Clase 149 — LoRA / QLoRA: fine-tuning eficiente de LLMs

Parte: 2 — Deep Learning · Fuente: Hu et al. (2021) LoRA + Dettmers et al. (2023) QLoRA.

Duración estimada: 100 min.

Clase 149 — LoRA / QLoRA: fine-tuning eficiente de LLMs

> *Parte: 2 — Deep Learning · Fuente: Hu et al. (2021) LoRA + Dettmers et al. (2023) QLoRA. Duración estimada: 100 min.*

Objetivo

Hacer fine-tuning de LLMs (Llama 3, Mistral, Qwen 2) en una GPU consumer con LoRA y QLoRA. Cubrir hyperparámetros (r, alpha, dropout, target_modules), inspección de trainable params, merge LoRA con base, y deployment.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar LoRA con peft.LoraConfig y get_peft_model.
- Configurar QLoRA con bitsandbytes 4-bit + BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_quant_type='nf4').
- Elegir rank r (típico 8-64), alpha (típico 2×r), target_modules.
- Mergear el LoRA adapter con el base para deploy: model.merge_and_unload().
- Calcular trainable params vs total: ~0.1-1 % es el ratio típico.

Temas

- LoRA matemáticamente: W' = W + (B·A)·alpha/r.
- QLoRA: NF4 quantization + double quant + paged optimizers.
- target_modules: ['q_proj', 'k_proj', 'v_proj', 'o_proj'] clásico; o all-linear.
- Save / load: solo el adapter (~10-50 MB) en lugar del full model.
- Merge + serve vs separated adapter.

Definiciones y características

- LoRA: matrices rank-r entrenables agregadas a Linear layers.
- r (rank): 8-16 para tareas chicas; 32-64 para datos grandes/cambios fuertes.
- alpha: scaling. Convención: alpha = 2r.
- NF4: NormalFloat 4-bit, optimizado para weights de redes pre-trained.
- Double quantization: cuantizar los scales también; ahorra ~0.4 bits/param.
- merge_and_unload(): devuelve un modelo "normal" con LoRA aplicado.

Dataset / recursos

- HuggingFace dataset de instrucción (Alpaca, Dolly, OpenAssistant).
- Modelo base: Llama 3 8B Instruct, Mistral 7B Instruct, Qwen 2 7B Instruct.
- Librerías: transformers, peft, bitsandbytes, accelerate, trl.

Ejercicios

1. LoRA básico: cargar Mistral 7B Instruct sin quantization; aplicar LoRA r=16. Inspeccionar trainable params (~0.5 %).
2. QLoRA: ahora con load_in_4bit=True. Verificar uso VRAM (~6 GB).
3. Train con TRL: SFTTrainer sobre 500 ejemplos custom. ~30 min en GPU consumer.
4. Inference con adapter: cargar base + LoRA y predecir.
5. Merge: model.merge_and_unload() → save como modelo normal.

Homework verificable

Fine-tune Mistral 7B Instruct con QLoRA para un dominio propio:

1. Dataset: 200-500 pares (instrucción, respuesta).
2. QLoRA r=16, target=all-linear, 3 épocas.
3. Inference con el adapter: 10 ejemplos test.
4. Reportar mejora cualitativa vs base.

Criterio de aceptación: respuestas más alineadas al dominio que el base; uso de VRAM < 16 GB durante training.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
OOM en Llama 7B + Adam	Sin quantization. Fix: QLoRA + paged_adamw
LoRA target_modules='all-linear' lento	Demasiados adapters. Fix: solo q/v y luego
Trainable params 0	Olvidaste get_peft_model. Fix: aplicarlo a
Resultados peor que base	LR mal, target_modules incompleto, o pocos
merge_and_unload modifica el base	Por design, en place. Fix: si necesitás ba

Preguntas frecuentes

rank r óptimo?

Empieza con 16. Tareas pequeñas: 8. Datos grandes / cambios sustanciales: 32-64.

¿LoRA o QLoRA?

QLoRA si la GPU no aguanta. Sino LoRA, ligeramente mejor calidad final.

Multi-adapter?

Sí — entrenar varios adapters (uno por tarea/idioma) y switchear en inference con model.set_adapter('name').

¿LoRA en visión / difusión?

Sí, mismo concepto. diffusers tiene LoRA support para SDXL fine-tuning.

DoRA, AdaLoRA?

Variantes: DoRA (weight-decomposed) y AdaLoRA (rank adaptativo). Mejoras marginales. LoRA "vanilla"

sigue siendo default.

Referencias

- Hu et al. (2021), LoRA, ICLR.
- Dettmers et al. (2023), QLoRA, NeurIPS.
- PEFT docs.
- TRL docs.

Siguiente clase

Clase 150 — DPO y RLHF: alineamiento de LLMs

Apéndice: notebook (primer bloque)

Implementamos LoRA desde scratch en numpy (CPU-friendly). La API real con HuggingFace + PEFT requiere GPU y se muestra en markdown.

```
import numpy as np
rng = np.random.default_rng(42)

# Toy: matriz base W (frozen) 100x100
d_in, d_out = 100, 100
W = rng.standard_normal((d_in, d_out)) * 0.1
print('W shape:', W.shape, '| params:', W.size)
```

Archivos complementarios

- notebook.ipynb