Clase 170 — TensorFlow.js (navegador)

Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Running a Model in a Web Page + TF.js docs. Duración estimada: 55 min.

Clase 170 — TensorFlow.js (navegador)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Running a Model in a Web Page + TF.js docs. Duración estimada: 55 min.*

Objetivo

Servir modelos directamente en el navegador con TensorFlow.js — corre client-side (privacidad, sin server cost, sin latency network). Alternativas modernas: ONNX Runtime Web, WebGPU-accelerated inference, transformers.js (Hugging Face) para LLMs en el browser.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Convertir un Keras model a TF.js format con tensorflowjs_converter.
- Cargar y hacer inference desde JavaScript en el browser.
- Conocer WebGL backend (default TF.js) y WebGPU (nuevo, más rápido).
- Usar transformers.js para correr modelos NLP/visión en browser.
- Reconocer cuándo conviene client-side vs server-side.

Temas

- Conversion: tensorflowjs_converter --input_format=keras model.keras tfjs_model/.
- JS API: const model = await tf.loadLayersModel('tfjs_model/model.json').
- Backends: WebGL (default), WASM, WebGPU.
- transformers.js: ONNX en browser, soporta BERT, GPT-2, Whisper.
- Edge cases: tamaño del modelo (~5-50 MB ideal), latencia primer load.

Definiciones y características

- TF.js: implementación de TF en JavaScript.
- WebGL backend: usa GPU del browser, default.
- WebGPU: API moderna, 2-5× más rápida que WebGL.
- WASM backend: CPU-only pero portable.
- transformers.js: HuggingFace en browser, ONNX-based.

Dataset / recursos

- Modelo Fashion-MNIST.
- Librerías: tensorflowjs, tensorflowjs-converter.

Ejercicios

1. Convert: tensorflowjs_converter --input_format=keras_saved_model servable/ tfjs_model/. Inspeccionar archivos.

2. HTML page: pagina simple que carga model.json, dibuja una imagen 28×28 en canvas, predice.
3. WebGPU: await tf.setBackend('webgpu'). Comparar velocidad vs WebGL.
4. transformers.js: import { pipeline } from '@xenova/transformers'. Correr sentiment-analysis en browser. 1 línea.
5. PWA: empaquetar como Progressive Web App con service worker para offline inference.

Homework verificable

Demo de Fashion-MNIST en browser:

1. Convertir modelo a TF.js.
2. Página HTML con canvas donde el usuario dibuja.
3. Botón "Predict" → muestra clase + probabilidades.

Criterio de aceptación: la demo funciona en Chrome/Firefox; predicciones son razonables para dibujos sencillos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
model.json not found	Path mal o servidor no sirve archivos está
OOM en browser con modelo grande	TF.js no es para GB-scale models. Fix: mod
WebGL fail en algunos browsers	Falta WebGL2. Fix: fallback a WASM.
Predicciones diferentes vs server	Diferencia en preprocesamiento. Fix: hacer
Modelo carga lento	Sin caching. Fix: Service Worker + cache.

Preguntas frecuentes

¿TF.js o ONNX Runtime Web?

TF.js para modelos TF; ORT Web para portabilidad. ORT Web también soporta WebGPU.

¿LLMs en browser?

Sí — modelos chicos (DistilBERT, TinyLlama, Whisper tiny) con quantization. WebLLM proyecto usa MLC y WebGPU para Llama 2 7B en browser.

¿Privacidad?

Client-side = data nunca sale del navegador. Ideal para healthcare, finanzas.

¿WebAssembly (WASM) vs WebGL?

WASM es CPU portable, lento. WebGL usa GPU. WebGPU es el futuro (5× WebGL).

¿Cuándo NO usar client-side?

Modelos grandes (>100 MB), datasets propietarios (no quieres exponerlos), latencia inaceptable en primer load.

Referencias

- Géron, cap. 19 — Running a Model in a Web Page.
- TensorFlow.js docs.
- transformers.js.
- ONNX Runtime Web.
- MLC WebLLM.

Siguiente clase

Clase 171 — Aceleración con GPU

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb