SIAFI

Deep Learning

Introducción a las Redes Neuronales

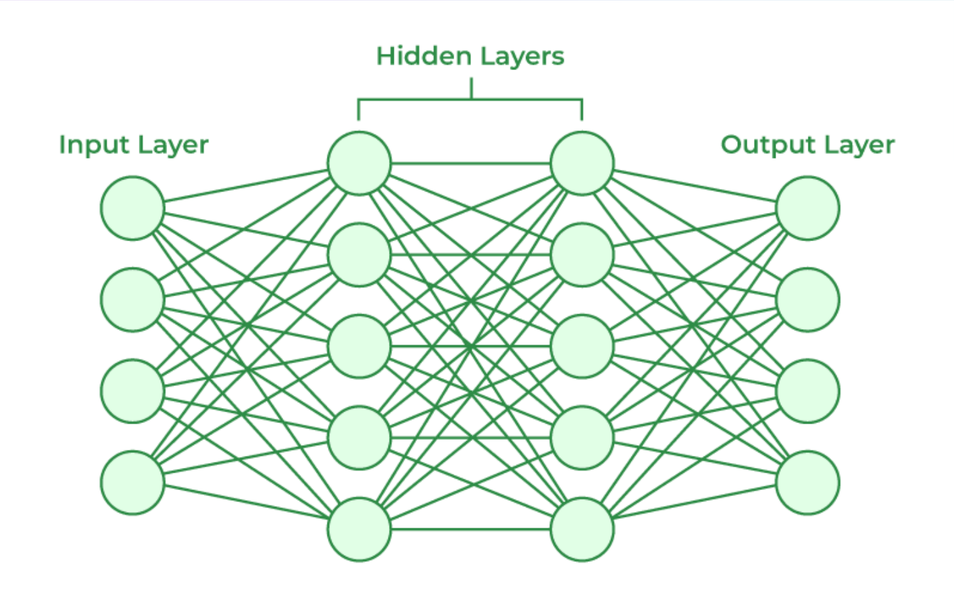
CLASE 02

CAPACITACIÓN

PROPEDÉUTICO 11/NOVIEMBRE/2024

¿Perceptrones multicapa?

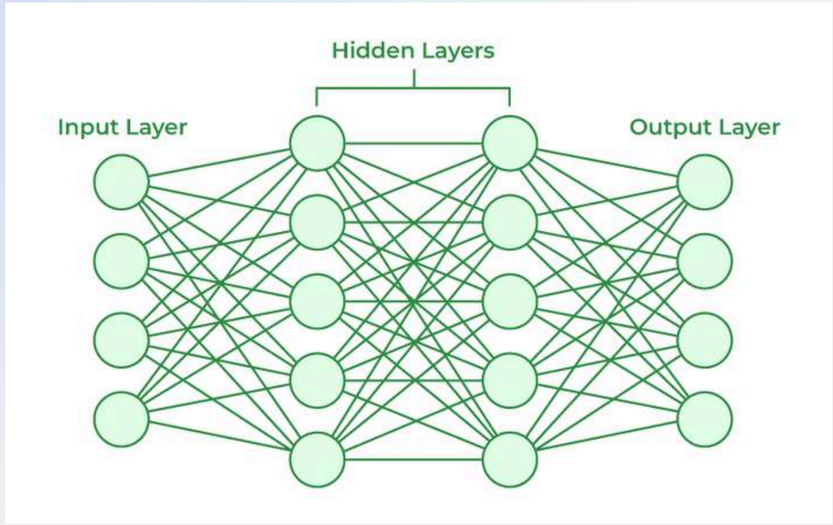
Un perceptrón multicapa (o MLP, por sus siglas en inglés, Multilayer Perceptron) es una red neuronal artificial compuesta por múltiples capas de neuronas que pueden aprender relaciones complejas entre entradas y salidas. Es un tipo de red neuronal profunda (DNN)



Capas

Las capas son un elemento fundamental para estructurar la red y definir su capacidad de aprendizaje, son conjuntos de neuronas organizadas en secuencia, y cada capa está conectada con la siguiente. Un perceptrón multicapa tiene al menos tres tipos de capas:

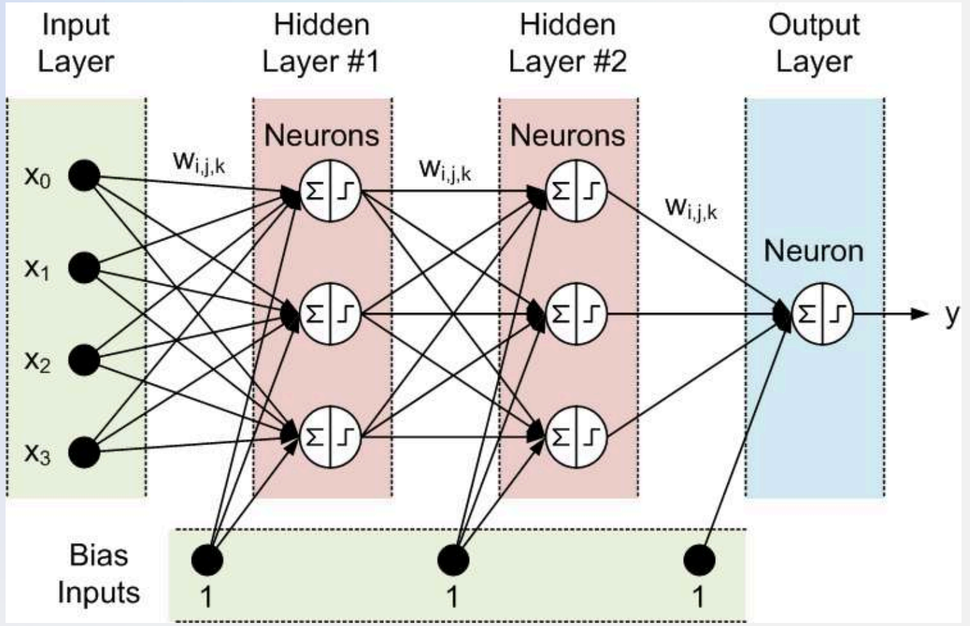
- Capa de entrada
- Capas ocultas
- Capa de salida



Capa de entrada

Recibe los datos de entrada. Cada neurona en esta capa representa una característica o atributo del dato que entra a la red.

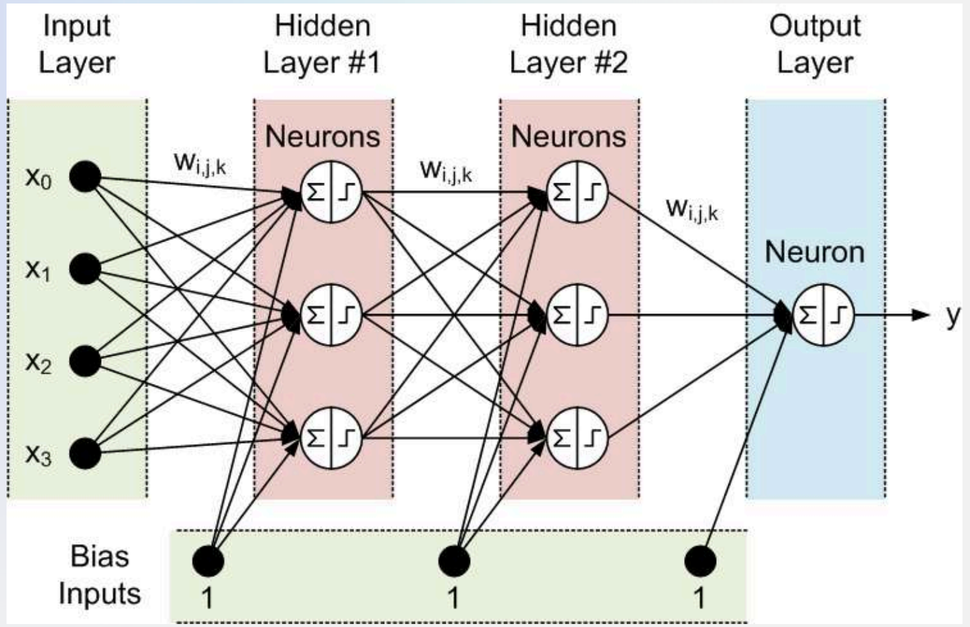
Sirve para recibir los datos que serán procesados. Cada neurona de esta capa representa una entrada (o característica) del problema.



Capas ocultas

Estas capas están entre la capa de entrada y la capa de salida. En un MLP, es posible tener varias capas ocultas (de ahí el término "multicapa"), y cada una está compuesta por múltiples neuronas.

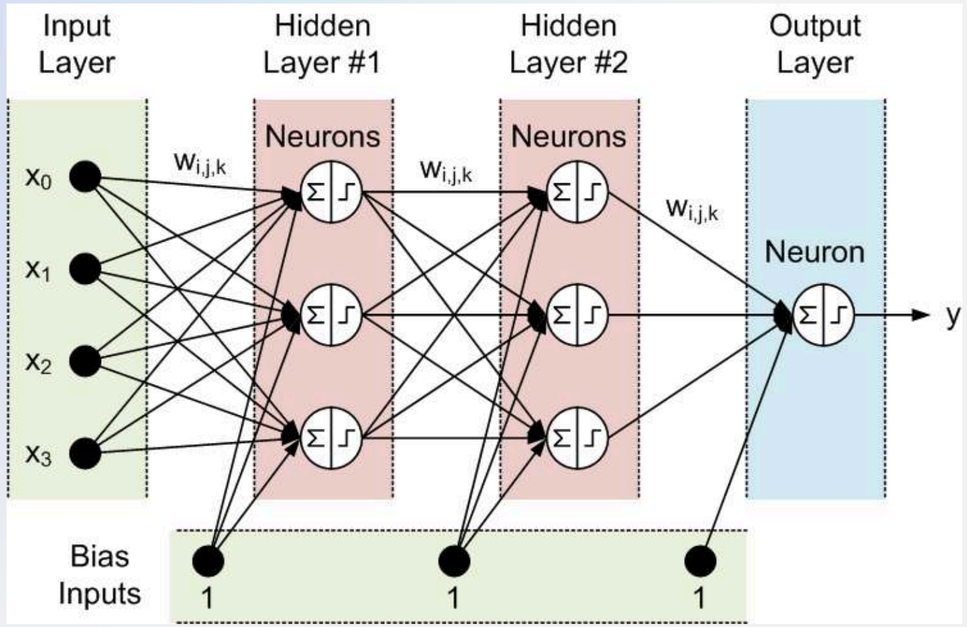
Las capas ocultas son las que realmente hacen el "trabajo" de aprendizaje de patrones en los datos. Al agregar más capas, la red se vuelve más profunda y capaz de aprender patrones complejos, ya que puede transformar y abstraer la información en pasos intermedios.



Capa de salida

Genera el resultado final de la red, que puede ser una predicción o clasificación en función de la tarea.

Produce la salida final, que puede ser una probabilidad (para clasificación), una etiqueta de clase o un valor numérico (para regresión).



¿Cuantas debe haber?

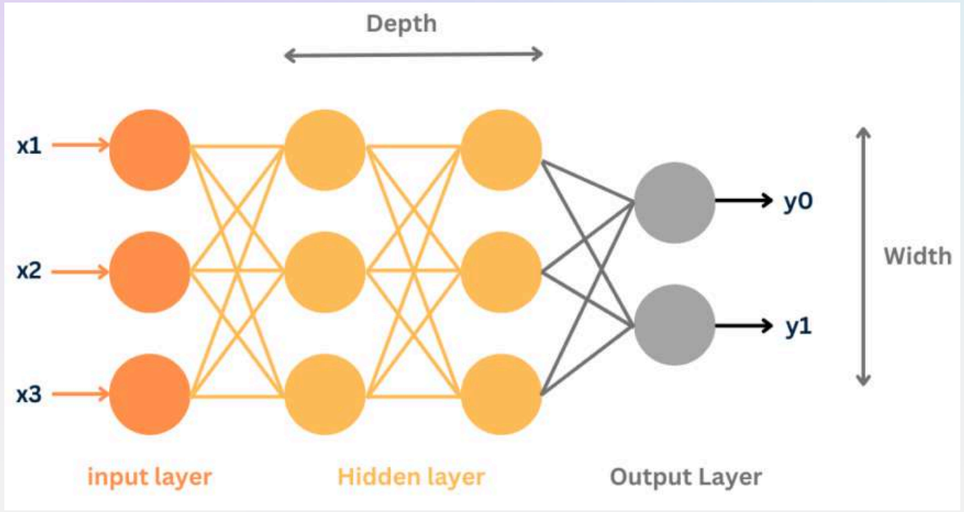
Depende de la complejidad del problema:

- **Red de pocas capas (1-2 capas ocultas)**: Suele funcionar bien para problemas relativamente sencillos donde la relación entre las características y la salida no es muy compleja. Un ejemplo podría ser la predicción de precios de casas con pocas variables.
- **Red profunda (más de 2 capas ocultas)**: Utilizadas para problemas más complejos, como reconocimiento de imágenes, procesamiento de lenguaje natural, y otras tareas donde los patrones en los datos son intrincados. Redes con muchas capas ocultas pueden aprender patrones de alta abstracción, aunque requieren más datos y potencia de cómputo para entrenarse.

Cositas sobre las capas

Consideraciones:

Profundidad vs. Ancho: Aumentar la profundidad (número de capas) permite captar patrones más complejos, mientras que aumentar el ancho (número de neuronas en cada capa) puede ayudar a la red a captar más detalles en los datos.



Tensorflow

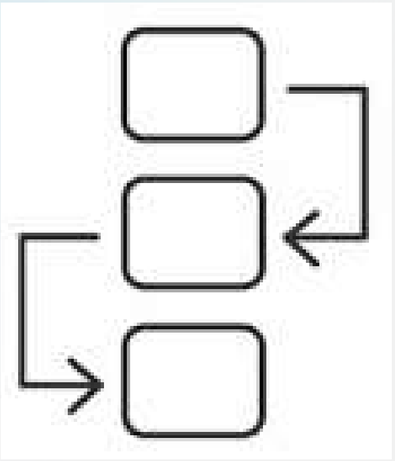
Es una biblioteca de código abierto desarrollada por **Google** para construir y entrenar modelos de inteligencia artificial y aprendizaje automático, especialmente redes neuronales profundas.
Es una herramienta popular en el campo del **deep learning** debido a su capacidad para manejar grandes cantidades de datos y realizar cálculos complejos de manera eficiente.

pip install tensorflow

tf.keras.Sequential

Clase en TensorFlow que permite **crear modelos** de aprendizaje profundo de forma rápida y sencilla mediante la pila Keras de TensorFlow. Se usa para construir **modelos secuenciales**, es decir, modelos en los que las capas se apilan una tras otra en un orden específico, de modo que la salida de cada capa se convierte en la entrada de la siguiente

Toma una lista de capas de una red neuronal y las ordena secuencialmente

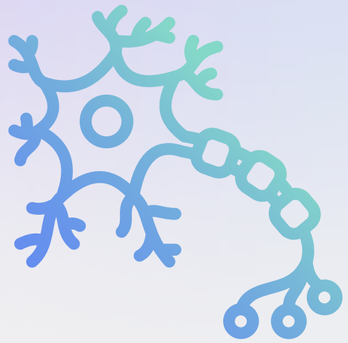


tensorflow.keras.layers

Dense

Dense es una capa completamente conectada o "fully connected layer":

Cada neurona de la capa actual se conecta con cada neurona d ela capa siguiente.



tensorflow.keras.layers Dense

Como parámetros importantes serían:

- **units**: Que son el número de neuronas de la capa
- **activation**: La función de activación que transforma la salida de la neurona
- **input_shape**: Define la forma de los datos que serán pasados a la red en la primera capa.

model.compile()

Se utiliza para **configurar el modelo** antes de entrenarlo. Es donde se definen el optimizador, la función de pérdida y las métricas que se utilizarán durante el entrenamiento y la evaluación del modelo.

Sus parámetros importantes son:

- **optimizer**: Define el algoritmo que se utilizará para minimizar la función de pérdida durante el entrenamiento. Puede ser una cadena (como 'adam', 'sgd', etc.) o una instancia de un optimizador como tf.keras.optimizers.Adam().

model.compile()

Otros de sus parámetros importantes son:

- **loss**: Especifica la función de pérdida que el modelo debe minimizar. Dependiendo del tipo de problema, se puede usar funciones como *categorical_crossentropy*, *binary_crossentropy*, *mean_squared_error*, etc.
- **metrics**: Lista de métricas para monitorear el desempeño del modelo. Comúnmente se usa **accuracy** para clasificación, **mae** para regresión, entre otros.

model.fit()

Se utiliza para entrenar el modelo. Durante el entrenamiento, el modelo ajusta los pesos de las capas para minimizar la función de pérdida utilizando los datos de entrada y las etiquetas de salida.

Sus parámetros importantes son:

- **x**: Los datos de entrada (pueden ser un array NumPy, un generador, un DataFrame de Pandas, etc.).
- **y**: Las etiquetas de salida correspondientes a las entradas x.

model.fit()

Otros de sus parámetros importantes son:

- **epochs**: Número de veces que el modelo pasará por todo el conjunto de datos durante el entrenamiento.
- **batch_size**: El número de ejemplos procesados en cada actualización de los pesos.
- **validation_data**: Un conjunto de datos para evaluar el modelo al final de cada época (puede ser una tupla (x_val, y_val)).
- **verbose**: Controla la cantidad de información mostrada durante el entrenamiento.

epochs

Un **epoch** es una iteración completa sobre todo el conjunto de datos durante el entrenamiento. Si tenemos un conjunto de datos grande, más epochs pueden ser necesarios para que el modelo aprenda correctamente.

Ojito ahí. Si usamos muchos epochs el modelo puede sobreajustarse, y si usamos pocos, el modelo puede no haber tenido suficiente tiempo para aprender patrones importantes por lo que su rendimiento sería pobre.

Para muchos problemas, su valor general va entre 10 y 50. Aunque lo mejor sería ver su curva de aprendizaje y ver cómo cambia la pérdida y presición.

batch_size

Número de ejemplos que el modelo procesa antes de actualizar sus pesos. En lugar de procesar todo el conjunto de datos a la vez, se procesan pequeños batches de datos.

Valores del batch: Pequeños estan entre 16 y 32, grandes entre 128 y 256.

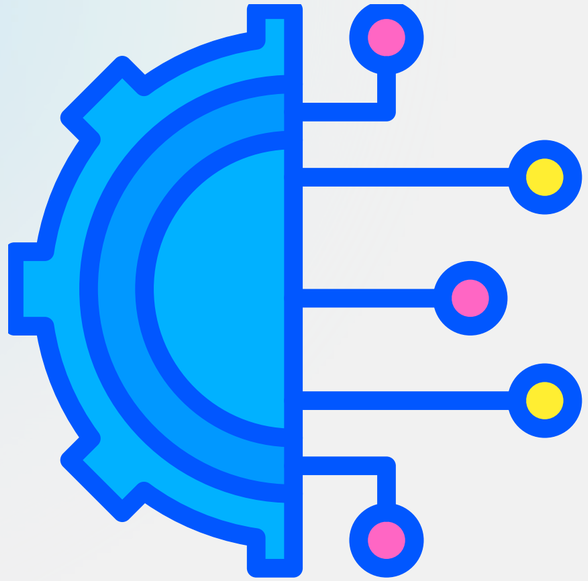
Como pauta general, 32 es un valor comúnmente utilizado en muchotes modelos y es un buen punto de partida. Si se tienen muchos datos usamos más batches.

Nota: El tamaño de batch también influye en la cantidad de memoria utilizada por la GPU

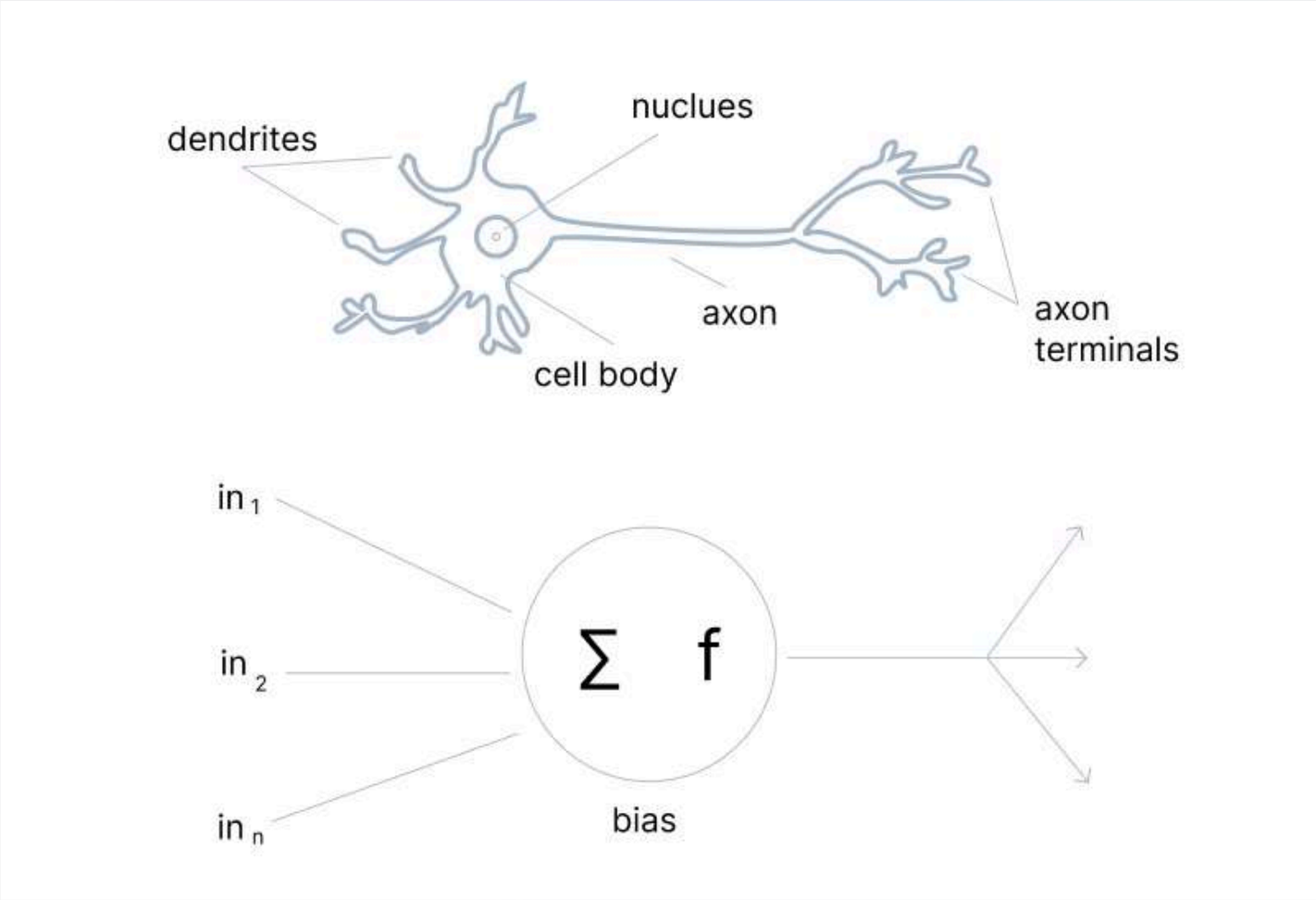
¿Funciones de activacion?

Una función de activación decide si una neurona debería ser activada o no. Esto significa si decidirá si la entrada de la neurona es importante o no en el proceso de predicción.

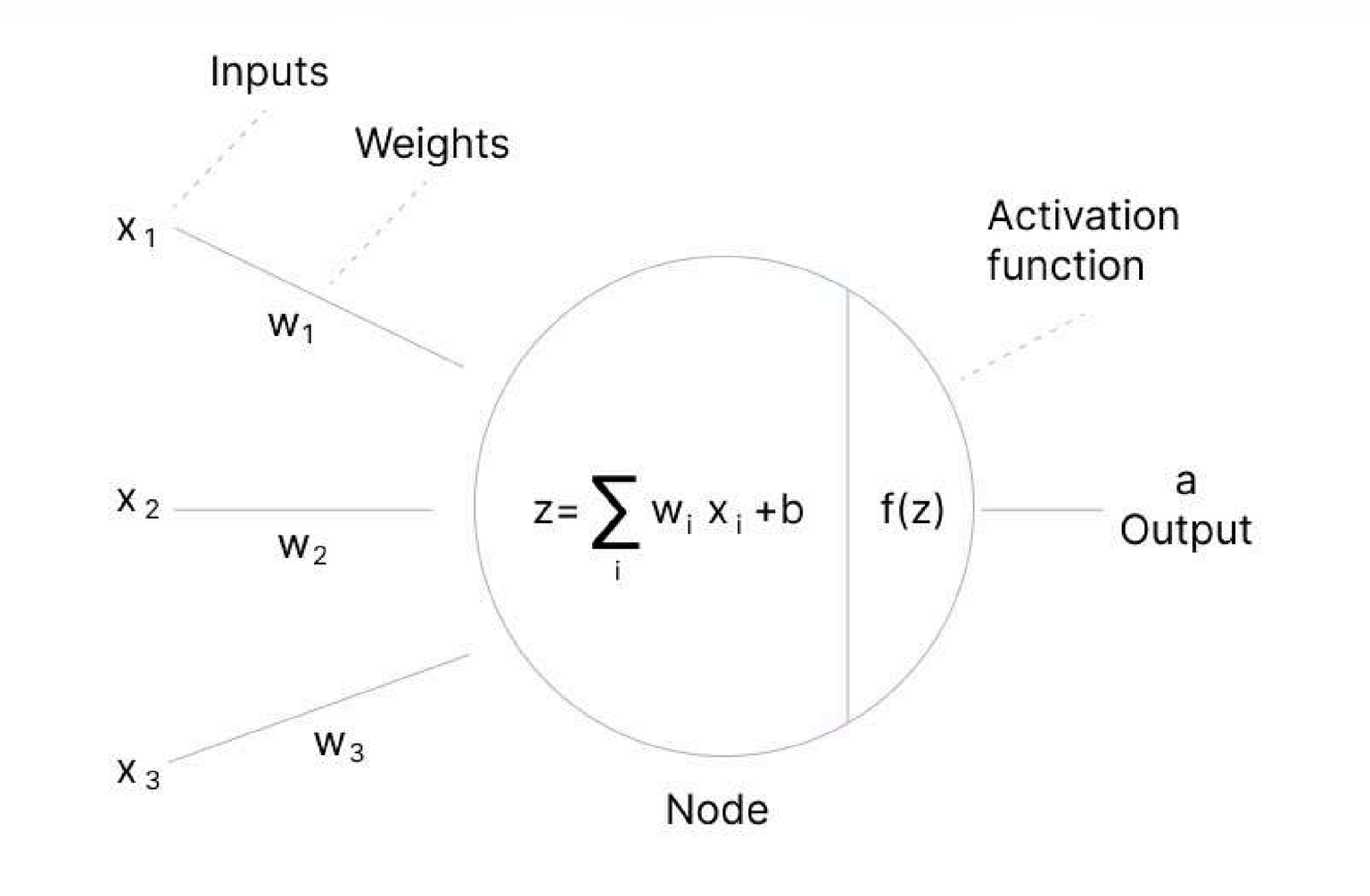
Su rol como función de Activación es derivar la salida de un conjunto de entradas que alimentan al nodo (O capa).



¿Funciones de activacion?



¿Funciones de activacion?



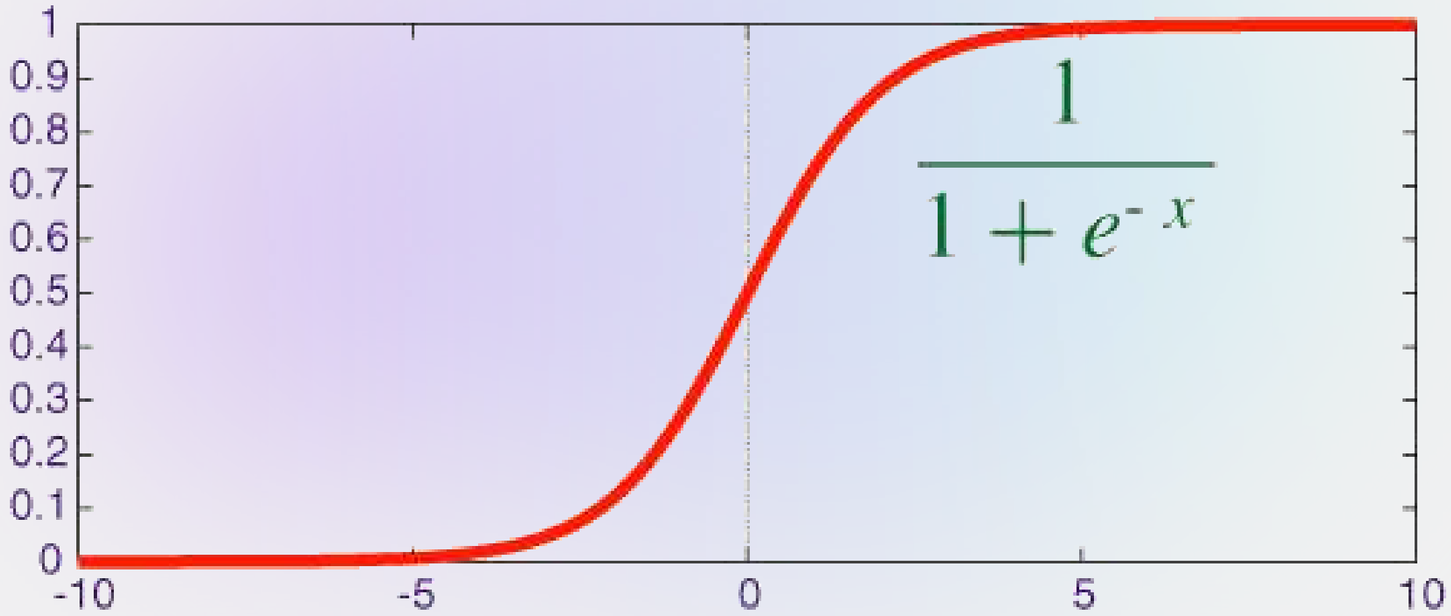
¿Funciones de activacion?

El propósito de una función de activación es permitir a la red aprender y representar patrones complejos en los datos.

Recuerden, intentar hacer el resultado algo no-lineal.

Funcion Sigmoid

Función que toma cualquier valor de entrada y lo convierte a un valor entre 0 y 1.



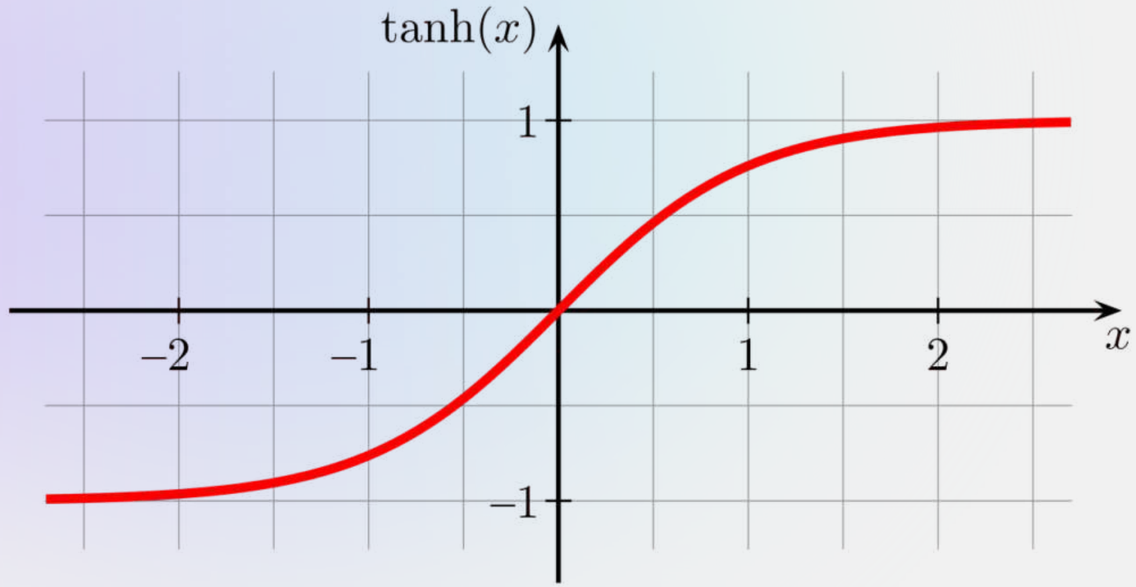
Se usa comúnmente en las capas de salida para problemas de clasificación binaria.

Función Tanh

Función que transforma un valor de entrada en un rango entre -1 y 1

$$Tanh$$

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

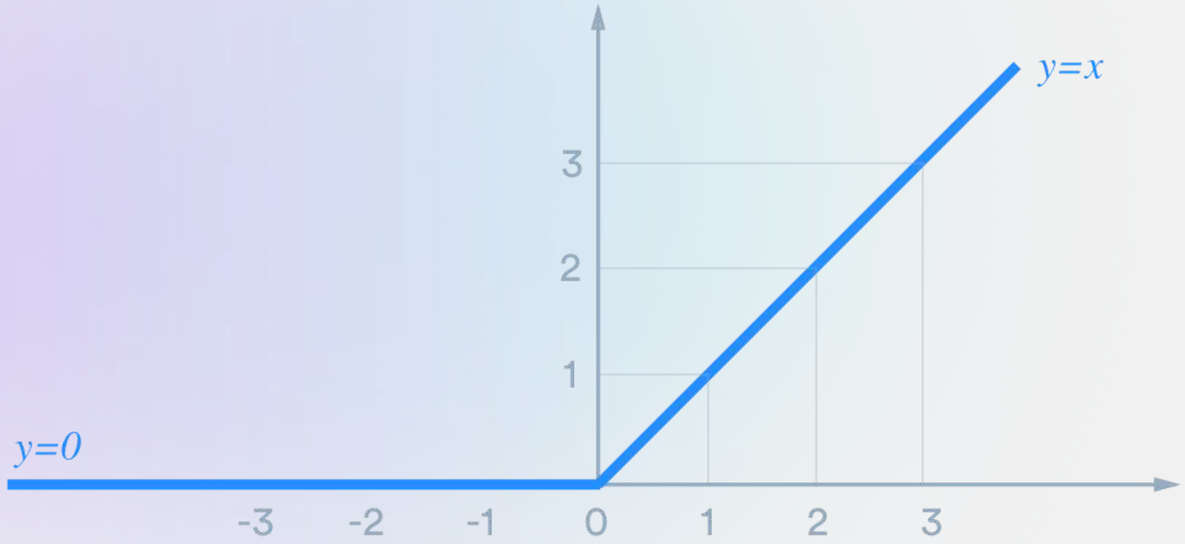


Se usa especialmente en las capas ocultas.

Función ReLU

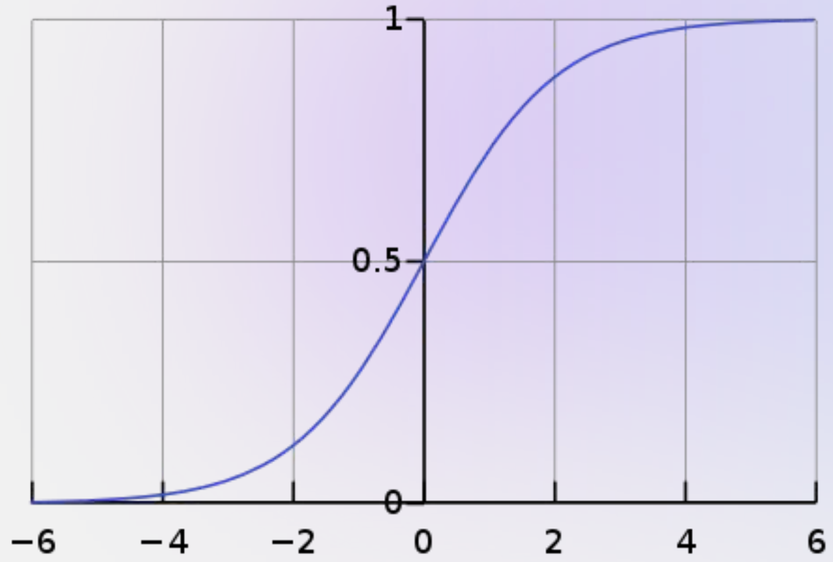
Cualquier valor de entrada *x* negativo se convierte en cero, mientras que los valores positivos permanecen sin cambios.

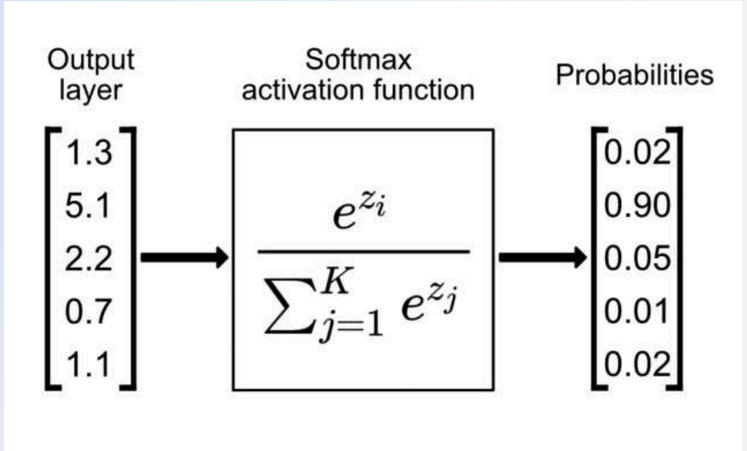
$$f'(x) = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$$



Función Softmax

Muy usada en clasificación multiclase. Esta función no está pensada para introducir la No-Linealidad (por ser aplicada en la última capa).





Se usa especialmente en la última capa.

Ejercicio