Clase 120 — Funciones y grafos (autograph)

Parte: 2 — Deep Learning · Fuente: Géron, cap. 12 § TF Functions and Graphs. Duración estimada: 55 min.

Clase 120 — Funciones y grafos (autograph)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 12 § TF Functions and Graphs. Duración estimada: 55 min.*

Objetivo

Entender qué hace @tf.function —compila una función Python a un grafo TF estático, acelerando 2-10× y permitiendo deploy en TF Serving / TFLite—. Conocer AutoGraph (traduce automáticamente if/for/while Python a operaciones TF), saber los gotchas clásicos (efectos colaterales, prints, listas Python) y cuándo el decorator deteriora la experiencia de debugging.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar @tf.function a una función custom y verificar speedup.
- Identificar cuándo NO usarlo (debugging, lógica con efectos colaterales no determinísticos).
- Entender retracing: cada vez que cambias la shape o dtype del input, TF reconstruye el grafo.
- Usar tf.function(input_signature=...) para evitar retracing.
- Diferenciar eager mode (default, dinámico) de graph mode (compilado).

Temas

- Eager vs graph execution.
- @tf.function y AutoGraph.
- Retracing: por qué pasar Python ints vs tensors causa retraces.
- Print en grafo: tf.print (corre en graph) vs print (solo en tracing).
- Cuándo @tf.function vale la pena (loops, training step) y cuándo no (one-shot, debugging).

Definiciones y características

- Eager mode: cada op se ejecuta inmediatamente. Default en TF 2+. Como NumPy.
- Graph mode: las ops se ensamblan en un grafo y se ejecutan después. Más rápido, deployable.
- @tf.function: convierte una función Python en un ConcreteFunction (grafo).
- AutoGraph: subsistema que traduce if/for/while a tf.cond/tf.while_loop.
- Retracing: rebuild del grafo cuando cambian shape/dtype de inputs. Caro.
- input_signature: tupla de TensorSpec que fija las shapes esperadas → no retracing.

Dataset / recursos

- Operaciones aisladas para medir velocidad.
- Librerías: tensorflow, time.

Ejercicios

1. Speedup básico: definir def f(x): return tf.reduce_sum(x * 2 + 3x + 1). Medir tiempo eager vs @tf.function-wrapped en un loop de 10 000 iteraciones.
2. Retracing: definir una función con @tf.function. Llamarla con tensores de shapes distintas; usar tf.config.experimental_run_functions_eagerly(True) y print para detectar retraces.
3. AutoGraph: una función con un for Python y un if. Verificar que se convierte correctamente (tf.autograph.to_code(f)).
4. tf.print vs print: dentro de @tf.function, demostrar que print solo se ejecuta en tracing (1ª llamada), tf.print siempre.
5. input_signature: fijar input_signature para evitar retracing con shape (None, 784).

Homework verificable

Implementar un training loop minimalista (sin Keras) con y sin @tf.function:

1. Modelo simple en numpy/TF (regresión lineal).
2. Loop manual de 1 000 iteraciones.
3. Mismo loop wrapped con @tf.function.
4. Comparar tiempo total.

Criterio de aceptación: la versión @tf.function debe ser ≥ 2× más rápida (en CPU, más en GPU).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
print no aparece dentro de @tf.function de	Solo corre en tracing. Fix: tf.print para
Función retrace en cada llamada	Pasás Python ints/strings como argumentos.
list.append() dentro de @tf.function no fu	Listas Python no se preservan en grafo. Fi
if x > 0: con tensor x → error o warning	AutoGraph debería convertirlo a tf.cond. F
np.random dentro de @tf.function siempre d	Se evalúa una vez en tracing. Fix: usar tf

Preguntas frecuentes

¿Cuándo @tf.function?

Para funciones que se llaman muchas veces (training step, custom layers complejas). NO para funciones one-shot, debugging, o muy simples.

¿@tf.function y debugging?

Hacé el debugging en eager (sin decorator). Una vez que funciona, agregás @tf.function para producción.

¿Cómo veo el grafo generado?

tf.autograph.to_code(f.python_function) te da el Python equivalente que generó AutoGraph. Útil para entender qué hizo.

¿jit_compile=True qué hace?

@tf.function(jit_compile=True) activa XLA, otra capa de optimización (fusión de ops). Mejora velocidad otro 1.5-3×, especialmente en TPU.

¿En Keras hace falta?

No para model.fit (ya está jitteado). Sí para custom training loops (clase 108) y para custom layers/models complejos.

Referencias

- Géron, cap. 12 — TF Functions and Graphs.
- TF — Better performance with tf.function.
- TF — AutoGraph.

Siguiente clase

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

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb