

# Redes Neuronales Artificiales

Artificial Neural Networks

Andrea Torres Calderón

[a.torres.c@tec.mx](mailto:a.torres.c@tec.mx)

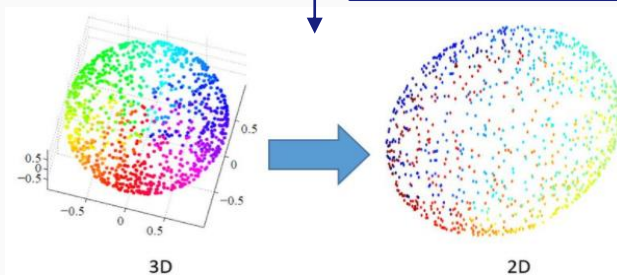
2023

# RECAPITULACIÓN

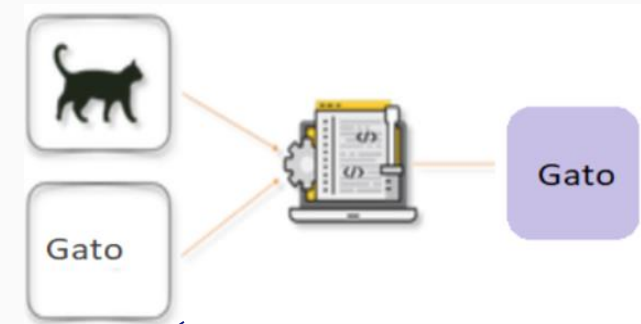
Aprendizaje no  
supervisado



Técnicas de  
reducción dimensional



Aprendizaje  
supervisado



Clasificación



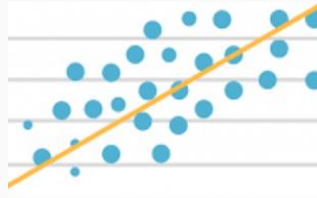
Regresión



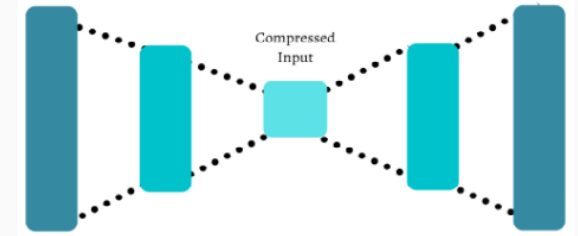
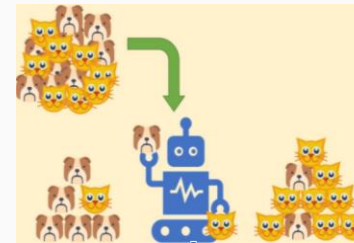
# TIPOS DE PROBLEMAS EN DONDE SE PUEDEN USAR LAS REDES NEURONALES

Una red neuronal puede resolver problemas de:

- Aprendizaje supervisado
- Aprendizaje no supervisado



Invierno de la IA

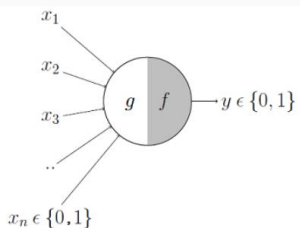


Las redes neuronales son modelos computacionales que tratan de emular el comportamiento del cerebro humano, caracterizado por el aprendizaje a través de la experiencia.

# HISTORIA DE LAS REDES NEURONALES

Origen de las  
redes neuronales

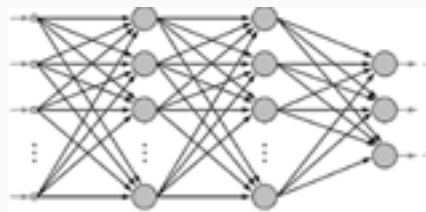
**Warren McCulloch  
& Walter Pitts**  
1943



Perceptrón sólo puede  
resolver problemas de  
clasificación limitados

**Marvin Minsky**

1969



Invierno de la IA

SVM

Deep Learning

**Geoffrey Hinton**

2006

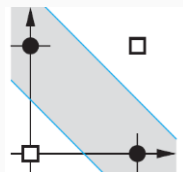


1940 1950 1960 1970 1980 1990 2000 2010 2020

1958

Red Perceptrón

**Frank Rosenblatt**



1985

Algoritmo de  
Backpropagation

**David Rumelhart &  
James McClelland**



2021

**¿Llegaron para  
quedarse?**

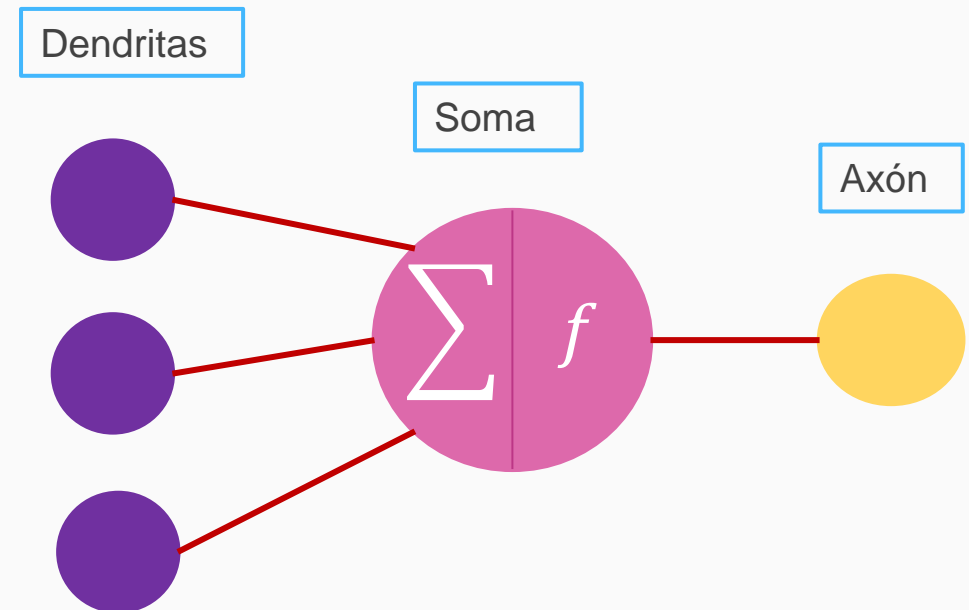
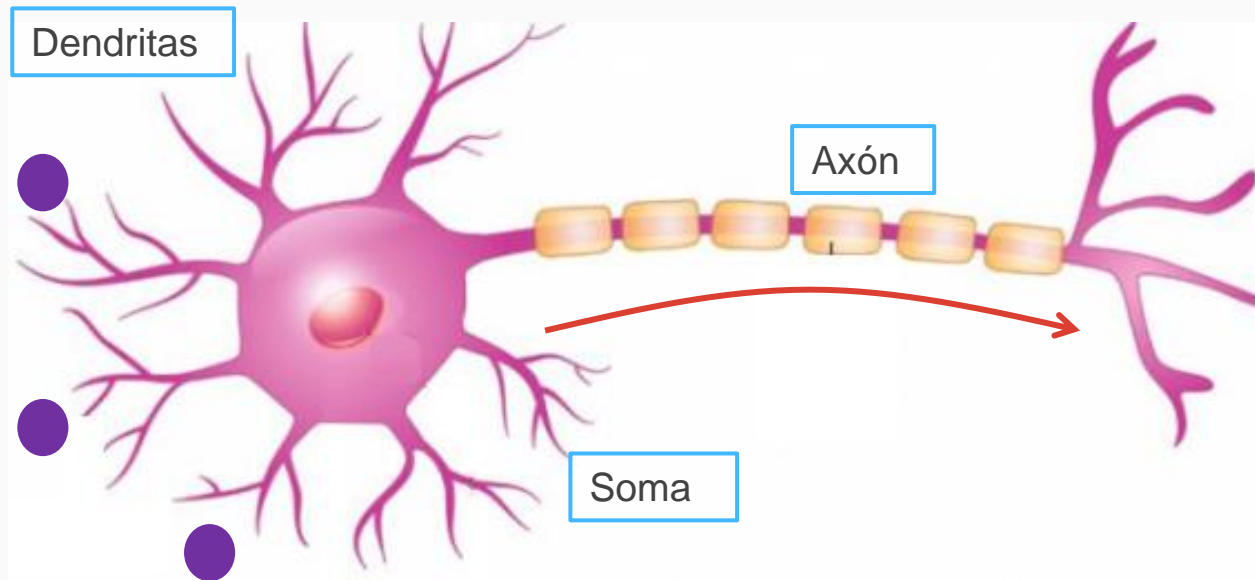
- Conjuntos de datos más grandes
- Mayor equipo de computo
- Nuevas arquitecturas

# INSPIRACIÓN BIOLÓGICA

Las redes neuronales artificiales están inspiradas en la estructura de la red neuronal biológica. Se constituyen principalmente de tres componentes:

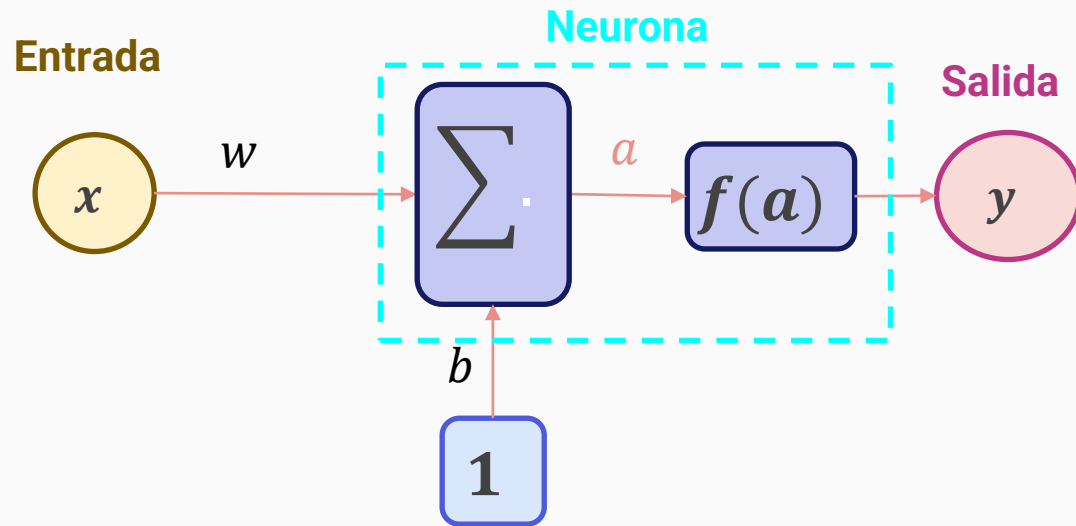
- Dendritas
- Cuerpo de la célula o soma
- Axón

Invierno de la IA



# NEURONA ARTIFICIAL

Una neurona artificial es una función matemática que toma una o más valores de entrada y regresa un valor numérico.

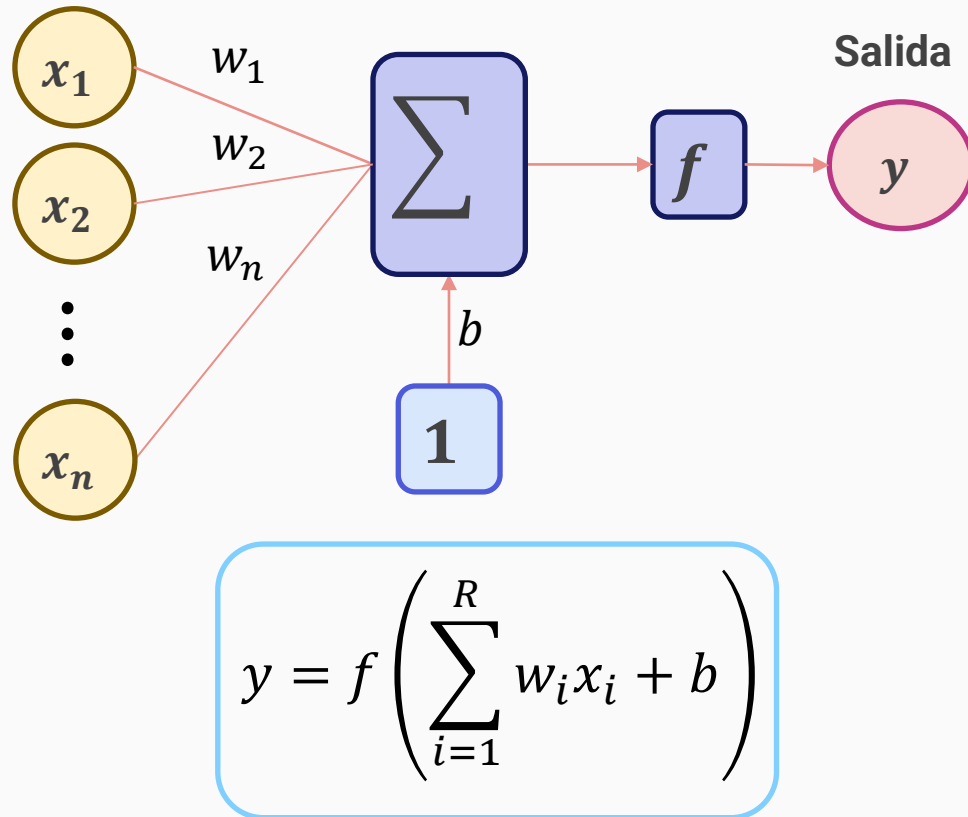


- $x$  Representa la entrada de los datos.
- $w$  Los pesos son valores que representan la intensidad de las entradas o la fuerza entre las conexiones entre las neuronas.
- $b$  Bias es una entrada artificial que siempre tiene por entrada el valor de 1.
- $f(n)$  Función de activación determina cuando se produce una señal de salida.

$$y = f(wx + b)$$

# NEURONA ARTIFICIAL

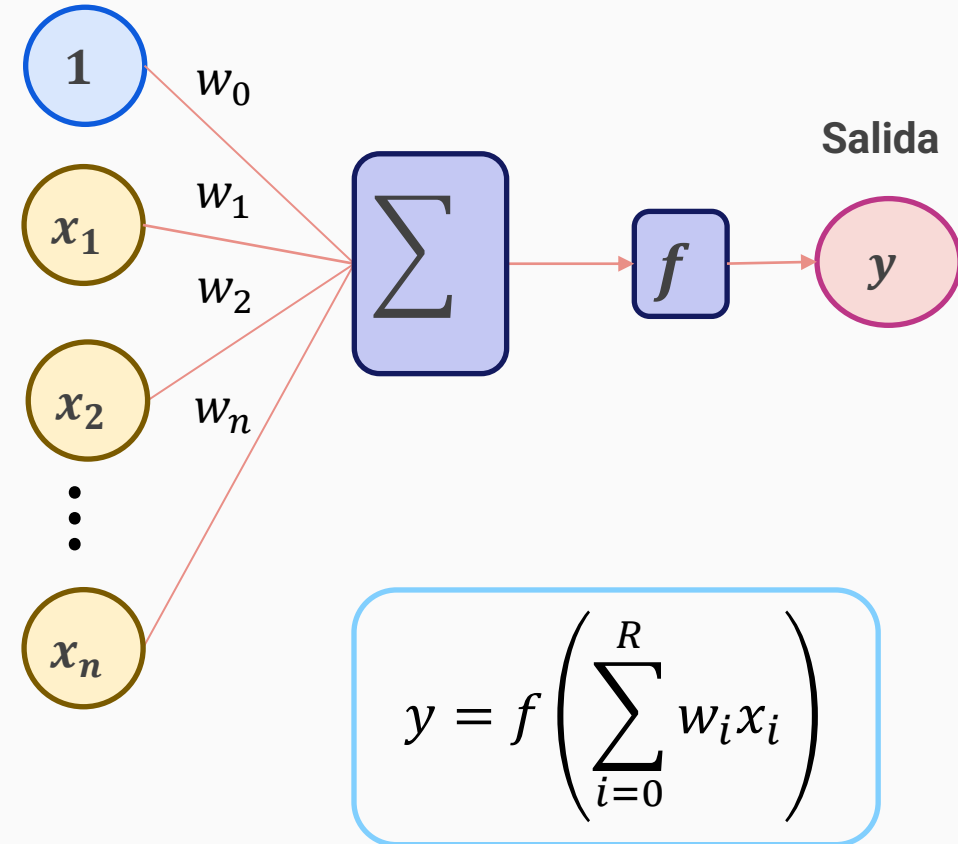
Entrada



$$y = f(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b)$$

Matricialmente  $y = f(W\mathbf{x} + \mathbf{b})$

Entrada



$$y = f(w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n)$$

Matricialmente  $y = f(W\mathbf{x})$

# FUNCIÓN DE ACTIVACIÓN

Una Función de activación determina la salida que generará una neurona en función de su entrada

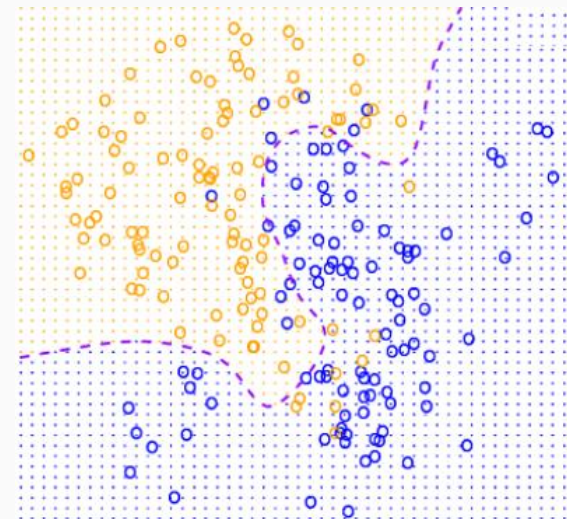
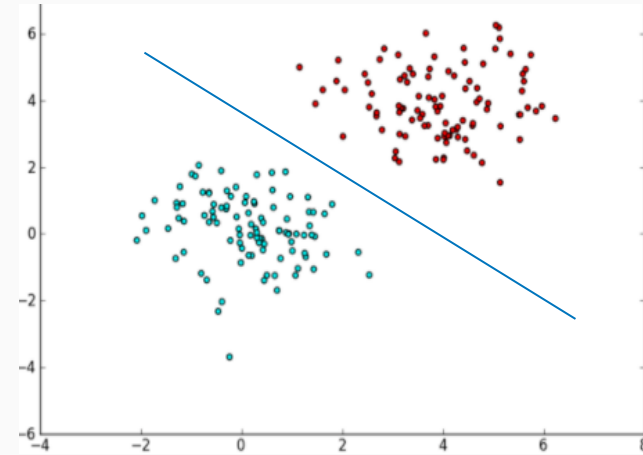
$$y = f(wx + b)$$

¿Qué pasaría si no utilizamos estas funciones de activación?

**La red se comportaría como una función lineal**

## Características:

- Introducir propiedades no lineales a la red, que permitan fronteras de decisión más complejas
- Se elige para satisfacer alguna especificación del problema
- Deben ser diferenciables



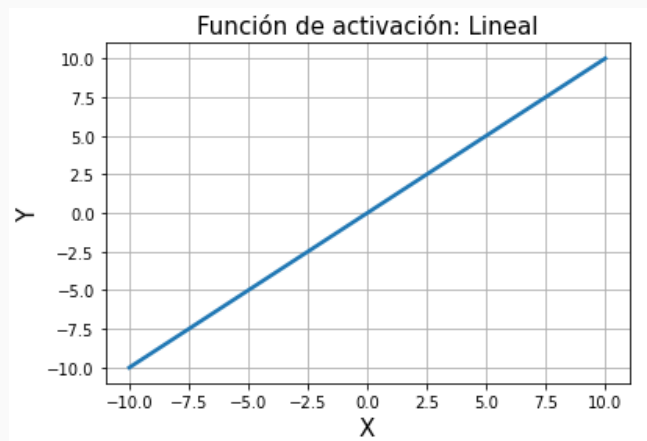


# FUNCIÓN DE ACTIVACIÓN

## Función lineal

- También conocida como función *identidad*
- La función de activación lineal no cambia la suma ponderada de la entrada

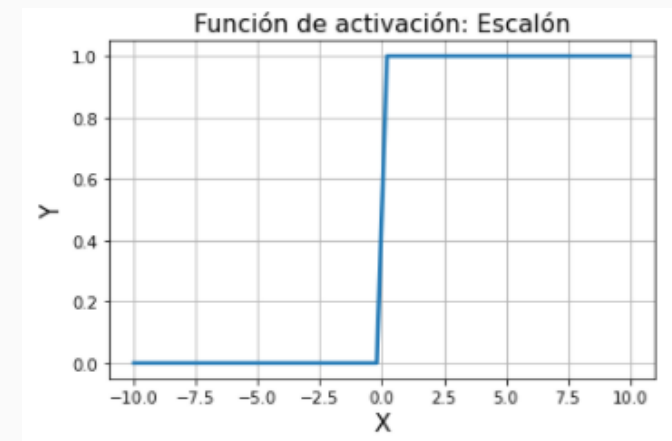
$$f(x) = x$$



## Función escalón

- Salida binaria

$$f(x) = \begin{cases} 0 & \text{para } x < 0 \\ 1 & \text{para } x \geq 0 \end{cases}$$

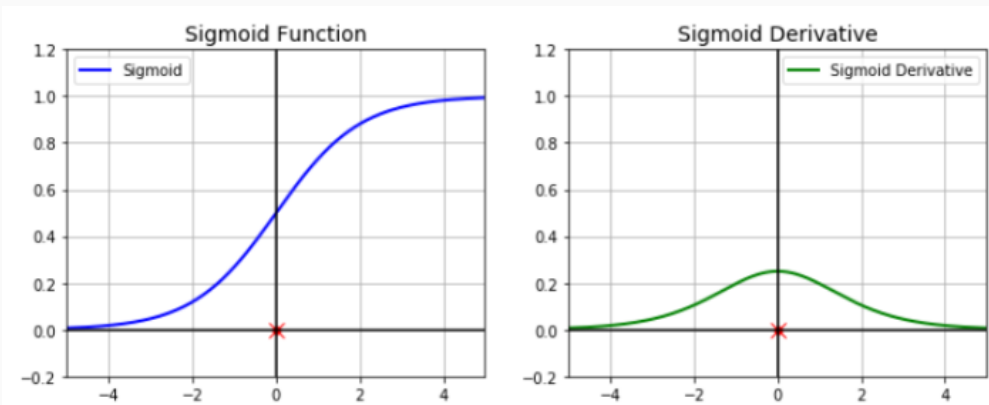


# FUNCIÓNES DE ACTIVACIÓN

## Función sigmoide

- También conocida como función **logística**
- La función toma como entrada cualquier valor real y a la salida regresa valores entre 0 y 1

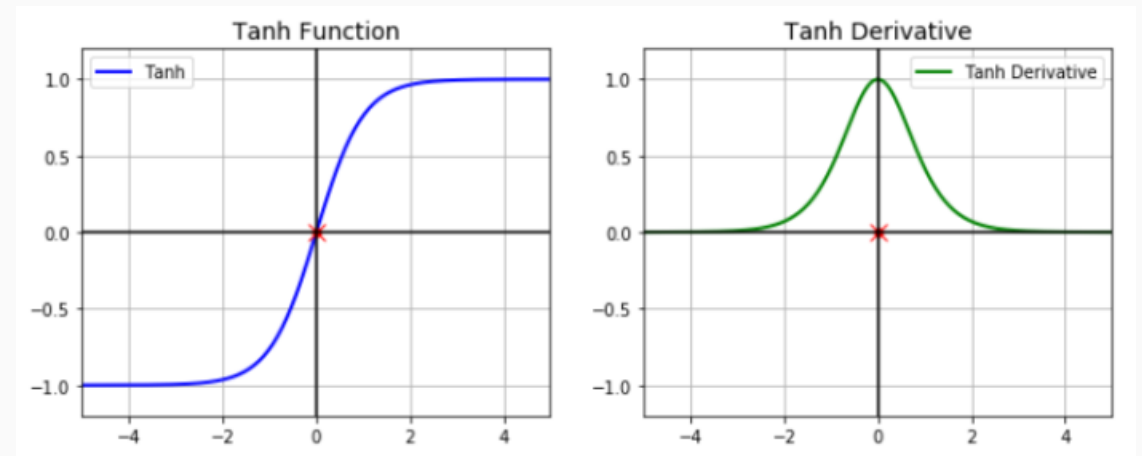
$$f(x) = \frac{1}{1 + e^{-x}}$$



## Función tangente hiperbólica

- También conocida como función **tanh**
- La función toma como entrada cualquier valor real y a la salida regresa valores entre -1 y 1

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

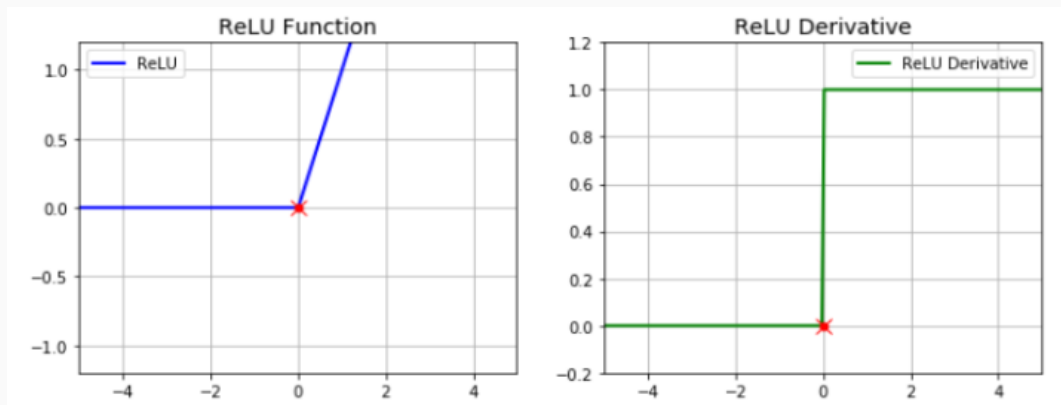


# FUNCIONES DE ACTIVACIÓN

## Función de activación lineal rectificada

- También conocida como función **ReLU**
- La función utilizada más común para capas ocultas
- Computacionalmente más eficiente

$$f(x) = \begin{cases} 0 & \text{para } x \leq 0 \\ x & \text{para } x > 0 \end{cases}$$



## Función de activación Leaky ReLU

- Variación de **ReLU**, contiene una pequeña pendiente positiva en el área negativa

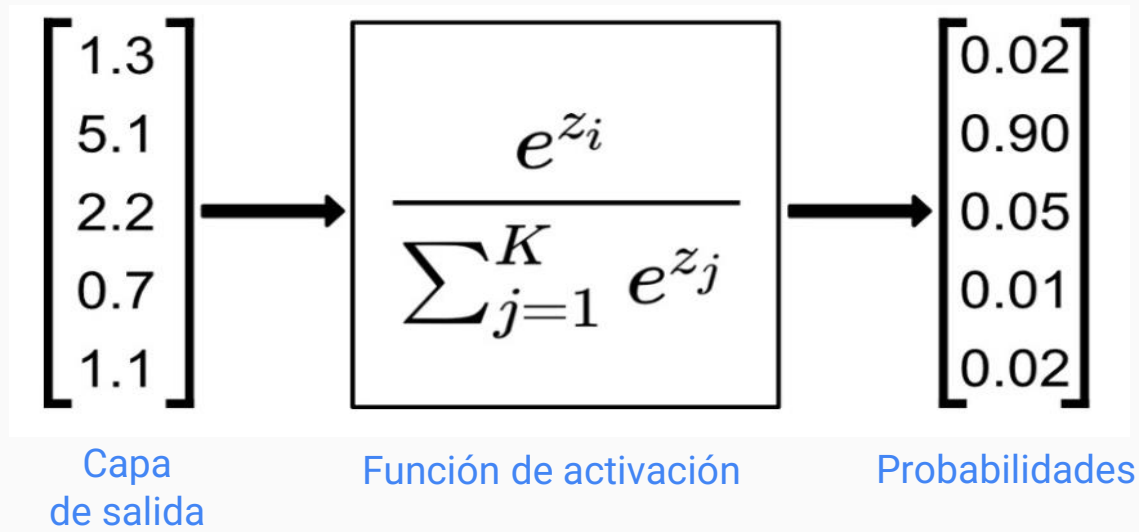
$$f(x) = \begin{cases} \alpha x & \text{para } x < 0 \\ x & \text{para } x \geq 0 \end{cases}$$



# FUNCIÓNES DE ACTIVACIÓN

## Función softmax

- Convierte un vector de números en un vector de probabilidades
- Cada valor de la salida de la función se interpreta como la probabilidad de pertenecer a cada clase



$$f(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Donde:

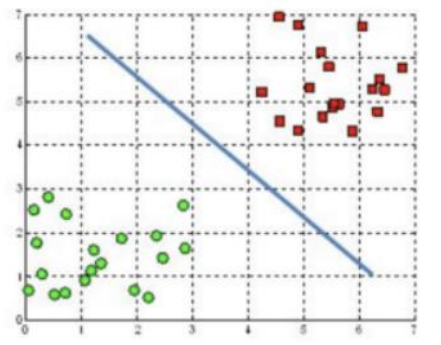
- $\vec{z}$  El vector de entrada a la función softmax, compuesto por  $(z_0, \dots, z_k)$
- $e^{z_i}$  Función exponencial que se aplica a cada elemento del vector de entrada.
- $K$  El número de clases en el multclasificador
- $\sum_{j=1}^K e^{z_j}$  Término de normalización. Asegura que todos los valores de salida estén en el rango  $(0, 1)$

# RED PERCEPTRÓN

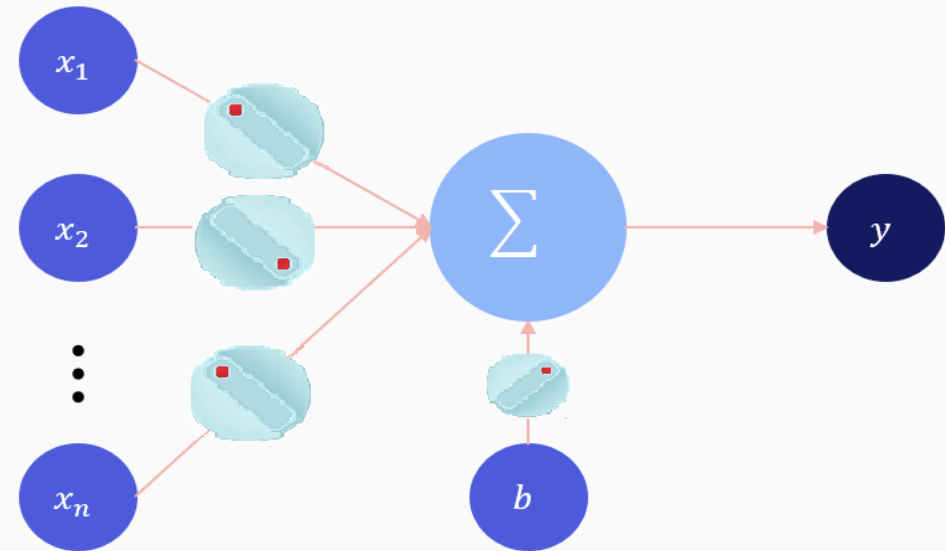
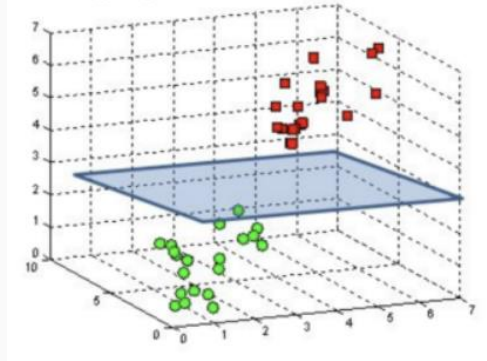
El perceptrón es la forma más simple de una red neuronal.

El objetivo del perceptrón es **encontrar el hiperplano** capaz de separar correctamente un conjunto de datos que sean linealmente separables.

$\mathbb{R}^2$



$\mathbb{R}^3$

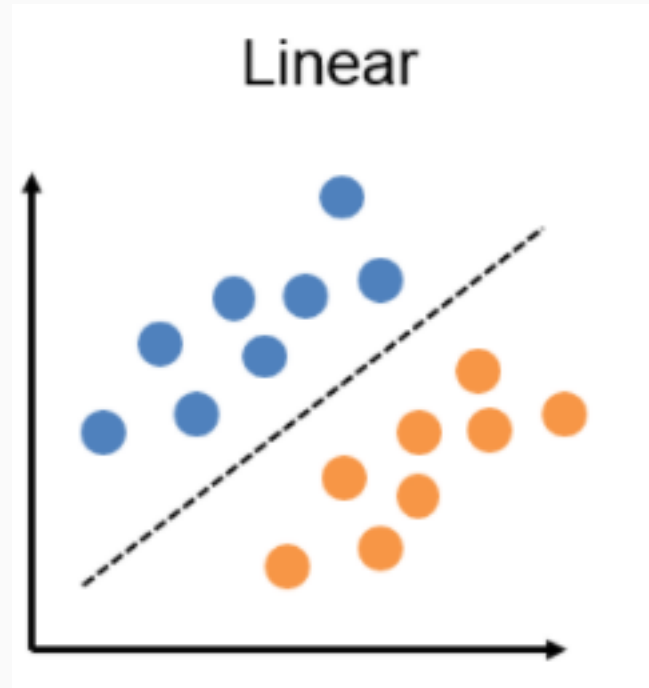


- **Aprendizaje** : Modificar los valores de  $b$  y  $w$
- **Regla de aprendizaje**: Realizar un ajuste automático de los parámetros de la red

# LIMITACIONES

## ¿Qué tipo de problemas puede resolver el perceptrón?

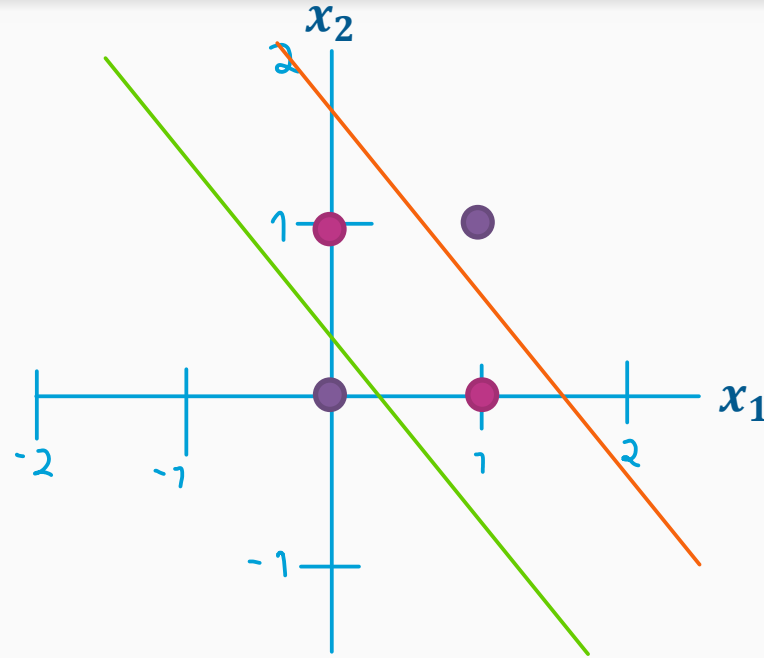
El perceptrón solo puede dividir el conjunto de datos en dos, por medio de una frontera de decisión lineal.



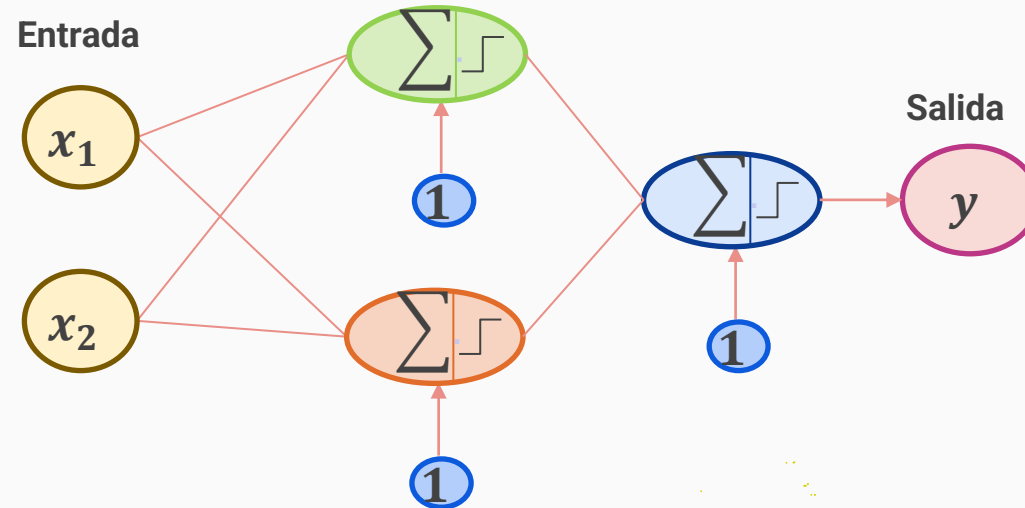
# RED PERCEPTRÓN MULTICAPA

Ejemplo: Compuerta XOR

$x_1$	$x_2$	$t$
0	0	0
0	1	1
1	0	1
1	1	0



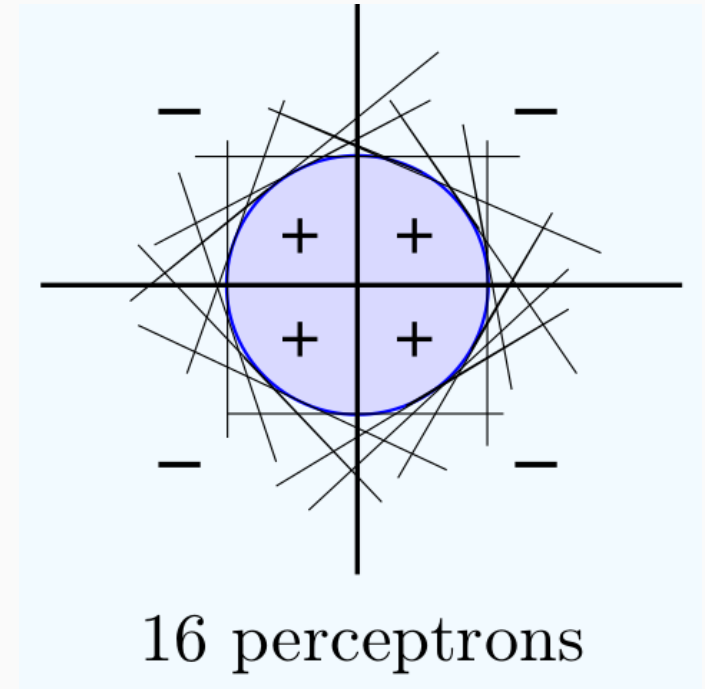
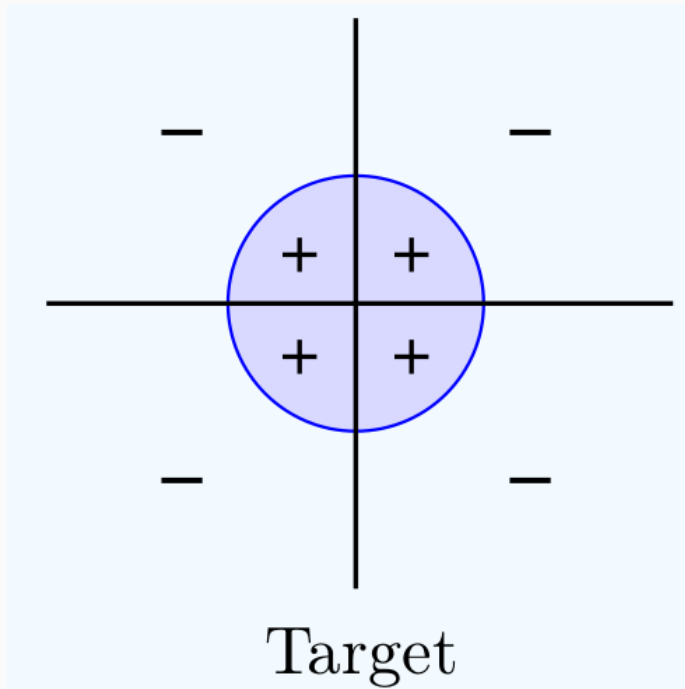
Con una neurona  
¡no es posible resolver  
este problema!



Con más neuronas podemos  
resolver problemas más complejos

# MULTILAYER PERCEPTRÓN

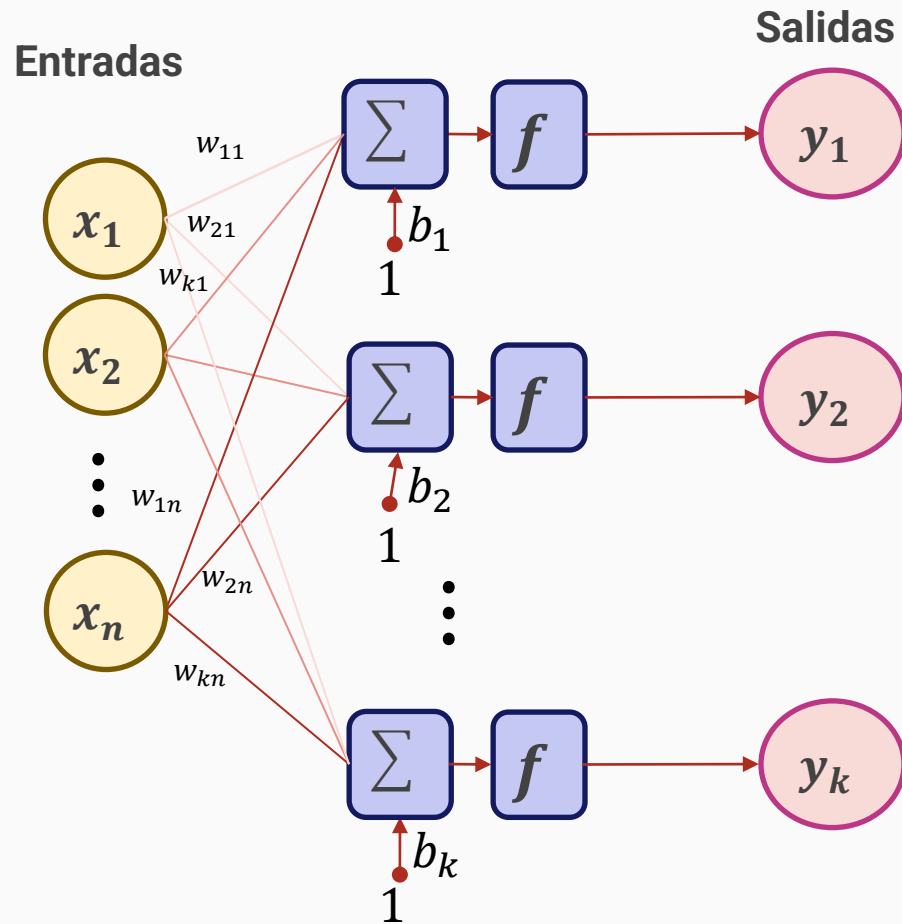
Perceptrón Multi Capa (MLP)  
para funciones no lineales





# ARQUITECTURA DE UNA NEURONA ARTIFICIAL

Una red de una capa con una o más entradas se puede describir de la siguiente manera:

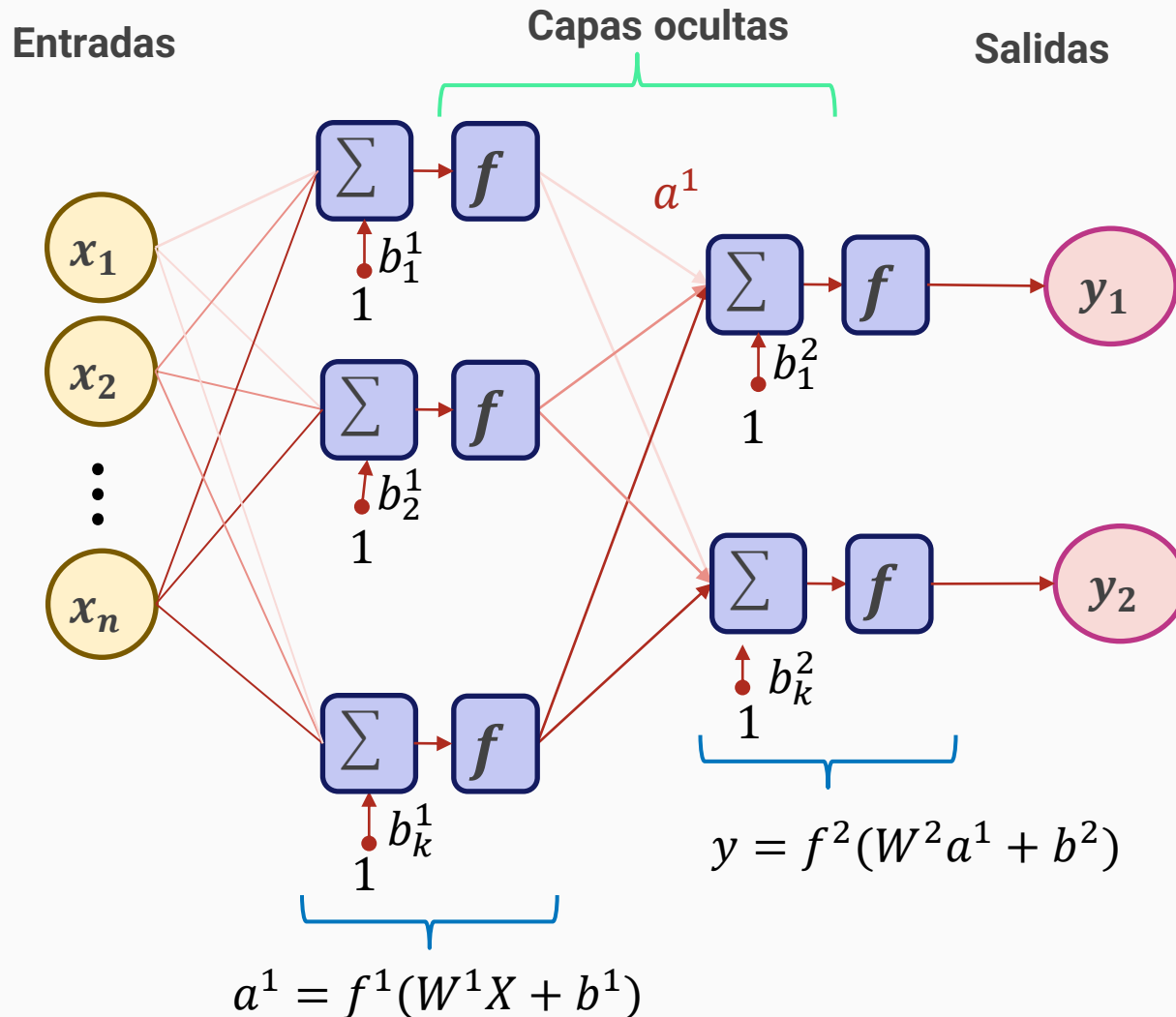


- Cada una de las entradas se conectan con cada una de las neuronas.
- Cada neurona tiene una salida, un bias y una función de activación.

$$y = f(Wx + b)$$

# ARQUITECTURA DE UNA NEURONA ARTIFICIAL

## Redes neuronales Multicapa

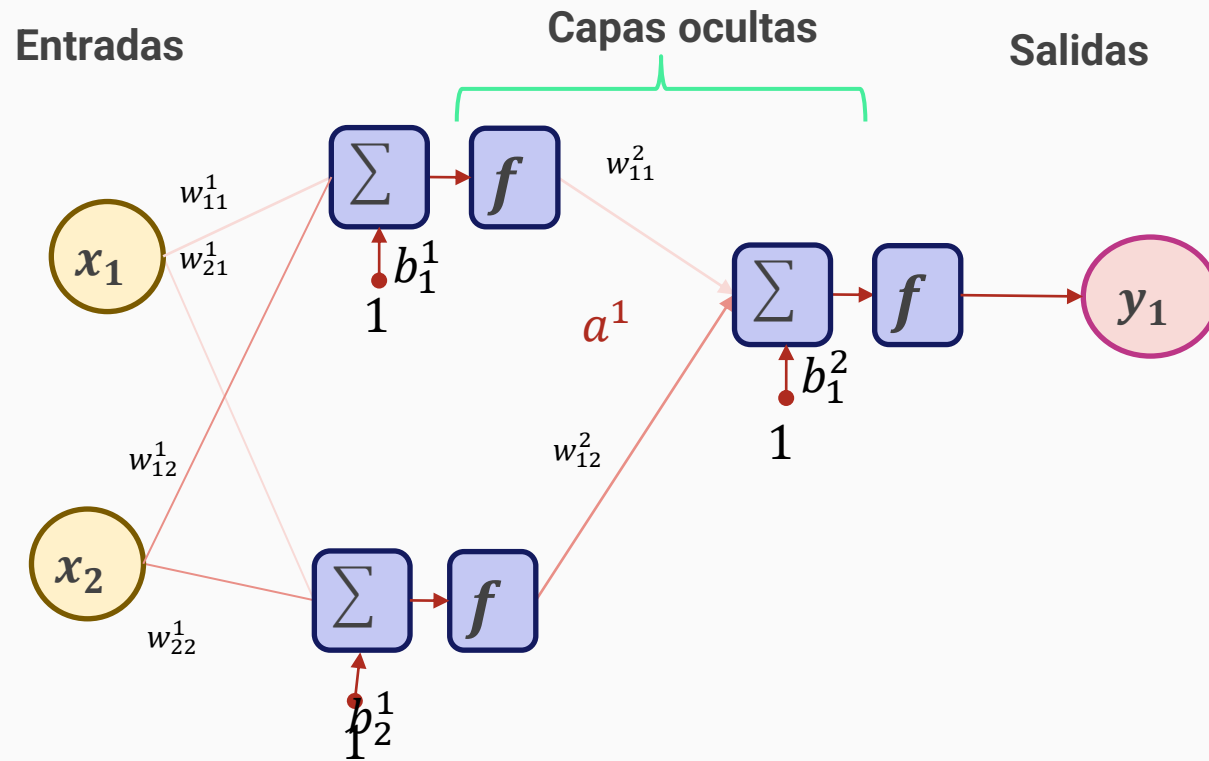


- **Capas completamente conectadas**, significa que cada nodo en una capa se conecta con todos los nodos de la siguiente capa.
- La capa que tiene la salida de la red neuronal, se llama **capa de salida**. Las otras capas se les denomina como **capas ocultas**.
- Una red neuronal multicapa tiene una o más **capas ocultas**.

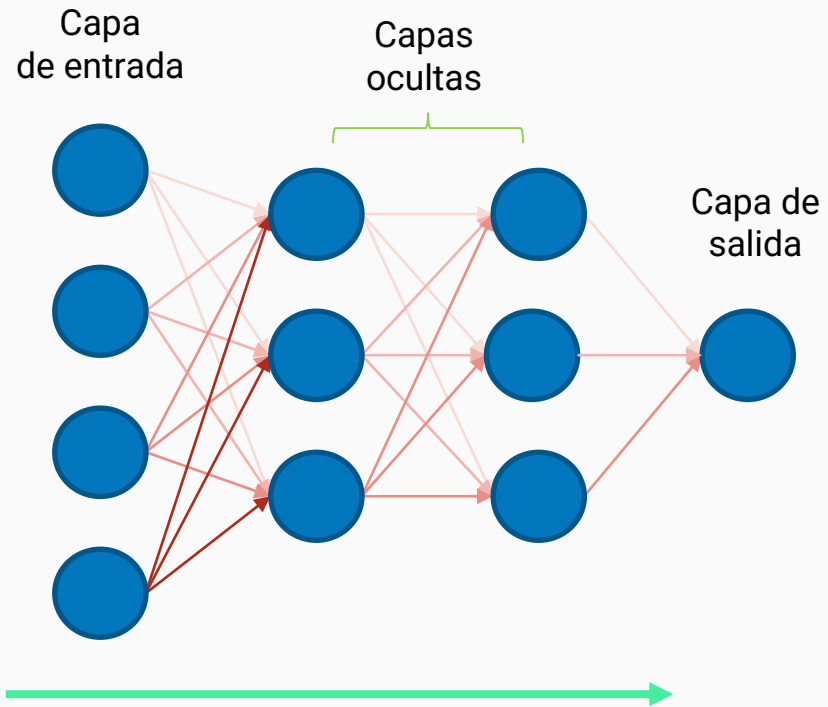
$$y = f^2(W^2a^1 + b^2)$$
$$y = f^2(W^2(f^1(W^1x + b^1)) + b^2)$$

# ARQUITECTURA DE UNA NEURONA ARTIFICIAL

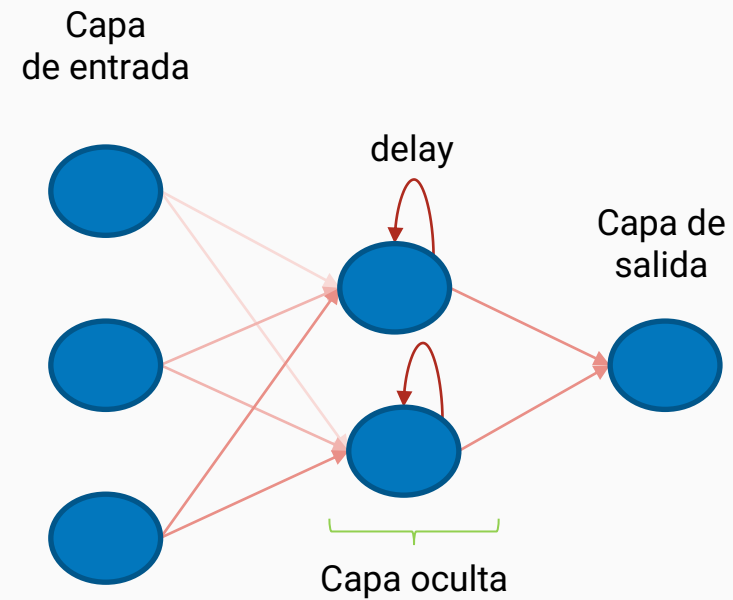
## Redes neuronales Multicapa



# DIRECCIÓN DEL FLUJO DE INFORMACIÓN

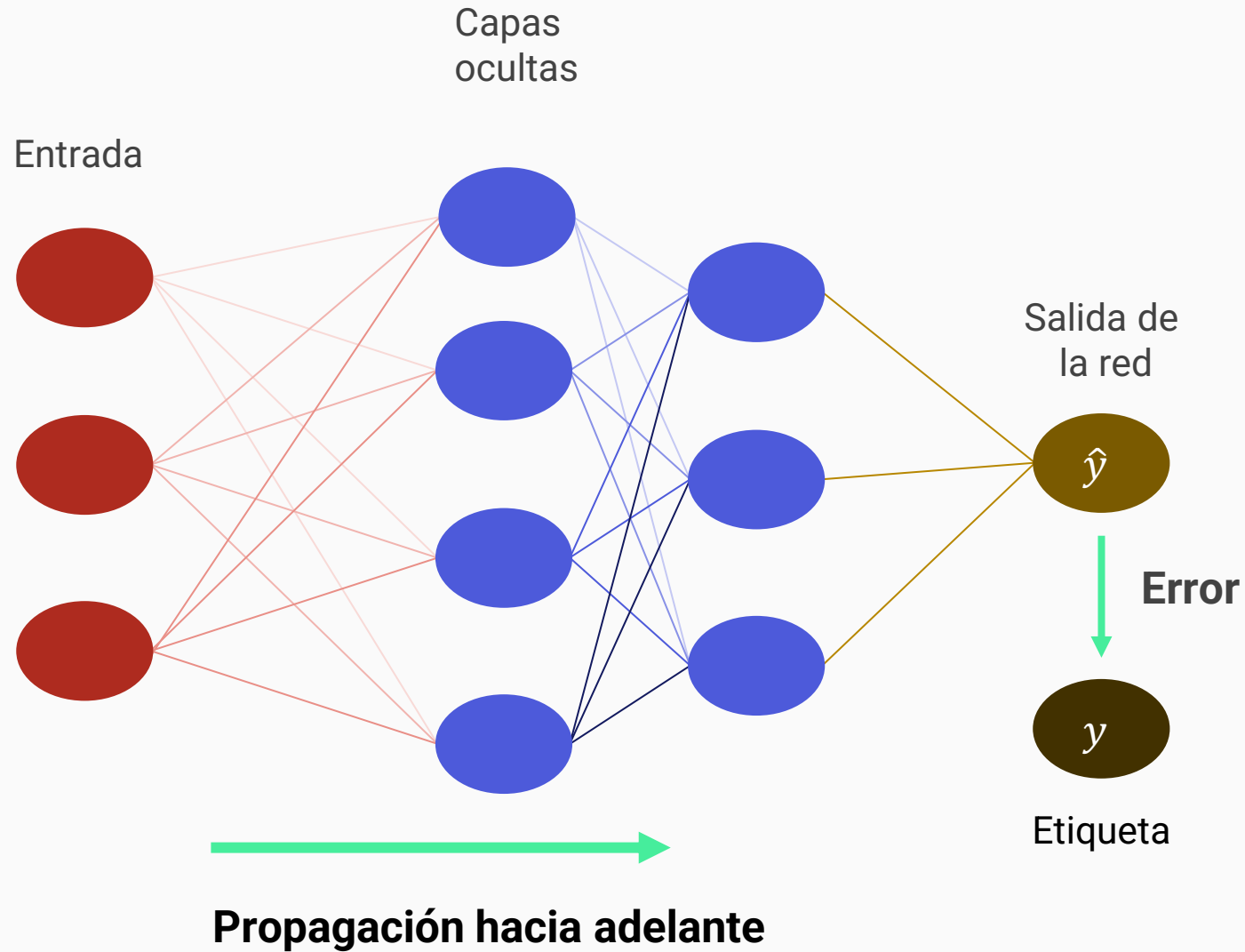


Redes feedforward



Redes recurrentes

# FUNCIÓN DE COSTO



## Época

Propagación hacia adelante de cada uno de los datos del conjunto de datos de entrenamiento y realizar el ajuste de los parámetros de la red.

Función objetivo o  
función de costo

# FUNCIÓN DE COSTO

## Error cuadrático medio (Mean Squared Error )

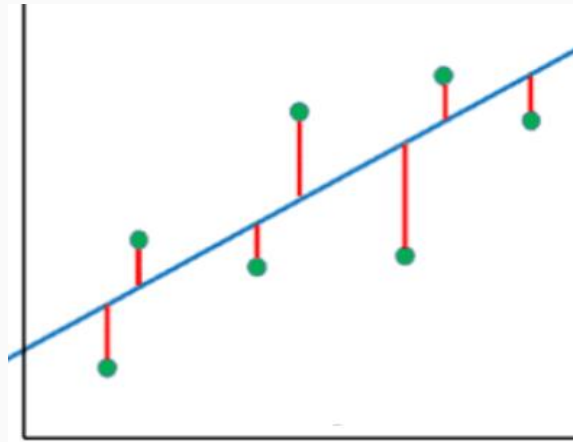
- La función de costo más utilizada para los problemas de regresión
- Penaliza desviaciones grandes
- Sensible a observaciones atípicas (outliers)

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

## Error absoluto medio (Mean Absolut Error )

- Robustez ante la presencia de observaciones atípicas
- Asigna valores sin ponderación

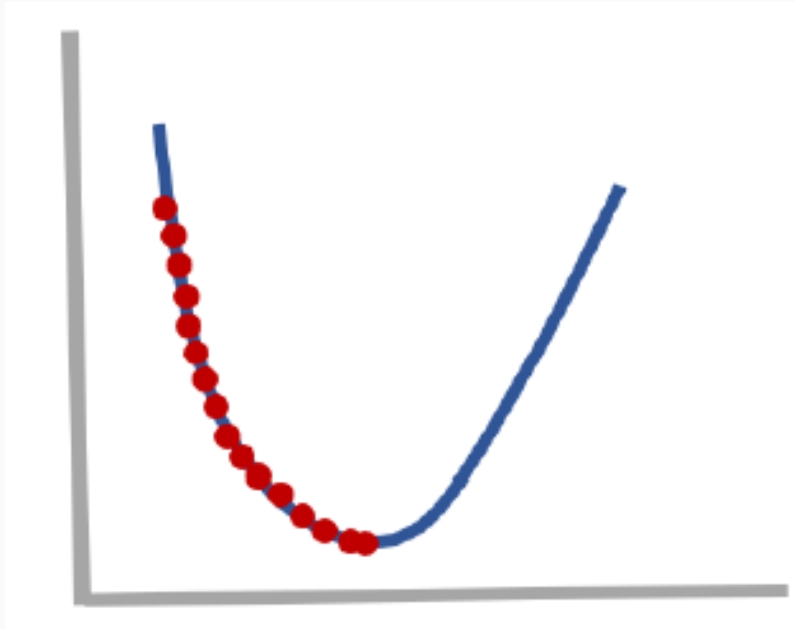
$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$



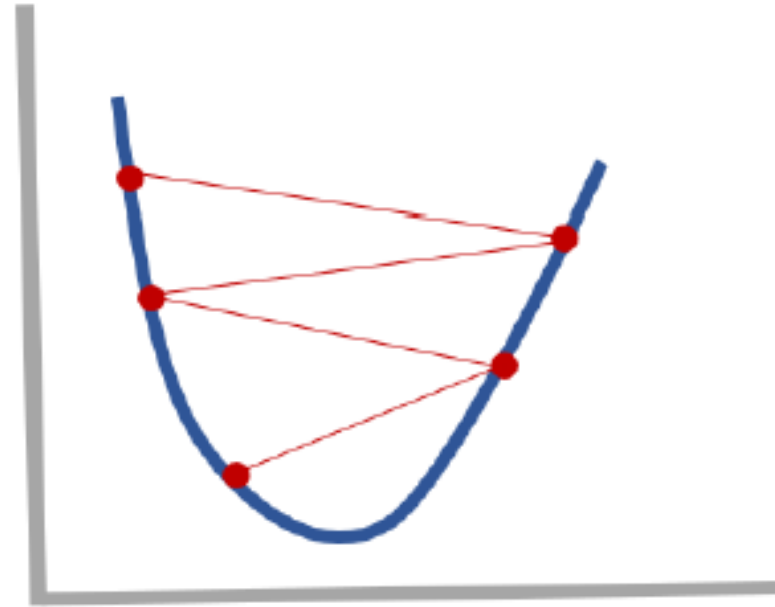
# GRADIENTE DESCENDENTE

**Factor de aprendizaje:** Determina la velocidad con la que se actualizan los parámetros de una red.

- Si el **factor de aprendizaje es muy pequeño**, requerirá de mas épocas de entrenamiento por lo que pueden tardar mucho tiempo en encontrar el mínimo, ya que hacen que los parámetros cambien lentamente.
- Si el **factor de aprendizaje es demasiado grande**, puede hacer que el modelo converja demasiado rápido y en ocasiones provoca oscilaciones que dificultan o incluso imposibilitan encontrar el mínimo.



Factor de aprendizaje muy pequeño



Factor de aprendizaje muy grande

# BACKPROPAGATION

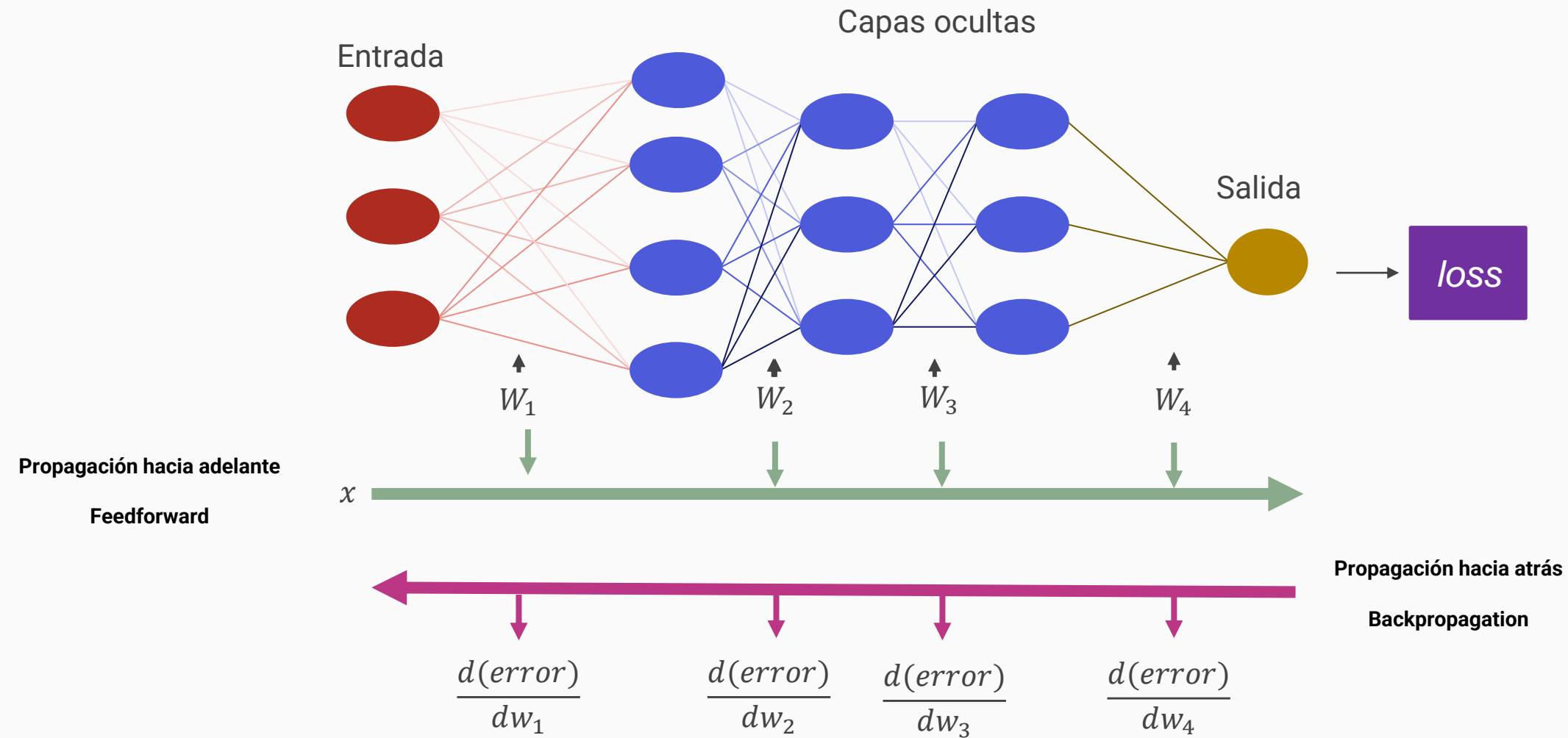


## Algoritmo:

1. Se inicializan los parámetros aleatoriamente
2. *for*  $i = 1$  to número épocas:
  - se propagan los pesos hacia adelante
  - se compara la salida de la red con la etiqueta (función de costo)
  - se calcula el gradiente para actualizar los parámetros



# BACKPROPAGATION



Backpropagation es un algoritmo que ejecuta Gradiente Descendente para ajustar todos los pesos en la red

# Terminología de las redes

- Pesos
  - Bias
- } **Parámetros**

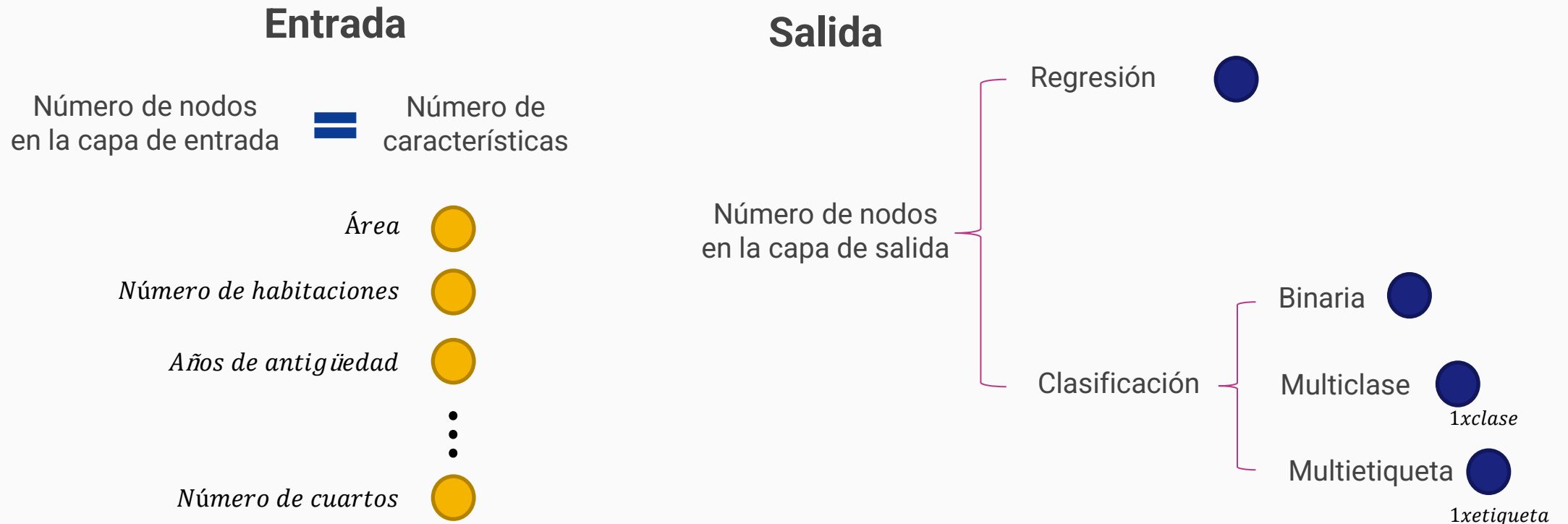
- Número de nodos
  - Número de capas
- } **Arquitectura**

- Función de activación
  - Función de costo
  - Optimizador
- } **Hiperparámetros**

## ¿Cómo elegir la arquitectura de la red neuronal?

- **Número de nodos en la capa de entrada y en la capa de salida**

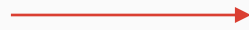
Está predeterminado por el problema que se quiere resolver



## ¿Cómo elegir la arquitectura de la red neuronal?

- Número de nodos en las capas ocultas

¡No hay forma de saberlo!

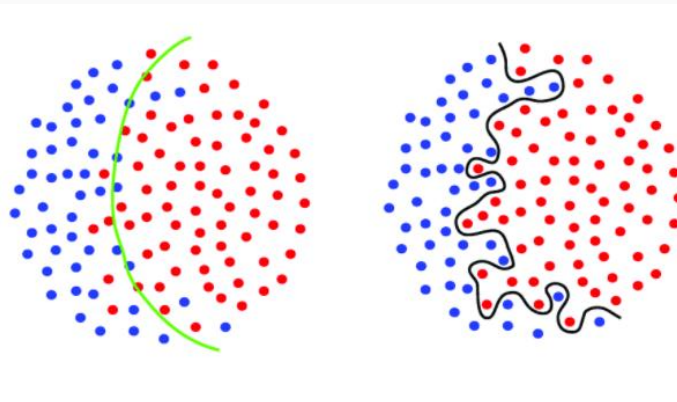


Área activa de investigación

- Número de capas

No existe una regla de dedo para determinar el número de capas ocultas  $\#capas \leq 3$

**Consideración:** Entre más capas ocultas, la red puede predecir funciones más complejas



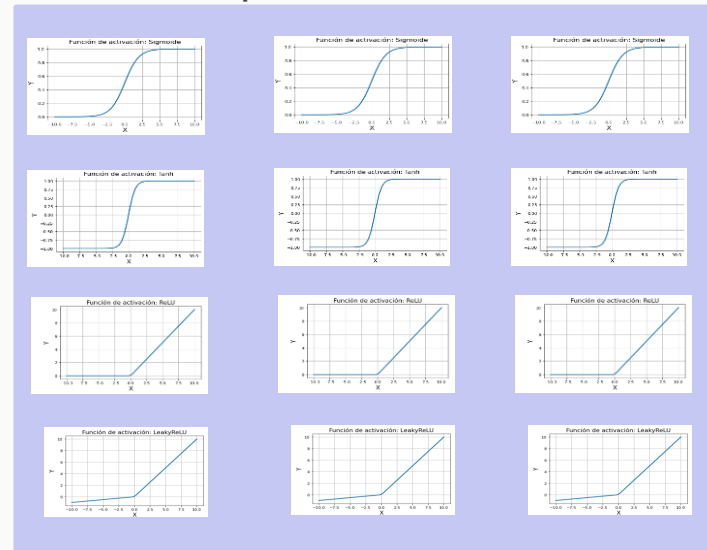
## ¿Cómo elegir las funciones de activación?

- **Funciones de activación en las capas ocultas**

Típicamente se utilizan funciones de activación no lineales en las capas ocultas. Esto permite al modelo a aprender funciones más complejas

Entre las funciones de activación más comunes en las capas ocultas:

- Sigmoide
- Tanh
- ReLU
- LeakyReLU



Capas ocultas

Lo más común es que todas neuronas que están en las capas ocultas tengan la misma función de activación.

## ¿Cómo elegir las funciones de activación?

### • Funciones de activación en las capa de salida

La función de activación en la capa de salida se debe escoger con base al tipo de predicción del problema que se está resolviendo.

### ❖ *Clasificación*

Funciones de activación:

- Sigmoid
- Softmax

✓ **Clasificación binaria.** Sigmoid

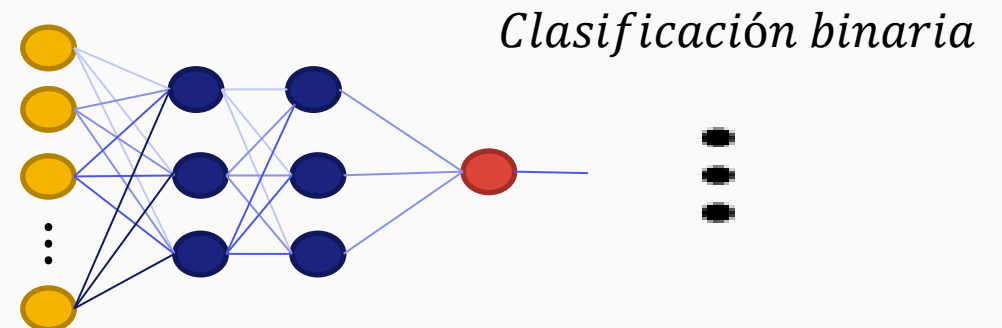
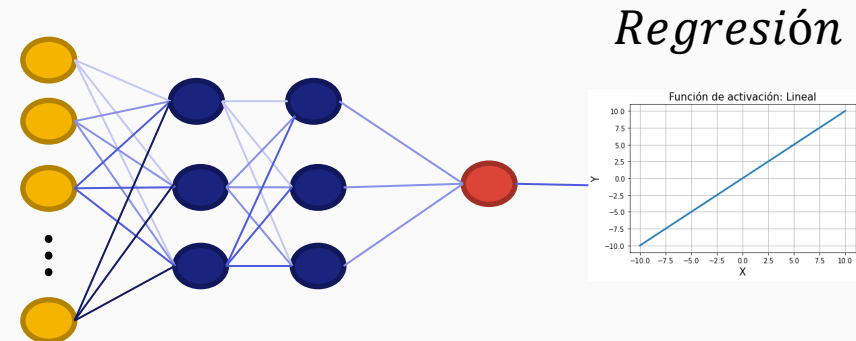
✓ **Clasificación multiclase.** Softmax

✓ **Clasificación multietiqueta.** Un nodo por cada etiqueta y una función de activación sigmoid

### ❖ *Regresión*

Funciones de activación:

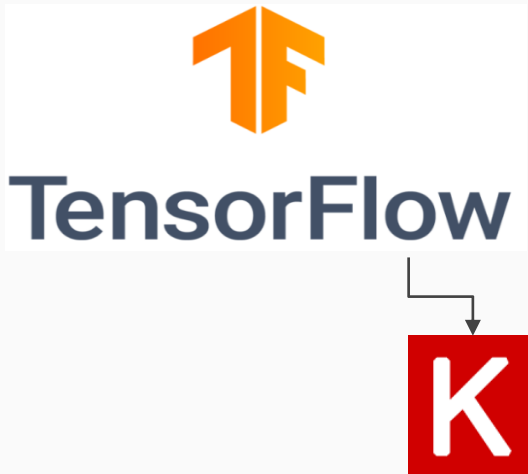
- Linear
- ReLU



# RESUMEN

Problema		# Nodos en la capa de entrada	# Capas ocultas	# Nodos en las capas ocultas	Función de activación en capas ocultas	# Nodos en la capa de salida	Función de activación en capa de salida	Función de costo
Regresión	-	1 por cada atributo	Depende del problema	Depende del problema	ReLU LeakyReLU	1	ReLU Linear	MSE MAE Huber
Clasificación	Binaria	1 por cada atributo	Depende del problema	Depende del problema	ReLU LeakyReLU	1	Sigmoide	Entropía cruzada binaria
	Multiclase	1 por cada atributo	Depende del problema	Depende del problema	ReLU LeakyReLU	1 por cada clase	Softmax	Entropía cruzada
	Multietiqueta	1 por cada atributo	Depende del problema	Depende del problema	ReLU LeakyReLU	1 por cada etiqueta	Sigmoide	Entropía cruzada

# FRAMEWORK



Instalación CPU en ambiente virtual 

```
(v_env)$ pip install tensorflow
```

Python 3.5 a 3.8  
Versión de tensorflow >2.0

Colaboratory 

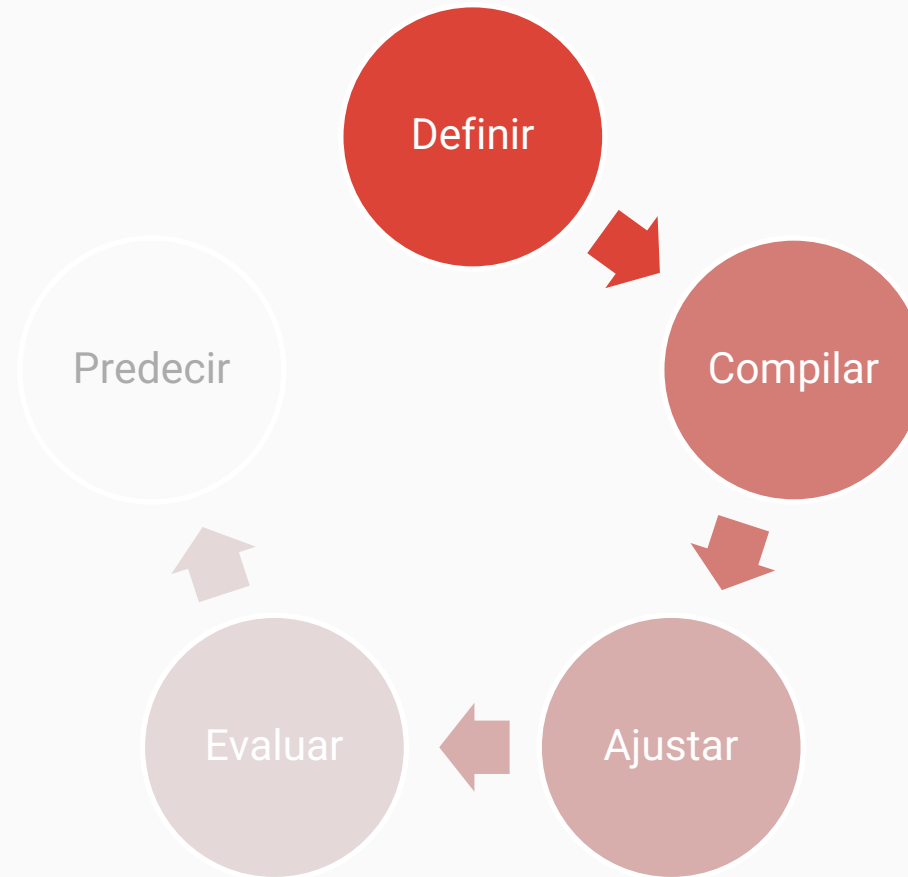
```
import tensorflow as tf
```

```
print(tf.__version__)
```



# INTRODUCCIÓN A TENSORFLOW

Un modelo tiene un ciclo de vida y está dividido en 5 pasos



# INTRODUCCIÓN A TENSORFLOW

## 1. Definición del modelo

Definir el modelo requiere de seleccionar el tipo de modelo que necesitamos y después la arquitectura o la topología de la red neuronal.

Desde esta perspectiva, la API define el número de capas, número de nodos y las funciones de activación.

# INTRODUCCIÓN A TENSORFLOW

Keras proporciona dos formas de definir un modelo:

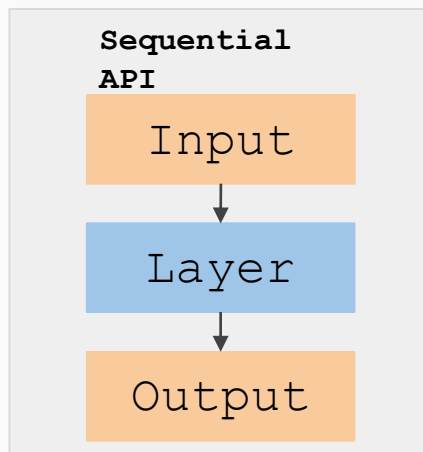
## Sequential API

Estructura secuencial donde cada capa tiene exactamente un tensor de entrada y un tensor de salida.

### Ejemplo:

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(10, input_shape=(8,)))
model.add(Dense(1))
```



## Functional API

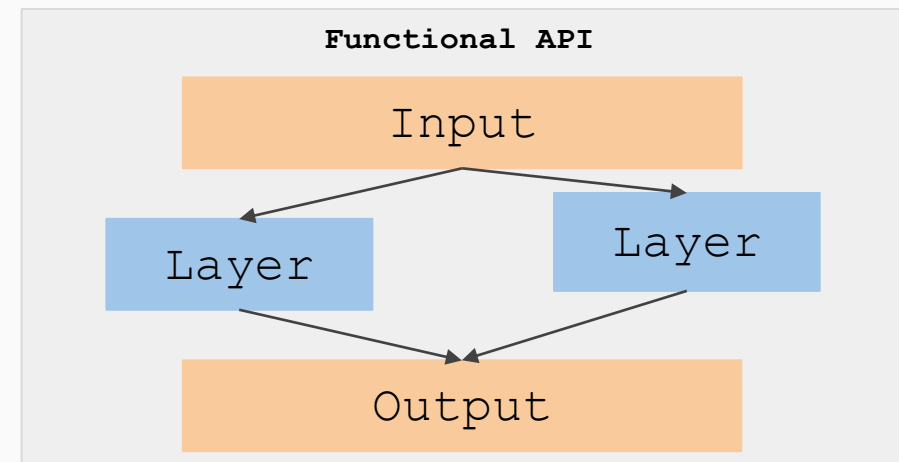
Se utiliza para crear modelos más flexibles que la Sequential API. Puede manejar modelos con topología no lineal, capas compartidas e incluso múltiples entradas o salidas.

### Ejemplo:

```
from tensorflow.keras.layers import Input, Flatten, Dense

input = Input(shape=(28,28))
flatten = Flatten()(input)
hidden1 = Dense(64, activation="relu")(flatten)
output = Dense(10, activation="softmax")(hidden1)

model = keras.Model(inputs = input, outputs = output)
```



## 2. Compilación del modelo

Requiere de:

- Seleccionar una función de costo
- Seleccionar un algoritmo para optimizar
- Agregar una métrica de evaluación para monitorear el modelo

Desde la perspectiva de la API, implica llamar una función para compilar el modelo con la configuración elegida, que preparará las estructuras de datos adecuadas requeridas para el uso eficiente del modelo

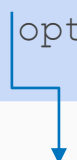
# INTRODUCCIÓN A TENSORFLOW


## 2. Compilación del modelo

- Optimizadores, función de costo y métrica

```
# compilar el modelo
model.compile(optimizer = 'sgd', loss = 'binary_crossentropy' , metrics = ['accuracy'])
```

```
# compilar el modelo
opt = tf.keras.optimizers.SGD(learning_rate = 0.01)
model.compile(optimizer = opt, loss = 'binary_crossentropy' , metrics = ['accuracy'])
```

  
tf.keras.optimizers.RMSprop  
tf.keras.optimizers.Adam

  
'binary\_crossentropy'  
'sparse\_categorical\_crossentropy'  
'mse'

Más optimizadores, funciones de costo y métricas disponibles en:

[https://www.tensorflow.org/api\\_docs/python/tf/keras/optimizers](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers)

[https://www.tensorflow.org/api\\_docs/python/tf/keras/losses](https://www.tensorflow.org/api_docs/python/tf/keras/losses)

[https://www.tensorflow.org/api\\_docs/python/tf/keras/metrics](https://www.tensorflow.org/api_docs/python/tf/keras/metrics)

## 3. Ajuste del modelo

Requiere de configurar:

- Número de épocas
- Tamaño del batch (número de muestras en la época para estimar el error)

Entrenar el modelo implica escoger un algoritmo de optimización para minimizar la función de costo y actualizar el modelo utilizando backpropagation

# INTRODUCCIÓN A TENSORFLOW

## 3. Ajuste del modelo

```
# ajuste del modelo  
model.fit(X, y, epochs=100, verbose=2)
```

Conjunto  
de datos

Etiquetas

`verbose=2` Despliega un  
resumen del progreso de la red  
durante cada época

`verbose=1` Despliega una  
barra de progreso

`verbose=0` No despliega un  
resumen del progreso de la red

```
Epoch 1/1  
 32/60000 [.....] - ETA: 229s - loss: 2.3527 - acc: 0.2188  
 64/60000 [.....] - ETA: 133s - loss: 2.3690 - acc: 0.1562  
 96/60000 [.....] - ETA: 101s - loss: 2.3519 - acc: 0.1562  
128/60000 [.....] - ETA: 85s - loss: 2.3071 - acc: 0.1875  
160/60000 [.....] - ETA: 75s - loss: 2.2376 - acc: 0.2125
```

## 4. Evaluación del modelo

La evaluación del modelo requiere de separar parte del conjunto de datos en un conjunto de datos de prueba reservado únicamente para evaluar el modelo. Estos deben ser datos que no se utilicen en el proceso de entrenamiento para que podamos obtener una estimación no sesgada del rendimiento del modelo al hacer predicciones sobre nuevos datos.

La velocidad de la evaluación del modelo es proporcional a la cantidad de datos que se utilizan para evaluar el modelo, aunque es mucho más rápida que el entrenamiento, ya que el modelo no cambia.

```
# evaluación del modelo  
loss = model.evaluate(X, y, verbose=0)
```



## 5. Realizar una predicción del modelo

Se llama a la función *predict* para hacer una predicción de una etiqueta de una clase, la probabilidad o un valor numérico

```
# realizar una predicción del modelo  
y_pred = model.predict(X)
```

# INTRODUCCIÓN A TENSORFLOW

## ¿Cómo guardar y cargar un modelo entrenado?

El modelo se guarda en un formato H5

Instalación en ambiente virtual 

```
(v_env)$ pip install h5py
```

- Guardar el modelo

```
model.save('model.h5')
```

- Cargar el modelo

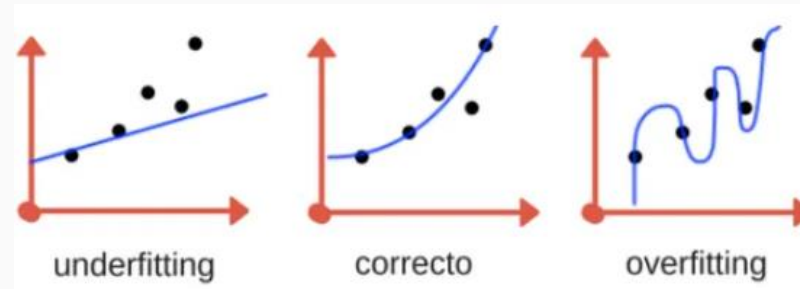
```
from tensorflow.keras.models import load_model
```

```
mlp = load_model('model.h5')  
y_pred = mlp.predict(x)
```

# INTRODUCCIÓN A TENSORFLOW

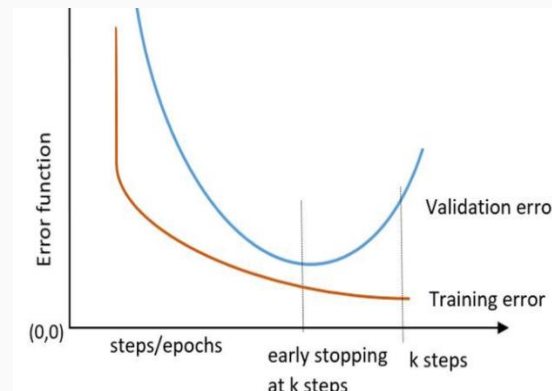
## ¿Cómo detener el entrenamiento en el momento indicado?

- Poco entrenamiento (número de épocas) —————→ *underfitting*
- Demasiado entrenamiento —————→ *overfitting*



## Early stopping

Monitorea la función de costo en el conjunto de datos de entrenamiento y de validación durante el entrenamiento. Tan pronto como el modelo comience a dar signos de overfitting, el proceso de entrenamiento se detiene



# INTRODUCCIÓN A TENSORFLOW

## ¿Cómo detener el entrenamiento en el momento indicado?

### Early stopping

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
# Configurar early stopping  
es = EarlyStopping(monitor='val_loss', patience=5)
```

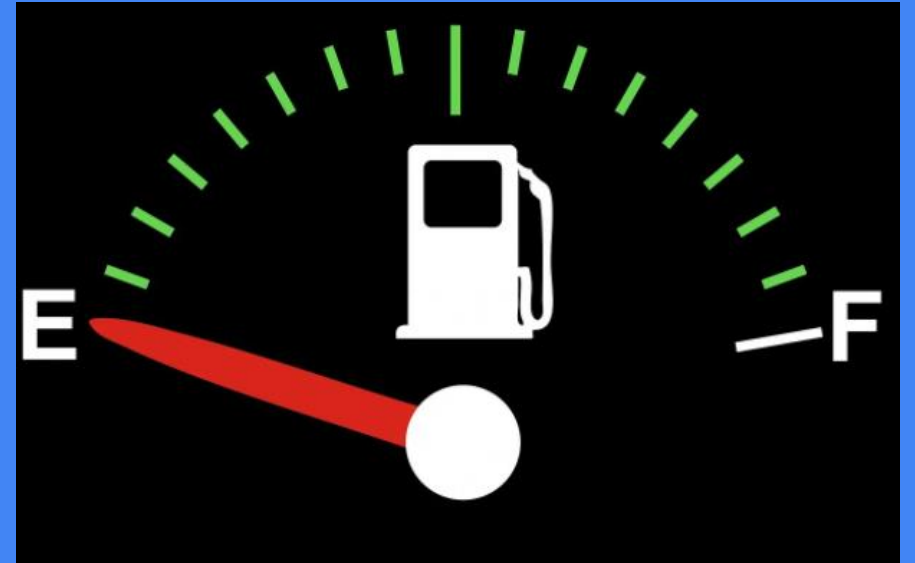
Indica que se va a  
monitorear

Número de épocas  
después de detectar  
overfitting

```
# Ajustar el modelo  
history = model.fit(X,y, epochs=200, callbacks=[es])
```

# Predicción del consumo de gasolina

Práctica 1



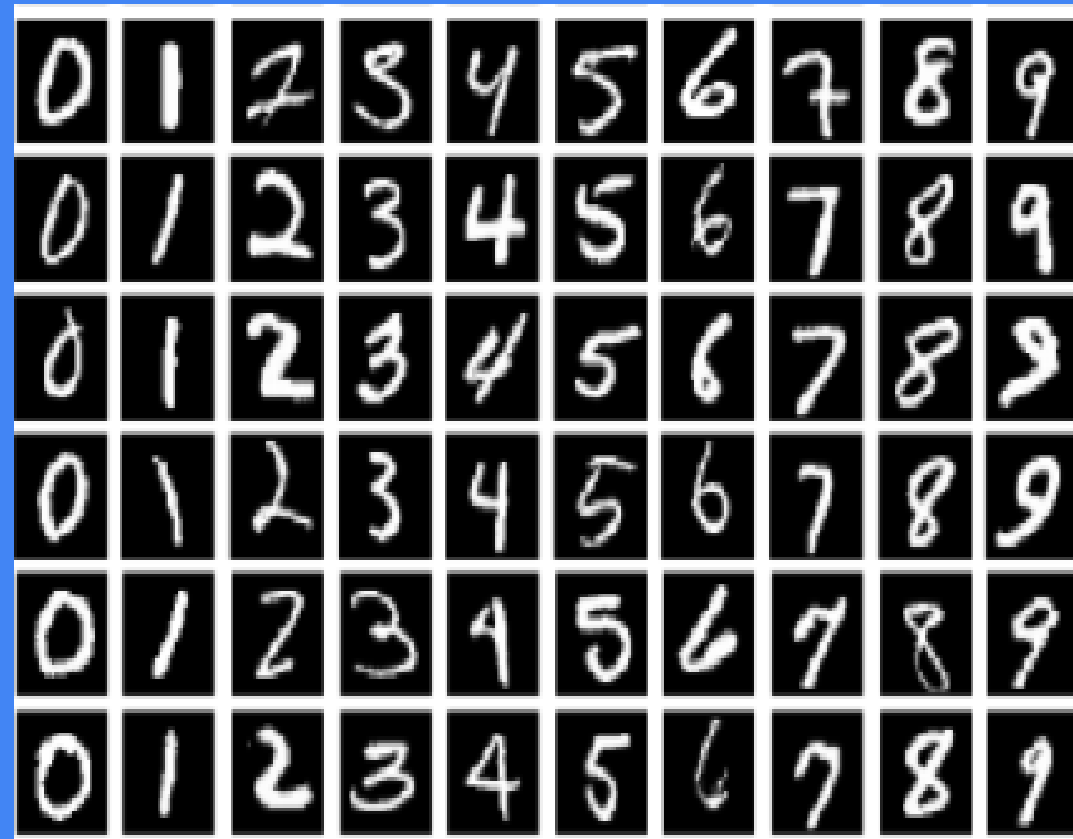
# CONJUNTO DE DATOS MNIST

## Características

- Conjunto de datos: 48 ejemplos
- Etiqueta:
  - Consumo de gasolina
- Características: 4
  - Impuesto a la gasolina
  - El ingreso per cápita
  - El número de millas de carretera pavimentada
  - Proporción de la población con licencia de conducir

# Clasificación de dígitos escritos a mano MNIST

## Práctica 2



# CONJUNTO DE DATOS MNIST

## Características

- Datos de **entrenamiento**: 60,000 imágenes
- Datos de **prueba**: 10,000 imágenes
- Número de clases: 10
- Imágenes de 28x28 píxeles
- Escala de grises
- Imágenes normalizadas y centradas

## Descargar el conjunto de datos

```
from tensorflow import keras

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```





# RED NEURONAL

Entrada



Imágenes

¿Qué características podemos elegir?

**Cada pixel es una característica**

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

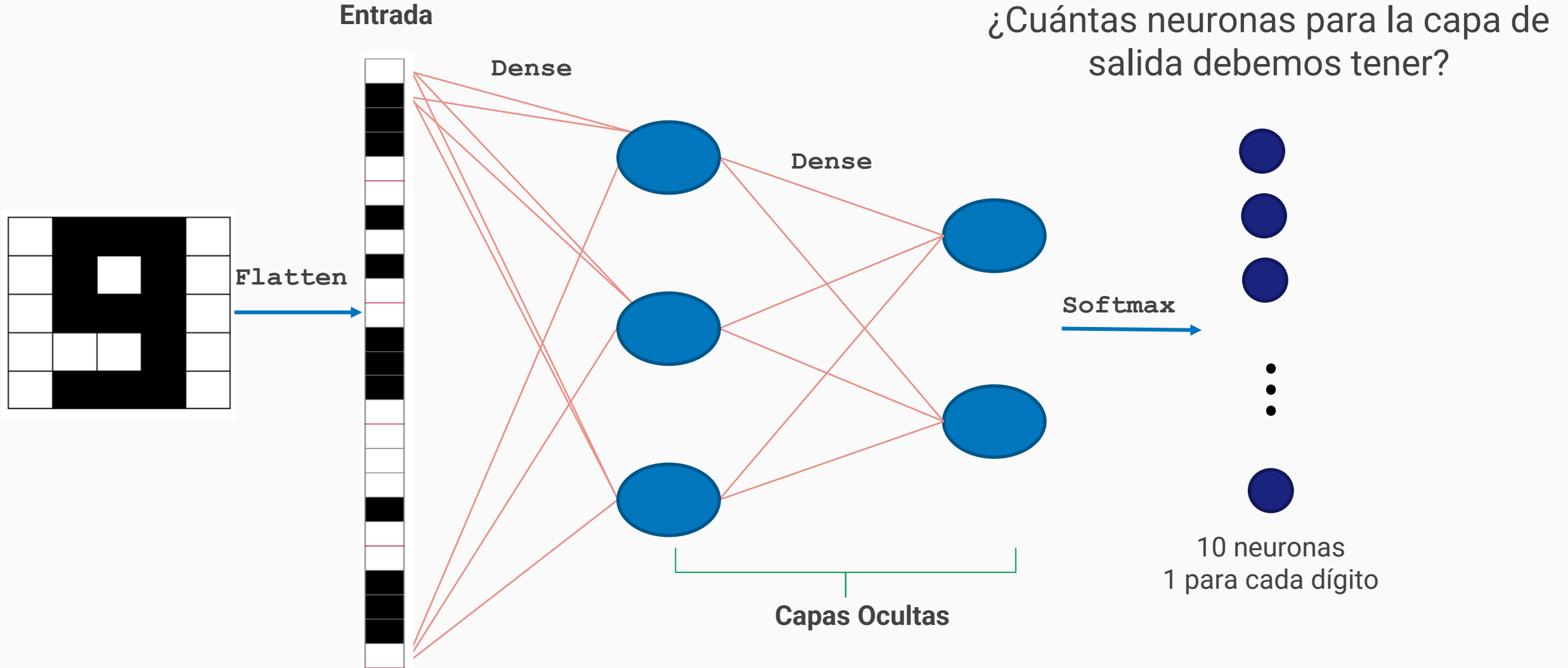
28x28x1

Flatten

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

784

# RED NEURONAL



# Clasificación Fashion MNIST











Ejercicio

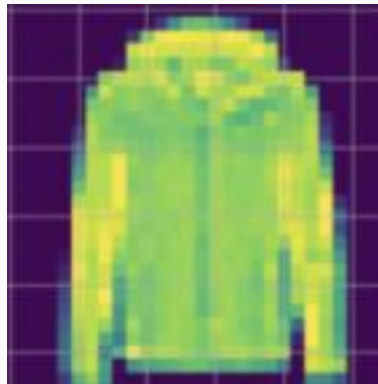


# CONJUNTO DE DATOS FASHION MNIST

## Características

- Datos de **entrenamiento**: 60,000 imágenes
- Datos de **prueba**: 10,000 imágenes
- Número de clases: 10
- Imágenes de 28x28 píxeles
- Escala de grises

0 T-shirt	
1 Trouser	
2 Pullover	
3 Dress	
4 coat	
5 sandal	
6 Shirt	
7 Sneaker	
8 Bag	
9 Ankle boot	



## Descargar el conjunto de datos

```
from tensorflow import keras

fmnist = tf.keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = fmnist.load_data()
```

## Reto

Lograr un error de validación  $\leq 0.3$  (alrededor del 89% de exactitud)