

Universidad
Rafael Landívar

Redes Neuronales **Artificiales**

Inteligencia Artificial





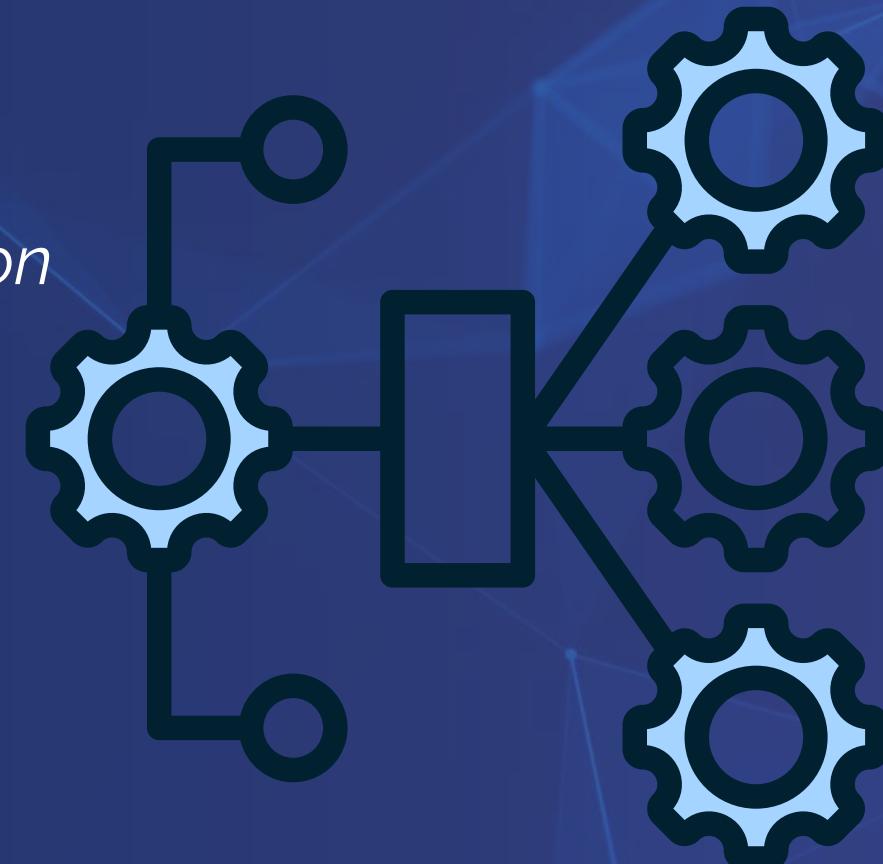
Agenda

Sesión 1: Perceptrón

Sesión 2: Perceptrón Multiclasificación y Multicapa

- Perceptrón Multiclasificación
- Perceptrón Multicapa:
 - Topología
 - Algoritmo: *Forward Propagation*
- Funciones de Activación
- Ejemplos y Ejercicios

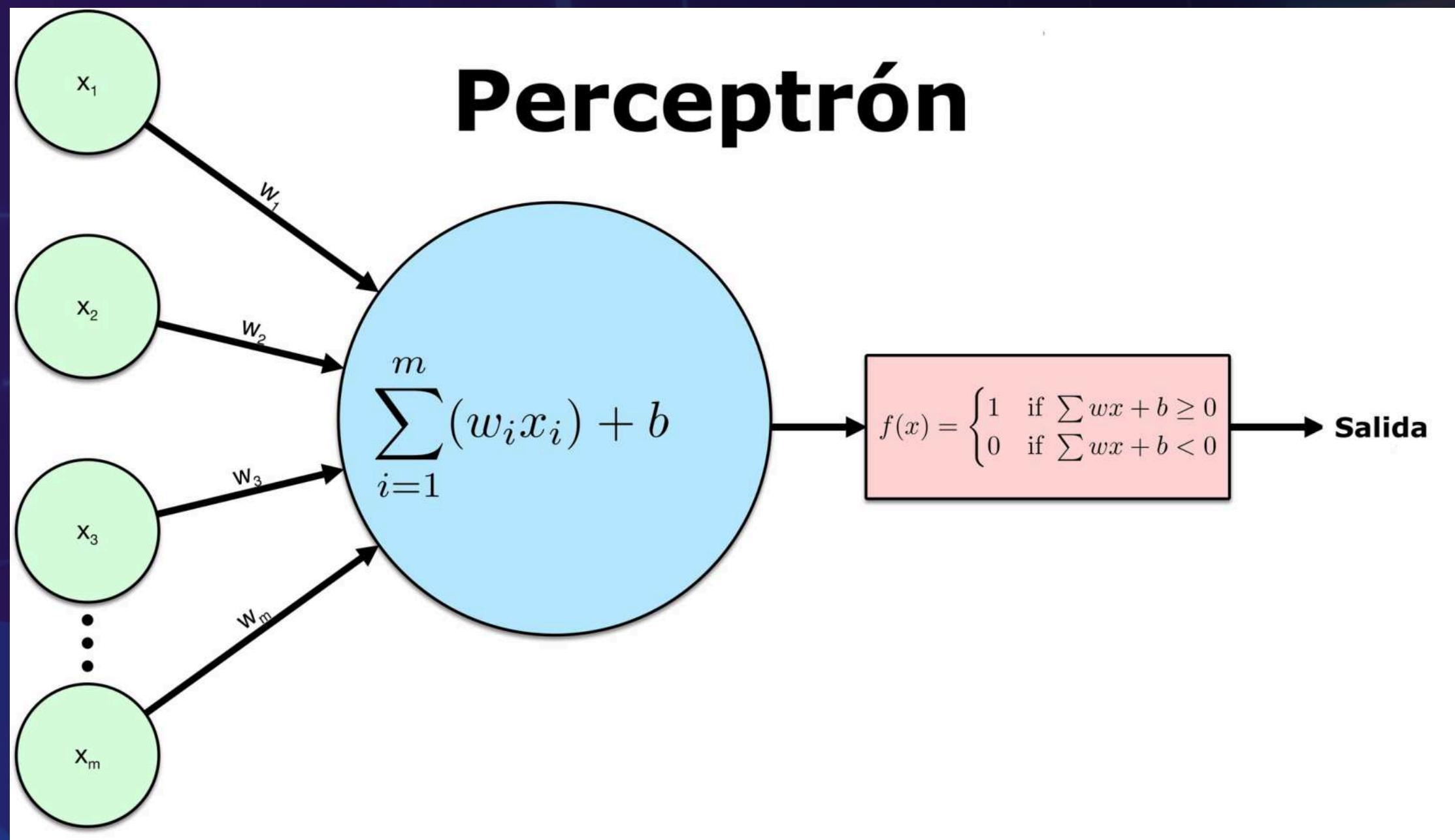
Sesión 3: Redes Neuronales





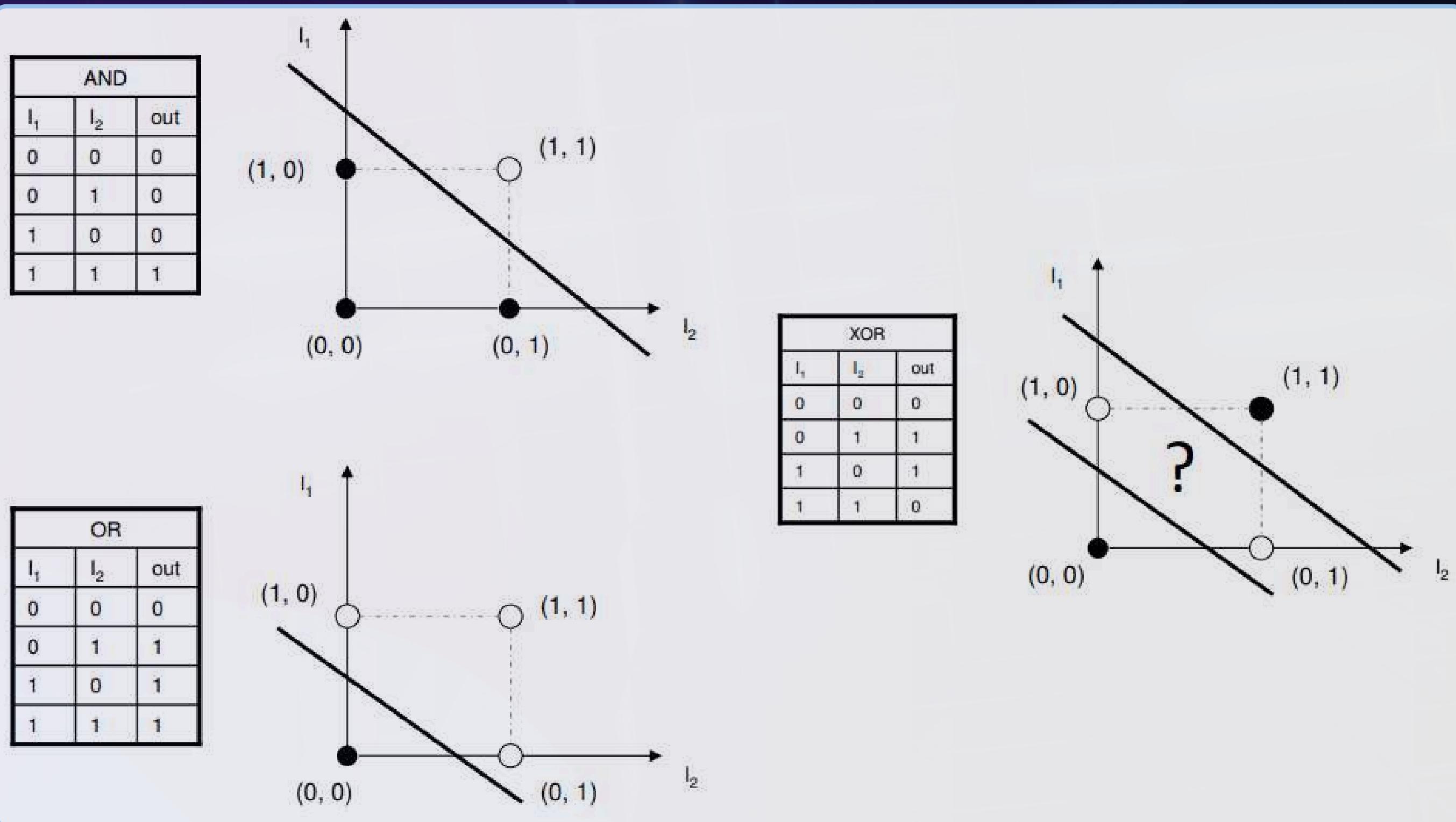
Perceptron

Perceptrón Simple





Perceptrón Simple



Limitaciones del Perceptrón Simple

El Perceptrón Simple es un modelo fundamental en el estudio de las redes neuronales, pero tiene varias limitaciones importantes, especialmente cuando se compara con modelos más avanzados...

1. Clasificador binario por naturaleza

- El perceptrón simple fue diseñado originalmente para clasificación binaria (dos clases).
- Para hacerlo multiclase hay que usar estrategias como One-vs-Rest (OvR) , lo cual no está integrado directamente en su diseño básico.

2. Solo resuelve problemas linealmente separables

- El perceptrón simple solo puede aprender fronteras de decisión lineales
- Esto significa que no puede resolver problemas no lineales

3. Función de Activación Limitada

- Usa la función escalón como función de activación
- La función no es derivable

¿Cómo clasificar más de dos clases?

Los perceptrones clásicos son binarios, pero podemos extenderlos para problemas con más de dos clases usando estrategias que combinan múltiples clasificadores binarios.





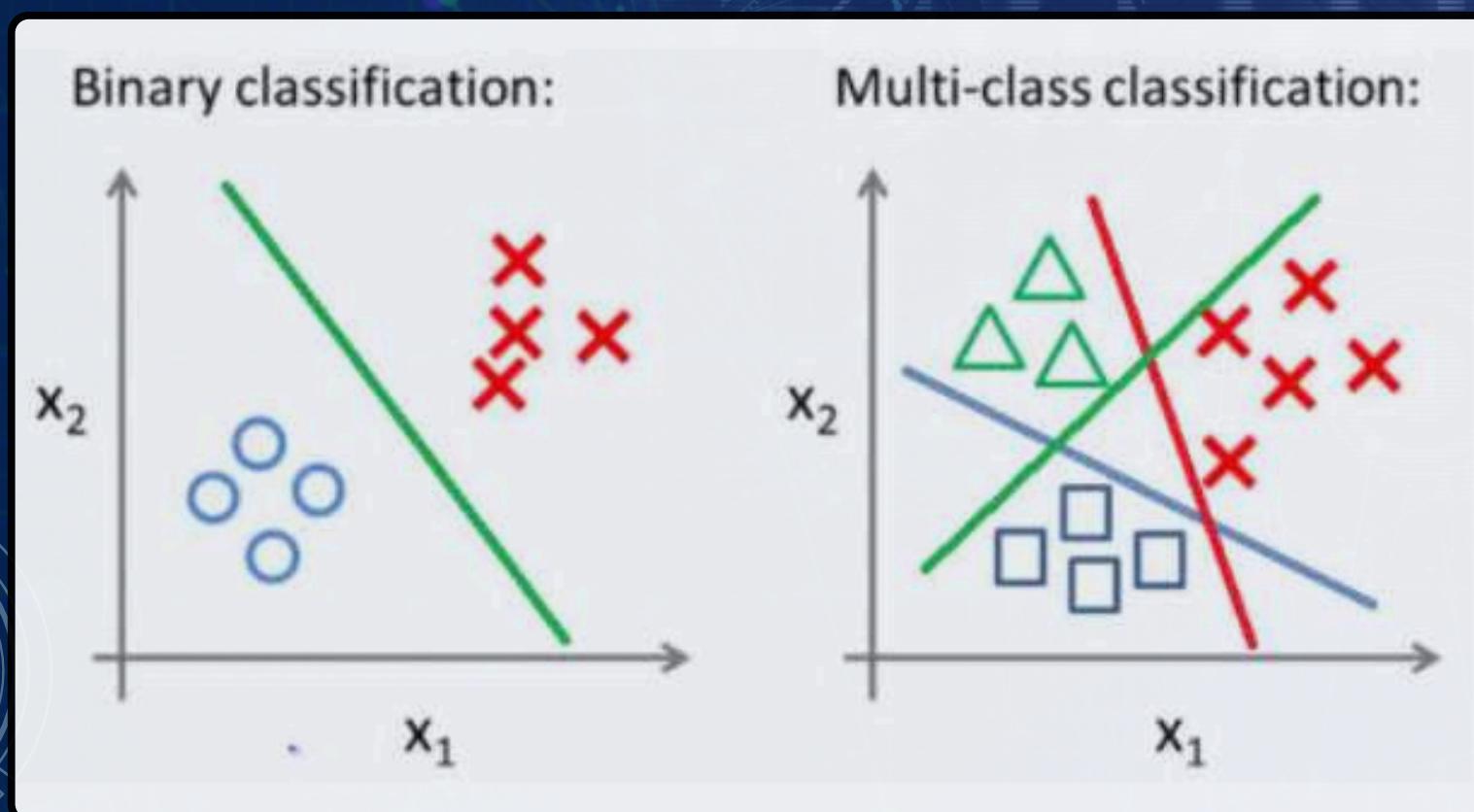
Estrategias Principales

ONE VS ALL

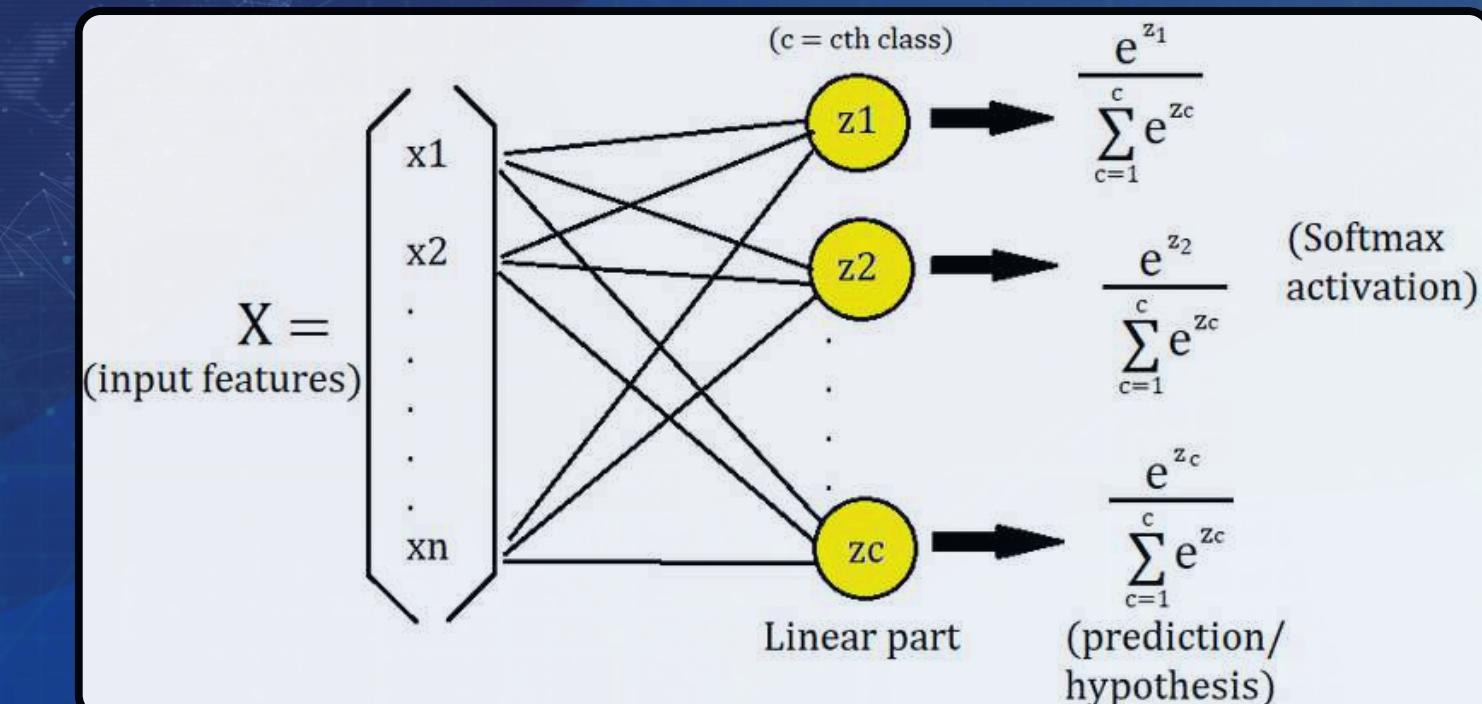
Se entrena un clasificador binario por cada clase, donde esa clase es positiva y todas las demás negativas. En la predicción, se elige la clase cuyo clasificador da la mayor confianza.

ONE VS ONE

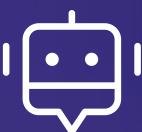
Se entrena un clasificador binario para cada par de clases posibles, es decir, $n(n-1)/2$ clasificadores. En la predicción, cada clasificador vota y se elige la clase con más votos.



SOFTMAX

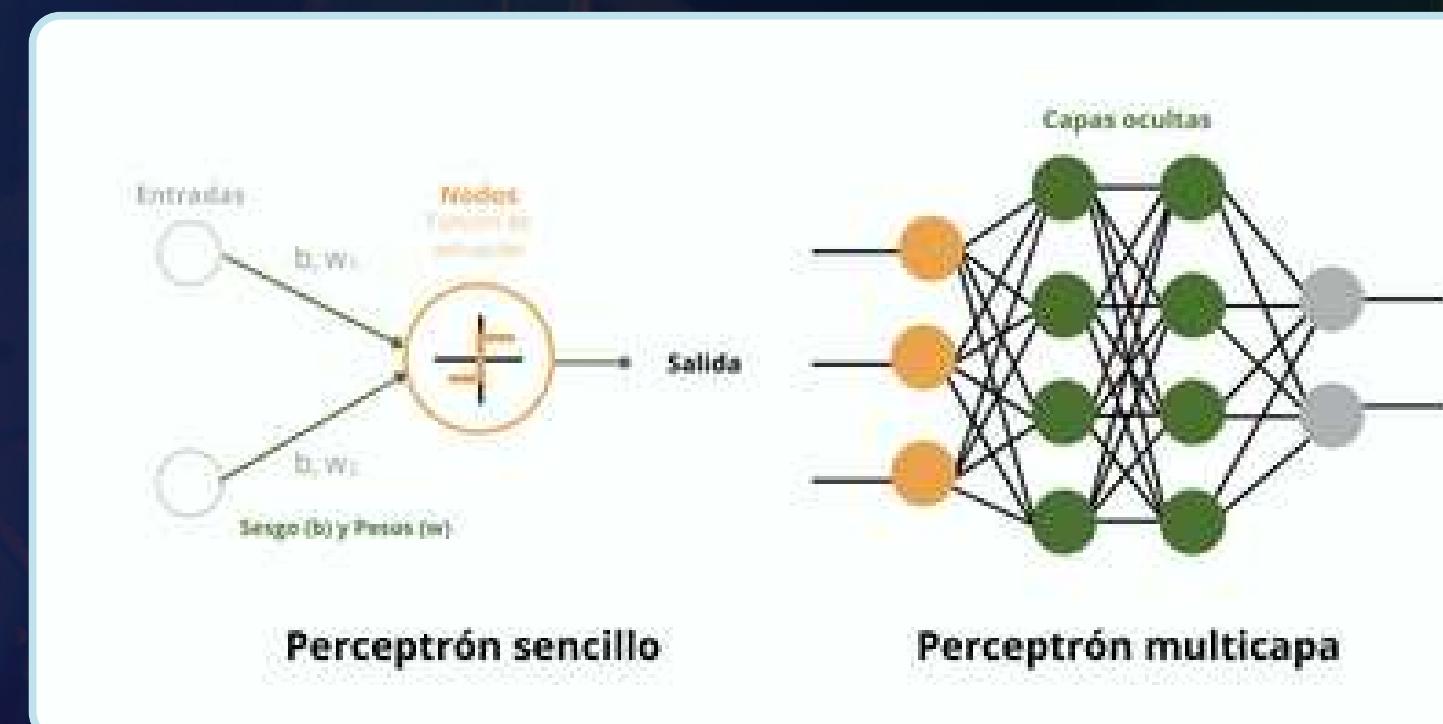
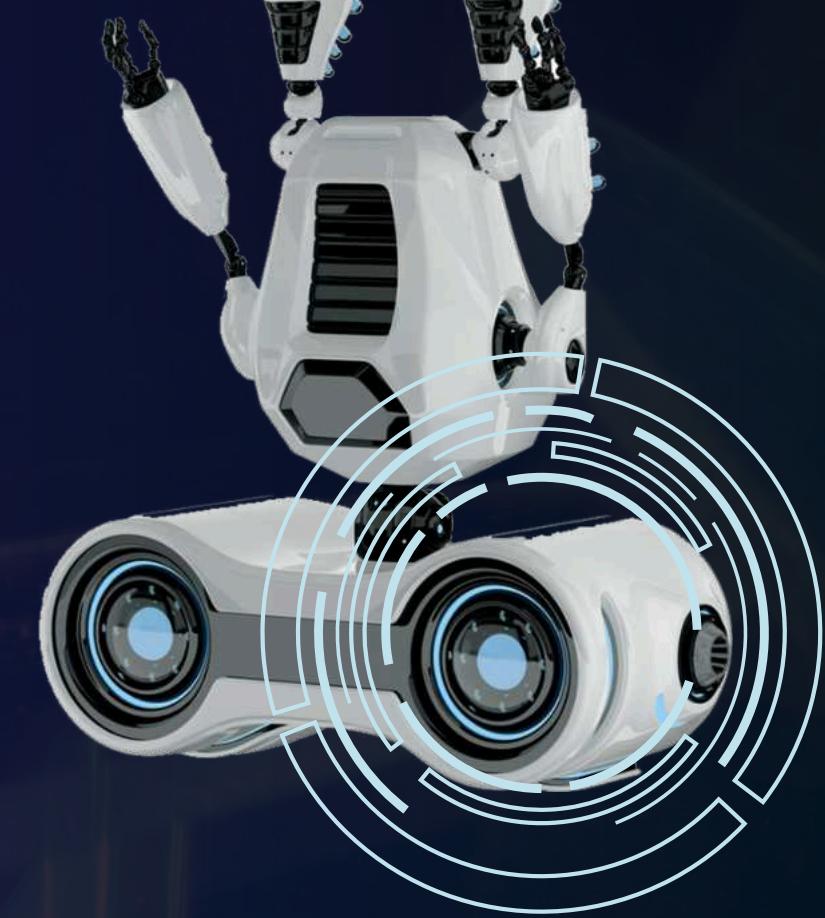
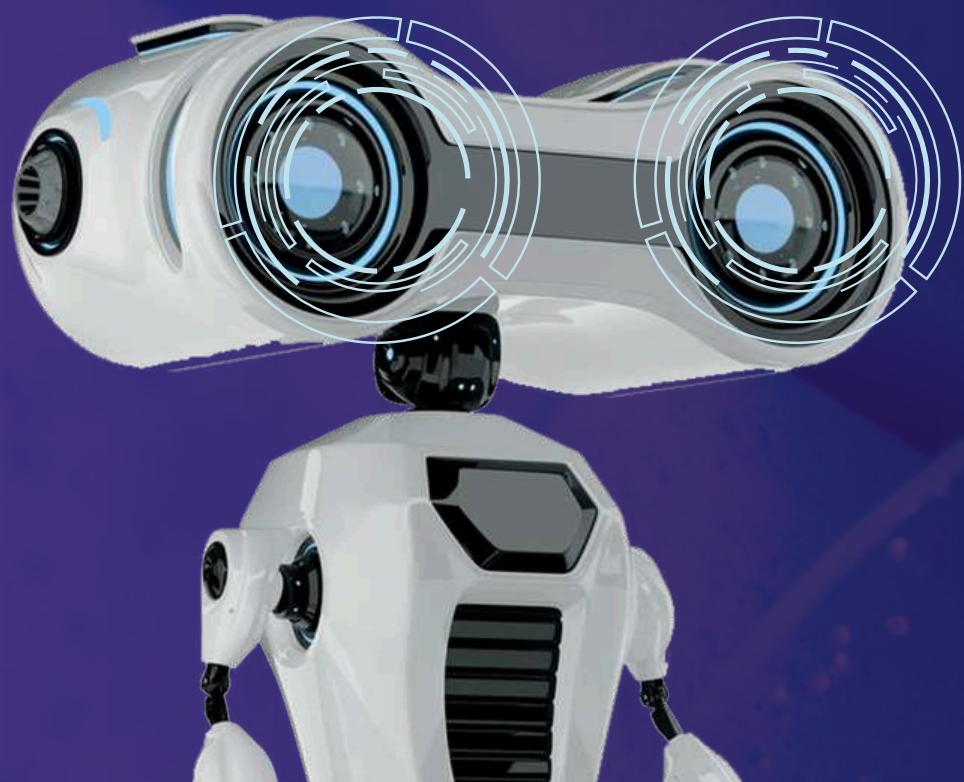






Perceptrón Multicapa

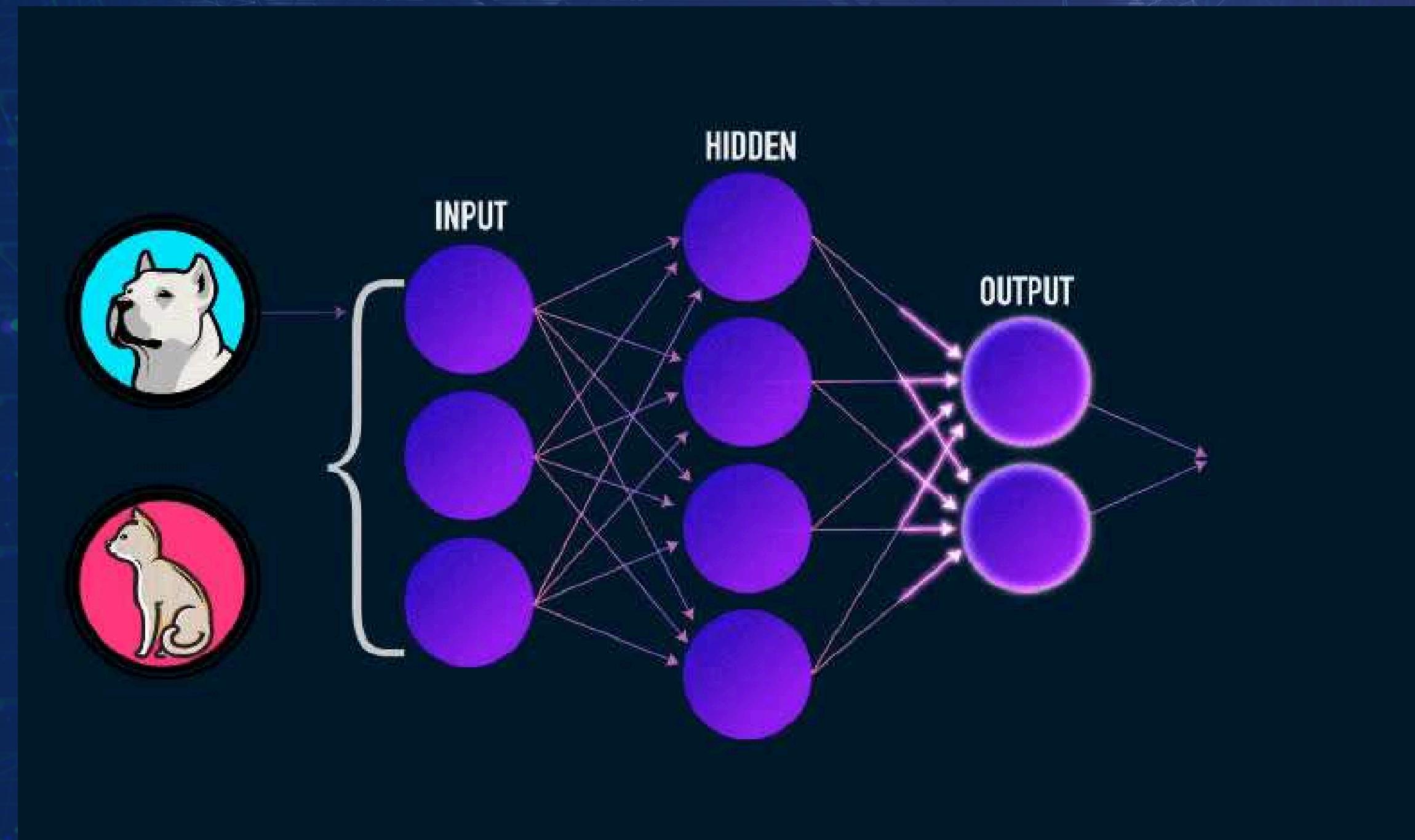
Entender cómo superar las limitaciones del perceptrón simple usando redes neuronales multicapa.





Perceptrón Multicapa

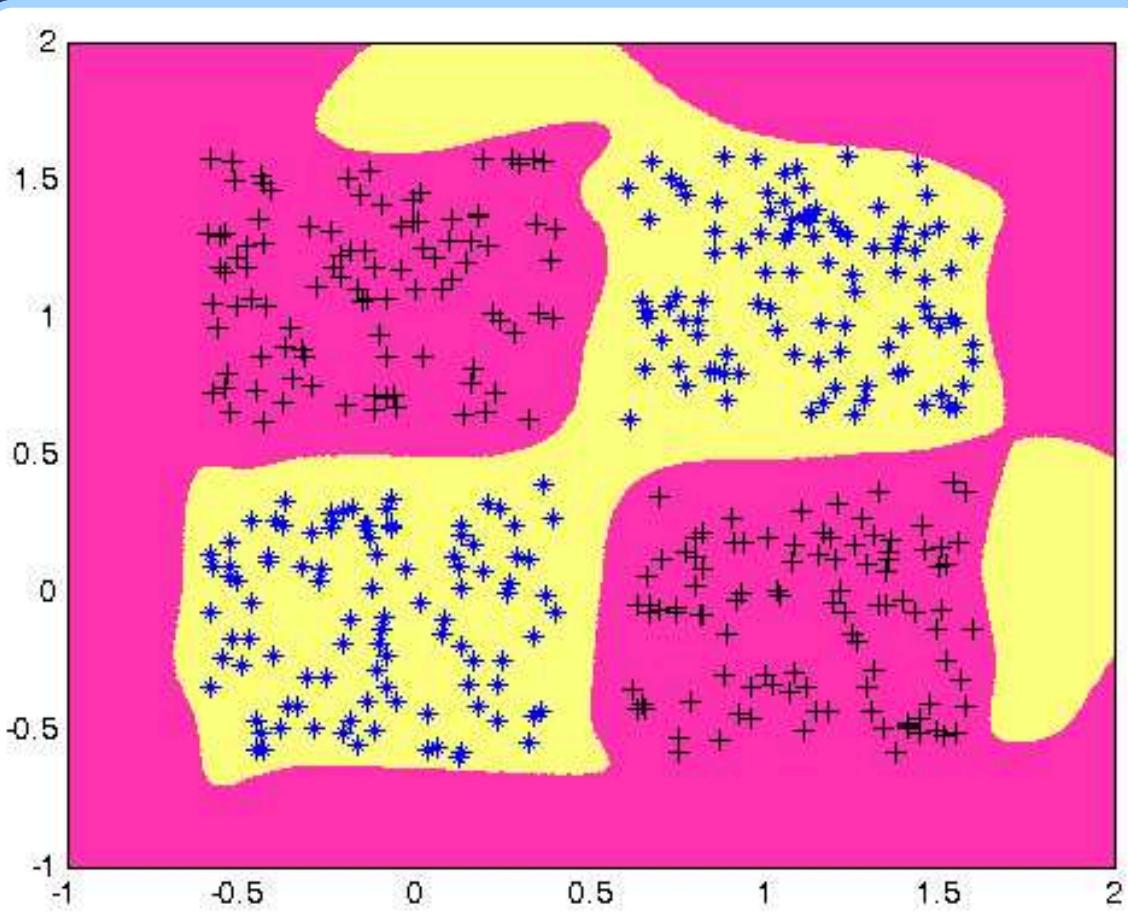
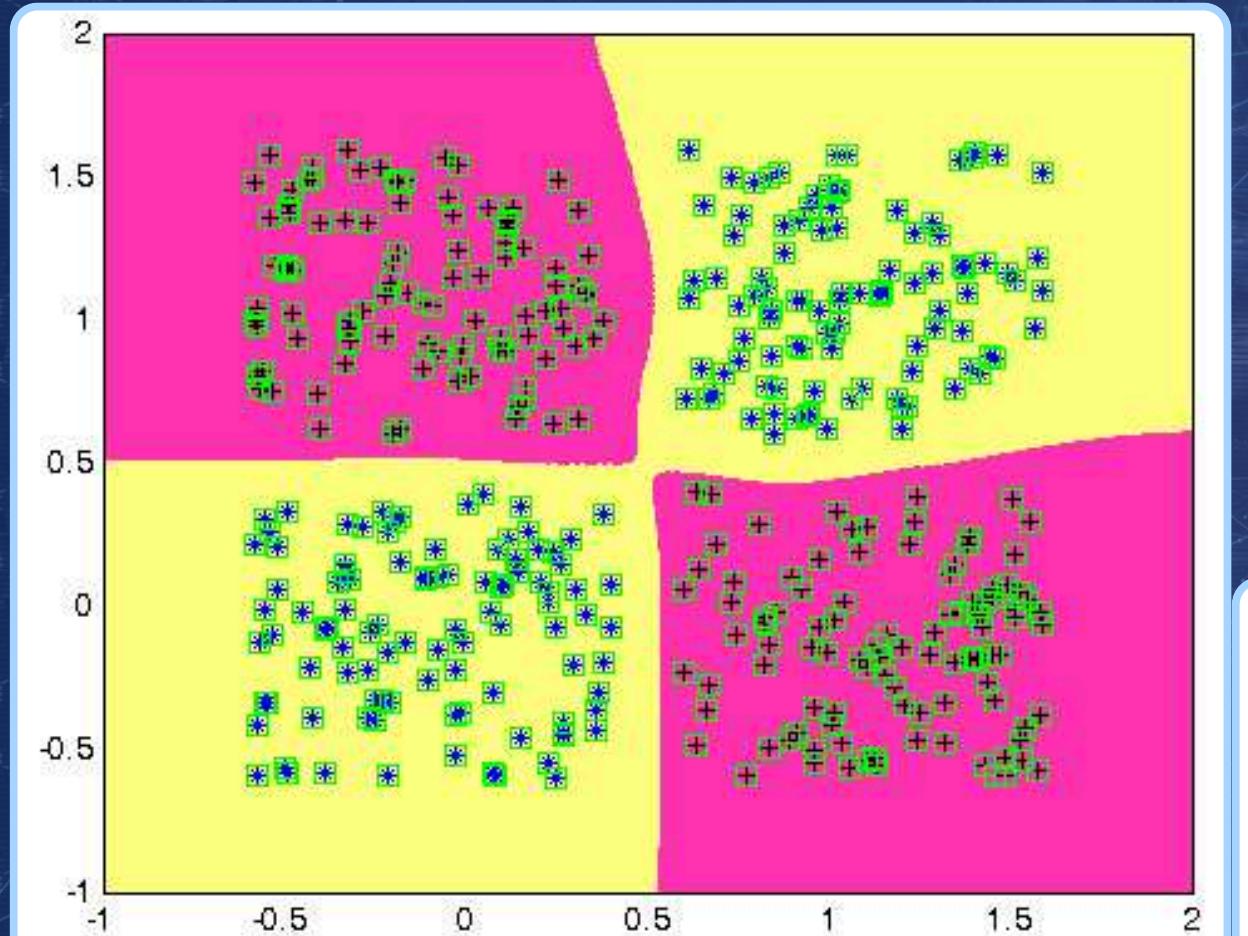
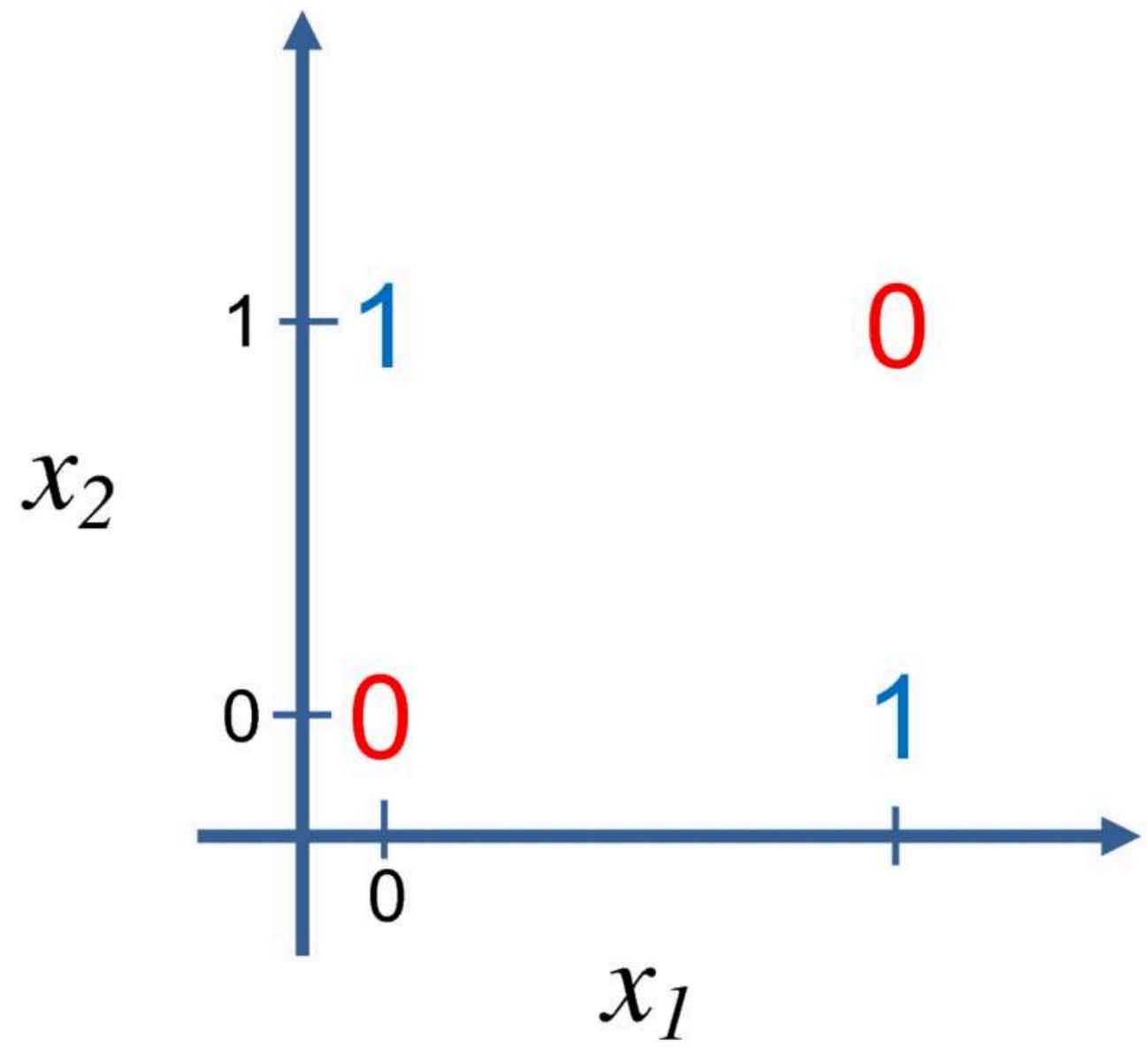
- Arquitectura con al menos una capa oculta.
- Cada neurona en una capa se conecta con todas las de la siguiente (topología totalmente conectada).
- Cada neurona aplica una función de activación no lineal

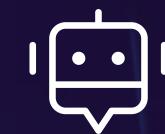




Perceptron Multicapa

- Encontrar solución a problemas que no se pueden separar linealmente





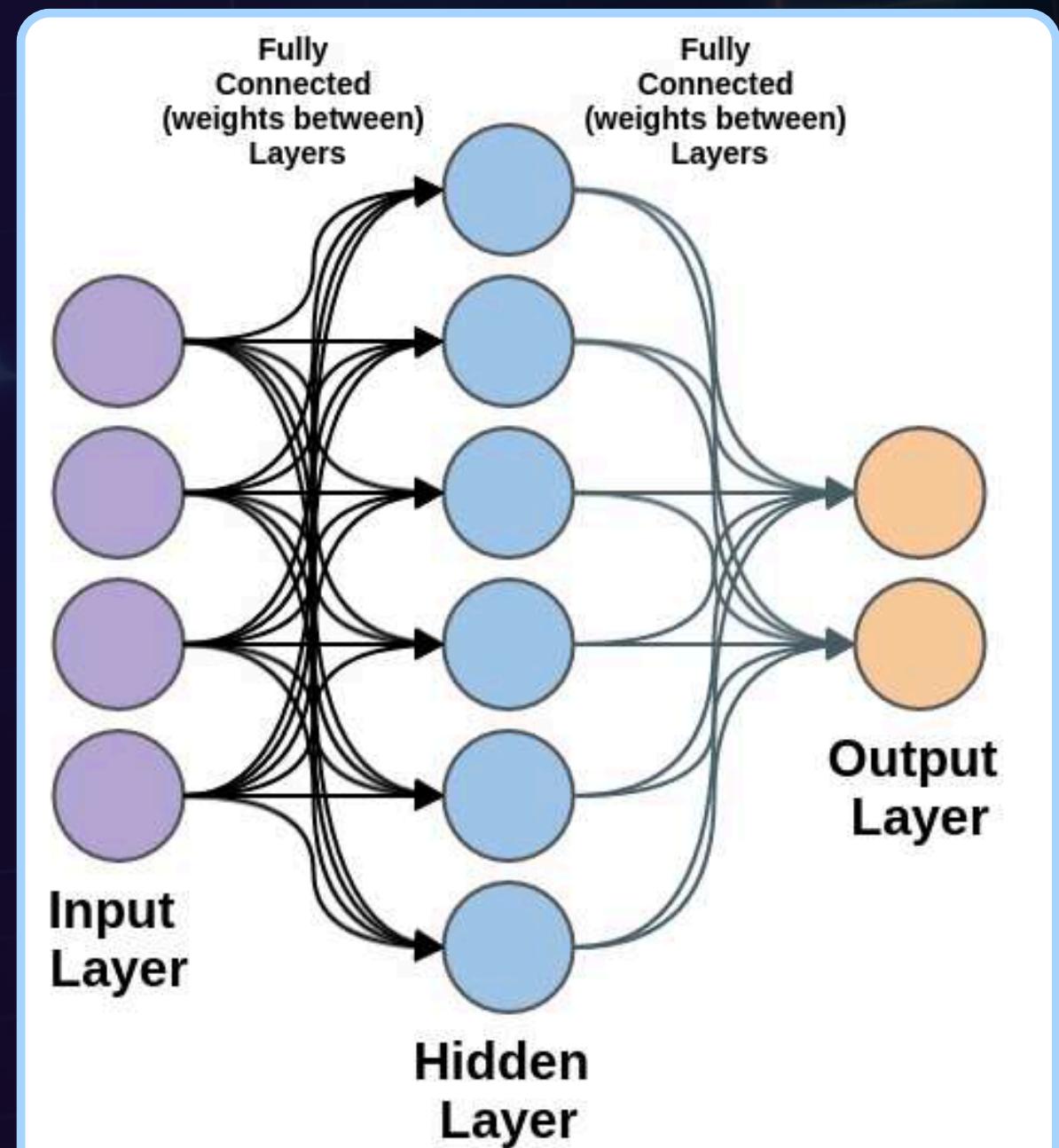
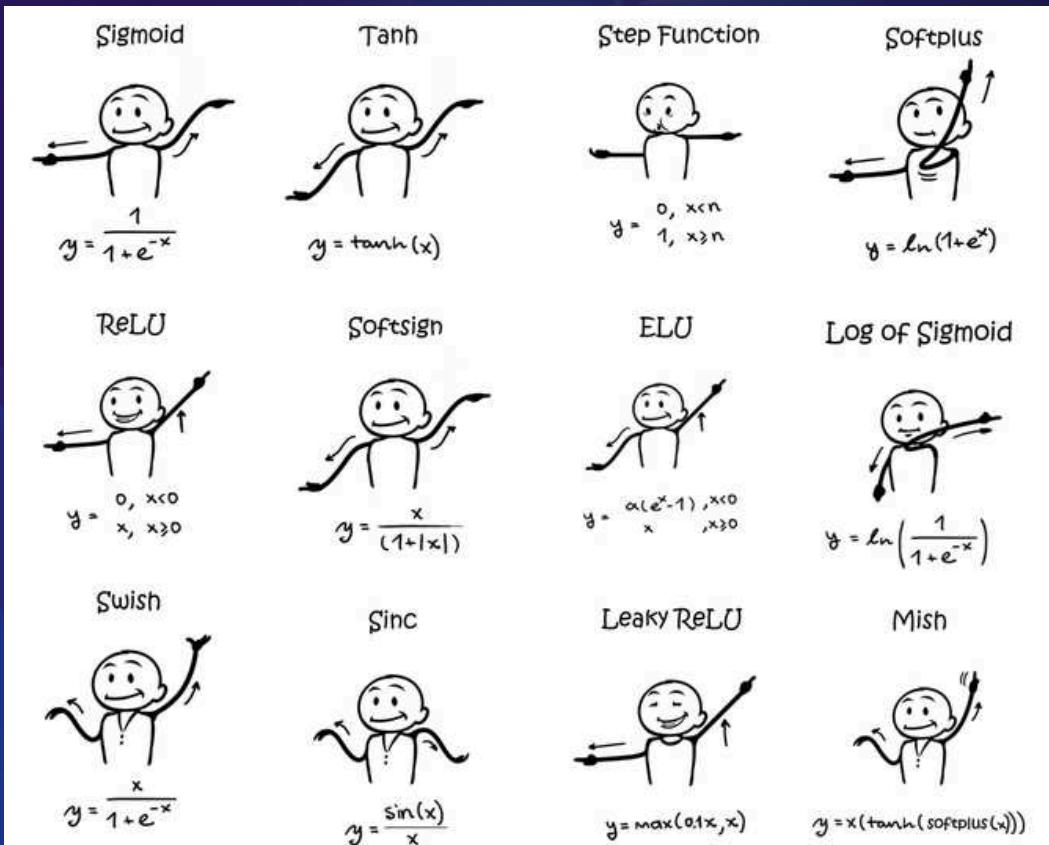
Perceptron Multicapa

Estructura

- Capa de entrada → Capa(s) oculta(s) → Capa de salida
- Cada neurona:

$$z = f(\sum w_i x_i + b)$$

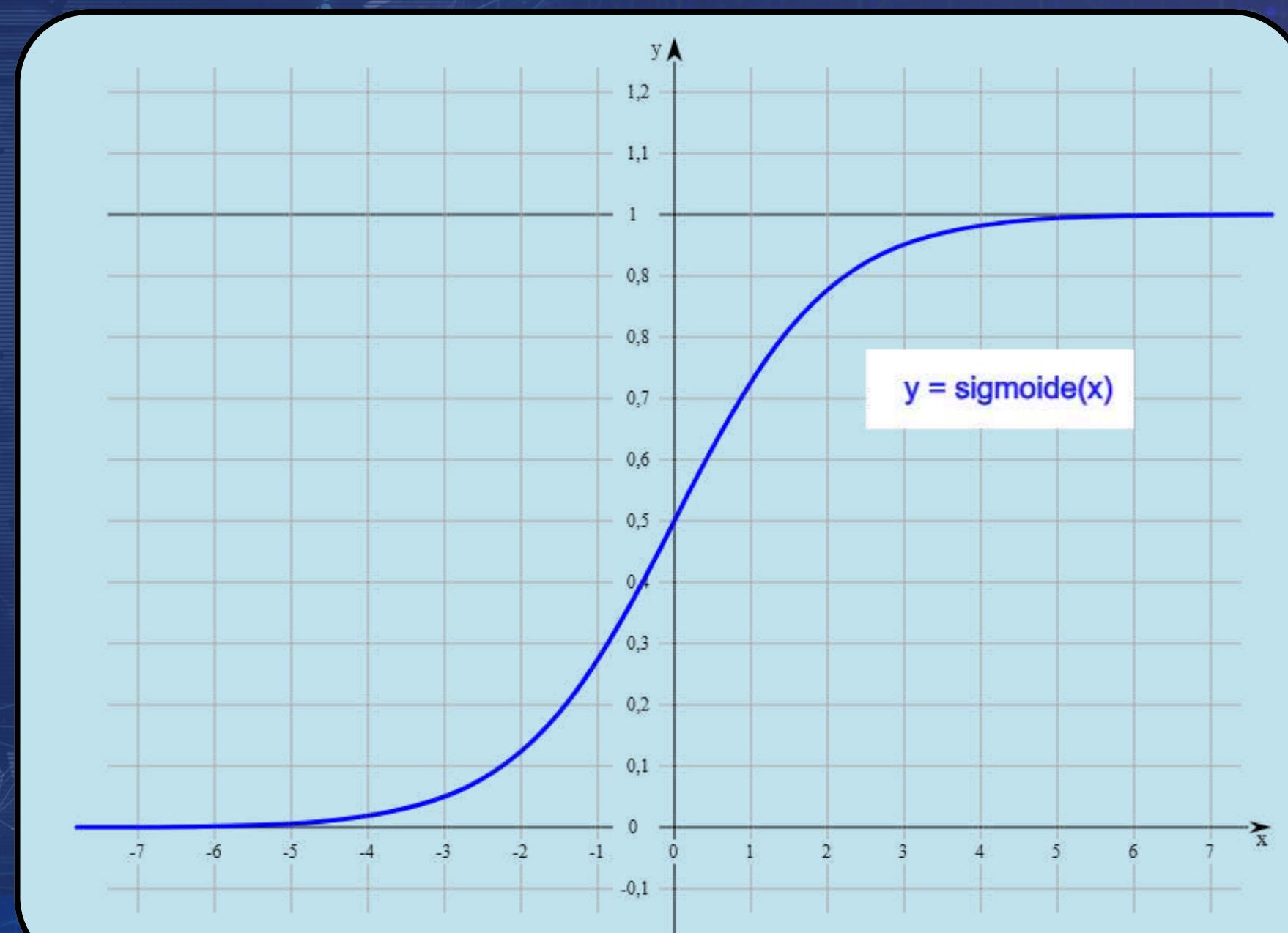
- Funciones de Activación





Función de Activación: Sigmoidal

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





Función de Activación: Sigmoide

Ventajas:

- Salida entre 0 y 1 → útil para probabilidades (especialmente en una red para clasif. binaria).
- Suave y derivable en todos los puntos.

Desventajas:

- Gradientes muy pequeños para valores extremos (problema de vanishing gradient).
- No centrada en cero → puede dificultar el aprendizaje.

Cuándo usarla:

- En la capa de salida para problemas de clasificación binaria.
- En redes pequeñas o simples.



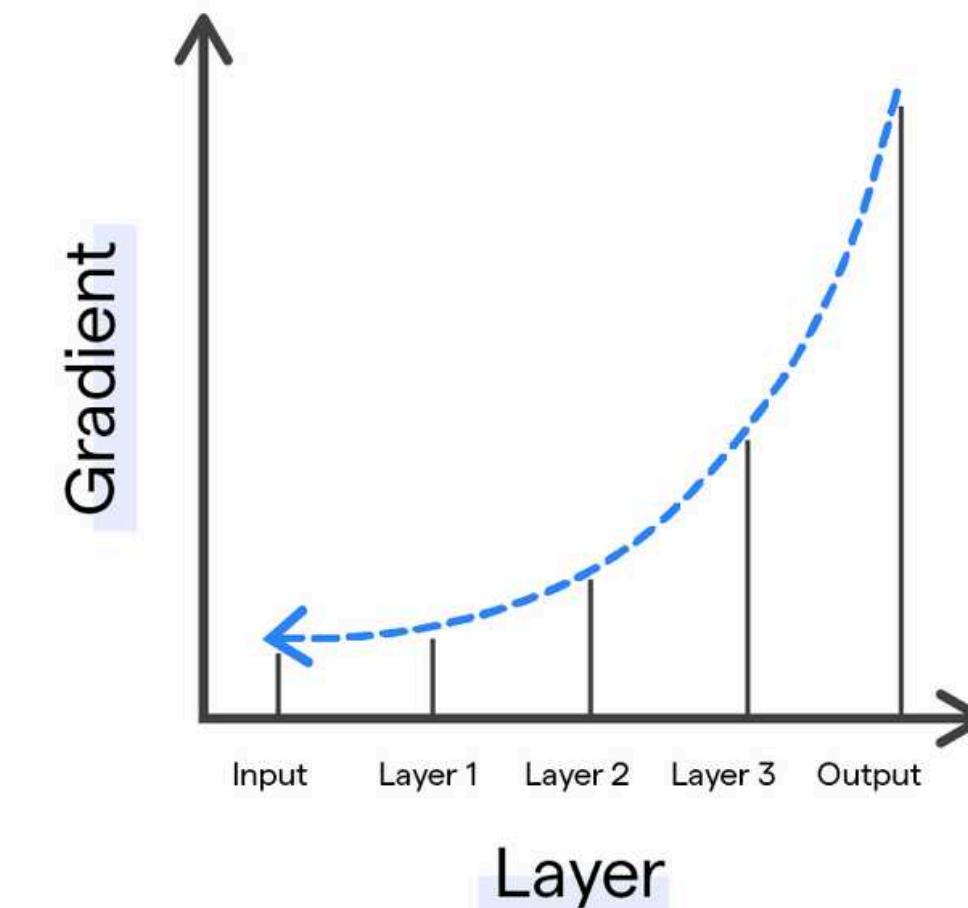
Vanishing Gradient

Es un problema que puede ocurrir durante el entrenamiento de redes neuronales profundas

Se refiere a la situación en la que los gradientes de las funciones de activación se vuelven extremadamente pequeños a medida que se retropropagan a través de las capas de la red.

Puede hacer que el proceso de entrenamiento sea muy lento

Vanishing Gradient Problem





Vanishing Gradient

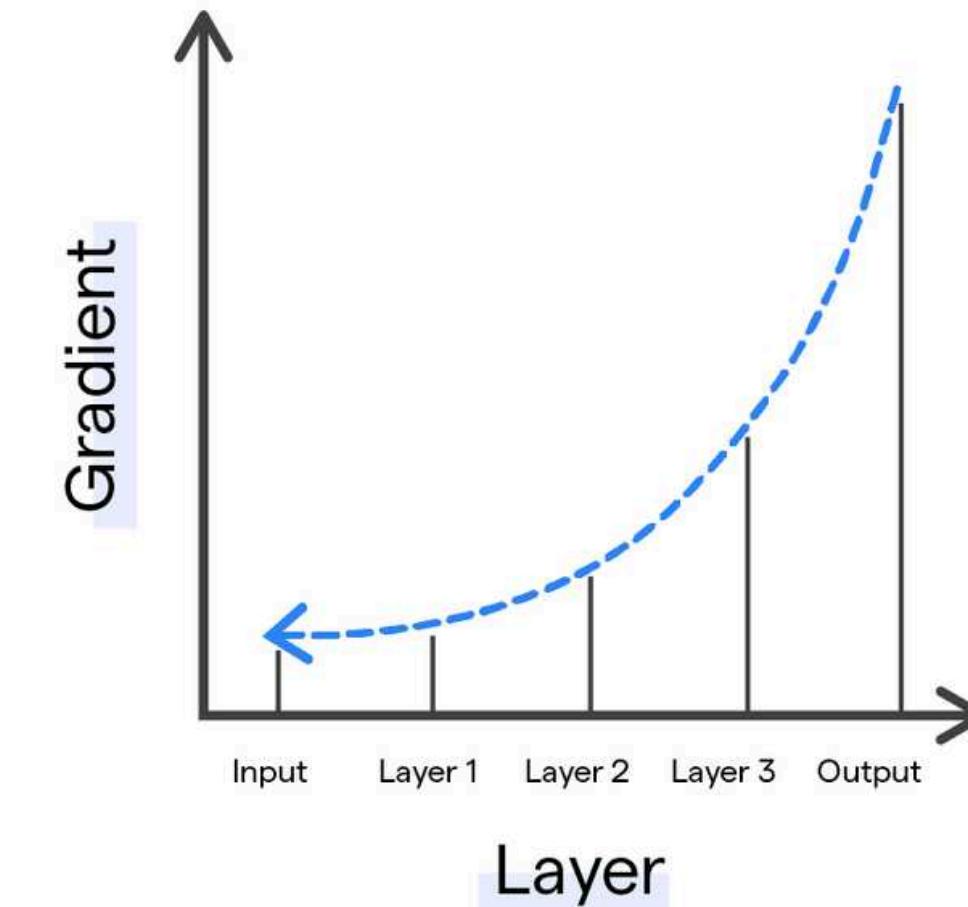
¿Por qué es un problema?

Cuando el gradiente se vuelve muy pequeño (casi cero), las actualizaciones de los pesos se vuelven muy pequeñas, y esto afecta negativamente el entrenamiento.

¿Cómo evitar el vanishing gradient?

- Funciones de activación alternativas
- Inicialización adecuada de los pesos
- Redes residuales
- Uso de capas de normalización

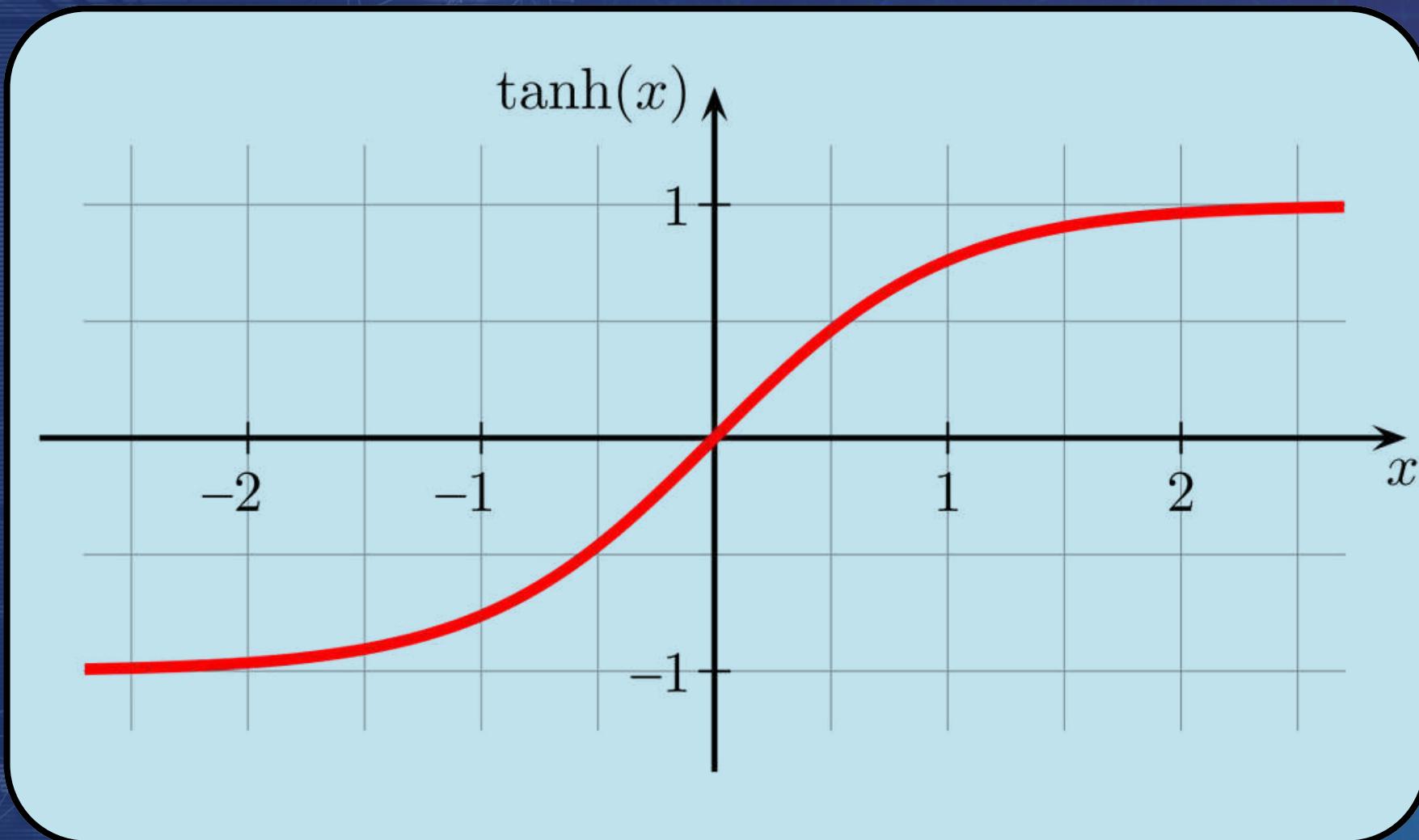
Vanishing Gradient Problem





Función de Activación: Tangente Hiperbolica

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





Función de Activación: Tangente Hiperbolica

Ventajas:

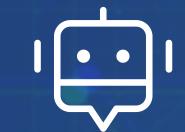
- Salida entre -1 y $1 \rightarrow$ centrada en cero, lo que mejora la dinámica del entrenamiento.
- Mejor gradiente que la sigmoide (aunque también sufre vanishing gradient).

Desventajas:

- Todavía tiene saturación en extremos \rightarrow riesgo de vanishing gradient.
- Ligeramente más costosa computacionalmente que ReLU.

Cuándo usarla:

- En capas ocultas de redes pequeñas o RNNs cuando necesitas una salida centrada en cero.



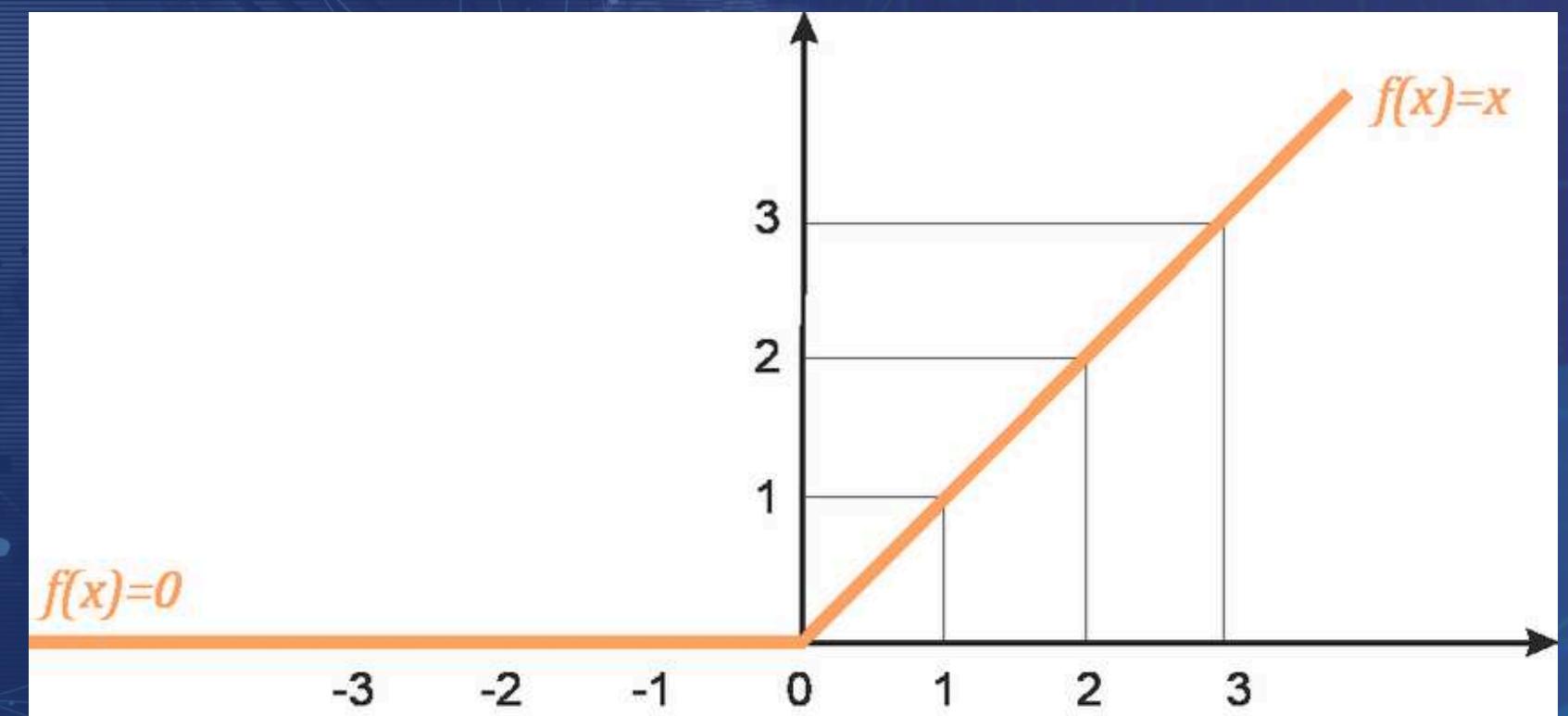
Función de Activación: ReLU (Unidad Lineal Rectificada)

Transforma la entrada de una neurona según la siguiente regla:

$$f(x) = \max(0, x)$$

Esto significa que:

- Si la entrada x es mayor o igual a 0, la salida es x .
- Si la entrada x es menor que 0, la salida es 0.





Función de Activación: ReLU

Ventajas:

- Muy eficiente computacionalmente, solo requiere una comparación.
- No sufre vanishing gradient para valores positivos, mantiene gradientes significativos
- **Promueve la esparsidad:** Al producir ceros para entradas negativas, algunas neuronas permanecen inactivas, lo que puede mejorar la eficiencia y generalización del modelo

Desventajas:

- **Neurona muerta:** Si una neurona recibe entradas negativas constantemente, su salida será siempre cero, y puede dejar de aprender.
- **No centrada en cero:** La salida es siempre no negativa, lo que puede afectar la dinámica del entrenamiento.

Cuándo usarla:

- Función por defecto para capas ocultas en redes profundas modernas.



Función de Activación: Softmax

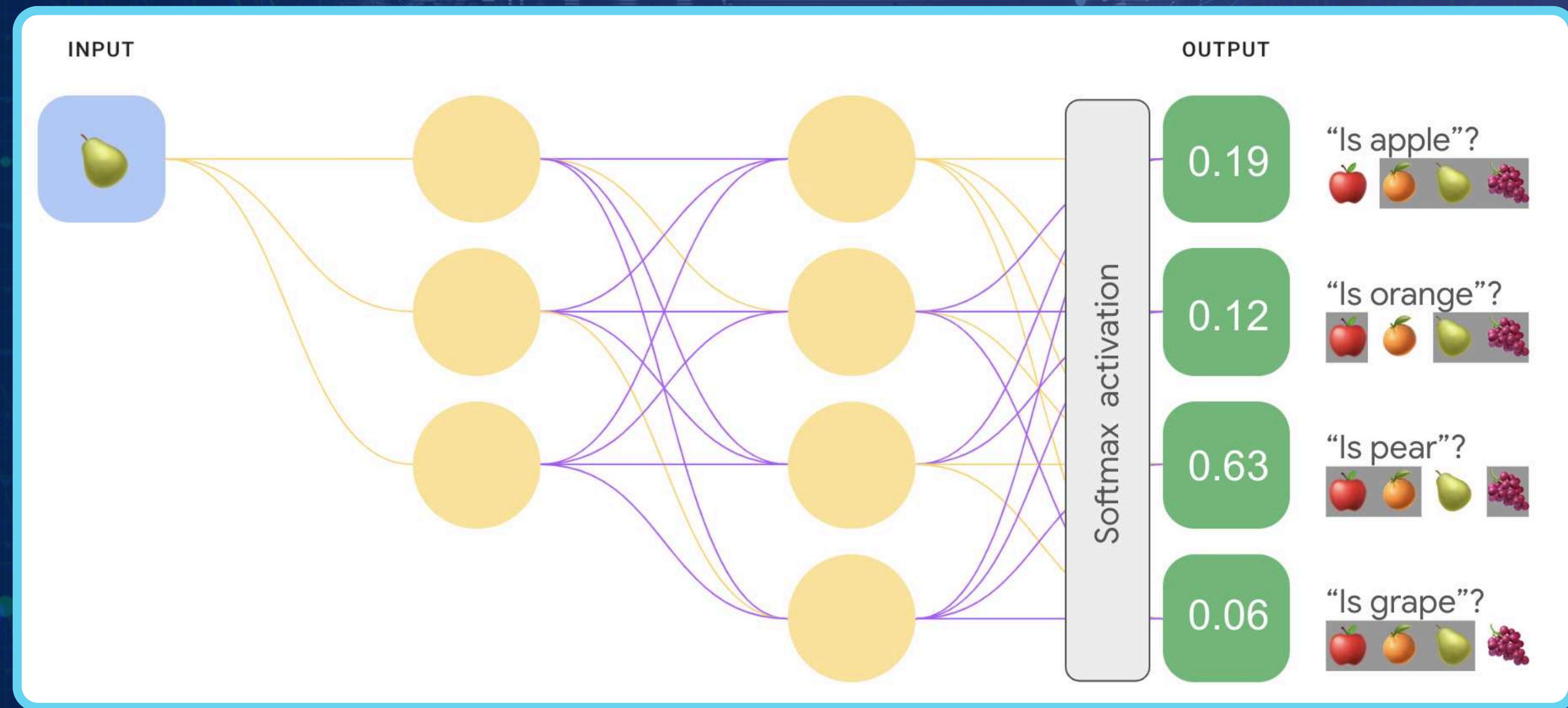
La función softmax convierte un vector de números reales (que pueden ser positivos o negativos) en un vector de probabilidades, donde:

- Todos los valores están entre 0 y 1.
- La suma de todos los valores es igual a 1.
- El valor más alto indica la clase predicha.

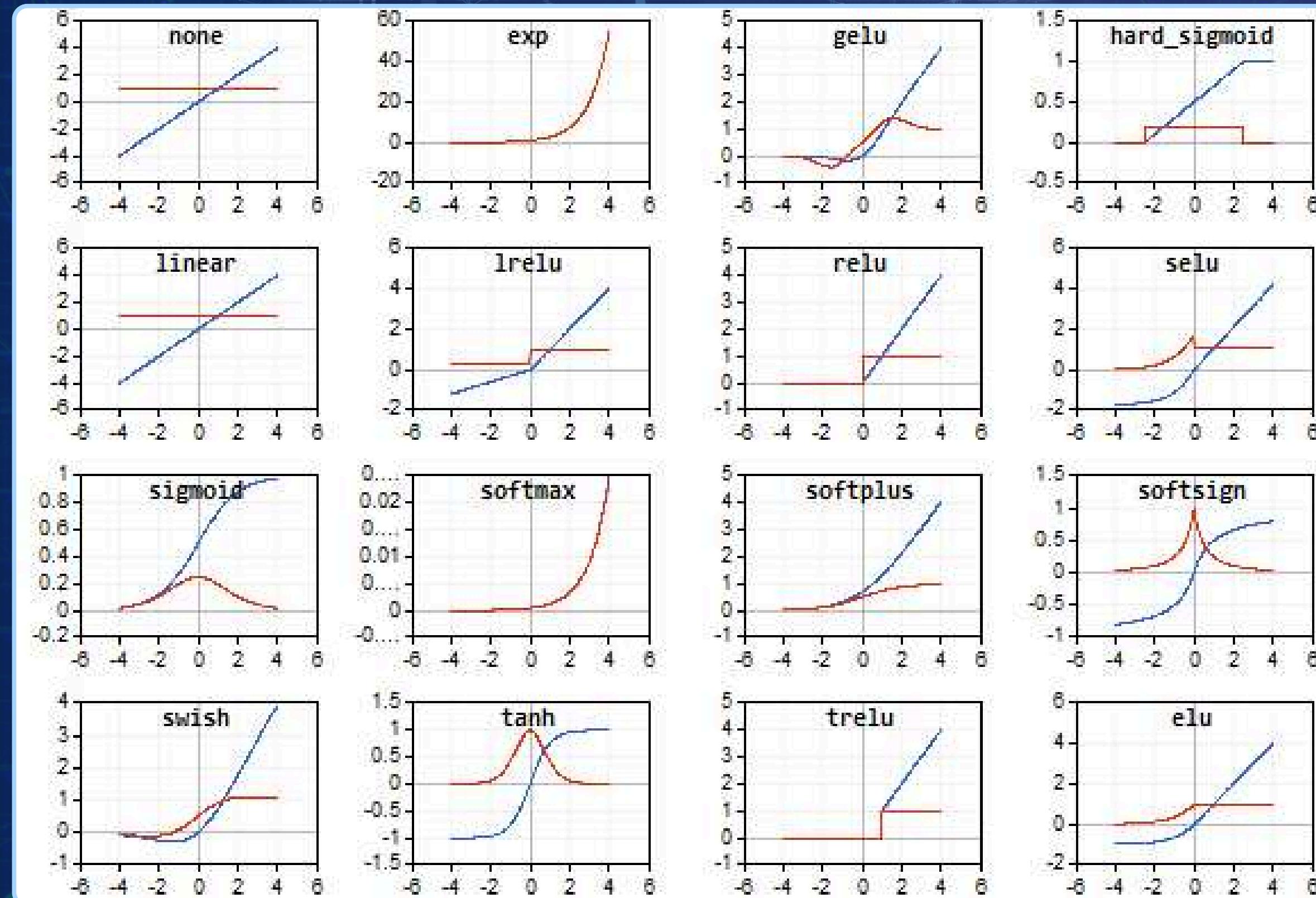
$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad \text{para } i = 1, \dots, n$$



Función de Activación: Softmax



Función de Activación y sus derivadas



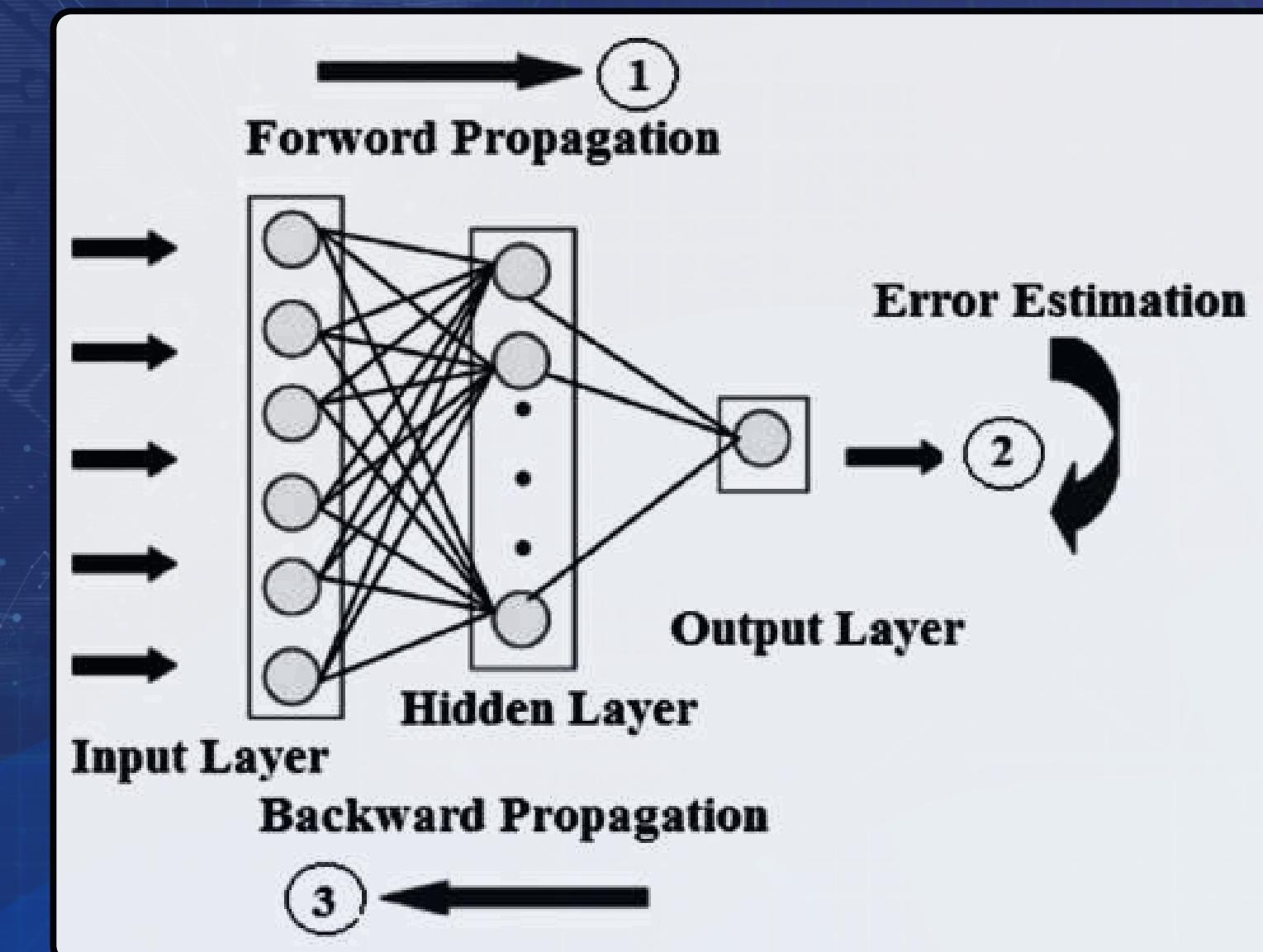


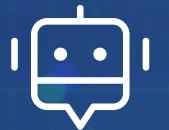
Ciclo de Entrenamiento

En esencia se compone de dos etapas que forman el ciclo de entrenamiento de una red neuronal y permiten que el MLP aprenda patrones complejos y no lineales.

Su algoritmo principal se basa en dos fases clave:

1. Propagación hacia adelante (Forward Propagation)
2. Propagación hacia atrás (Backpropagation)



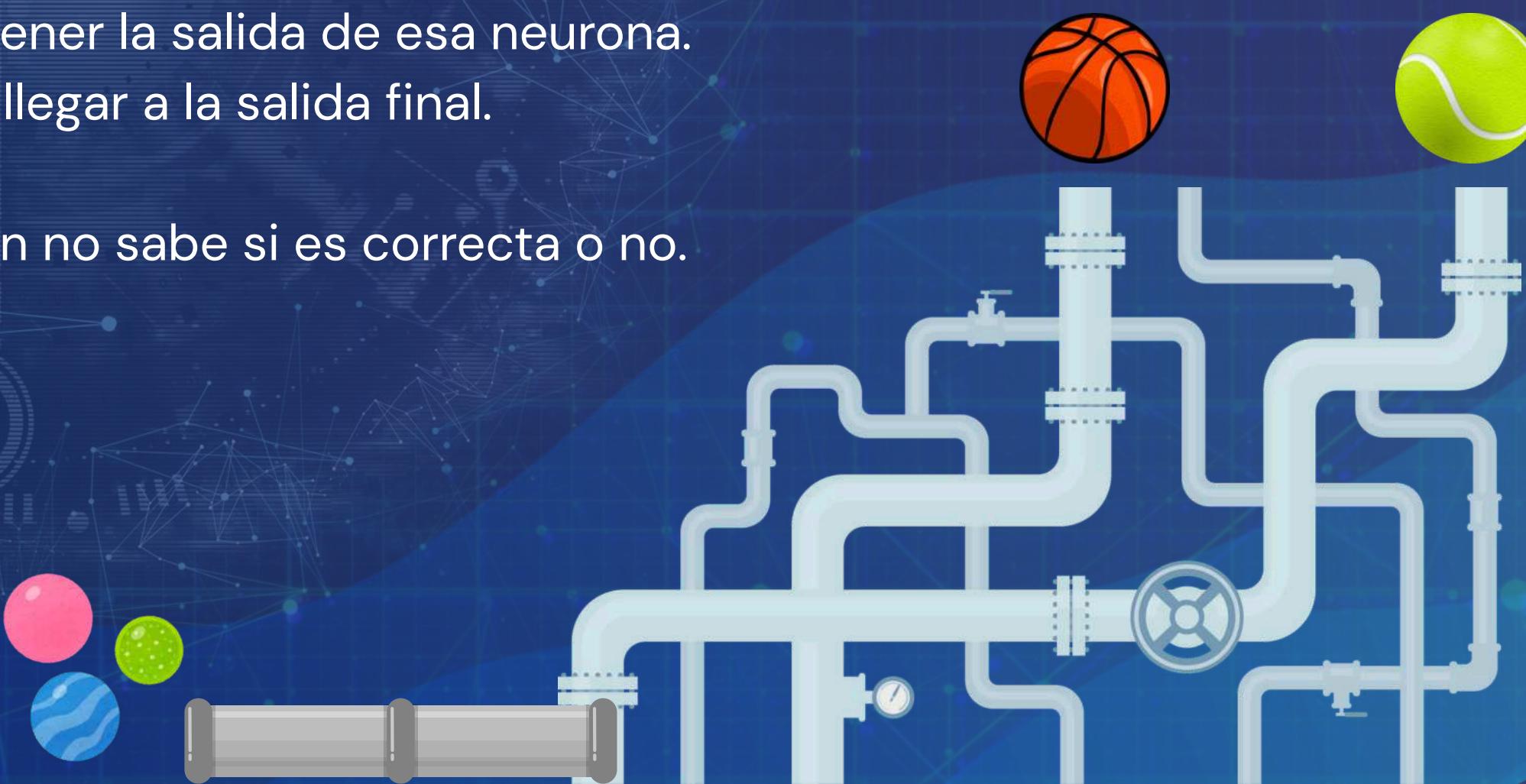


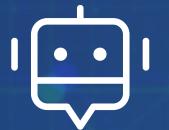
Forward Propagation

Imagina que lanzas una pelota desde el inicio de una tubería con varias válvulas (las capas de la red). En cada punto, la pelota (la información) pasa por una transformación (función de activación) que la lleva a la siguiente válvula.

- Tomas los datos de entrada (por ejemplo, una imagen, un número, etc.).
- Multiplicas cada entrada por su peso y sumas el sesgo.
- Aplicas una función de activación para obtener la salida de esa neurona.
- Repites este proceso capa por capa hasta llegar a la salida final.

Resultado: la red te da una predicción, pero aún no sabe si es correcta o no.





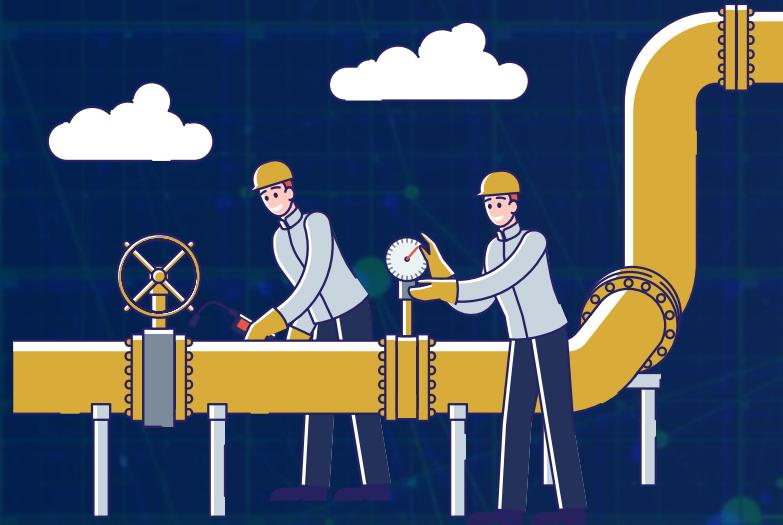
Backpropagation

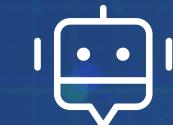
Ahora imagina que comparas el resultado de la red con lo que debería haber salido.

Si está mal, envías un mensaje de corrección hacia atrás por la tubería.

- Calculas el error (la diferencia entre la salida esperada y la obtenida).
- Usas ese error para calcular cuánto influenció cada peso en el error.
- Ajustas los pesos en dirección contraria al error, usando una técnica llamada descenso del gradiente.

Resultado: la red aprende al modificar sus pesos para cometer menos error la próxima vez.





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



Algoritmo

- Entrenamiento
 - *Ajuste de pesos (Parámetros)*
- Forward Propagation
(Propagación Hacia Adelante)
- Cálculo de la pérdida (Loss Function)
- Backpropagation (Propagación Hacia Atrás)



- Entrenamiento
 - Este proceso (forward → loss → backward → update) se repite durante varias épocas o hasta que el modelo converja.
- Hiperparámetros
 - Tasa de aprendizaje
 - Número máximo de épocas
 - Número de capas ocultas
 - Número de neuronas por capa oculta
 - Función de Activación



Algoritmo

1 . Inicialización

- Se define la arquitectura: *número de capas, número de neuronas por capa.*
- Se inicializan aleatoriamente:
 - Los pesos w_{ij} de cada conexión entre neuronas.
 - Los sesgos (biases) b_j .
- Se determina:
 - Función de activación (ej. *sigmoide, ReLU, tanh*).
 - Tasa de aprendizaje η .
 - Número de épocas.



Algoritmo

2. Forward Propagation (Propagación Hacia Adelante)

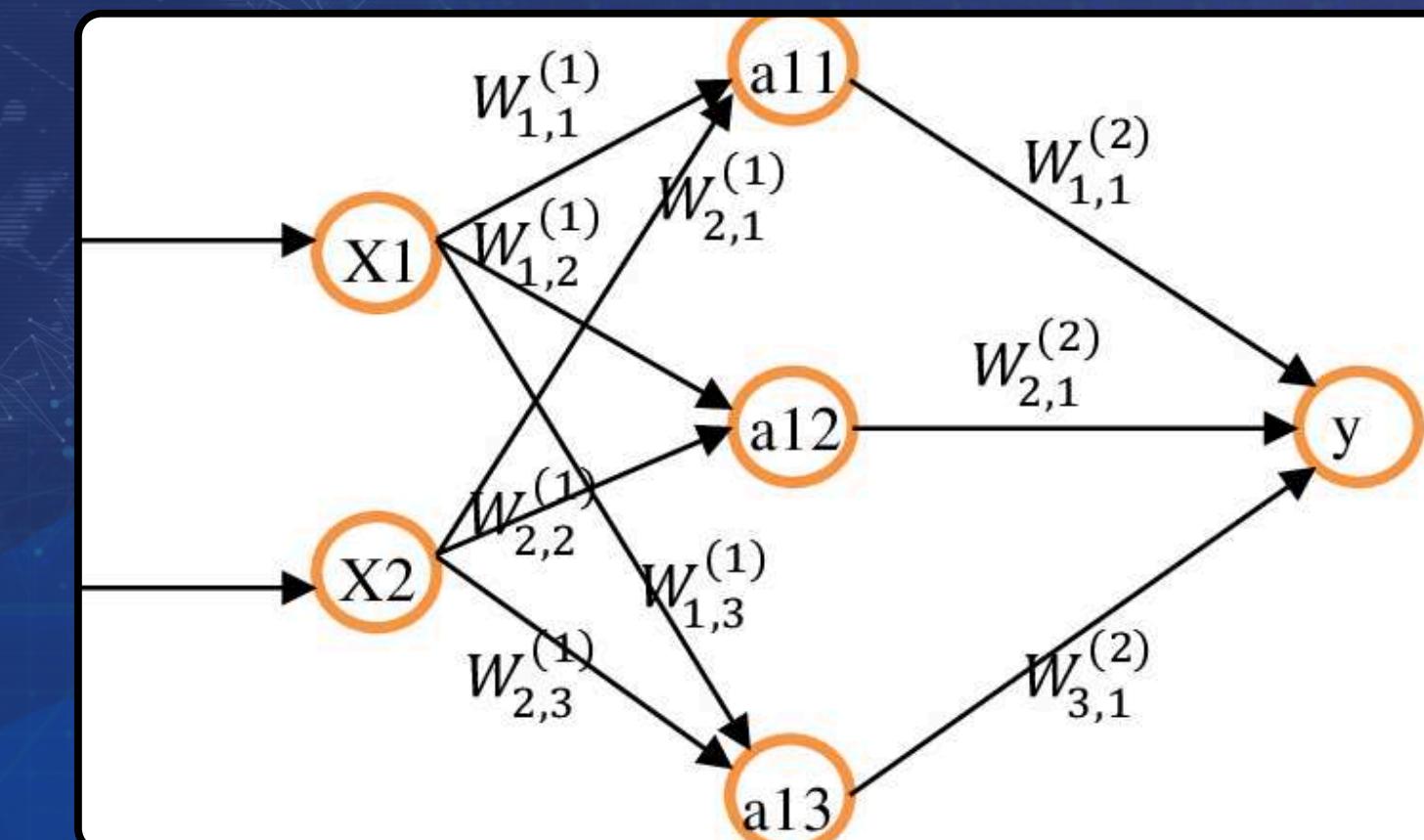
Se calcula la salida de la red para una entrada dada, pasando los datos a través de todas las capas.

1. **Entrada a la red:** se pasa por cada neurona de cada capa (oculta y de salida).
2. **Cálculo en cada neurona:**

$$z_j^{(l)} = \sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}$$
$$a_j^{(l)} = f(z_j^{(l)})$$

Donde:

- $z_j^{(l)}$: entrada neta de la neurona j en la capa l
- $a_j^{(l)}$: salida (activación)
- f : función de activación (sigmoid, tanh, ReLU, etc.)





Algoritmo

3. Cálculo del error

Se compara la predicción final $\hat{y} = a^{(L)}$ con el valor real y usando una *función de pérdida*.

Ejemplos comunes:

- Error Cuadrático Medio (MSE) → para regresión
- Entropía Cruzada Binaria → para clasificación binaria
- Entropía Cruzada Categórica → para clasificación multiclasa

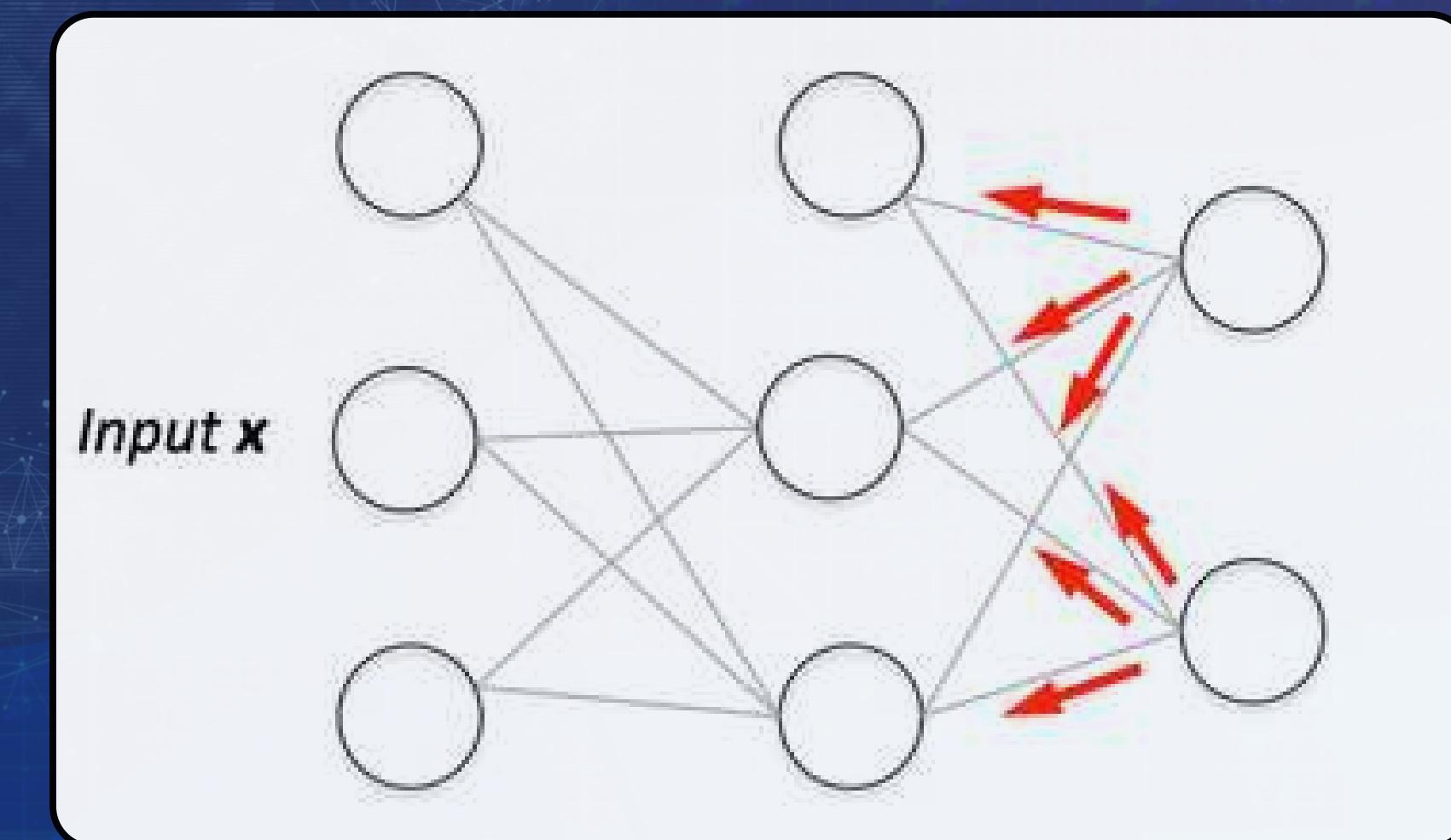


Algoritmo

4. Retropropagación del error (Backpropagation)

Este paso ajusta los pesos para reducir el error:

- Error en la capa de salida
- Propagar el error hacia atrás para cada capa l





Algoritmo

5. Actualización de Pesos y Sesgos

Usamos los gradientes para ajustar los parámetros con descenso del gradiente :

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

$$b_j^{(l)} \leftarrow b_j^{(l)} - \eta \cdot \delta_j^{(l)}$$

Donde:

- η : tasa de aprendizaje (learning rate)

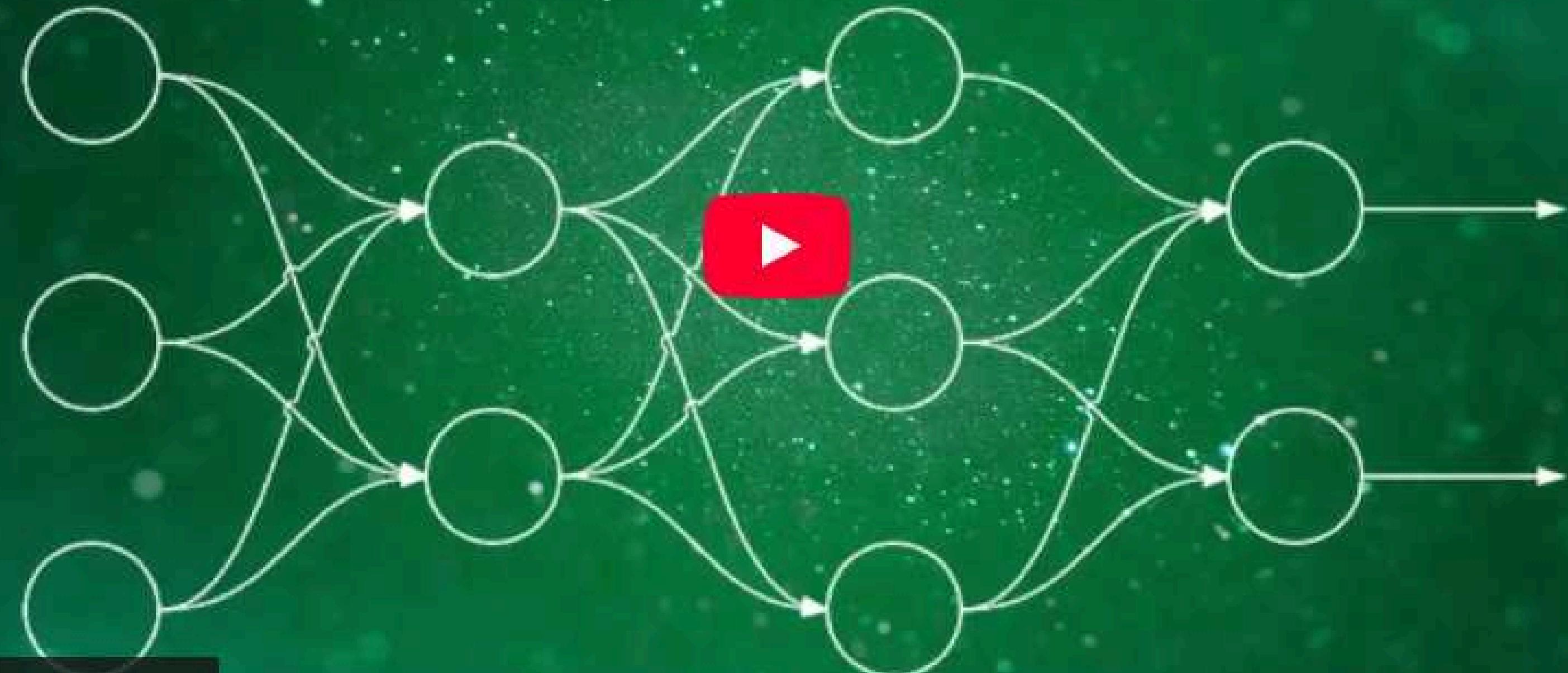


¿Qué es una Red Neuronal? Parte 2 : La Red | DotCSV



Share

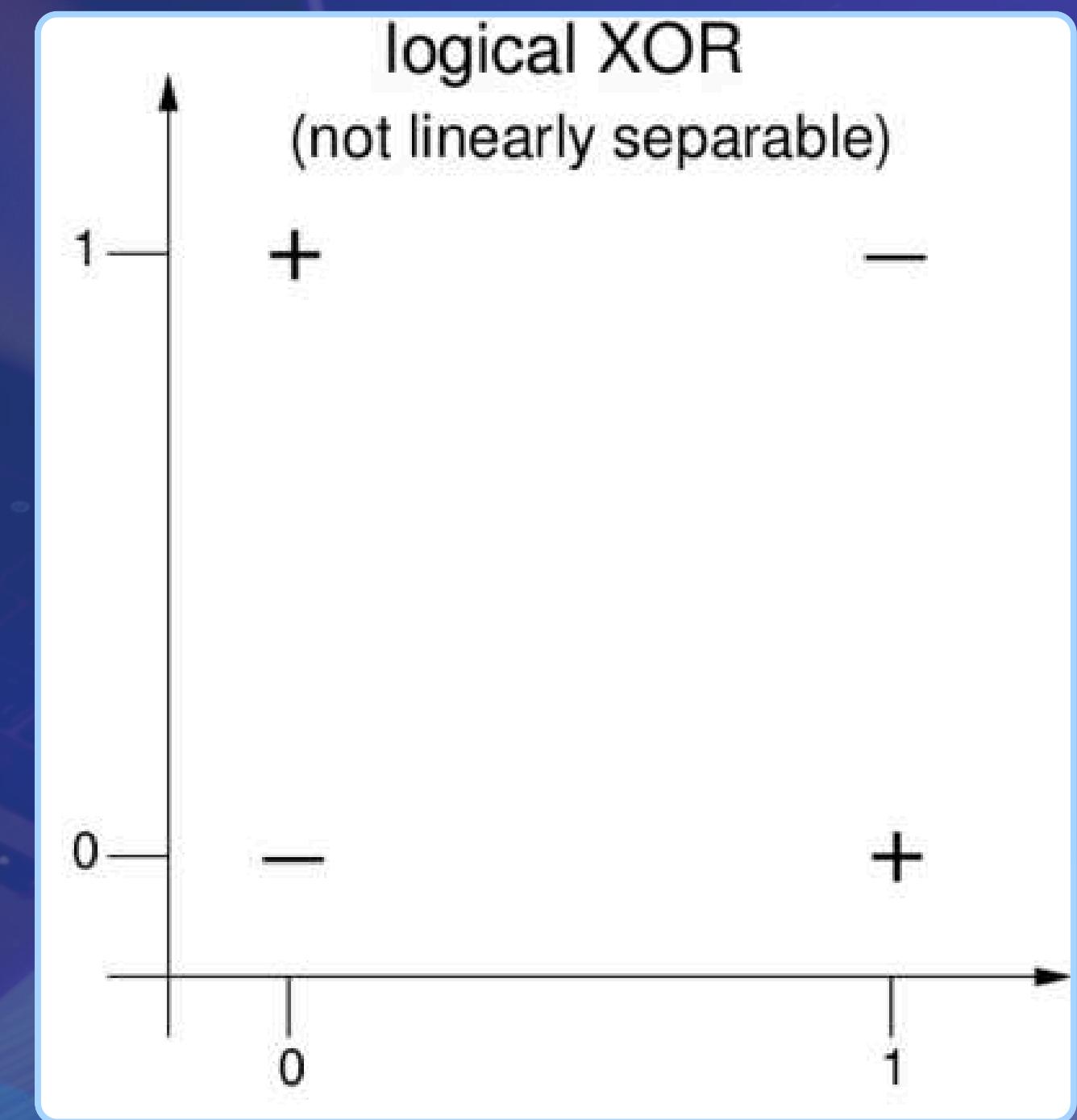
PARTE 2 | LA RED



Watch on YouTube

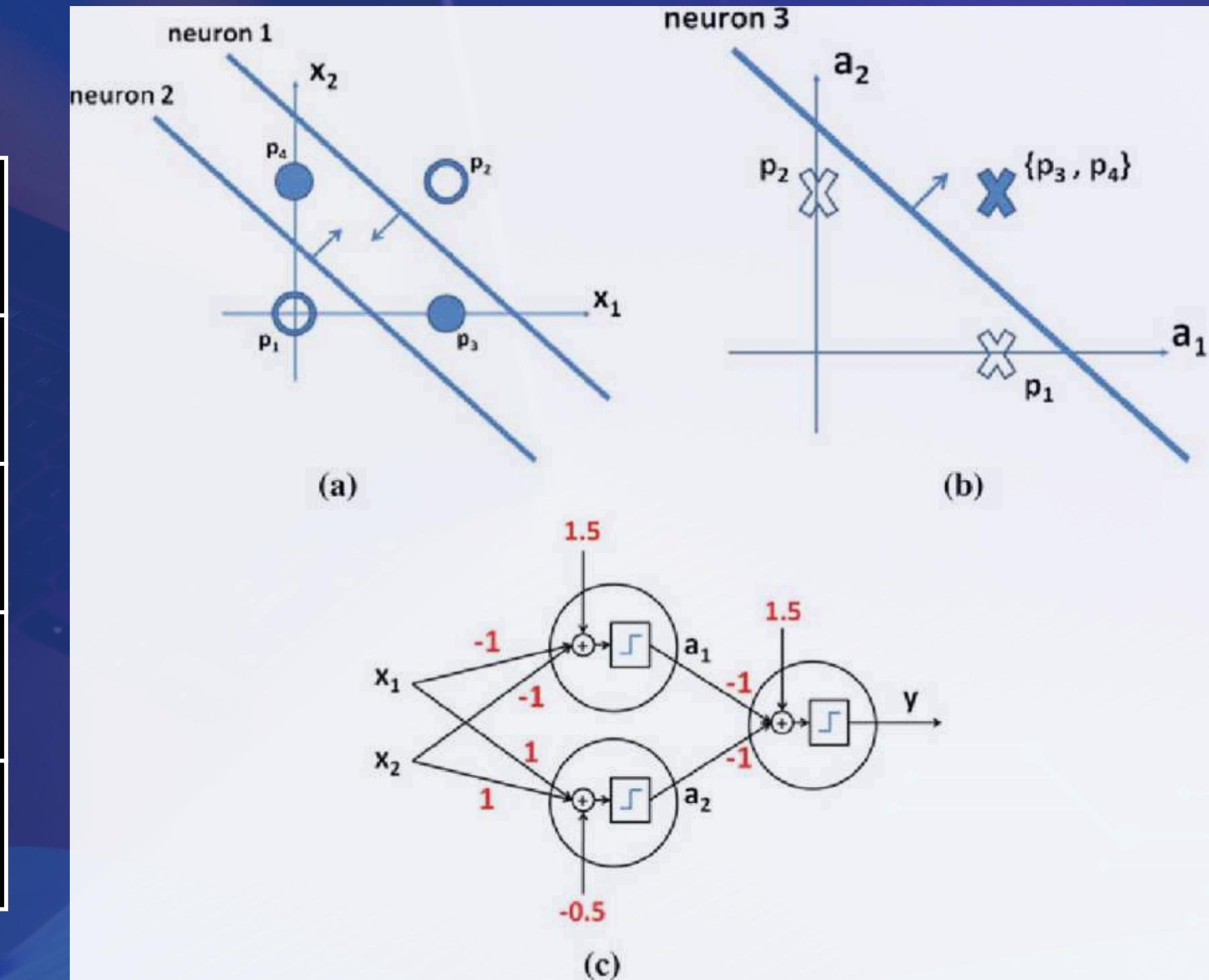
Ejemplo Forward Propagation

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



Ejemplo Forward Propagation

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





Ejemplo Forward Propagation

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

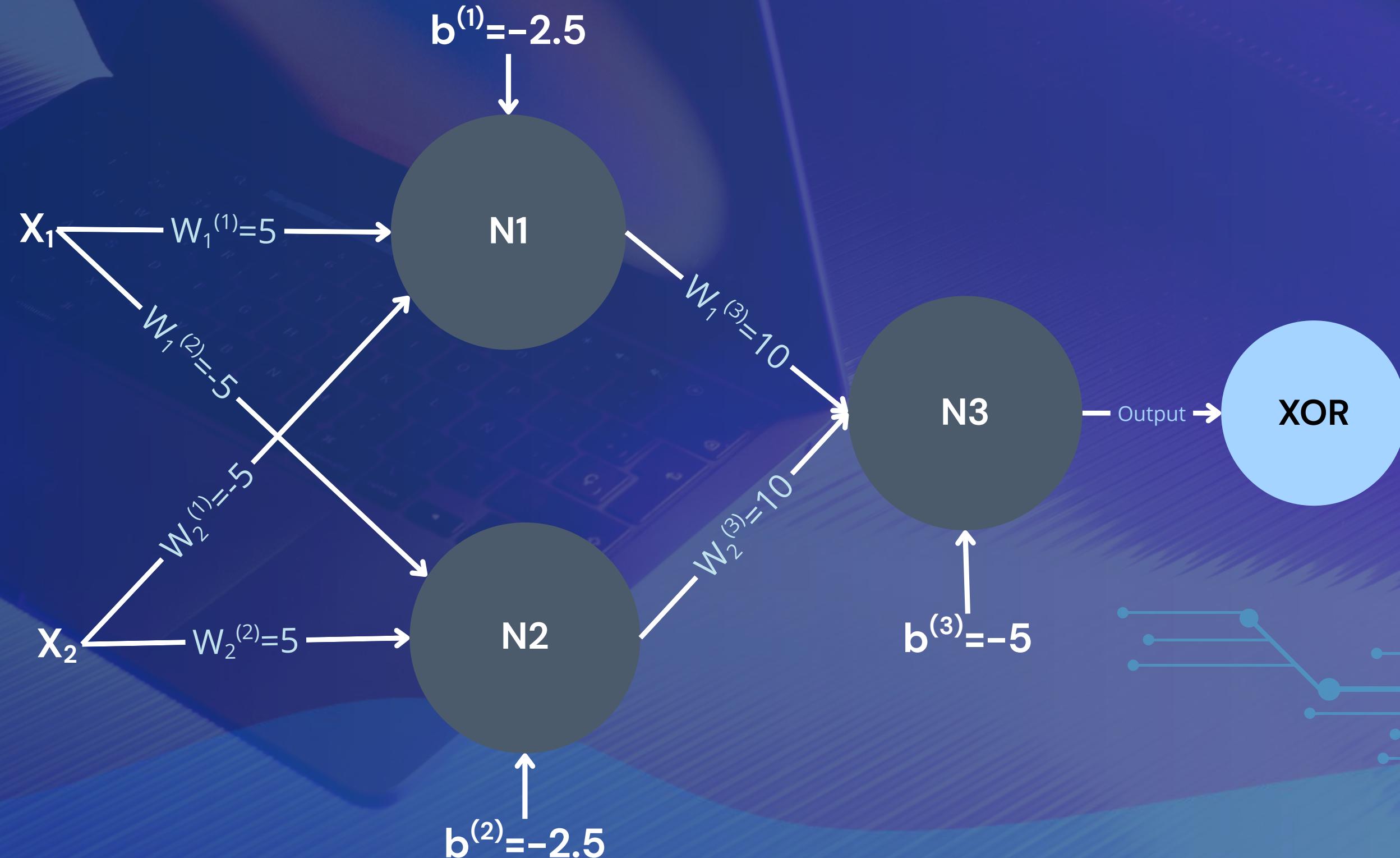
- Arquitectura del MLP:
 - Capas: 2 capas (1 oculta, 1 salida).
 - Neuronas: 2 en la capa oculta, 1 en la salida.
- Funciones de activación:
 - Oculta: Sigmoide ($\sigma(z) = 1 / (1 + e^{-z})$).
 - Salida: Sigmoide.



Ejemplo Forward Propagation

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

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





Ejemplo Forward Propagation

Ejemplo	x_1	x_2	N_1	N_2	N_3 (Salida)	y
1	0	0	0.0759	0.0759	0.0298	0
2	0	1	0.00055	0.9241	0.9858	1
3	1	0	0.9241	0.00055	0.9858	1
4	1	1	0.0759	0.0759	0.0298	0

Ejemplo Loss Function

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

- Arquitectura del MLP:
 - Capas: 2 capas (1 oculta, 1 salida).
 - Neuronas: 2 en la capa oculta, 1 en la salida.
- Funciones de activación:
 - Oculta: Sigmoide ($\sigma(z) = 1 / (1 + e^{-z})$).
 - Salida: Sigmoide.
- Función de costo:
 - Error cuadrático medio (MSE).
Simplicidad didáctica.

Ejemplo Loss Function

Concepto	¿Qué hace?	Ecuación
Función de pérdida (Loss Function)	Mide el error en un solo ejemplo.	$L=1/2 (a^{(l)}-y)^2$ para cada ejemplo
Función de costo (Cost Function)	Mide el promedio del error sobre todo el conjunto de entrenamiento.	$J=1/N \sum(L)$



Ejemplo Loss Function

$$\mathcal{L}(y^{(i)}, a^{(3)(i)}) = \frac{1}{2}(a^{(3)(i)} - y^{(i)})^2$$

Ejemplo	x_1	x_2	Predictión	y	Función de Pérdida
1	0	0	0.0298	0	$\frac{1}{2}(0.0298 - 0)^2 \approx 0.000444$
2	0	1	0.9858	1	$\frac{1}{2}(0.9858 - 1)^2 \approx 0.000101$
3	1	0	0.9858	1	$\frac{1}{2}(0.9858 - 1)^2 \approx 0.000101$
4	1	1	0.0298	0	$\frac{1}{2}(0.0298 - 0)^2 \approx 0.000444$





Ejemplo Loss Function

Calcular costo total para la época 1

$$J = \frac{1}{2N} \sum_{i=1}^N (a^{(i)} - y^{(i)})^2$$

$$\begin{aligned} & (1/2^4)(0.000888 + 0.0002016 + 0.0002016 + 0.000888) \\ &= 1/8 * 0.0021792 \\ &= 0.0002724 \end{aligned}$$





But what is a neural network? | Deep learning chapter 1

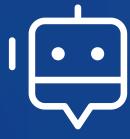


Share



Watch on





Próxima sesión:

Deep Learning

- *Fundamentos*
- *Redes Neuronales*
 - *Backpropagation*
 - *Arquitecturas*
 - *Tipos*