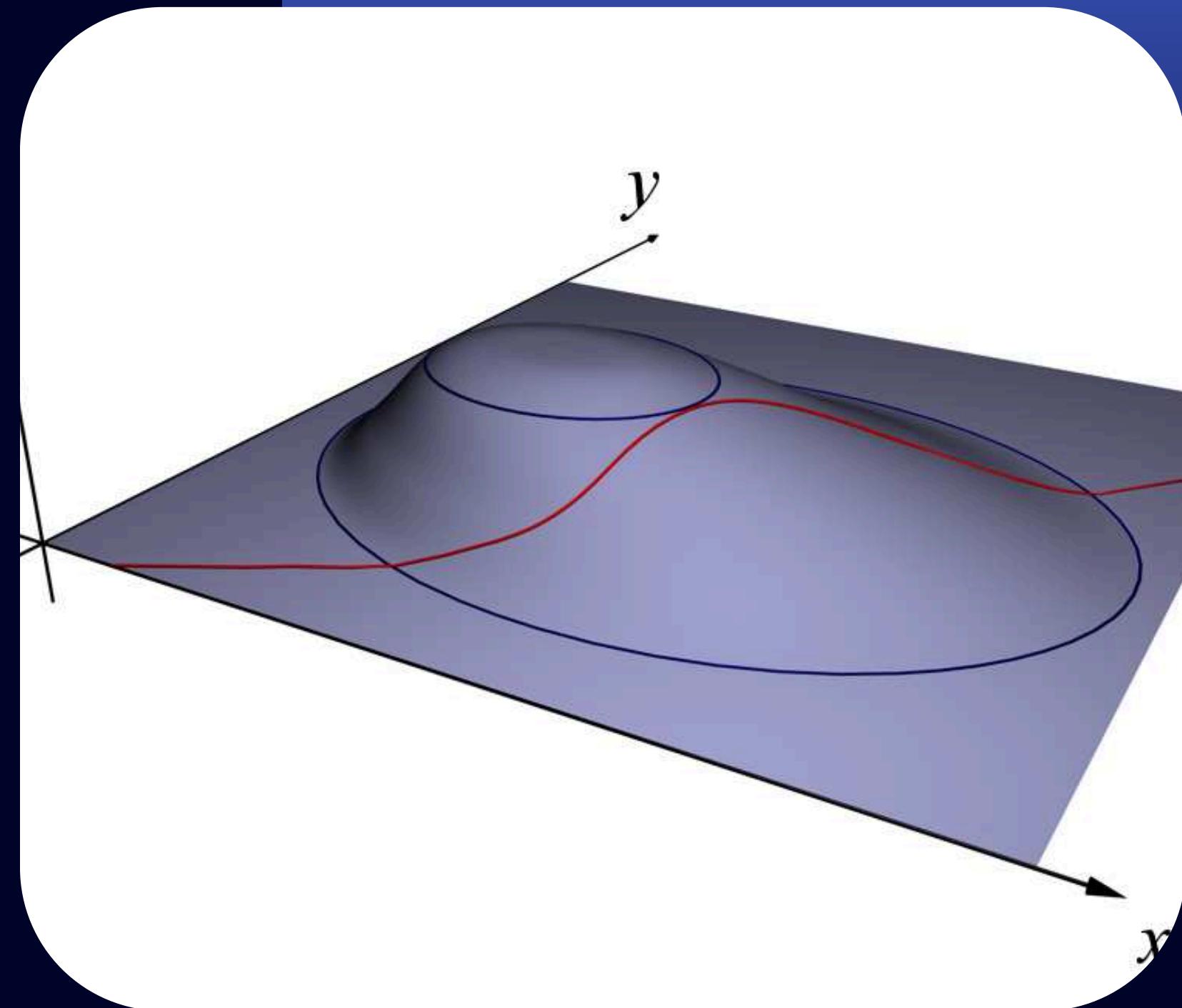
Para obtener el minimo de la funcion:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i [y_i(w \cdot x_i + b) - 1]$$

derivamos con respecto w y b igualando a 0

$$\frac{\partial \mathcal{L}}{\partial w} = 0 \implies w = \sum_{i=1}^N \alpha_i y_i x_i$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \implies \sum_{i=1}^N \alpha_i y_i = 0.$$



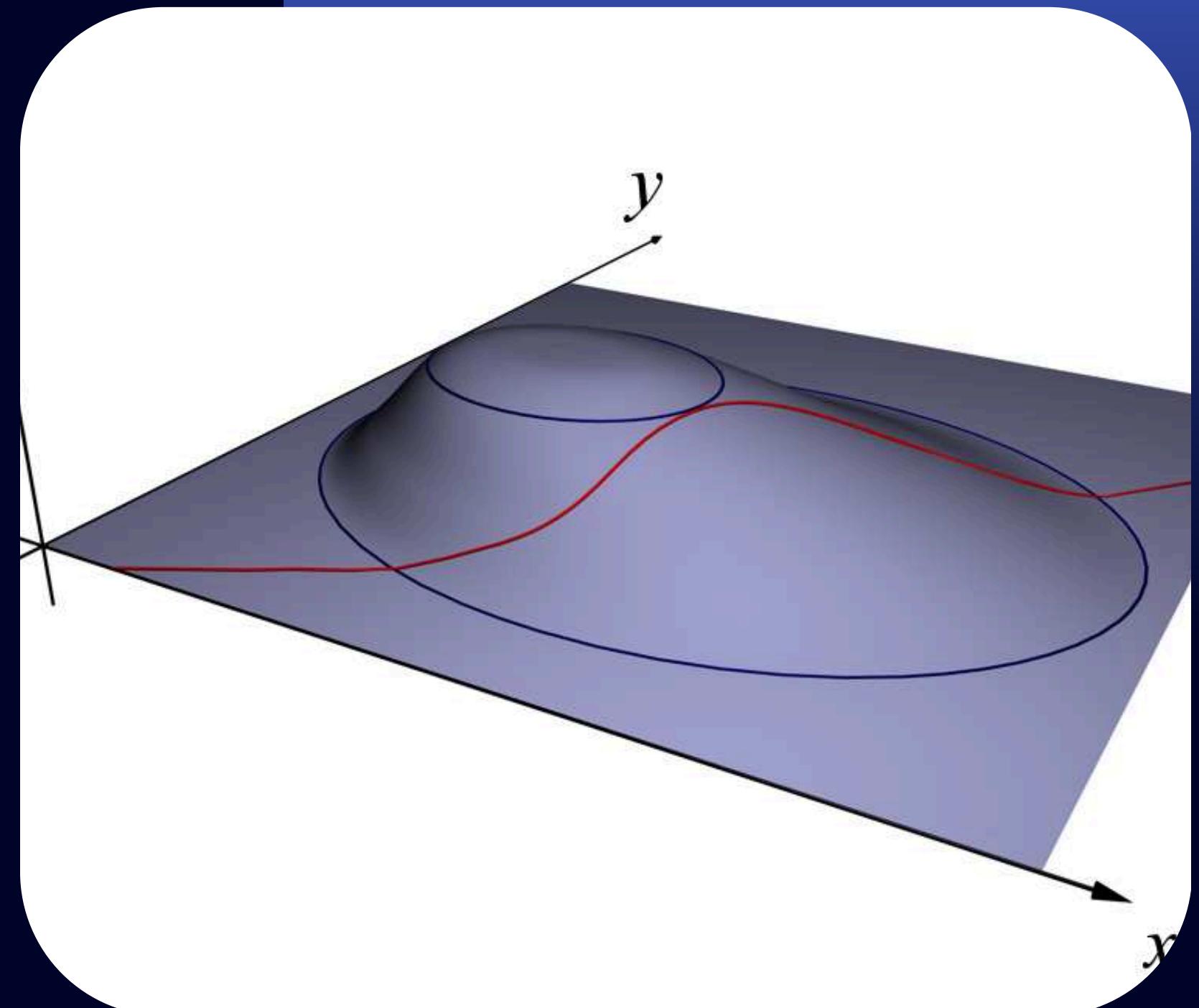
Demostracion cuando tenemos Xi Parametros

- Reemplazamos w en de la funcion de Lagrange:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i [y_i(w \cdot x_i + b) - 1]$$

entonces

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$





MÁQUINAS DE SOPORTE VECTORIAL: ¡explicación COMPLETA!



Share

¿escalador?

¿embalador?



Watch on



Ejemplo Practico

X_1	X_2	Clasificación
1	2	A
2	3	A
3	3	B
6	5	B
7	8	B

Suponiendo que hemos encontrado los siguientes valores después del entrenamiento:

$$w_1 = 1$$

$$w_2 = 1$$

$$b = -4$$

¿El hiperplano es?

$$x_1 + x_2 - 4 = 0$$

Para el punto nuevo **(4,3)**

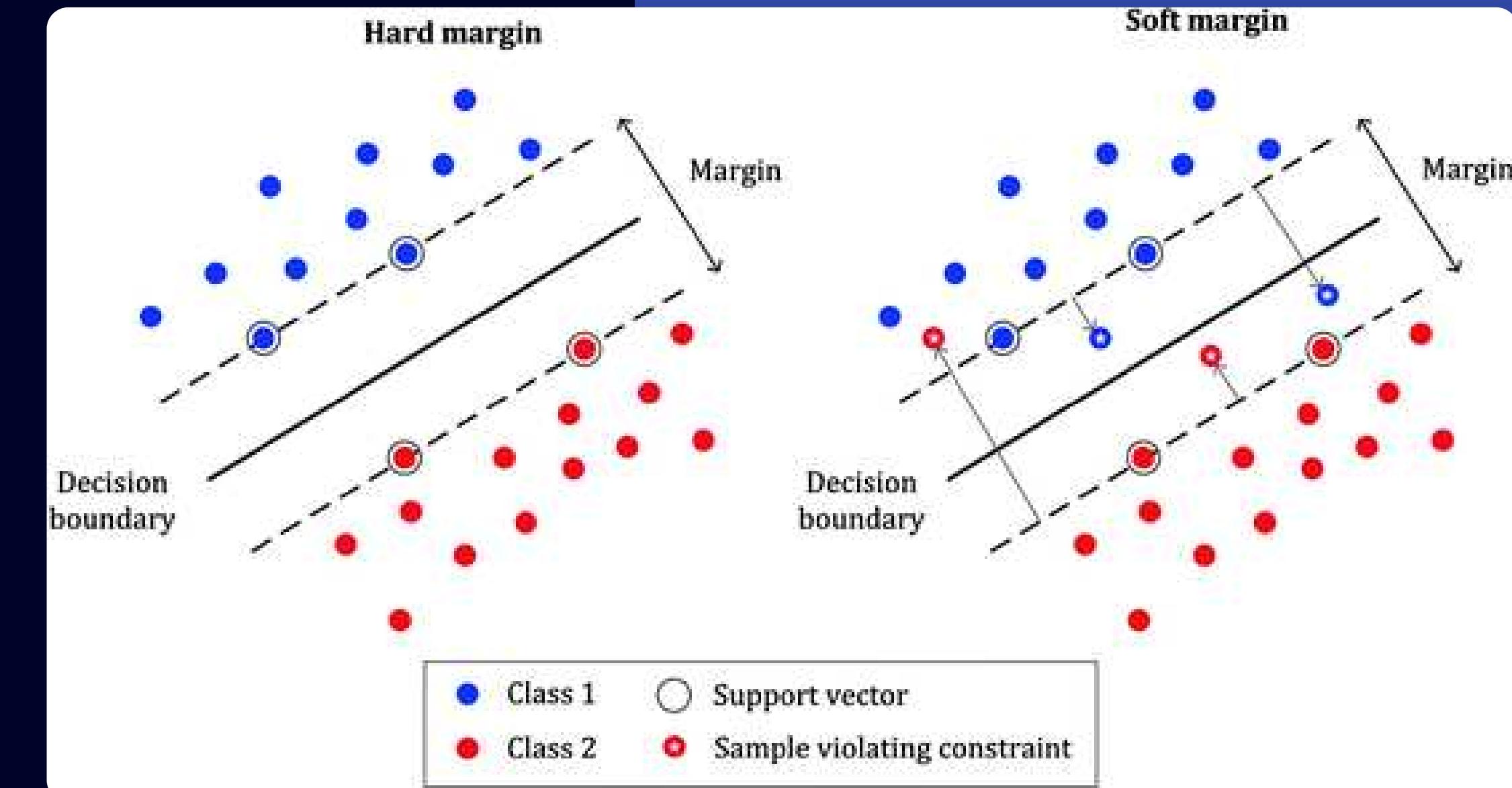
Clase = $sign(1 \times 4 + 1 \times 3 - 4) = sign(3) = 1 \rightarrow \text{Clase B}$

SVM - Margen Blando

El margen blando permite a la SVM cometer algunos errores de clasificación a cambio de lograr una mejor generalización del modelo.

En margen blando, se permiten:

- 1 Puntos dentro del margen (penalizados).
- 2 Algunos puntos mal clasificados (también penalizados).



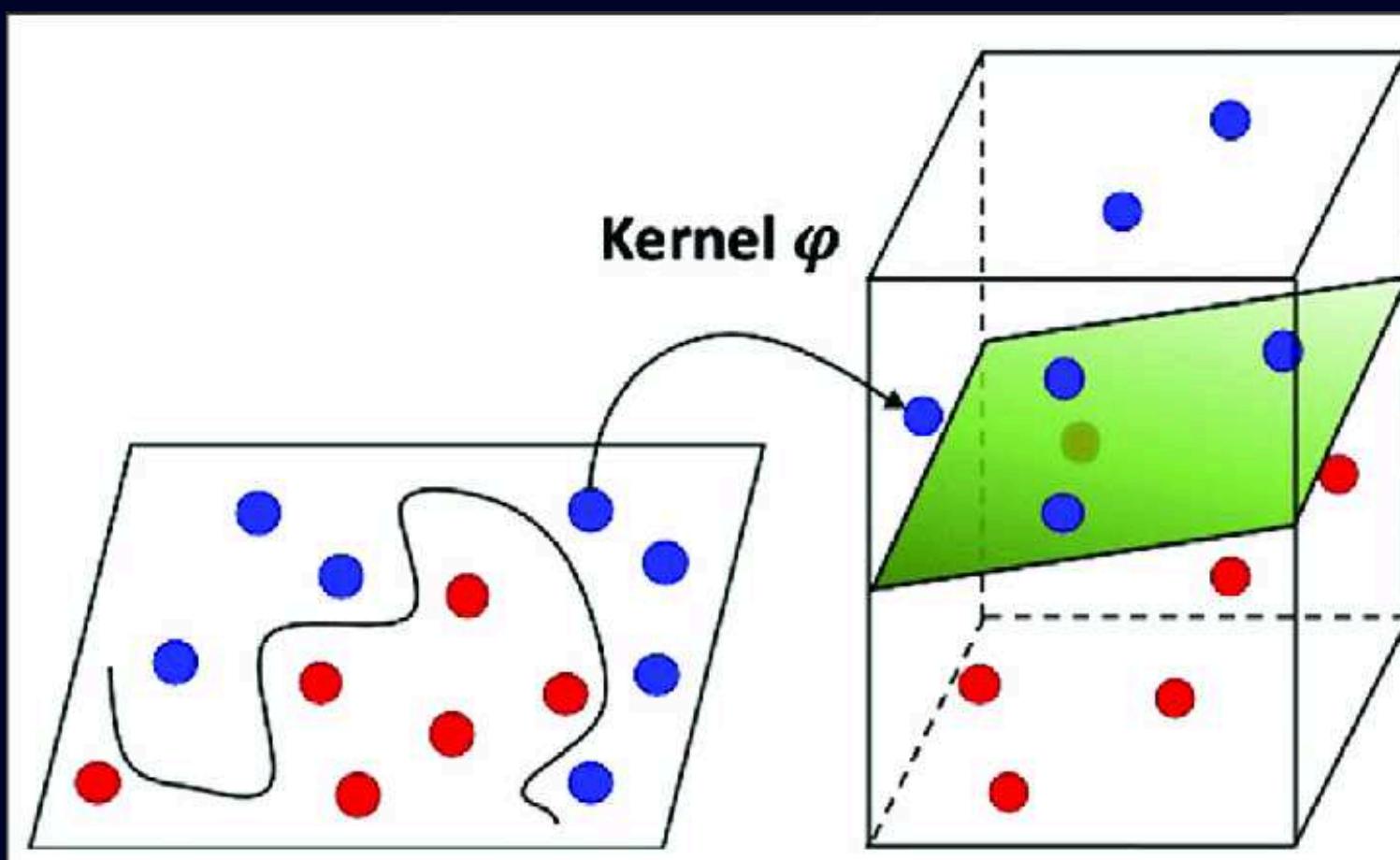
{ Esto se logra introduciendo variables de holgura ξ_i (x_i) para cada punto mal clasificado o dentro del margen.

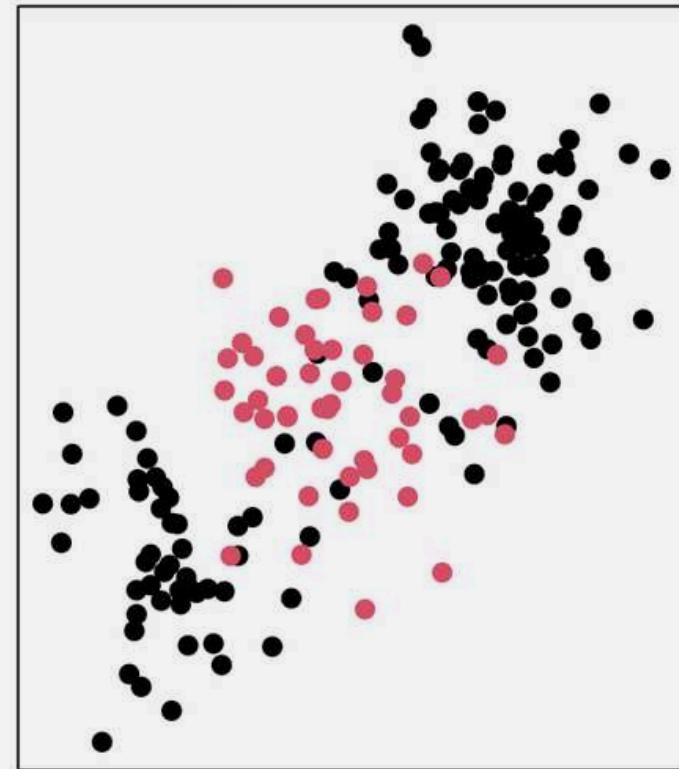
Aumento de la dimensión

La dimensión de un conjunto de datos puede transformarse combinando o modificando cualquiera de sus dimensiones. Por ejemplo, se puede transformar un espacio de dos dimensiones en uno de tres aplicando la siguiente función:

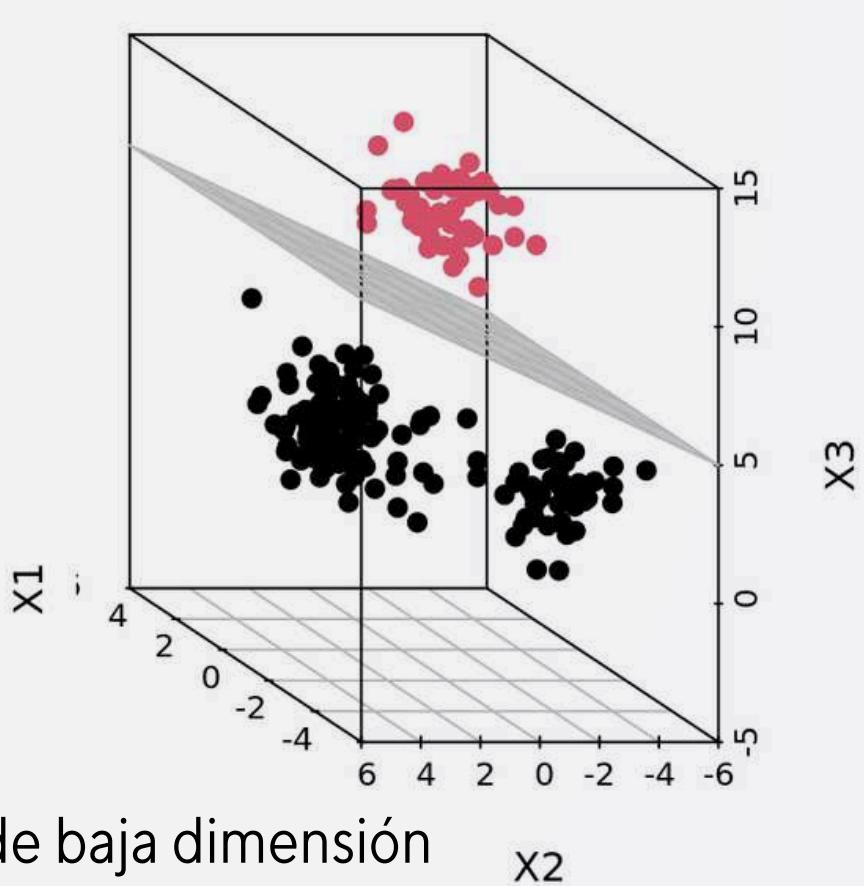
$$f(x_1, x_2) = \left(x_1^2, \sqrt{2x_1}, x_2^2 \right)$$

Esta es solo una de las infinitas trasformaciones posibles, ¿Cómo saber cuál es la adecuada? Es aquí donde los kernel entran en juego. Un kernel (K) es una función que devuelve el resultado del dot product entre dos vectores realizado en un nuevo espacio dimensional distinto al espacio original en el que se encuentran los vectores. Aunque no se ha entrado en detalle en las fórmulas matemáticas empleadas para resolver el problema de optimización, esta contiene un dot product.





Kernels convierten datos de baja dimensión
a una dimensión superior.



Kernel Tricky Transformación de Espacios

Muchos problemas de clasificación no son lineales. El kernel trick permite transformar datos no lineales en espacios lineales, donde SVM puede encontrar un hiperplano óptimo.

1 Kernel Lineal

Para datos separables

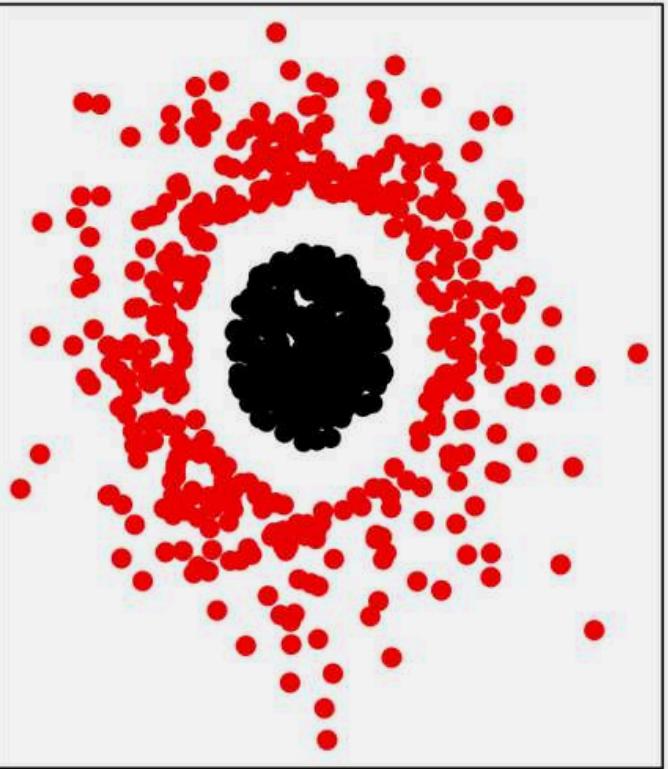
2 Kernel Polinomial

Permite curvas en la
separación

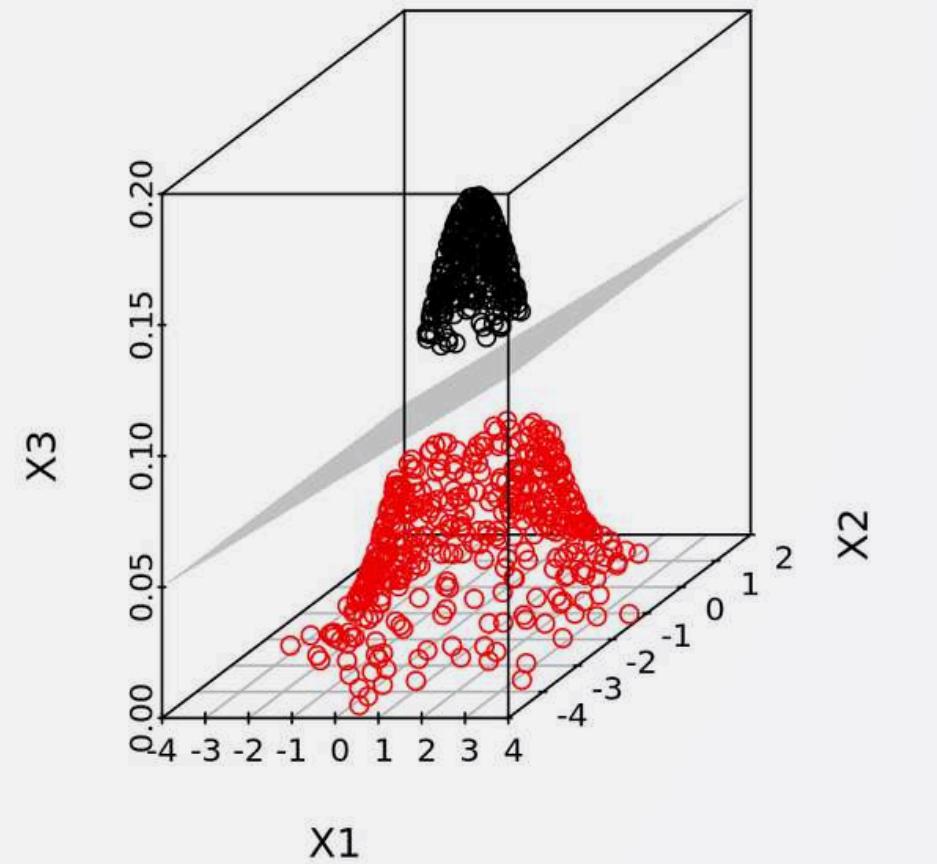
3 Kernel Radial Basis Function (RBF)

Popular para datos no lineales

Kernels convierten datos de baja dimensión
a una dimensión superior.



x1



Kernel Tricky y Transformación de Espacios

Imagina un conjunto de datos donde los puntos de una clase forman un círculo rodeado por otra clase. En este caso, ninguna línea recta podrá separar ambas clases en el espacio original.

1 Kernel Lineal

La función del kernel es simplemente:

$$K(x_i, x_j) = x_i \cdot x_j$$

2 Kernel Polinomial

Función del kernel:

$$K(x_i, x_j) = (x_i \cdot x_j + c)^d$$

El grado d define la complejidad del modelo (mayores valores permiten curvas más complejas).

3 Kernel Radial Basis Function (RBF)

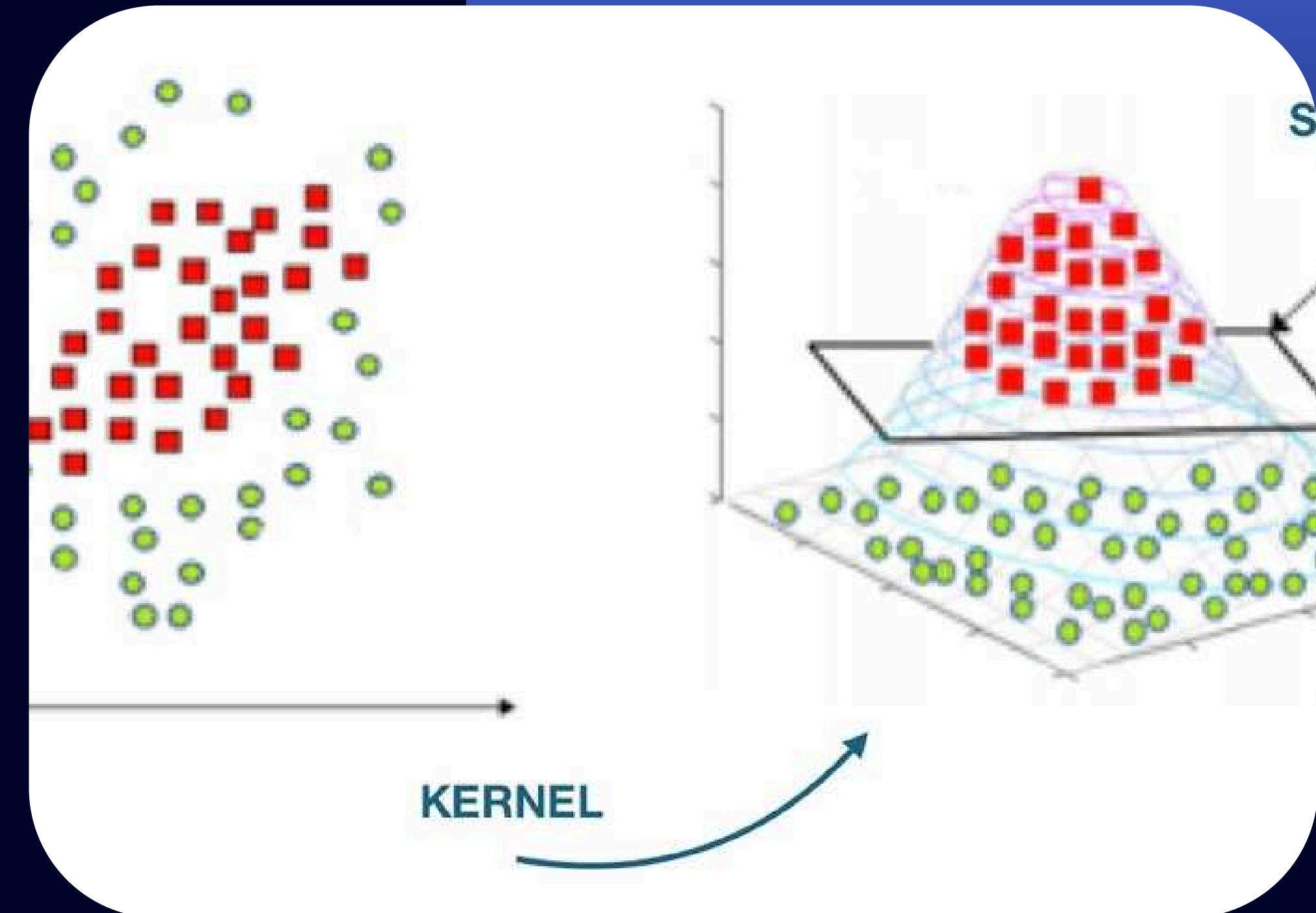
Función del kernel:

$$K(x_i, x_j) = \exp(-\gamma ||x_i, x_j||^2)$$

Kernel Radial

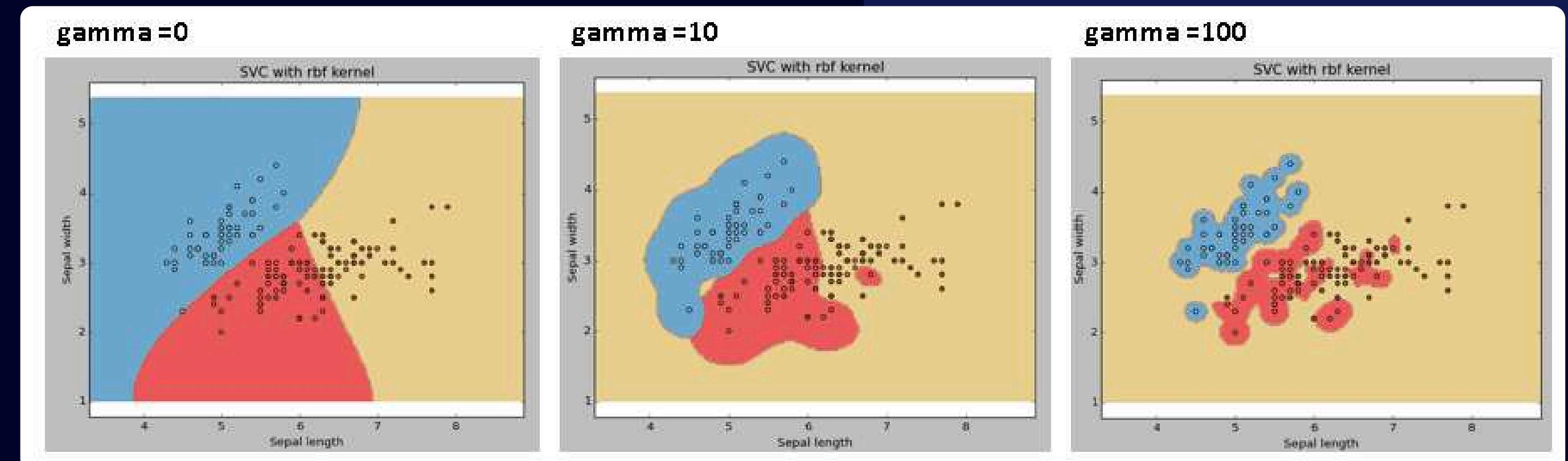
- El kernel radial, también conocido como RBF (Radial Basis Function), es el tipo de kernel más utilizado en máquinas de soporte vectorial (SVM) cuando el problema no es linealmente separable. Permite que el SVM encuentre fronteras de decisión no lineales al transformar los datos a un espacio de mayor dimensión.

$$K(x_i, x_j) = e^{-\gamma ||x_i, x_j||^2}$$



Kernel Radial

- γ (*gamma*) define la "anchura" de la influencia de cada punto de entrenamiento.
 - γ grande: influencia muy local puede sobreajustar (overfitting).
 - γ pequeño: influencia amplia puede subajustar (underfitting).



Aplicación del Kernel a la Ecuación de Optimización

El lagrangiano original para SVM (con kernel lineal) es:

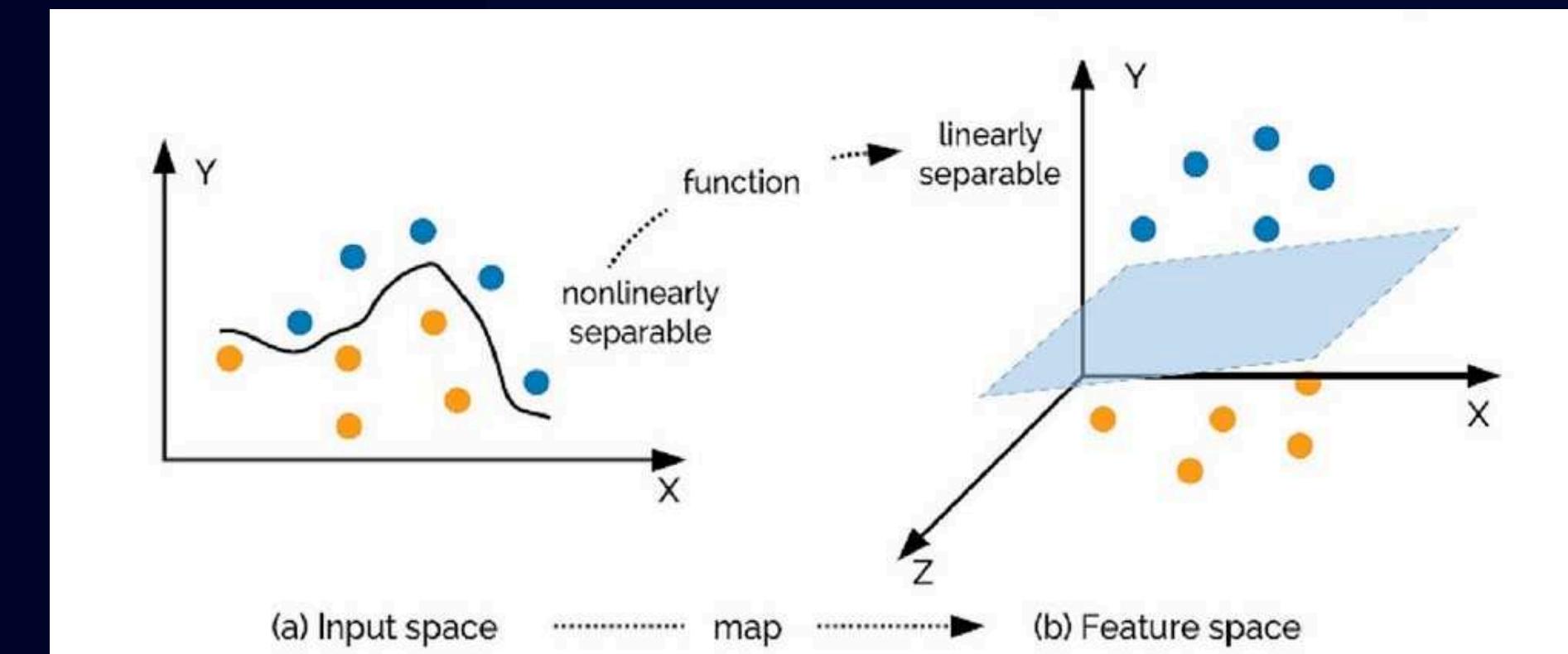
$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1]$$

Al aplicar las condiciones de optimalidad, se reemplaza $\mathbf{w} \cdot \mathbf{x}_i$ con el kernel $K(\mathbf{x}_i, \mathbf{x}_j)$ es:

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Sujeto a:

$$\sum_{i=1}^n \alpha_i y_i = 0 \quad 0 \leq \alpha_i \leq C \quad \forall i$$





PERCEPCIÓN — **COMPUTACIONAL** — COMPUTER VISION

Fundamentos

CONTENIDOS

FUNDAMENTOS

VISION POR COMPUTADORA

FLUJO DE TRABAJO

FILTROS

**REPRESENTACIÓN DIGITAL
DE UNA IMAGEN**

¿Qué es Percepción Computacional?

- Capacidad de un sistema para adquirir, procesar y comprender datos del entorno.
- En IA, simula cómo los humanos perciben: visión, audio, tacto.
- Se apoya en sensores, algoritmos y modelos de ML/DL.



¿Qué es Visión por Computadora?

Rama de la percepción computacional enfocada en extraer información de imágenes y videos.

Objetivo: convertir píxeles en conocimiento útil.



Tareas

Estas tareas permiten que los sistemas de visión artificial "vean" e interpreten el mundo visual, imitando la visión humana.

Visión por computadora

Clasificar



Detectar



Segmentar

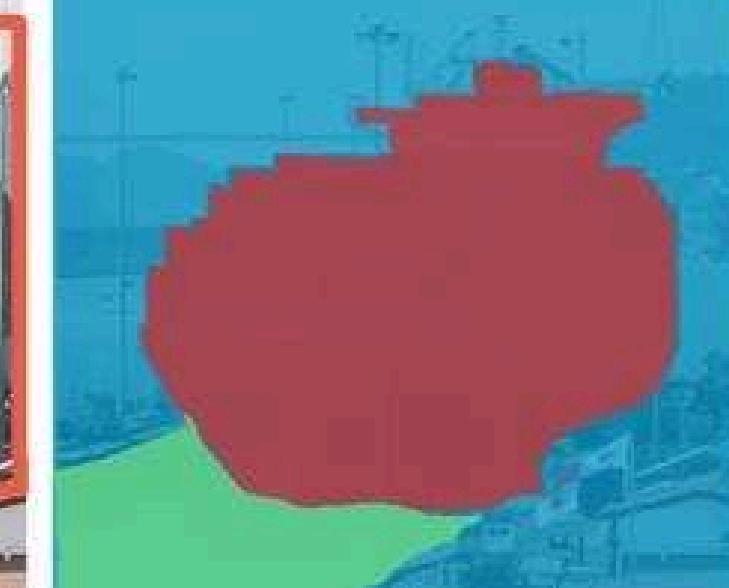


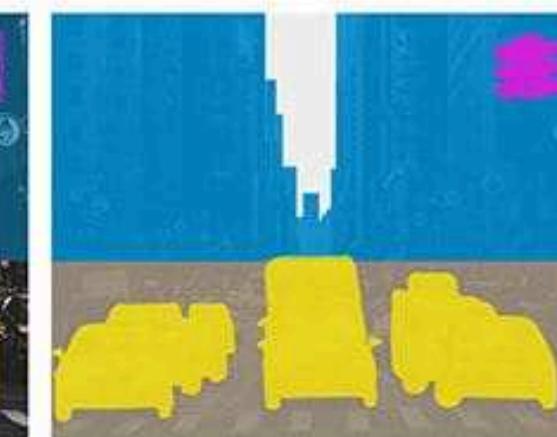
Image Classification



Object Detection



Image Segmentation



Object Tracking



Flujo Típico



ADQUISICIÓN

Es el proceso mediante el cual se captura una imagen o un conjunto de imágenes desde una fuente visual, como una cámara, un escáner, un video o sensores.



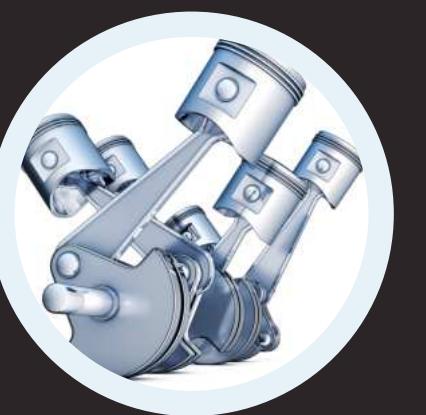
PREPROCESAMIENTO

Son las transformaciones iniciales que se aplican a la imagen para mejorar su calidad o prepararla para una mejor interpretación por parte del algoritmo.



EXTRACCIÓN DE CARACTERÍSTICAS

Es la etapa donde se identifican y cuantifican atributos importantes de la imagen que ayudarán al modelo a reconocer patrones.



PROCESAMIENTO

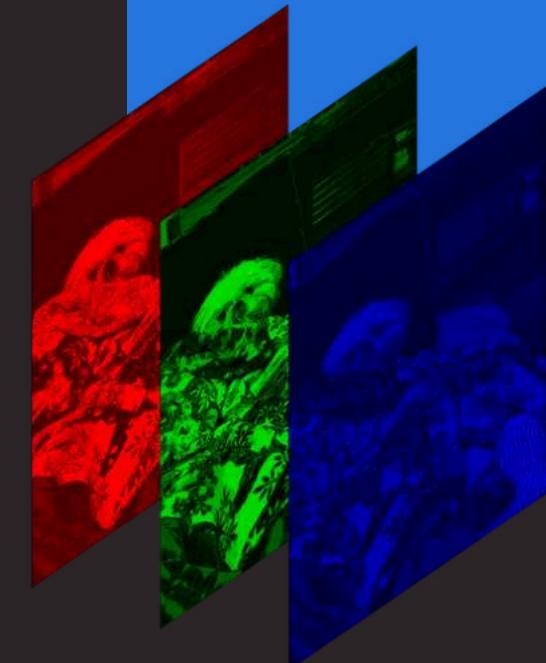
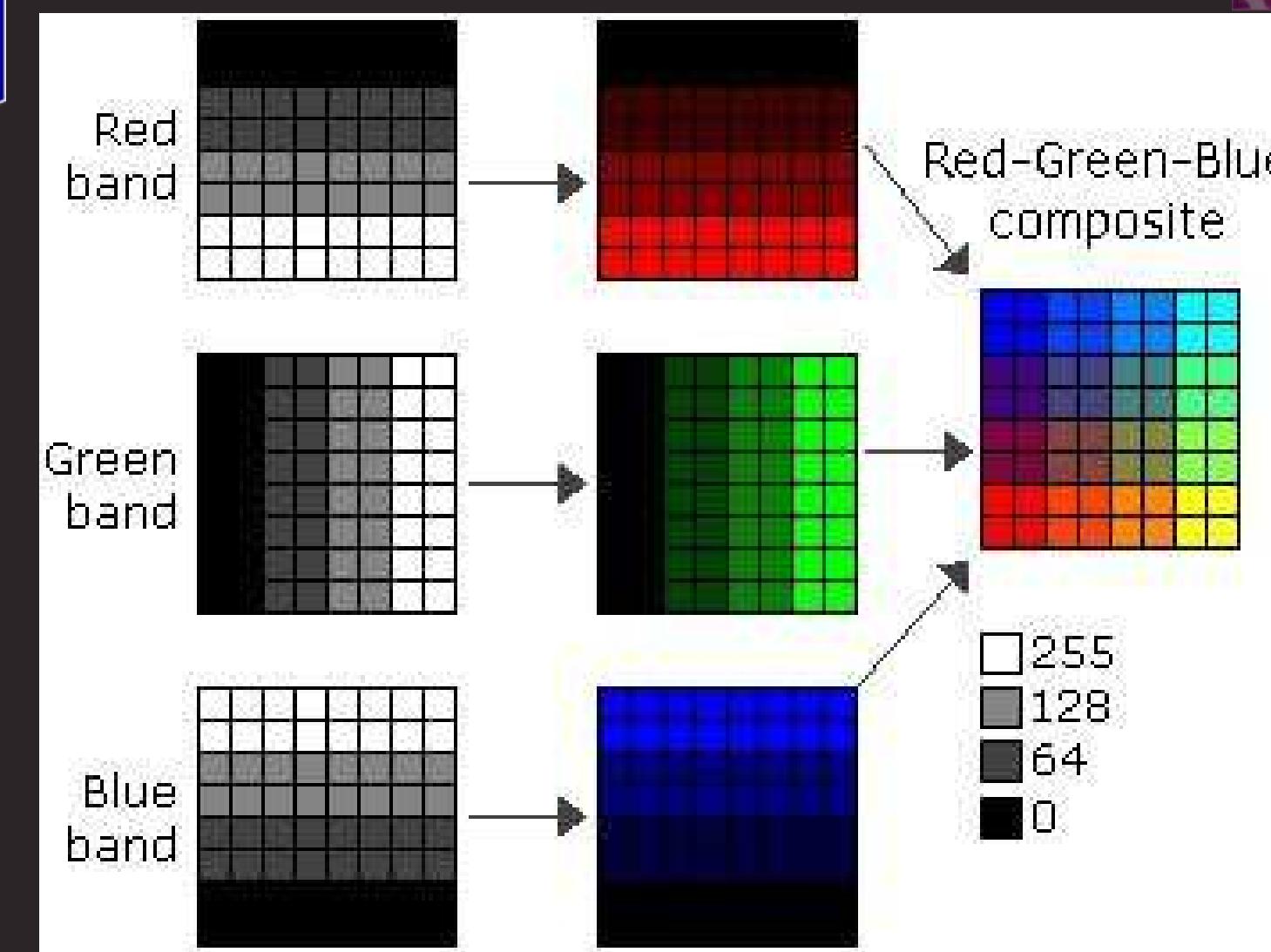
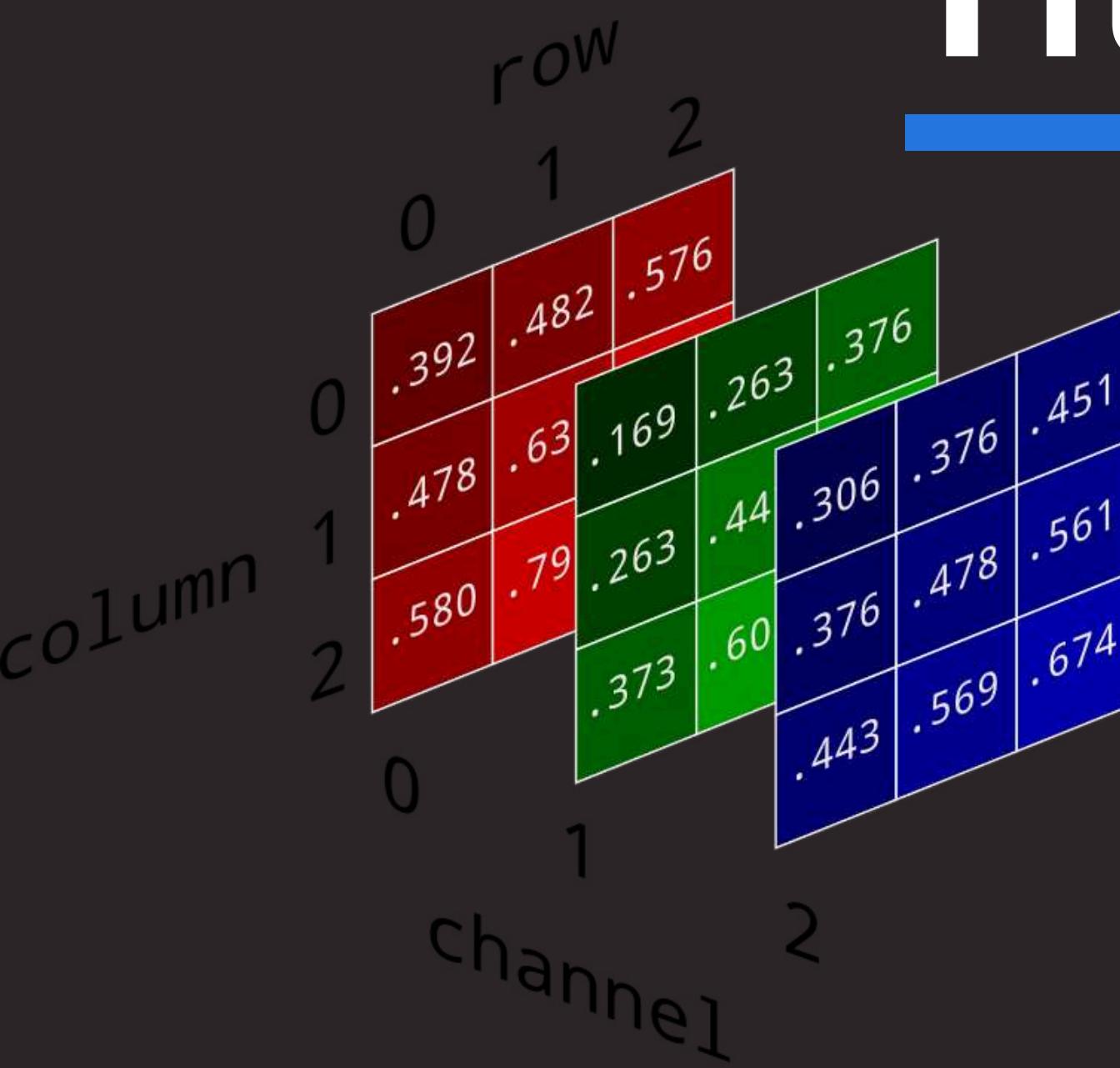
Se aplica un algoritmo de aprendizaje automático o profundo que analiza las características extraídas para clasificar, detectar, segmentar o seguir objetos.



TOMA DE DECISIÓN

Es la fase final, donde se interpretan los resultados del modelo para realizar una acción o emitir una salida comprensible.

Preprocesamiento

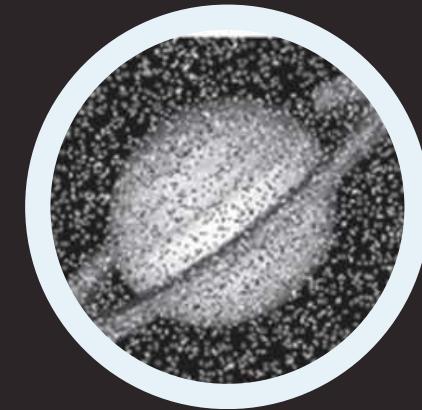


Preprocesamiento



ESCALA DE GRISES

Reducir la complejidad del procesamiento al eliminar la información de color.



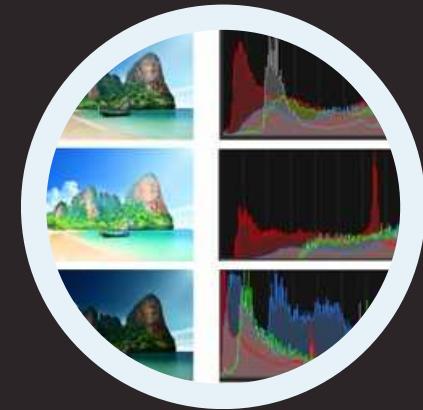
REDUCCIÓN DE RUIDO

Mejorar la detección de bordes o contornos al eliminar variaciones pequeñas de píxeles.



UMBRALIZACIÓN (THRESHOLDING)

Para simplificar una imagen y separar objetos de interés del fondo, convirtiéndola en una imagen binaria (solo blanco y negro).



AJUSTE DE BRILLO Y CONTRASTE

Resaltar contornos y formas dentro de la imagen.



REDIMENSIONAMIENTO (RESIZING)

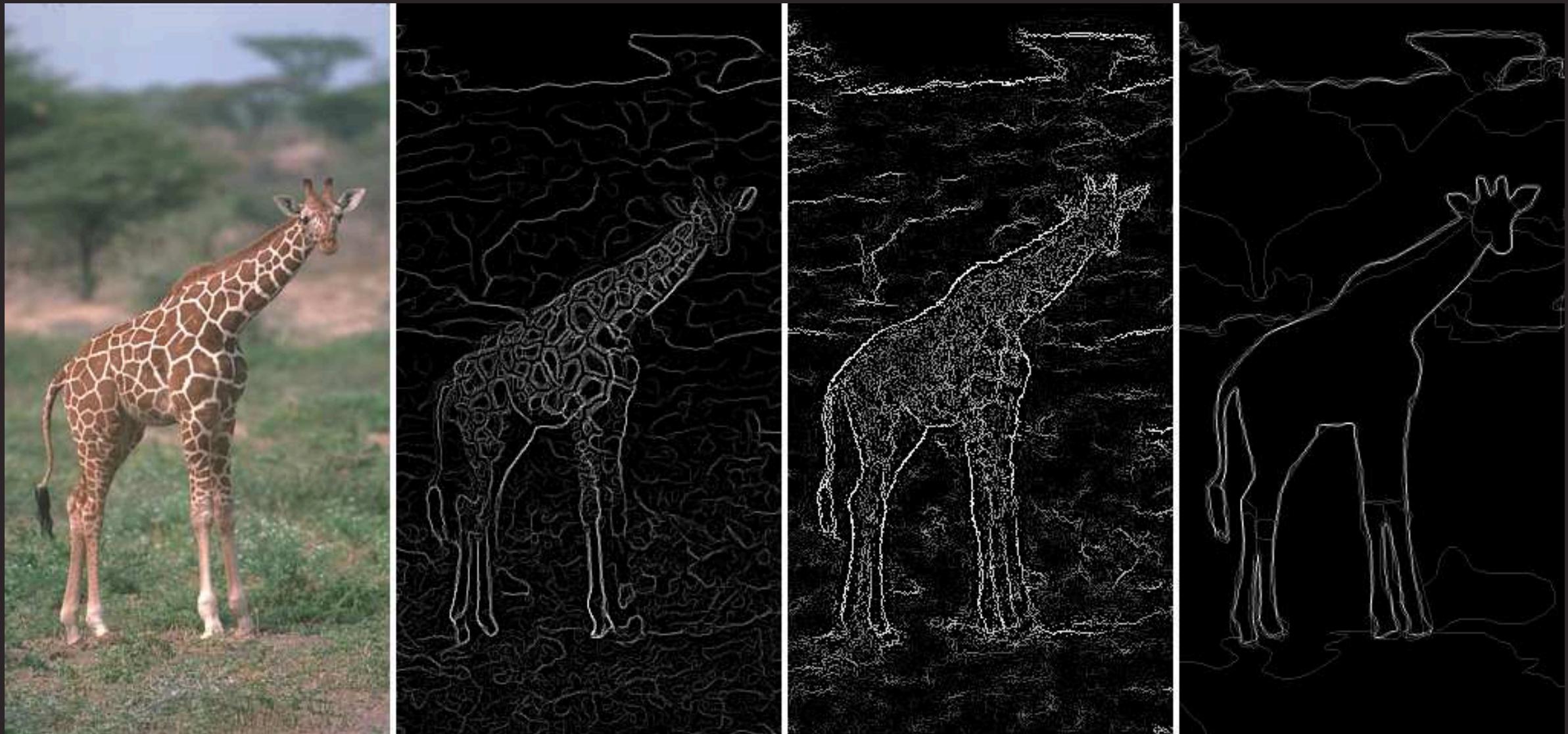
- Imágenes más pequeñas → menos píxeles → menos cálculos → entrenamiento más rápido.
- Los modelos requieren que todas las imágenes tengan la misma dimensión (ej. 224x224).

Extracción de Características



DETECCIÓN DE BORDES

Resaltar contornos y formas dentro de la imagen.

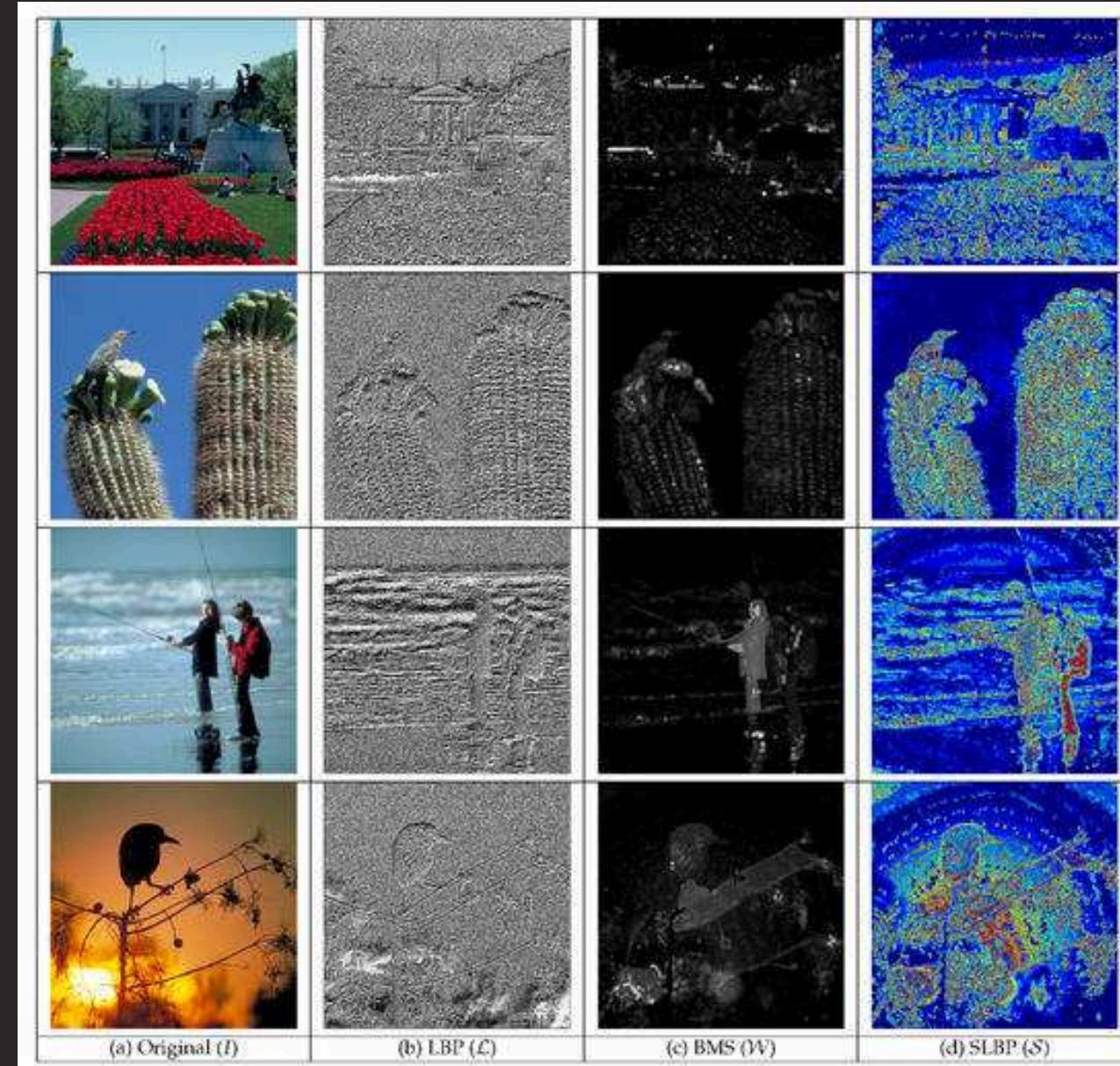


Extracción de Características

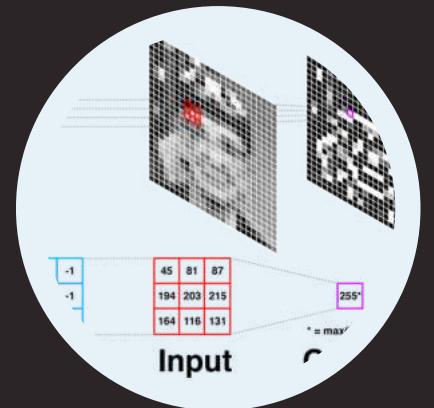


EXTRACCIÓN DE TEXTURA

- Describir la textura de una imagen comparando píxeles vecinos.
- Identificar patrones repetitivos como rugosidad o suavidad.



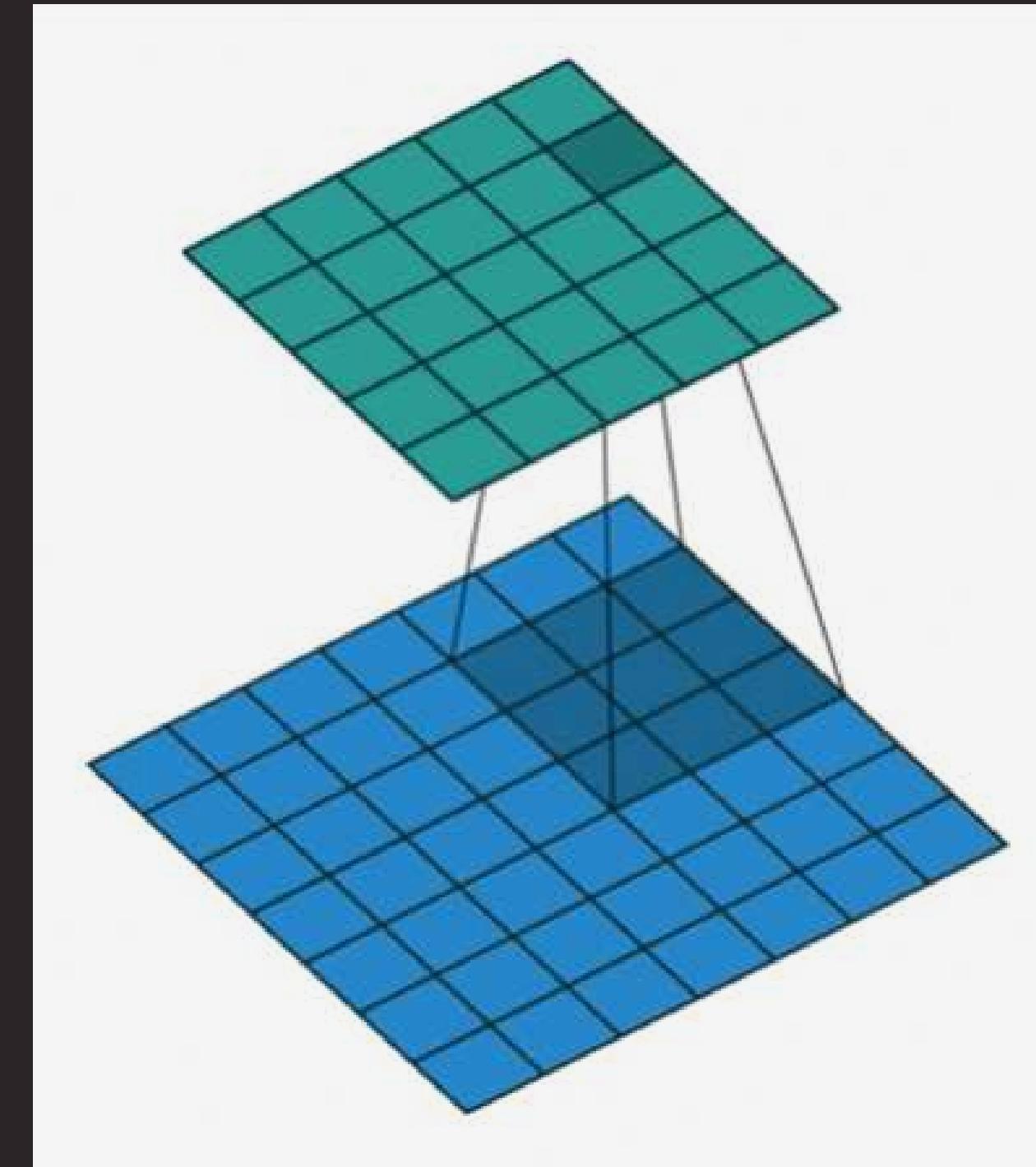
¿Cómo se logran tratar las imágenes?



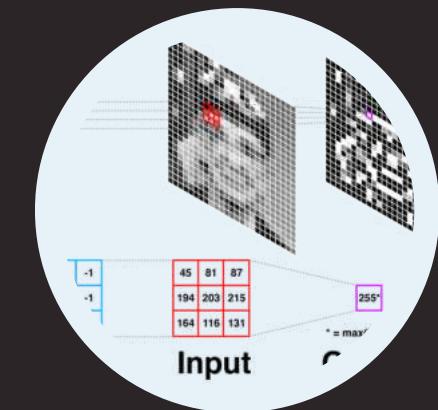
CONVOLUCIÓN

Es aplicar un filtro o kernel sobre una imagen para transformar sus valores de píxeles.

La operación implica recorrer la imagen con una pequeña matriz (el kernel), multiplicando y sumando los valores en cada posición.



¿Cómo?



CONVOLUCIÓN

- Es aplicar un filtro o kernel sobre una imagen para transformar sus valores de píxeles.
- La operación implica recorrer la imagen con una pequeña matriz (el kernel), multiplicando y sumando los valores en cada posición.

20	23	30	31
22	21	29	30
23	24	32	33
29	31	34	37

1	1	1
1	1	1
1	1	1

Filtro promedio

1	1	1
1	2	1
1	1	1

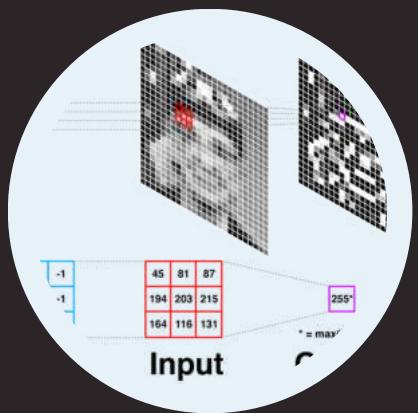
Filtro Gaussiano

$$\frac{20 + 23 + 30 + 22 + 21 + 29 + 23 + 24 + 32}{9} = 24,8$$

$$\frac{20 + 23 + 30 + 22 + 21 + 29 + 23 + 24 + 32}{10} = 24,5$$

CONVOLUCIÓN

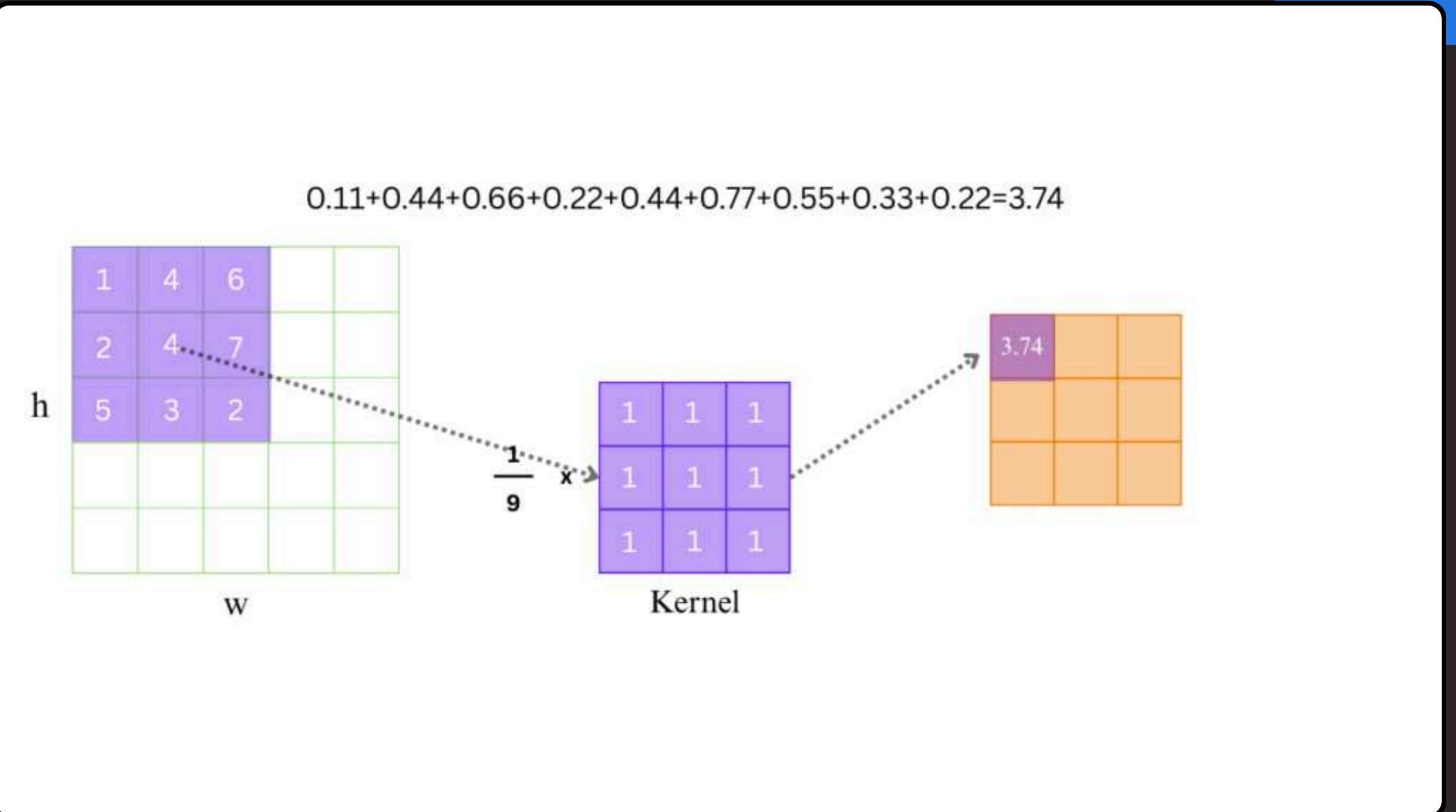
PREPROCESAMIENTO



SUAVIZAR IMÁGENES

También llamado blur, se aplica con el filtro de la Media y sirve para:

- Eliminar ruido y variaciones bruscas entre píxeles.
- Suavizar bordes y transiciones fuertes.

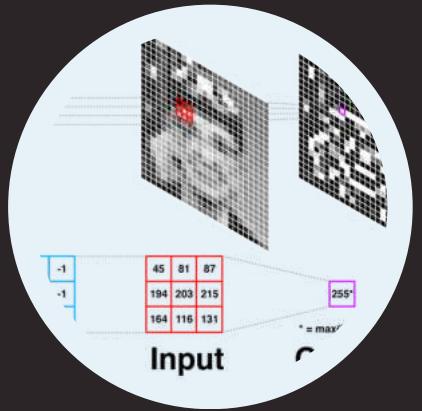


Este kernel se mueve sobre la imagen, y en cada posición:

1. Se multiplican los valores del kernel por los píxeles correspondientes de la imagen.
2. Se suman esos productos.
3. Ese valor se asigna al píxel central de la región.

CONVOLUCIÓN

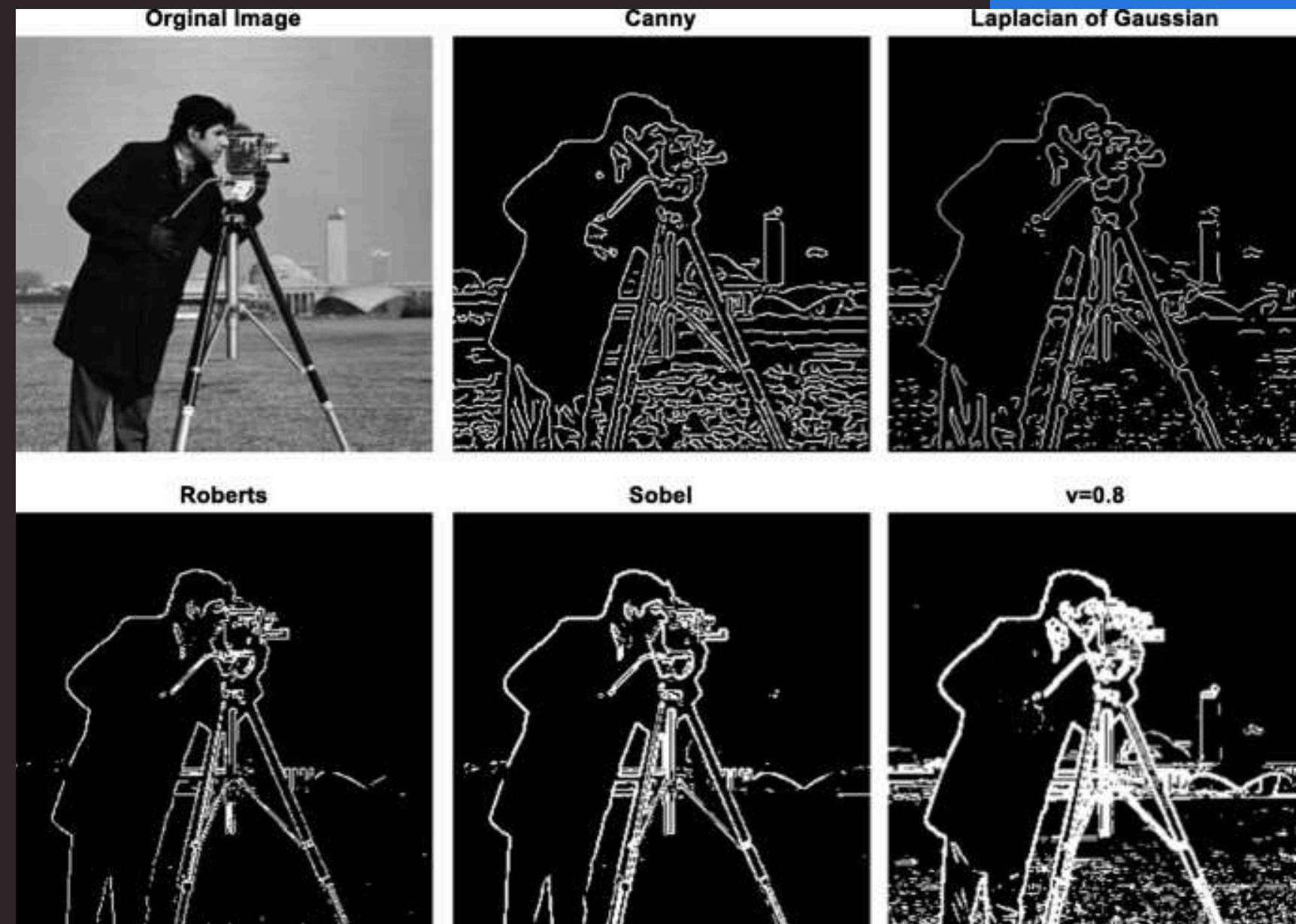
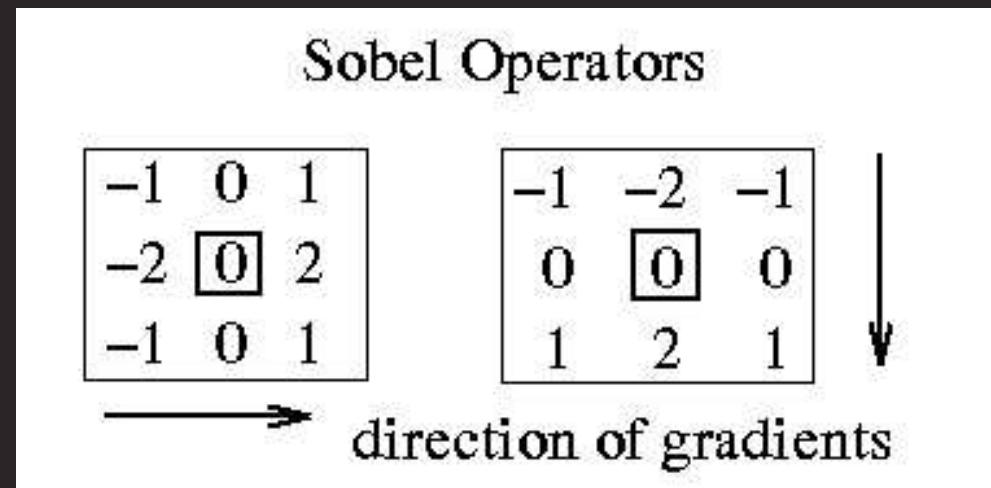
EXTRACCIÓN DE CARACTÉRISTICAS



DETECTOR BORDES

La convolución se usa para:

- Detectar bordes, esquinas, texturas, patrones (ej. filtros Sobel, Prewitt, HOG).
- Extraer descripciones espaciales en redes neuronales convolucionales (CNN).

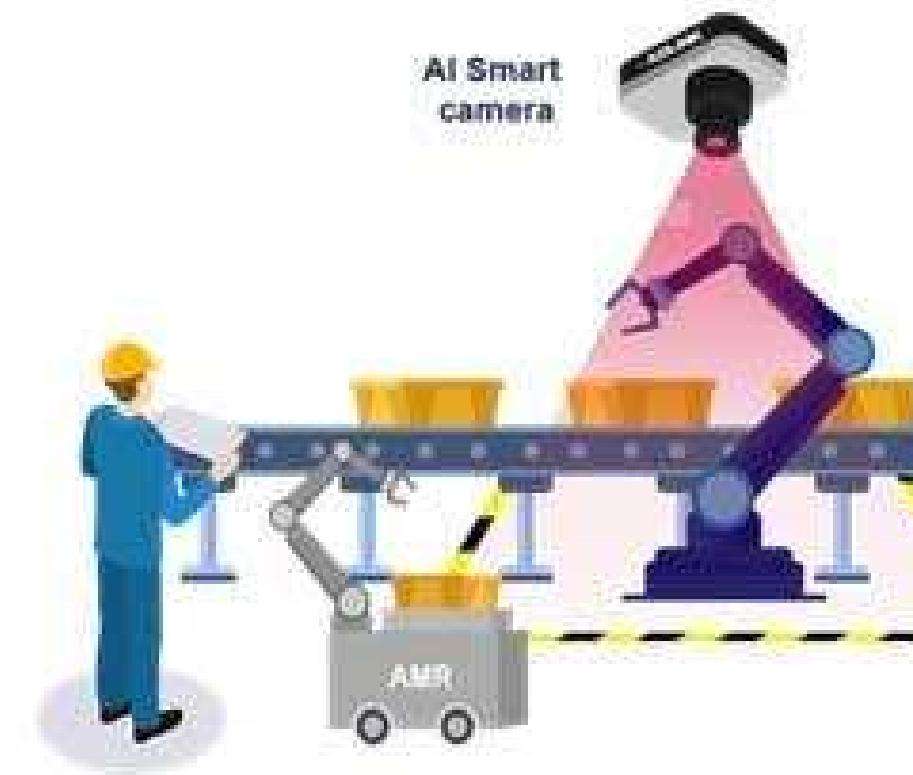


$M^+ = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$	$M^- = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}$	(a) Roberts operator
$HM = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix}$	$VM = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$	(b) Prewitt operator
$HM = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$	$VM = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$	(c) Sobel operator
		operator
$M = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$		(d) Laplacian operator

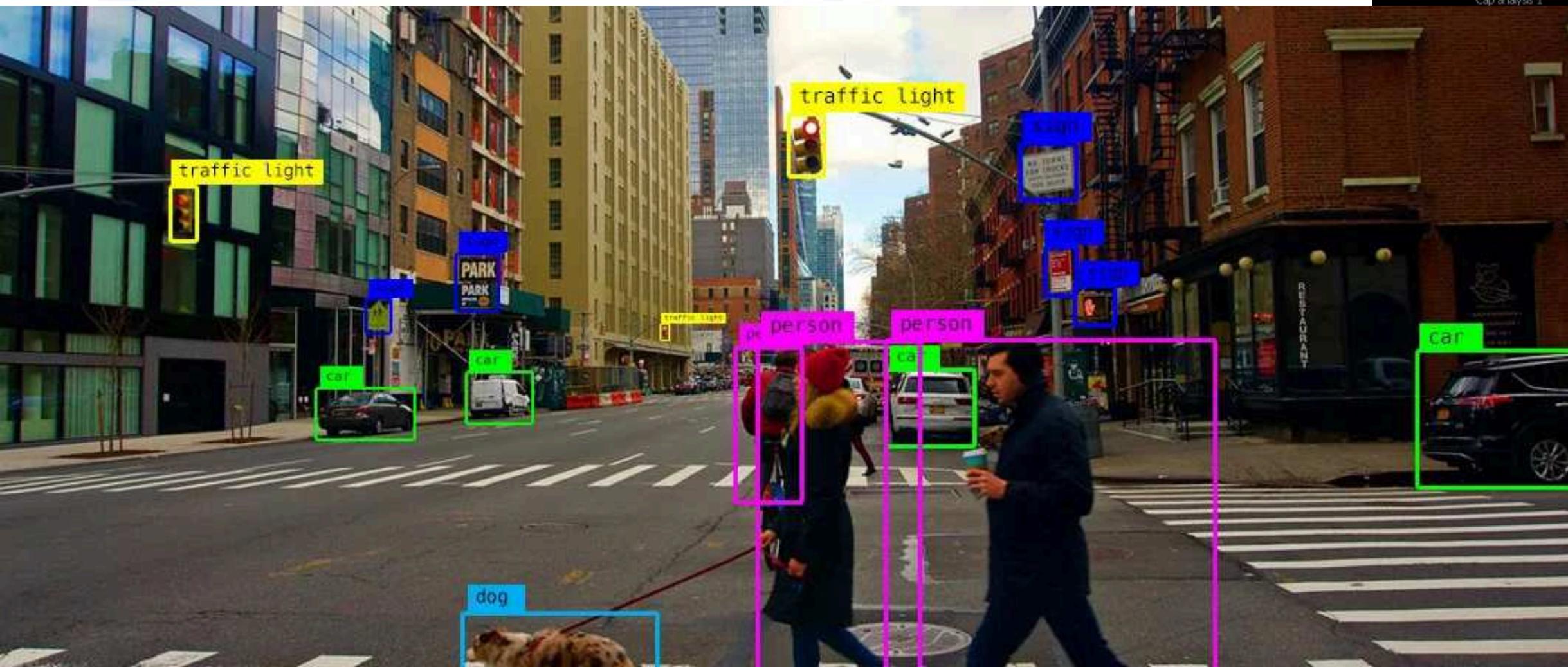
APLICACIONES



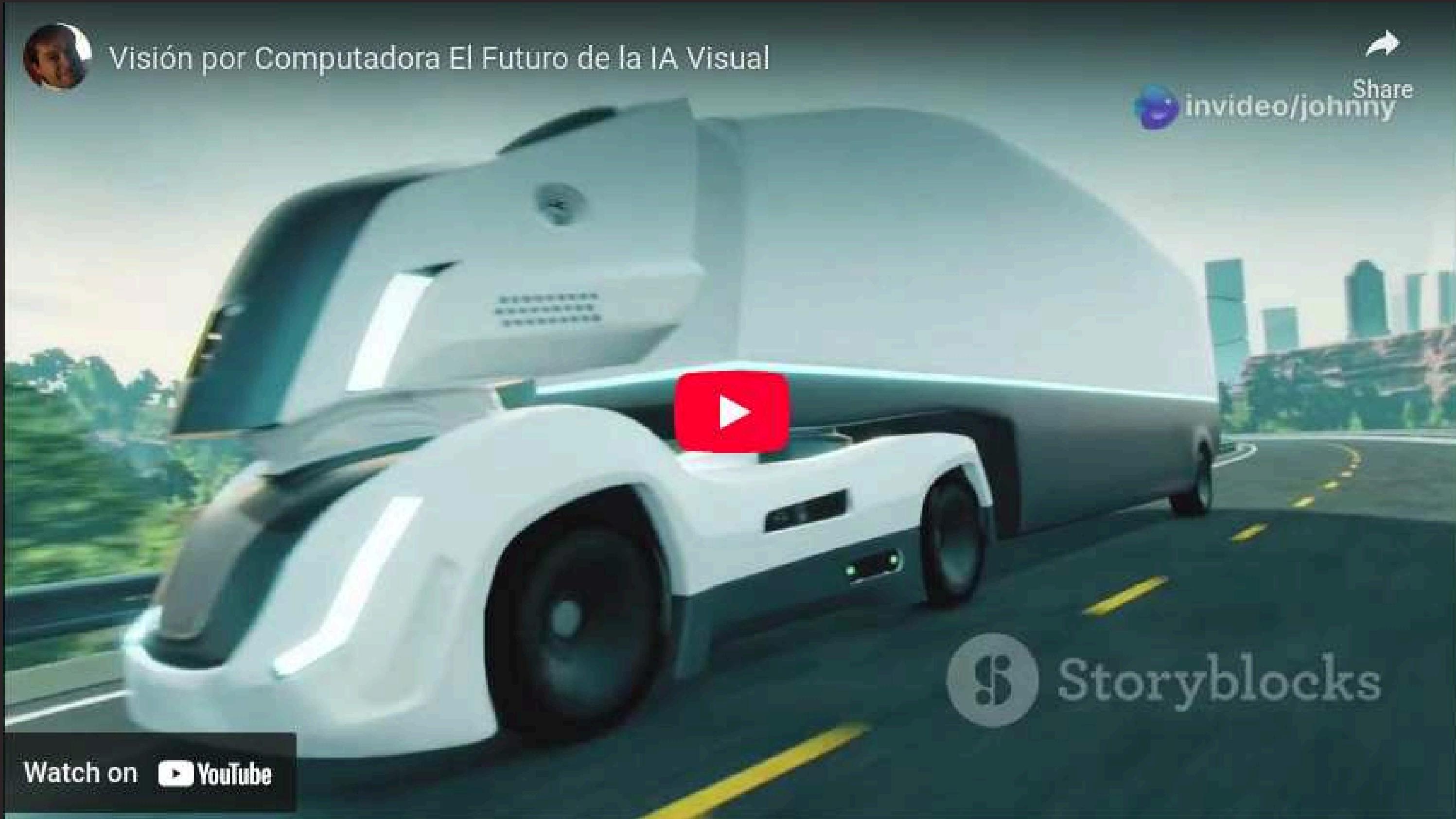
Safety light
curtain

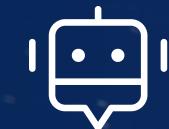


This screenshot shows a software interface for inspecting bottles. The top right corner displays the title "Unfold 4C Demo". The main area features four camera feeds labeled "Camera 1", "Camera 2", "Camera 3", and "Camera 4". Below the cameras, four images of a bottle with a red cap are shown. The left side of the screen contains a sidebar with "Results" and "Value" sections, listing various inspection parameters like "Cap analysis 1" through "Cap analysis 4" and "Unfold". The bottom right corner shows a detailed view of the bottle caps with red arrows pointing to specific defects. The status bar at the bottom indicates the date and time: "[4/19/2023 10:48:02 AM] [W6] Inspection disabled".



APLICACIONES





Universidad
Rafael Landívar

Redes Neuronales **Artificiales**

Inteligencia Artificial





Agenda

Sesión 1: Perceptrón

- *Historia*
- *Estructura*
- *Algoritmo*
- *Ejemplos y Ejercicios*
- *Limitaciones*

Sesión 2: Perceptrón Multiclasa y Multicapa

Sesión 3: Redes Neuronales



Perceptrón

En el campo del aprendizaje automático, el perceptrón es un algoritmo utilizado para el aprendizaje supervisado de clasificadores binarios, es decir, funciones que determinan si una entrada (representada como un vector de números) pertenece a una clase u otra.

Se trata de un clasificador lineal, lo que significa que realiza sus predicciones basándose en una función lineal que combina un conjunto de pesos con las características de entrada.

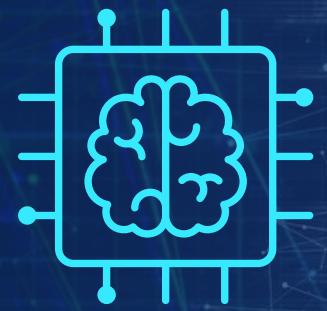


El algoritmo del perceptrón se remonta a finales de la década de 1950. Su primera implementación se realizó en hardware especializado, y fue una de las primeras redes neuronales artificiales construidas en la historia.



Perceptron

Breve Introducción Histórica



BIOINSPIRACION

El perceptrón fue propuesto por Frank Rosenblatt en 1958, inspirado por el funcionamiento de las neuronas biológicas.



DESARROLLO

Desarrollado durante la Guerra Fría con financiamiento militar (US Office of Naval Research).



PRIMER INVIERNO DE LA IA

El perceptrón fue criticado duramente por Minsky y Papert en 1969, lo que detuvo el progreso de las redes neuronales por décadas.



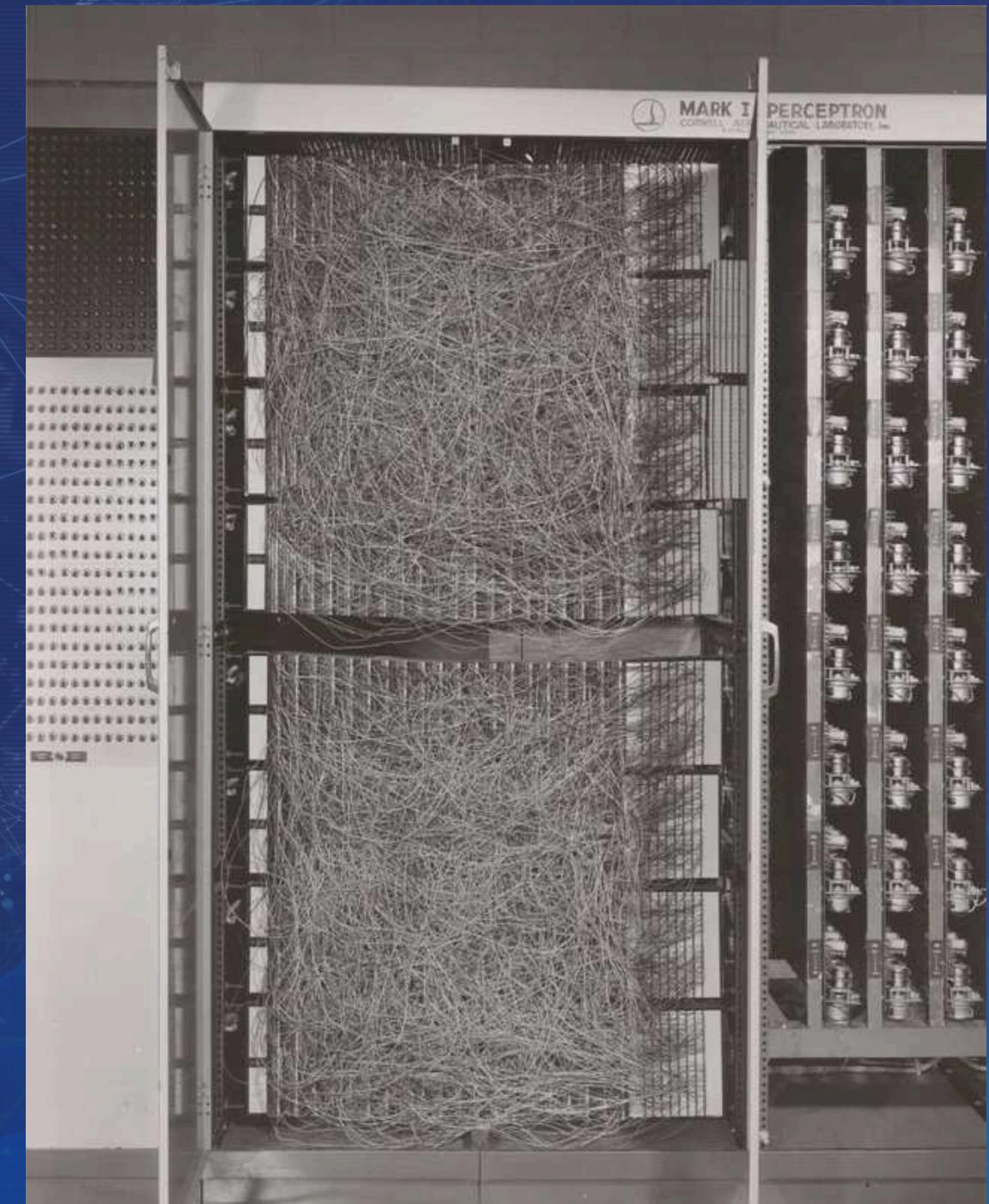
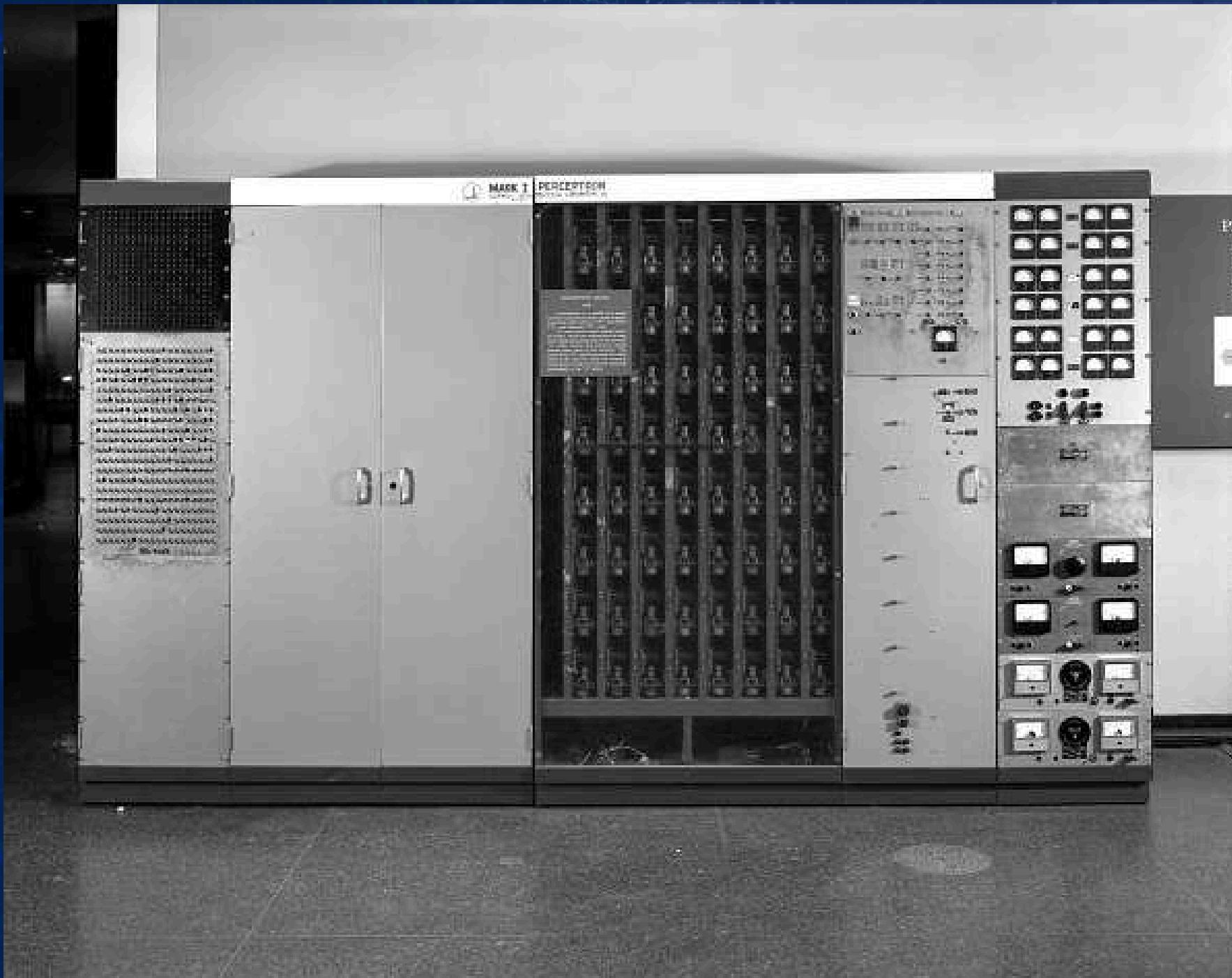
Video unavailable
[Watch on YouTube](#)





Perceptron

Breve Introducción Histórica





Perceptron

Breve Introducción Histórica

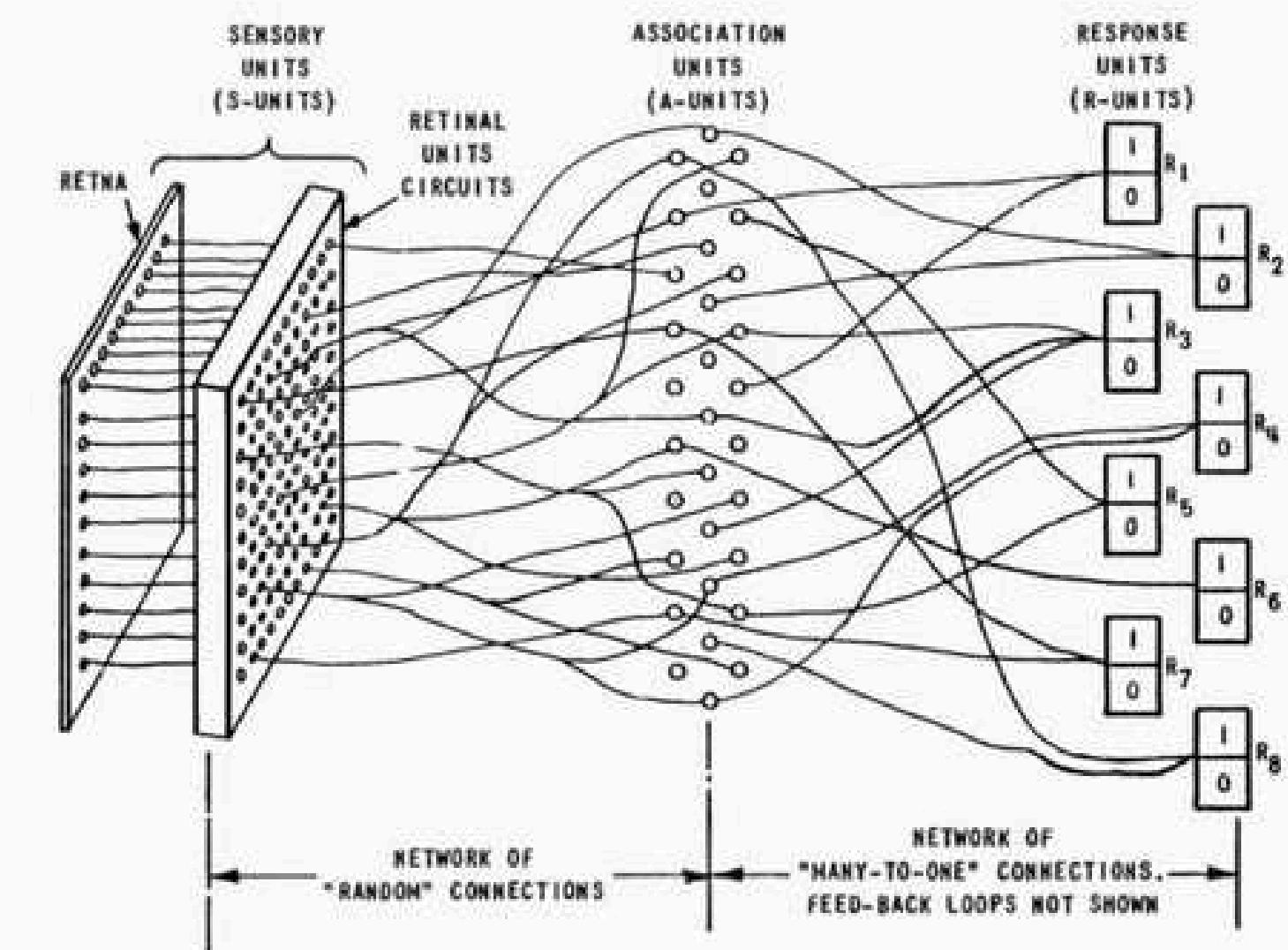
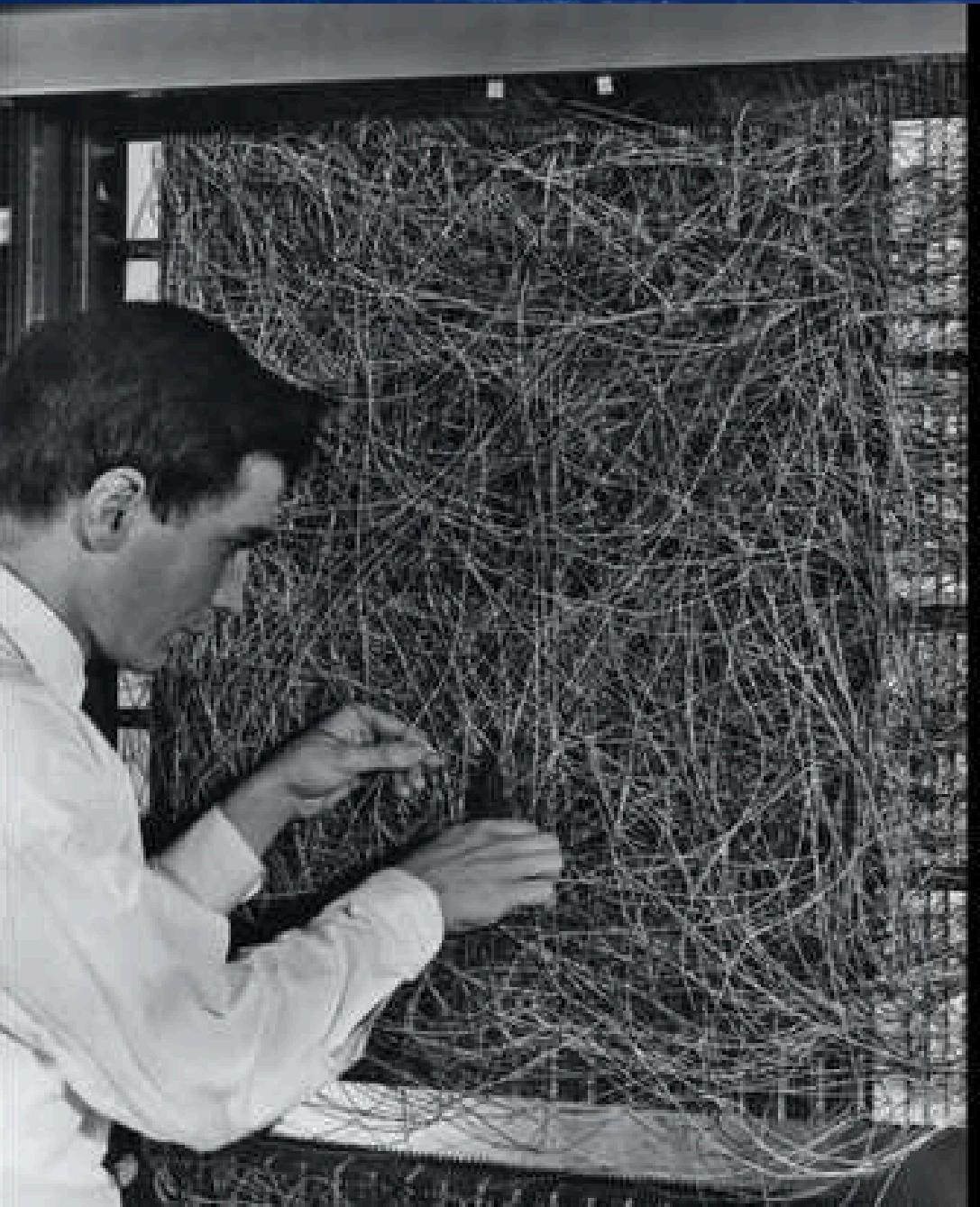


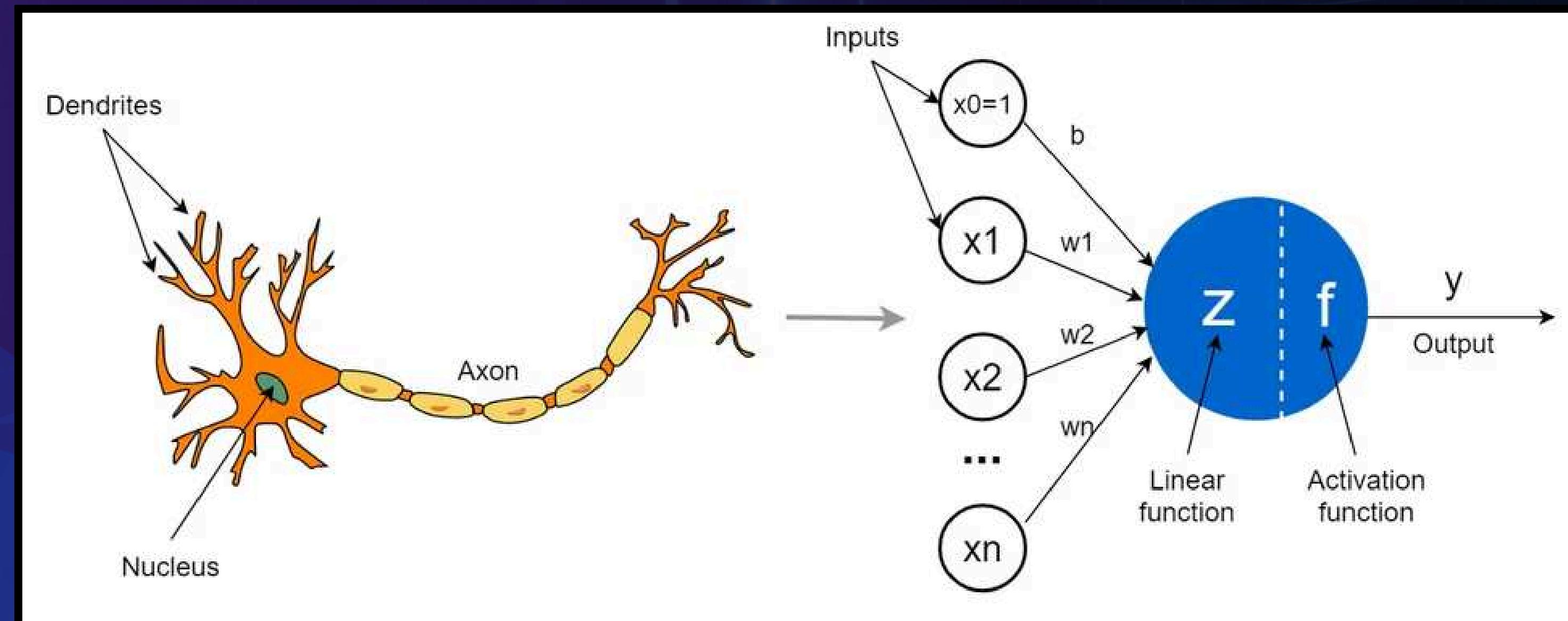
Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON

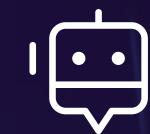


Perceptron

Estructura

- Una neurona artificial toma varias entradas numéricas, cada una con un peso asociado.
- Se calcula una suma ponderada, a la que se le aplica una función de activación para producir una salida.



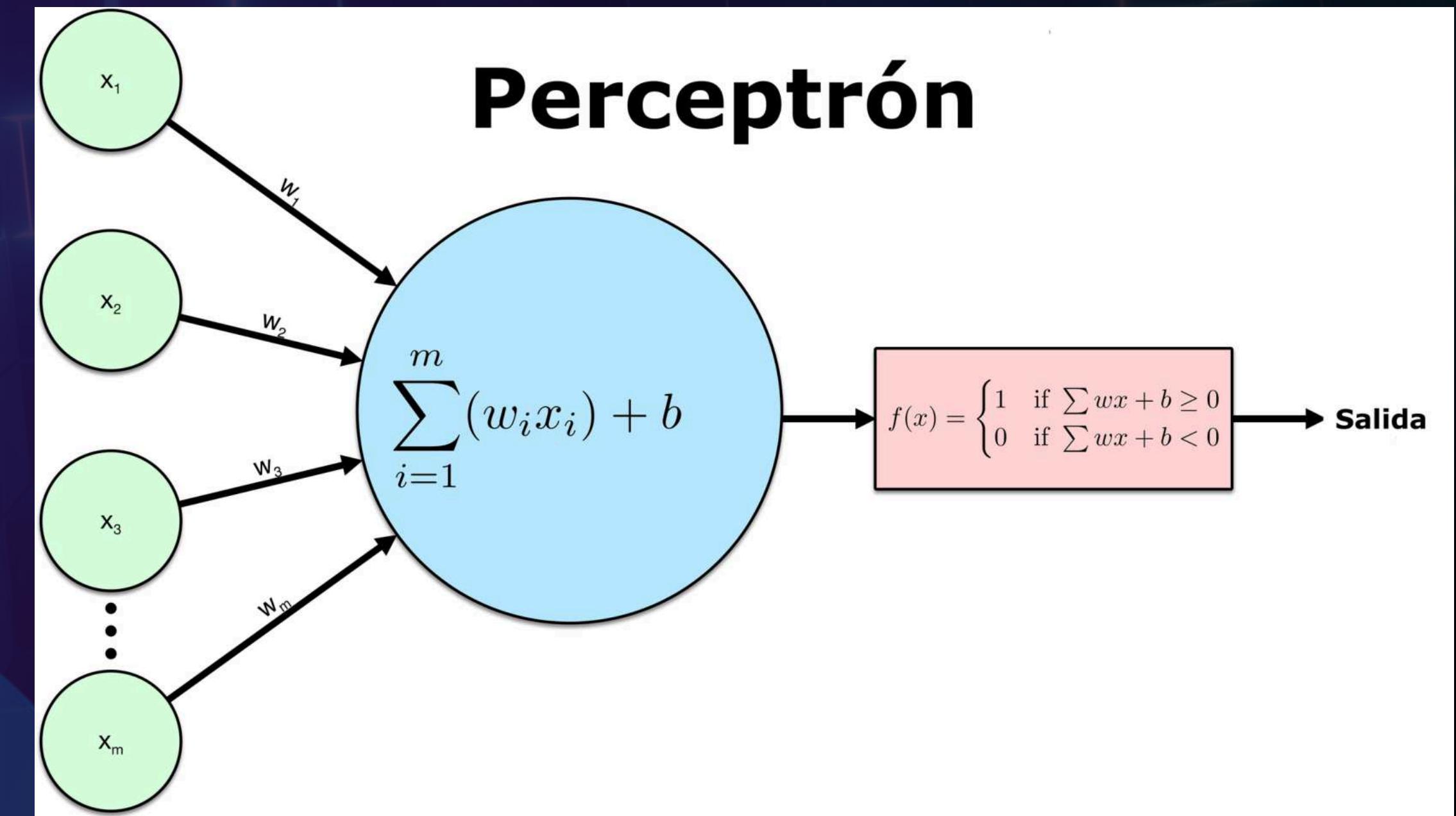
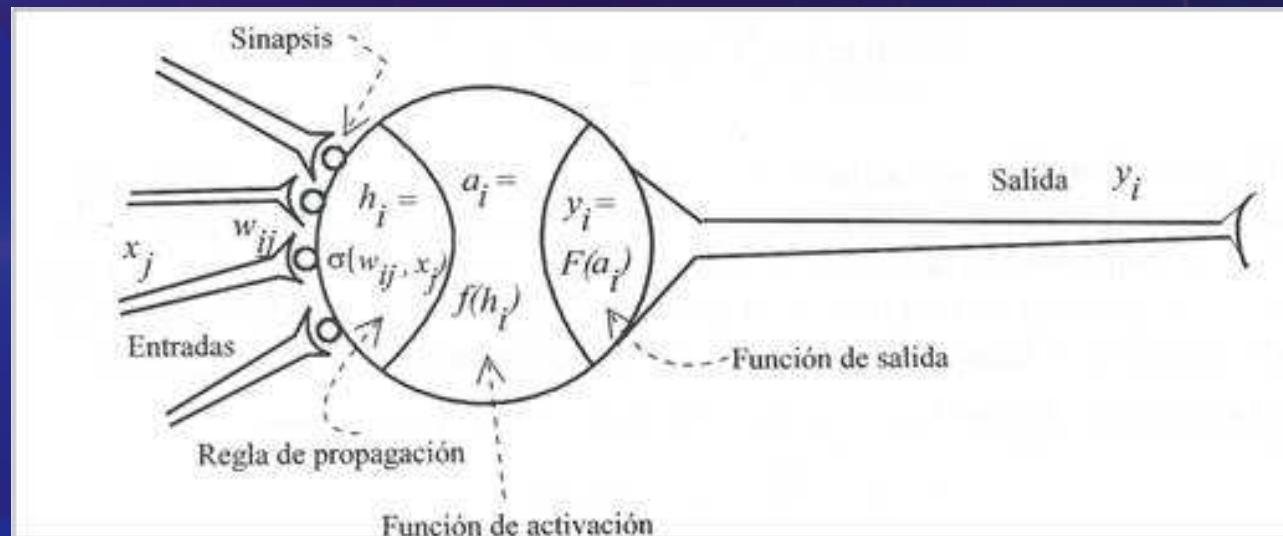


Perceptron

Estructura

Componentes clave:

1. **Entradas (x_1, x_2, \dots, x_n):** Datos o características
2. **Pesos (w_1, w_2, \dots, w_n):** Parámetros ajustables
3. **Suma ponderada**
4. **Función de activación**

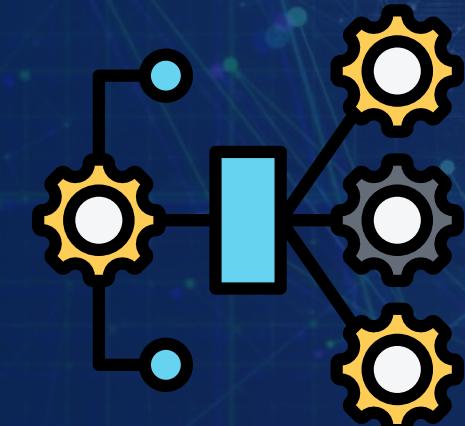




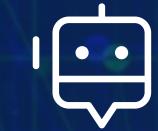
Perceptron

Algoritmo

- Entrenamiento
 - *Ajuste de pesos (Parámetros)*
- Hiperparámetros
 - *Tasa de aprendizaje η*
 - *Épocas*



- Entrenamiento
 - Se compara la salida deseada y_i con la salida estimada \hat{y}_i
 - Si hay error, se ajustan los pesos
 - El sesgo b se ajusta igual
- Hiperparámetros
 - *El error promedio $< \eta$, ó*
 - Se alcanza el máximo de épocas definido



Perceptron

Algoritmo

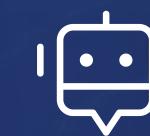
Pasos del entrenamiento:

- *Inicialización:* Pesos aleatorios pequeños (ej. [-0.5, 0.5]) y sesgo (ej. $b=0.1$).
- *Para cada muestra:*
 - Calcular salida y
 - Error: $e = y - y_{\text{pred}}$
- *Regla de actualización:*

$$w_i^{\text{nuevo}} = w_i^{\text{viejo}} + \eta \cdot e \cdot x_i$$

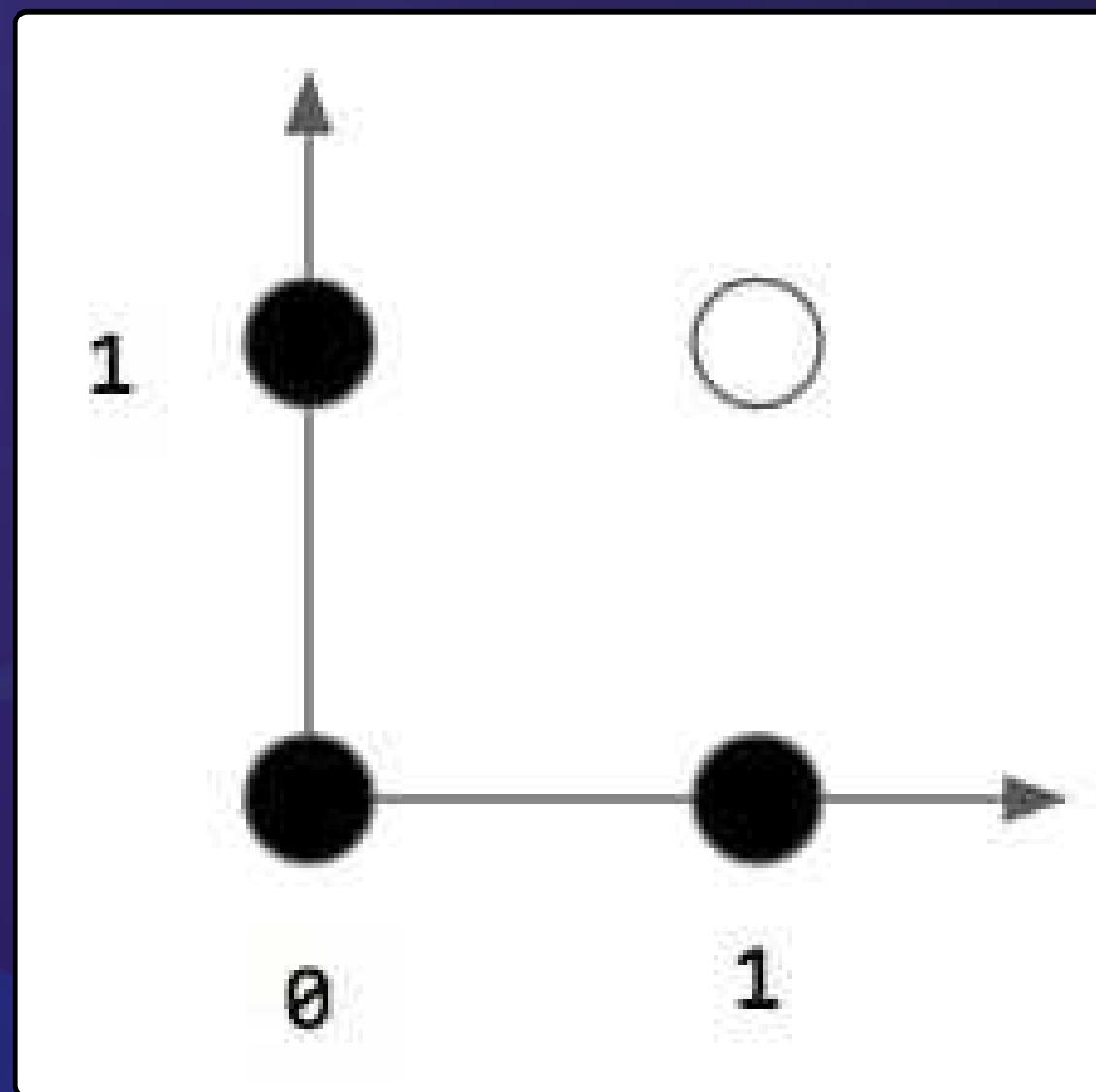
$$b_i^{\text{nuevo}} = b_i^{\text{viejo}} + \eta \cdot e \cdot 1$$

- Repetir hasta $\text{error promedio} < \text{umbral}$ o máximo épocas.



Perceptron

Ejemplo



ENTRADA X_1	ENTRADA X_2	SALIDA ESPERADA (Y)
0	0	0
0	1	0
1	0	1
1	1	1



Perceptron

Ejemplo

Para cada entrada del conjunto de entrenamiento:

Calcular:

$$z = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

Obtener la salida con la función escalón:

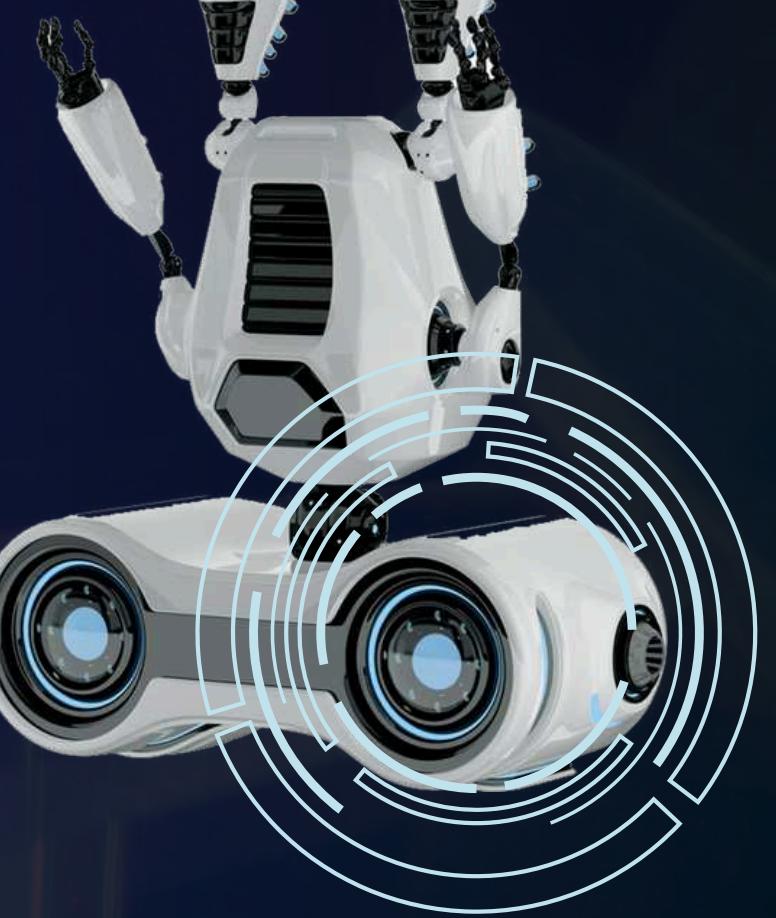
$$y_{\text{pred}} = \text{step}(z)$$

Calcular error:

$$\text{error} = y - y_{\text{pred}}$$

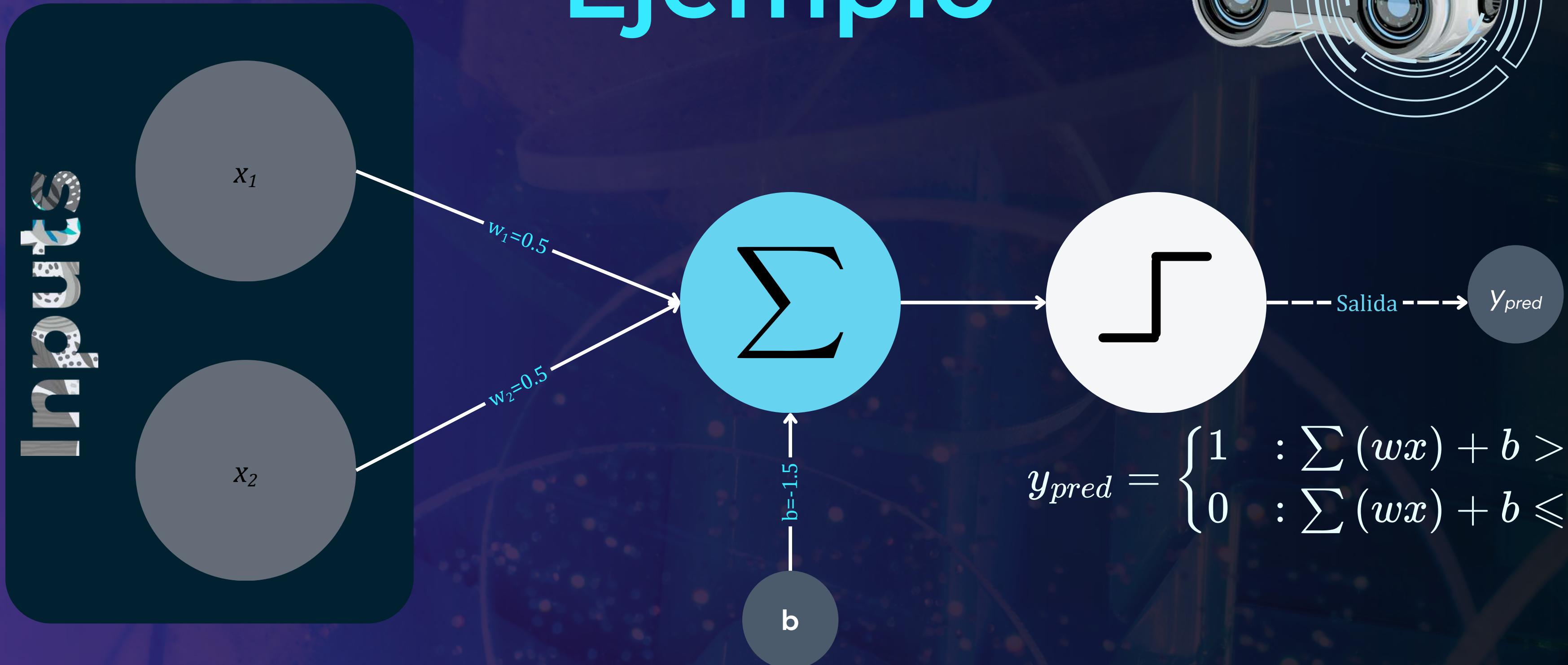
Actualizar pesos y bias:

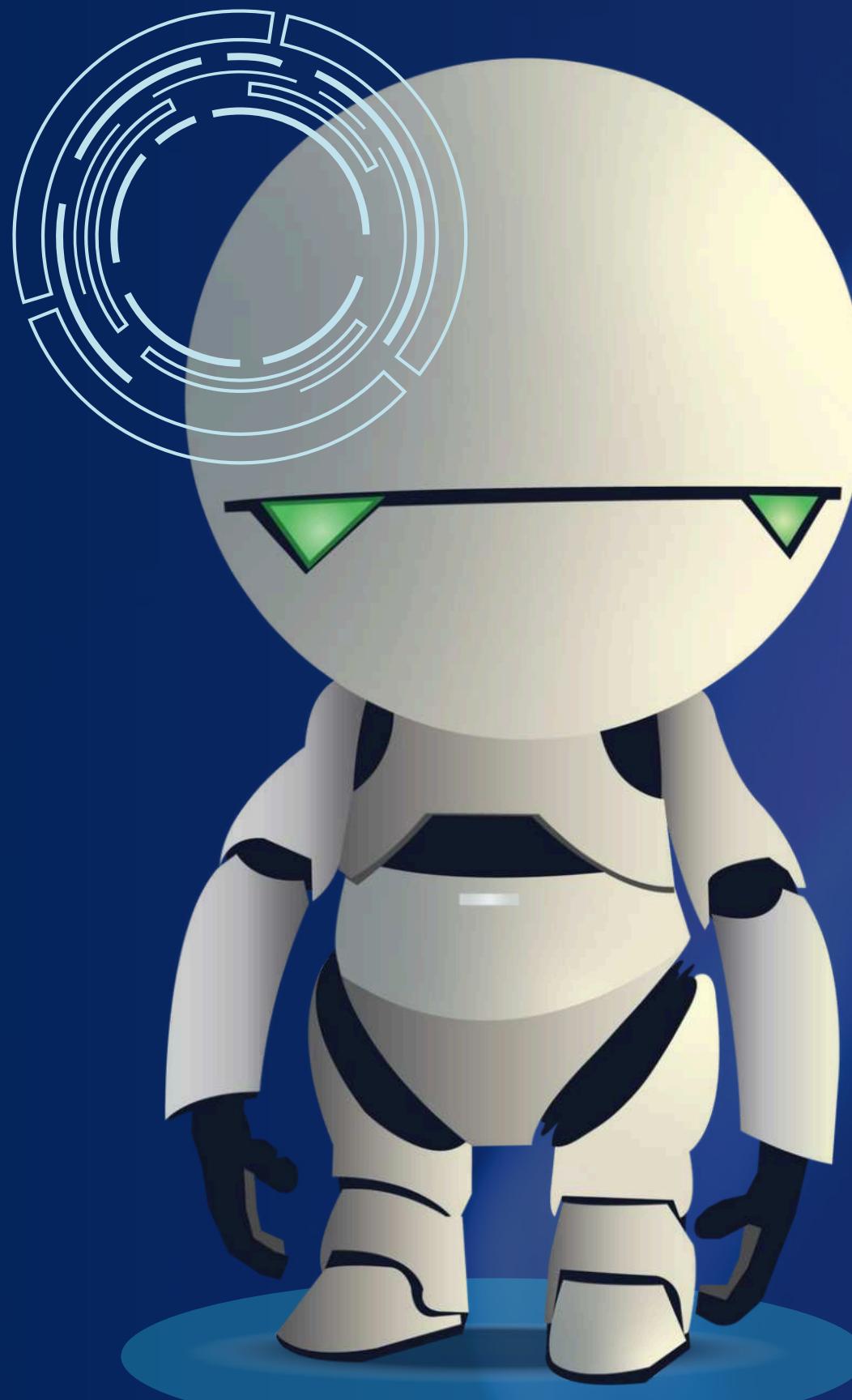
$$\begin{aligned}w_i &= w_i + \eta \cdot \text{error} \cdot x_i \\b &= b + \eta \cdot \text{error}\end{aligned}$$



Perceptron

Ejemplo





Ejemplo: Operador Lógico AND

- Pesos: $w_1=0.5, w_2=0.5$
- Bias (sesgo): $b = -1.5$
- Tasa de aprendizaje: $\eta=1$
- Función de activación:
 - Escalón estricto

$$y_{pred} = \begin{cases} 1 & : z > 0 \\ 0 & : z \leq 0 \end{cases}$$



Perceptron

Ejemplo

Época 1

1) Ejemplo (0, 0) $y = 0$

$$z=0.5(0)+0.5(0)+(-1.5)=-1.5 \quad y_{pred}=0 \quad \text{Error} = 0-0 = 0 \rightarrow \text{No se actualiza}$$

2) Ejemplo (0, 1) $y = 0$

$$z=0.5(0)+0.5(1)+(-1.5)=-1.0 \quad y_{pred}=0 \quad \text{Error} = 0-0 = 0 \rightarrow \text{No se actualiza}$$

3) Ejemplo (1, 0) $y = 0$

$$z=0.5(1)+0.5(0)+(-1.5)=-1.0 \quad y_{pred}=0 \quad \text{Error} = 0-0 = 0 \rightarrow \text{No se actualiza}$$

4) Ejemplo (1, 1) $y = 1$

$$z=0.5(1)+0.5(1)+(-1.5)=-0.5 \quad y_{pred}=0 \quad \text{Error} = 1-0 = 1 \rightarrow \text{(error)}$$



Perceptron

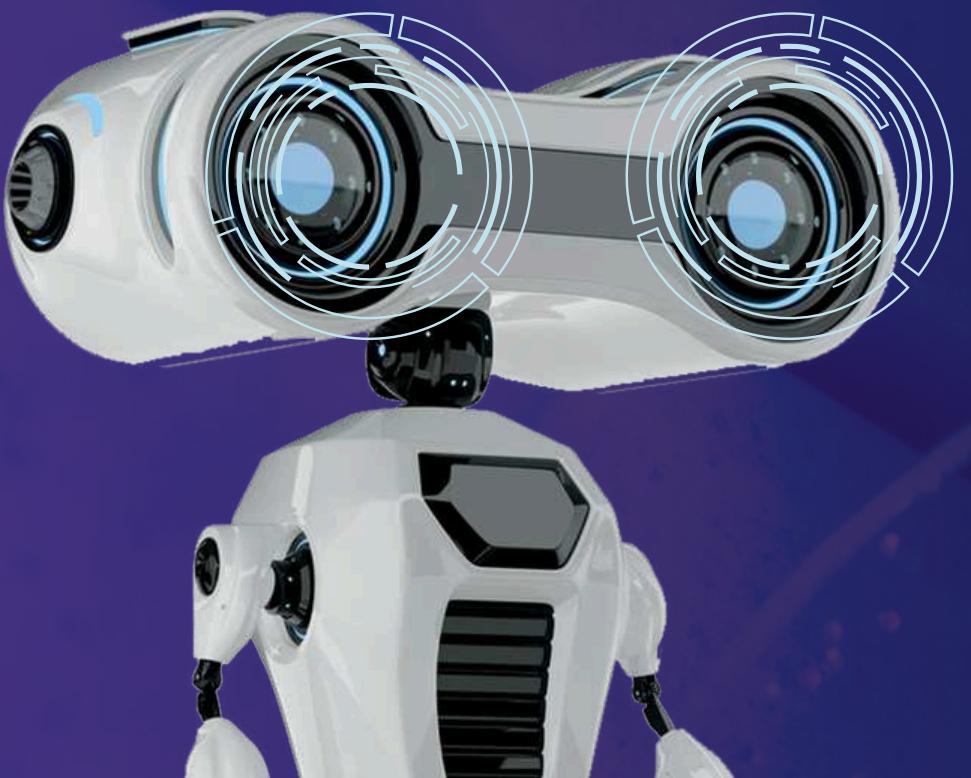
Ejemplo

Actualización:

$$w_1 = 0.5 + 1(1 - 0)(1) = 1.5$$

$$w_2 = 0.5 + 1(1 - 0)(1) = 1.5$$

$$b = -1.5 + 1(1 - 0) = -0.5$$





Perceptron

Ejemplo

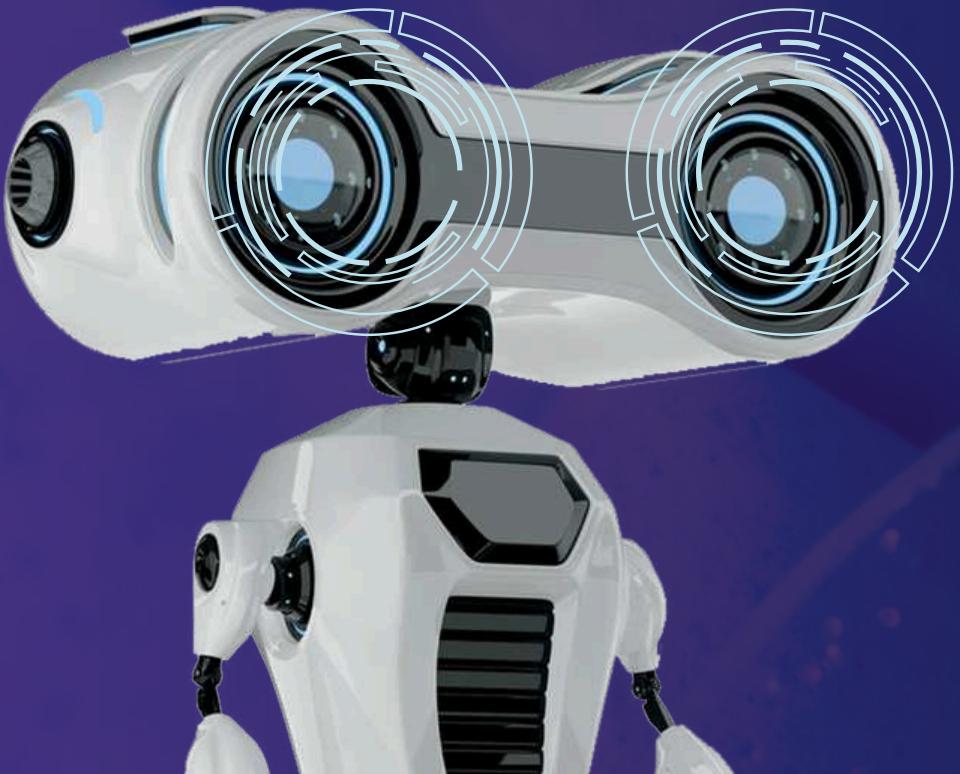
Época 2

1) Ejemplo (0, 0) $y = 0$

$$z=1.5(0)+1.5(0)+(-0.5)=-0.5 \quad y_{pred}=0 \quad Error = 0-0 = 0 \rightarrow \text{No se actualiza}$$

2) Ejemplo (0, 1) $y = 0$

$$z=1.5(0)+1.5(1)+(-0.5)=1.0 \quad y_{pred}=1 \quad Error = 0-1 = -1 \rightarrow \underline{\text{(error)}}.$$





Perceptron

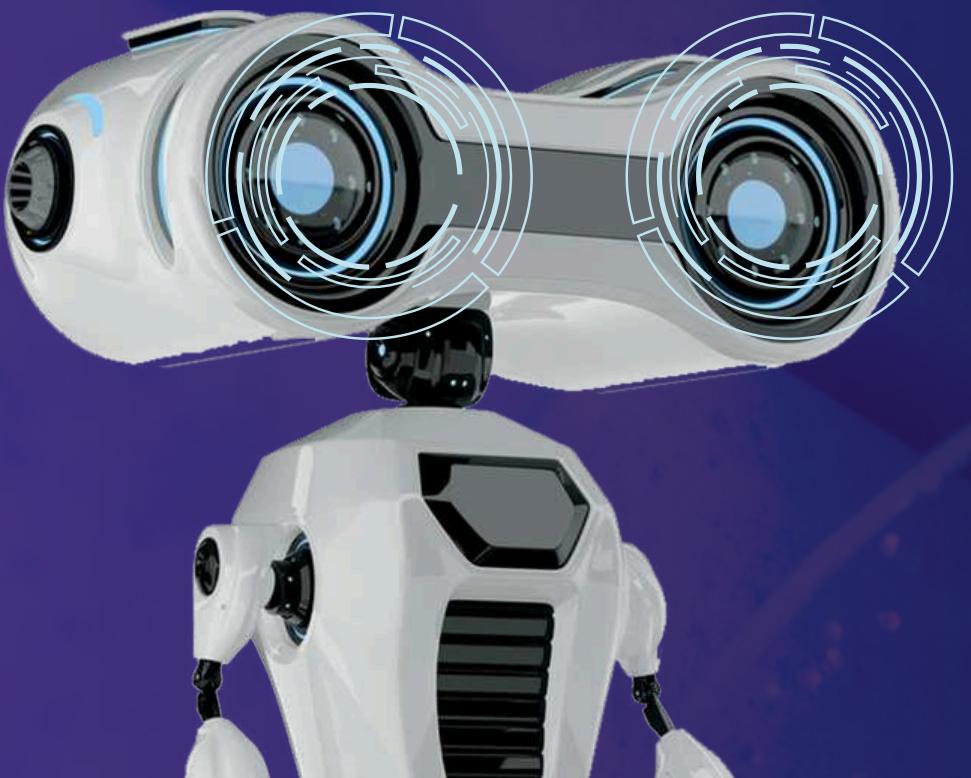
Ejemplo

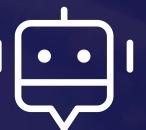
Actualización:

$$w_1 = 1.5 + 1(0-1)(0) = 1.5$$

$$w_2 = 1.5 + 1(0-1)(1) = 0.5$$

$$b = -0.5 + 1(0-1) = -1.5$$





Perceptron

Ejemplo

Época 2

1) Ejemplo (0, 0) $y = 0$

$$z=1.5(0)+1.5(0)+(-0.5)=-1 \quad y_{pred}=0 \quad Error = 0-0 = 0 \rightarrow \text{No se actualiza}$$

2) Ejemplo (0, 1) $y = 0$

$$z=1.5(0)+1.5(1)+(-0.5)=-0.5 \quad y_{pred}=1 \quad Error = 0-1 = -1 \rightarrow (\text{error})$$

Nuevos pesos:

$$w_1=1.5$$

$$w_2=0.5$$

$$b=-1.5$$

3) Ejemplo (1, 0) $y = 0$

$$z=1.5(1)+0.5(0)+(-1.5)=0 \quad y_{pred}=0 \quad Error = 0-0 = 0 \rightarrow \text{No se actualiza}$$

4) Ejemplo (1, 1) $y = 1$

$$z=1.5(1)+0.5(1)+(-1.5)=0.5 \quad y_{pred}=1 \quad Error = 1-1 = 0 \rightarrow \text{No se actualiza}$$



Perceptron

Funciones Step

Variante	Fórmula	Valor en z=0	Descripción
Clásica	$f(x) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$	f(0)=1	Activa en cero y valores positivos
Strict step	$f(x) = \begin{cases} 0 & z \leq 0 \\ 1 & z > 0 \end{cases}$	f(0)=0	Solo activa con valores estrictamente positivos



Perceptron

Ejercicio

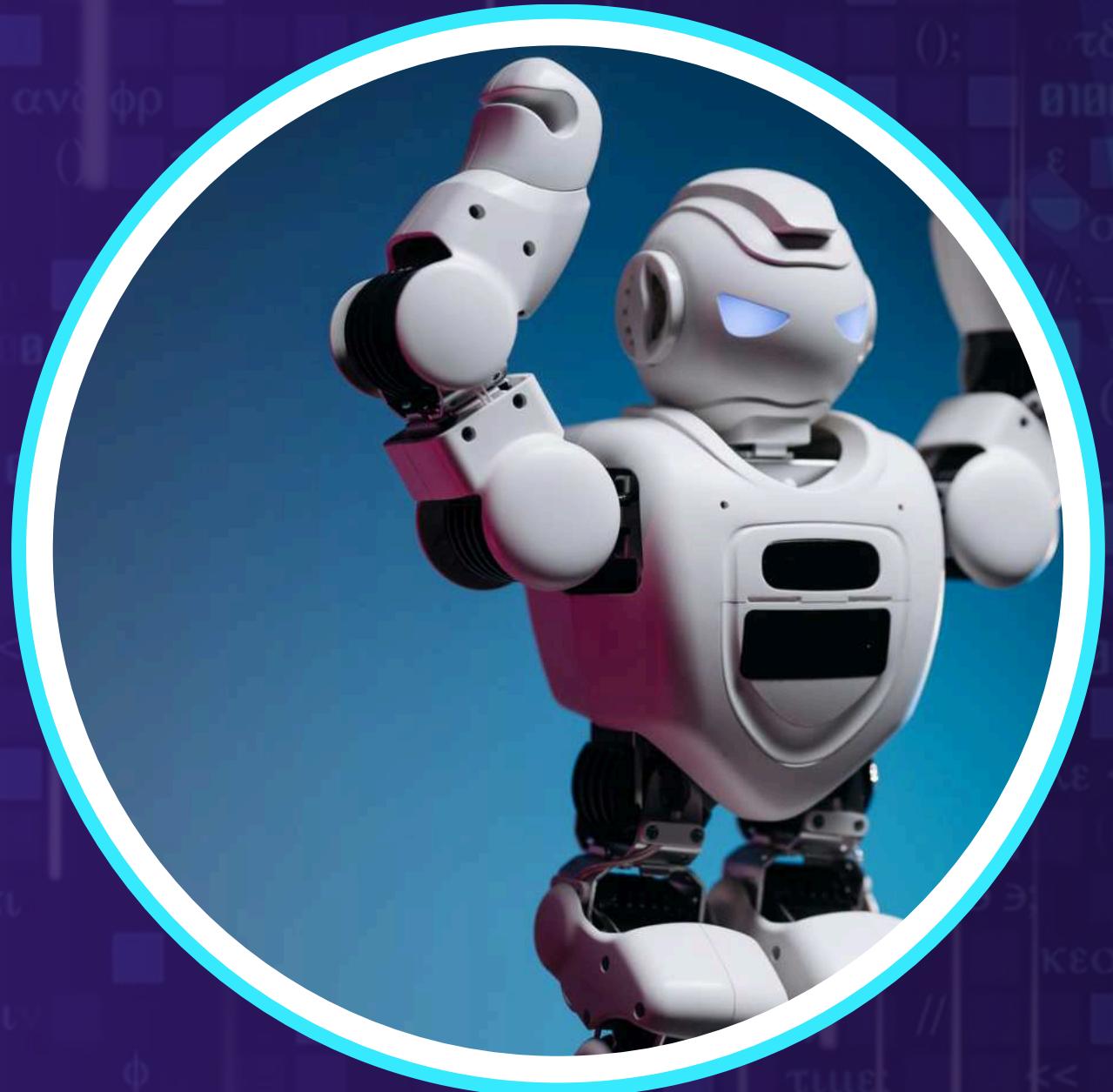
ENTRADA X ₁	ENTRADA X ₂	SALIDA ESPERADA (Y)
0	0	0
0	1	0
1	0	0
1	1	1

- Pesos: $w_1=0.5, w_2=0.5$
- Bias (sesgo): $b = -1.5$
- Tasa de aprendizaje: $\eta=1$
- Función de activación:
 - Escalón Clásico

$$y_{pred} = \begin{cases} 1 & : z \geqslant 0 \\ 0 & : z < 0 \end{cases}$$



Variantes



Perceptrón Online (Aprendizaje Secuencial o Incremental)

- Se actualizan los pesos cada vez que se procesa un ejemplo de entrenamiento.
- Ventajas:
 - Reacciona rápidamente a nuevos datos.
 - Ideal para flujos de datos o aprendizaje en tiempo real.

Perceptrón Offline (Aprendizaje por Lotes o Batch)

- Se procesa todo el conjunto de entrenamiento antes de actualizar los pesos.
- Ventajas:
 - Puede ser más estable.
 - Reduce ruido de datos individuales.
- En el caso del perceptrón: se suele simular ejecutando una época completa (pasar por todos los datos) y luego realizar una actualización basada en el conjunto de errores.

Variantes



¿Cuál usar?

- En la práctica educativa y con datasets pequeños (como los de OR, AND, XOR), el modo online es el más común y más fácil de entender.
- En problemas más grandes o si hay mucha variabilidad en los datos, el modo offline o mini-batch puede ser preferido.





Limitaciones Clave

1. **Separabilidad lineal:** Solo resuelve problemas con frontera decisional recta.
2. **XOR problem:** Falla completamente (motivó redes multicapa).
3. **Datos ruidosos:** Sensible a outliers (no tiene regularización).
4. **Clases desbalanceadas:** Tiende a favorecer clases mayoritarias.



Ejercicio en Casa:

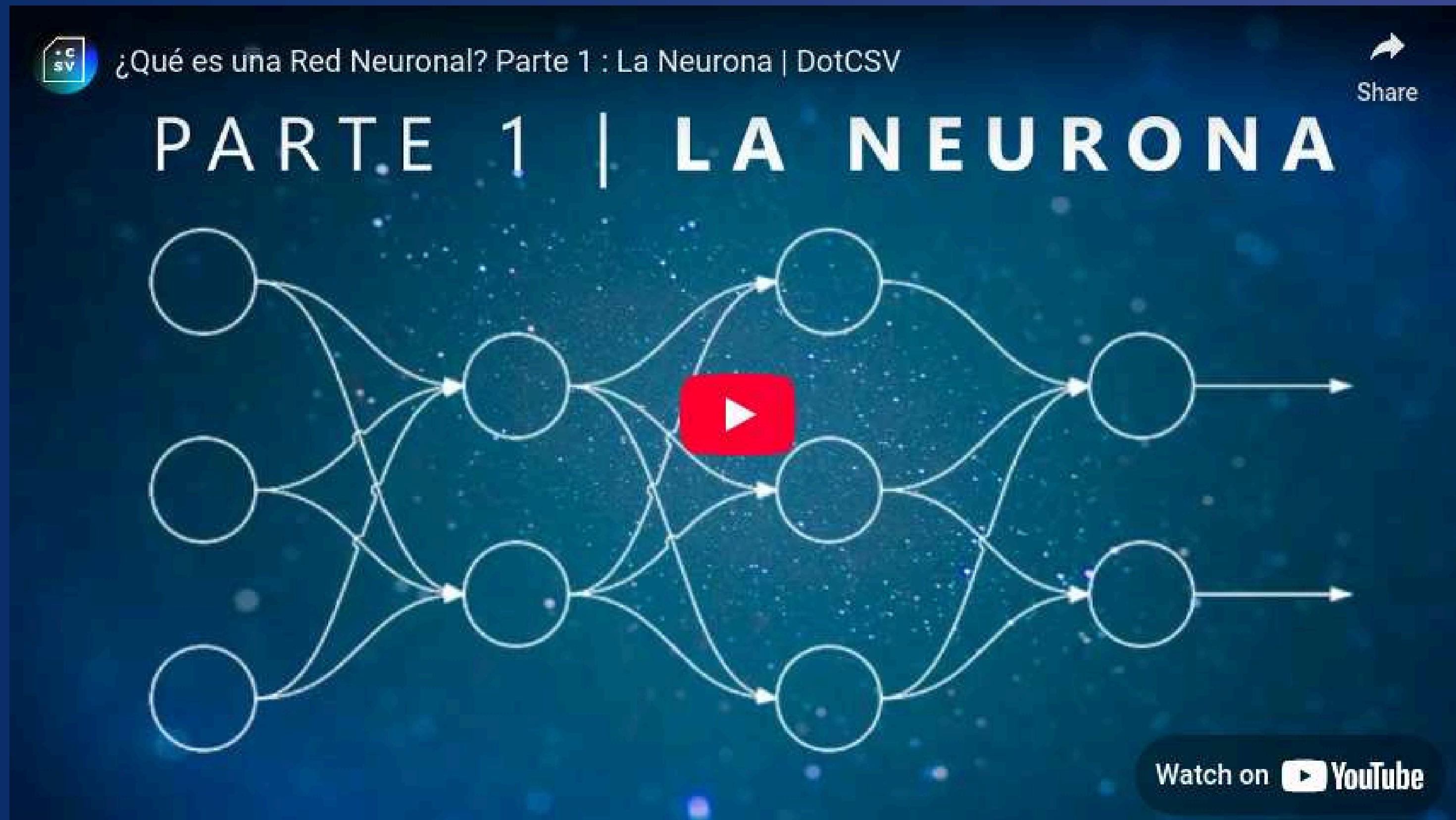
ENTRADA X ₁	ENTRADA X ₂	SALIDA ESPERADA (Y)
0	0	0
0	1	1
1	0	1
1	1	0

- Pesos: $w_1=0.5, w_2=0.5$
- Bias (sesgo): $b = -1.5$
- Tasa de aprendizaje: $\eta=1$
- Función de activación:
 - Escalón Clásico

$$y_{pred} = \begin{cases} 1 & : z \geqslant 0 \\ 0 & : z < 0 \end{cases}$$



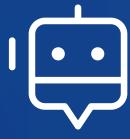
Resumen





Conclusión

El perceptrón sentó las bases para el deep learning, pero su incapacidad con problemas no lineales llevó al desarrollo de arquitecturas más complejas (MLP, CNN). Su simplicidad lo hace ideal para introducir conceptos de aprendizaje automático supervisado.



Próxima sesión:

Perceptrón Multiclasificación

Perceptrón Multicapa:

- *Topología*
- *Algoritmo: Forward Propagation*

Funciones de Activación



Universidad
Rafael Landívar

Redes Neuronales **Artificiales**

Inteligencia Artificial





Agenda

Sesión 1: Perceptrón

Sesión 2: Perceptrón Multiclasa y Multicapa

Sesión 3: Redes Neuronales Artificiales

- Modelos de Caja Negra
- Deep Learning
- RNA
 - Backpropagation
 - Tipos
 - Arquitecturas





Redes Neuronales: De la Neurona al Perceptrón Multicapa en 9 minutos. El problema X...



Share

REDES NEURONALES



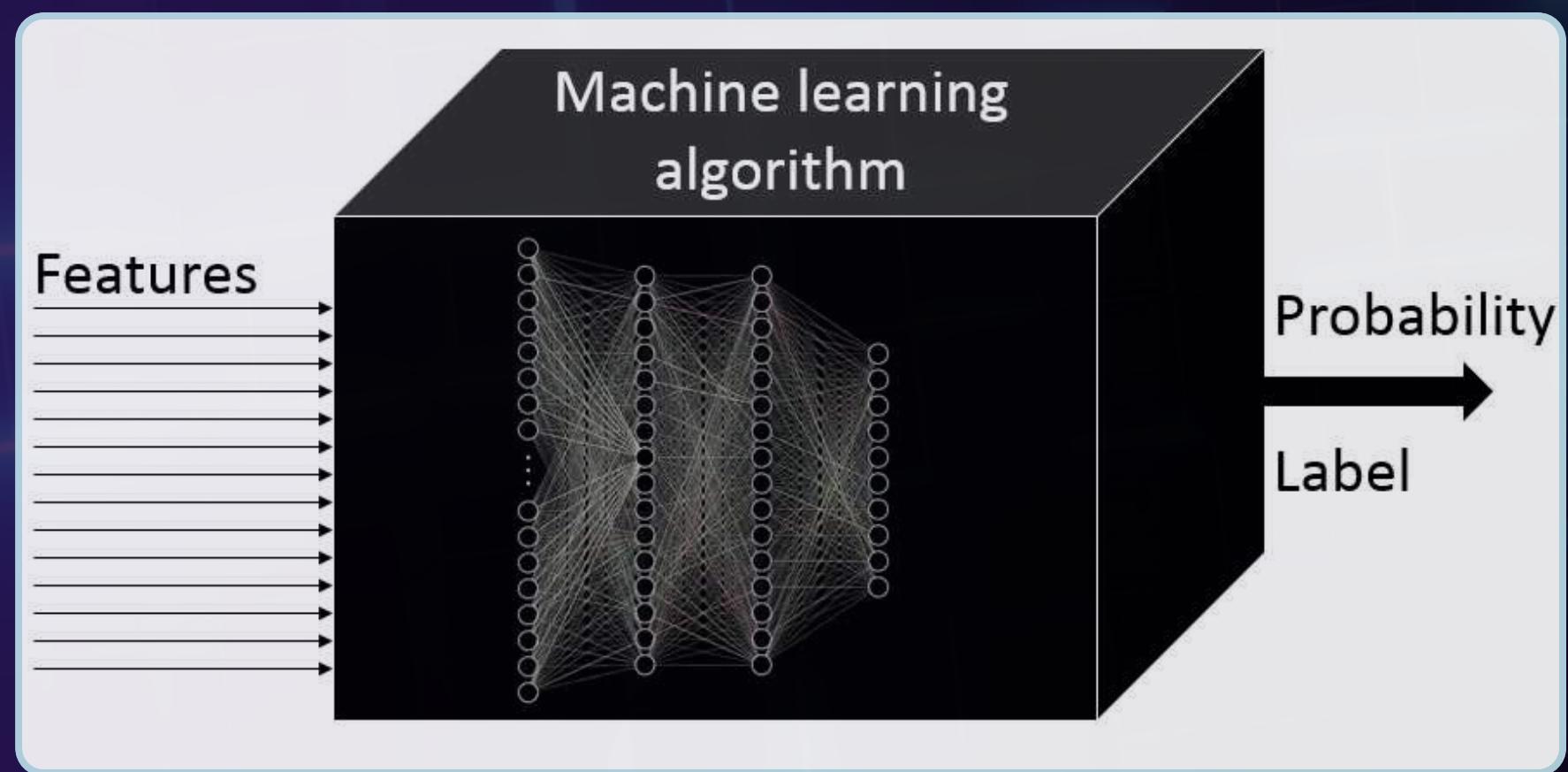
desde CERO

Watch on



Modelos de Caja Negra

Los modelos de caja negra son sistemas computacionales cuyo funcionamiento interno es desconocido o incomprendible para los usuarios, como los utilizados por Netflix o Facebook para recomendaciones

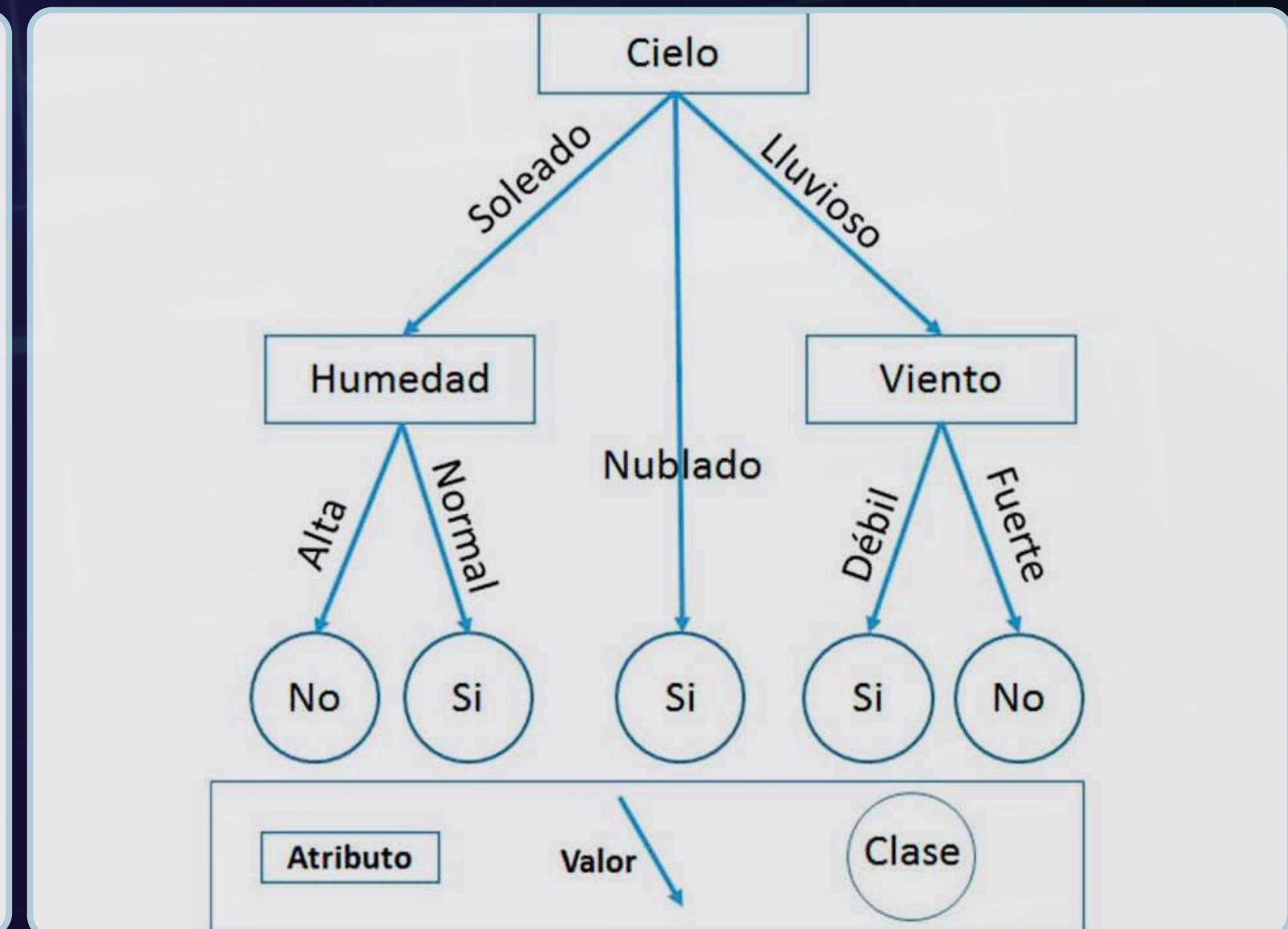
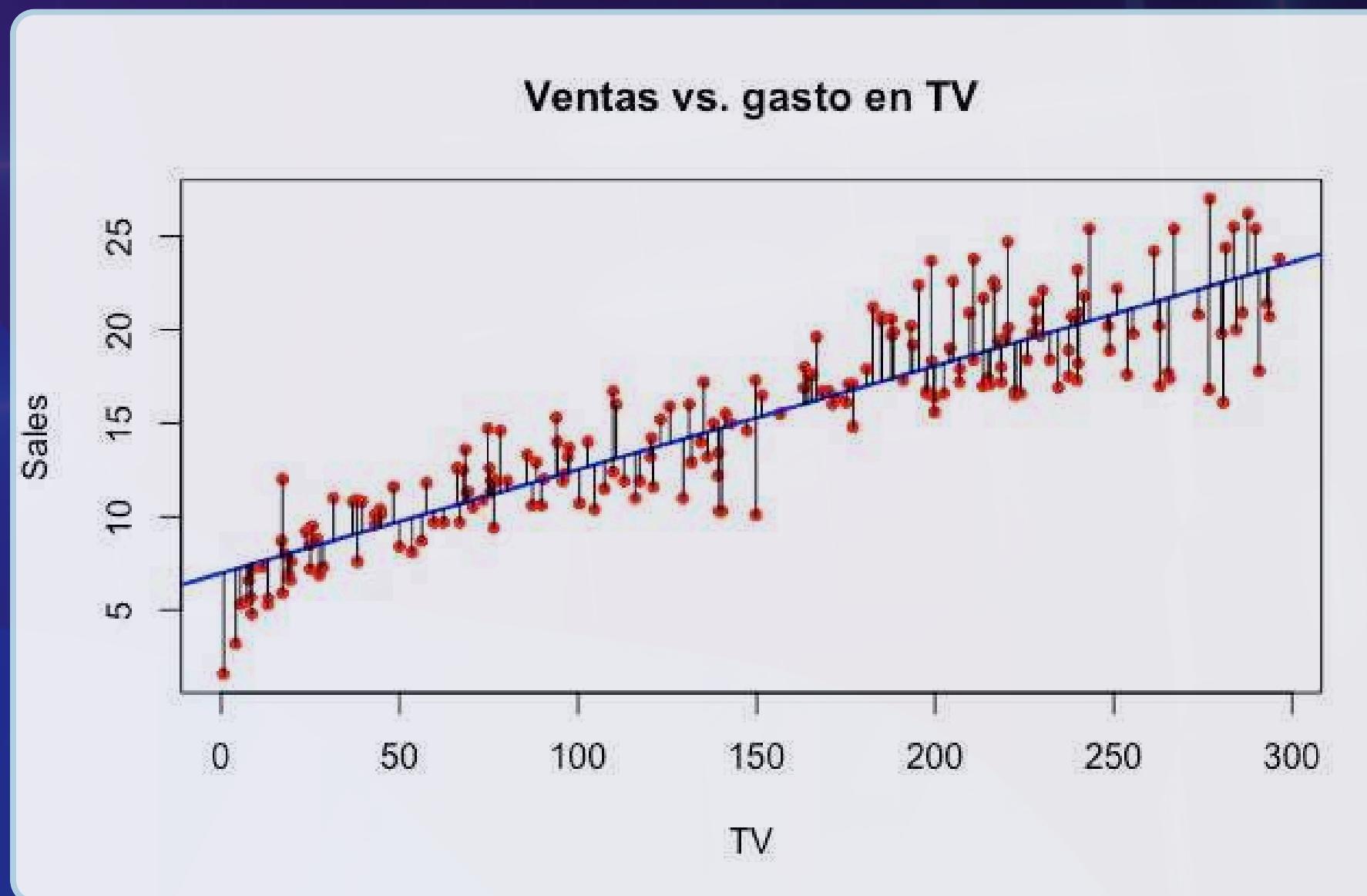


Aunque puede generar predicciones precisas, no es posible interpretar fácilmente cómo llega a sus conclusiones.



Modelos Interpretables

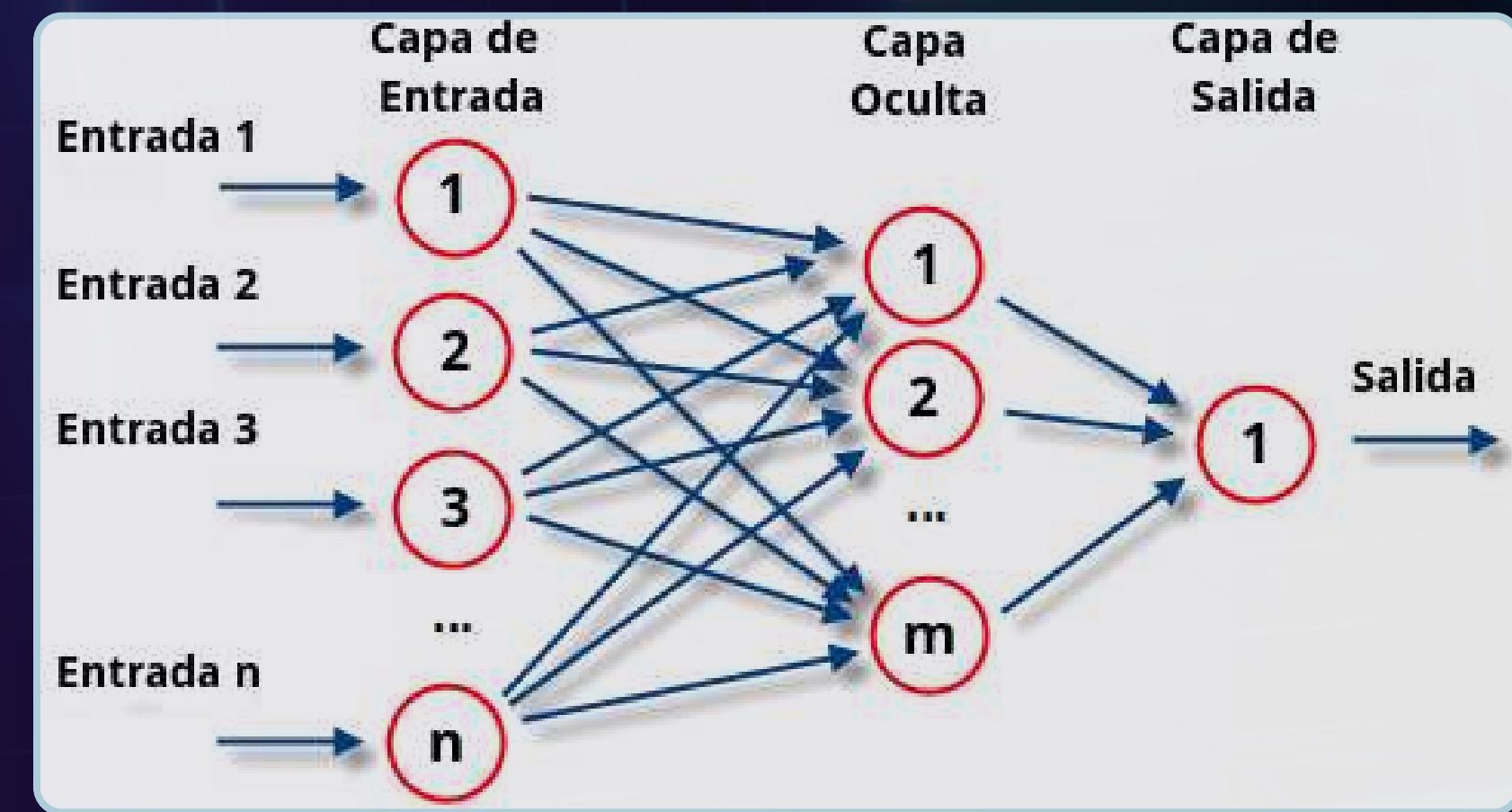
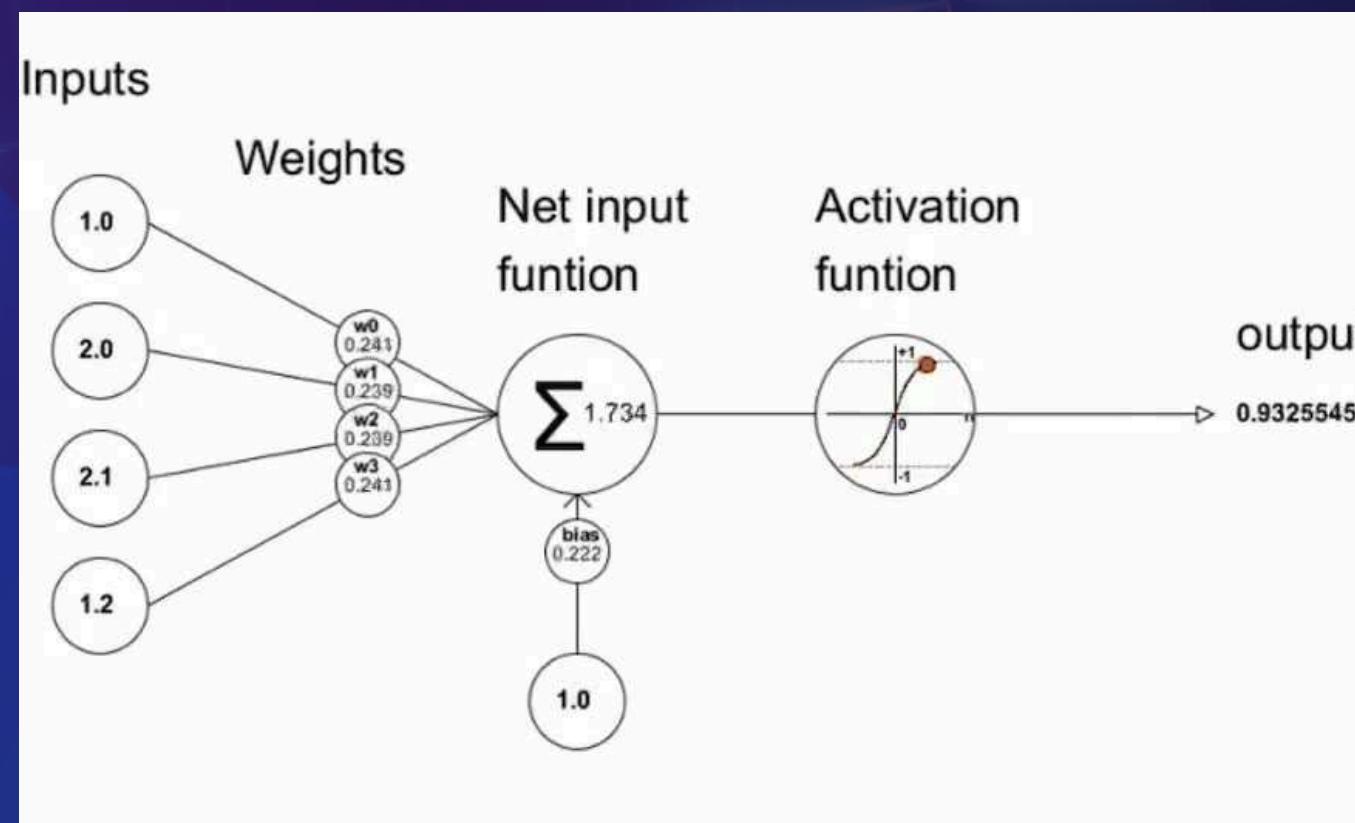
Un modelo interpretable permite a los usuarios entender cómo se toman las decisiones. Ejemplos clásicos son la regresión lineal, la regresión logística y los árboles de decisión, donde cada parámetro tiene una interpretación clara y rastreable.

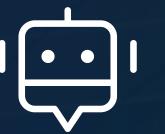


De MLP a redes profundas: ¿Por qué más capas?

El MLP es una red neuronal artificial básica con:

- Una capa de entrada
- Una o más capas ocultas
- Una capa de salida
- Cada capa tiene neuronas que realizan operaciones lineales seguidas de una función de activación (como ReLU, tanh o sigmoid).

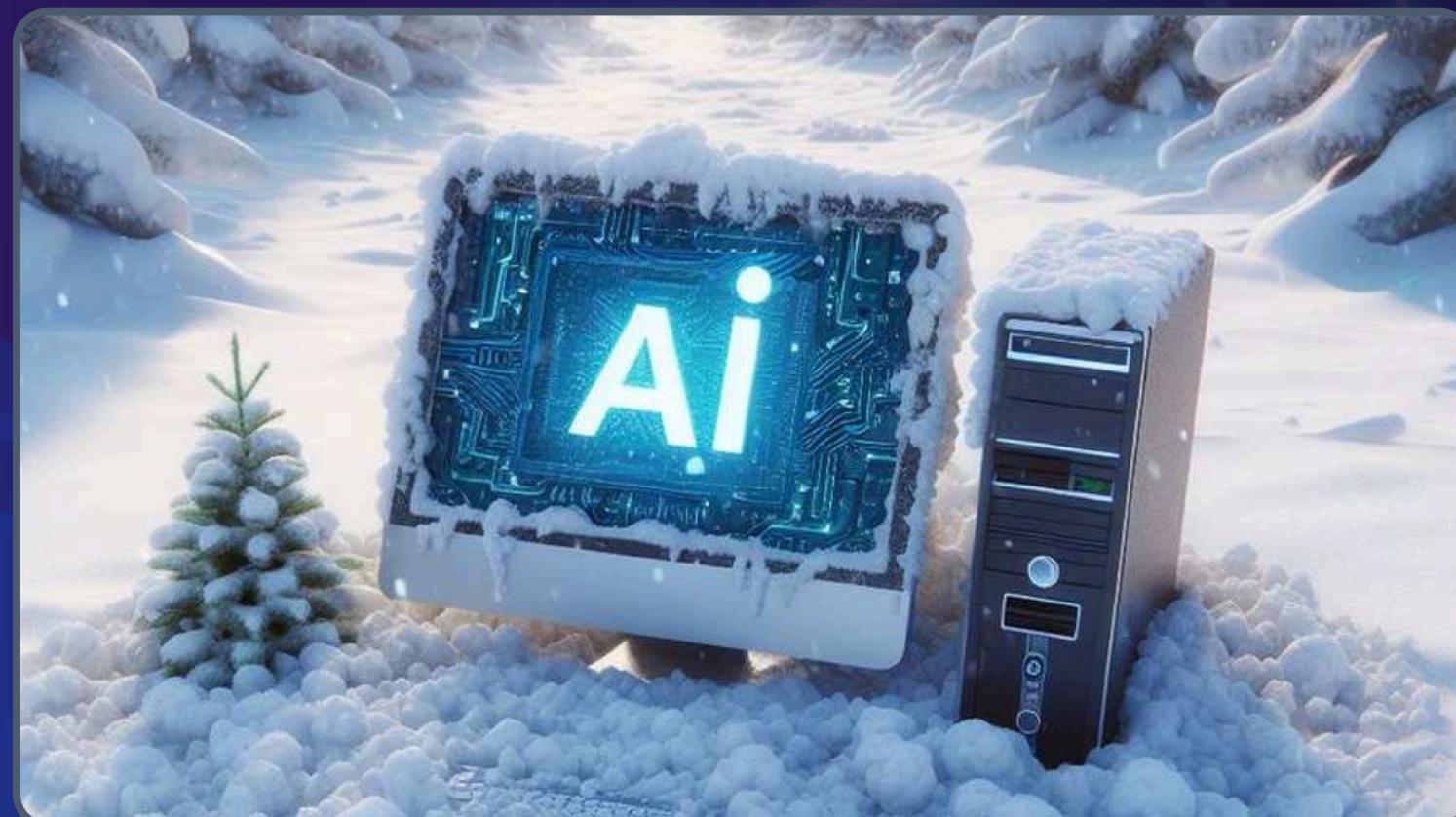




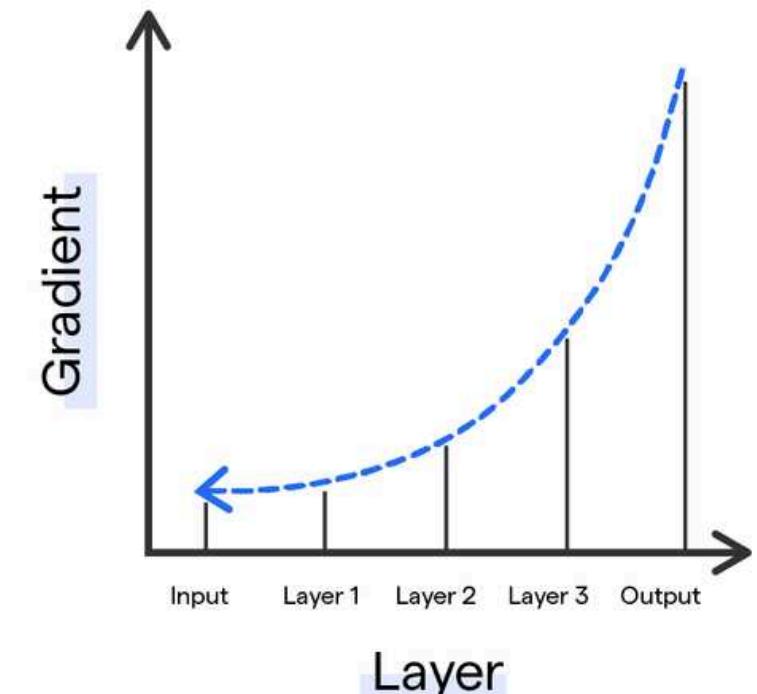
De MLP a redes profundas: ¿Por qué más capas?

Limitaciones del MLP clásico:

- Limitantes descubiertos que generaron el segundo invierno...
- No escala bien a tareas de alta dimensión como imágenes o video.
- Tiene dificultades para representar relaciones jerárquicas o no lineales complejas.
- Necesita una gran cantidad de neuronas si no tiene profundidad.



Vanishing Gradient Problem





Segundo invierno de las redes neuronales

- Período de estancamiento en la investigación y desarrollo
- Finales de los años 80 y principios de los 2000
- Enfrentaron críticas y desinterés debido a varias limitaciones técnicas y conceptuales

¿Qué causó este segundo invierno?

- **Limitaciones del algoritmo de backpropagation:** Algoritmo introducido en los años 70 y popularizado en 1986, presentaba el problema del gradiente desvanecido dificultaba el entrenamiento de redes profundas
- **Falta de poder computacional:** Las computadoras de la época no contaban con la capacidad de procesamiento
- **Competencia de otros métodos:** Durante este período, otros enfoques de aprendizaje automático, como las máquinas de vectores de soporte (SVM) y los métodos estadísticos, demostraron ser más efectivos y eficientes en diversas tareas
- **Escasez de datos**

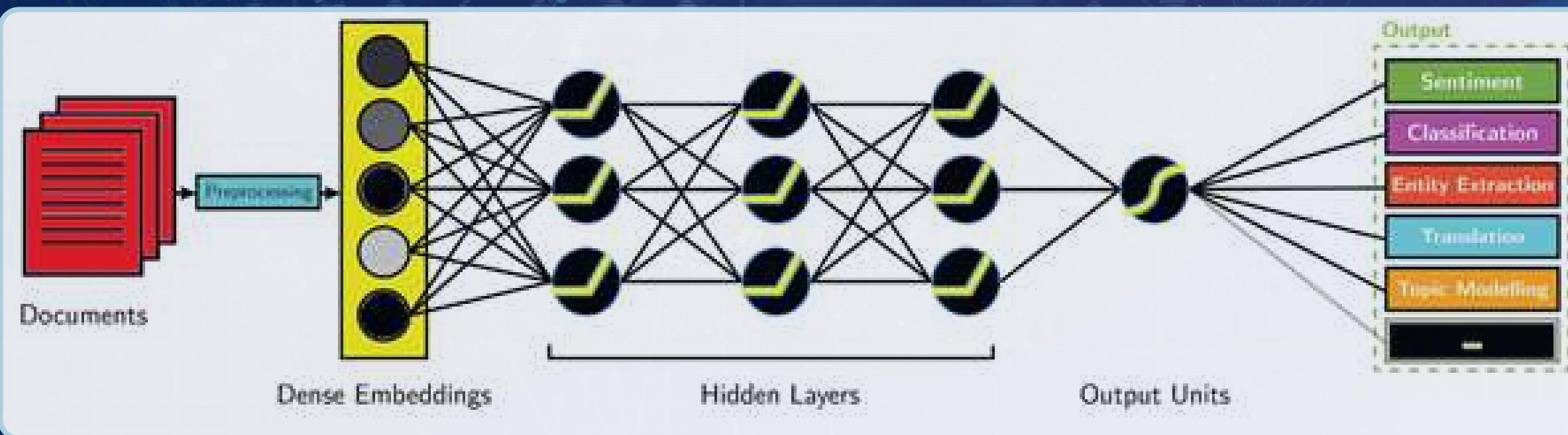


¿Cómo se superó este invierno?

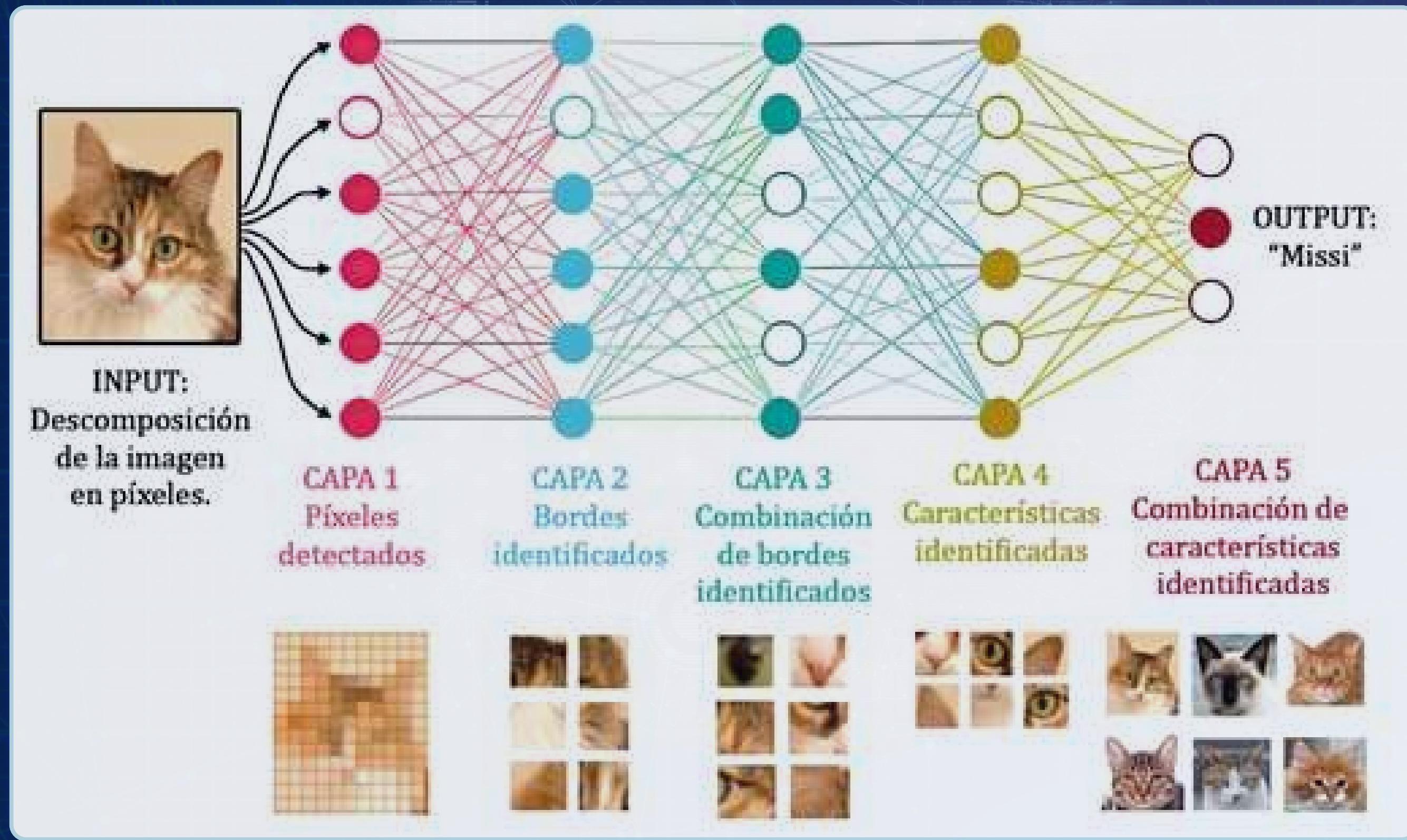
- **Avances en hardware:** El desarrollo de unidades de procesamiento gráfico (GPU)
- **Disponibilidad de grandes conjuntos de datos:** La recopilación y etiquetado de grandes volúmenes de datos
- **Innovaciones en algoritmos:** Se introdujeron nuevas arquitecturas y técnicas, como las redes convolucionales (CNN) y las redes residuales (ResNet), que mitigaron problemas anteriores y mejoraron el rendimiento en tareas específicas.
- **Éxitos prácticos:** Aplicaciones exitosas en reconocimiento de imágenes, procesamiento del lenguaje natural y otros campos demostraron la eficacia de las redes neuronales

De MLP a redes profundas: ¿Por qué más capas?

- Profundidad = más capas ocultas.
- Cada capa aprende una representación más abstracta que la anterior.
- Por ejemplo, las capas jerárquicas en PLN:
 - Capas iniciales – Nivel léxico (Palabras individuales)
 - Capas intermedias – Nivel sintáctico (Relaciones gramaticales)
 - Capas profundas – Nivel semántico (Significado global)
- Las redes profundas procesan texto en niveles: primero entienden las palabras, luego las relaciones, y finalmente el significado completo.
- Esto permite resolver tareas complejas con una arquitectura más eficiente y modular.

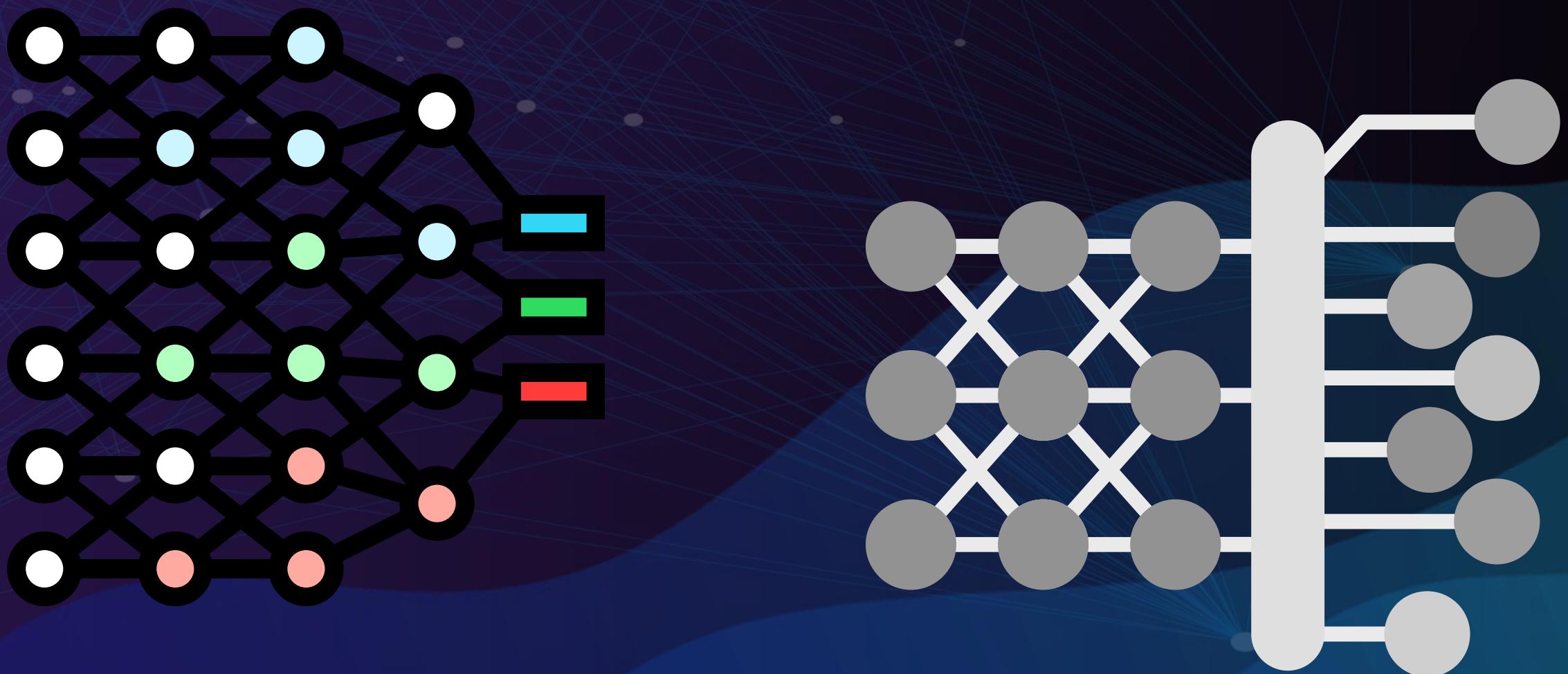


De MLP a redes profundas: ¿Por qué más capas?



Deep Learning

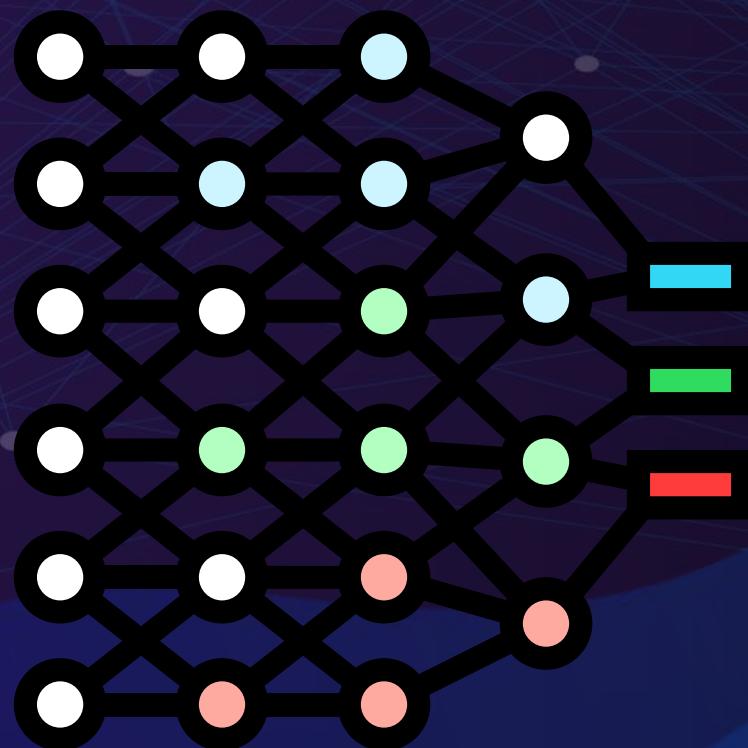
El Deep Learning es una subárea del aprendizaje automático que utiliza redes neuronales con múltiples capas ocultas. Estas redes son capaces de aprender directamente de los datos sin necesidad de diseñar manualmente características o reglas.



Deep Learning

Características principales:

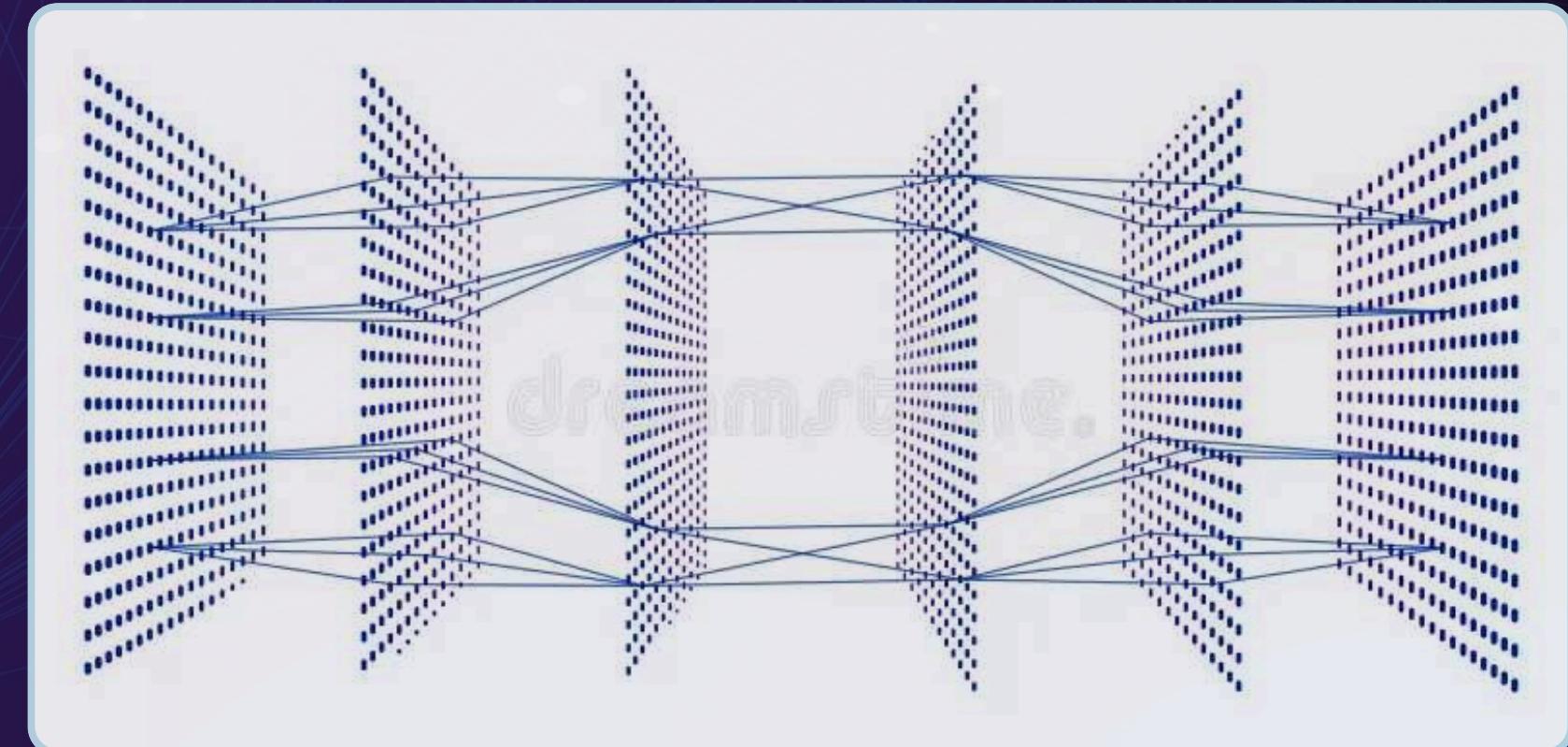
- Composición de múltiples capas no lineales.
- Aprendizaje jerárquico de representaciones: desde lo simple a lo complejo.
- Capacidad de generalización en tareas como clasificación, detección, generación de contenido.
- Requiere una gran cantidad de datos y cómputo (frecuentemente se usan GPUs).



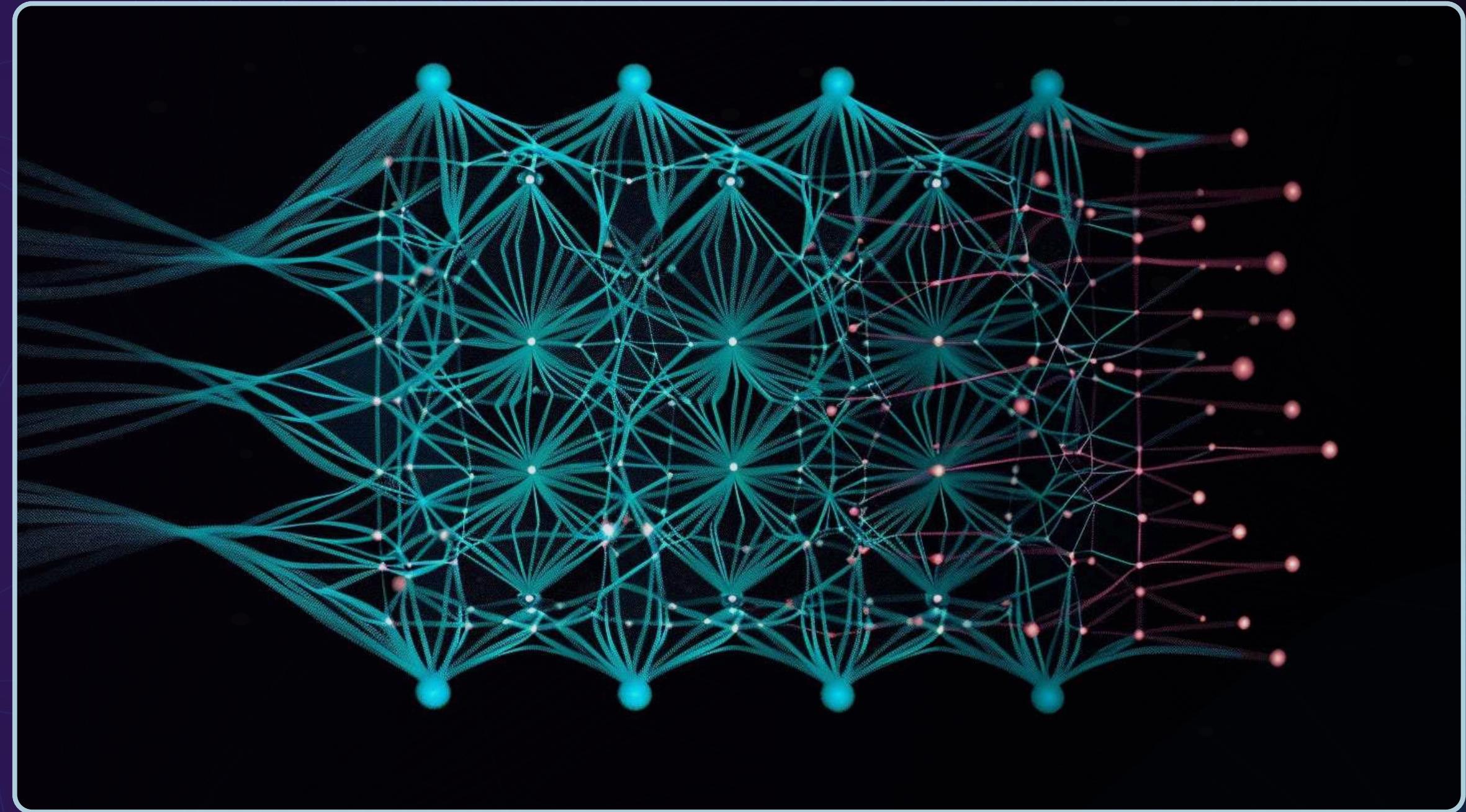
Deep Learning

Las redes neuronales artificiales (RNA) se estructuran en capas de entrada, ocultas y salida, donde cada neurona procesa información mediante funciones matemáticas.

El MLP (Perceptrón Multicapa) clásico muestra limitaciones en tareas de alta dimensionalidad y correlaciones no lineales, debido a su capacidad limitada para abstraer características jerárquicas.



Deep Learning



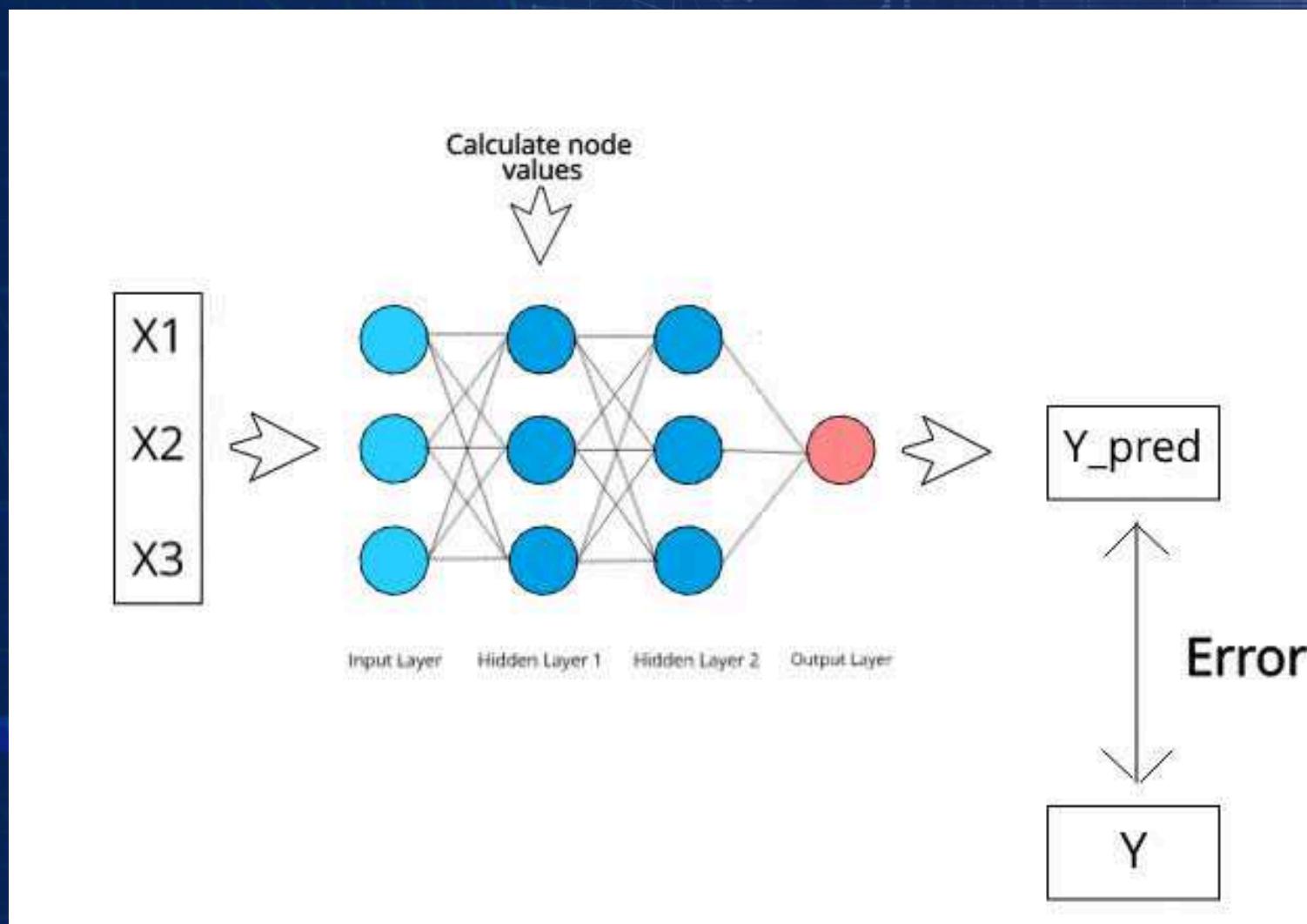
Aumentar la profundidad de la red permite modelar relaciones complejas mediante transformaciones sucesivas.



Ciclo de entrenamiento

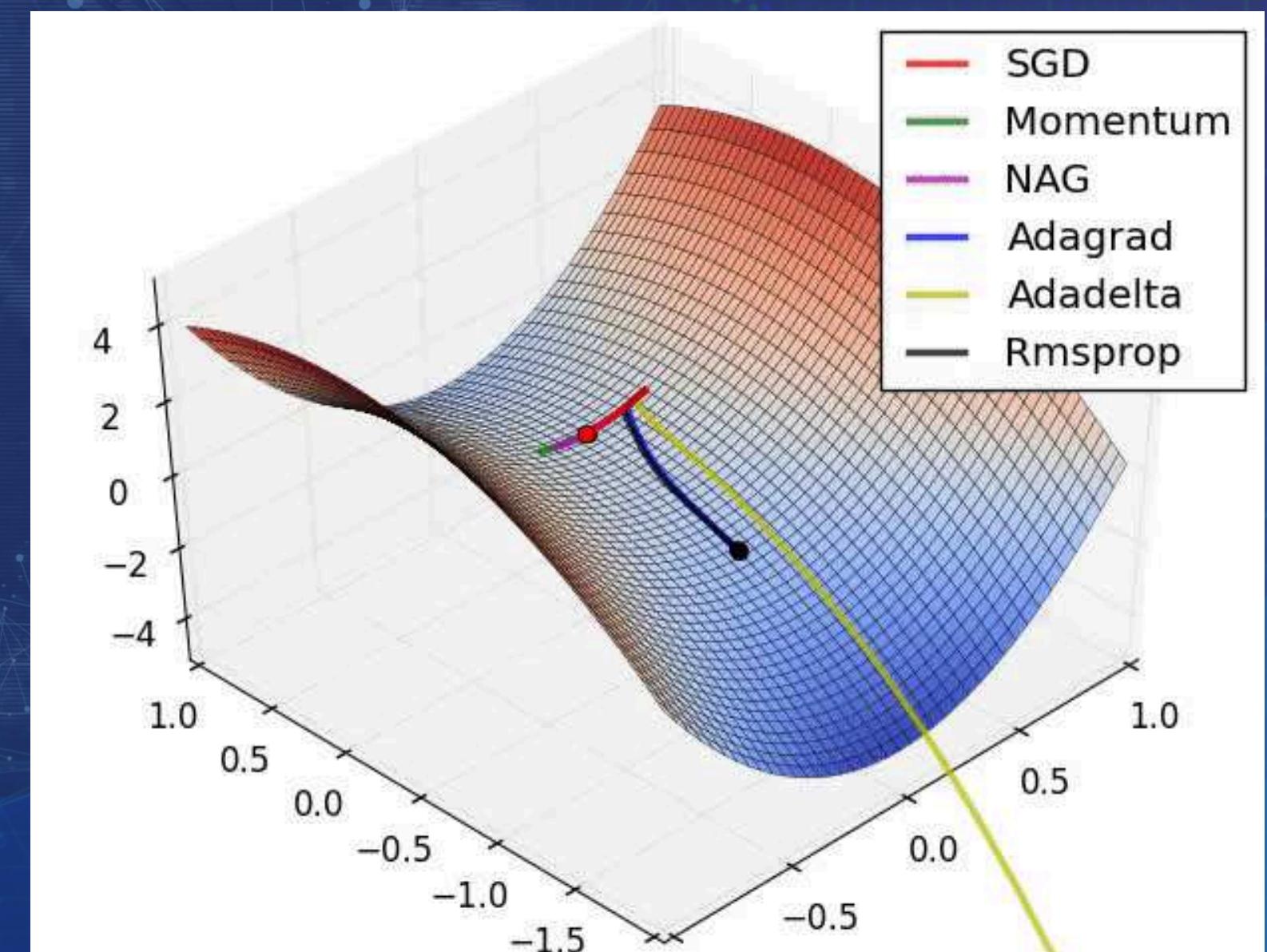
BACKPROPAGATION

El ciclo de entrenamiento es el proceso que permite a una red neuronal aprender a partir de datos etiquetados. Este ciclo se repite una y otra vez durante el entrenamiento, con el objetivo de minimizar el error entre las predicciones de la red y los valores reales.



OPTIMIZACIÓN

Los optimizadores son algoritmos que deciden cómo actualizar los pesos a partir de los gradientes.





¿Qué es una Red Neuronal? Parte 3 : Backpropagation | DotCSV

↗
Share

P A R T E 3 | B A C K P R O P .



Watch on YouTube



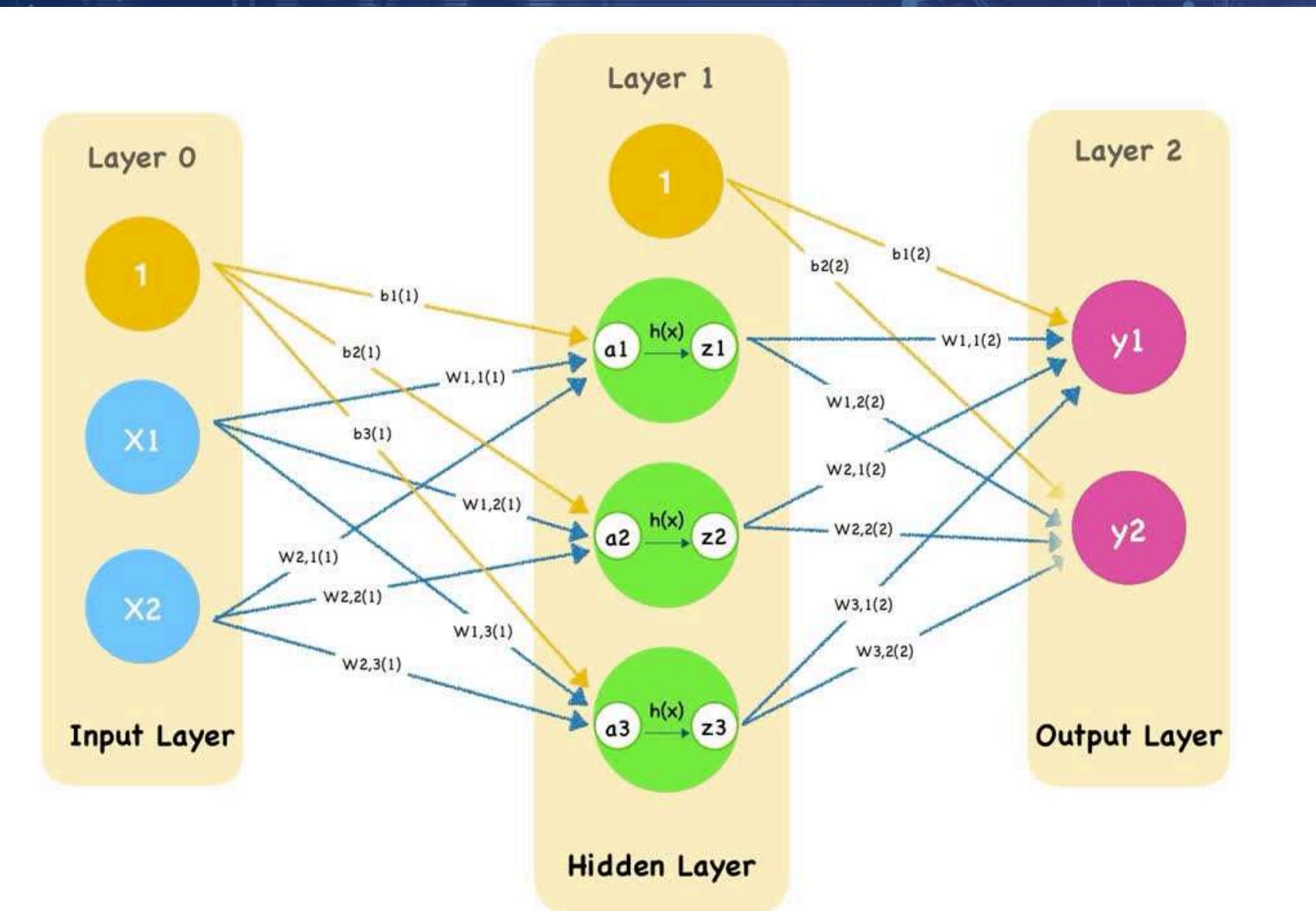
Ciclo de entrenamiento

Paso 1: Hacer una predicción

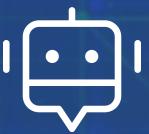
¿Cómo?

(Nombre de la etapa)

FORWARD PROPAGATION



También se conoce de forma coloquial como:
Forward pass o
Feed forward



Ciclo de entrenamiento

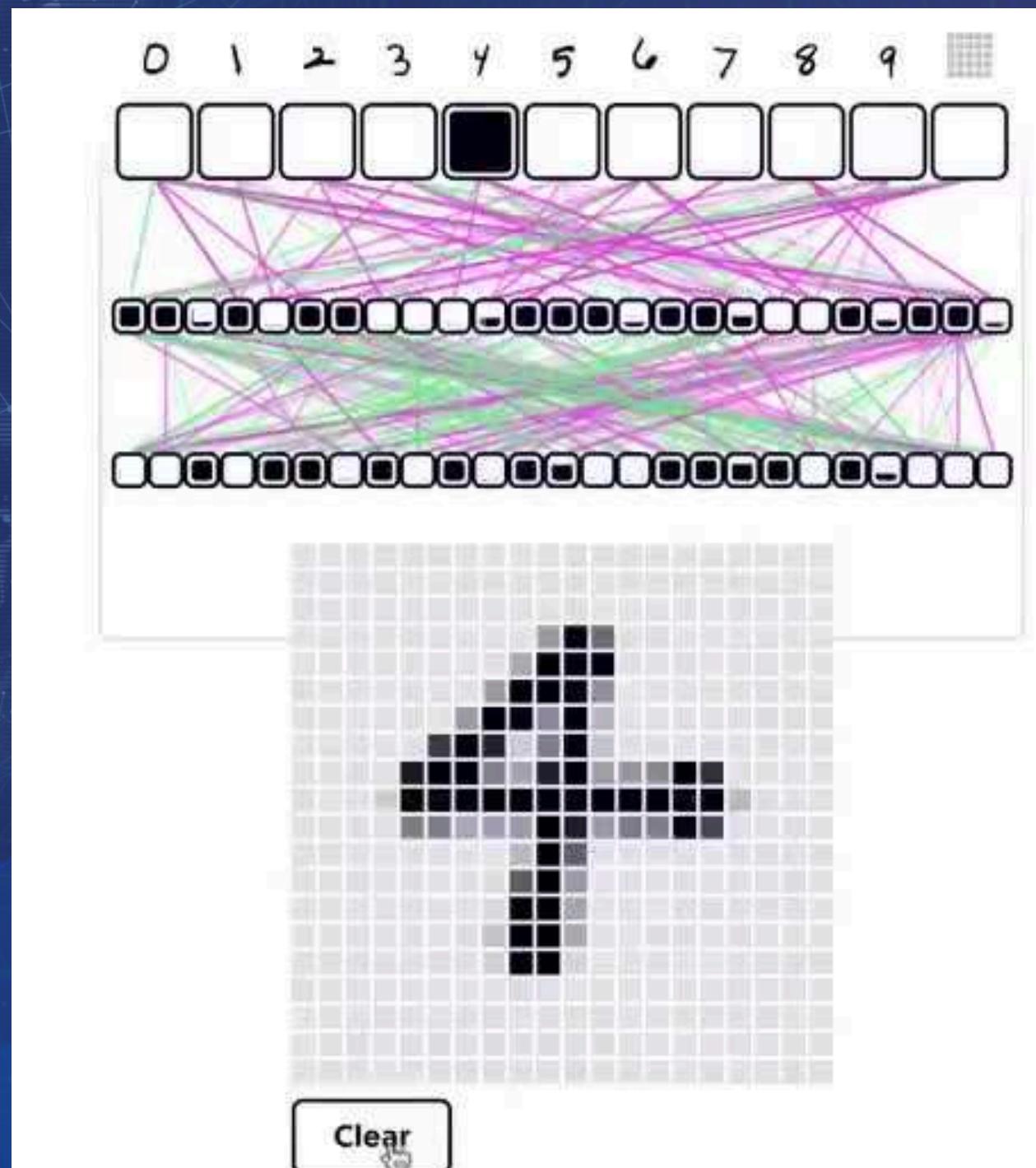
Paso 1: Forward Propagation

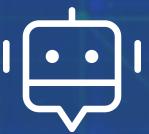
Recorrer los datos desde la entrada hasta la salida de la red neuronal

Ejemplo simple:

Supón que estás entrenando una red para reconocer dígitos escritos a mano (del 0 al 9).

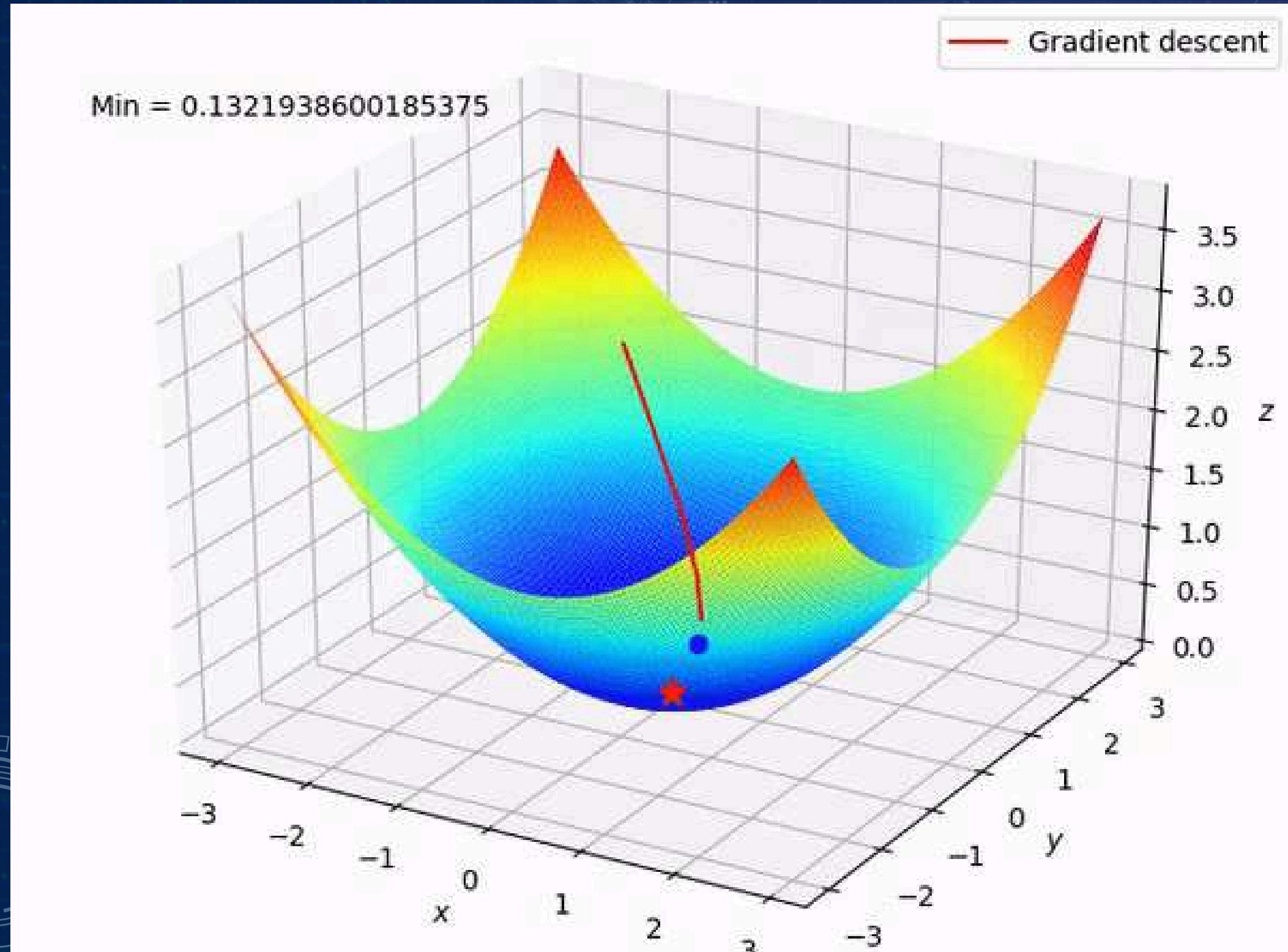
1. Tú le das a la red una imagen
2. La red pasa esa imagen por todas sus capas (aplicando pesos y funciones de activación)
3. Al final, te da una salida como $[0.05, 0.10, 0.8, \dots, 0.01]$ → esto significa que probablemente sea un "2".





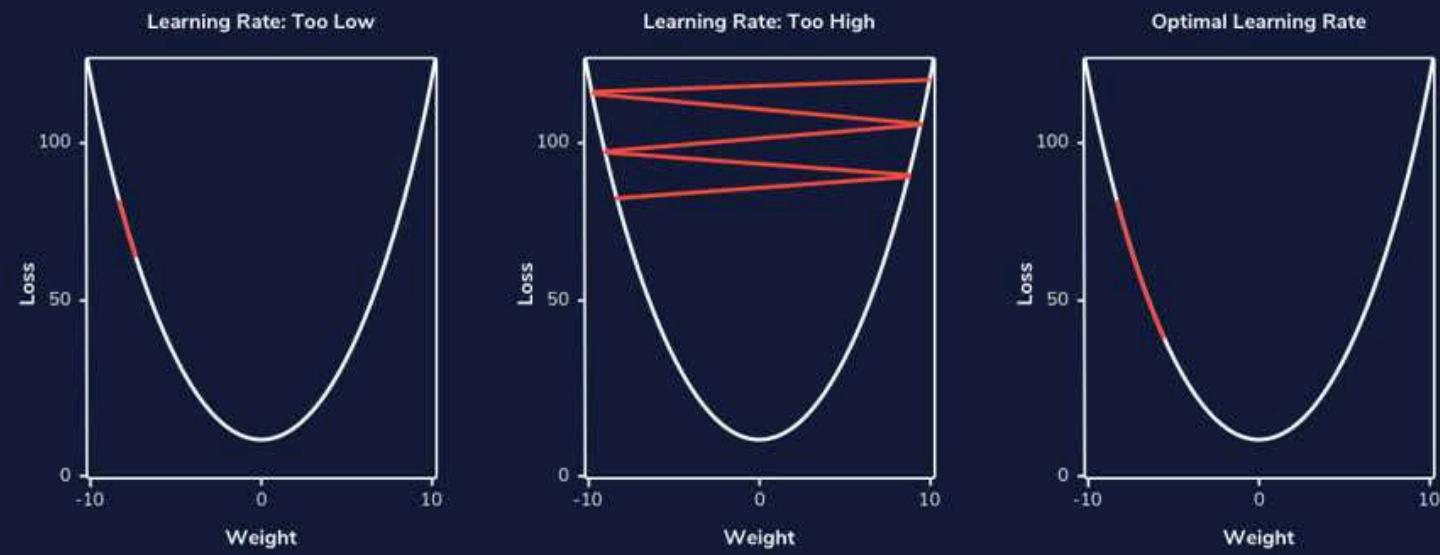
Ciclo de entrenamiento

Paso 2: Cálculo del Error



Ejemplo simple:

- Si la red dijo: “Creo que es un 7” pero en realidad es un 5, entonces la función de pérdida nos dice cuánto se equivocó la red.



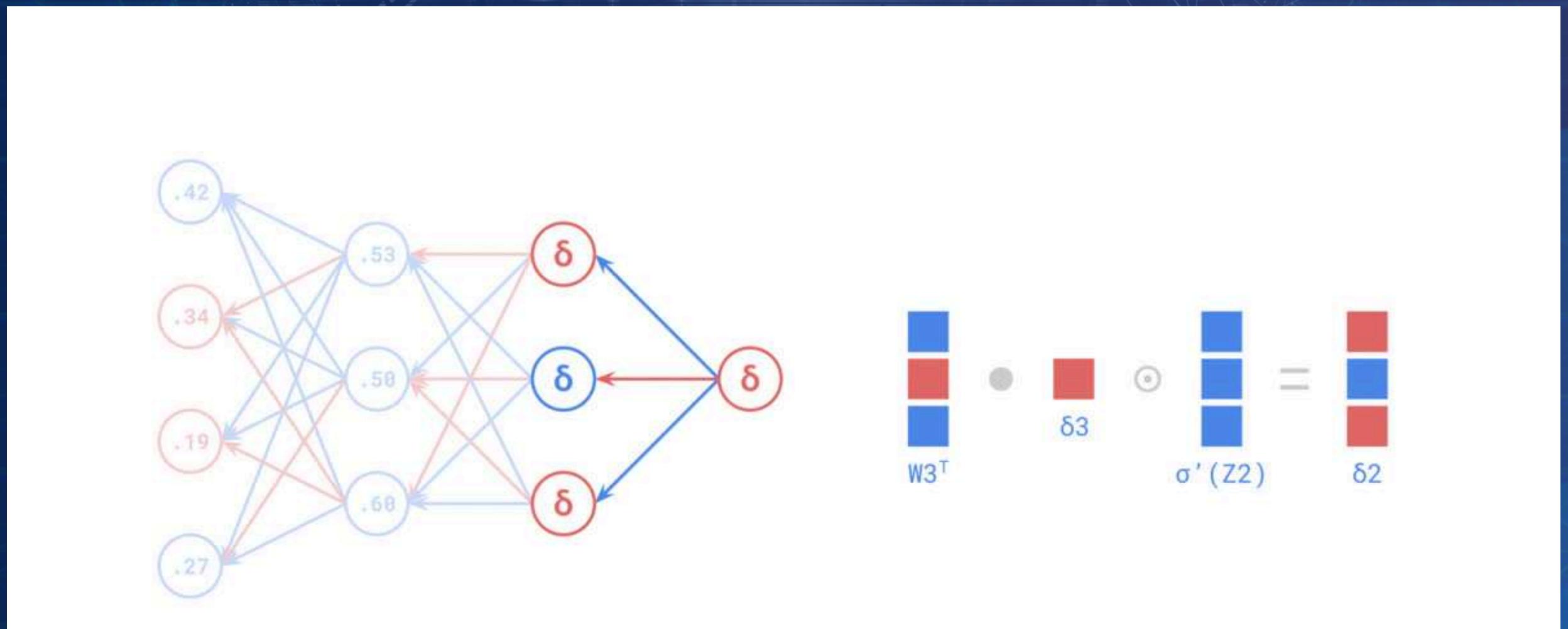


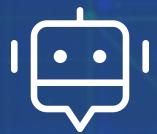
Ciclo de entrenamiento

Paso 3: Aprendiendo de los errores

Backpropagation usa el cálculo diferencial (más específicamente, el gradiente descendente) para calcular cómo afecta cada peso al error total.

1. Calculamos cuánto contribuyó cada peso y sesgo al error.
2. Esto se hace calculando la derivada parcial del loss respecto a cada parámetro .
3. Estas derivadas nos indican hacia dónde debemos mover los pesos para reducir el error.





Ciclo de entrenamiento

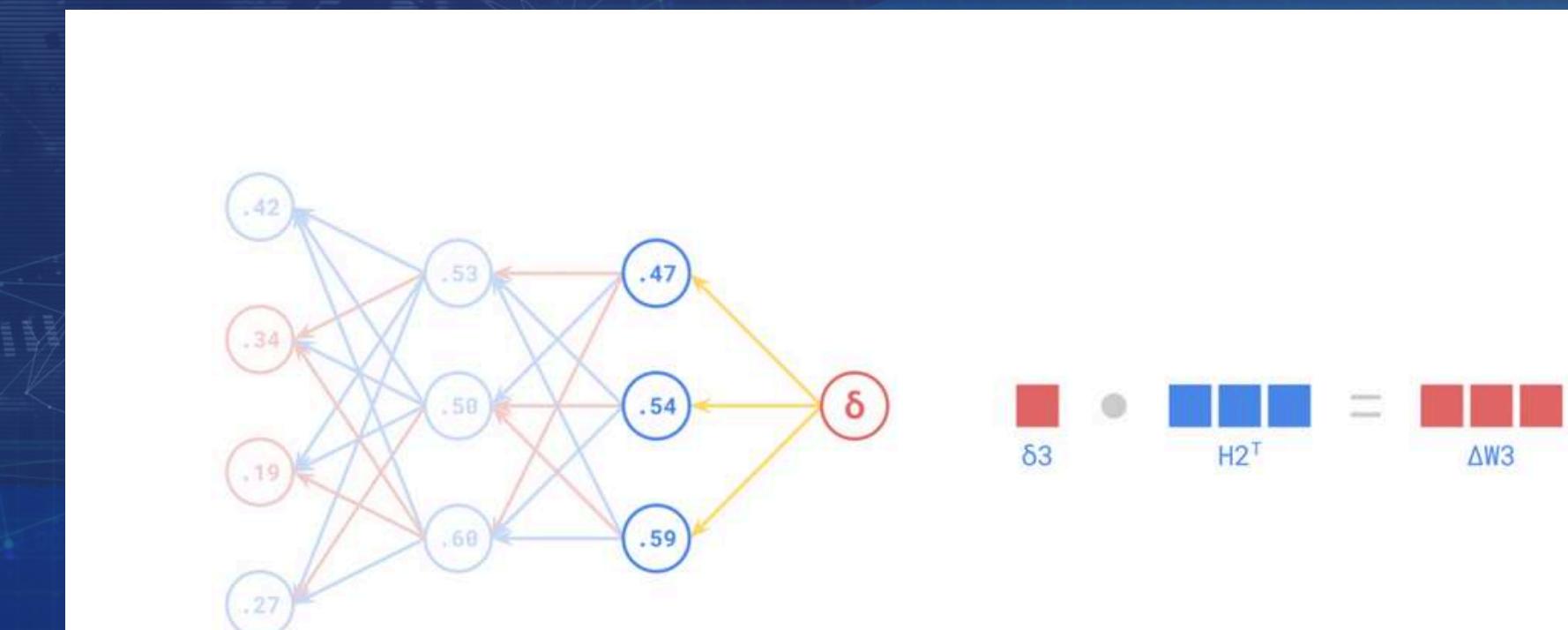
Paso 4: Ajuste de los parámetros

Con los gradientes calculados, ahora actualizamos los pesos y sesgos de la red.

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \cdot \delta_j^{(l)} \cdot a_i^{(l-1)}$$
$$b_j^{(l)} \leftarrow b_j^{(l)} - \eta \cdot \delta_j^{(l)}$$

Donde:

- $w_{ji}^{(l)}$: peso que conecta la neurona i de la capa $l-1$ con la neurona j de la capa l .
- $b_j^{(l)}$: sesgo de la neurona j en la capa l .
- η : tasa de aprendizaje (learning rate).
- $\delta_j^{(l)}$: error delta de la neurona j en la capa l .
- $a_i^{(l-1)}$: salida de la neurona i en la capa $l-1$.

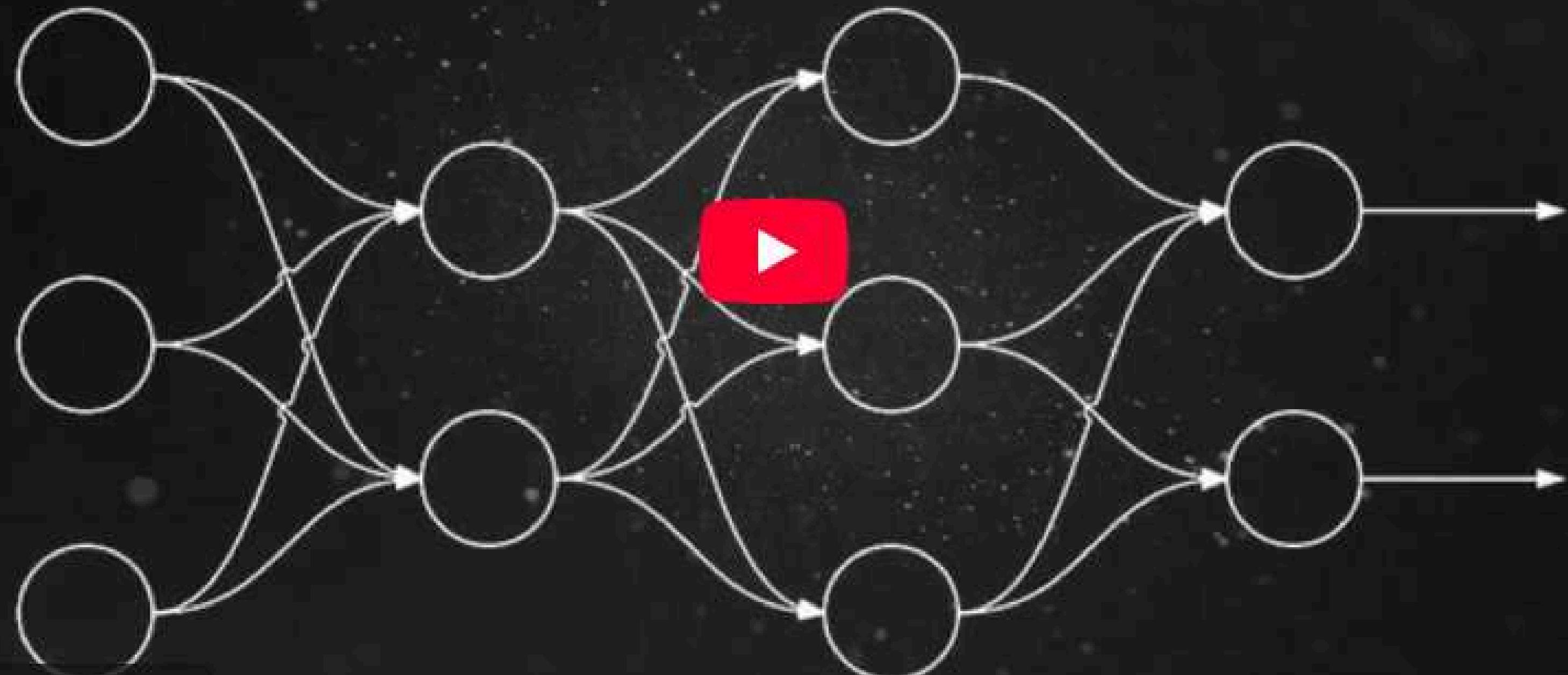




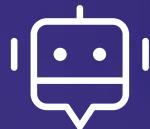
¿Qué es una Red Neuronal? Parte 3.5 : Las Matemáticas de Backpropagation | DotCSV

Share

PARTE 3.5 | FÓRMULAS



Watch on YouTube



Arquitecturas de Deep Learning

- Redes feedforward profundas (DNN)
- Arquitecturas recurrentes
- Arquitecturas convolucionales
- Redes generativas

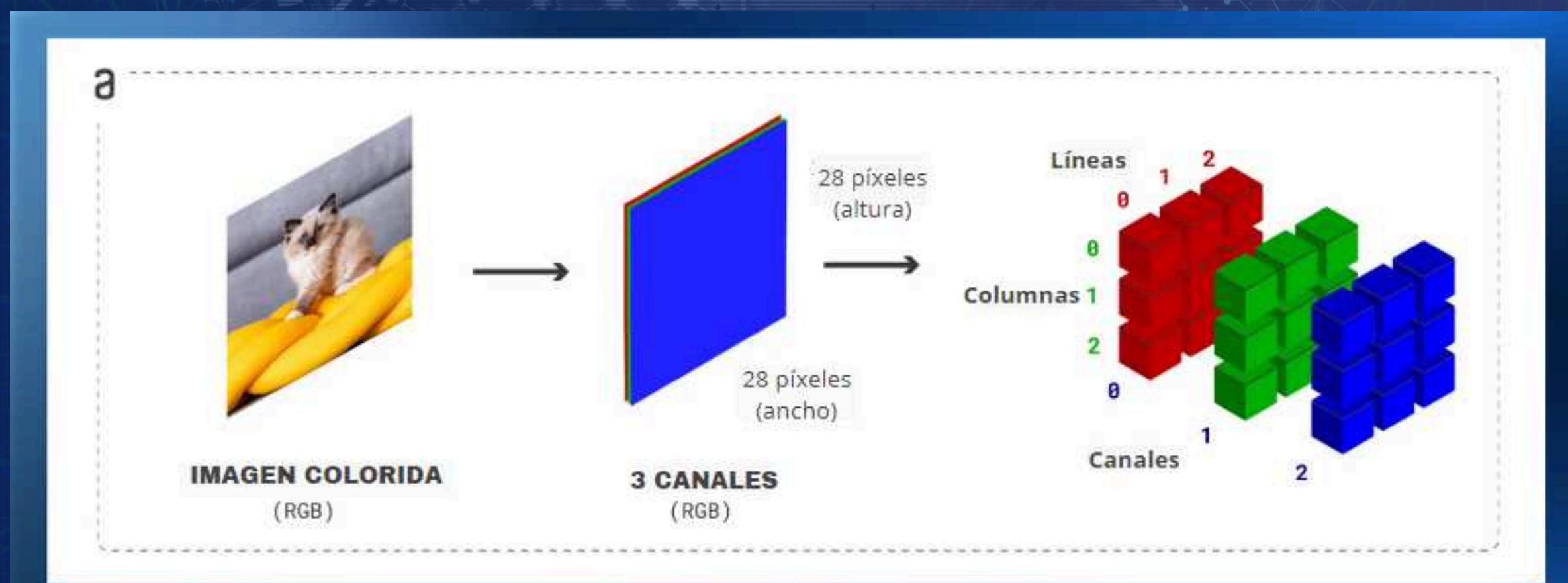


CNN

Las imágenes tienen estructura espacial:

- Anchura
- Altura
- Canales

Para una imagen 28x28 píxeles, una capa totalmente conectada tendría 784 entradas por cada neurona.

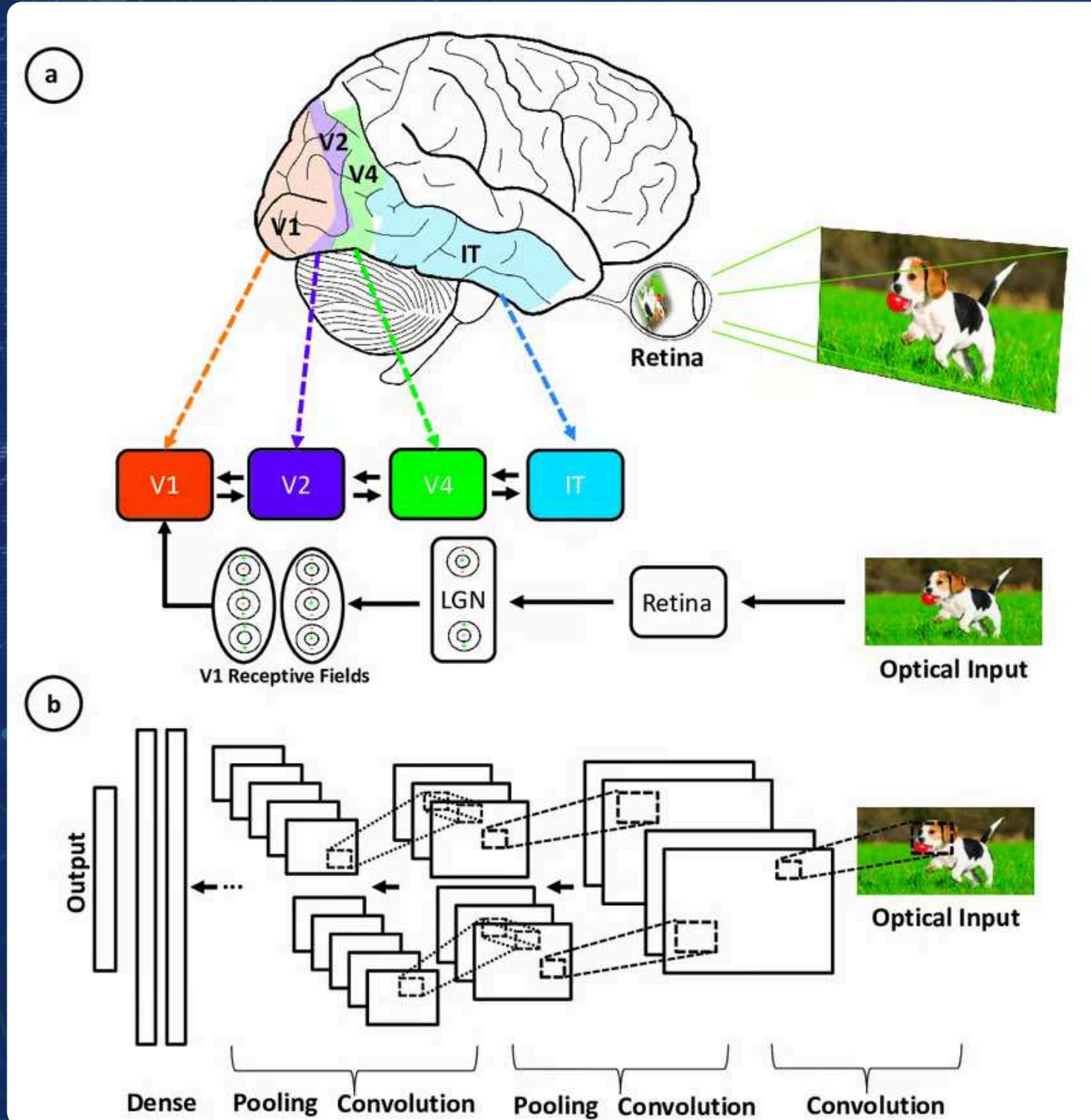




CNN

Las redes neuronales convolucionales (CNN, por sus siglas en inglés) son un tipo especializado de red neuronal artificial diseñada para procesar y analizar datos visuales, como imágenes y videos.

Inspiración biológica:
células en la corteza visual
del cerebro que responden
a regiones específicas del
campo visual.



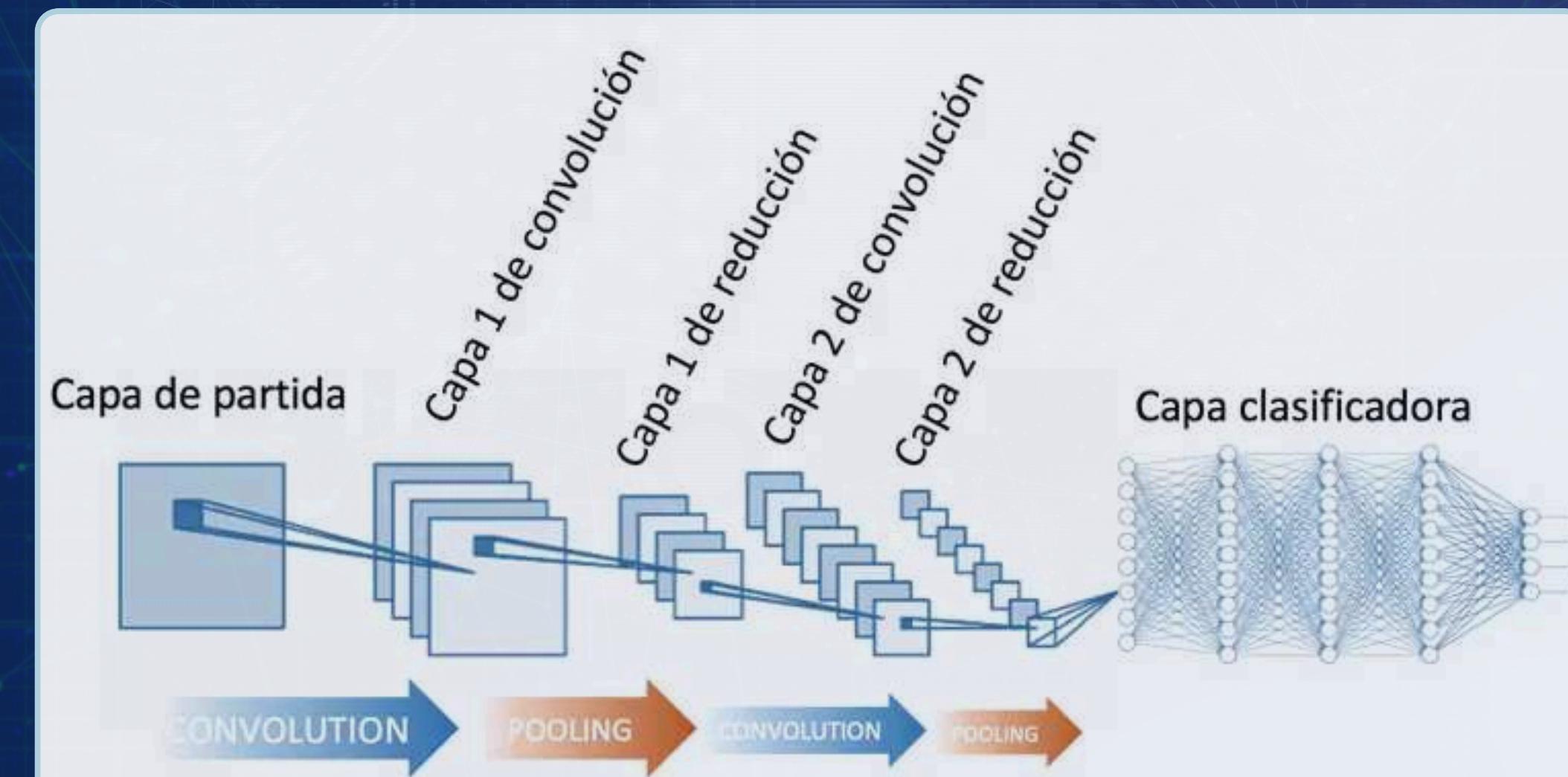
Las CNN procesan imágenes imitando el funcionamiento del ojo humano. Cada neurona de la red trabaja sobre una pequeña región de la imagen (campo receptivo), y a través de varias capas, la red aprende a identificar desde patrones simples (bordes, líneas) hasta patrones complejos (formas, objetos)



CNN

El flujo general es:

- Entrada: Imagen en formato de matriz de píxeles.
- Capas convolucionales: Aplican filtros (kernels) para extraer características locales.
- Capas de activación: Usualmente la función ReLU, que introduce no linealidad.
- Capas de agrupamiento (pooling): Reducen la dimensionalidad y retienen las características más importantes.
- Capas totalmente conectadas: Realizan la clasificación final basándose en las características extraídas

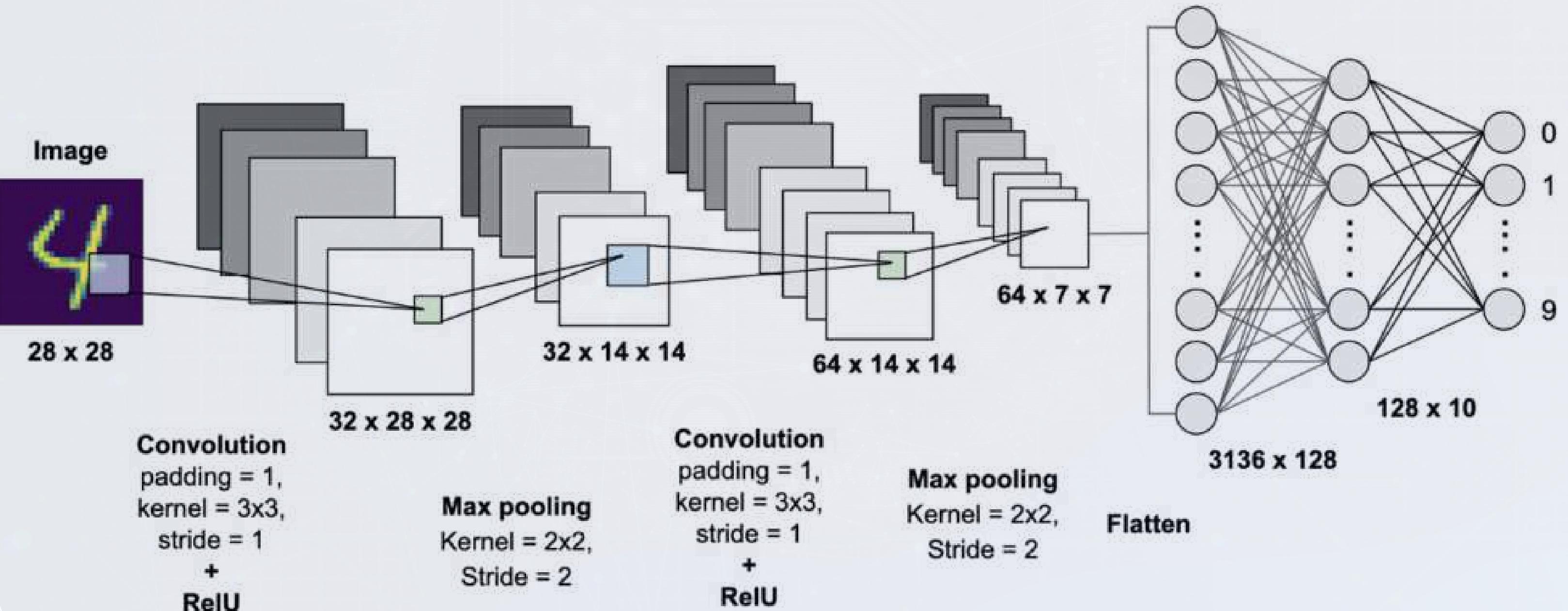


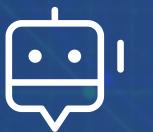


CNN



Keras

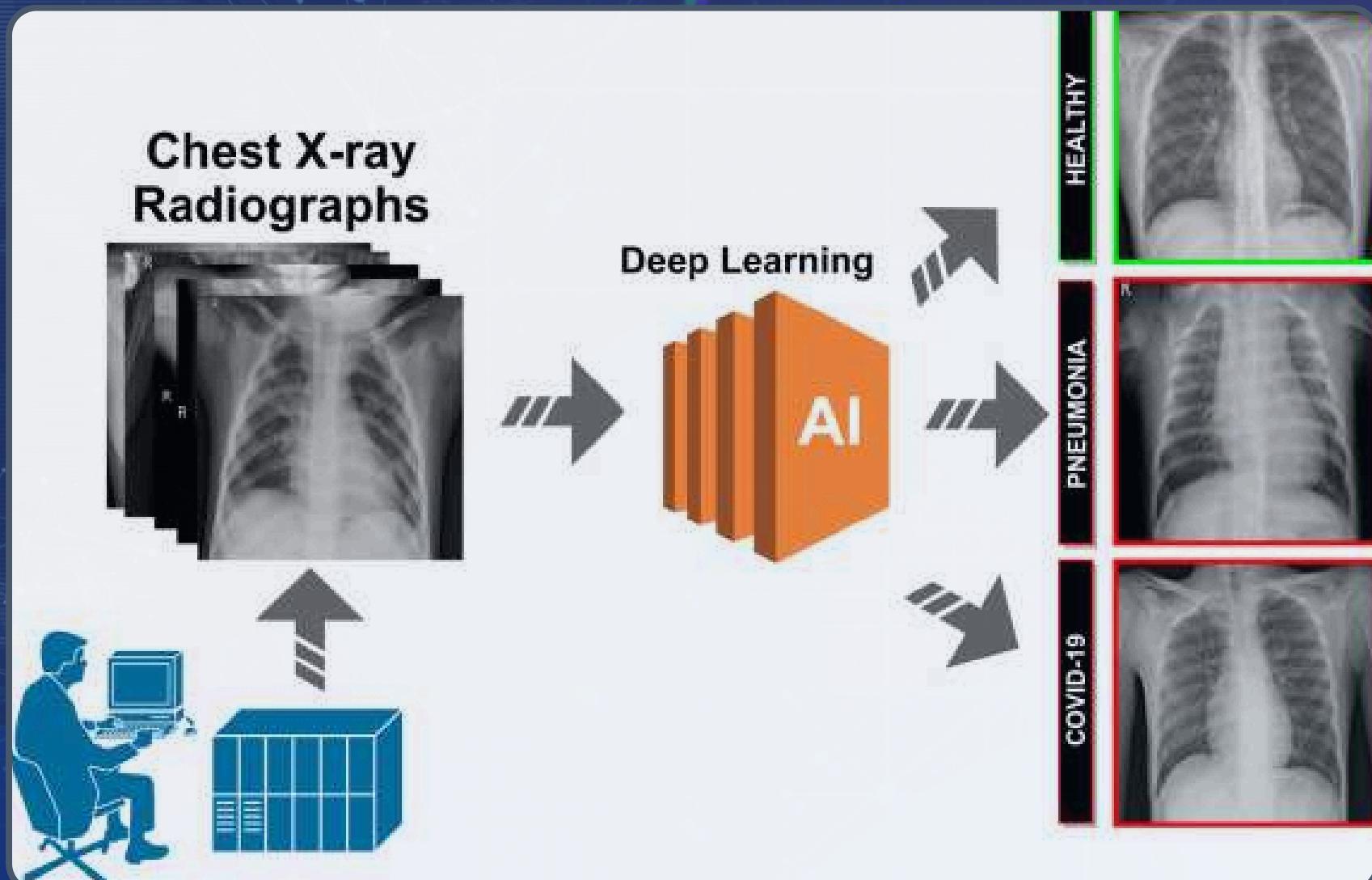
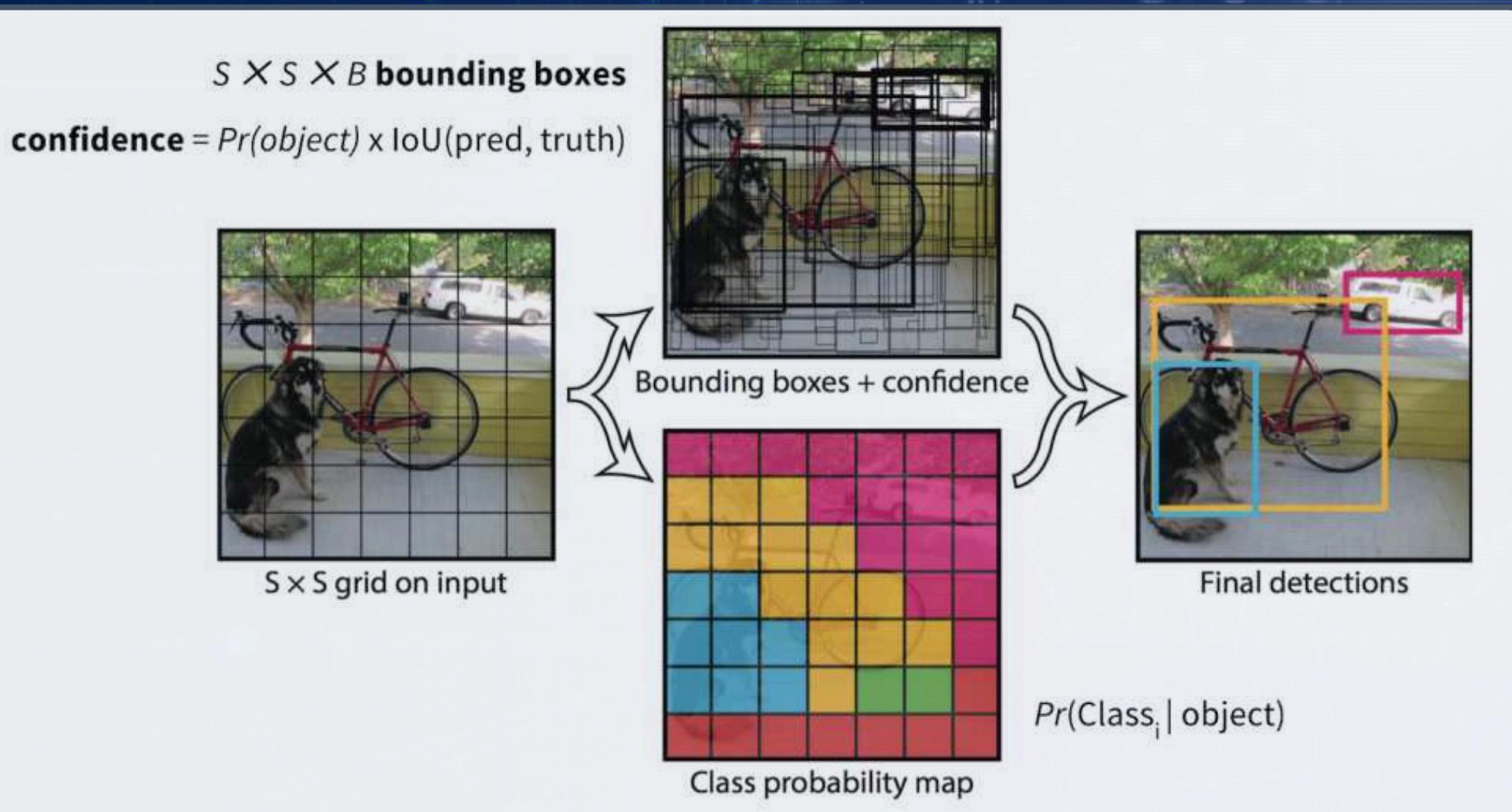




CNN

Aplicaciones:

- Clasificación de imágenes
- Detección de objetos
- Reconocimiento facial
- Reconocimiento de escritura manuscrita
- Diagnóstico médico (detección de tumores, COVID en rayos X)





¡Redes Neuronales CONVOLUCIONALES! ¿Cómo funcionan?

REDES NEURONALES CONVOLUCIONALES

Share



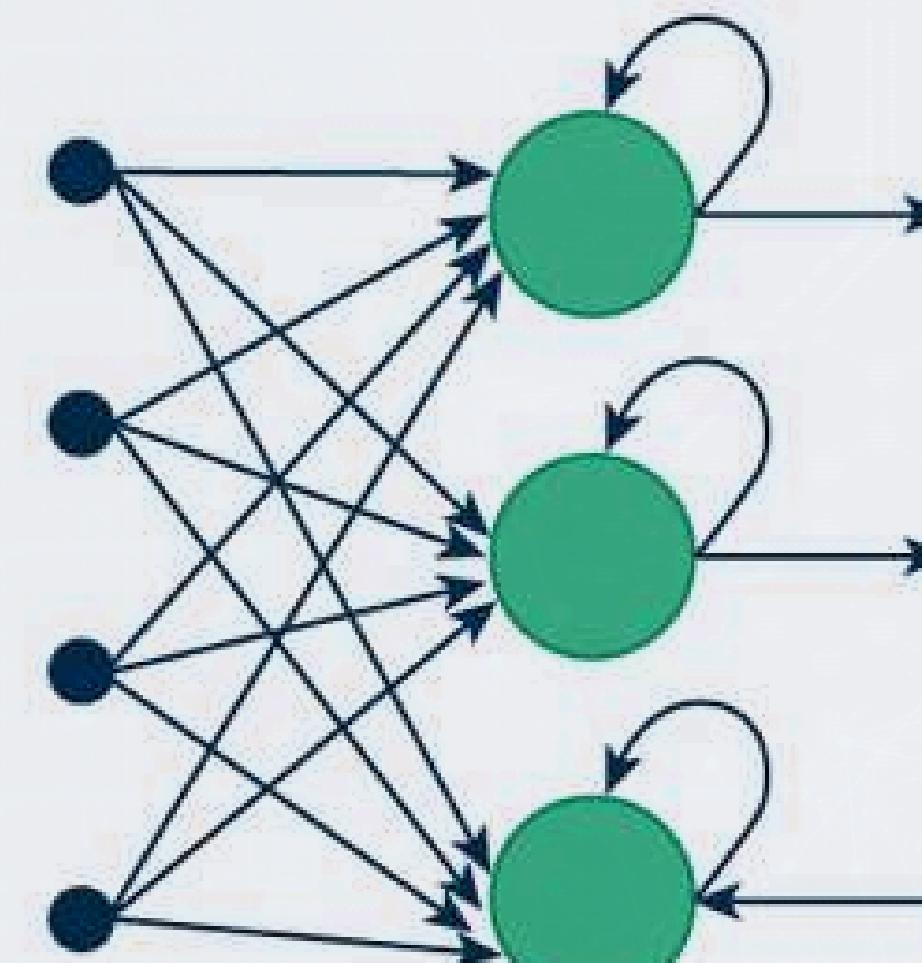
Watch on



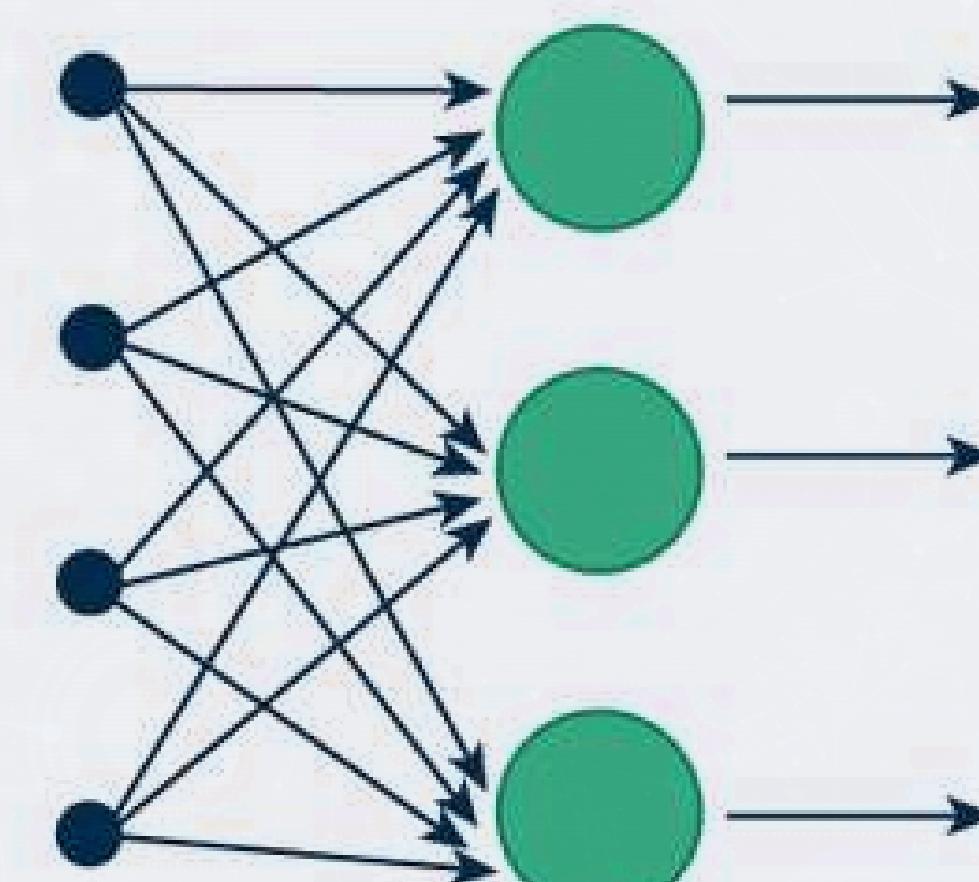


Arquitecturas Recurrentes

Es un tipo de red neuronal diseñada específicamente para procesar datos secuenciales, es decir, aquellos en los que el orden importa, como texto, audio, series temporales, etc.



(a) Recurrent Neural Network



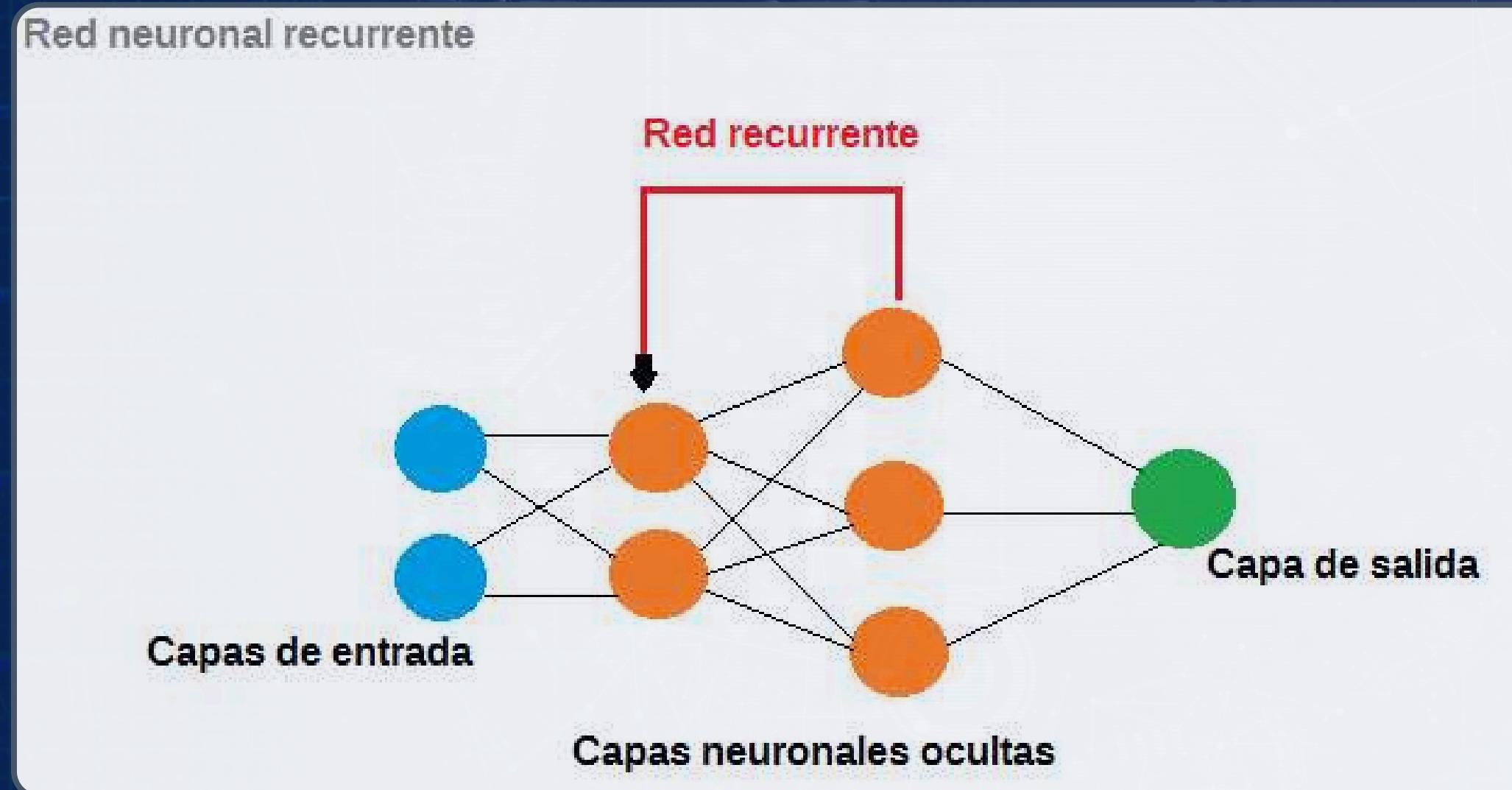
(b) Feed-Forward Neural Network

A diferencia de las redes neuronales tradicionales, las RNN pueden recordar información de entradas anteriores gracias a su arquitectura con conexiones recurrentes, lo que les permite mantener un estado interno o "memoria" a lo largo de la secuencia



Arquitecturas Recurrentes

Funcionamiento

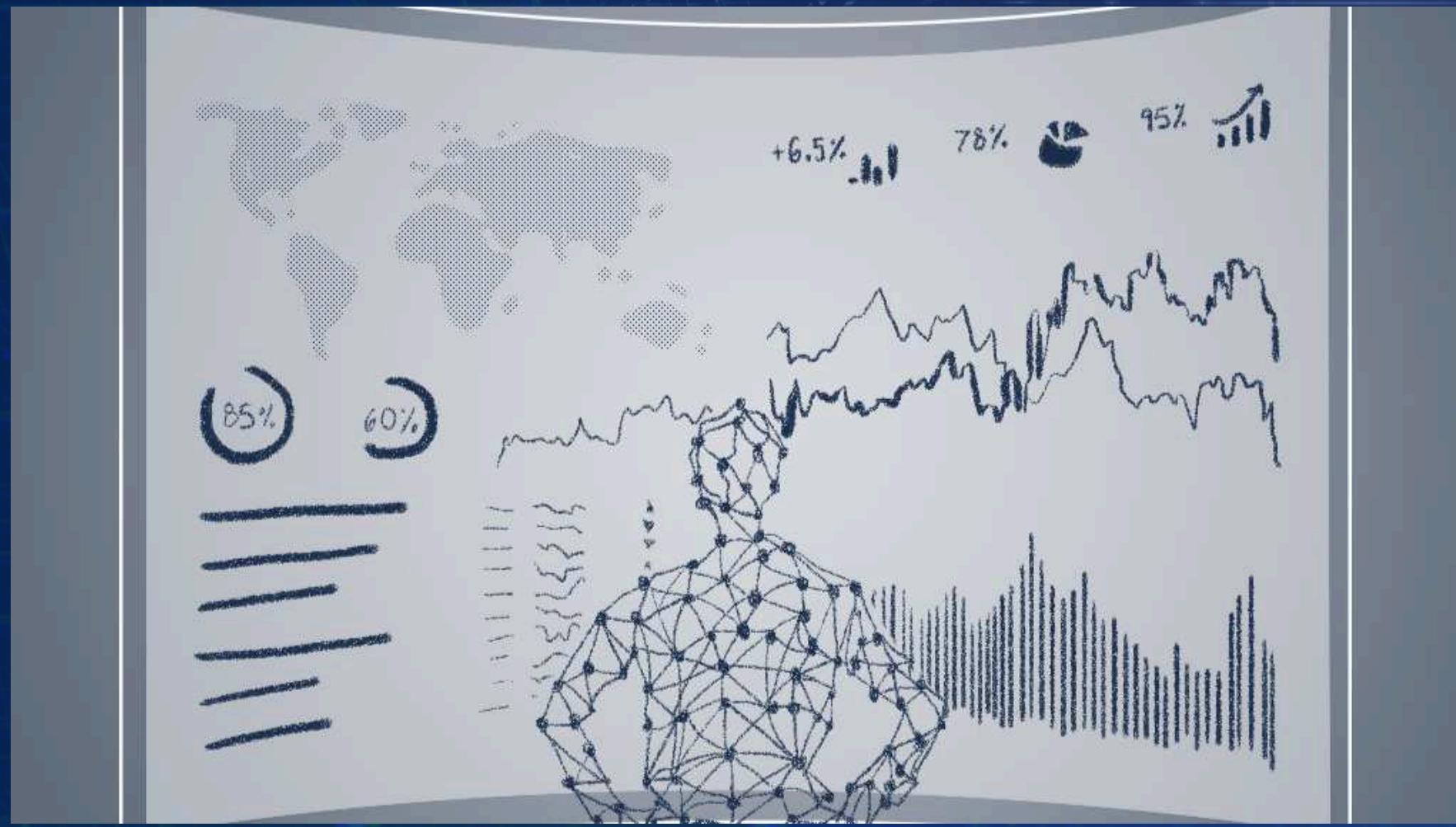


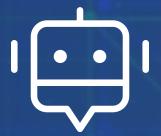
1. La red recibe el primer elemento de la secuencia y lo procesa.
2. El resultado se almacena en el estado interno y se utiliza junto con el siguiente elemento de la secuencia.
3. Este proceso se repite para cada elemento, permitiendo que la red "recuerde" información relevante de pasos anteriores

Arquitecturas Recurrentes

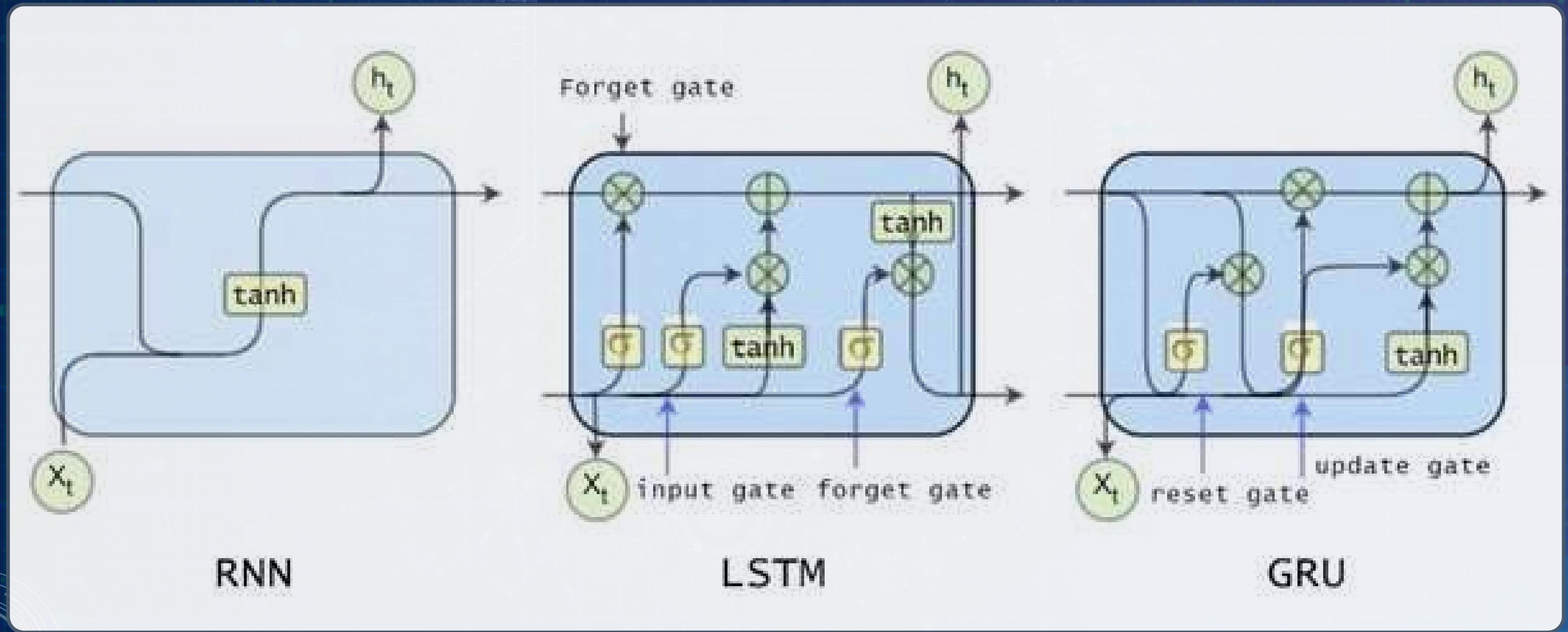
Su memoria de secuencias previas permite aplicarse muy bien en:

1. Procesamiento de lenguaje natural (traducción, generación de texto, análisis de sentimiento)
2. Reconocimiento de voz y síntesis de audio
3. Predicción de series temporales (finanzas, clima)
4. Generación de música y secuencias





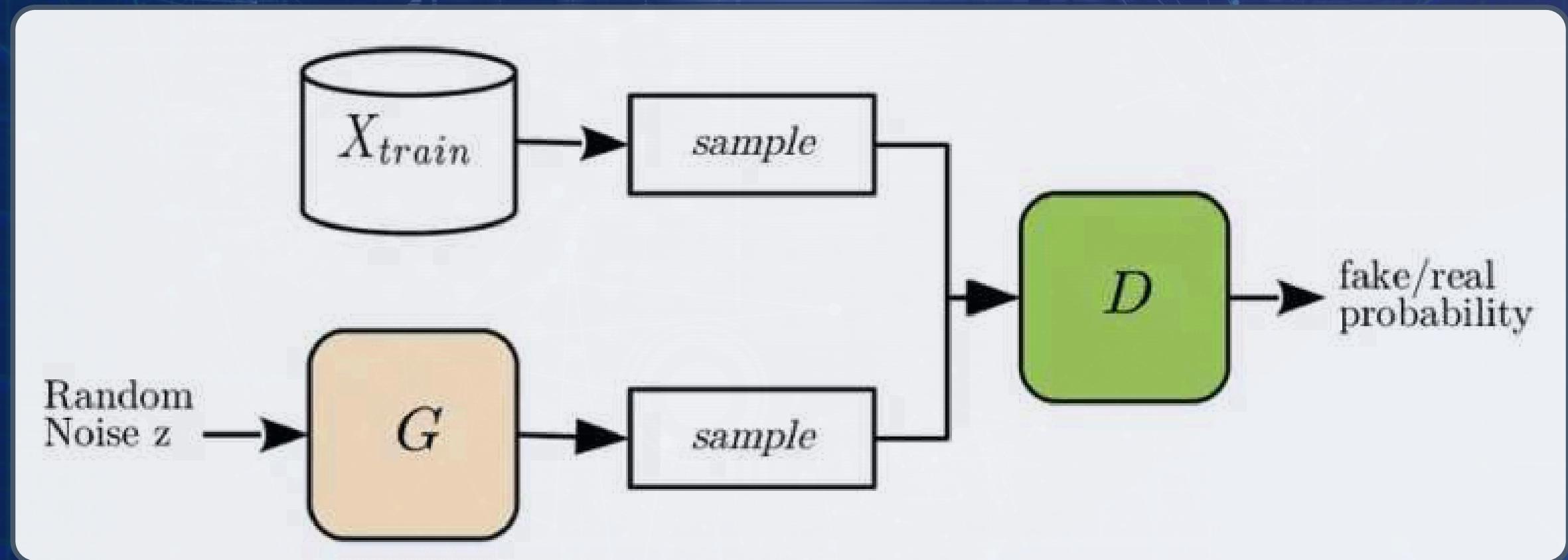
Arquitecturas Recurrentes





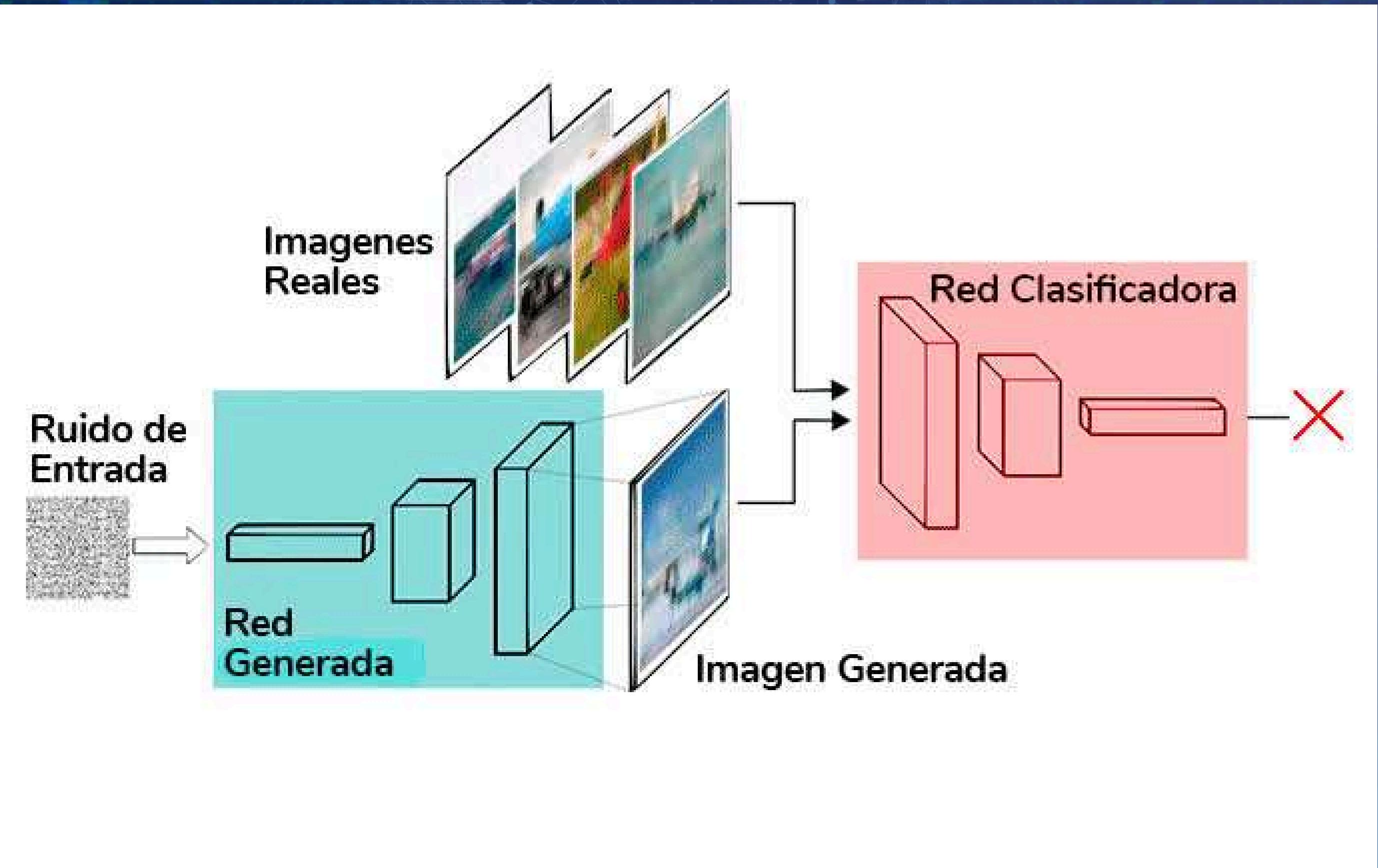
Redes Generativas

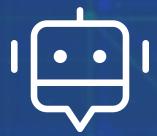
- ¿Qué significa “generar” datos? Crear imágenes, texto, audio o video similares pero nuevos a los del conjunto de entrenamiento.
- Son modelos capaces de aprender la distribución de probabilidad de un conjunto de datos y generar nuevos ejemplos que parezcan provenir de esa distribución.





Redes Generativas





Redes Generativas

- Ejercicio Sugerido: <https://anderfernandez.com/blog/como-crear-una-red-generativa-antagonica-gan-en-python/>

