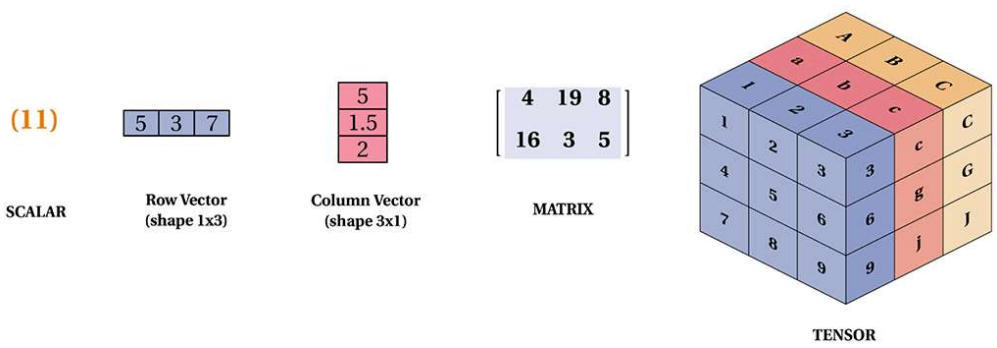
Redes Neuronales

Contents

Tensores

Los tensores son simplemente **contenedores de números**. Puedes pensar en ellos como otra forma de llamar a las **matrices multidimensionales**. El concepto de tensor permite extender el uso de matrices a un número arbitrario de dimensiones. []

Fig.1.
Tensores



Nota. Tomada de Tensorflownet, s. f., https://tensorflownet.readthedocs.io/en/latest/Tensor.html.

Concepto	Descripción	Ejemplo
Escalar (0D)	Un solo número.	5 (un valor que podría representar una intensidad de luz)
Vector (1D)	Una lista de números.	[255, 128, 0, 128, 64, 192, 0, 255, 128] (valores aplanados de una imagen en escala de grises)

Matriz (2D)	Una tabla de números.	Representación 2D de una imagen en escala de grises con 2 píxeles de altura y 2 píxeles de ancho:
		`[[255, 128],`
		` [128, 64]]`

Tensor (3D)	Un conjunto de matrices, donde cada matriz representa un canal de color.	Representa una imagen RGB de 2 píxeles de altura y 2 píxeles de ancho, donde cada conjunto de 3 números representa los valores RGB para cada píxel:
		`[`
		`  [[255, 0, 0], [0, 255, 0]],`
		`  [[0, 0, 255], [255, 255, 0]]`
		`]`

Tensor (4D)	Un conjunto de tensores 3D, representando un lote de datos, como un conjunto de imágenes.	Un lote de imágenes 20 imagenes de 2x2 RGB

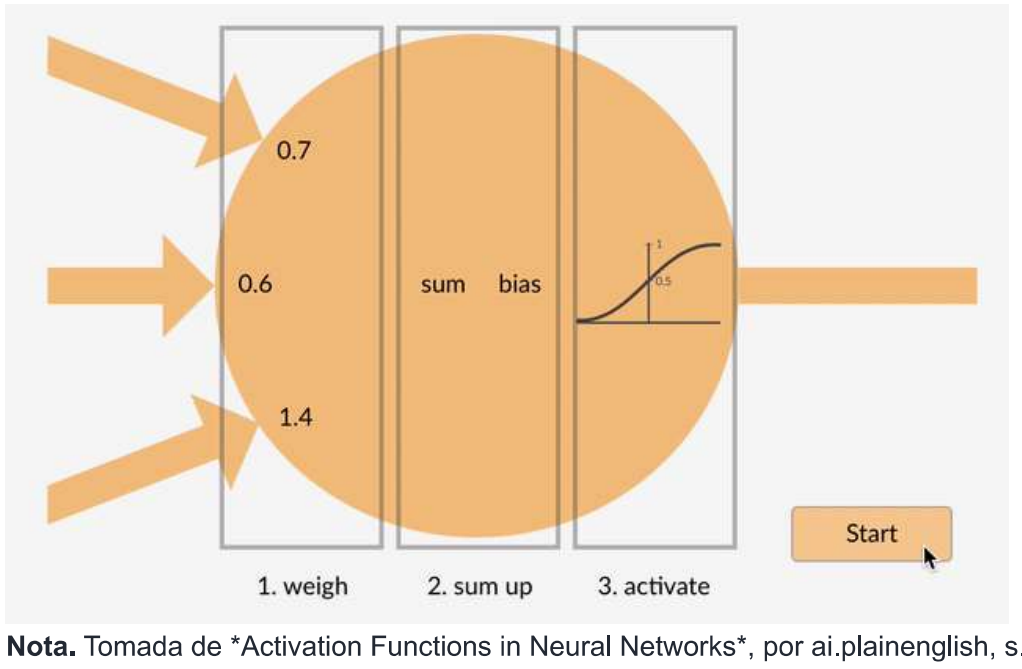
Importancia de los Tensores en Machine Learning

- *Representación de Datos:* Los tensores permiten manejar datos en múltiples dimensiones, como imágenes (tensor 3D) y conjuntos de datos (tensor 4D).
- *Operaciones Matemáticas:* Facilitan operaciones matemáticas eficientes, esenciales para el entrenamiento de modelos.
- *Generalización:* Permiten aplicar los mismos principios a diferentes tipos de datos, simplificando la construcción de modelos.
- *Frameworks de Deep Learning:* Son la base de bibliotecas como TensorFlow y PyTorch, optimizando la manipulación de datos.
- *Escalabilidad:* Soportan grandes volúmenes de datos, mejorando la velocidad y eficiencia del entrenamiento.
- *Facilitación del Aprendizaje Profundo:* Son cruciales para las transformaciones en redes neuronales, permitiendo el aprendizaje de características complejas.

Perceptrón y Función de Activación

El perceptrón es una unidad básica de procesamiento en redes neuronales que toma múltiples entradas, aplica pesos y bias, y utiliza una función de activación para producir una salida.

Fig.2.
Representación de un perceptrón



Nota. Tomada de *Activation Functions in Neural Networks*, por ai.plainenglish, s. f., https://ai.plainenglish.io/activation-functions-in-neural-networks-3d8211678fb2.

Estructura del Perceptrón

- *Entradas*: El perceptrón recibe varias entradas (características), representadas por un vector. Cada entrada puede ser un número que representa algún atributo del dato que se está analizando (por ejemplo, el color de un pixel en una imagen).
- *Pesos*: Cada entrada tiene un peso asociado que indica su importancia en la decisión que el perceptrón tomará. Estos pesos se inicializan aleatoriamente y se ajustan durante el proceso de entrenamiento.
- *Bias*: Se añade un valor llamado **bias** (sesgo) a la suma ponderada de las entradas. El bias permite al perceptrón ajustar su salida de manera que no dependa únicamente de las entradas.
- *Suma Ponderada*: El perceptrón calcula una **suma ponderada** de las entradas, que se expresa matemáticamente como

$$z = \sum (w_i \cdot x_i) + b$$

donde:

- w_i son los **pesos** asociados a cada entrada,
- x_i son las **entradas** (características) del dato,
- b es el **bias** (sesgo) que permite ajustar la salida del modelo.

Función de Activación

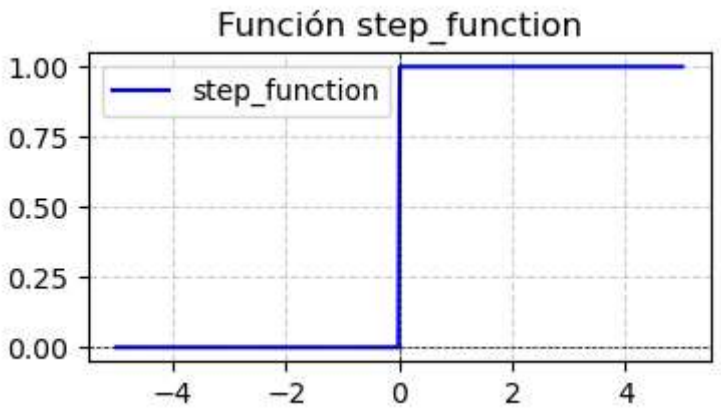
- La suma ponderada z se pasa a través de una **función de activación** con el objetivo de introducir no linealidad al modelo matemático dado por la red (notese que la composición de sumas ponderadas es a su vez una suma ponderada).

```
import numpy as np
import matplotlib.pyplot as plt

def plot_act_fun(f, x_range=(-5, 5)):
    title=f"Función {f.__name__}"
    x = np.linspace(*x_range, 400)
    y = f(x)

    plt.figure(figsize=(4, 2))
    plt.plot(x, y, label=f.__name__, color='b')
    plt.axhline(0, color='black', linewidth=0.5, linestyle="--")
    plt.axvline(0, color='black', linewidth=0.5, linestyle="--")
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.title(title)
    plt.legend()
    plt.show()
```

Función Escalón

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

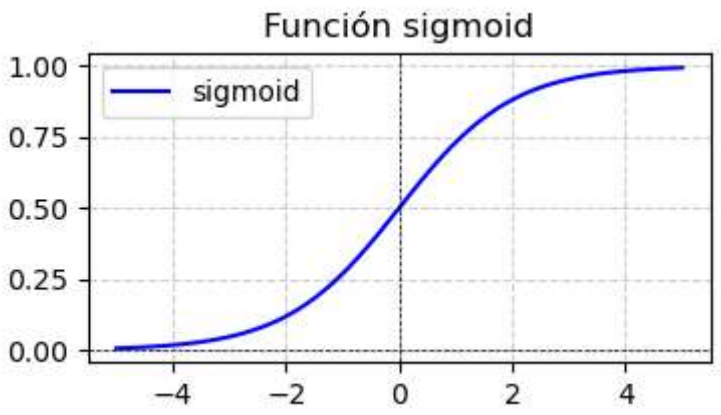
```python
def step_function(x):
    return np.where(x >= 0, 1, 0)
plot_act_fun(step_function)
```



Función Sigmoide

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

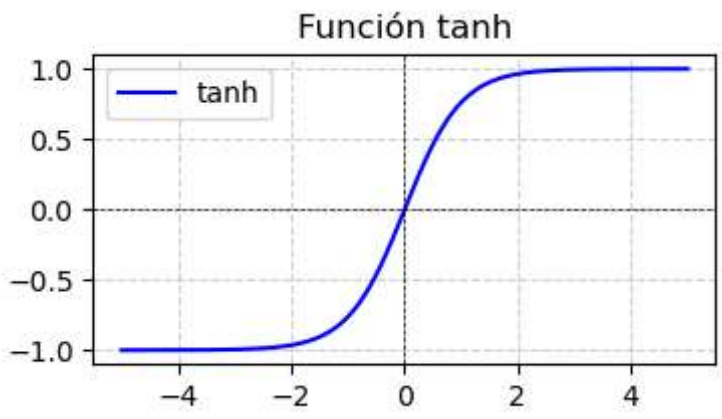
```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

plot_act_fun(sigmoid)
```



Función Tangente Hiperbólica (Tanh)

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

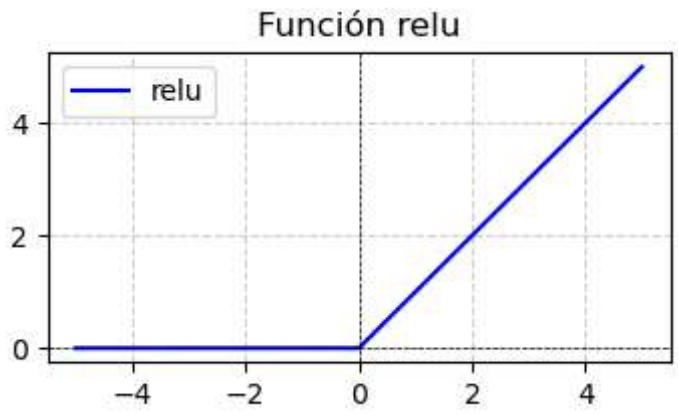
```python
def tanh(x):
    return np.tanh(x)
plot_act_fun(tanh)
```



Función ReLU (Rectified Linear Unit)

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

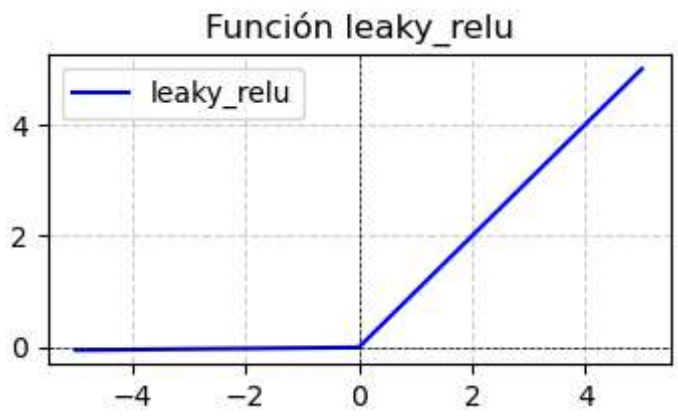
```python
def relu(x):
    return np.maximum(0, x)
plot_act_fun(relu)
```



Función Leaky ReLU

$$LeakyReLU(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

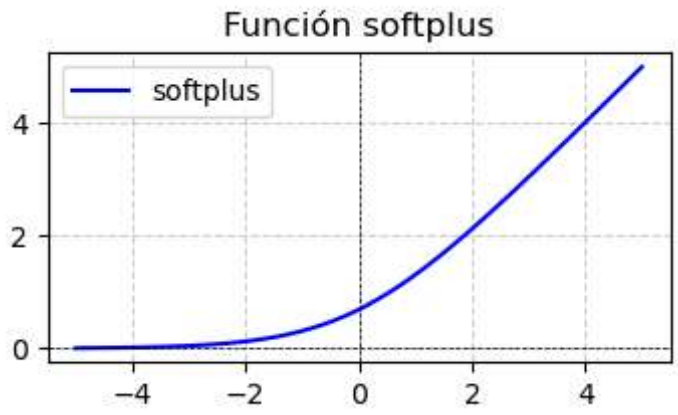
```
def leaky_relu(x, alpha=0.01):
    return np.where(x > 0, x, alpha * x)
plot_act_fun(leaky_relu)
```



Función Softplus

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

```
def softplus(x):
    return np.log(1 + np.exp(x))
plot_act_fun(softplus)
```



Redes Neuronales

Las redes neuronales son una técnica avanzada de aprendizaje automático que emula el funcionamiento del cerebro humano para aprender patrones y características directamente de los datos. Esta técnica utiliza una arquitectura compuesta por múltiples capas de neuronas o perceptrones, que incluyen una capa de entrada, varias capas ocultas y una capa de salida.

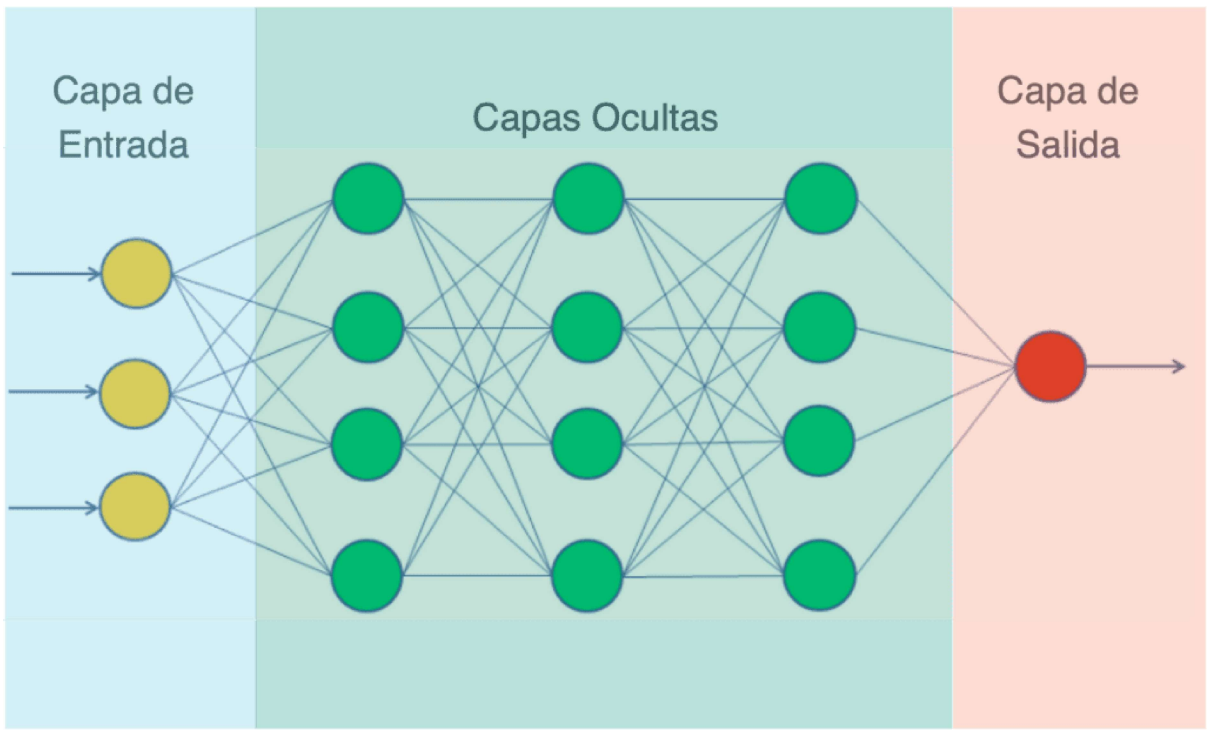
En una red neuronal, una capa es una colección de neuronas que trabajan juntas para procesar información.

Estructura en Capas:

- *Capa de Entrada:* Aquí es donde se introducen los datos. Cada neurona en esta capa representa una característica o atributo del conjunto de datos.
- *Capas Ocultas:* Estas son las capas intermedias que realizan cálculos complejos. Cada neurona en estas capas toma entradas, las multiplica por pesos (que se ajustan durante el proceso de entrenamiento) y aplica una función de activación para determinar su salida.

- *Capa de Salida:* Esta es la capa final, donde se generan las predicciones o clasificaciones basadas en las transformaciones realizadas por las capas anteriores.

Fig.3.
Esquema de red neuronal artificial



Nota. Tomada de Aprende IA, s. f., https://aprendeia.com/que-son-las-redes-neuronales-artificiales/.

Capa densa (capa totalmente conectada o 'fully connected layer')

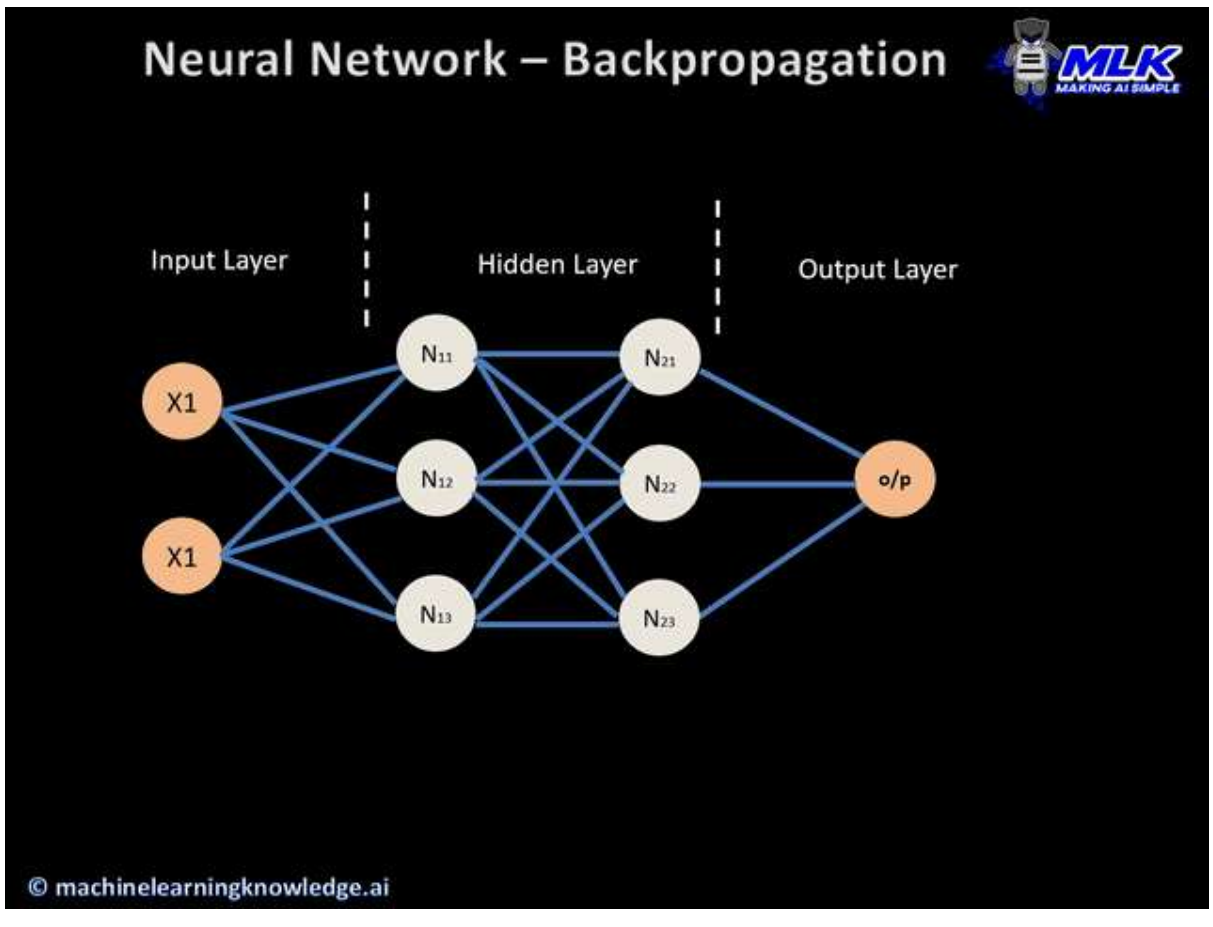
Es un tipo de capa en una red neuronal en la que cada neurona está conectada a todas las neuronas de la capa anterior.

Uso en Redes Neuronales: Las capas densas se utilizan comúnmente en la parte final de la arquitectura de redes neuronales, donde es necesario tomar decisiones basadas en las características extraídas por capas anteriores. Normalmente asociado con el concepto de función de activación (más adelante se vera en más detalle).

Backpropagation

Es un algoritmo de aprendizaje en redes neuronales que ajusta los pesos minimizando el error. Funciona en dos fases: **propagación hacia adelante**, donde la información fluye hasta la salida, y **propagación hacia atrás**, donde el error se distribuye y los pesos se actualizan mediante gradiente descendente. La **tasa de aprendizaje** controla cuánto cambian los pesos en cada actualización: valores altos aceleran el aprendizaje pero pueden ser inestables, mientras que valores bajos lo hacen más lento pero preciso.

Fig.4.
Ilustración del algoritmo de retropropagación (Backpropagation)



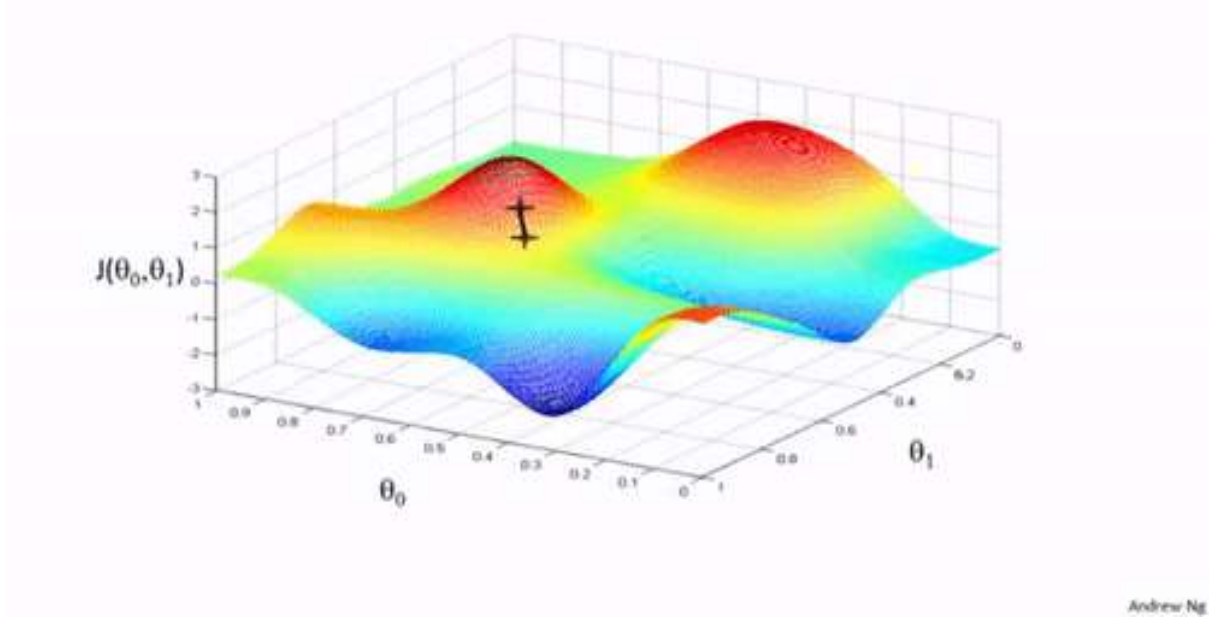
Nota. Tomada de *Backpropagation for Dummies*, por Analytics Vidhya, s. f., https://medium.com/analytics-vidhya/backpropagation-for-dummies-e069410fa585.

> **Función de pérdida o función de coste**
>
> Es una función matemática que mide qué tan bien o mal está funcionando un modelo de aprendizaje automático. Su objetivo es cuantificar la diferencia entre las predicciones del modelo y los valores reales. Para regresión, se usan el *Error Cuadrático Medio (MSE)* y el *Error Absoluto Medio (MAE)*.Para clasificación, se usa la Entropía Cruzada.

Descenso del Gradiente en Redes Neuronales

En el contexto de las redes neuronales, el descenso del gradiente es un proceso iterativo que ajusta los parámetros del modelo, los *pesos (w)* y los *sesgos (b)* para minimizar la función de pérdida *(L)*, que mide la diferencia entre las predicciones del modelo y los valores reales.

Fig.5.
Representación del descenso del gradiente



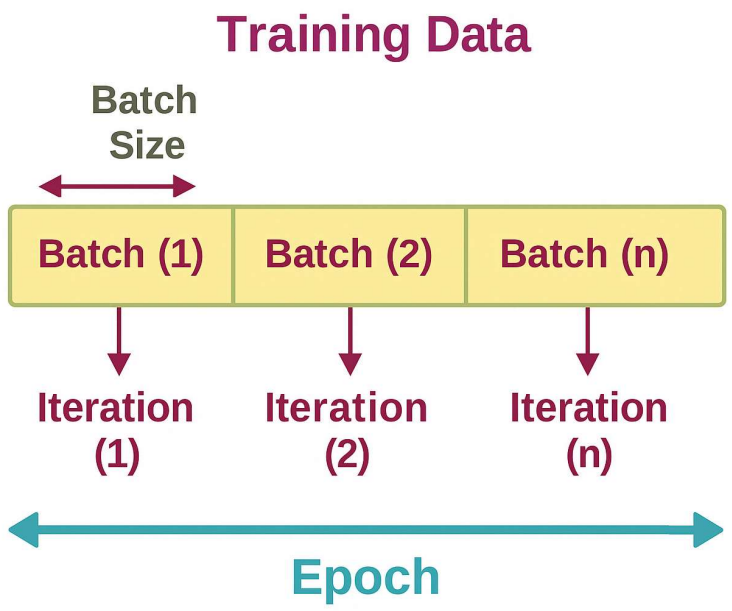
Nota. Tomada de *Backpropagation for Dummies*, por Analytics Vidhya, s. f., https://medium.com/analytics-vidhya/backpropagation-for-dummies-e069410fa585.

Hiperparámetros

Los hiperparámetros son valores del modelo que se deben configurar antes del inicio del proceso de entrenamiento y que no son modificados por el modelo. Estos incluyen:

- **Tasa de aprendizaje (learning rate):** Determina el tamaño de los pasos que se dan durante la actualización de los pesos. Una tasa de aprendizaje demasiado alta puede provocar que el modelo no converja, mientras que una tasa demasiado baja puede hacer que el entrenamiento sea muy lento.
- **Número de épocas (epochs):** Es la cantidad de veces que el algoritmo de entrenamiento recorre todo el conjunto de entrenamiento.
- **Tamaño del lote (batch size):** Define cuántas muestras se utilizan para calcular el error y actualizar los pesos en cada iteración. Puede afectar la estabilidad y la velocidad del entrenamiento.

Fig.6.
Epochs y batch size



Nota. Imagen generada con inteligencia artificial por ChatGPT (OpenAI), 2025.

```

- **Arquitectura de la red:** Esto incluye el número de capas, el número de neuronas o kernel en cada capa y el tipo de funciones de activación utilizadas.

# Caso de estudio: Entrena un perceptrón para simular la compuerta lógica AND

Definimos los datos de entrada $x$ y la salida deseada $y$

Los datos de entrada se definen en una matriz $x$, donde cada fila representa un conjunto de entradas para la función lógica AND. La salida deseada se define en el vector $y$.

| $x\_1$ | **x_2** | **y** |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

1. Definir la función de activación (escalón), esta funciom decide si una neurona “se activa” (`1`) o no (`0`), dependiendo de si el valor de entrada es mayor o menor que cero. Matemáticamente, se puede expresar como:

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

```python
def step_function(x):
    return 1 if x >= 0 else 0

salida=step_function(0)
salida
```

```
1
```

2. Definir los **datos de entrada** `X` que alimentarán la neurona, así como la **salida esperada** `y` que se desea obtener.

```python
import numpy as np

X = np.array([[0, 0],  # and(0, 0) = 0
              [0, 1],  # and(0, 1) = 0
              [1, 0],  # and(1, 0) = 0
              [1, 1]]) # and(1, 1) = 1

y = np.array([0, 0, 0, 1])  # salidas deseadas
```

3. Inicializar pesos, bias y tasa de aprendizaje
```

```
w = np.zeros(2)  # 2 entradas → 2 pesos, todos en 0
b = 1            # Bias empieza en 1
learning_rate = 1
```

4. El **entrenamiento** comienza con la **primera época**, que consiste en una pasada completa por todos los datos de entrenamiento.
   Durante esta etapa, el modelo empieza a ajustar sus parámetros (pesos y sesgo) para aprender a hacer predicciones.

- **Se calcula el producto punto entre las entradas y los pesos, más el bias:**

$$z = X[i] \cdot w + b$$

$$z = \sum_{j=1}^{n} x_j \cdot w_j + b = X_i \cdot w + b$$

```
z = np.dot(X, w) + b
print(z)
```

```
[1. 1. 1. 1.]
```

5. Aplicar la función de activación a todo el vector

```
def step_function(x):
    return np.where(x >= 0, 1, 0)

y_predic = step_function(z)
print("Predicciones:", y_predic)
```

```
Predicciones: [1 1 1 1]
```

En la tabla vemos las predicciones

x_1	x_2	y **(real)**	$\hat{y}$ **(predicho)**
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	1

6. Se calcula el error, comparamos la **salida real** `y` con la **predicción** `y_predic` obtenida por el modelo. El error se calcula como:

$$\text{error} = y - \hat{y}$$

```
error = y - y_predic
error
```

```
array([-1, -1, -1,  0])
```

7. Se actulizan los pesos y el bias

$$w = w + \eta \cdot X^{\top} \cdot \text{error}$$

$$w = w + \eta \cdot \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix}^{\top} \cdot \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix}$$

$$w = w + \eta \cdot \begin{bmatrix} x_{11} \cdot e_1 + x_{21} \cdot e_2 + x_{31} \cdot e_3 + x_{41} \cdot e_4 \\ x_{12} \cdot e_1 + x_{22} \cdot e_2 + x_{32} \cdot e_3 + x_{42} \cdot e_4 \end{bmatrix}$$

Donde

- $\mathbf{w}$: vector de pesos.
- η : tasa de aprendizaje (*learning rate*).
- $\mathbf{X}$: matriz de entrada, donde cada fila representa un ejemplo y cada columna una característica.

```
w += learning_rate * np.dot(X.T, error)
b += learning_rate * np.sum(error)

print(w,b)
```

```
[-1. -1.] -2
```

Actividad

Corre la segunda época para el Perceptrón AND

- Asegúrate de que has ejecutado la primera época y de que tienes guardados los valores de los pesos (w) y el bias (b) que obtuviste al final de esa primera fase.

Ejercicio en TensorFlow Playground: Búsqueda de la mejor red neuronal

En este ejercicio se explora cómo los parámetros de una red neuronal afectan su capacidad de **clasificar datos**. Usaremos la herramienta interactiva TensorFlow Playground https://playground.tensorflow.org/, que permite entrenar redes de manera visual y experimentar de forma sencilla.

La idea es que no te limites a correr un modelo una sola vez, sino que explores, compares y observes cómo cambian los resultados dependiendo de tus decisiones.

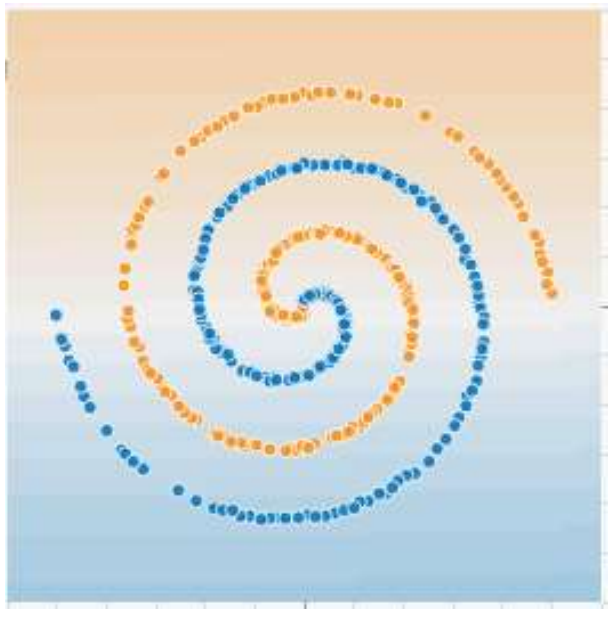
Objetivo

El propósito es **encontrar una configuración de red neuronal que logre separar de la mejor manera posible los puntos naranjas y azules** que aparecen en el espacio de datos.

1. Configuración del dataset

- Ingresa al Playground y selecciona el conjunto de datos que se muesta en Fig.7.

Fig.7.
Data set



Nota. Playground, s. f., https://playground.tensorflow.org/.

`

- Usa únicamente las variables **X1** y **X2** como entradas.
- Mantén un nivel de ruido moderado **0.5**.

2. Experimentación con parámetros
   A continuación, modifica la red poco a poco. Cambia solo un parámetro a la vez y observa el efecto:

- Varía el número de *capas ocultas* .
- Prueba con diferentes cantidades de *neuronas por capa*.
- Cambia la *función de activación* (ReLU, Tanh, Sigmoid).
- Ajusta la *tasa de aprendizaje*: prueba con valores bajos, medios y altos.

3. Evidencia del mejor modelo
   Cuando hayas encontrado la configuración que clasifica mejor:

- *Toma un pantallazo del modelo en TensorFlow Playground.*
- *Guarda la imagen como PDF, nómbralo con tu nombre completo usando guiones bajos (ejemplo: LUISA_FERNANDA_RODRIGUEZ.pdf) y, finalmente, súbelo al siguiente link. https://igacoffice365-my.sharepoint.com/:f:/g/personal/luisaf_rodriguez_igac_gov_co/EkulU8Fas5BAvM3LWRpnsGwBwoxsce8gHZsXm1n0IVjRow?e=Y7XH2P

Conclusiones

- los tensores son estructuras clave para representar datos en múltiples dimensiones, lo que permite modelar entradas, pesos y salidas dentro de una red neuronal de forma flexible y escalable.
- Las funciones de activación introducen no linealidad, permitiendo modelar relaciones complejas.
- Se exploraron conceptos de propagación hacia adelante y retropropagación del error. Este ciclo de ajustes de pesos es el núcleo del aprendizaje en redes neuronales, guiado por funciones de pérdida y algoritmos de optimización como el gradiente descendente.

Referencias

- TensorFlow.NET. (2025). *Chapter 1. Tensor*. https://tensorflownet.readthedocs.io/en/latest/Tensor.html
- García, R. (2021). EL PERCEPTRÓN: Una red neuronal artificial para clasificar datos. *Revista Modelos Matemáticos*. http://www.economicas.uba.ar/institutos_y_centros/revista-modelos-matematicos/
- Analytics Vidhya. (s.f.). *Backpropagation for Dummies*. *Medium*. https://medium.com/analytics-vidhya/backpropagation-for-dummies-e069410fa585
- Robles, R. (s.f.). *Descenso del Gradiente*. https://www.geogebra.org/m/e3whx8bf
- ai.plainenglish. (s. f.). Activation functions in neural networks. Medium. Recuperado de https://ai.plainenglish.io/activation-functions-in-neural-networks-3d8211678fb2
- Aprende IA. (s. f.). *¿Qué son las redes neuronales artificiales? *Recuperado de https://aprendeia.com/que-son-las-redes-neuronales-artificiales/