Tensores, Autograd y Grafos Computacionales

Cómo PyTorch calcula gradientes (sin magia)

Cesar Garcia

2025

Introducción

Objetivos de la sesión

- Entender **cómo** PyTorch representa cálculos matemáticos
- Ver **de dónde salen los gradientes**
- Interpretar `.backward()` como recorrido de un grafo
- Preparar la implementación manual de una red neuronal

¿Qué parte del entrenamiento de una red te resulta más "mágica" hasta ahora?

¿Qué es un Tensor?

Más que un array

Un **tensor** es una generalización matemática:

- Escalar $\rightarrow$ tensor 0D
- Vector $\rightarrow$ tensor 1D
- Matriz $\rightarrow$ tensor 2D
- Tensor ND $\rightarrow$ imágenes, secuencias, lotes

En PyTorch, un tensor **no solo almacena datos**, también participa en un grafo.

¿Qué tipo de tensor representaría una imagen RGB de 224×224?

Tensores en PyTorch

Propiedades relevantes para deep learning

- Tienen forma (`shape`)
- Tienen tipo (`dtype`)
- Pueden vivir en CPU o GPU
- **Pueden recordar operaciones**
  Esta "memoria" es la base del cálculo automático de gradientes.

¿Por qué un array de NumPy no es suficiente para entrenar una red neuronal?

Gradientes

Sensibilidad local

El gradiente mide:

Cómo cambia una salida ante cambios pequeños en un parámetro

En aprendizaje automático:

Parámetros = pesos y sesgos

Gradientes indican ***cómo ajustarlos***

Los gradientes son **locales**, no globales.

Si un gradiente es cero, ¿qué nos dice eso sobre ese parámetro?

El problema clásico

Derivadas a mano no escalan

- Redes reales tienen millones de parámetros
- Cada parámetro requiere una derivada parcial
- El proceso manual es **inviable en la práctica**

  Necesitamos un sistema automático y confiable.

¿El problema es matemático o práctico? ¿Por qué?

Autograd

Diferenciación automática en PyTorch

Autograd es el sistema que:

1. Registra operaciones matemáticas
2. Construye un **grafo computacional**
3. Aplica la **regla de la cadena**

El grafo se construye **dinámicamente**, mientras el código se ejecuta.

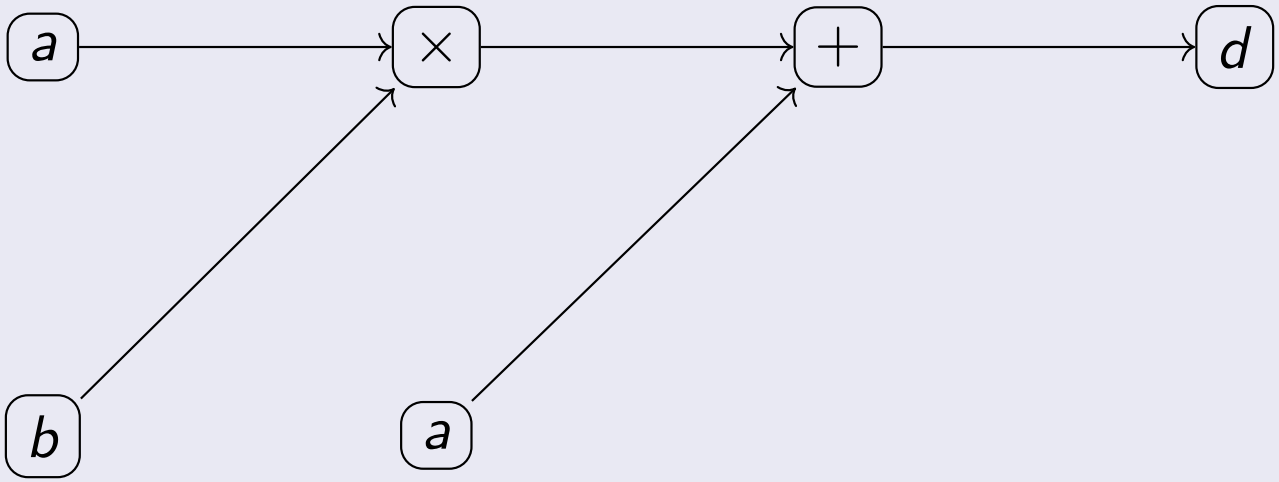
¿Por qué es importante que el grafo sea dinámico y no fijo?

Grafo Computacional

Formula

$$d = a \cdot b + a$$

Intuición visual



Cada nodo representa **una operación real ejecutada en el código**.

¿Por qué el tensor a aparece dos veces en este grafo?

Forward y Backward

Dos fases del mismo proceso

- **Forward pass**: calcular valores
- **Backward pass**: propagar gradientes

  El backward pass no "invierte" el cálculo:
  Aplica la regla de la cadena siguiendo las dependencias del grafo

¿El backward pass necesita volver a ejecutar el forward? ¿Por qué?

Qué NO es magia

Desmitificando el entrenamiento

- PyTorch no “adivina” derivadas
- No hay reglas especiales para redes neuronales
- Todo se reduce a:
  - sumas
  - multiplicaciones
  - regla de la cadena

Las redes neuronales son **grafos grandes**, no objetos especiales.

¿Qué operación básica aparece en todas las capas de una red?

¿Por qué esto importa?

Conexión con redes neuronales

- Cada capa = operaciones en un grafo
- Entrenar = recorrer el grafo hacia atrás
- Optimizadores = usar los gradientes calculados

Sin **autograd**, no hay deep learning moderno.

¿Qué parte de este proceso cambiaría si la red tuviera más capas?

Resumen

- Tensores almacenan datos **y relaciones**
- Autograd construye grafos computacionales
- `.backward()` aplica la regla de la cadena
- Esto es la base de backpropagation

¿Qué concepto de esta sesión es imprescindible para construir una red desde cero?