Clase 121 — Custom training loops (+ PyTorch & PyTorch Lightning)

Parte: 2 — Deep Learning · Fuente: Géron, cap. 12 § Custom Training Loops + docs PyTorch, Lightning. Duración estimada: 85 min.

Clase 121 — Custom training loops (+ PyTorch & PyTorch Lightning)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 12 § Custom Training Loops + docs PyTorch, Lightning.*
> *Duración estimada: 85 min.*

Objetivo

Escribir un training loop manual en TF con GradientTape — control absoluto sobre cada paso (útil para GANs, RL, multi-step optimizers, debugging). Conocer el equivalente en PyTorch (el framework dominante de la industria en 2026) y cómo PyTorch Lightning abstrae los boilerplate del loop, devolviendo la productividad de Keras con la flexibilidad de PyTorch.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Escribir un training loop TF con for batch in dataset: with GradientTape() as tape: ...; grads = tape.gradient(loss, vars); optimizer.apply_gradients(...).
- Hacer el equivalente en PyTorch: optimizer.zero_grad(); loss.backward(); optimizer.step().
- Usar PyTorch Lightning para el mismo problema con boilerplate mínimo (LightningModule.training_step).
- Reconocer cuándo el loop manual es necesario (modelo con dos optimizadores, schedule custom step-wise, RL).
- Comparar productividad y flexibilidad de los 3 frameworks.

Temas

- Loop básico TF: epochs → batches → tape → grads → apply.
- tape.watch(...) para watch tensors que no son tf.Variable.
- Métricas y logging manual.
- Equivalente PyTorch.
- Lightning como capa de abstracción.
- Complemento moderno: PyTorch + Lightning en paralelo a TF/Keras.

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 108a — PyTorch fundamentos: tensores, autograd, nn.Module
- Clase 108b — PyTorch Lightning: Trainer + distributed

Definiciones y características

- GradientTape: contexto TF que registra operaciones para autodiff.

- tape.watch(t): forzar a tape a registrar un tensor no-Variable.
- tape.gradient(loss, vars): calcular gradientes.
- optimizer.apply_gradients(zip(grads, vars)): aplicar la update.
- PyTorch .backward(): aplica autodiff sobre el grafo dinámico construido en el forward.
- requires_grad: flag PyTorch que indica si un tensor necesita gradientes.

Dataset / recursos

- Fashion-MNIST o MNIST.
- Librerías: tensorflow, torch, lightning (pip install lightning).

Ejercicios

1. TF loop manual: entrenar un MLP en Fashion-MNIST con GradientTape. Implementar logging de loss y métrica manual.
2. PyTorch equivalente: reimplementar el mismo loop en PyTorch.
3. Lightning: mismo problema con LightningModule.
4. Speedup con jit: tf.function en TF; torch.compile en PyTorch. Medir.
5. Multi-optimizer: con TF, escribir un loop que aplica un optimizer para las capas frozen-ish (LR bajo) y otro para las nuevas (LR alto). Esto en model.fit requiere mucho boilerplate.

Homework verificable

Implementar el mismo problema (Fashion-MNIST, MLP [256, 128]) en los 3 frameworks:

1. TF con custom training loop.
2. PyTorch puro.
3. PyTorch Lightning.

Reportar para cada uno: # líneas de código, tiempo de training, accuracy.

Criterio de aceptación: las 3 versiones llegan a accuracy similar (±0.5 pp). Lightning ≈ Keras en LOC; PyTorch puro tiene más boilerplate; TF custom loop también.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Loss no baja en TF custom loop	Olvidaste tape.gradient(loss, model.traina
En PyTorch, gradientes acumulados batch a	Olvidaste optimizer.zero_grad() al inicio
RuntimeError: cudnn error en PyTorch	Versión de PyTorch incompatible con CUDA i
Lightning espera LightningDataModule cuand	Lightning acepta ambos; verificar firma de
BatchNorm no se actualiza en TF custom loo	Hay que pasar training=True explícitamente

Preguntas frecuentes

¿TF o PyTorch para empezar?

Keras (TF) es más amigable para empezar. PyTorch es más demandado en producción 2026. Saber ambos

es el estándar profesional.

¿Lightning vs Keras de cuál sale mejor?

Lightning te abre el ecosistema PyTorch (Hugging Face). Keras 3 ahora soporta backends multi (TF, JAX, PyTorch) — la diferencia disminuye.

¿zero_grad() por qué existe en PyTorch?

Para permitir gradient accumulation (varios .backward() sin step para batches grandes virtuales). Si no necesitás eso, llamá zero_grad() al inicio del batch siempre.

¿Distributed training más fácil en cuál?

Lightning, por mucho. trainer = L.Trainer(strategy='ddp', devices=4) y ya. En TF necesitás tf.distribute.MirroredStrategy con strategy.scope():. PyTorch puro requiere torch.distributed.init_process_group manual.

¿Hugging Face usa PyTorch o TF?

Casi todo PyTorch desde 2022. Los modelos modernos (Llama, Mistral, Mixtral, etc.) están solo en PyTorch.

Referencias

- Géron, cap. 12 — Custom Training Loops.
- PyTorch tutorials.
- PyTorch Lightning docs.
- Falcon et al. (2019), PyTorch Lightning.
- Hugging Face — Trainer (built on Lightning concepts).

Siguiente clase

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

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb