Inteligencia Artificial

Apuntes de Clase — 14 de agosto de 2025

1st Kendall Rodríguez Camacho
Escuela de Ingenieria en Computación
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
Kenrodriguez@estudiantec.cr

***Abstract*—En este documento se presentan los apuntes correspondientes a la clase del 14 de agosto de 2025 del curso de Inteligencia Artificial. En primer lugar, se incluye un resumen de la sesión anterior, en el que se revisan conceptos de IA y Machine Learning. Posteriormente, se introduce el álgebra lineal aplicada mediante Python, empleando librerías como PyTorch, NumPy y Pandas, con el objetivo de familiarizarse con las herramientas y su implementación práctica.**

***Index Terms*—Machine Learning, MLOps, Models, Deep Learning, Generative AI, Vectores, Tensores, Matrices**

I. Introduction

La Inteligencia Artificial incluye diversas técnicas, entre ellas el Machine Learning o aprendizaje automático, que se encarga de que los modelos aprendan de los datos. En esta clase se presentó un resumen de los diferentes paradigmas de resolución de problemas en IA, los distintos tipos de aprendizaje y de las etapas que componen el ciclo de vida de un modelo de ML. Además, se introdujo el álgebra lineal de forma práctica utilizando Python y librerías como NumPy, Pandas y PyTorch, trabajando con conceptos como tensores, matrices y vectores.

II. Noticias

A. ChatGPT 5, ¿un fracaso?

A pocos días de su lanzamiento por parte de OpenAI, ChatGPT 5 comenzó a recibir críticas por parte de la comunidad. Muchos usuarios cuestionaron si realmente representaba una mejora frente a ChatGPT 4, señalando que en varias ocasiones el modelo tarda demasiado en generar una respuesta y, en algunos casos, produce errores o respuestas de calidad inferior a lo esperado.

B. Alexandr Wang y la inversión de Meta

Alexandr Wang, un empresario de 28 años, es el fundador de Scale AI, una startup especializada en inteligencia artificial y en el etiquetado masivo de datos para entrenar modelos. En 2025, Meta realizó una inversión significativa en la compañía, destacando el interés de la empresa en reforzar su estrategia en IA.

Wang se convirtió en uno de los multimillonarios más jóvenes creados por sí mismos, y su trabajo en Scale AI lo posiciona como una figura relevante en el desarrollo y aplicación de tecnologías de inteligencia artificial avanzada.

III. Tipos de aprendizaje

En Machine Learning existen distintos tipos de aprendizaje, cada uno con aplicaciones y características particulares:

- Supervisado: Se dispone de datos con entradas y salidas conocidas $X = \{X_i, Y_i\}, i = 1..N$. El modelo aprende a predecir la salida Y a partir de la entrada X.
- No supervisado: Solo se tienen entradas sin etiquetas. El modelo identifica patrones o estructuras ocultas en los datos.
- Semi-supervisado: Algunos datos están etiquetados y otros no. Útil cuando etiquetar todos los datos es costoso.
- Auto-supervisado: El modelo genera etiquetas a partir de los propios datos, sin necesidad de intervención manual. Es ampliamente utilizado en procesamiento de lenguaje natural (NLP) y en redes neuronales concurrentes. Ejemplo: predecir la siguiente palabra en una oración.
- Aprendizaje por refuerzo: El agente aprende mediante recompensas y penalizaciones, ajustando sus acciones para maximizar la recompensa futura. Ejemplo: entrenar un agente para jugar Mario Bros, donde el modelo aprende a avanzar, evitar obstáculos y recolectar recompensas.
- Few-shot: Aprende a partir de pocos ejemplos (3-4 muestras). Ejemplo: modelos de lenguaje que generan respuestas correctas con muy pocos ejemplos.
- One-shot: Aprende con un solo ejemplo. Ejemplo: reconocimiento facial usando una sola foto de referencia.
- Zero-shot: Generaliza a tareas nuevas sin ejemplos directos, basándose en conocimientos previos.

Machine Learning visto desde la Ciencia

Desde la perspectiva científica, el fin es generar conocimiento, entender patrones y crear teorías. Las métricas se utilizan para cuantificar qué tan bien funciona un modelo, aportando una base objetiva para comparar resultados y enfoques.

En este contexto, los roles principales incluyen:

- Data Scientist: Experimenta con modelos, selecciona algoritmos, ajusta hiperparámetros y evalúa resultados.
- Research Scientist: Investiga nuevos algoritmos, publica artículos científicos y desarrolla teorías para avanzar la inteligencia artificial.

Machine Learning visto desde la Ingeniería

Desde la perspectiva de la ingeniería, el enfoque está en implementar, mantener y optimizar modelos en producción. Entre las principales tareas se incluyen transformar modelos para reducir su tamaño y aumentar la velocidad, crear pipelines de datos que alimenten los modelos de forma automática y monitorear métricas de rendimiento y tiempos de respuesta.

Además, las prácticas de MLOps se aplican para operacionalizar el aprendizaje automático, de manera similar a cómo DevOps se utiliza para el desarrollo y despliegue de software.

IV. Paradigmas de resolución de problemas

Entre los principales paradigmas de resolución de problemas en Inteligencia Artificial se incluyen:

float

A. Problemas de búsqueda

Se busca encontrar el camino más eficiente hacia una solución.

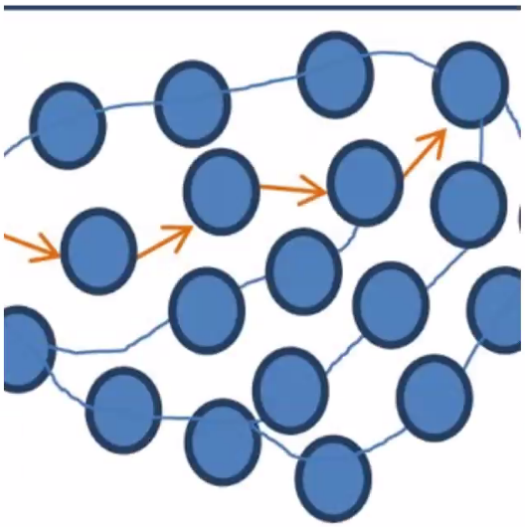

Fig. 1. Ejemplo de problemas de búsqueda.

B. Problemas de optimización

Cuando existe un gran número de soluciones posibles, hallar la mejor solución absoluta puede ser difícil.

1) Solución local: Mejor solución dentro de un área específica.

2) Solución global: Mejor solución en todo el espacio de posibles soluciones.

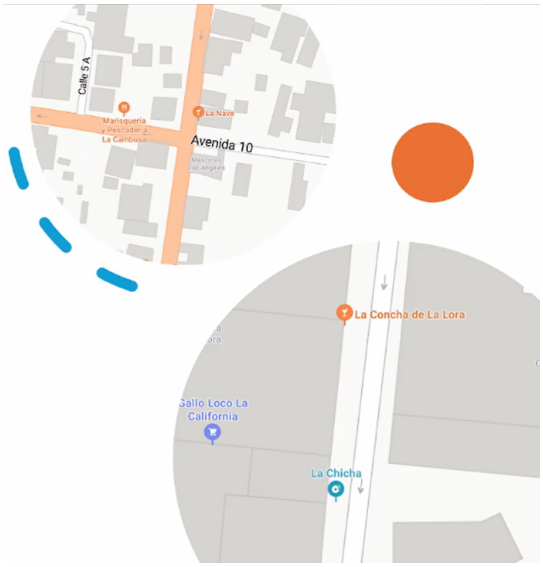

Fig. 2. Ejemplo de soluciones locales y globales en optimización.

C. Predicción y clasificación

1) Predicción: Estimar un valor futuro o desconocido, encontrando patrones según los datos disponibles. Por ejemplo, cuánta gasolina quedará en un carro.

2) Clasificación: Asignar elementos a categorías predefinidas, basándose en sus características. Por ejemplo, identificar el modelo de un carro según sus partes.

Fig. 3. Ejemplo de predicción y clasificación.

D. Agrupamiento (Clustering)

Descubrir patrones o grupos naturales en los datos, utilizando diferentes aspectos para formar grupos con distintas formas.

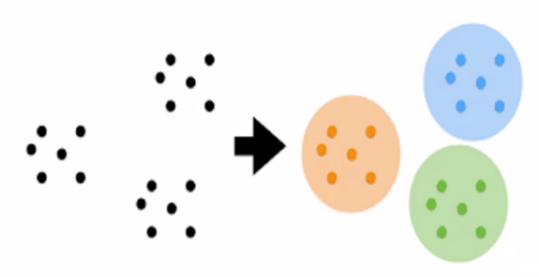

Fig. 4. Ejemplo de agrupamiento (clustering).

V. Modelos

1) Determinista: Un modelo determinista produce siempre el mismo resultado para un mismo conjunto de entradas. *Ejemplo:* ¿Hay luz a mediodía? Para las mismas condiciones, la respuesta siempre será la misma: sí.

2) Estocástico: Un modelo estocástico puede producir diferentes resultados para el mismo conjunto de entradas, dependiendo de probabilidades o factores aleatorios. *Ejemplo:* ¿Cuál será el clima a mediodía? Aunque las condiciones iniciales sean similares, el clima puede variar: puede estar soleado, nublado o lloviendo.

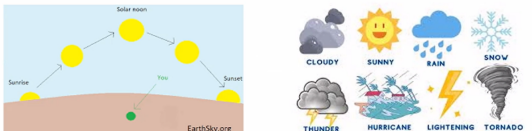

Fig. 5. Comparación entre modelos determinista y estocástico usando el ejemplo del clima.

VI. Jerarquía de la Inteligencia Artificial

1) Inteligencia Artificial (IA): Algoritmos que imitan la inteligencia humana, capaces de tomar decisiones o resolver problemas.

2) Machine Learning (ML): Métodos estadísticos que permiten a los modelos aprender de los datos, como regresión ó árboles de decisión.

3) Deep Learning (DL): Redes neuronales profundas utilizadas para problemas complejos, incluyendo visión por computadora y procesamiento de lenguaje natural (NLP).

4) Generative AI (GenAI): Modelos capaces de generar contenido nuevo, como texto, imágenes o audio, a partir de patrones aprendidos.

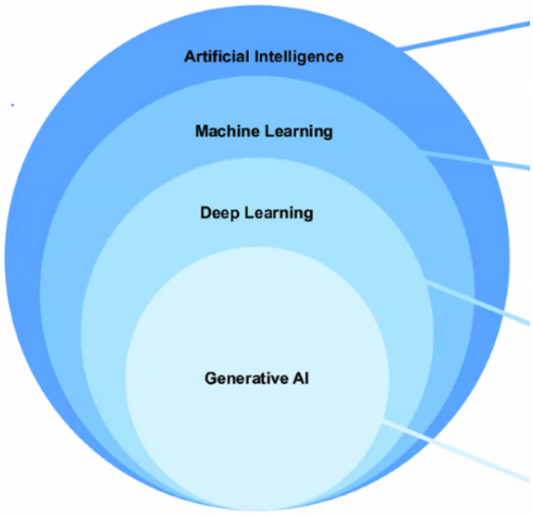

Fig. 6. Jerarquía de la IA

VII. Pipeline de Machine Learning

1) Data Acquisition: Recolectar datos relevantes y de calidad.

2) Data Preparation: Limpiar y transformar los datos, entregándolos en un formato adecuado para su análisis, eliminando duplicados, valores faltantes o incorrectos, y aplicando normalización o escalamiento de los valores.

3) Feature Engineering: Crear y seleccionar las variables (features) más relevantes para entregar al modelo únicamente las necesarias para su entrenamiento y análisis.

4) Model Selection: Elegir el modelo que mejor se adapta al problema, considerando no solo su desempeño, sino también su explicabilidad, es decir, qué tan fácil es interpretar y entender cómo toma decisiones.

5) Model Training: Entrenar el modelo y ajustar los hiperparámetros, utilizando técnicas como Grid Search. Durante este proceso, el modelo realiza optimización basada en los datos para mejorar su desempeño y reducir errores.

6) Model Deployment: Poner el modelo en producción, integrarlo con aplicaciones y monitorear su desempeño.

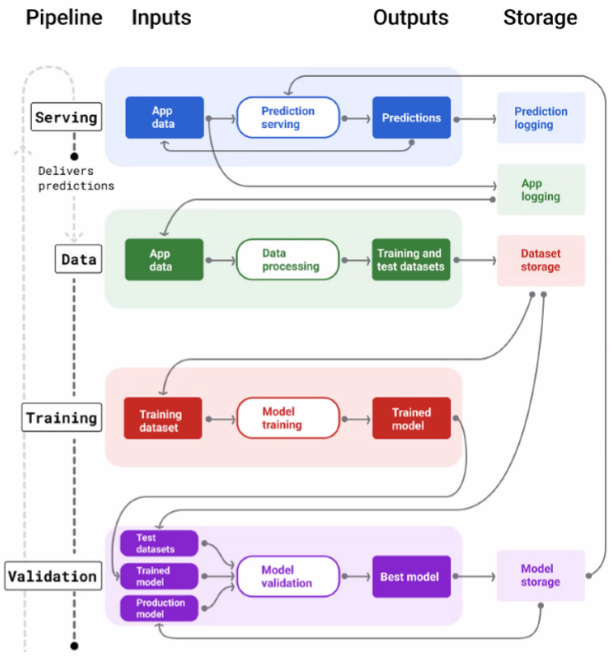

Fig. 7. Pipeline típico de Machine Learning desde la adquisición de datos hasta el despliegue del modelo.

VIII. Manejo y manipulación de datos con PyTorch y Pandas en Python

En esta sección se abordan las técnicas básicas para procesar y manipular datos utilizando PyTorch, junto con Pandas y NumPy. Se explicará cómo crear y transformar tensores, realizar operaciones sobre ellos y cargar datos desde archivos CSV para su análisis y uso en modelos de aprendizaje automático.

A. Anaconda y manejo de ambientes

Anaconda es una distribución que incluye lenguajes como Python y R, junto con herramientas para gestionar paquetes y entornos de forma aislada. Su principal utilidad es permitir manejar dependencias en diferentes ambientes, evitando que un proyecto interfiera con otro, lo cual es muy útil en ciencia de datos y machine learning.

Por ejemplo, para instalar PyTorch en un ambiente específico se puede usar:

```
# Activar el ambiente
conda activate ML  # o el nombre del ambiente

# Instalar PyTorch dentro del ambiente
    activado
pip install torch
```

Esto asegura que PyTorch se instale únicamente en el ambiente seleccionado, sin afectar otros proyectos o configuraciones.

B. Configuración

Para trabajar con datos y tensores, se necesita importar algunas librerías clave:

```
import torch   # Para crear y manipular
    tensores
import numpy as np  # Para arreglos y
    operaciones numericas
import pandas as pd # Para manejar datos
    tabulares
```

C. Creación de tensores

Un tensor es un arreglo de números que puede tener una o varias dimensiones. - Si tiene una dimensión, se llama vector. - Con dos dimensiones se llama matriz. - Con más de dos dimensiones, simplemente se denomina tensor de orden k.

PyTorch permite crear tensores ya inicializados. Por ejemplo, `arange(n)` genera un vector con valores de 0 hasta $n-1$, almacenado en memoria principal y listo para operaciones en CPU.

Cada valor dentro del tensor se llama elemento. Por ejemplo, el tensor `x` creado con `arange(12)` tiene 12 elementos. Se puede inspeccionar el número total de elementos con `numel()` y la forma del tensor (tamaño de cada eje) con `shape`:

```
x = torch.arange(12, dtype=torch.float32)
x.numel()  # 12 elementos
x.shape    # (12,)
```

1) Redimensionamiento (reshape): El método `reshape` permite cambiar la forma de un tensor sin copiar sus datos. Por ejemplo, un vector de 12 elementos puede transformarse en una matriz de 3 filas y 4 columnas:

```
X = x.reshape(3, 4)
X  # Matriz de 3x4
```

2) Tensores pre-inicializados: Se pueden crear tensores ya con valores específicos, como ceros, unos o números aleatorios, útiles por ejemplo para parámetros de modelos:

```
zeros = torch.zeros((2, 3, 4))  # Tensor de
    ceros
ones = torch.ones((2, 3, 4))    # Tensor de
    unos
randn = torch.randn(3, 4)       # Valores
    aleatorios
```

La forma $2 \times 3 \times 4$ indica 2 bloques, cada uno con 3 filas y 4 columnas.

3) Creación de Tensores desde listas de Python: PyTorch permite convertir listas de Python o arreglos de NumPy directamente en tensores:

```
A = torch.tensor([[2, 1, 4, 3],
                  [1, 2, 3, 4],
                  [4, 3, 2, 1]], dtype=torch.
                      float32)
A
```

Esto genera un tensor de 3 filas y 4 columnas con los valores especificados.

D. Indexación y segmentación (slicing)

PyTorch permite acceder a elementos, filas, columnas o submatrices de manera similar a NumPy:

```
fila_ultima = A[-1]      # Ultima fila
submatriz = A[1:3]       # Filas 1 y 2
A[1, 2] = 9      # Asigna 9 al elemento en la
    fila 1, columna 2
A[:2, :] = 35    # Asigna 35 a todas las
    columnas de las dos primeras filas
```

Esto funciona también para tensores de más de dos dimensiones.

E. Operaciones elemento a elemento

En PyTorch, las operaciones aritméticas se aplican elemento por elemento, generando un nuevo tensor:

```
x = torch.tensor([1.0, 2, 4, 8])
y = torch.tensor([2, 2, 2, 2])

add, sub, mul, div, exp = x + y, x - y, x * y,
     x / y, x ** y
```

Esto permite realizar suma, resta, multiplicación, división y potencia de forma directa sobre cada elemento.

F. Concatenación de tensores

Se pueden unir varios tensores en uno solo usando `torch.cat`, especificando el eje sobre el cual concatenar. Por ejemplo, concatenando dos matrices a lo largo de las filas (`dim=0`) se suman las filas, y a lo largo de las columnas (`dim=1`) se suman las columnas:

```
X = torch.arange(12, dtype=torch.float32).
    reshape((3,4))
Y = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3,
    4], [4, 3, 2, 1]])
torch.cat((X, Y), dim=0)  # Concatenar filas
torch.cat((X, Y), dim=1)  # Concatenar
    columnas
```

Además, se puede construir tensores binarios mediante comparaciones. Por ejemplo, `X == Y` genera un tensor donde cada elemento es 1 si coincide y 0 si no:

```
X == Y  # Comparacion elemento a elemento
```

G. Indexación lógica

Se pueden crear máscaras booleanas para seleccionar elementos que cumplan cierta condición. Por ejemplo, comparando dos tensores se obtiene un tensor de valores `True` o `False`:

```
mask = X == Y  # True donde los elementos
    coinciden, False en caso contrario
```

H. Broadcasting

El *broadcasting* permite realizar operaciones entre tensores de distintas formas, expandiendo automáticamente sus dimensiones sin duplicar datos:

```
a = torch.arange(3).reshape((3, 1))  # Forma 3
    x1
b = torch.arange(2).reshape((1, 2))  # Forma 1
    x2
broadcast = a + b  # Tensor resultante de
    forma 3x2
```

I. Operaciones in-place

Las operaciones *in-place* modifican directamente el tensor original, ahorrandose memoria. Esto es útil cuando se maneja muchos parámetros y se quiere evitar crear copias innecesarias.

```
before = id(Y)          # Se guarda la
    direccion de memoria original de Y
Y = Y + X               # Se crea un nuevo
    tensor con la suma; Y ahora apunta a nueva
     memoria
id(Y) == before         # False, la memoria de
    Y cambio

Z = torch.zeros_like(Y)  # Se crea un tensor Z
     con la misma forma que Y, lleno de ceros
Z[:] = X + Y             # Modifica el
    contenido de Z directamente (in-place),
    sin cambiar su direccion de memoria
```

Se recomienda usar in-place para eficiencia, pero con cuidado si varias variables apuntan al mismo tensor, para evitar inconsistencias.

J. Conversión a NumPy

PyTorch permite convertir tensores a arreglos de NumPy y viceversa, sin duplicar los datos en memoria:

```
A_np = A.numpy()           # Tensor a arreglo
    NumPy
type(A_np)

A_back = torch.from_numpy(A_np)  # Arreglo
    NumPy a tensor
type(A_back)
```

K. Carga de datos desde CSV

Para trabajar con datos externos, se puede usar `pandas` y luego convertir a tensores de PyTorch. Se pueden aplicar codificación *one-hot* y completar valores faltantes:

```
import pandas as pd
import torch

df = pd.read_csv('../data/house_tiny.csv')
         # Leer CSV
inputs = df.iloc[:, :2]
                            # Seleccionar
    columnas
inputs = pd.get_dummies(inputs, dummy_na=True)
      # One-hot encoding
```

```
inputs = inputs.fillna(inputs.mean())
             # Completar valores faltantes

X_csv = torch.tensor(inputs.to_numpy(dtype=
    float))  # Convertir a tensor
X_csv
```

IX. Conceptos básicos de álgebra lineal

En esta sección, como continuación de la anterior, se presentan los fundamentos matemáticos que sustentan la manipulación de datos. Se introducen escalares, vectores, matrices y tensores, junto con sus operaciones básicas.

A. Escalar

Un escalar es un valor numérico único que representa una sola cantidad. En PyTorch se puede representar como un tensor con un solo elemento:

```
escalar = 6
```

B. Vector

Un vector es un arreglo unidimensional de escalares. Cada elemento del vector es un escalar.

```
import numpy as np

vector = np.array([1, 2, 3])
```

```
Vector:

[1 2 3]
```

C. Matriz

Al igual que los escalares son tensores de orden 0 y los vectores son tensores de orden 1, una matriz es un tensor de orden 2. Es un arreglo bidimensional de escalares.

```
import numpy as np

matriz = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
```

```
Matriz:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

D. Tensor

Cuando se trabaja con datos de más de dos dimensiones, se utilizan tensores. Son arreglos de orden 3 o superior.

```
import numpy as np

tensor = np.array([[[1, 2, 3], [4, 5, 6]],
                   [[7, 8, 9], [10, 11, 12]]])
```

```
tensor:

[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

E. Hadamard Product

El producto Hadamard corresponde a la multiplicación elemento a elemento de dos matrices o tensores de igual forma. En Python se puede realizar usando el operador `*`.

```
A = np.array([[1, 2],
              [3, 4]])
B = np.array([[5, 6],
              [7, 8]])

C = A * B
```

```
C:

[[ 5 12]
 [21 32]]
```

F. Propiedades básicas de la aritmética de tensores

Sumar o multiplicar un escalar con un tensor produce un tensor de la misma forma que el original, donde cada elemento se ve afectado por el escalar:

```
a = 2
X = torch.arange(24).reshape(2, 3, 4)
a + X          # Suma escalar elemento a
    elemento
(a * X).shape  # Multiplicacion escalar,
    mantiene la forma
```

```
tensor([[[ 2,  3,  4,  5],
         [ 6,  7,  8,  9],
         [10, 11, 12, 13]],
        [[14, 15, 16, 17],
         [18, 19, 20, 21],
         [22, 23, 24, 25]]])
torch.Size([2, 3, 4])
```

1) Reducción: Se pueden sumar los elementos de un tensor usando `sum()`:

```
x = torch.arange(3, dtype=torch.float32)
x, x.sum()  # Vector [0,1,2] y su suma
```

```
tensor([0., 1., 2.]), tensor(3.)
```

Para un tensor multidimensional, `sum()` por defecto reduce todos los ejes:

```
A = torch.arange(6, dtype=torch.float32).
    reshape(2, 3)
A.shape, A.sum()  # Suma de todos los
    elementos
```

```
torch.Size([2, 3]), tensor(15.)
```

Se puede especificar un eje para sumar a lo largo de filas o columnas:

```
A, A.sum(axis=0), A.sum(axis=1)
```

```
tensor([[0., 1., 2.],
        [3., 4., 5.]]),
tensor([3., 5., 7.]),
tensor([3., 12.])
```

Reducir a múltiples ejes simultáneamente es equivalente a sumar todos los elementos:

```
A.sum(axis=[0,1]) == A.sum()
```

```
True
```

La media se calcula con `mean()`, equivalente a la suma dividida por el número de elementos:

```
A.mean(), A.sum()/A.numel()
A.mean(axis=0), A.sum(axis=0)/A.shape[0]
```

```
tensor(2.5000), tensor(2.5000)
tensor([1.5, 2.5, 3.5]), tensor([1.5, 2.5,
    3.5])
```

2) Suma sin reducción: Si se quiere conservar la forma del tensor tras sumar:

```
sum_A = A.sum(axis=1, keepdims=True)

A_normalized = A / sum_A  # Broadcasting para
    normalizar filas
A, sum_A, A_normalized
```

```
tensor([[0., 1., 2.],
        [3., 4., 5.]]),
tensor([[ 3.],
        [12.]]),
tensor([[0.0000, 0.3333, 0.6667],
        [0.2500, 0.3333, 0.4167]])
```

3) Suma acumulada: Se puede calcular la suma acumulada con `cumsum()`:

```
A.cumsum(axis=0)  # Suma acumulada a lo largo
    de las filas
```

```
tensor([[0., 1., 2.],
        [3., 5., 7.]])
```

REFERENCES

[1] Steven Pachecho Portuguez, *Clase sobre Algebra Lineal y manipulación de datos con PyTorch, Pandas y NumPy*, Tecnológico de Costa Rica, 2025.