Clase 122 — PyTorch fundamentos: tensores, autograd, nn.Module

Parte: 2 — Deep Learning · Fuente: PyTorch tutorials + Howard & Gugger, Deep Learning for Coders with fastai & PyTorch. Duración estimada: 90 min.

Clase 122 — PyTorch fundamentos: tensores, autograd, nn.Module

> *Parte: 2 — Deep Learning · Fuente: PyTorch tutorials + Howard & Gugger, Deep Learning for Coders with fastai & PyTorch. Duración estimada: 90 min.*

Objetivo

Aprender PyTorch —el framework dominante en research y en LLMs/multimodal 2026—. Cubrir: tensores (similar a NumPy, en GPU), autograd (requires_grad, .backward()), nn.Module (forma de definir modelos), Dataset/DataLoader para data pipelines. Equivalencias 1:1 con Keras/TF de las clases anteriores.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Crear tensores: torch.tensor, torch.zeros, torch.randn, device='cuda'.
- Aplicar autograd: x.requires_grad_(True); y = f(x); y.backward(); x.grad.
- Definir un MLP custom: class Net(nn.Module): def __init__(self): ...; def forward(self, x): ....
- Escribir el loop manual: optimizer.zero_grad(); loss.backward(); optimizer.step().
- Usar Dataset y DataLoader para pipelines de datos.

Temas

- Tensors vs ndarray, .to(device).
- Computation graph dinámico (vs estático TF1) — define-by-run.
- requires_grad y autograd.
- nn.Module, nn.Linear, nn.Sequential, nn.functional.
- Loss funcs: nn.CrossEntropyLoss, nn.MSELoss.
- Optim: torch.optim.Adam(model.parameters(), lr=...).
- Dataset + DataLoader(num_workers, pin_memory).

Definiciones y características

- torch.tensor: similar a np.ndarray pero con GPU + autograd.
- nn.Module: clase base de todos los modelos. __init__ declara capas, forward define forward.
- nn.Parameter: tensor con requires_grad=True registrado automáticamente.
- optimizer.zero_grad(): OBLIGATORIO al inicio de cada batch (sino se acumulan).
- model.train() / model.eval(): cambia comportamiento de BatchNorm y Dropout.
- with torch.no_grad():: contexto para inference, no construye grafo.

Dataset / recursos

- Fashion-MNIST vía torchvision.datasets.
- Librerías: torch, torchvision.

Ejercicios

1. Tensores: crear, mover a GPU, operaciones básicas. Comparar con NumPy.
2. Autograd: x = torch.tensor([2.0], requires_grad=True); y = x**3; y.backward(); print(x.grad) → debe ser 12.
3. MLP custom: definir clase con 2 nn.Linear + ReLU. Verificar model.parameters().
4. Training loop manual: Fashion-MNIST, 1 época, reportar loss.
5. DataLoader: DataLoader(dataset, batch_size=32, shuffle=True, num_workers=2). Iterar.

Homework verificable

Reproducir el modelo de Fashion-MNIST de Clase 091 en PyTorch:

1. MLP [300, 100, 10].
2. CrossEntropy loss, Adam(1e-3).
3. EarlyStopping manual (cuando val_loss no baja por 5 épocas).
4. Reportar test accuracy.

Criterio de aceptación: accuracy ≥ 0.87 (igual que la versión Keras); código compacto y legible.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Olvido optimizer.zero_grad()	Gradientes acumulados. Fix: agregar al ini
Modelo no aprende	Olvidaste .to(device) en model o data. Fix
RuntimeError: Trying to backward through t	Reusar el grafo. Fix: loss.backward(retain
Eval con model.train() (BN/Dropout activos	Métricas distorsionadas. Fix: model.eval()
DataLoader lento	num_workers=0. Fix: num_workers=4, pin_mem

Preguntas frecuentes

¿PyTorch o TF/Keras?

PyTorch para research, LLMs, multimodal, HuggingFace ecosystem. Keras para tabular/imagen estándar. Saber ambos.

¿torch.compile mejora velocidad?

Sí — model = torch.compile(model) en PyTorch 2.0+: 2-5× speedup automático sin cambiar código.

¿Equivalente de Keras fit?

PyTorch puro no lo tiene. Lo provee Lightning (clase 108b) y HuggingFace Trainer.

¿nn.Sequential o nn.Module custom?

Sequential para pilas lineales. Custom Module para cualquier topología no trivial.

¿GPU mac (Apple Silicon)?

device='mps' desde PyTorch 1.12. Funcional, ~50-80 % de NVIDIA performance.

Referencias

- PyTorch tutorials.
- Howard & Gugger, Deep Learning for Coders with fastai & PyTorch.
- Paszke et al. (2019), PyTorch: An Imperative Style, High-Performance Deep Learning Library, NeurIPS.

Siguiente clase

Clase 123 — PyTorch Lightning: Trainer, callbacks, distributed

Apéndice: notebook (primer bloque)

Intentamos import torch. Si no está, fallback con numpy y autograd manual.

```
USE_TORCH = False
try:
    import torch
    USE_TORCH = True
    print('torch:', torch.__version__, '| device cpu')
except Exception as e:
    print('torch no disponible. Fallback numpy + autograd manual. Motivo:', type(e).__name__)
import numpy as np
np.random.seed(42)
```

Archivos complementarios

- notebook.ipynb