PRESENTACIÓN 3: REDES NEURONALES

Machine Learning II

Jose Alexander Fuentes Montoya Ph.D(c)

REDES NEURONALES

OBJETIVOS

Después de esta presentación, el estudiante será capaz de:

- Comprender el Perceptrón de Capa Única (Single-Layer Perceptron)
- Entender el problema XOR
- Conocer distintas funciones de activación

- Profundizar en el concepto, algoritmo e implementación de Perceptrón Multicapa (Multi-layer Perceptron, MLP)
- Ver cómo el Perceptrón Multicapa resuelve el problema XOR
- Aprender el algoritmo de retropropagación (Backpropagation)

INTRODUCCIÓN

Nuestro cerebro recibe señales a través de las neuronas, las procesa y produce respuestas. Por lo general, los receptores envían información a las neuronas, que llega al cerebro. El cerebro procesa y devuelve la respuesta a los efectores. Este concepto fue expuesto por Cajal [1]. Aunque las neuronas son más lentas que los *logic gates*, su cantidad permite procesar de forma rápida.

La **Figura 3-1** muestra la estructura de una neurona:
- **Dendritas**: zonas receptoras
- **Cuerpo celular**: procesa las entradas
- **Axón**: transmite las señales

Las neuronas se conectan entre sí mediante **sinapsis**.

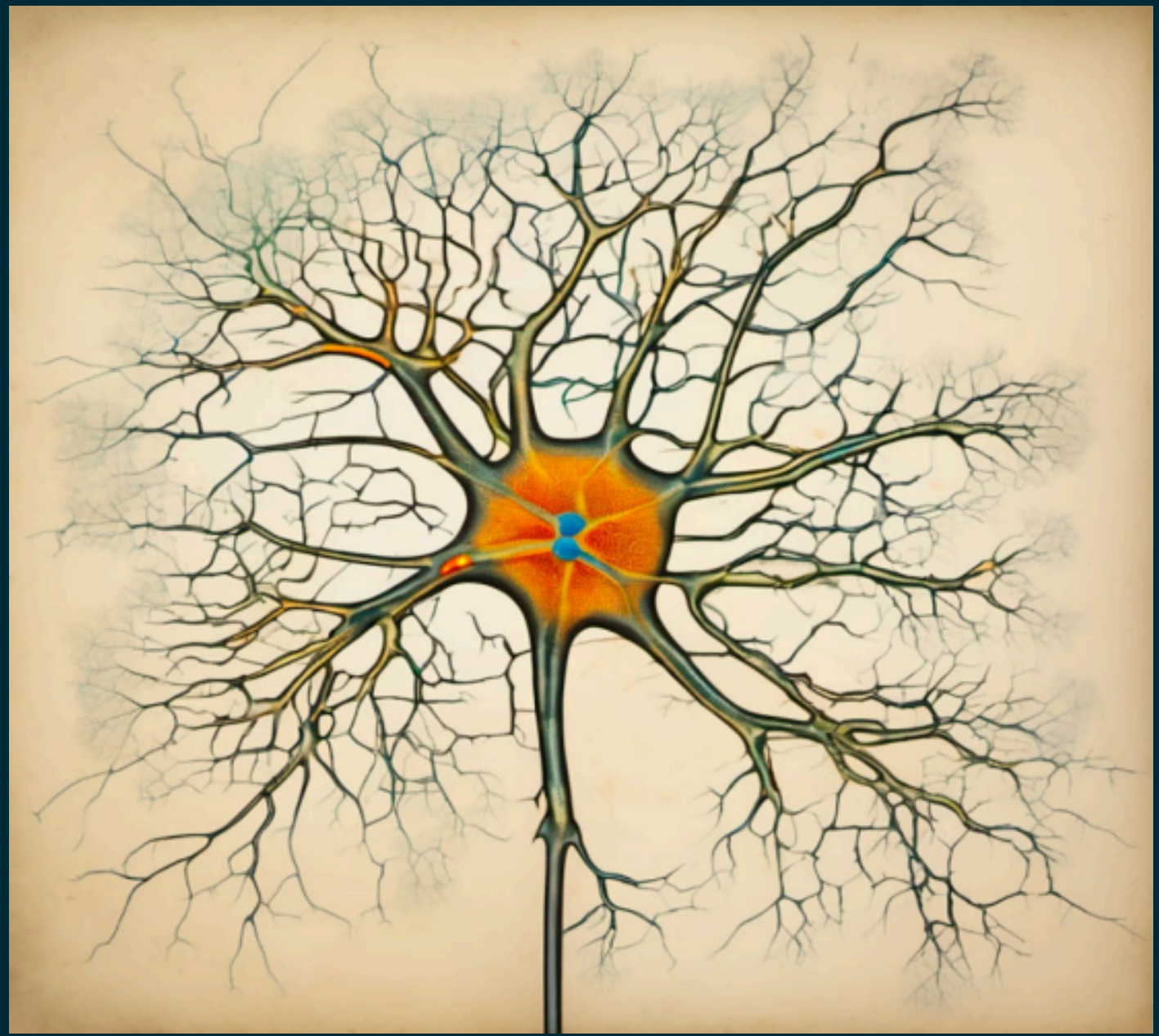


Figura 3-1. Imagen de una neurona generada con IA (https://pixlr.com/image-generator/)

Un **modelo computacional** similar se ve en la **Figura 3-2**, que recibe una entrada bidimensional y clasifica en dos clases.
- Recibe entradas (X_1, X_2).
- Las multiplica por pesos (W_1, W_2).
- Suma los resultados y lo pasa a una función.
- Si es mayor que cierto umbral, la salida es 1; si no, 0.

Este modelo actúa como **clasificador binario**.

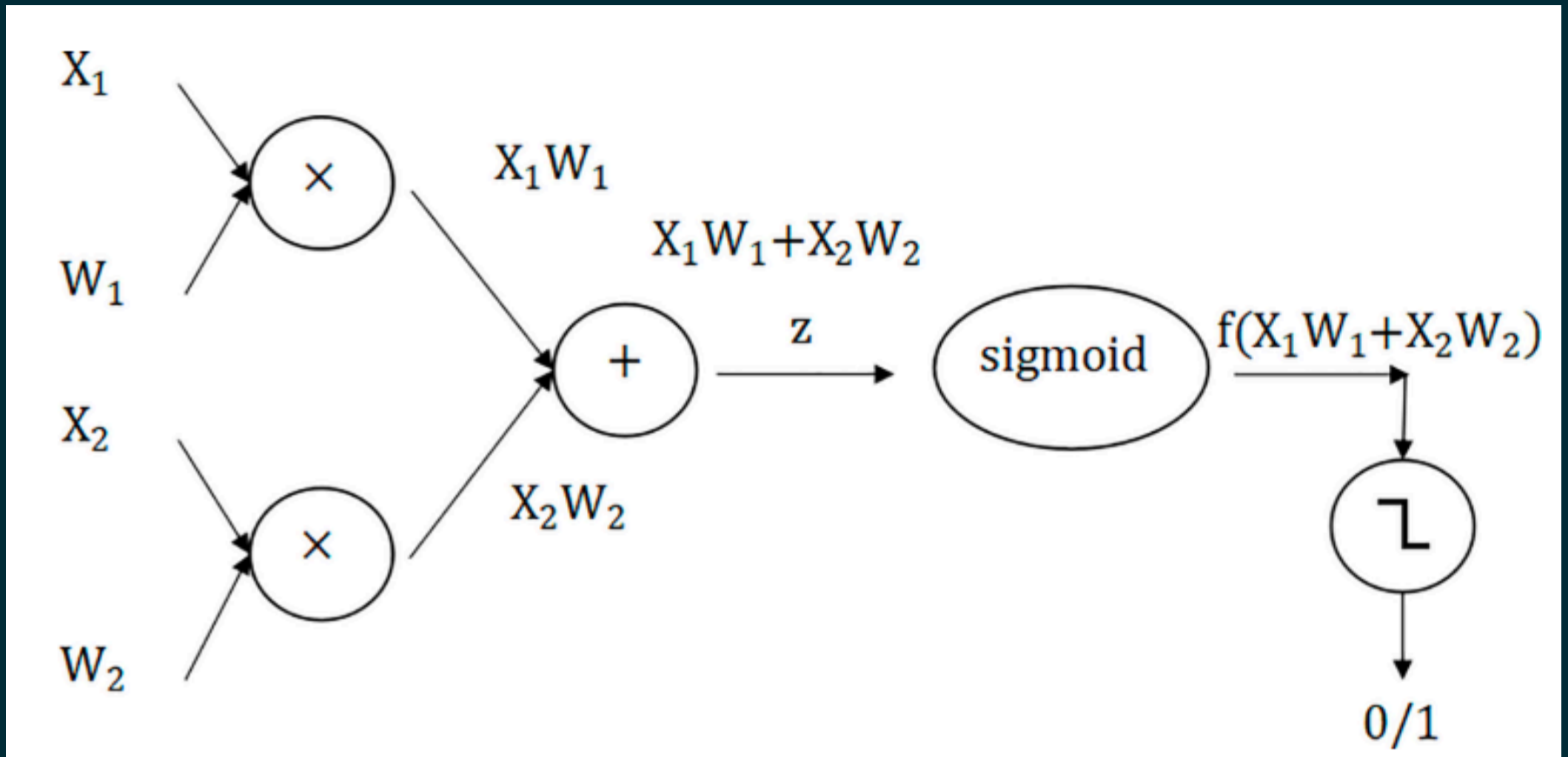


Figura 3-2. Modelo computacional basado en la estructura de una neurona

SLP (Single-Layer Perceptron) puede extenderse a d entradas. Destacamos:

- Núm. neuronas en capa de entrada = núm. entradas
- Pesos = "importancia" de cada entrada
- La combinación lineal se pasa por una **función de activación**
- Luego, se compara con el umbral (para clasificación binaria)

El **Rosenblatt Perceptron** (por Frank Rosenblatt [2]) es una versión con entradas/pesos continuos. El **McCulloch & Pitts** es uno con entradas/pesos binarios (excitatorios/inhibitorios), útil para implementar *logic gates* linealmente separables.

PERCEPTRÓN DE CAPA ÚNICA (SLP)

El **Single-Layer Perceptron** es un **clasificador lineal** que separa dos clases con una línea (2D), un plano (3D) o un hiperplano (más dimensiones). **No** puede clasificar datos no linealmente separables. Veamos cómo funciona:

1. Se multiplican las entradas X por los pesos W y se suma un **bias** b.

$$U = \sum_i (W_i \cdot X_i) \; + \; b$$

2. Se aplica una **función de activación** f:

$$V_i = f(U_i)$$

3. Por ejemplo, con función sigmoide,

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

4. Según un umbral, decidimos salida 1 o 0 (en clasificación). Para regresión, no habría ese paso.

Los pesos se inicializan aleatoriamente y se ajustan por cada iteración.

Algoritmo SLP resumido:

1. Inicializar pesos W y bias b aleatoriamente.
2. Para cada muestra X_i:
   - Calcular $U_i = \sum(W_i X_i) + b$.
   - Calcular salida $\hat{y} = f(U_i)$.
   - Actualizar W y b con la regla:

$$W \leftarrow W - \alpha f(1-f)\big(\hat{y} - y\big)X$$

$$b \leftarrow b - \alpha f(1-f)\big(\hat{y} - y\big)$$

3. Repetir hasta convergencia o hasta agotar iteraciones.

(Donde α es la tasa de aprendizaje.)

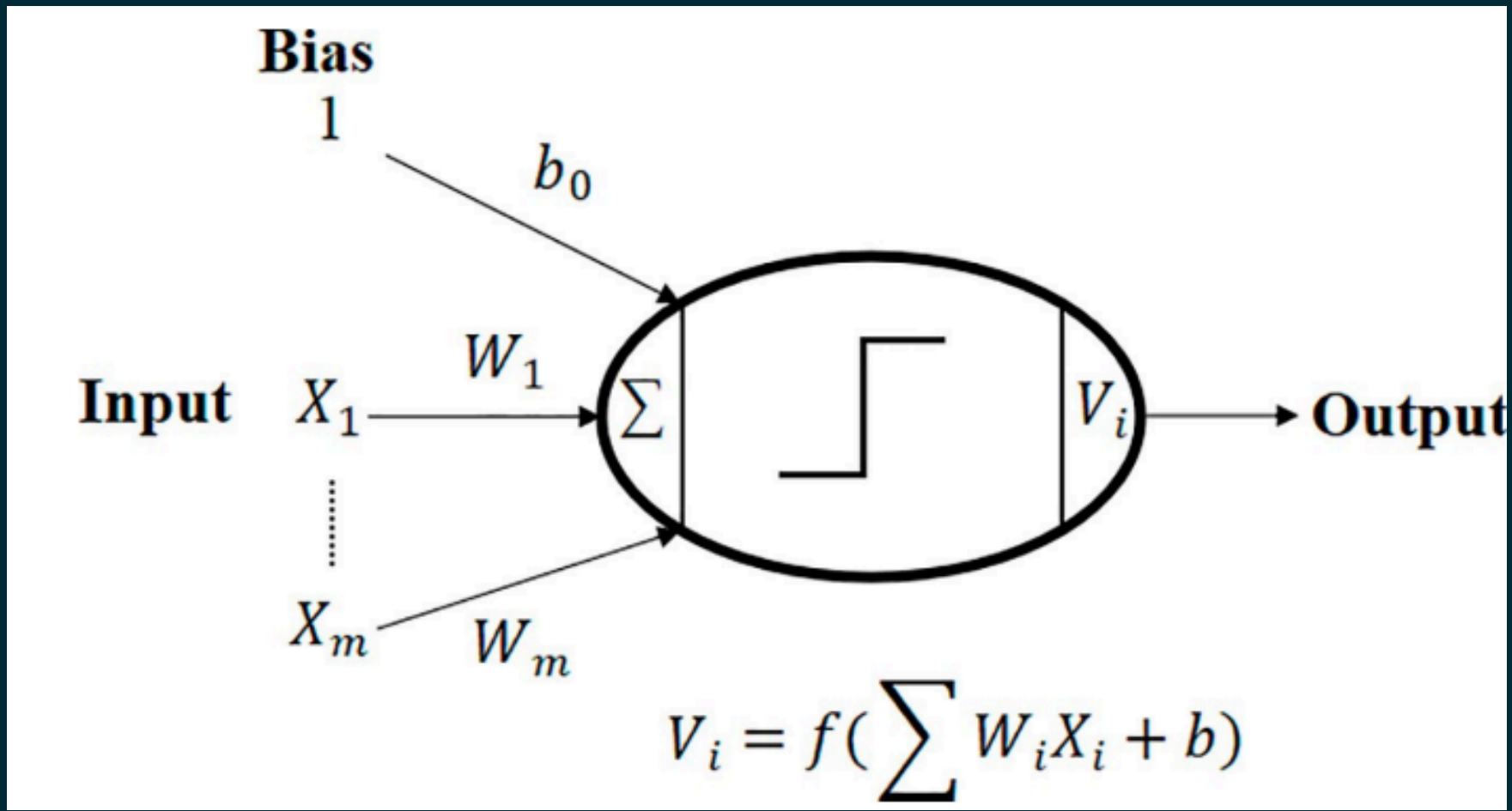


Figura 3-3. Single-Layer Perceptron model

IMPLEMENTACIÓN DE UN SLP

A continuación se muestra un enlace para ver un **ejemplo en Python** que clasifica IRIS (solo 2 clases) implementando SLP *from scratch*:

Ver código en Google Colab

(El IRIS se limita a 100 muestras para binario. Se normalizan datos, se usa sigmoide, se entrena y se evalúa.)

Luego, se analiza la influencia de α (learning rate). Se observa que la precisión varía según α.

PERCEPTRÓN CON SKLEARN

De forma más directa, `sklearn.linear_model.Perceptron` implementa un SLP. Por ejemplo, se puede usar en el dataset de **Breast Cancer** (569 muestras, 30 rasgos). Tabla 3-1 resume las funciones claves:

Función	Descripción
`Perceptron()`	Inicializa el algoritmo.
`perceptron.fit(X_train, y_train)`	Ajusta/entrena el modelo con el train set.
`perceptron.predict(X_test)`	Predice la clase para cada muestra en X_test.
`accuracy_score(y_test, y_pred)`	Calcula la exactitud del modelo.

(Se pueden ver detalles en la documentación de sklearn.)

SLP CON KERAS

Para implementar SLP con Keras, se crea un **modelo secuencial** con una sola capa densa y una neurona de salida.
Ejemplo en Breast Cancer:

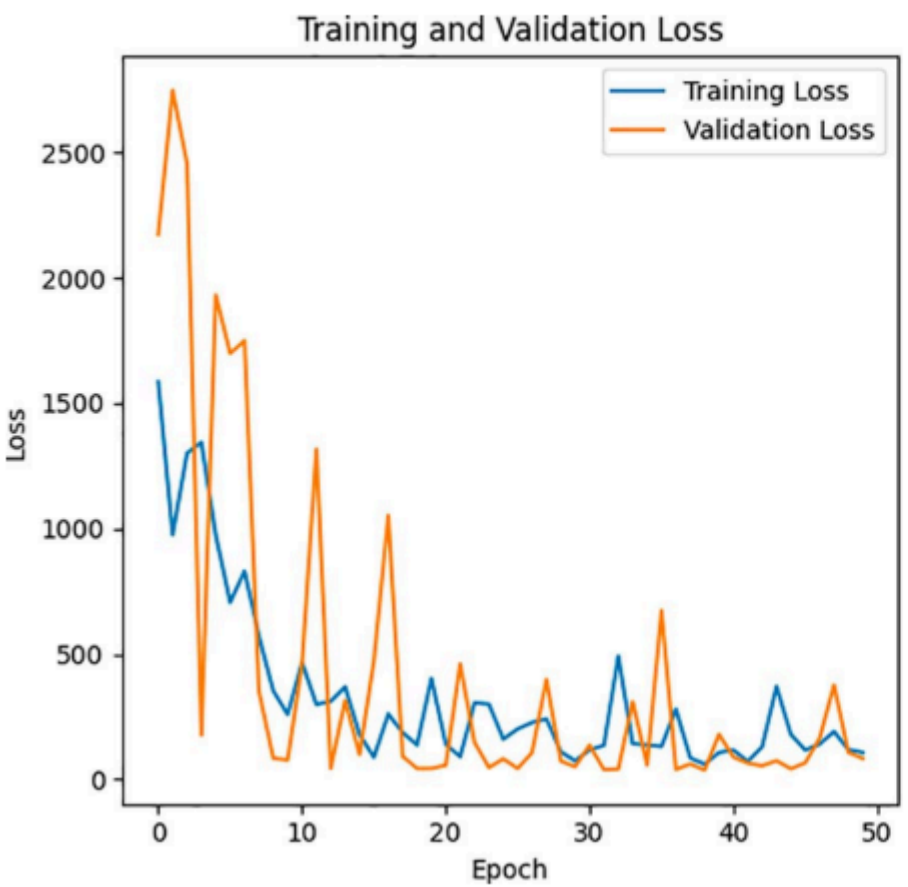
Ver código en Google Colab

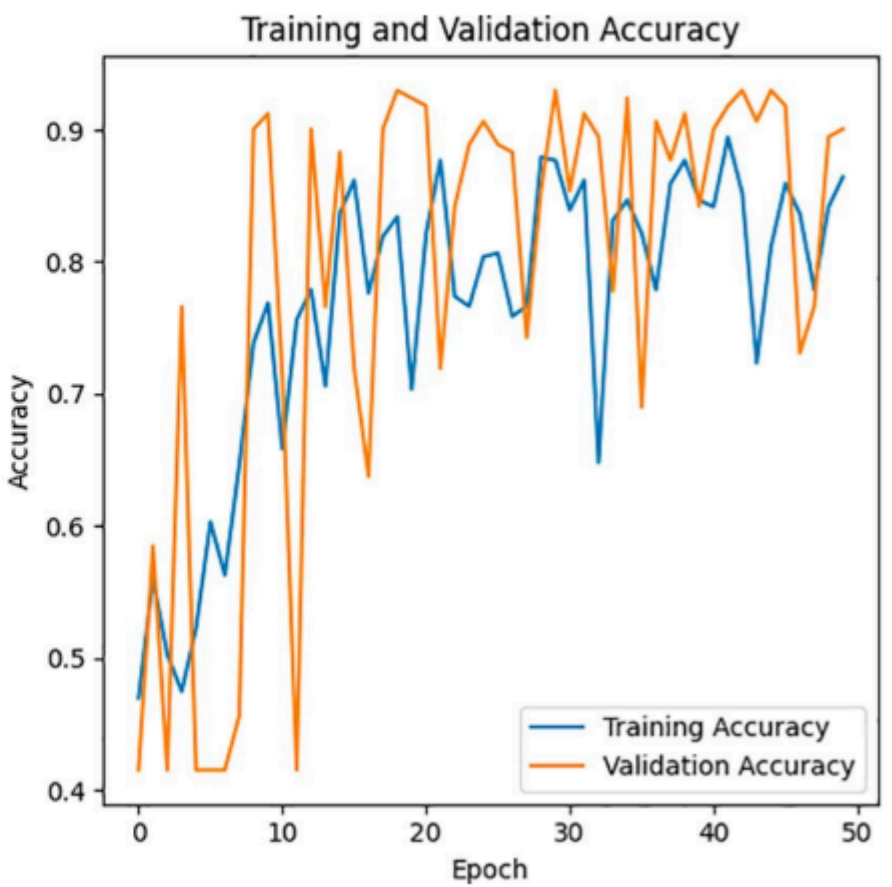
Ver código en Google Colab

1. Usamos `Dense(units=1, input_dim=X.shape[1], activation='sigmoid')`.
2. Compilamos con `optimizer='sgd'`, `loss='binary_crossentropy'`, `metrics=['accuracy']`.
3. Entrenamos y graficamos `loss` y `accuracy` en train y test.

La Figura 3-5 muestra cómo la pérdida baja y la precisión sube con las épocas.

Output:

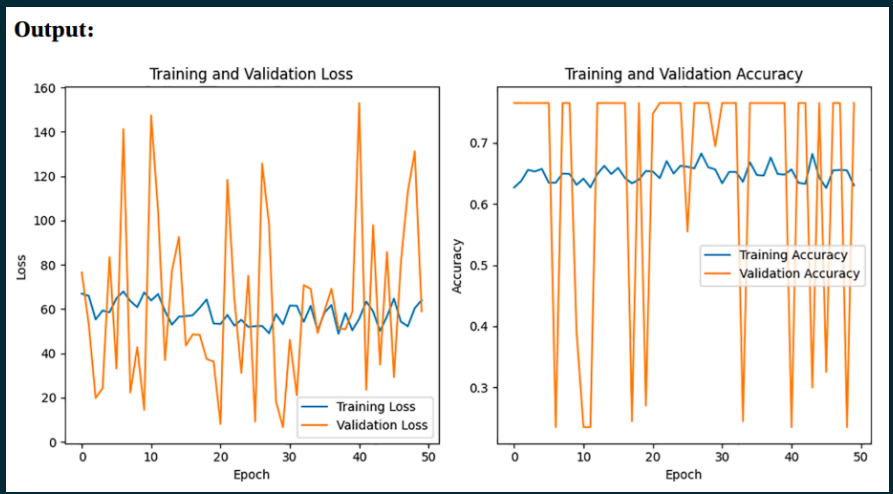




EJEMPLO ADICIONAL CON MYOCARDIAL INFARCTION COMPLICATIONS

Se tienen 1700 muestras y 109 características (tras preprocesar). Misma arquitectura y entrenamiento. La Figura 3-6 muestra la evolución de pérdida y exactitud. Resultados en la tabla 3-2.

Ver código en Google Colab



SLP No.	Dataset	Modelo (neurona salida)	Accuracy	Loss
1.	Breast Cancer	`slp model_1`	0.907	84.899
2.	Myocardial Infarction Complications	`slp model_1`	0.7697	57.6161

PROBLEMA XOR

La clásica tabla XOR:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

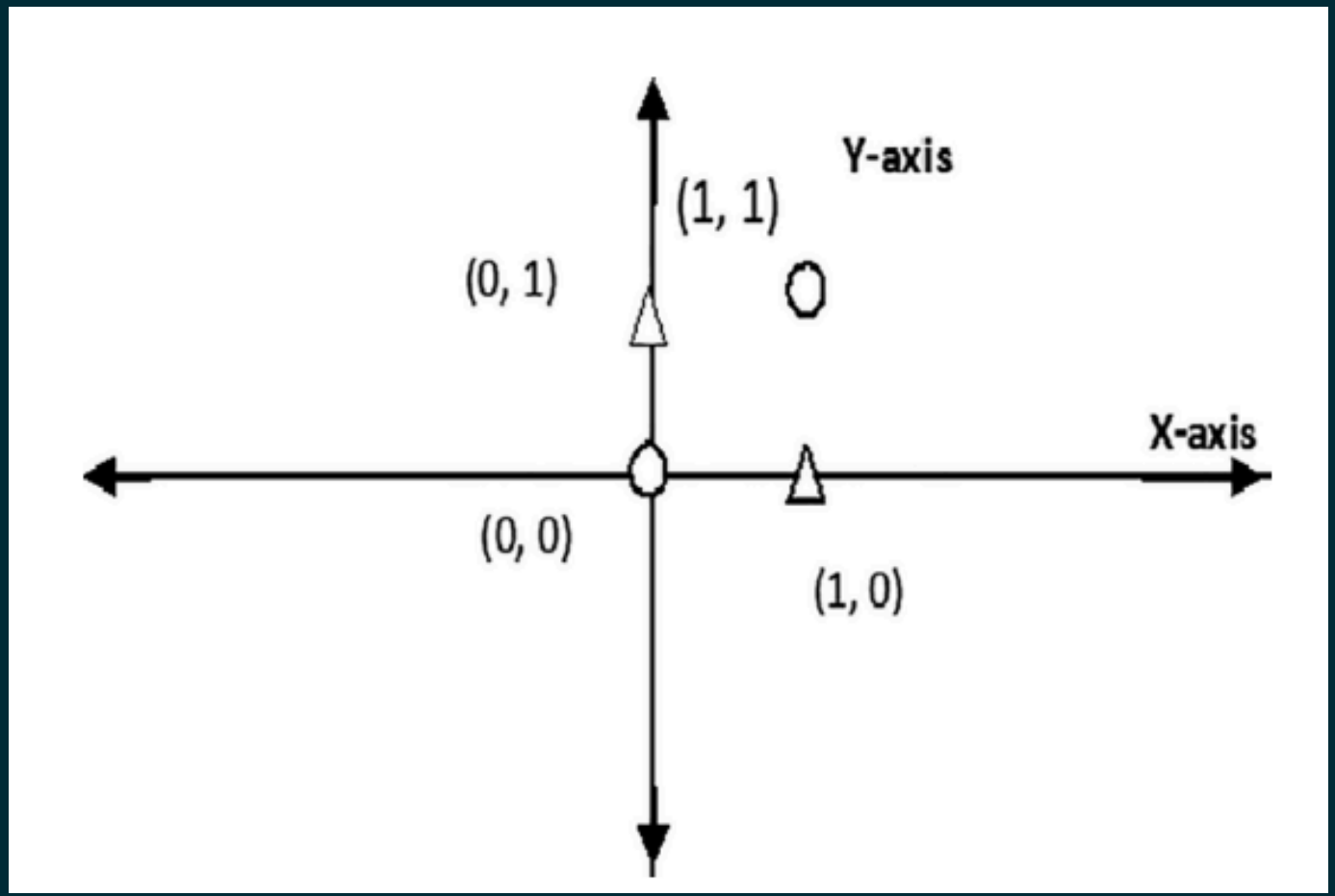


Figura 3-7. Problema XOR)

En 2D, **Y=0** (círculo) y **Y=1** (triángulo) no se separan con una **línea**. SLP no logra clasificarlo porque no es linealmente separable. El **MLP** (Multi-layer Perceptron) sí lo consigue.

FUNCIONES DE ACTIVACIÓN

Las funciones de activación transforman la suma ponderada en la neurona. Las más comunes:

1. **Sigmoide**

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

- Salida $(0, 1)$.
- Ideal para probabilidades.
- Sufre *vanishing gradient* en redes profundas.

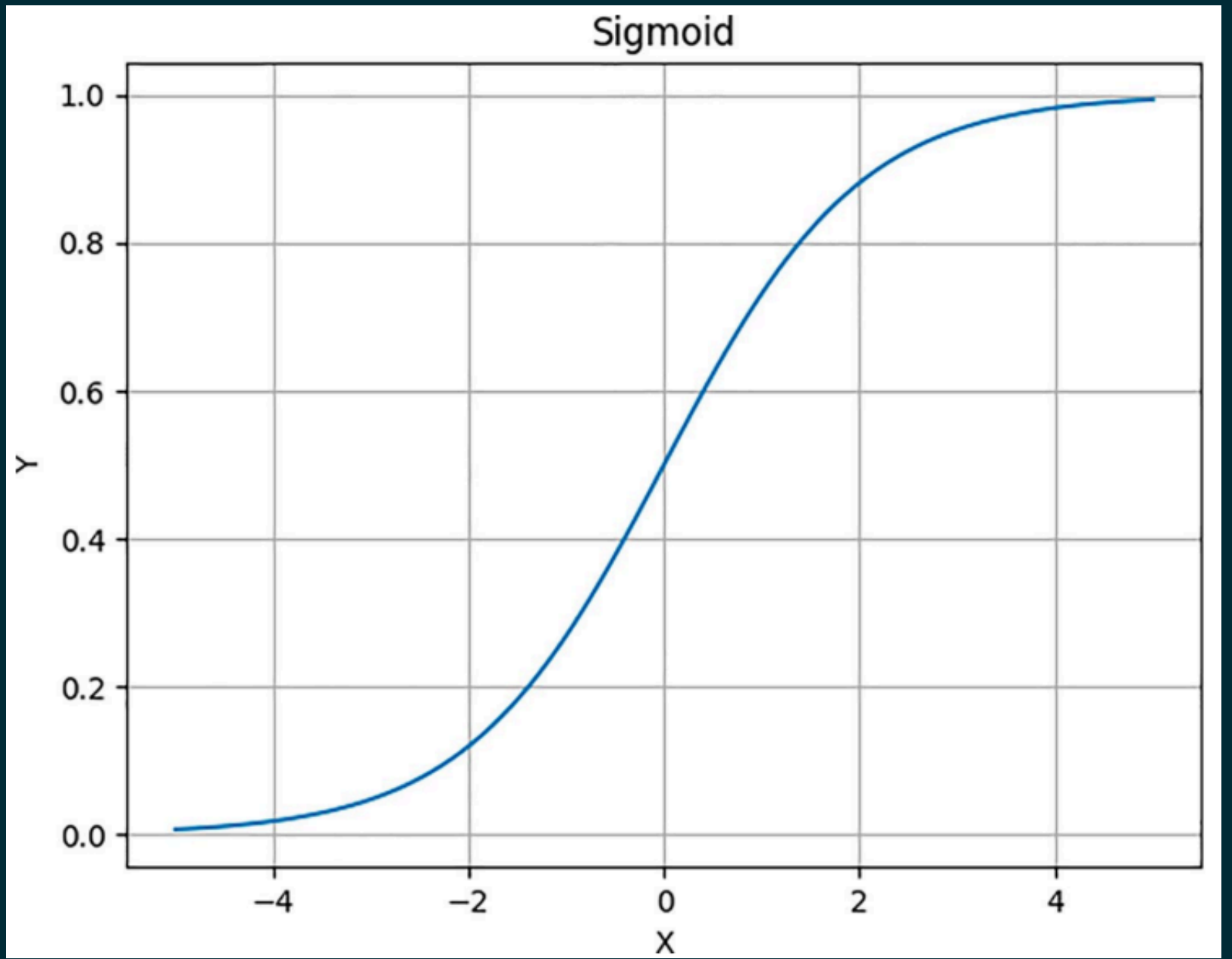


Figura 3-8. Función Sigmoide)

2. **tanh**

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

- Rango $(-1, 1)$.
- Zero-centered.
- También padece *vanishing gradient*.

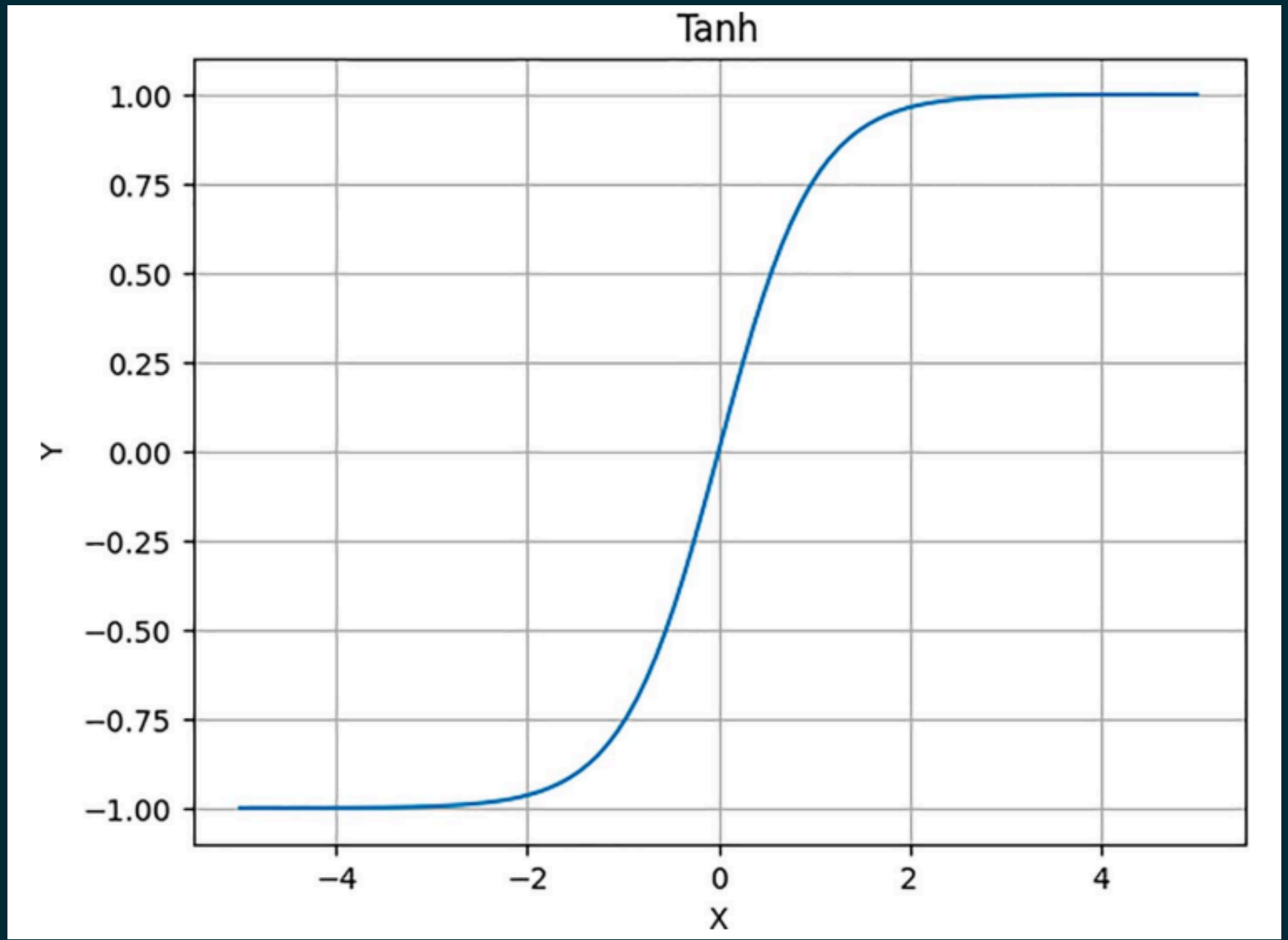


Figura 3-9. Función Tanhiperbólica)

3. **ReLU (Rectified Linear Unit)**

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

- Eficiente, salida $[0, \infty)$.
- Minimiza *vanishing gradient*.
- Problema: valores negativos => 0, “neurona muerta”. Además, no está acotada arriba => “exploding gradient” en ciertos casos.

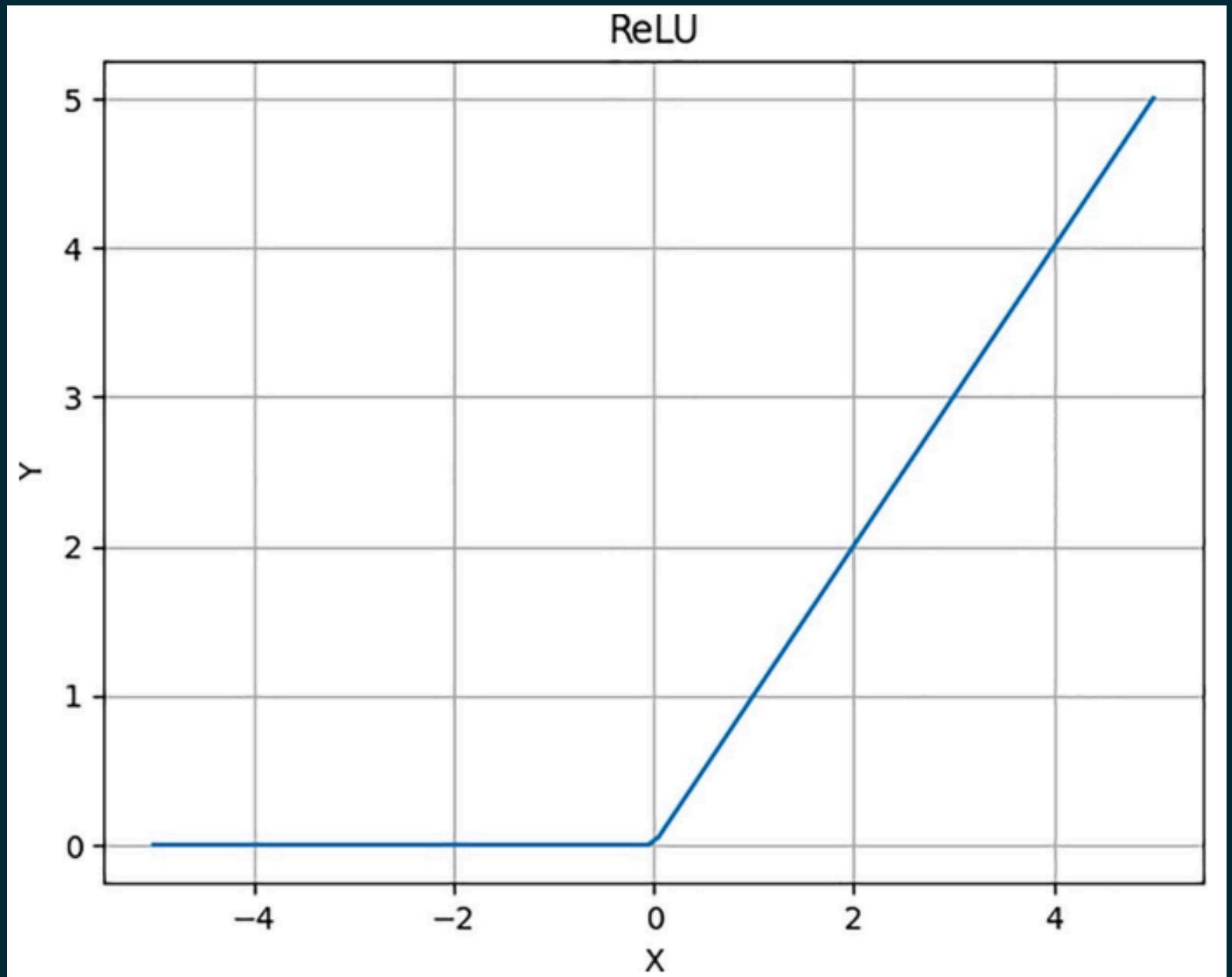


Figura 3-10. Función ReLU)

REFERENCIAS

1. Definición informal de ML.
2. Definición de IA.
3. Definición formal de ML (Tom Mitchell).
4. Minsky, M. & Papert, S. (1969). *Perceptrons*. MIT Press.
5. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Learning representations by back-propagating errors*. *Nature*, 323(6088), 533–536.

6. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). *Gradient-based learning applied to document recognition*. *Proceedings of the IEEE*, 86(11), 2278–2324.
7. Hubel, D. H., & Wiesel, T. N. (1962). *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*. *Journal of Physiology*, 160, 106–154.
8. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks*. In *Proceedings of the 25th International Conference on Neural Information Processing Systems* (NeurIPS).
9. He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep residual learning for image recognition*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR).