De la Investigación a la Producción: Fine-tuning eficiente y Aplicaciones RAG

Métodos Generativos, curso 2025-2026

Guillermo Iglesias, guillermo.iglesias@upm.es
Jorge Dueñas Lerín, jorge.duenas.lerin@upm.es
Edgar Talavera Muñoz, e.talavera@upm.es

5 de noviembre de 2025

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos | UPM

Contenidos

Fine-tuning y Transfer Learning

Full Fine-tuning: la aproximación tradicional

¿Qué es?

- Actualizar **todos los parámetros** del modelo pre-entrenado
- Usar datos etiquetados de la tarea objetivo
- Entrenar con learning rate bajo (no olvidar conocimiento previo)

Ventajas

- Máximo rendimiento posible
- Simple de implementar
- Flexibilidad total

Desventajas

- Alto coste computacional
- Necesita más datos
- Riesgo de overfitting
- Catastrophic forgetting

Full Fine-tuning: la aproximación tradicional

Dos problemas principales

1. Un modelo completo por tarea

BERT-base: 110M parámetros -> cada tarea requiere su propia copia del modelo.

2. Coste de entrenamiento

Durante el entrenamiento, se actualizan todos los parámetros:

- La memoria necesaria crece con el número total de parámetros.
- Más gradientes que almacenar → mayor uso de GPU/VRAM.
- Entrenamientos más lentos y costosos.

Parameter-Efficient Fine-Tuning (PEFT)

Idea central: ¿Y si solo entrenamos una **pequeña parte** del modelo?

Ventajas de PEFT

- Menos memoria durante entrenamiento
- Entrenamiento más rápido
- Evita catastrophic forgetting
- Múltiples tareas con mismo modelo base

Principales métodos:

1. **LoRA (Low-Rank Adaptation):** se añaden matrices de bajo rango a los pesos del modelo para ajustarlos sin modificarlos directamente.
2. **Adapters:** se insertan pequeñas capas nuevas entre las existentes; solo estas se entrenan.
3. **Prompt / Prefix Tuning:** se entrenan vectores de entrada adicionales que guían al modelo, sin tocar sus parámetros internos.

LoRA: Low-Rank Adaptation

Idea clave:

- Los pesos pre-entrenados W_0 permanecen **congelados**
- Añadimos matrices de **bajo rango** A y B
- Solo entrenamos A y B

Formulación matemática:

$$h = W_0 x + \Delta W x$$

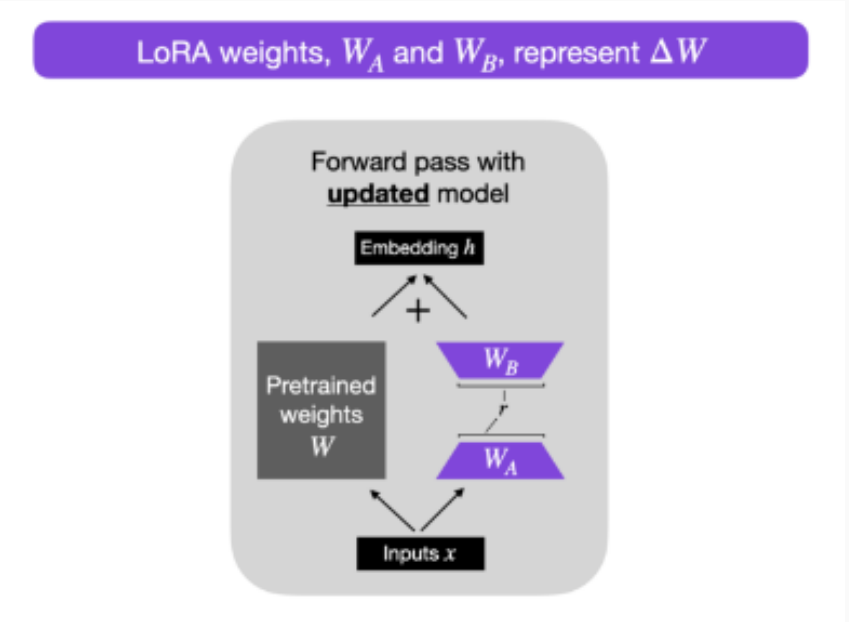
$$\Delta W = BA$$

- $x \in \mathbb{R}^k$: vector de entrada.
- $h \in \mathbb{R}^d$: salida del modelo (por ejemplo, activación de una capa lineal).
- $W_0 \in \mathbb{R}^{d \times k}$: pesos originales (congelados).
- $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$: matrices entrenables de bajo rango.
- $r \ll \min(d, k)$: controla cuántos parámetros nuevos se introducen.

LoRA: Low-Rank Adaptation

Hiperparámetros:

- r: rank (típicamente 8, 16, 32)
- α: scaling factor



Esquema de LoRA

https://learnopencv.com/fine-tuning-llms-using-peft/

Ejemplo
GPT-3 (175B parámetros) LoRA: solo 37M parámetros entrenables (0.02%)

Optimización y Eficiencia

Más técnicas para modelos eficientes

Objetivo: reducir el tamaño y el coste de los modelos sin perder demasiado rendimiento.

1. Knowledge Distillation

- Modelo grande (**teacher**) entrena a uno más pequeño (**student**).
- El estudiante aprende a imitar las salidas o distribuciones del maestro.
- Permite obtener modelos ligeros con un rendimiento similar.

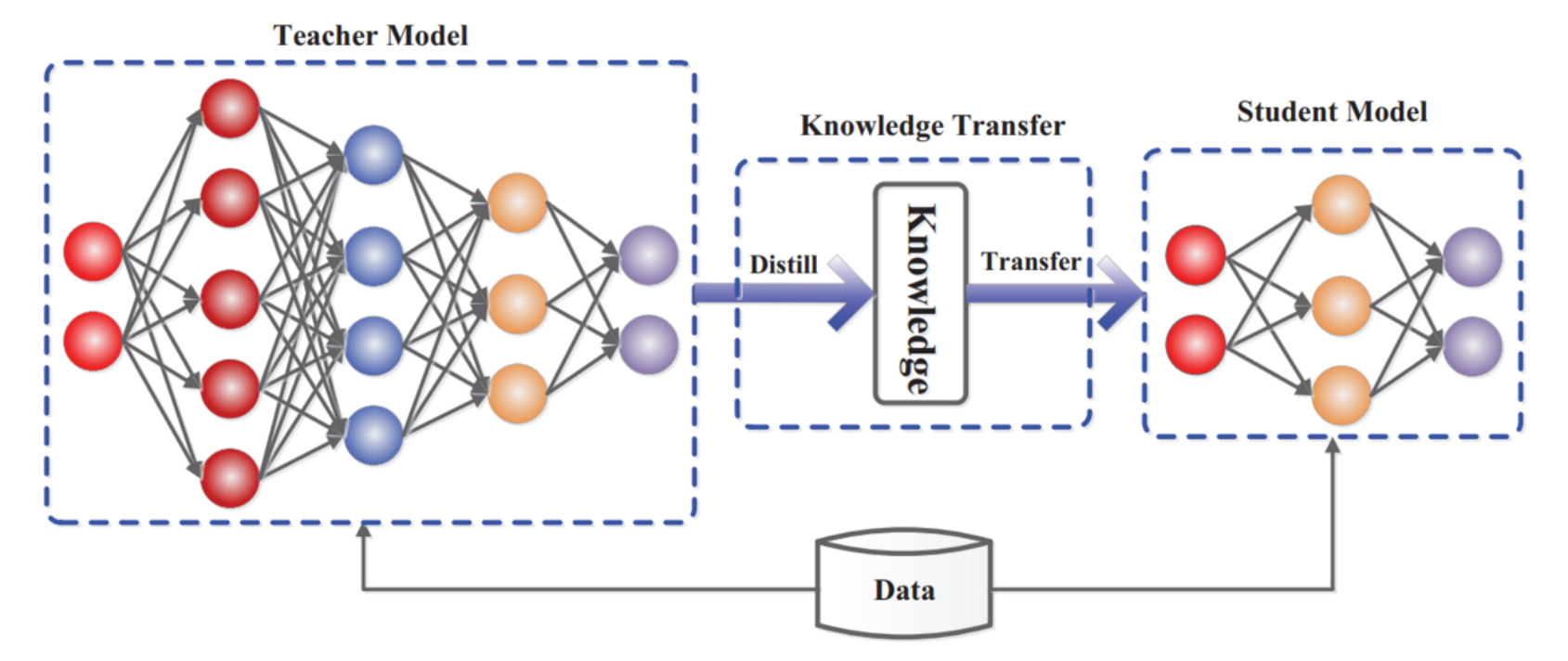
2. Quantization

- Representar los pesos con menos bits (ej. `float32` → `int8`).
- Reduce tamaño del modelo y uso de memoria.
- Acelera la inferencia con hardware optimizado.

Idea común

Ambas técnicas buscan modelos **más pequeños, rápidos y eficientes**, manteniendo un rendimiento aceptable para tareas reales.

Knowledge Distillation: Teacher-Student



Esquema de Knowledge Distillation

Objetivo: Transferir conocimiento de un modelo grande (teacher) a uno pequeño (student)

Knowledge Distillation: Teacher-Student

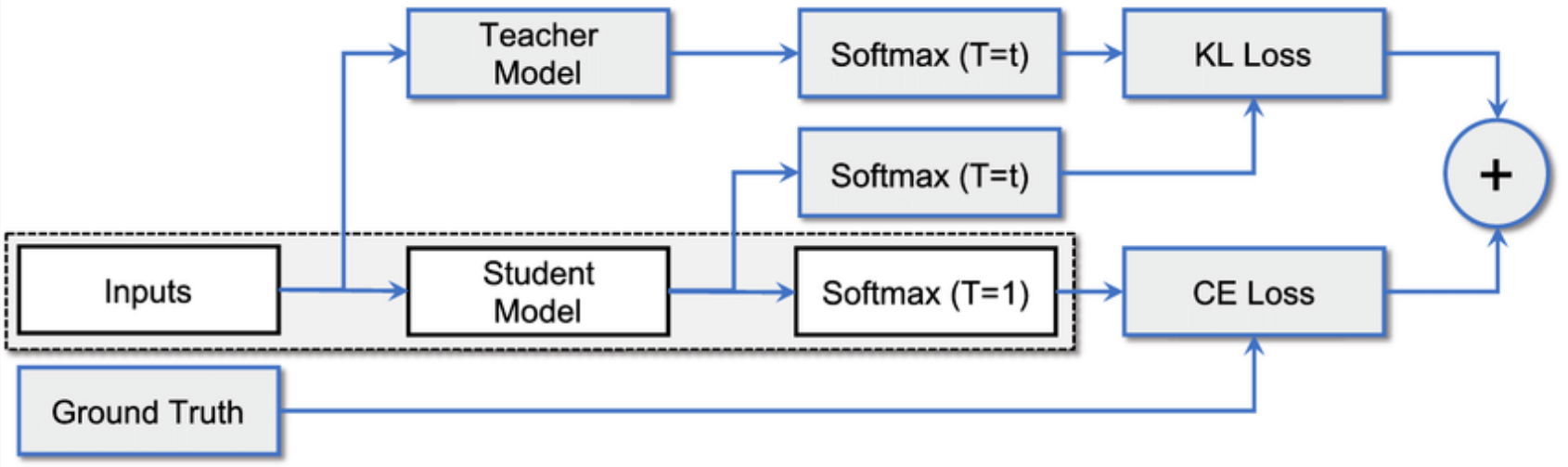
Proceso:

1. Entrenar modelo grande (teacher)
2. Usar outputs del teacher como ”soft targets”
3. Entrenar modelo pequeño (student) para imitarlo

¿Por qué funciona?

- Las probabilidades suaves contienen más información
- El student aprende relaciones entre clases
- No solo la clase correcta, sino las similitudes

Knowledge Distillation: Función de Pérdida



Knowledge Distillation

Pérdida combinada:

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{CE}(\text{hard targets}) + (1 - \alpha) \cdot \mathcal{L}_{KL}(\text{soft targets})$$

donde:

- $\mathcal{L}_{CE}$: cross-entropy con etiquetas reales
- $\mathcal{L}_{KL}$: KL-divergence con distribución del teacher
- α: balance entre ambas pérdidas

Knowledge Distillation: Temperatura y caso de éxito

Caso de éxito

DistilBERT (Hugging Face, 2019):

- Modelo reducido en un 40% de parámetros respecto a BERT-base.
- Velocidad de inferencia un 60% mayor.
- Retiene el 97% del rendimiento.

Fuente de información

Quantization: Reduciendo Precisión

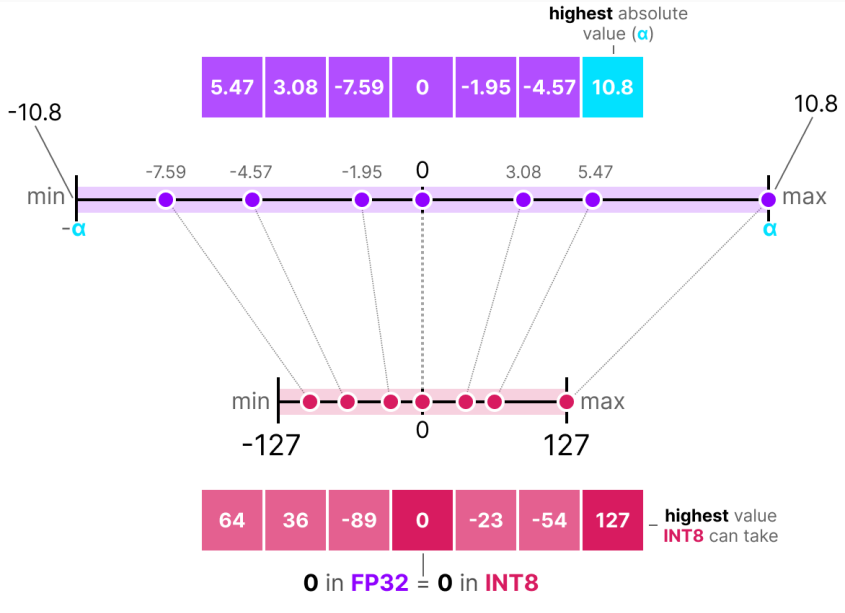
Idea: Representar pesos con menos bits

Tipos de precisión:

- **FP32**: 32 bits (full precision)
- **FP16**: 16 bits (half precision)
- **INT8**: 8 bits (integer)
- **INT4**: 4 bits (muy agresivo)

Beneficios:

- Menos memoria
- Inferencia más rápida
- Importante para deployment



Quantization uniforme

Trade-off

Menor precisión → posible pérdida de calidad

QLoRA: Quantization + LoRA

Innovación: Combinar lo mejor de ambos mundos

Cómo funciona:

1. Cargar modelo base en **4-bit** (quantizado)
2. Añadir **adaptadores LoRA** en precisión normal
3. Fine-tune solo los LoRA
4. Resultado: modelo cuantizado + adaptadores entrenados

Ventajas:

- Memoria mínima durante entrenamiento
- Permite fine-tune modelos gigantes (65B+) en una GPU
- Rendimiento competitivo con full fine-tuning

RAG

El Problema del Conocimiento Estático

Limitaciones de los LLMs:

- Conocimiento ”congelado” en el entrenamiento
- No saben eventos recientes
- No tienen acceso a info privada/corporativa
- Pueden ”alucinar” información

Ejemplo:

- GPT-4 (cutoff: sep 2021)
- Pregunta: ”¿Quién ganó el Mundial 2022?”
- Respuesta: no lo sabe

Solución

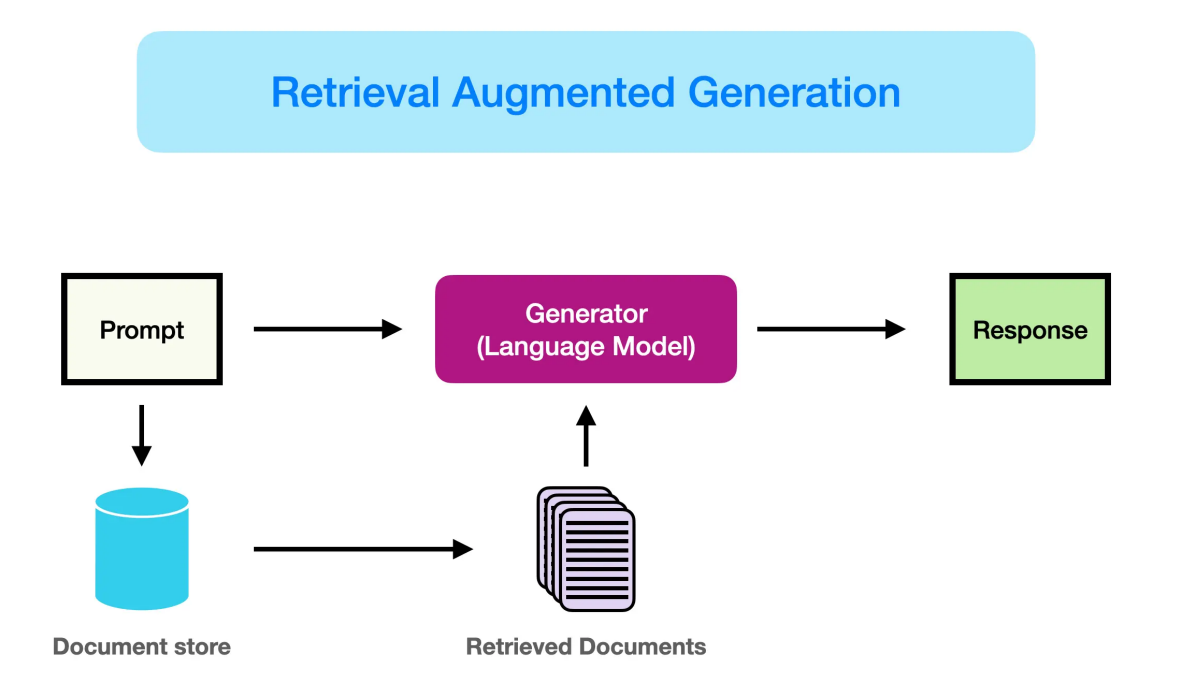
RAG: Retrieval-Augmented Generation

Darle al modelo acceso a información externa

Casos de uso

- QA sobre documentación
- Chat con PDFs
- Búsqueda semántica
- Knowledge bases corporativas

RAG: Arquitectura



Arquitectura típica de RAG

Componentes:

1. **Knowledge Base**: documentos, artículos, base de datos
2. **Retriever**: busca documentos relevantes (encoder, sentence-transformers)
3. **Generator**: genera respuesta usando contexto (LLM, T5, GPT)

RAG: ¿Cómo Funciona?

Pipeline paso a paso:

1. **Indexación** (offline):
   - Procesar documentos
   - Generar embeddings (ej: sentence-transformers)
   - Guardar en vector database (FAISS, Pinecone, Weaviate)
2. **Retrieval** (online):
   - Usuario hace pregunta
   - Convertir pregunta a embedding
   - Buscar documentos más similares (cosine similarity, etc.)
3. **Generation** (online):
   - Construir prompt: pregunta + contexto recuperado
   - LLM genera respuesta basada en contexto

Clave
El modelo no ”memoriza” información, la **busca y usa** cuando la necesita

RAG: Ejemplo de Prompt

Prompt construido por RAG

```
Use the following context to answer the question.
Context:
[Documento 1]: Los transformers fueron introducidos
en 2017 en el paper "Attention is All You Need"...
[Documento 2]: BERT es un modelo encoder-only que
usa masked language modeling...

Question: ¿Cuándo se inventaron los transformers?

Answer:
```

Ventajas:

- Respuestas basadas en fuentes verificables
- Actualización fácil (actualizar base de datos, no reentrenar)
- Reduce alucinaciones

RAG: Consideraciones Técnicas

Retriever:

- **Dense retrieval**: embeddings (BERT, sentence-transformers)
- **Information retrieval techniques**: TF-IDF, BM25
- **Hybrid**: combinar ambos

Chunking Strategy:

- ¿Cuánto texto por chunk? ¿Cuánto overlapping?

Vector Databases:

- FAISS (Facebook): in-memory, muy rápido
- Pinecone, Milvus: managed, escalables
- ChromaDB: simple, open-source

Trade-offs:

- Más documentos recuperados → más contexto pero más coste
- Calidad de retrieval crítica para calidad de respuesta

Práctica: Fine-tuning y Aplicaciones RAG

Recursos

- PEFT Library: *https://github.com/huggingface/peft*
- LoRA Paper: *https://arxiv.org/abs/2106.09685*
- QLoRA Paper: *https://arxiv.org/abs/2305.14314*
- RLHF Blog (OpenAI): *https://openai.com/research/learning-from-human-preferences*
- LangChain: *https://www.langchain.com/*
- LlamaIndex: *https://www.llamaindex.ai/*
- RAG Survey: *https://arxiv.org/abs/2312.10997*