

# Introducción a ML – Sesión 2

Cesar Garcia

2025

# Objetivos

- Comprender neuronas artificiales.
- Conocer pesos, bias (sesgo), activaciones.
- Entender flujo de información (forward propagation).
- Familiarizarse con funciones de activación comunes.

# ¿Qué es una Red Neuronal?

Una red neuronal es una función compuesta por varias transformaciones simples:

## 1. Capas

- **Capa de entrada:** recibe las características del problema.
- **Capas ocultas:** combinan pesos, sesgos y activaciones para aprender patrones.
- **Capa de salida:** produce la predicción final.

## 2. Neuronas

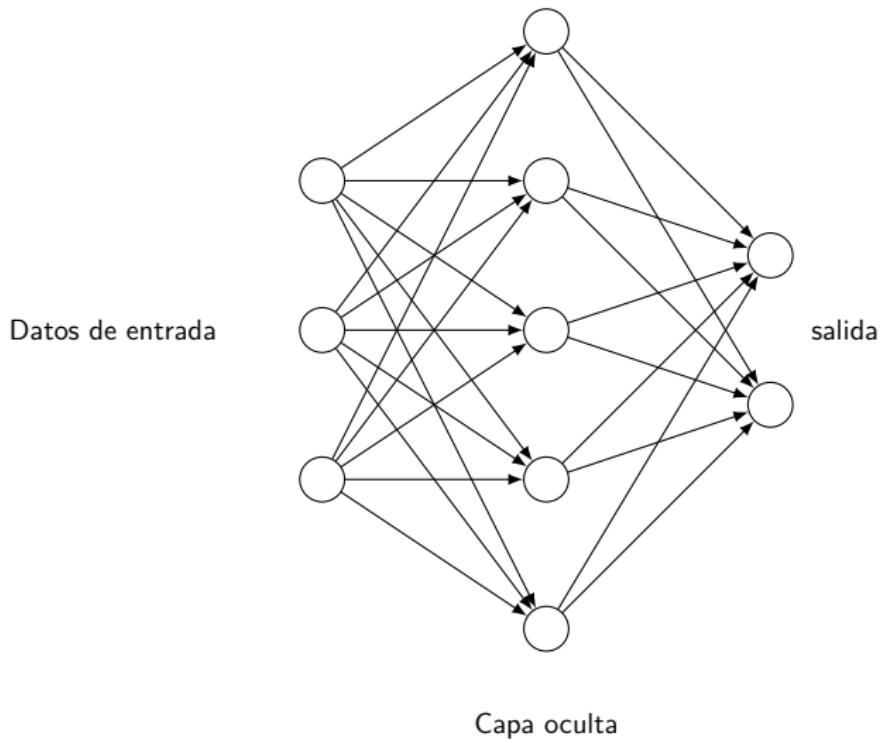
Cada neurona realiza:

$$z = Wx + b \quad \Rightarrow \quad a = f(z)$$

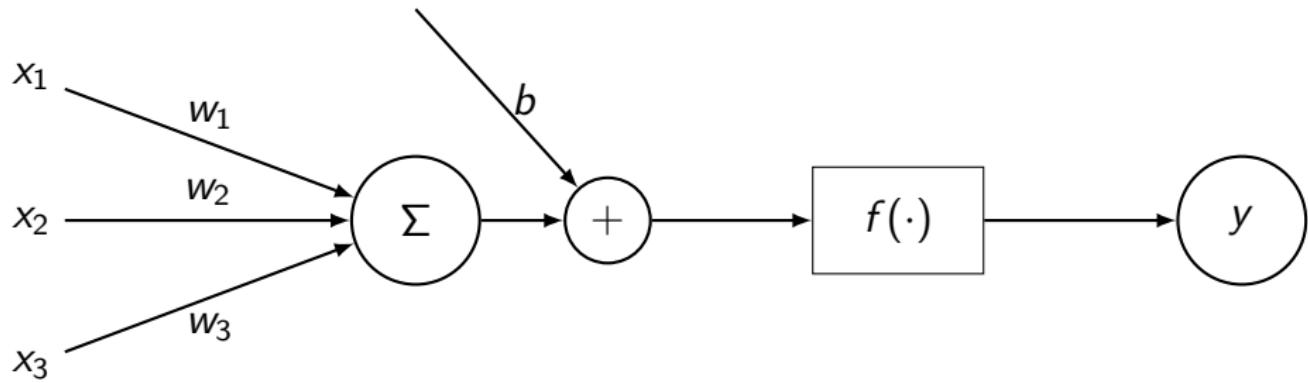
## 3. Conexiones y aprendizaje

- Pesos que conectan una capa con la siguiente.
- La red ajusta estos pesos durante el entrenamiento para minimizar el error.

# Red Neural



# Perceptron



$$y = f\left(\sum_{i=1}^n w_i x_i + b\right)$$

# Todo es Algebra Lineal

$$z = Wx + b$$

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \end{bmatrix}_{5 \times 3}$$
$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{3 \times 1}$$
$$+ \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}_{5 \times 1}$$

# Forward Propagation

## Propagación hacia adelante (Forward Propagation)

- Tomamos las entradas ( $x$ ) del modelo.
- En cada capa calculamos una combinación lineal: ( $z = Wx + b$ ).
- Aplicamos una función de activación: ( $a = f(z)$ ).
- Repetimos capa por capa hasta obtener la salida ( $\hat{y}$ ).

### Ejemplo: red con 1 capa oculta

$$z^{(1)} = W^{(1)}x + b^{(1)}$$

$$a^{(1)} = f(z^{(1)})$$

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)}$$

$$\hat{y} = g(z^{(2)})$$

## ¿Qué es una función de activación?

- Es una transformación no lineal aplicada después de la combinación lineal  $z = Wx + b$ .
- Permite que la red neuronal aprenda funciones complejas, no sólo líneas rectas.
- Sin activaciones no lineales, toda la red sería equivalente a una sola transformación lineal.
- Diferentes activaciones producen comportamientos distintos en el aprendizaje.

## ¿Por qué necesitamos no linealidad?

### Idea clave

- Cada capa lineal realiza una transformación del tipo  $z = Wx + b$ .
- Si encadenamos varias capas solo lineales (sin función de activación), la composición sigue siendo **otra transformación lineal**.
- Eso significa que una “red profunda” sin activaciones no lineales sería equivalente a **una sola capa lineal**.
- No podríamos aproximar funciones complejas ni separar datos con fronteras no lineales.

¿Por qué necesitamos no linealidad?

## Viéndolo del punto de vista Algebraico

$$z^{(1)} = W^{(1)}x + b^{(1)}$$

$$z^{(2)} = W^{(2)}z^{(1)} + b^{(2)}$$

$$= W^{(2)}(W^{(1)}x + b^{(1)}) + b^{(2)}$$

$$= (W^{(2)}W^{(1)})x + (W^{(2)}b^{(1)} + b^{(2)})$$

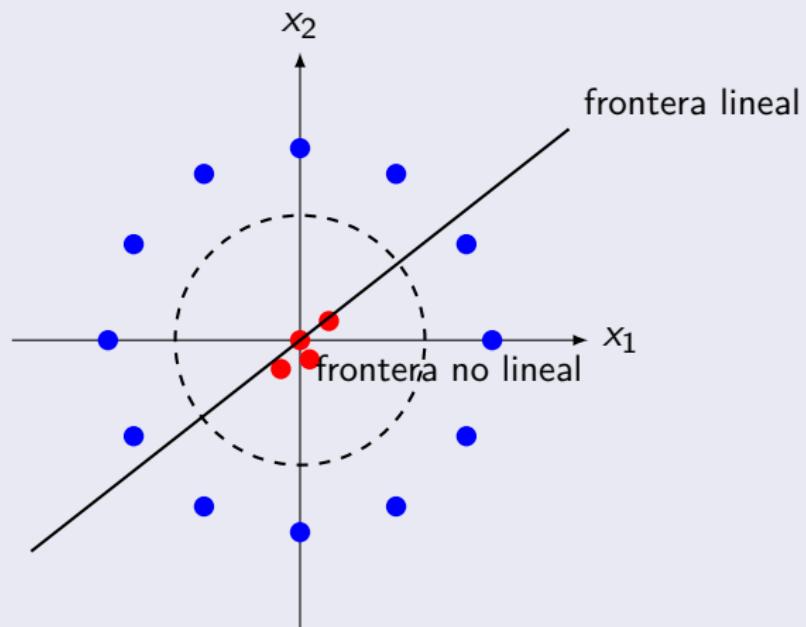
Sigue siendo de la forma:

$$z^{(2)} = W^*x + b^*$$

es decir, **lineal**.

# ¿Por qué necesitamos no linealidad?

## Visualización



Una recta no puede separar las clases,  
pero una frontera curva (no lineal) sí.

# Teorema de Aproximación Universal

Un perceptrón multicapa (MLP) con **una sola capa oculta**, pero con suficientes neuronas, puede aproximar **cualquier función continua** en un conjunto compacto. Esto requiere una función de activación **no lineal** (ReLU, Sigmoid, Tanh, etc.).

## Enunciado formal

*Para toda función continua  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ; y para todo  $\varepsilon > 0$ , existe una red neuronal con **una capa oculta** y una activación no lineal  $f(\cdot)$  tal que la red aproxima a  $f$  con un error menor que  $\varepsilon$ . en todo el dominio compacto.*

- En teoría, **no necesitamos muchas capas** para representar funciones complejas.
- En la práctica, las redes profundas aprenden **mejor, más rápido y con menos neuronas** que una red extremadamente ancha de una sola capa.

# Funciones de Activación

## Funciones de activación comunes

- ReLU
- Sigmoid
- Tanh
- GELU

## Otras populares (incluidas para referencia)

- Leaky ReLU
- ELU
- Swish (SiLU)

## Popular para salidas

- Softmax

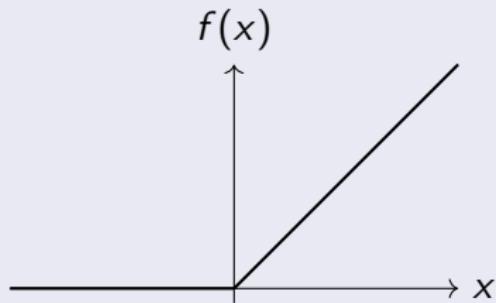
# Función de activación: ReLU

## Rectified Linear Unit (ReLU)

- Deja pasar solo valores positivos.
- Muy usada en redes profundas por su simplicidad y buen comportamiento en la práctica.
- Problema: neuronas pueden “morir” si la entrada es siempre negativa.

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

## Visualización



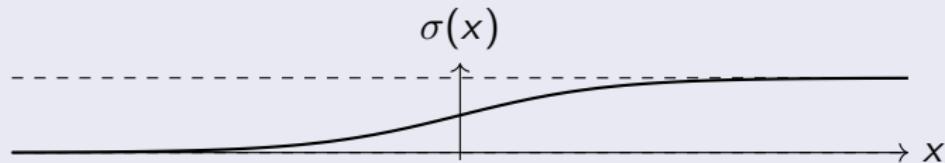
# Función de activación: Sigmoid

## Sigmoid / logística

- Comprime cualquier valor real al intervalo ((0, 1)).
- Interpretación probabilística: salida como “probabilidad”.
- Problema: gradientes muy pequeños para valores grandes en magnitud (saturación).

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

## Visualización



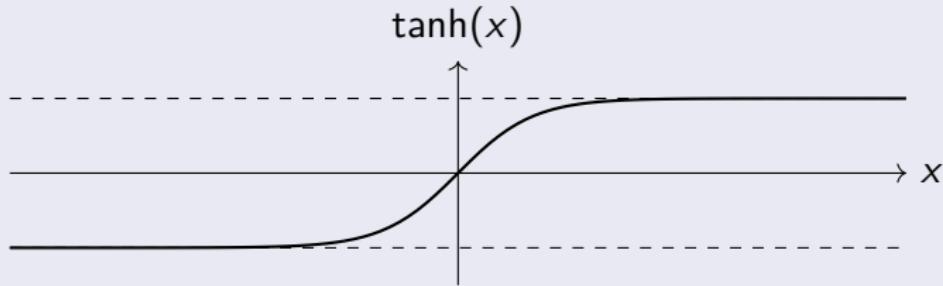
# Función de activación: Tanh

## Tangente hiperbólica

- Similar a la sigmoide pero centrada en 0.
- Rango en  $((-1, 1))$ .
- Útil cuando queremos activaciones con media cercana a cero.

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

## Visualización



# Función de activación: GELU

## Gaussian Error Linear Unit (GELU)

- Suaviza la ReLU usando una distribución normal.
- Muy popular en modelos modernos (por ejemplo, Transformers).
- Activa más fuertemente valores positivos grandes, pero de forma suave.

Definición conceptual:

$$\text{GELU}(x) = x \Phi(x)$$

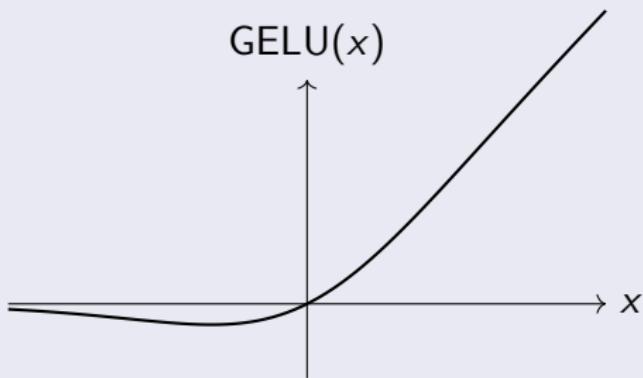
donde  $\Phi(x)$  es la CDF de la normal estándar.

Aproximación práctica usada en la mayoría de implementaciones:

$$\text{GELU}(x) \approx 0.5 x \left( 1 + \tanh \left( \sqrt{2/\pi} (x + 0.044715 x^3) \right) \right)$$

# Función de activación: GELU

## Visualización



# Función de activación: Softmax

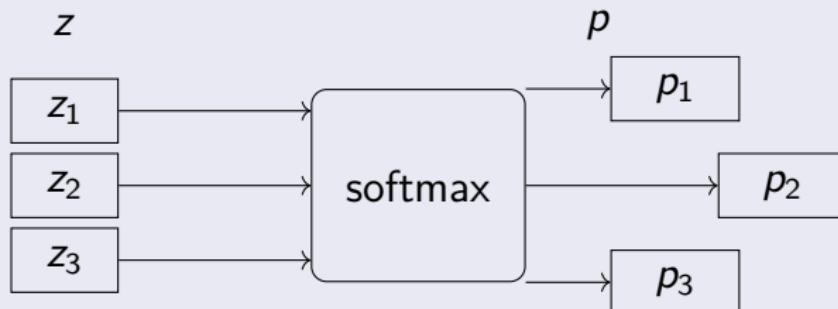
## Softmax

- Convierte un vector de scores en un vector de probabilidades.
- Muy usada en la capa de salida para clasificación multiclas.
- La suma de las salidas es 1.

Para una salida de dimensión  $K$ :

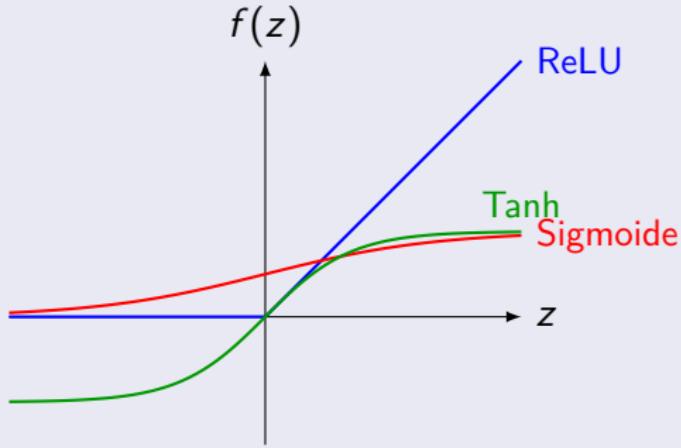
$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}, \quad i = 1, \dots, K$$

## Visualización



# Funciones de Activación – Comparación

## Visualización



# Función de activación: Leaky ReLU

## Leaky Rectified Linear Unit

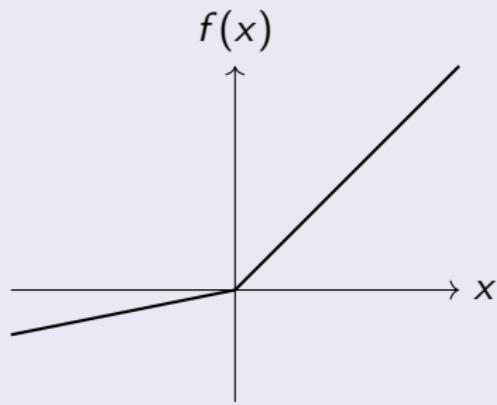
- Variante de ReLU que permite un pequeño gradiente cuando  $x < 0$ .
- Evita el problema de “neuronas muertas” (pesos que nunca se actualizan).
- Parámetro  $\alpha$  controla la pendiente en la parte negativa (típicamente  $\alpha = 0.01$ ).

$$\text{LeakyReLU}(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$

con  $\alpha > 0$  pequeño.

# Función de activación: Leaky ReLU

## Visualización



## Exponential Linear Unit (ELU)

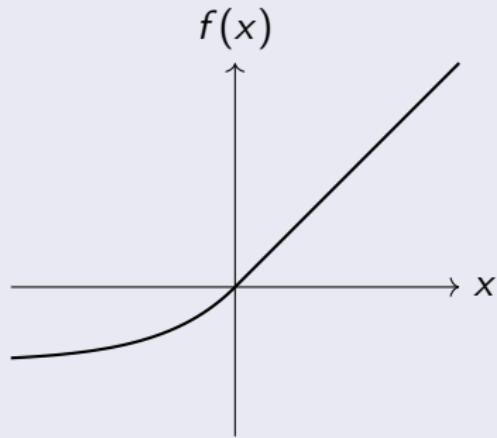
- Similar a ReLU para  $x \geq 0$ , pero suave en la parte negativa.
- Produce activaciones negativas que ayudan a centrar la salida alrededor de 0.
- Reduce la “dead zone” de ReLU, manteniendo gradientes en  $x < 0$ .

$$\text{ELU}(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

Típicamente  $\alpha = 1$ .

# Función de activación: ELU

## Visualización



# Función de activación: Swish (SiLU)

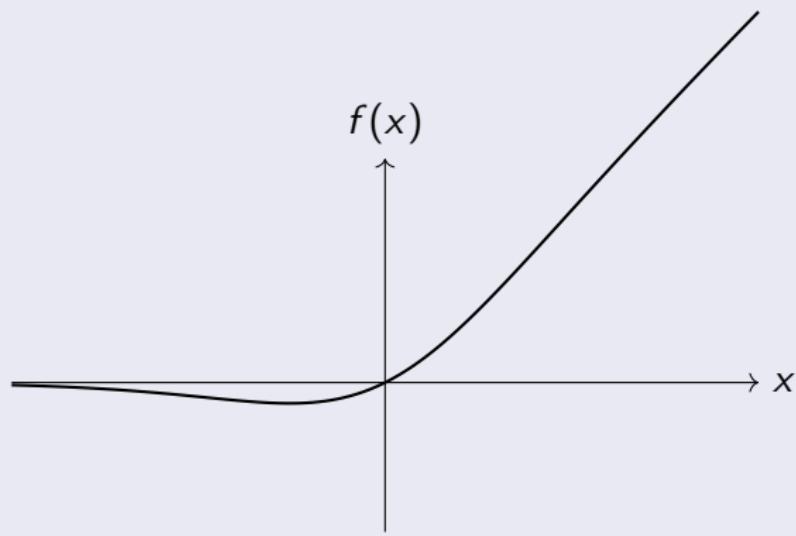
## Swish - Imagen / SiLU - Sigmoid Linear Unit

- Activación moderna usada en redes eficientes (por ejemplo, EfficientNet).
- Suaviza la ReLU y la hace “auto-puerta”: la propia entrada decide cuánto se deja pasar.
- Similar a GELU en comportamiento: valores positivos grandes pasan casi linealmente, los negativos se atenúan suavemente.

$$\text{Swish}(x) = x \sigma(x) \quad \text{donde} \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

# Función de activación: Swish (SiLU)

## Visualización



# Tensorflow Playground

Tinker With a **Neural Network** Right Here in Your Browser.  
Don't Worry, You Can't Break It. We Promise.

