De la Investigación a la Producción: Fine-tuning y Optimización

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

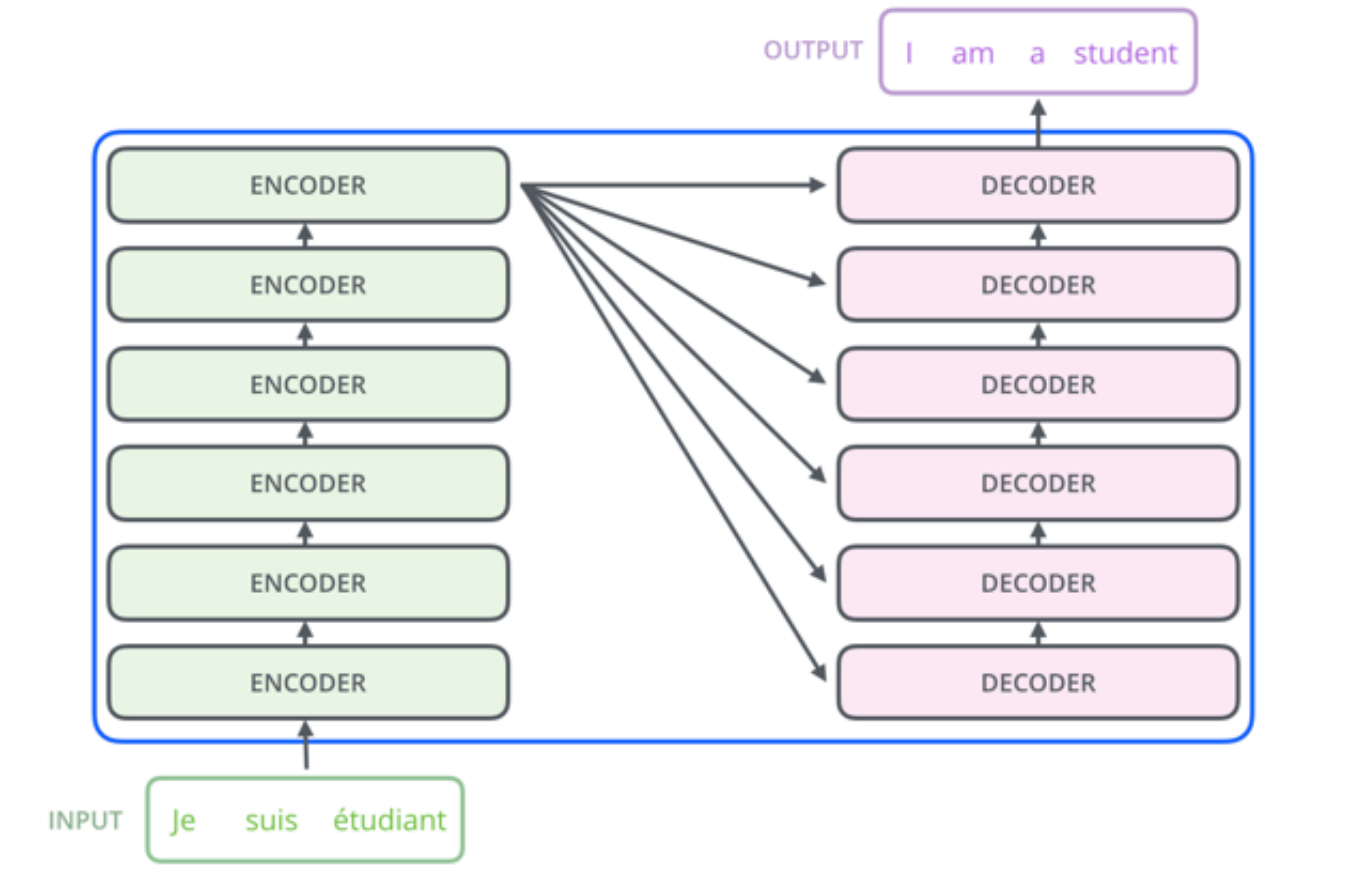
5 de noviembre de 2025

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos | UPM

Contenidos

Fine-tuning y Transfer Learning

Recordatorio: Transfer Learning



Transfer Learning = Pretraining + Fine-tuning

- **Pretraining**: aprender representaciones generales (upstream task)
- **Fine-tuning**: adaptar a tarea específica (downstream task)

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

Problema principal

BERT-base: 110M parámetros → guardar modelo completo para cada tarea

Parameter-Efficient Fine-Tuning (PEFT)

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

Ventajas de PEFT

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

Principales técnicas:

1. **LoRA** (Low-Rank Adaptation)
2. **Adapters**
3. **Prompt Tuning / Prefix Tuning**

LoRA: Low-Rank Adaptation

Idea clave:

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

Esquema de LoRA

Matemáticamente:

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

donde $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, y $r \ll d$

Hiperparámetros:

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

Ejemplo
GPT-3 (175B parámetros)

LoRA: solo 37M parámetros entrenables (0.02%)

Adapters: Módulos Insertables

Arquitectura:

- Insertar **módulos pequeños** entre capas del transformer
- Estructura típica: down-projection → activation → up-projection
- Solo entrenar los adapters

Adapter entre capas

Características:

- Bottleneck architecture (reducir dimensión)
- Residual connection
- Típicamente $< 5\%$ de parámetros del modelo

Comparación
LoRA: modifica pesos existentes

Adapters: añade nuevas capas

Ventajas:

- Modular: fácil añadir/quitar
- Un modelo base, múltiples adapters

Prompt Tuning / Prefix Tuning

Concepto:

- En lugar de entrenar pesos del modelo...
- Entrenar **embeddings continuos** que se añaden al input
- Modelo completamente congelado

Tipos:

- **Prompt Tuning**: soft prompts al inicio
- **Prefix Tuning**: vectores en cada capa
- **P-Tuning v2**: más sofisticado

Extremadamente eficiente:

- $<$ 0.1% de parámetros entrenables
- Ideal para modelos muy grandes

Ejemplo

Input original: `Classify: [text]`

Con prompt tuning: `[P1][P2][P3] Classify: [text]`

donde $[P_i]$ son vectores entrenables

Nota

Diferentes de los ”prompts” que escribimos. Estos son vectores continuos, no texto.

Comparación de Métodos PEFT

Método	Params	Rendimiento	Velocidad	Memoria
Full Fine-tuning	100%	? ? ? ? ?	? ?	? ?
LoRA	0.1-1%	? ? ? ?	? ? ? ?	? ? ? ? ?
Adapters	2-5%	? ? ? ?	? ? ?	? ? ? ?
Prompt Tuning	<0.1%	? ? ?	? ? ? ? ?	? ? ? ? ?

Recomendación práctica

- **Full fine-tuning**: si tienes recursos y datos abundantes
- **LoRA**: mejor balance general (nuestra recomendación)
- **Adapters**: si necesitas múltiples tareas
- **Prompt Tuning**: para modelos gigantes (>100B params)

Casos de Uso: Fine-tuning en la Práctica

1. Clasificación específica de dominio

- Ejemplo: análisis de sentimiento de reviews médicas
- Base: BERT o RoBERTa
- Método: LoRA con $r = 16$

2. Question Answering en documentos corporativos

- Base: T5 o BERT (QA)
- Datos: preguntas-respuestas del dominio
- Método: Full fine-tuning o LoRA

3. Resumen de artículos científicos

- Base: BART o T5
- Datos: papers + abstracts
- Método: LoRA (memoria limitada)

Consideración importante
Siempre evaluar en conjunto de test para evitar overfitting

Optimización y Eficiencia

Knowledge Distillation: Teacher-Student

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

Caso de éxito
DistilBERT

- 40% menos parámetros
- 60% más rápido
- 97% del rendimiento de BERT

Knowledge Distillation: Función de Pérdida

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

Soft targets con temperatura:

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

donde T (temperatura) suaviza la distribución

Intuición
$T > 1 \rightarrow$ distribución más suave $\rightarrow$ más información sobre similitudes entre clases

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 (4x con INT8)
- Inferencia más rápida (2-4x)
- Importante para deployment

Tipo	Memoria	Velocidad
FP32	1x	1x
FP16	0.5x	1.5-2x
INT8	0.25x	2-4x
INT4	0.125x	3-6x

Trade-off

Menor precisión → posible pérdida de calidad

En la práctica: INT8 funciona muy bien

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-tunear solo los adaptadores
4. Resultado: modelo cuantizado + adaptadores entrenados

Ventajas:

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

Ejemplo impactante
LLaMA-65B

Sin QLoRA:

- 130 GB VRAM
- Imposible en 1 GPU

Con QLoRA:

- 48 GB VRAM
- Cabe en A100 (80GB)

Paper clave
"QLoRA: Efficient Finetuning of Quantized LLMs" (2023)

Otras Técnicas de Optimización

1. Pruning (Poda)

- Eliminar conexiones/neuronas poco importantes
- Structured vs. Unstructured pruning
- Puede reducir hasta 90% de parámetros con pérdida mínima

2. Weight Sharing

- Compartir pesos entre capas (ej: ALBERT)
- Reduce parámetros sin reducir profundidad

3. Mixed Precision Training

- FP16 para forward/backward pass
- FP32 para actualización de pesos
- 2x más rápido, misma precisión

4. Gradient Checkpointing

- Trade-off: memoria $\leftrightarrow$ velocidad

RLHF y Modelos Instructivos

El Problema: Modelos Pre-entrenados vs. Útiles

Modelo pre-entrenado:

- Predice siguiente token
- No optimizado para seguir instrucciones
- Puede generar contenido inapropiado
- No alineado con valores humanos

Lo que queremos:

- Seguir instrucciones
- Ser útil
- Ser honesto
- Ser seguro (harmless)

Ejemplo

Prompt: "¿Cómo hago una tortilla?"

GPT-3 base: "¿Cómo hago una tortilla? ¿Cómo hago un pastel? ¿Cómo..."

ChatGPT (con RLHF): "Para hacer una tortilla necesitas: huevos, aceite, sal..."

Solución

RLHF: Reinforcement Learning from Human Feedback

RLHF: Las Cuatro Fases

1. **Pretraining** (modelo base)
   - Entrenamiento estándar con next-token prediction
   - Ejemplo: GPT-3, LLaMA base
2. **Supervised Fine-Tuning (SFT)**
   - Humanos escriben ejemplos de alta calidad
   - Fine-tunear el modelo con estos ejemplos
   - Típicamente 10K-100K ejemplos
3. **Reward Modeling (RM)**
   - Modelo genera múltiples respuestas
   - Humanos las rankean (mejor → peor)
   - Entrenar modelo de recompensa que predice rankings
4. **Reinforcement Learning (PPO)**
   - Optimizar con Proximal Policy Optimization
   - Maximizar recompensa según modelo RM
   - KL-penalty para no alejarse demasiado del SFT

RLHF: Diagrama del Proceso

Pipeline completo de RLHF (Fuente: OpenAI)

Modelos que usan RLHF
ChatGPT, GPT-4, Claude, Llama 2 Chat, Mistral Instruct

RLHF: Por qué es Importante

Impacto en la práctica:

- ChatGPT → explosión de uso de LLMs
- Modelos más seguros y útiles
- Mejor seguimiento de instrucciones
- Reducción de outputs tóxicos

Comparación (GPT-3 vs. GPT-3.5-turbo):

- GPT-3: modelo base
- GPT-3.5-turbo: + SFT + RLHF
- Diferencia dramática en utilidad

Estadística
OpenAI reporta: RLHF mejora alineación en $> 2x$ comparado con solo SFT

RLHF: Limitaciones y Desafíos

1. Coste de Anotación

- Necesita muchas comparaciones humanas
- Anotadores especializados (costoso)
- Difícil de escalar

2. Reward Hacking

- El modelo aprende a ”engañar” al reward model
- Respuestas que parecen buenas pero son incorrectas
- Sycophancy: decirle al usuario lo que quiere oír

3. Sesgos y Valores

- ¿Qué valores codificamos?
- Diferentes culturas, diferentes preferencias
- Riesgo de amplificar sesgos de anotadores

4. Alternativas en Desarrollo

RAG y Agentes

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)

RAG: Consideraciones Técnicas

Retriever:

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

Chunking Strategy:

- ¿Cuánto texto por chunk? (típicamente 200-500 tokens)
- ¿Overlapping? (mejor capturar contexto)

Vector Databases:

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

Trade-offs:

- Más documentos recuperados → más contexto pero más coste

Agentes LLM: Introducción

¿Qué es un agente?

- LLM como ”cerebro” que decide acciones
- Capacidad de usar **herramientas externas**
- Loop de razonamiento → acción → observación

Herramientas típicas:

- Calculadora
- Búsqueda web (Google, Wikipedia)
- APIs (weather, stock prices, etc.)
- Ejecutar código (Python REPL)
- Acceso a bases de datos

Arquitecturas:

- **ReAct**: Reasoning + Acting

Ejemplo

Pregunta: ”¿Cuál es la raíz cuadrada de 1764?”

Agente:

1. Reconoce que necesita cálculo
2. Usa herramienta ”calculadora”
3. Obtiene resultado: 42
4. Responde al usuario

Agentes: ReAct Pattern

ReAct: Reasoning + Acting (interleaved)

Ejemplo de trace

Question: What is the elevation of the highest point in California?

Thought 1: I need to search for the highest point in California

Action 1: Search[highest point in California]

Observation 1: Mount Whitney is the highest point...

Thought 2: Now I need to find its elevation

Action 2: Search[Mount Whitney elevation]

Observation 2: 4,421 meters

Thought 3: I have the answer

Action 3: Finish[4,421 meters]

Frameworks para Agentes

1. LangChain

- Framework más popular
- Abstrae agents, chains, tools
- Gran ecosistema de integraciones

2. LlamaIndex

- Especializado en RAG y data connectors
- Indexación y query de documentos
- Múltiples estrategias de retrieval

3. Otros

- AutoGPT, BabyAGI: agentes autónomos
- Semantic Kernel (Microsoft)
- Haystack

Nota
Campo muy activo, nuevas herramientas cada mes

Agentes: Limitaciones

Desafíos actuales:

- **Confiabilidad**: pueden fallar o entrar en loops
- **Coste**: muchas llamadas a LLM ($)
- **Latencia**: múltiples pasos → respuesta lenta
- **Evaluación**: difícil medir rendimiento
- **Seguridad**: ejecución de código, acceso a APIs

Buenas prácticas:

- Limitar número de pasos
- Validar acciones antes de ejecutar
- Logging detallado para debugging
- Empezar simple, añadir complejidad gradualmente

Estado actual (2024-2025)
Prometedores para casos de uso específicos, pero aún no ”production-ready” para casos generales

Práctica: Fine-tuning y Optimización

En este notebook exploraremos:

- Full fine-tuning (baseline)
- PEFT con LoRA
- Quantization (INT8)
- QLoRA (bonus)
- Mini RAG (conceptual)

Compararemos: parámetros entrenables, tiempo, memoria, y accuracy.

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*