Fine-Tuning de LLMs

Inteligencia Artificial e Ingeniería del Conocimiento

Constantino Antonio García Martínez

Universidad San Pablo Ceu

Aproximaciones a Fine-Tuning (FT)

Fine-Tuning de LLMs

Disponemos de un **LLM pre-entrenado** en un corpus masivo. ¿Cómo **adaptarlo** para resolver nuestros problemas específicos?

Classification FT *Especialización en una tarea concreta*

- Máxima precisión en una tarea (clasificación, NER, QA...)
- Usa features del LLM para un clasificador downstream

Instruction FT *Seguir instrucciones en lenguaje natural, posiblemente con varias tareas*

- Ejecutar múltiples tareas mediante instrucciones
- Mantiene capacidad generativa del modelo

RLHF/RLAIF *Alinear con preferencias humanas*

- Respuestas útiles, honestas, seguras
- Aprende de preferencias (ranking, no labels)

Classification Fine-Tuning

Classification Fine-Tuning: Motivación

Problema: Tenemos un modelo pre-entrenado poderoso, pero necesitamos especializarlo en nuestra tarea.

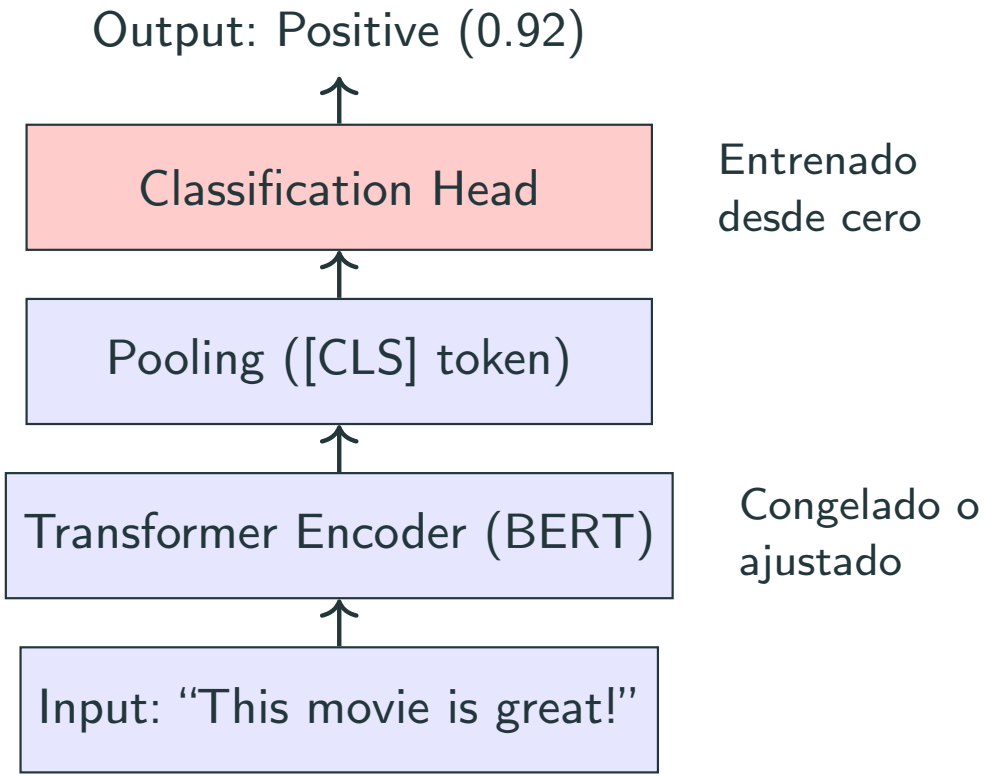
Solución: Fine-tuning supervisado (SFT)

- Dataset etiquetado: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$
- Continuar entrenamiento con ejemplos de la tarea
- Optimizar loss específico de la tarea

Example: Clasificación de sentimientos

- Base: BERT pre-entrenado en textos generales
- Fine-tuning: reviews con etiquetas positivo/negativo
- Resultado: clasificador especializado en sentimientos

Arquitectura para Classification FT



Loss: Cross-entropy

$$\mathcal{L} = -\sum_{i=1}^{N} \log p(y_i|x_i;\theta)$$

Dos Estrategias de Fine-Tuning

Feature Extraction

- Congelar encoder pre-entrenado
- Solo entrenar classification head
- Rápido, pocos parámetros
- Útil con pocos datos

$\theta_{\text{encoder}} \rightarrow$ frozen

$\theta_{\text{head}} \rightarrow$ trainable

Full Fine-Tuning

- Actualizar todo el modelo
- Mejor rendimiento
- Más costoso
- Requiere más datos

$\theta_{\text{encoder}} \rightarrow$ trainable

$\theta_{\text{head}} \rightarrow$ trainable

Demo: Classification Fine-Tuning

Example: Classification Fine-Tuning con BERT: `demo_classification_ft.py`
Vamos a hacer fine-tuning de BERT para clasificación de sentimientos usando el dataset IMDb.

Objetivo: Clasificar reviews de películas como positivas o negativas.

Transformers Library: API Básica

Hugging Face Transformers: librería estándar para LLMs

Clases principales:

- `AutoTokenizer`: tokenización automática
- `AutoModel`: modelo base (solo encoder)
- `AutoModelFor*`: modelo + task-specific head

Catastrophic Forgetting

Problema: Durante fine-tuning, el modelo puede "olvidar" conocimiento del pre-entrenamiento

Ejemplo:

- Pre-training: conocimiento general del lenguaje
- Fine-tuning en dominio médico
- Resultado: excelente en medicina, pobre en tareas generales

Mitigación:

- Learning rate pequeño
- Pocas epochs
- Regularización
- PEFT (siguiente sección)

Trade-off: Especialización vs. generalización

Parameter-Efficient Fine-Tuning (PEFT)

¿Por Qué Full Fine-Tuning es Caro?

Exercise: Tamaño de GPT-3

Consideremos GPT-3 (175B parámetros), cada uno de 2 Bytes. ¿Cuál es el tamaño en memoria (aproximado) para hacer fine-tuning del modelo completo?

¿Por Qué Full Fine-Tuning es Caro?

Exercise: Tamaño de GPT-3

Consideremos GPT-3 (175B parámetros), cada uno de 2 Bytes. ¿Cuál es el tamaño en memoria (aproximado) para hacer fine-tuning del modelo completo?

$\Rightarrow$ **Necesitamos métodos más eficientes**: PEFT

PEFT: Motivación

Idea central: No necesitamos actualizar *todos* los parámetros

Observación empírica: Los modelos pre-entrenados tienen representaciones "redundantes" o de bajo rango intrínseco[1]

Estrategia PEFT:

1. Congelar el modelo base (θ_0)
2. Introducir un conjunto pequeño de parámetros entrenables ($\Delta\theta$)
3. Solo optimizar $\Delta\theta$

$$\theta = \theta_0 + \Delta\theta$$

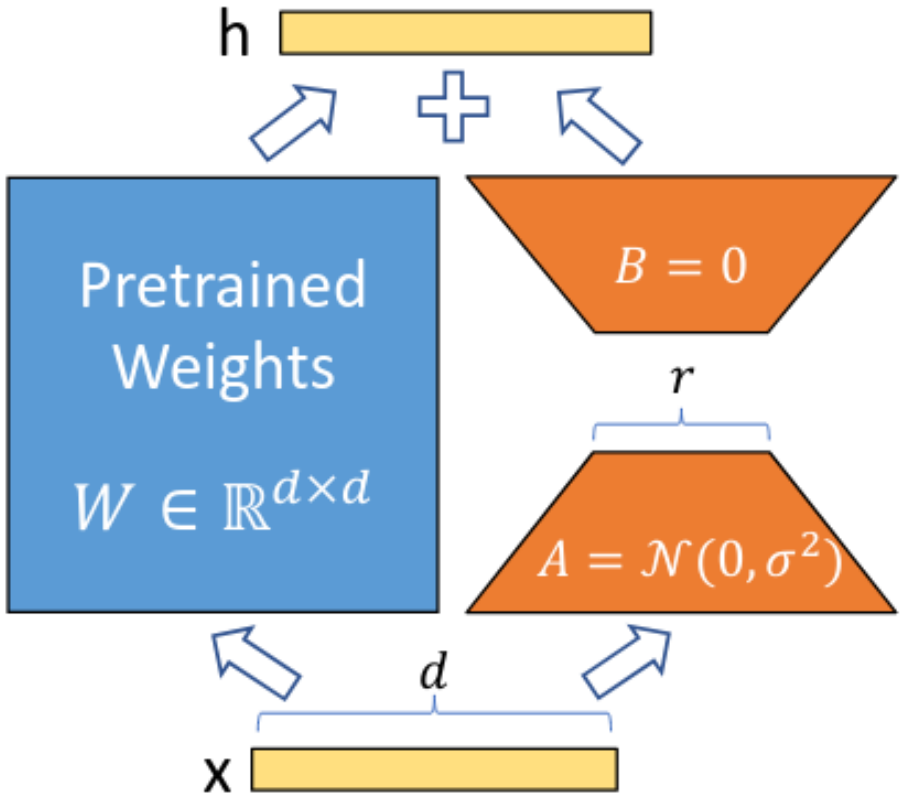
donde $|\Delta\theta| \ll |\theta_0|$ (típicamente $< 1\,\%$)

[1] "Rango" = número de dimensiones independientes. Rango bajo $\Rightarrow$ estructura redundante que se puede comprimir.

Parameter-Efficient Fine-Tuning (PEFT)

LoRA

LoRA: Low-Rank Adaptation



[2]

$$h = W_0 x + \frac{\alpha}{r} BAx$$

[2] Fuente: Hu et al. https://arxiv.org/pdf/2106.09685

LoRA: Ejemplo Numérico

Exercise: Ahorro con LoRA
Capa attention de GPT-3: $d = 12288$, $k = 12288$

Si se emplea LoRA con $r = 8$, ¿cuántos parámetros entrenables tenemos? ¿Qué porcentaje representa respecto al original?

LoRA: Ventajas

1. **Eficiencia de parámetros**
   - $< 1\,\%$ de parámetros entrenables
   - Múltiples adaptadores comparten modelo base
2. **Eficiencia de memoria**
   - No necesitamos gradientes para W_0
   - Menor uso de GPU memory
3. **Sin latencia en inferencia**
   - Podemos pre-computar: $W = W_0 + BA$
   - Inferencia idéntica al modelo original
4. **Modularidad**
   - Cambiar de tarea = cambiar adaptadores
   - Un modelo base, múltiples especializaciones

QLoRA (Quantized LoRA) y otros PEFT

Idea: Combinar LoRA con cuantización del modelo base

- Modelo base: cuantizado a 4 bits (en lugar de 16/32 bits)
- LoRA adapters: precisión completa (16 bits)

Exercise: GPT-3 + QLoRA
¿Cuánta memoria necesitamos para fine-tuning de GPT-3 con QLoRA (4 bits)?

Asume que los adaptadores QLoRA solo se aplican a las 4 capas de atención.

Otros PEFT: Adapter layers, Prefix tuning, Prompt tuning.

Demo: LoRA Fine-Tuning

Example: `demo_lora_classification.py`
Repetimos clasificación de sentimientos con LoRA en lugar de full fine-tuning.

Instruction Fine-Tuning

Classification FT vs Instruction FT

Classification FT:

- Input: texto
- Output: etiqueta (clase discreta)
- Modelo: encoder + head
- Tarea: una sola (clasificación)

Instruction FT:

- Input: instrucción en lenguaje natural
- Output: texto generado
- Modelo: decoder (generativo)
- Tarea: múltiples (seguir instrucciones)

Ejemplo: Datos de Classification FT

Dataset: Pares (texto, etiqueta)

```
Texto: "This movie was great!"
Label: Positive (1)

Texto: "Terrible film, waste of time."
Label: Negative (0)
```

Objetivo: Clasificar correctamente

Ejemplo: Datos de Instruction FT

Dataset: Pares (instrucción, respuesta)

```
Instruction: "Traduce al francés: I love you"
Response: "Je t'aime"

Instruction: "Resume este texto: [...]"
Response: "El texto explica que..."

Instruction: "¿Cuál es la capital de Francia?"
Response: "La capital de Francia es París."
```

Objetivo: Generar respuesta apropiada a cualquier instrucción

Formato de Datos

Estructura típica:

```
\texttt{<instruction>}
  [Texto de la instrucción]
\texttt{</instruction>}
\texttt{<response>}
  [Texto de la respuesta]
\texttt{</response>}
```

Variantes:

- Alpaca: `Instruction`, `Input`, `Output`
- FLAN: Template-based
- Dolly: `Instruction`, `Context`, `Response`

El formato exacto varía, pero la idea es la misma.

Loss Function: Causal Language Modeling

Loss: Predecir siguiente token (como en pre-training)

$$\mathcal{L} = -\sum_{t=1}^{T} \log p(y_t | y_{<t}, x)$$

donde x = instrucción, y = respuesta

Importante:

- Solo se computa loss sobre tokens de la respuesta
- Tokens de la instrucción se usan como contexto (sin loss):

  Truco: Usar mask para ignorar tokens de instrucción

  Secuencia completa:

  `[Instruction] Traduce: I love you [Response] Je t'aime`

  Loss mask:

  `[0 0 0 0 0 0] [1 1 1 1]`
- El modelo NO sabe si la respuesta es "correcta"
- Solo aprende a imitar el patrón de los datos

Generalización Multi-Tarea

Fenómeno emergente: El modelo generaliza a tareas no vistas

Entrenamiento:
1000 ejemplos de traducción
1000 ejemplos de resumen
1000 ejemplos de QA

Test:
“Escribe un poema sobre el mar"$\rightarrow$ Funciona (nunca vio poemas)

Por qué funciona:

- El modelo aprende el patrón “instruction $\rightarrow$ response"
- No solo aprende tareas específicas
- Aprende a “seguir instrucciones” en general

Instruction FT: Ventajas y Desventajas

Ventajas:

- Un solo modelo para múltiples tareas
- Generaliza a tareas no vistas
- Interfaz natural (lenguaje)
- No necesita arquitectura específica por tarea

Desventajas:

- Requiere modelo generativo (más grande)
- Necesita datos de alta calidad (pares bien formados)
- Menor rendimiento que Classification FT en tarea específica
- Más costoso en inferencia (genera token a token)

Datasets Comunes

Alpaca (52K ejemplos):

- Generado por GPT-3.5
- Formato: Instruction, Input, Output
- Open source

Dolly (15K ejemplos):

- Creado por humanos (Databricks)
- Diversas categorías

FLAN (1.8M ejemplos):

- Google, múltiples datasets
- Templates para crear instrucciones

OpenOrca, WizardLM, ...

Demo: Instruction Fine-Tuning

Example: `demo_instruction_ft.py`
Fine-tuning de GPT-2 (pequeño) con subset de Alpaca + LoRA

RLHF/RLAIF y Alignment

De Instruction FT a RLHF

Limitación de Instruction FT:

- Aprende de ejemplos: instrucción $\rightarrow$ respuesta
- No captura preferencias sutiles
- Difícil enseñar conceptos como "útil", "seguro", "honesto"

RLHF (Reinforcement Learning from Human Feedback):

- Aprende de comparaciones: “A es mejor que B"
- Usa reward model + algoritmo RL (típicamente PPO)
- Alinea modelo con valores humanos

Existen alternativas (DPO, RLAIF) que también usan feedback pero con algoritmos diferentes.

Usado (en distintas variantes) en ChatGPT, Claude, Gemini, LLaMA-2, etc.

RLHF: Los 3 Pasos

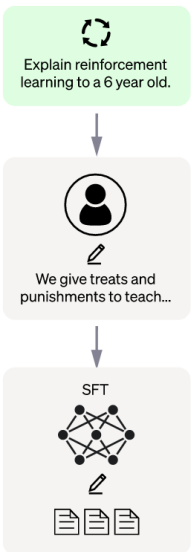
Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.



Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

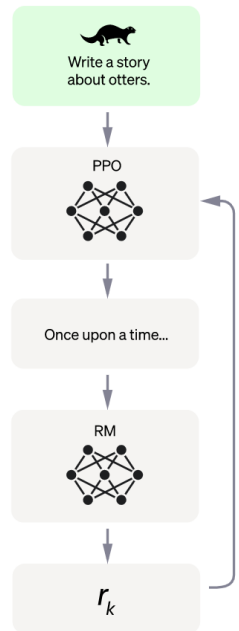
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



RLAIF: AI Feedback en vez de Humano

RLAIF (RL from AI Feedback): Usar IA para generar comparaciones

Dónde encaja: Reemplaza Step 2 (parcial o totalmente)

- En vez de humanos rankeando respuestas
- Un modelo de IA genera los rankings
- Basado en principios o constitución

Constitutional AI (Claude/Anthropic):

- Modelo se auto-critica según principios
- "Constitución": reglas de comportamiento
- Reduce dependencia de anotadores humanos

Ventaja: Escalable, más barato que RLHF puro

En la práctica: La mayoría de modelos comerciales combinan RLHF + RLAIF (híbrido) en el Step 2.

Alignment: El Objetivo Final

Alignment = Alinear modelo con valores humanos

Los 3H (Anthropic):

1. **Helpful**: Respuestas útiles que ayudan al usuario
2. **Honest**: Respuestas veraces, admite incertidumbre
3. **Harmless**: Respuestas seguras, no dañinas

Técnicas para alignment:

- RLHF/RLAIF: Método principal
- Constitutional AI: Principios explícitos
- Red teaming: Búsqueda adversarial de fallos (**Jailbreaking**)
- Filtering: Eliminar datos problemáticos

Alignment es un problema abierto y en evolución activa.