

# **Redes Neuronales:**

# **Perceptrón**

**Julieth Sofia Moreno**

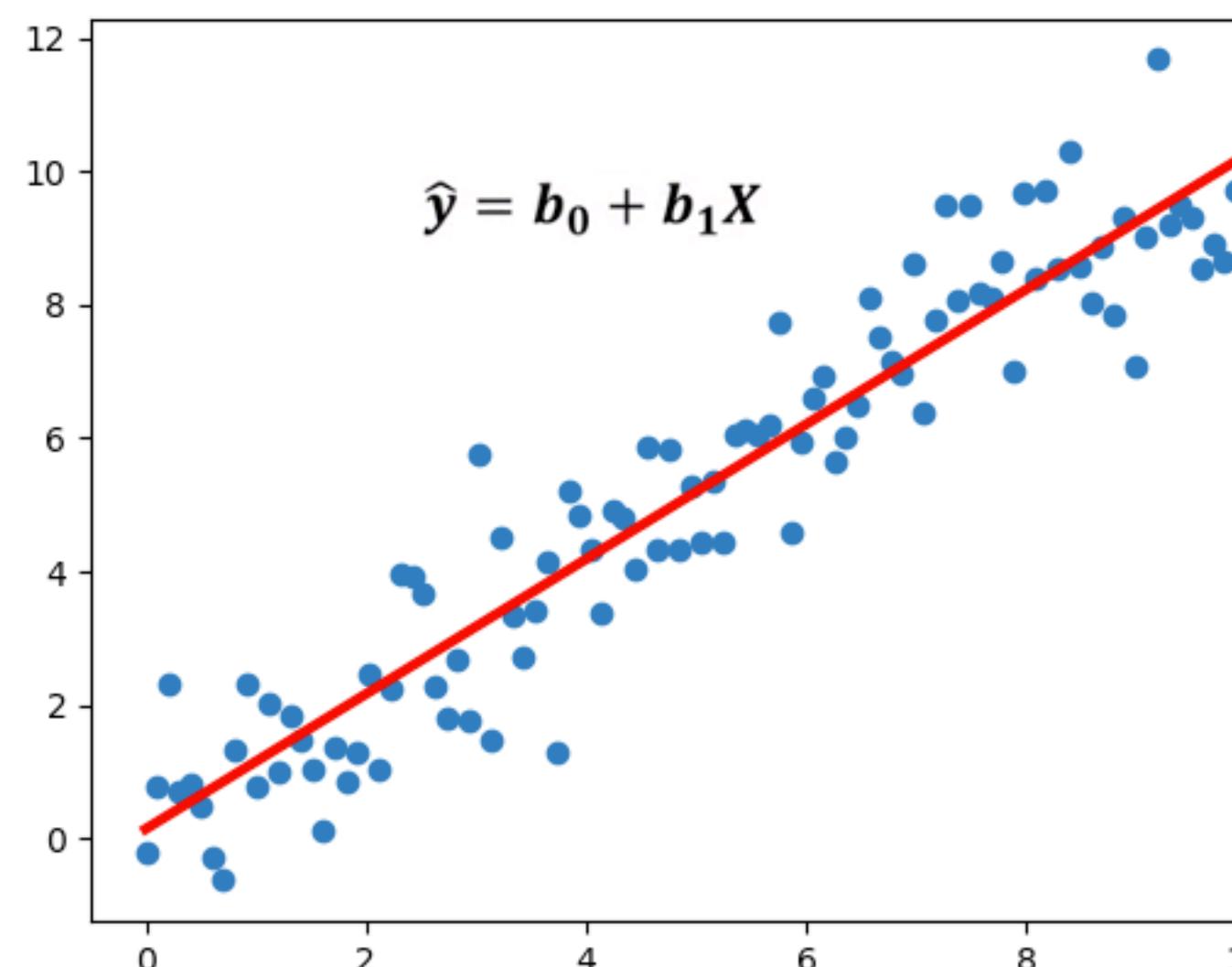
**Ariadna Sofia Contreras abril**

**Nicolás Suárez Gutiérrez**

**Grupo 11- Minería de Datos**

# INTRODUCCIÓN

Uno de los problemas más comunes en el mundo de la ciencia de datos es el de la predicción del valor de ciertas variables a partir de otras variables para resolver problemas de clasificación o regresión. Los problemas de clasificación y de regresión constituyen las dos principales clases de lo que se llama aprendizaje automático supervisado, perceptrón, un modelo matemático que es entrenado para aprender a predecir una o más variables.



# Introducción

En el proceso de entrenamiento, el modelo debe entender cómo afectan las distintas características, variables, y así, posteriormente, conociendo solamente las demás variables, ser capaz de predecir o clasificar en una variable final.

Existen varios modelos de aprendizaje distintos para resolver problemas como:

- **K-Nearest Neighbors** (usado sobre todo para clasificación).
- La **regresión logística** (usada para clasificación y regresión) .
- La **regresión lineal** (para regresión).

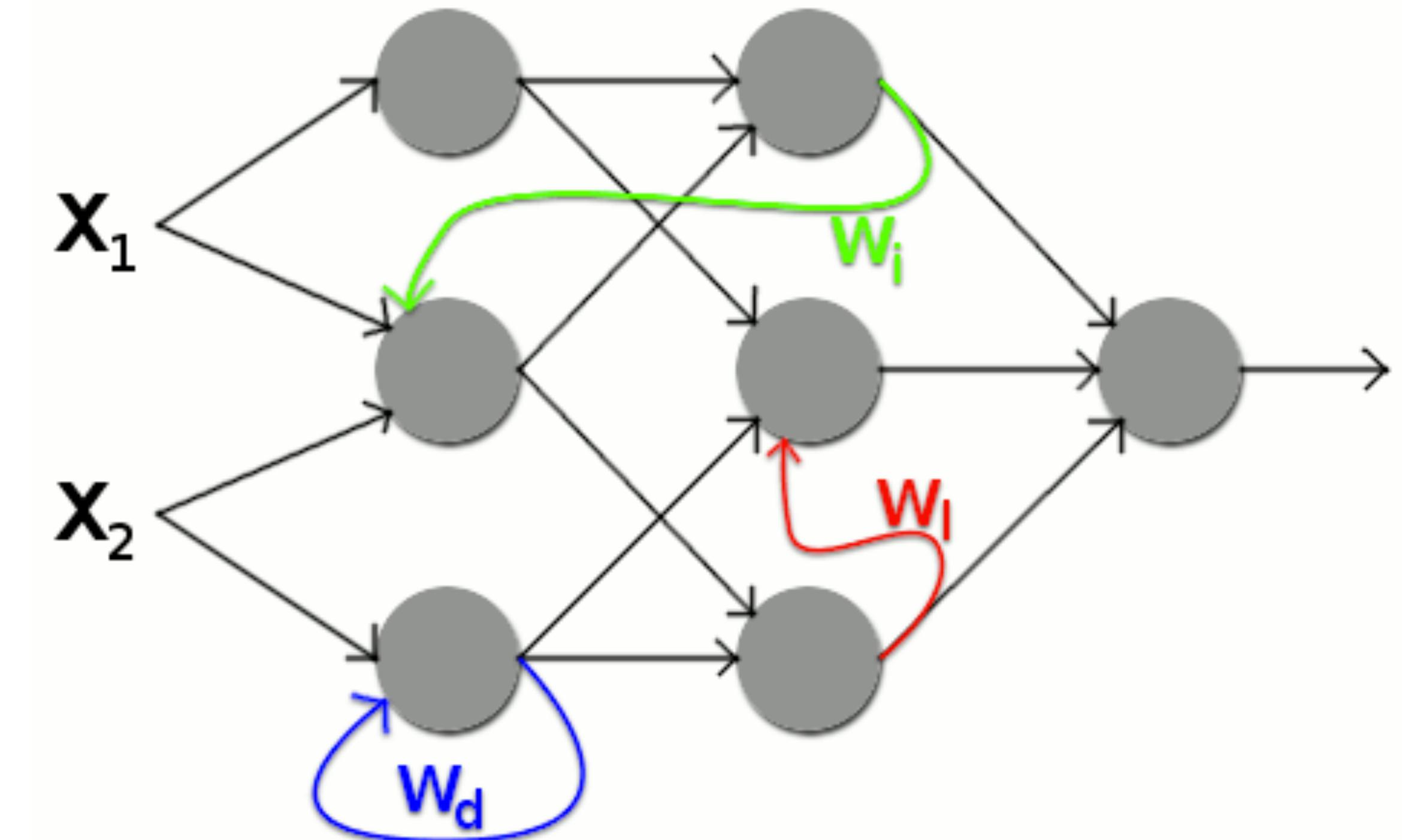
# **Redes Neuronales Artificiales**

# Redes Neuronales

## Introducción

Las redes neuronales están formadas por billones de neuronas conectadas entre ellas, y es esta arquitectura la que permite el aprendizaje de los animales.

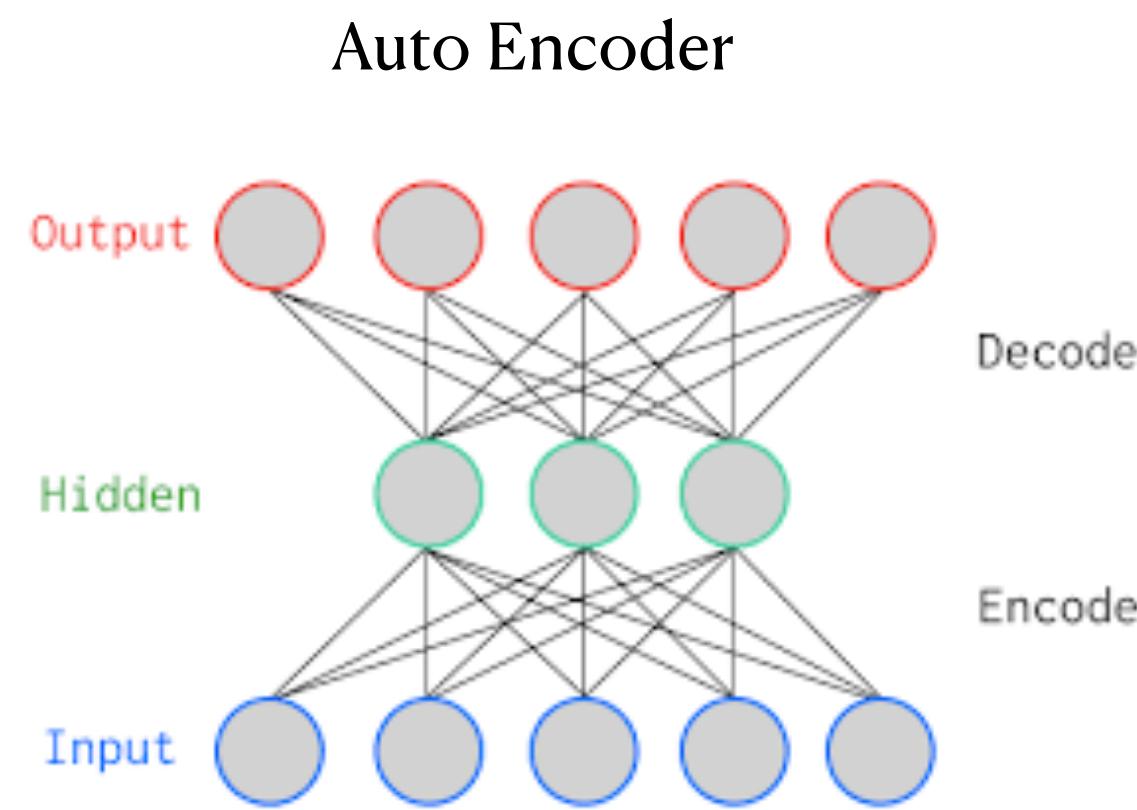
Con base en eso planteamos las redes neuronales artificiales.



**Pues bien, las redes neuronales artificiales pretenden imitar el funcionamiento de las redes neuronales biológicas. La principal unidad constituyente de las redes neuronales artificiales es el perceptrón simple. Este no es más que la representación matemática de una neurona, y de hecho imita su estructura y las funciones de cada una de sus partes.**

## **PERCEPTRÓN**

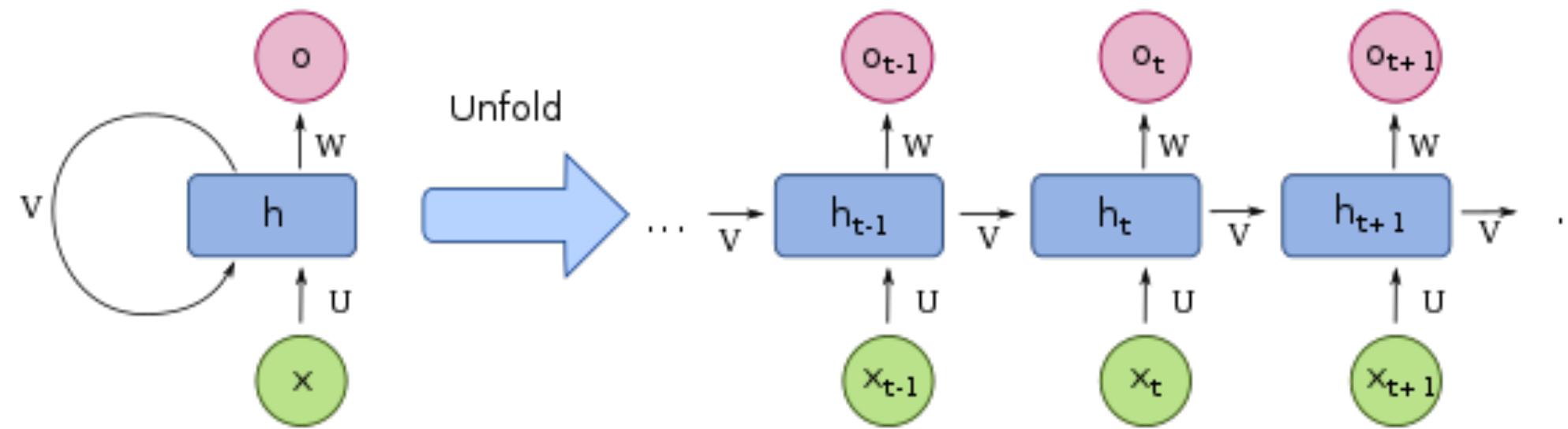
# INTRODUCCIÓN



<https://upload.wikimedia.org/wikipedia/commons/o/o6/AutoEncoder.png>

Debido a la aparición de problemas nuevos en cuanto al tratamiento y procesamiento de la información es que las redes neuronales se han venido desarrollado constantemente, permitiendo así que existan diversos tipos de estructuras para una red neuronal, las cuáles dependen del propósito con el cual se desea construir dicha red.

Red recurrente



[https://commons.wikimedia.org/wiki/File:Recurrent\\_neural\\_network\\_unfold.svg](https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg)

Algunos ejemplos de estructuras son:

- Perceptrón
- Auto-encoder
- Red recurrente
- Red convolucional

A partir de esto podemos entender un poco mejor que es un perceptrón.

# **Reseña Histórica**

## **Frank Rosenblatt**

Frank Rosenblatt fue un psicólogo estadounidense, considerado en la actualidad como el padre del Deep learning.

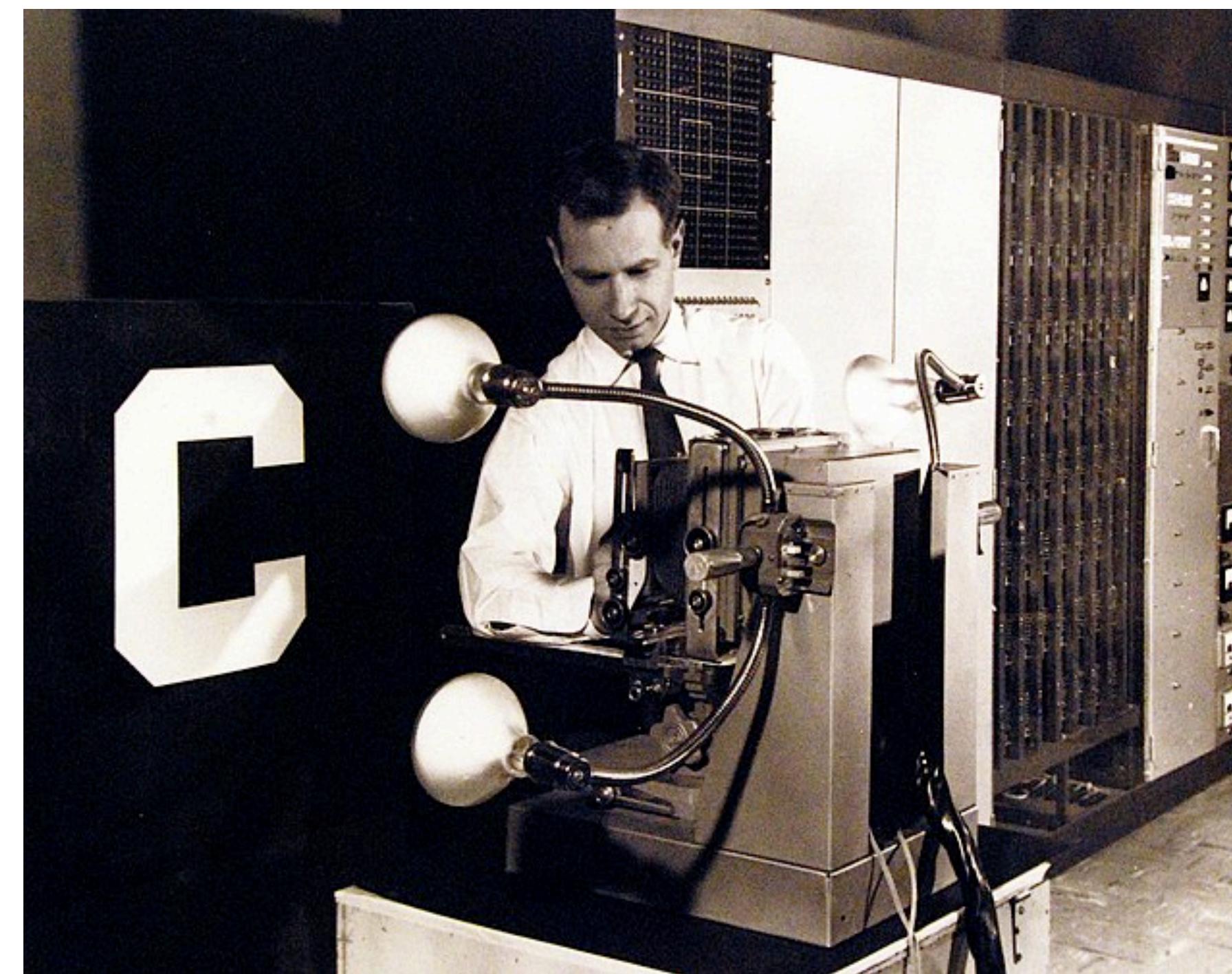
- Nació el 11 de julio de 1928 en Nueva York.
- Estudió psicología en el instituto Bronx de Ciencia.
- Posteriormente obtuvo la licenciatura en letras y un doctorado en filosofía.
- Jefe de la sección de sistemas cognitivos del Laboratorio Cornell Aeronáutico.
- Falleció el 11 de julio de 1971.

Tras su muerte las investigaciones en redes neuronales se detuvieron hasta la época de los 80 cuando nuevos investigadores retomaron el trabajo de Rosenblatt.



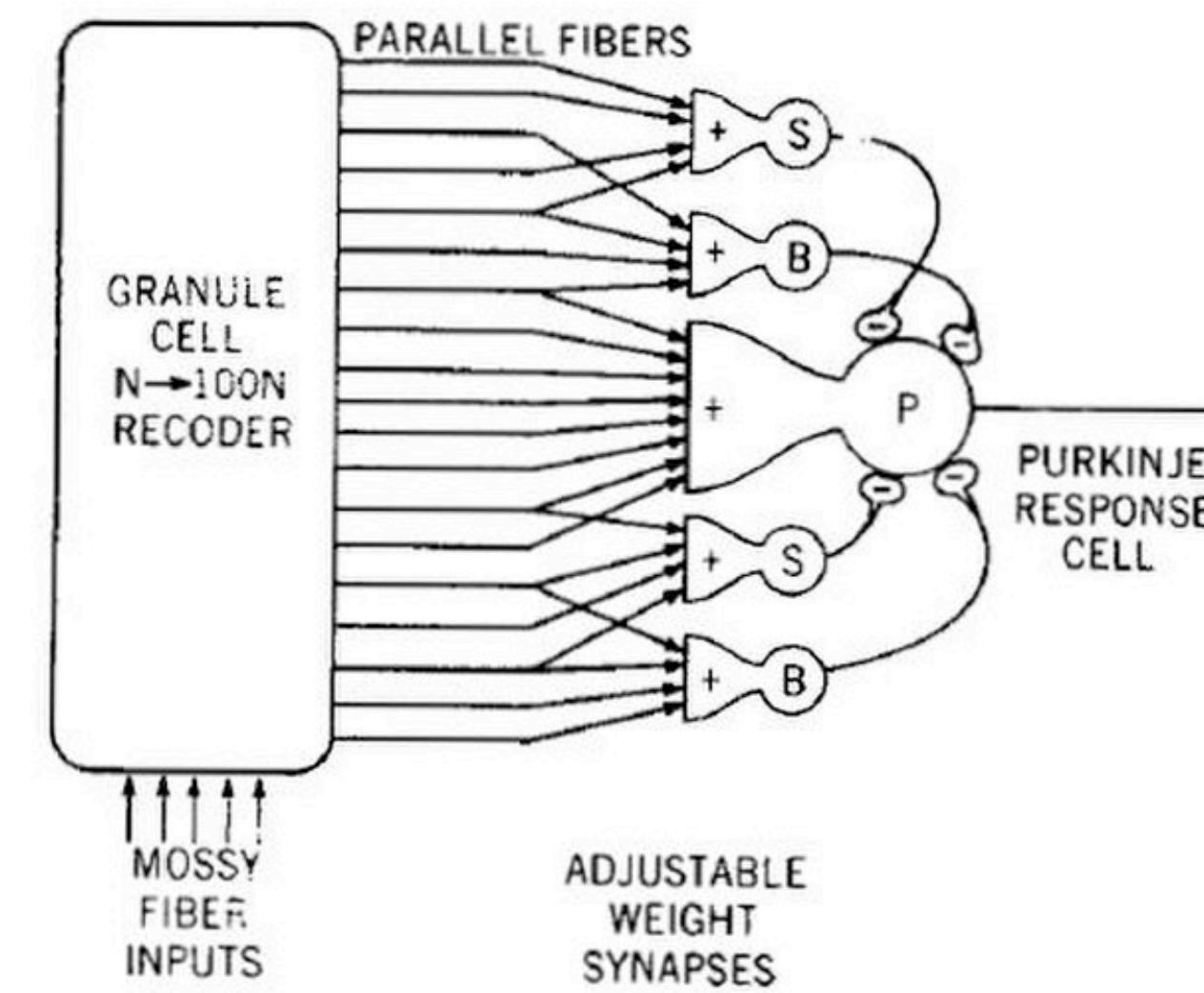
# Mark I Perceptrón

- El trabajo de Frank Rosenblatt, fue el principal referente del “Mark I perceptrón” el primer computador diseñado para crear redes neuronales artificiales.



# Mark I Perceptrón

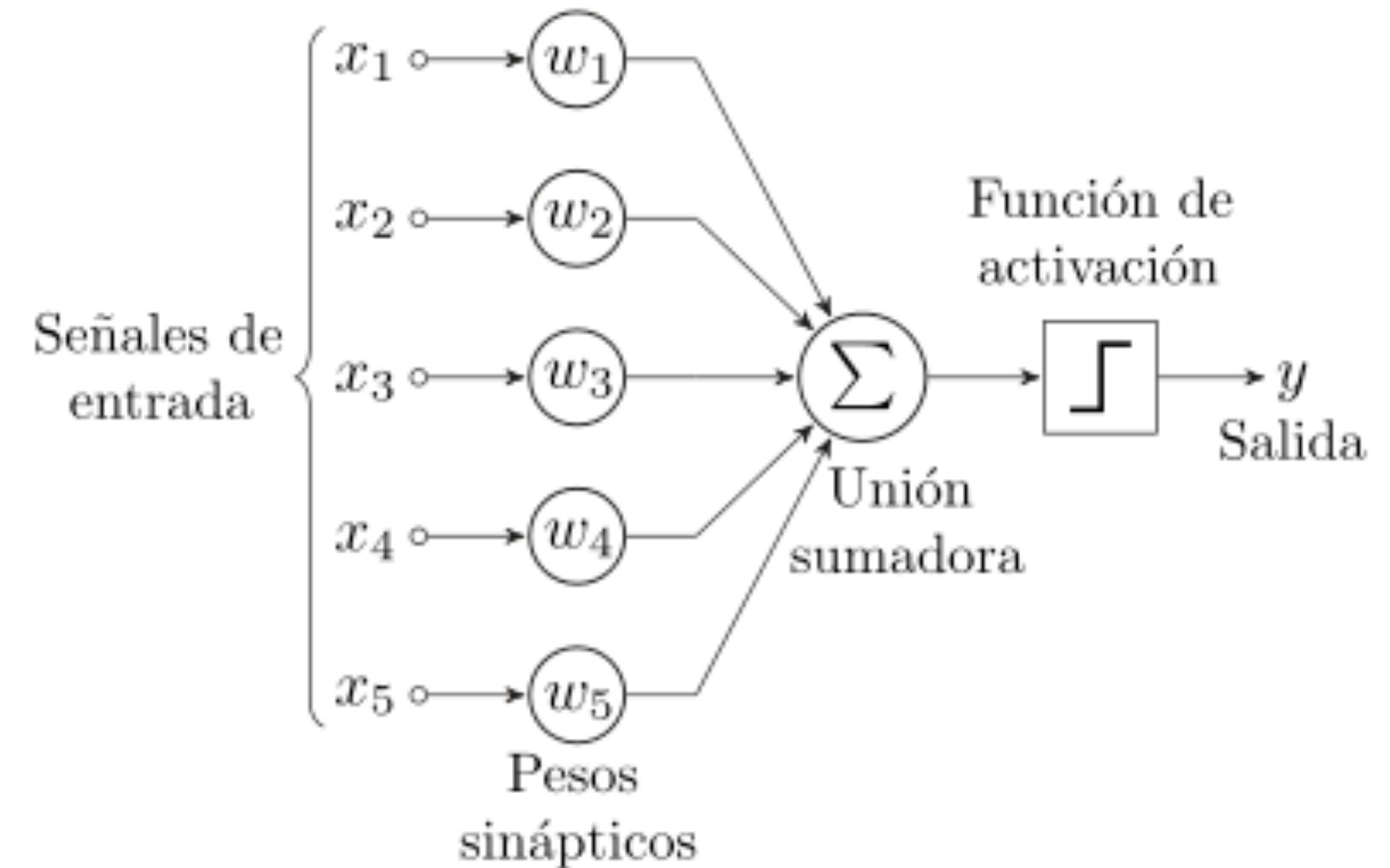
- Podemos clasificarlo como deep learning pues no hacía falta ponerle parámetros preestablecidos, pues el mismo programa construye sus parámetros cada vez que realiza una repetición del experimento, la función del mark I es descifrar el sexo de rostros humanos.



# ¿Qué es un perceptrón?

es una estructura de red neuronal

- un perceptrón funciona como una neurona, pues recibe unas señales de entrada, las cuales pondera por los pesos sinápticos y las suma, posteriormente la función de activación que nos da unas señales para lograr "encender" la neurona,



# ¿Qué es un perceptrón?

podemos resumir el perceptrón simple como una función matemática que devuelve como resultado 0's y 1's y el perceptrón multicapa una generalización de este el cual además admite y predice variables discretas o continuas ya sea para un problema de clasificación de regresión. Entonces se puede entender como un algoritmo clasificador el cual puede hacer una predicción con base a unos valores de entrada y un aprendizaje previo.

1	0	0	1
1	0	1	0
0	1	1	1
1	1	0	1

# **Fundamentación matemática**

**Funcionamiento general**

# Partes de un Perceptrón

## 1. Entradas

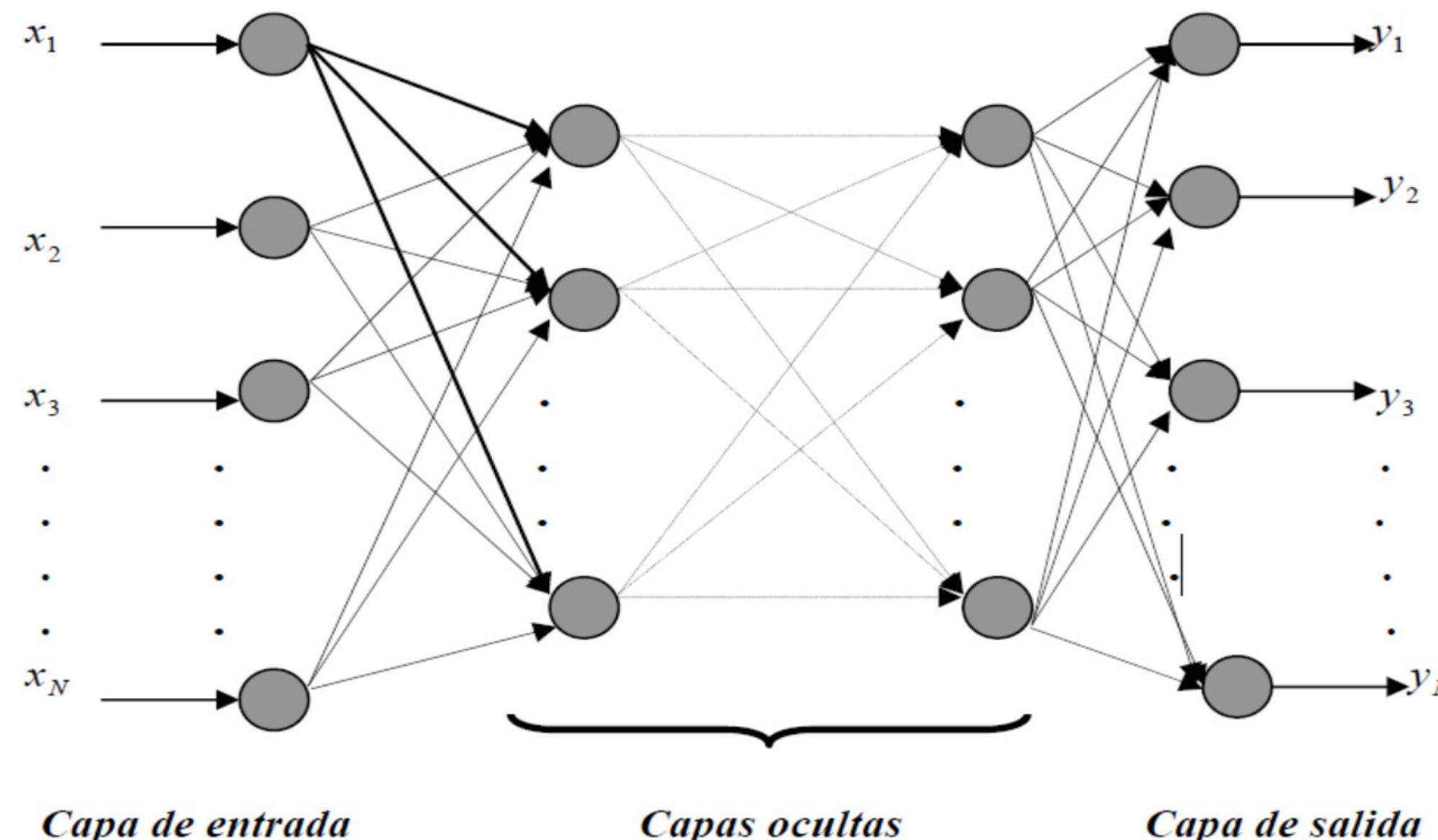
Dependiendo del modelo las entradas pueden ser binarias o continuas, cada valor se guarda en un  $x_j$  vector de tamaño  $n$ .

En el caso de un multicapa la salida de la neurona oculta  $i$  son las entradas de las siguiente neurona  $i+1$ .

# Partes de un Perceptrón

## 2. Pesos sinápticos asociados a las entradas

Representan la intensidad de interacción  $W_i$  entre cada neurona presináptica j y la neurona postsináptica i, el producto sera un  $W_{ij}$  estos son guardados en una Matriz  $W$ .



# Partes de un Perceptrón

## 3. Regla de propagación

Proporciona el valor del potencial postsináptico,  $h(t)$ , de la neurona  $i$  en función de sus pesos y entradas.

$$\sigma(w_{ij}, x_i(t))$$

$$\left( \sum_{j=1}^n w_{ij}x_j \right)$$

# Partes de un Perceptrón

## 4. Función de activación que representa simultáneamente la salida de la neurona y su estado de activación:

Proporciona el estado de activación actual,  $a$ , de la neurona  $i$  en función de su estado anterior,  $a - 1$ , y de su potencial post-sináptico actual. En muchos modelos de ANS (Artificial Neuronal System) se considera que el estado actual de la neurona no depende de su estado anterior, sino únicamente del actual.

# Perceptrón simple

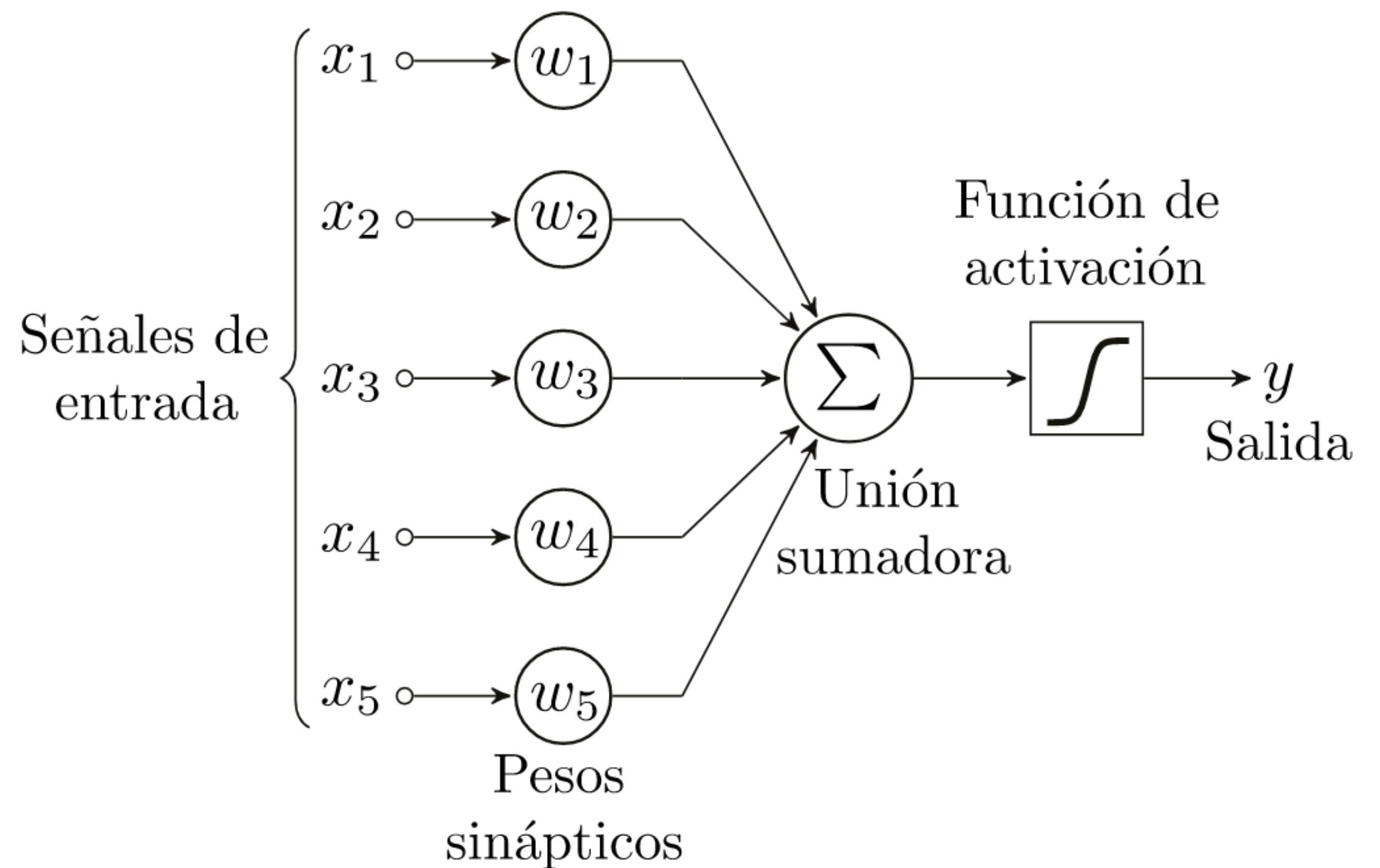
Al tener  $n$  neuronas de entrada y  $m$  de salida un perceptrón simple se puede expresar de la siguiente forma

$$y_i(t) = H \left( \sum_{j=1}^n w_{ij} x_j \right), \forall i \quad 1 \leq i \leq m$$

Donde  $H(\cdot)$  es conocida como función de activación, la cual puede ser la función escalón de Heaviside, y  $t$  el umbral de decisión, se puede concluir que el perceptrón simple estaría formado por dispositivos de umbral.

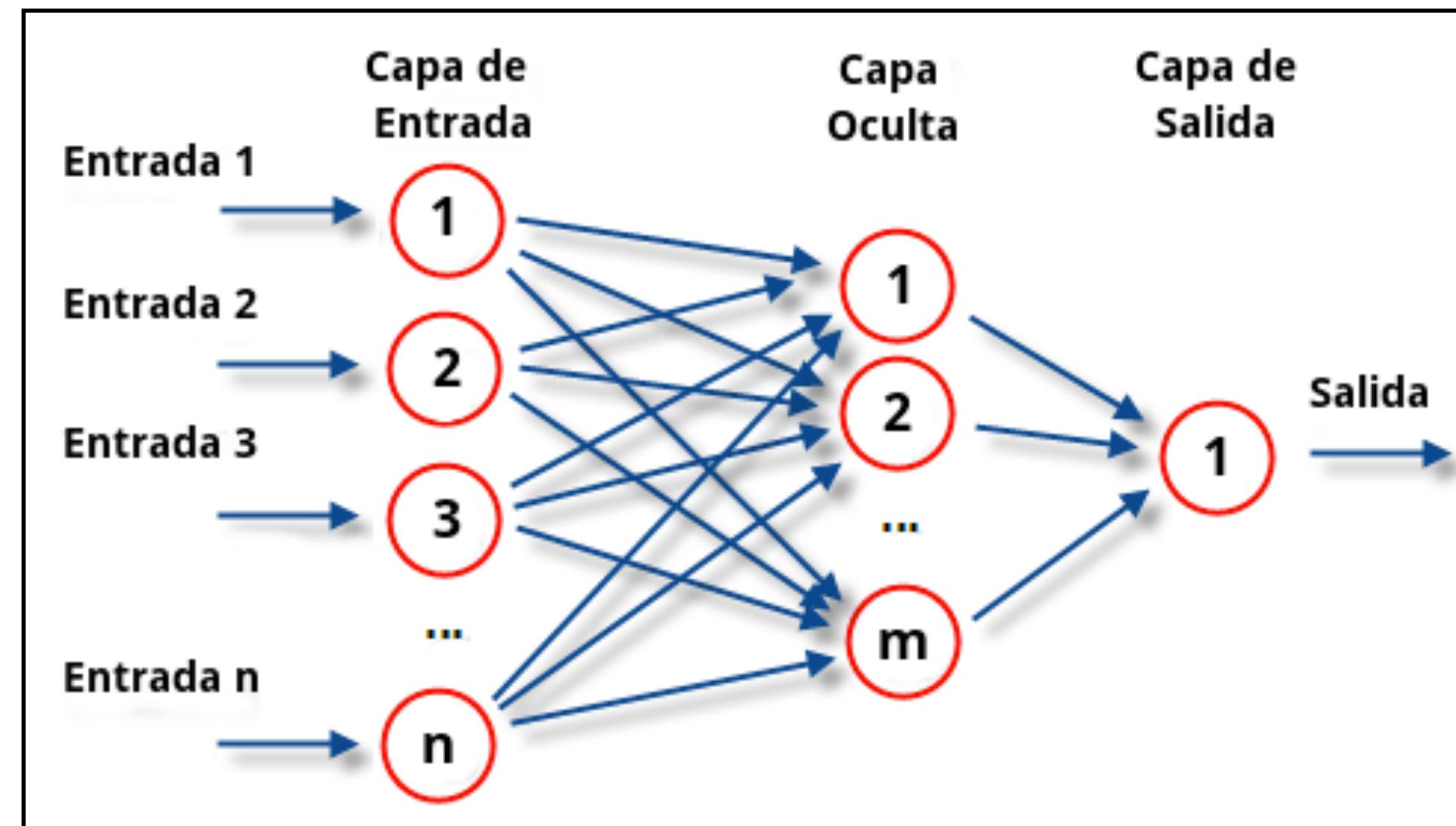
**Como se puede observar la regla de propagación que se usa es la suma ponderada del producto escalar por el vector de entrada, el cual se debe superarse en cierto umbral.**

Nota: el perceptrón simple es útil para la representación de funciones booleanas



# Perceptrón Multicapa

La ventaja de esta generalización es su algoritmo de aprendizaje (algoritmo de retropropagación)

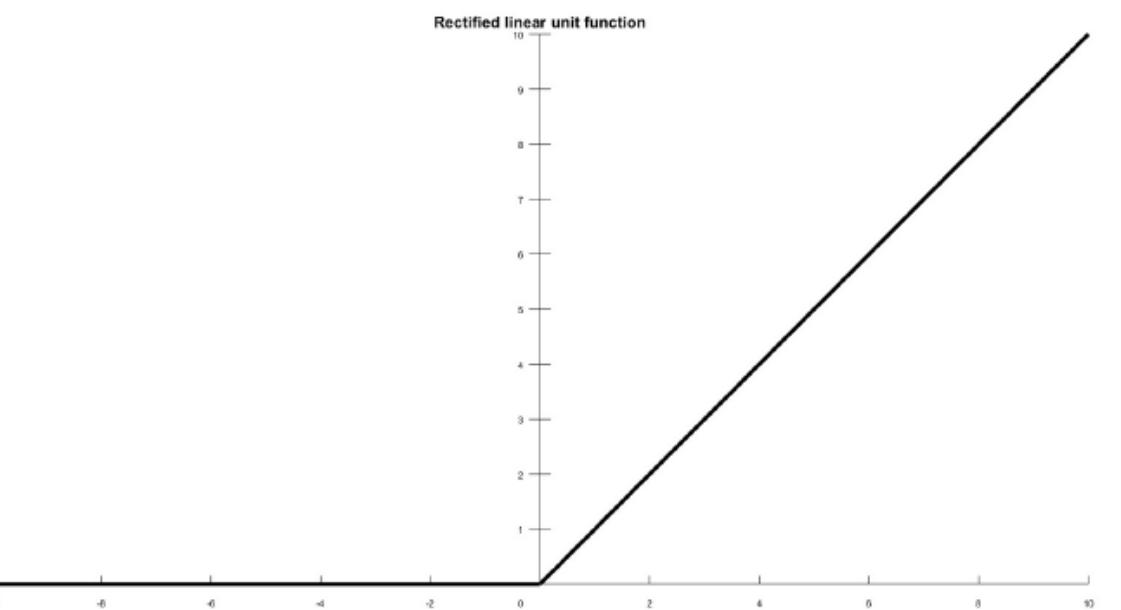


El perceptrón multicapa es una generalización del perceptrón simple, con las siguientes particularidades:

- Se utilizan una o más capas ocultas
- Se permiten neuronas de entrada continuas
- Las funciones de activación son de tipo sigmoide

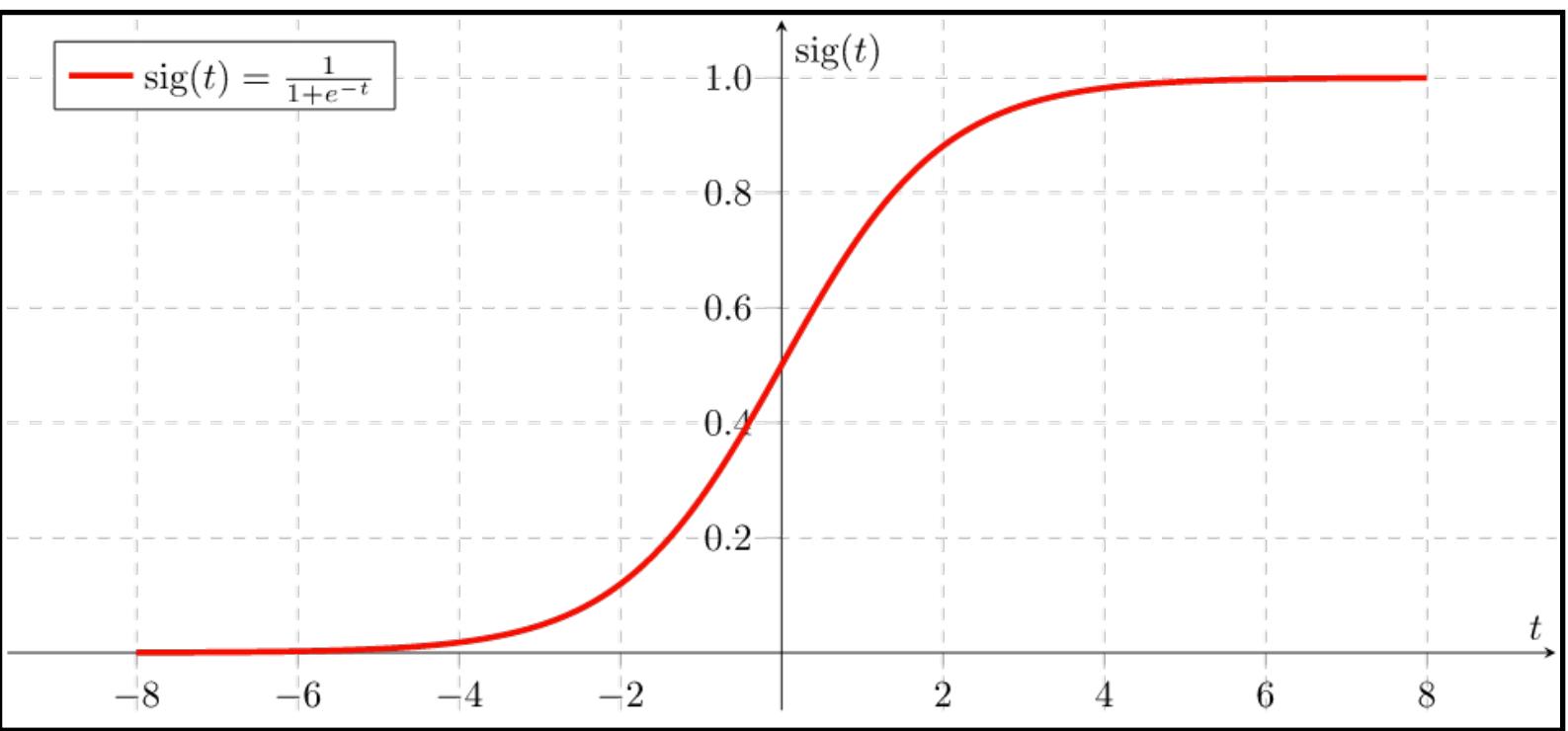
# Funciones de activación

## RELU



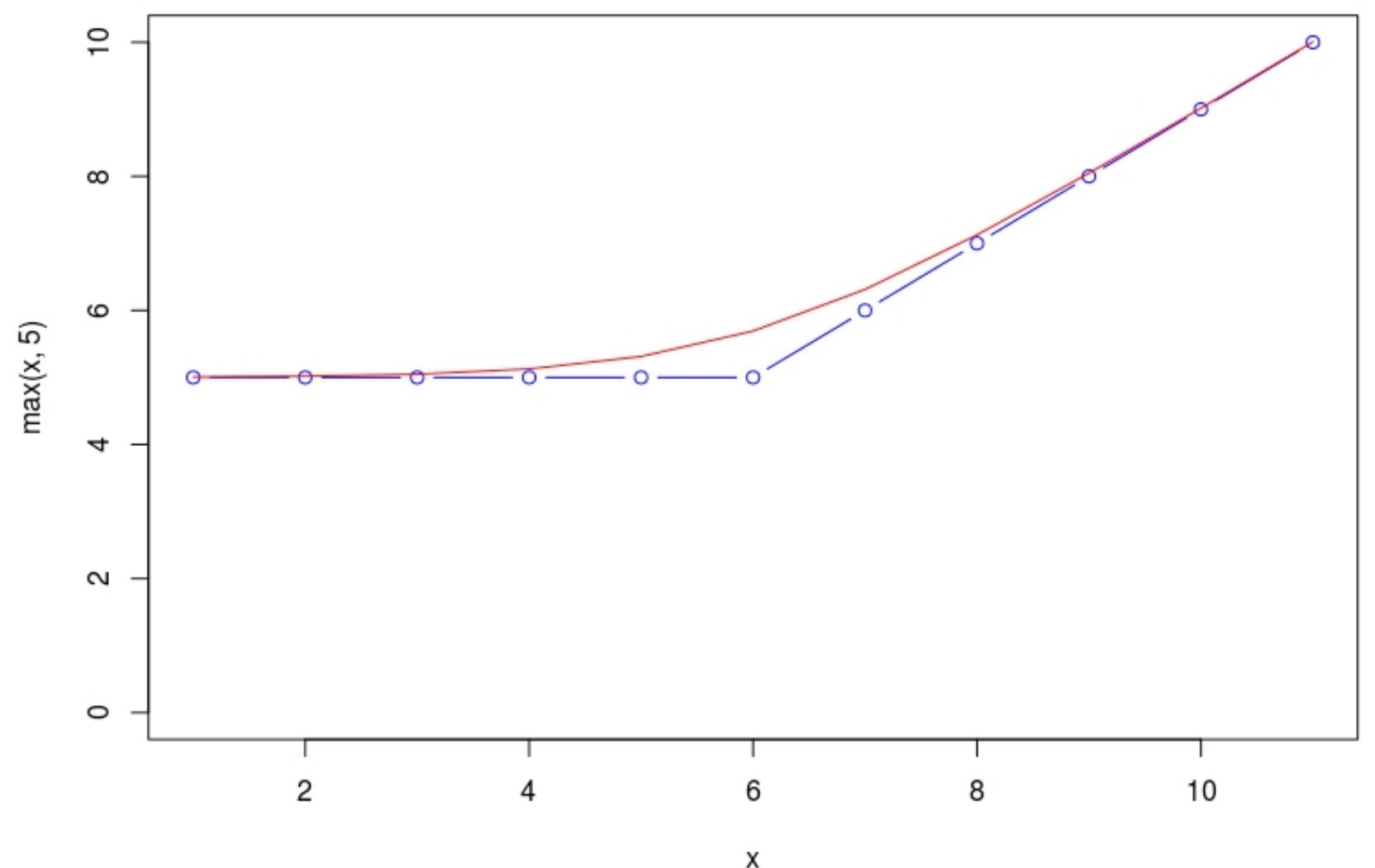
$$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x \mathbf{1}_{x>0}$$

## SIGMOIDE



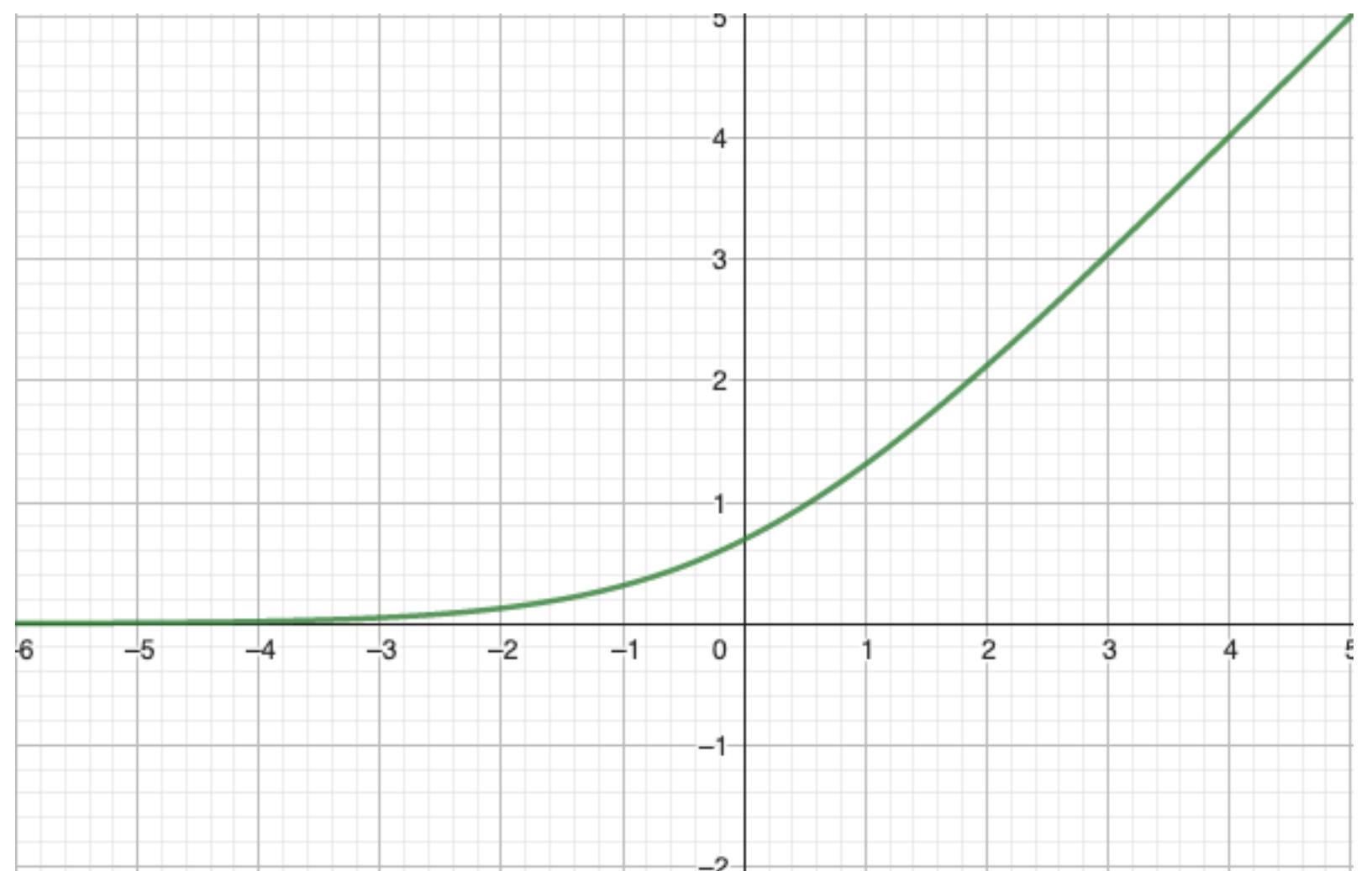
$$\sigma(x) \doteq \frac{1}{1 + e^{-x}}$$

## Softmax



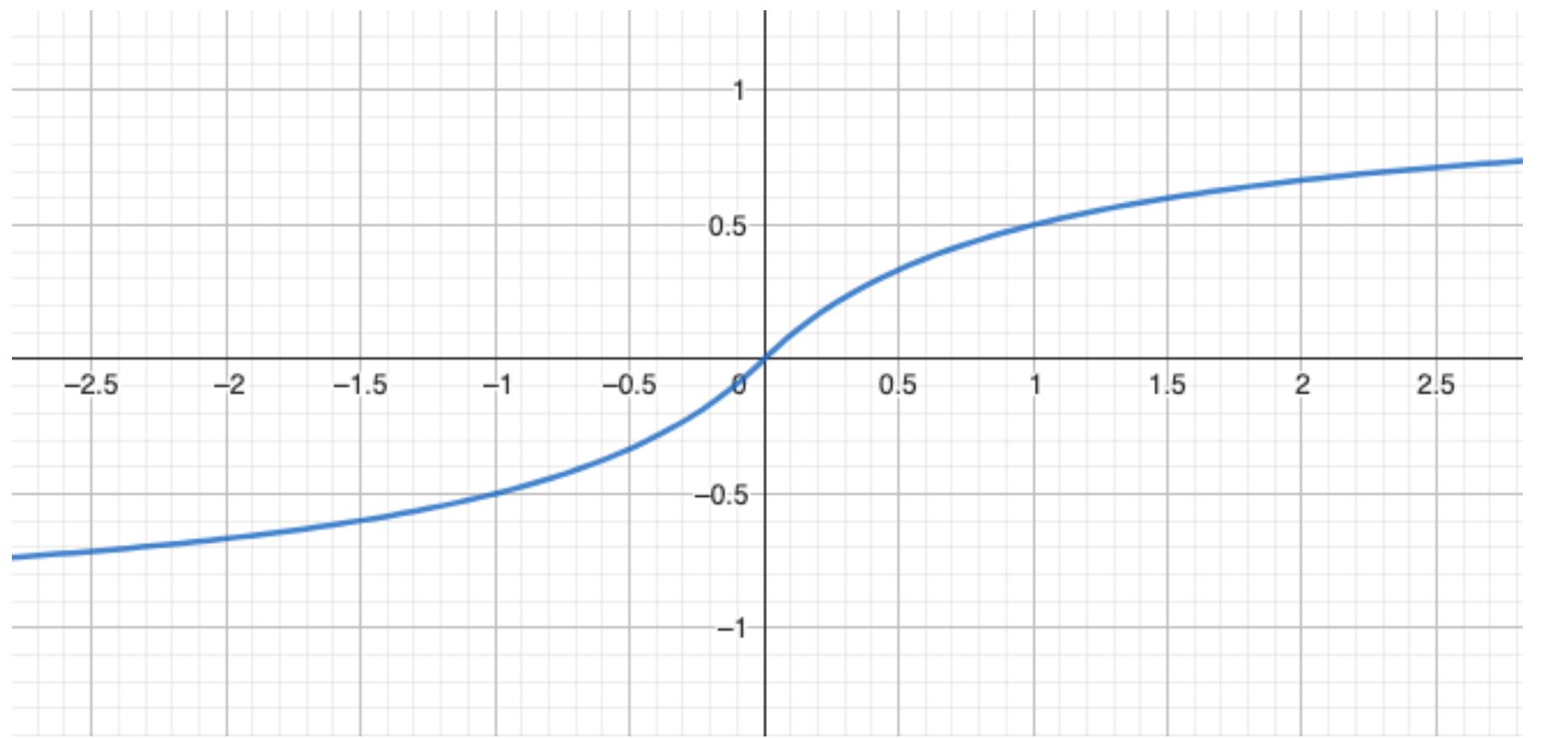
Se puede programar mediante el comando `exp(x) / tf.reduce_sum(exp(x))`  
el cual usa el paquete tensorflow

## Soft plus



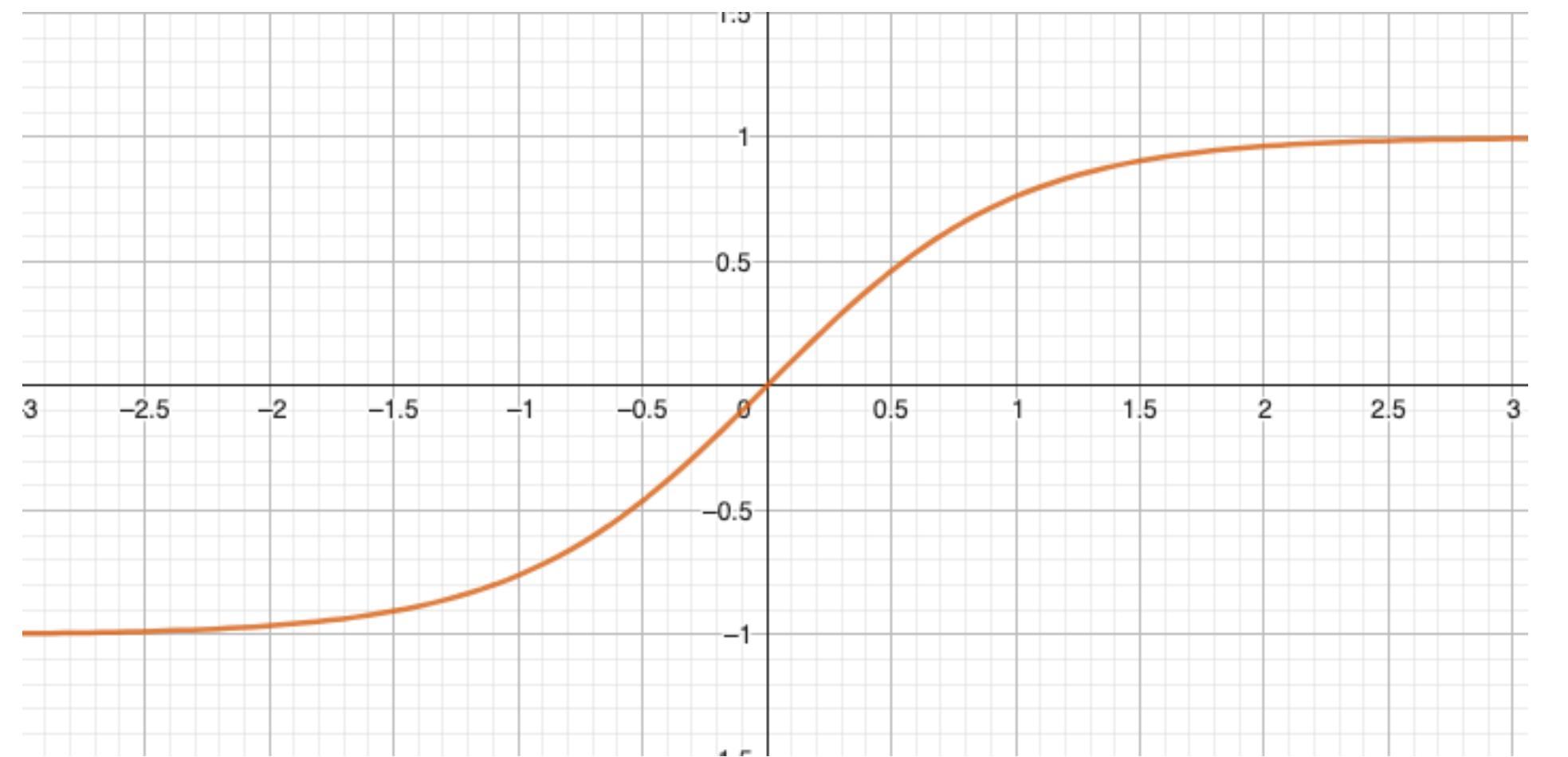
$$\ln(1 + e^x)$$

## Softsign



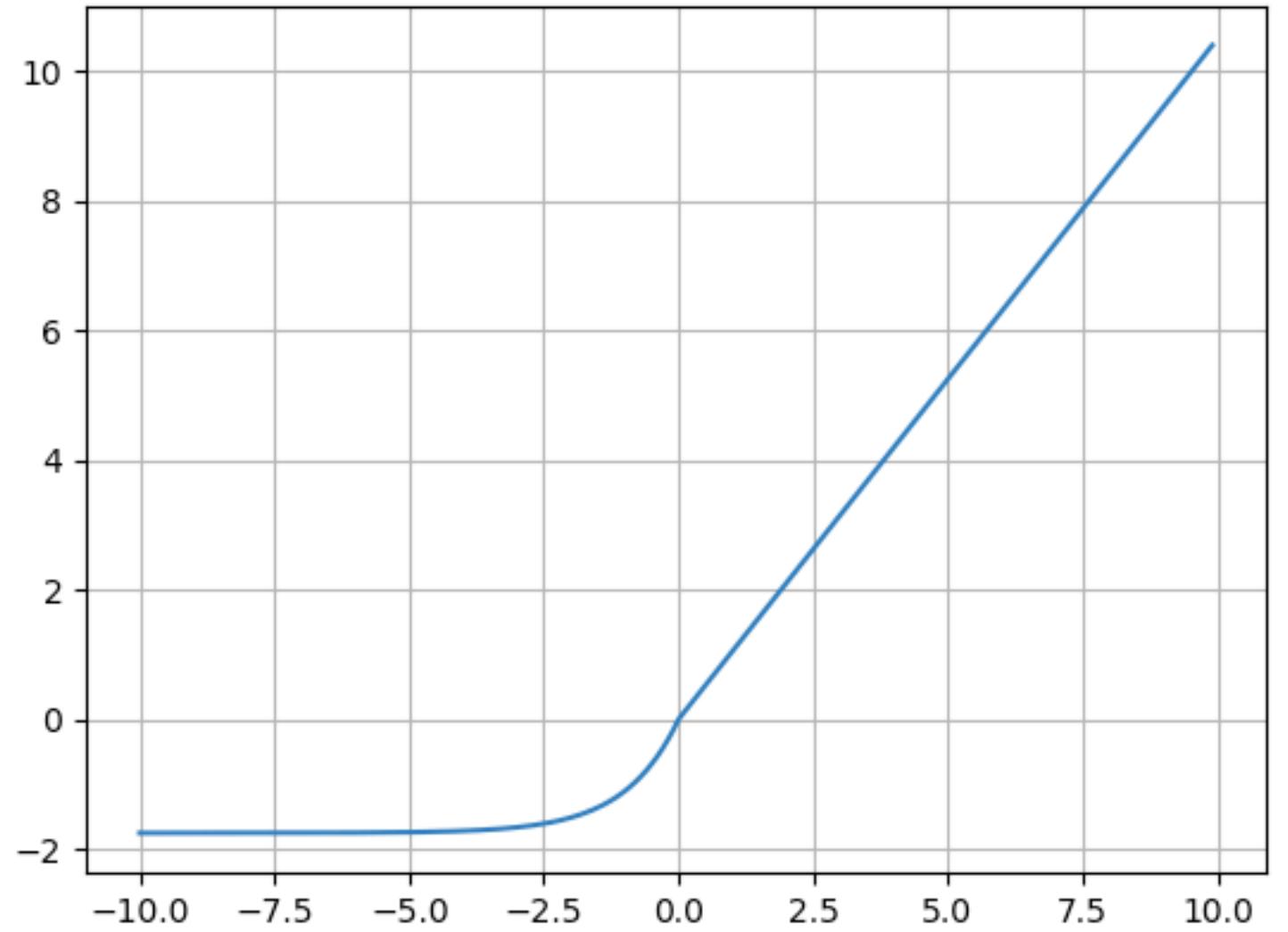
$$f(x) = \frac{x}{|x| + 1}$$

## Tanh



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

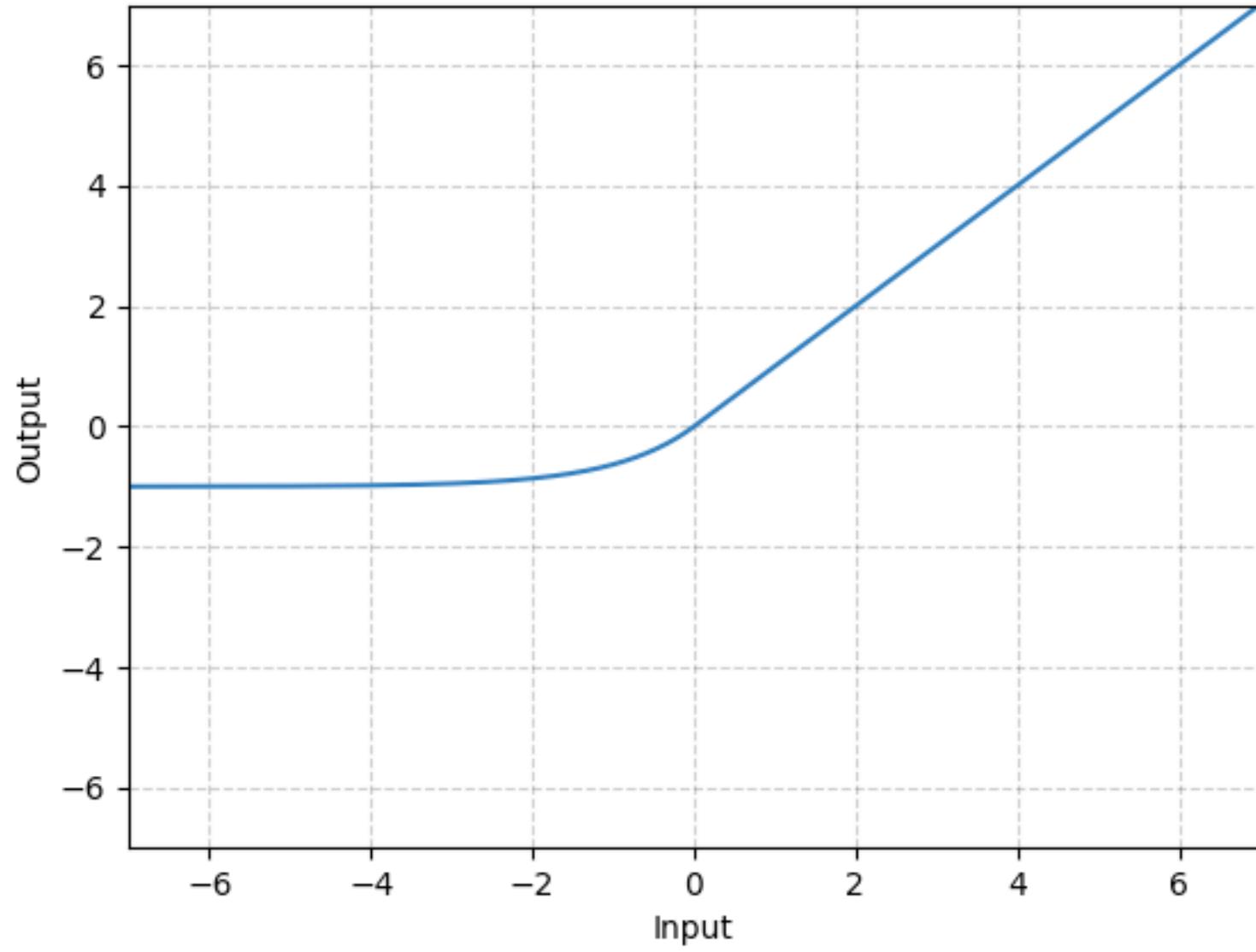
## SELU



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

with parameters  $\lambda = 1.0507$  and  
 $\alpha = 1.67326$

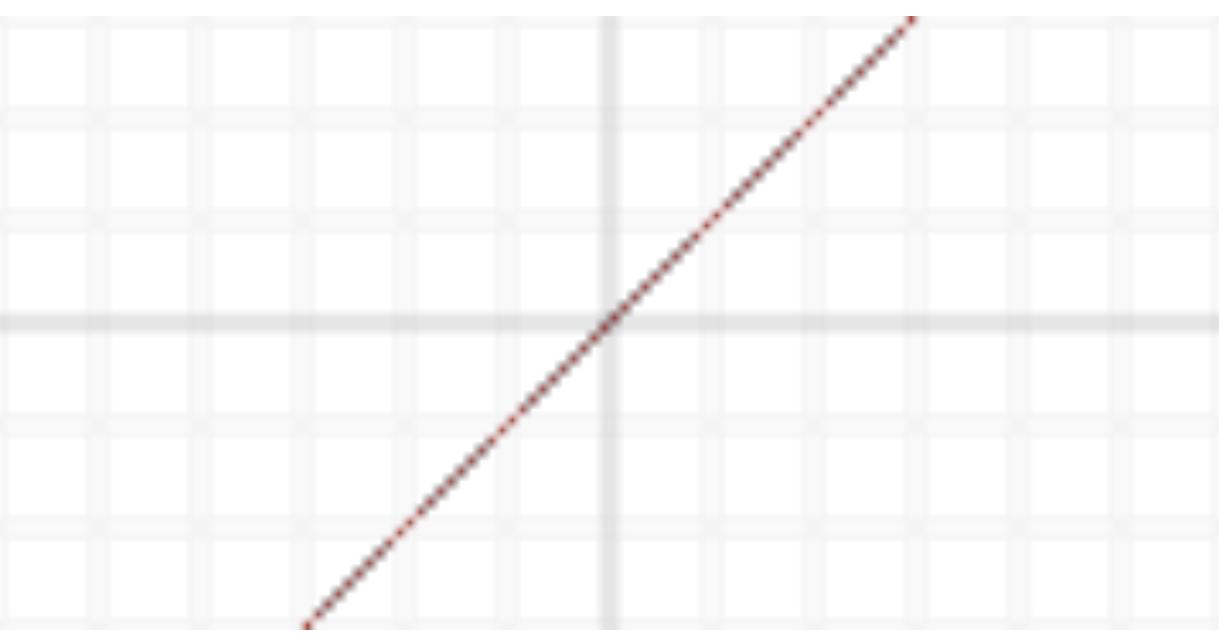
## ELU



$$\begin{cases} \alpha (e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

with parameter  $\alpha$

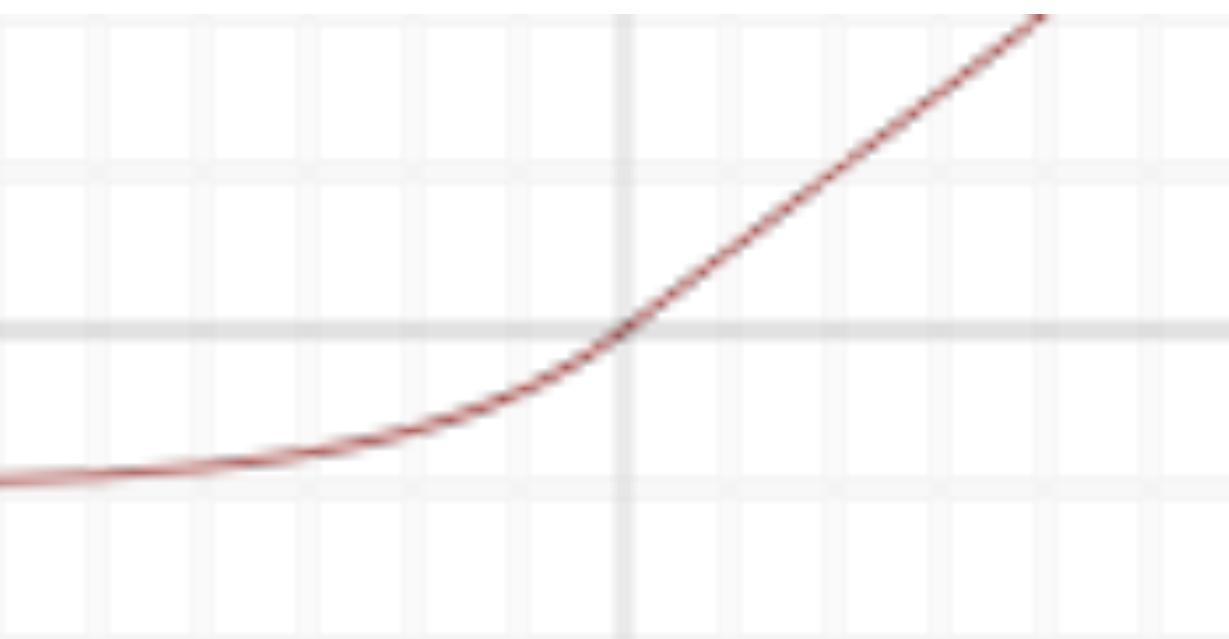
Lineal



Es la unica que se usa en el perceptron simple

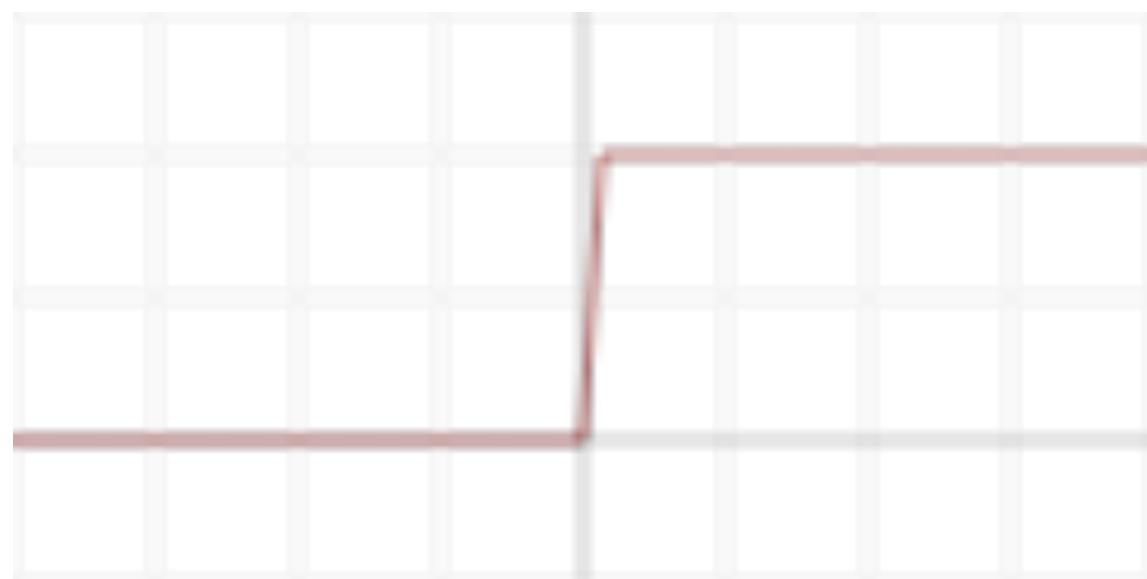
$$x = y$$

Exponencial



$$e^x - \alpha$$

Binary step



$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

# Aprendizaje

**Para un perceptrón existen dos tipos de aprendizaje, uno utiliza una tasa de aprendizaje, de modo que amortigua el cambio de los valores de los pesos, y el otro no utiliza tasa de aprendizaje.**

**Sea  $\delta$  la salida esperada y  $\alpha$  una constante tal que  $0 < \alpha < 1$ .**

**el primer tipo de aprendizaje actualiza los pesos de la siguiente manera**

$$w'_j = w_j + \alpha(\delta - y)x_j$$

**Mientras que el segundo tipo de aprendizaje utiliza la regla de actualización**

$$w'_j = w_j + (\delta - y)x_j$$

**Los parámetros se van a actualizar después de cada iteración solamente si la salida  $Y$  difiere de la estimada  $\delta$**

# Entrenamiento

*guiado*

Es de notar que para realizar el proceso de entrenamiento se debe definir una función de error, que indique la diferencia entre el valor estimado y el real. La más común es el error cuadrático medio (ECM)

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} (y_d - \delta_d)^2$$

donde:

- $D$  es el conjunto de datos de entrenamiento
- $y_d$  es la salida correcta
- $\delta_d$  es la estimación de la salida mediante el perceptrón

# Gradiente Descendiente

**El método del gradiente descendiente se basa en buscar la dirección en la que la pequeña variación del vector de pesos hace que el error decrezca más rápidamente.**

**Para encontrar esta dirección se usa el gradiente de la función de error con respecto al vector de pesos**

$$\nabla E(\mathbf{w}) = \left[ \frac{\partial E}{\partial w_0}, \dots, \frac{\partial E}{\partial w_n} \right]$$

**Así luego la actualización de los pesos queda de la siguiente manera:**  $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$   $\Delta \mathbf{w} = -\eta \nabla E(\mathbf{w}) = \eta(y - \delta)x_i$

**donde  $\eta$  es la tasa de aprendizaje**

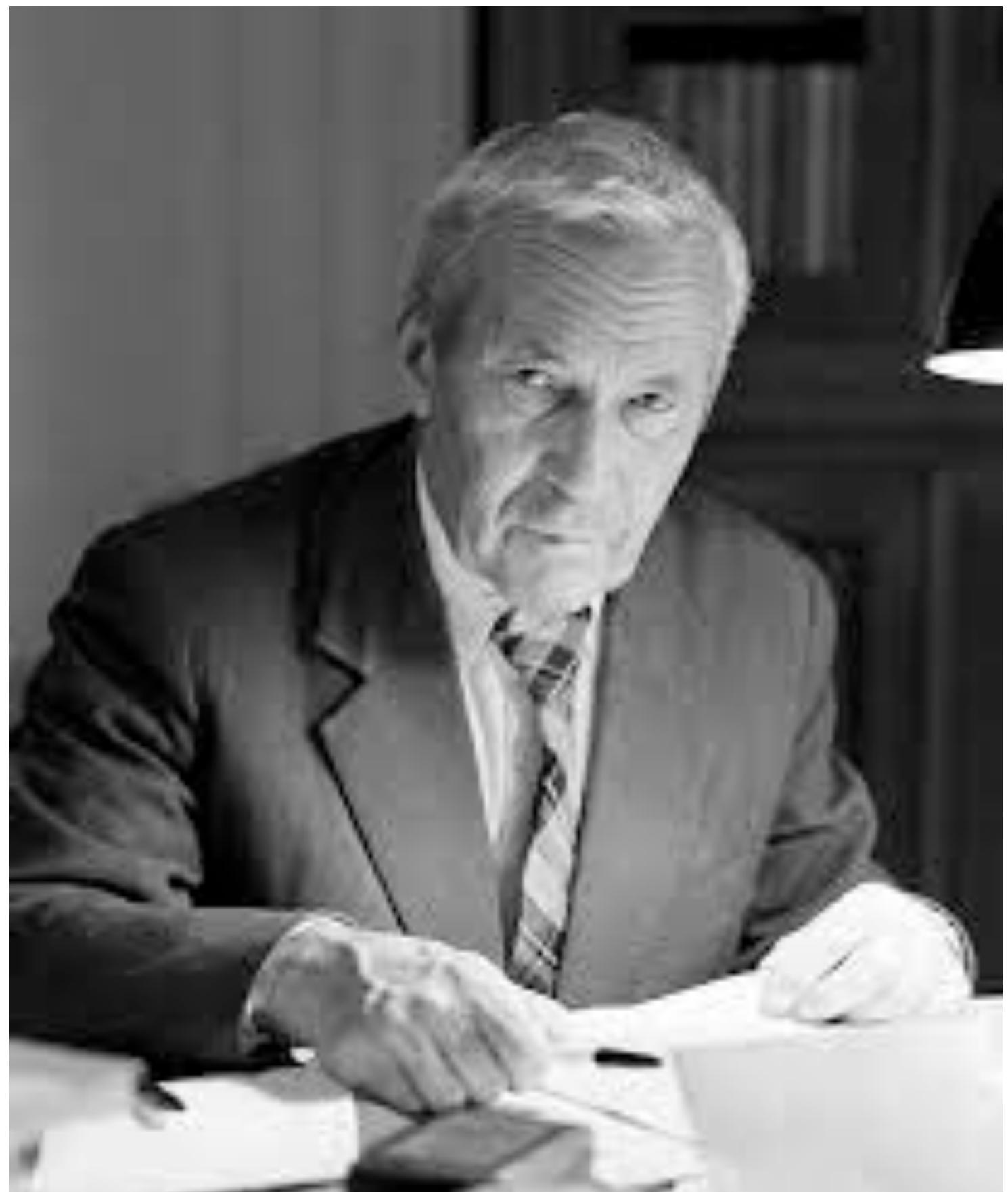
\*\*Nota: esta tasa de aprendizaje normalmente se fija en un valor entre 0.2 y 0.8\*

**Otros métodos para optimizar la función de error, también llamada función de costo, son:**

- Adam
- Gradiente descendiente estocástico
- \*L-BFGS y L-BFGS-B (optimización quasi-Newton)\*
- Batch

# Teorema de Kolmogorov

- Dada cualquier función continua  $f: (0,1)^n \rightarrow R^m$ , existe una RNA (Red neuronal artificial) de 3 capas, de propagación hacia adelante, con  $n$  neuronas de proceso en la capa de entrada,  $m$  en la de salida y  $(2n + 1)$  en la capa oculta; que implementa dicha función de forma exacta.



# Desventajas de un perceptrón

- El tiempo de cómputo necesario para el entrenamiento puede llegar a ser muy alto. Como hemos visto, es un problema asumible, pues una vez entrenada, la red puede congelar sus pesos y continuar funcionando
- *Parálisis de red:* Cuando los pesos alcanzan valores muy altos, la función sigmoide hace que la salida de la capa oculta sea muy cercana a 0 ó 1. De las ecuaciones del algoritmo de BP(Back Propagation) se deduce que el incremento de pesos en esos casos es prácticamente nulo, lo cual produce una parálisis en el entrenamiento. Una posible solución es añadir una pequeña cantidad de ruido a la salida de la neurona.
- *Mínimos locales:* la función de error de una red compleja está llena de valles y picos, y la propia naturaleza del algoritmo puede producir la caída en uno de los valles, que no es necesariamente el mínimo. Una posible solución es incrementar el número de neuronas ocultas, pero como se ha visto, estos produce otros efectos secundarios indeseables.

# Ejemplo Regresión

**Se va a considerar un conjunto de datos simulados entre los cuales va a haber una relación lineal entre las covariables ( $x,y$ ) y la variable respuesta ( $\phi$ )**

	Unnamed: 0	x	y	phi
0	0	73.517682	44.800793	1116.826387
1	1	32.756284	28.724427	1066.212487
2	2	-13.308013	2.458485	490.735354
3	3	62.884936	5.696075	-1130.689639
4	4	47.861635	29.955114	782.448037
...	...	...	...	...
166	166	84.174447	7.278451	-1540.365854
167	167	-23.119203	9.013067	1145.697153
168	168	-5.747350	18.067861	1308.279480
169	169	16.419795	12.980806	450.696513
170	170	46.245703	36.231065	1222.891304

171 rows x 4 columns

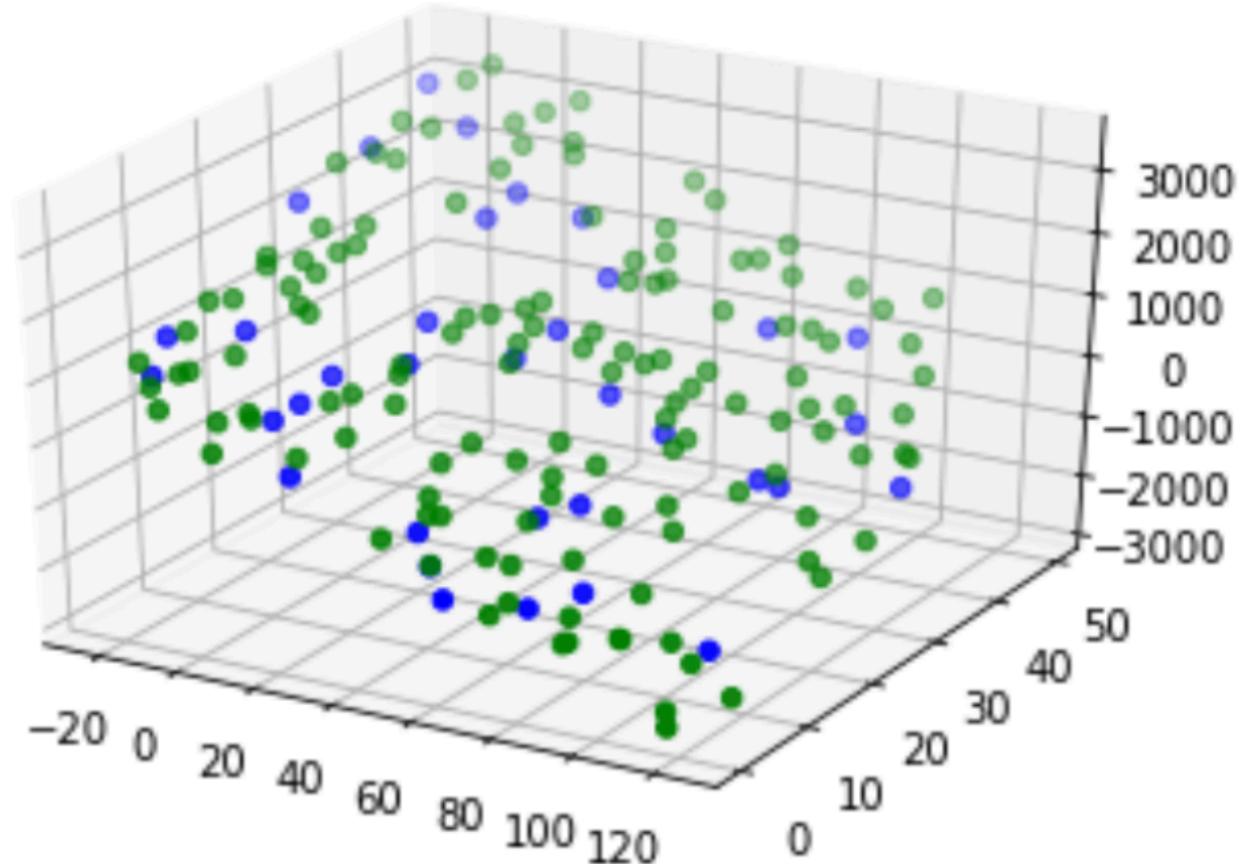
Coeficientes de correlación de Pearson

$$P_{x\phi} = -0.7401154624153701$$

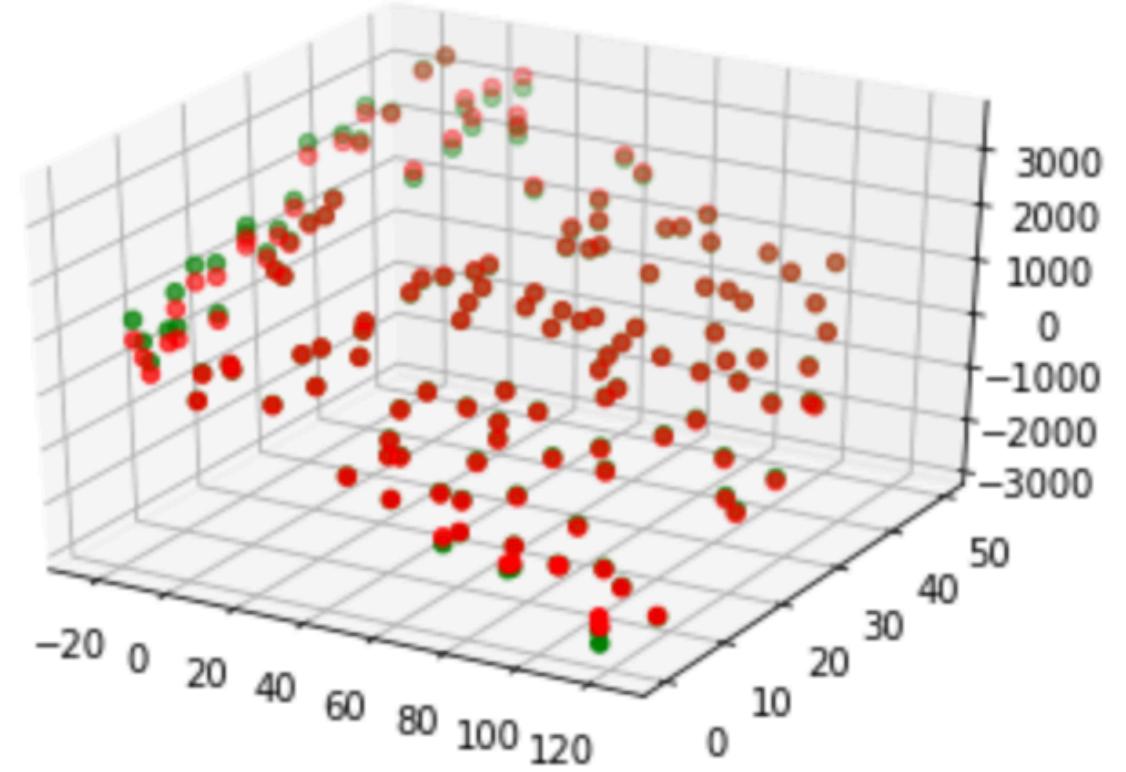
$$P_{y\phi} = 0.6490587313264213$$

Como se puede observar según los coeficientes de correlación de Pearson, las relaciones lineales entre cada una de las covariables y la variable respuesta son considerable.

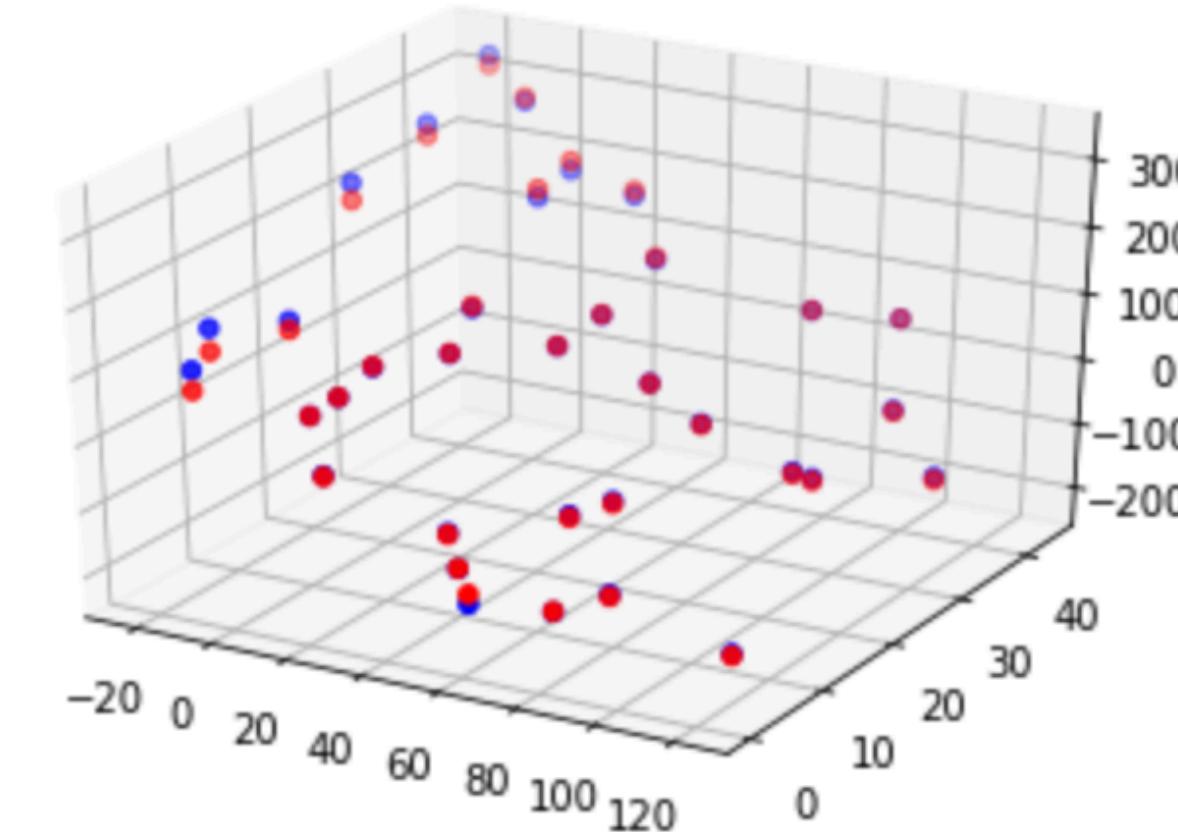
## Conjuntos de datos



Entrenamiento



Prueba



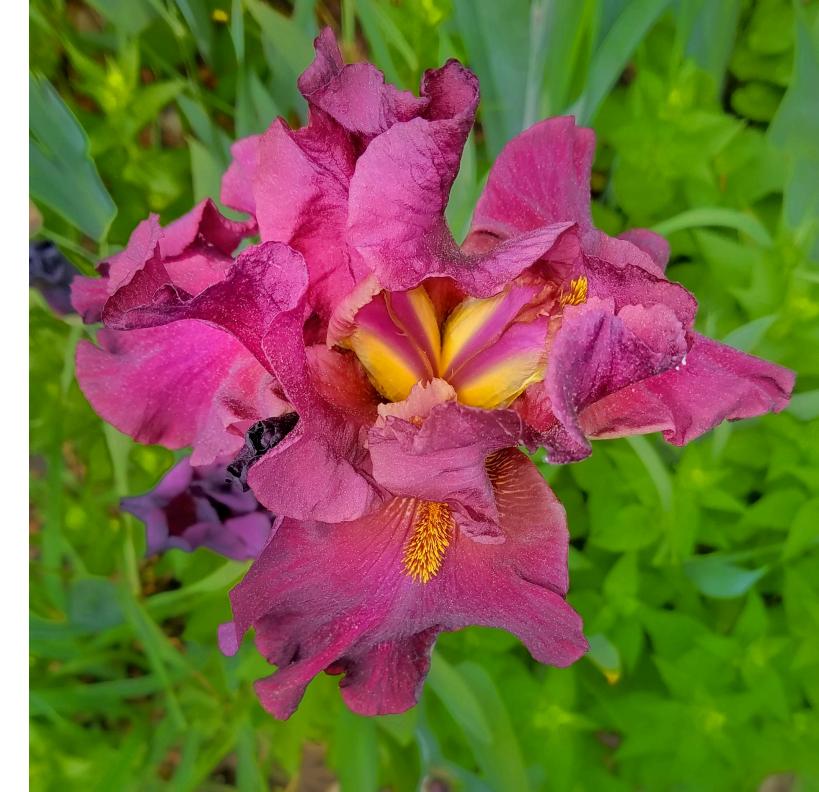
<b>Score del conjunto de entrenamiento:</b>	<b>0.9946045007351835</b>
<b>Score del conjunto de prueba:</b>	0.9931586091143421
<b>el ECM</b>	del conjunto de prueba es 660.8743818351024

# Ejemplo Clasificación

**Se va a considerar el conjunto de datos "Iris", un conjunto de datos multivariantes creado por Ronald Fisher el cual contiene 50 muestras de cada una de tres especies de Iris (Iris setosa, Iris virginica e Iris versicolor). Se midió cuatro rasgos de cada muestra: el largo y ancho del sépalo y pétalo, en centímetros.**

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	Species_cat
0	5.1	3.5	1.4	0.2	Iris-setosa	1
1	4.9	3.0	1.4	0.2	Iris-setosa	1
2	4.7	3.2	1.3	0.2	Iris-setosa	1
3	4.6	3.1	1.5	0.2	Iris-setosa	1
4	5.0	3.6	1.4	0.2	Iris-setosa	1
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica	0
146	6.3	2.5	5.0	1.9	Iris-virginica	0
147	6.5	3.0	5.2	2.0	Iris-virginica	0
148	6.2	3.4	5.4	2.3	Iris-virginica	0
149	5.9	3.0	5.1	1.8	Iris-virginica	0

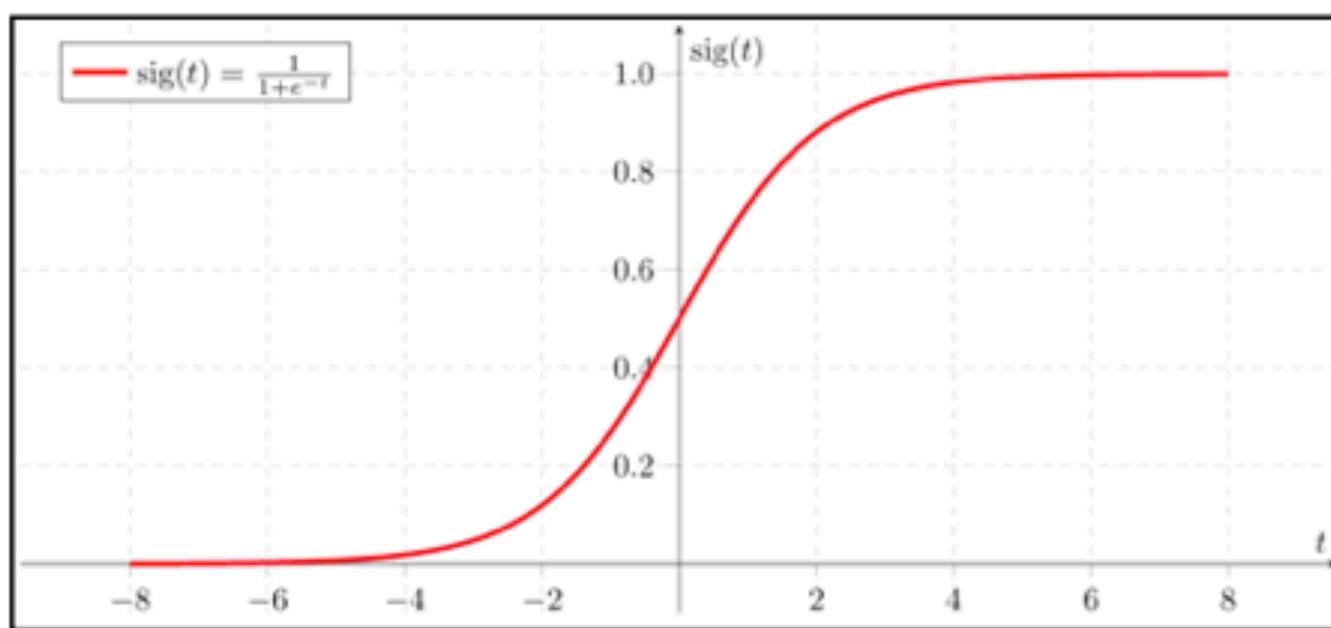
150 rows x 6 columns



Se va a utilizar un modelo de redes neuronales (perceptrón multicapa), con una sola capa para poder predecir la especie de una planta (Iris-setosa u otra), esto según las características medidas de su pétalo y sépalo

**Se va a construir paso a paso una clase que defina el perceptrón a utilizar con el fin de entender el paso a paso que se realiza dentro de este tipo de modelo, teniendo una taza de aprendizaje de 0.009 , una función de activación sigmoide, una tolerancia del  $1e - 10$  y una función de costo logarítmica.**

### SIGMOIDE



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

Para un mejor ajuste se va a dividir el conjunto de datos en entrenamiento y prueba, el primero con el fin de que el modelo aprenda y el segundo para comprobar la capacidad predictiva de este.

```
Costo inicial = 0.6931471805599453
Costo final = 0.06318518750775518
Número de epochs = 1000
Pesos estimados para el modelo
[[ 0.23201916]
 [ 0.87625961]
 [-1.38301407]
 [-0.63738961]]
Sesgo estimado para el modelo
0.16149407399667534
```

para visualizar mejor el ajuste de este modelo se muestra la matriz de confusión respectiva del conjunto de prueba:

