Clase 124 — tf.data API

Parte: 2 — Deep Learning · Fuente: Géron, cap. 13 § The Data API. Duración estimada: 65 min.

Clase 124 — tf.data API

Parte: 2 — Deep Learning · Fuente: Géron, cap. 13 § The Data API. Duración estimada: 65 min.

Objetivo

Construir pipelines de datos eficientes con tf.data.Dataset: leer desde memoria/archivos/CSV, transformar (map, filter), mezclar (shuffle), batchear (batch), prefetch (paraleliza CPU↔GPU). Saber por qué un buen pipeline de datos es la diferencia entre "GPU al 30 %" y "GPU al 95 %".

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Crear datasets desde varias fuentes: from_tensor_slices, list_files, TextLineDataset.
- Encadenar transformaciones: .map(fn, num_parallel_calls=tf.data.AUTOTUNE), .filter, .shuffle(buffer), .batch(N), .prefetch(tf.data.AUTOTUNE).
- Reconocer el orden correcto: cache → shuffle → batch → prefetch.
- Usar tf.data.AUTOTUNE y profilear con TensorBoard Profiler.
- Saber cuándo cache() vale la pena (datasets que caben en RAM).

Temas

- Lazy evaluation: el dataset es un grafo, no datos cargados.
- shuffle(buffer_size): buffer chico → mal mezclado; buffer = dataset_size → perfecto pero RAM.
- batch(N) → cada elemento es ahora un mini-batch.
- prefetch: solapamiento CPU (loading) con GPU (training).
- interleave para leer múltiples archivos en paralelo.
- Métricas: tf.data.experimental.assert_cardinality.

Definiciones y características

- Dataset.from_tensor_slices: convierte un array NumPy en dataset.
- map(fn, num_parallel_calls=AUTOTUNE): aplica fn a cada elemento; AUTOTUNE elige hilos.
- shuffle(buffer_size): mezclado con buffer rotativo.
- batch(N, drop_remainder=False): agrupa en batches.
- prefetch(n): precarga n batches mientras la GPU procesa el actual.
- cache(): guarda en memoria (o archivo) después de la primera época.
- AUTOTUNE: TF mide y ajusta el número de hilos automáticamente.

Dataset / recursos

- Fashion-MNIST cargado vía tf.data.
- CSV grande para testear pipelines a archivo (anticipa 110 TFRecord).
- Librerías: tensorflow.

Ejercicios

1. Dataset desde NumPy: ds = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(1024).batch(32).prefetch(tf.data.AUTOTUNE). Iterar y verificar shapes.
2. Map con normalización: .map(lambda x, y: (tf.cast(x, tf.float32)/255., y), num_parallel_calls=tf.data.AUTOTUNE).
3. Cache: comparar tiempo del 1er epoch vs 2do epoch con y sin .cache().
4. Buffer chico: comparar shuffle con buffer_size=10 vs buffer_size=len(data). Inspeccionar el primer batch.
5. Profilear: usar tf.profiler (vía TensorBoard) y verificar dónde está el bottleneck — data loading vs compute.

Homework verificable

Pipeline production-ready para Fashion-MNIST:

1. from_tensor_slices con shuffle, augmentation simple (random flip), normalización, batch=128, prefetch.
2. Cachear el dataset (entra en RAM).
3. Train un MLP por 10 épocas; medir tiempo total.
4. Comparar contra pasar x, y directo a model.fit.

Criterio de aceptación: el pipeline tf.data debe igualar o ser un poco más rápido que pasar arrays directos; con cache, el 2do epoch en adelante es notablemente más rápido (skip de I/O y map).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
shuffle().batch() muy mal mezclado	Buffer = 1000 con dataset de 60 000 → veci
Cache + shuffle en orden incorrecto	.cache().shuffle(...): ok. .shuffle(...).c
.batch(32, drop_remainder=False) y el últi	A veces deseado, a veces rompe modelos cus
GPU idle 70 %	El pipeline es el bottleneck. Fix: prefetc
.map(py_function(...)) lento	Las py_function no se vectorizan ni se eje

Preguntas frecuentes

¿Cuál es el orden canónico de las ops?

Dataset → map → cache → shuffle → batch → prefetch. Cache después de map (para no remap), antes de shuffle (cache de elementos individuales).

¿prefetch(AUTOTUNE) vs prefetch(N)?

AUTOTUNE casi siempre. Mide y ajusta. Solo fijá N si tenés constraints muy específicos.

¿batch(32) o batch(128)?

Depende del modelo y la GPU. Imágenes en VRAM grande: 128. Modelos grandes/Transformers: 8-32. Probá y mirá VRAM.

¿tf.data o PyTorch DataLoader?

Si usás TF: tf.data. PyTorch: DataLoader (similar concepto, num_workers). En 2026, los pipelines de Hugging Face usan datasets library que tiene buen interfaz para ambos.

¿Pipeline en GPU?

Algunas ops sí (decode JPG en GPU con nvidia.dali o tf.image). En general el pipeline corre en CPU y el modelo en GPU; prefetch solapa.

Referencias

- Géron, cap. 13 — The Data API.
- TF — tf.data: Build TensorFlow input pipelines.
- TF — Better performance with the tf.data API.

Siguiente clase

Clase 125 — TFRecord

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb