Clase 167 — ONNX y ONNX Runtime: portabilidad e inference optimizada

Parte: 2 — Deep Learning · Fuente: ONNX docs + ONNX Runtime team blogs. Duración estimada: 80 min.

Clase 167 — ONNX y ONNX Runtime: portabilidad e inference optimizada

> *Parte: 2 — Deep Learning · Fuente: ONNX docs + ONNX Runtime team blogs. Duración estimada: 80 min.*

Objetivo

Dominar ONNX (formato intermedio) y ONNX Runtime (runtime cross-platform) — la solución portable para inference: entrenás en TF/PyTorch/JAX, exportás a ONNX, corrés en cualquier hardware (CPU, GPU NVIDIA, GPU AMD, mobile, browser, edge). Conocer TensorRT (NVIDIA-specific, mayor performance).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Exportar modelos: torch.onnx.export, tf2onnx.convert.
- Cargar e inferir con onnxruntime.InferenceSession.
- Elegir execution provider: CPU, CUDA, TensorRT, OpenVINO, DirectML, CoreML.
- Optimizar modelos: graph optimization, quantization int8/fp16.
- Reconocer cuándo ONNX vs TF Serving vs vLLM (LLMs).

Temas

- ONNX como protocolo (protobuf): operator set + tensor types.
- Conversión: cada framework tiene su exporter.
- Verificación: outputs deben coincidir framework ↔ ONNX (±1e-5).
- Optimization: graph simplification, layer fusion.
- Quantization: dynamic, static (con calibration), QAT.
- Execution providers: priority order.

Definiciones y características

- ONNX: formato; archivo .onnx.
- Opset version: versión del set de ops. Higher = newer ops (ej. opset 17 incluye attention).
- InferenceSession: objeto runtime que carga modelo y ejecuta.
- Execution Provider (EP): backend de hardware. ORT elige el primero disponible.
- onnxruntime-gpu: pip package específico para CUDA.
- onnxruntime-directml: Windows GPU (AMD/Intel via DirectX).

Dataset / recursos

- Modelo de clases anteriores (Fashion-MNIST o ResNet preentrenado).
- Librerías: onnx, onnxruntime o onnxruntime-gpu, tf2onnx (TF), torch.onnx built-in.

Ejercicios

1. PyTorch → ONNX: torch.onnx.export(model, dummy_input, 'model.onnx', opset_version=17, input_names=['input'], output_names=['output']).
2. TF → ONNX: python -m tf2onnx.convert --saved-model dir/ --output model.onnx --opset 17.
3. Inference: sess = ort.InferenceSession('model.onnx', providers=['CUDAExecutionProvider', 'CPUExecutionProvider']). Verificar outputs.
4. Graph optimization: sess_options.graph_optimization_level = ort.GraphOptimizationLevel.ORT_ENABLE_ALL.
5. Quantization: onnxruntime.quantization.quantize_dynamic('model.onnx', 'model_q.onnx', weight_type=QuantType.QUInt8).

Homework verificable

Exportar y deployar un modelo CNN ya entrenado:

1. PyTorch ResNet50 preentrenado → ONNX.
2. Quantization dynamic.
3. Comparar latencia + accuracy: PyTorch CUDA vs ORT CUDA vs ORT CPU vs ORT CPU quantized.
4. Verificar correctness (output diff < 1e-4 vs PyTorch).

Criterio de aceptación: ORT CUDA ≥ PyTorch CUDA en velocidad; quantization reduce tamaño 4× con < 1 pp accuracy loss.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
opset version X not supported	EP version vieja. Fix: actualizar onnxrunt
Output diff vs framework grande	Conversión incompleta (custom ops). Fix: v
CUDA provider no disponible	Falta onnxruntime-gpu install. Fix: pip in
Dynamic axes mal	Mal export. Fix: dynamic_axes={'input': {0
Quantization rompe accuracy	Modelo sensible. Fix: usar static quantiza

Preguntas frecuentes

ONNX vs TensorRT?

ONNX Runtime + TensorRT EP combina ambos: ORT como interfaz, TensorRT como backend para GPUs NVIDIA. Best of both.

ONNX en mobile?

ONNX Runtime Mobile: optimizado para Android/iOS. Soporta CoreML, NNAPI delegates.

ONNX en browser?

ONNX.js — corre en WebGL/WebGPU/WASM.

LLMs con ONNX?

Posible (especialmente con onnxruntime-genai). vLLM/TGI siguen siendo mejores para LLMs grandes.

Triton vs ORT?

Triton (NVIDIA) puede servir modelos ONNX como uno de sus backends. Triton para infra multi-modelo; ORT solo para inference.

Referencias

- ONNX.
- ONNX Runtime.
- ONNX Runtime Mobile.
- ONNX Runtime GenAI.

Siguiente clase

Clase 168 — Despliegue en Vertex AI

Apéndice: notebook (primer bloque)

Entrenamos sklearn LogReg, exportamos a ONNX (con fallback) y benchmarkeamos inferencia.

```
import numpy as np, time, json, pickle
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
rng = np.random.default_rng(42)

X, y = make_classification(n_samples=2000, n_features=20, n_informative=10, random_state=42)
X_train, X_test = X[:1500], X[1500:]
y_train, y_test = y[:1500], y[1500:]

clf = LogisticRegression(max_iter=500).fit(X_train, y_train)
print(f'sklearn acc test: {clf.score(X_test, y_test):.3f}')
```

Archivos complementarios

- notebook.ipynb