dgroove Tecnologia emozionale.

ICT Training Center

Il tuo partner per la Formazione e la Trasformazione digitale della tua azienda

Note

Spring AI

Generative Artificial Intelligence con Java

Simone Scannapieco

Corso avanzato per Venis S.p.A, Venezia, Italia

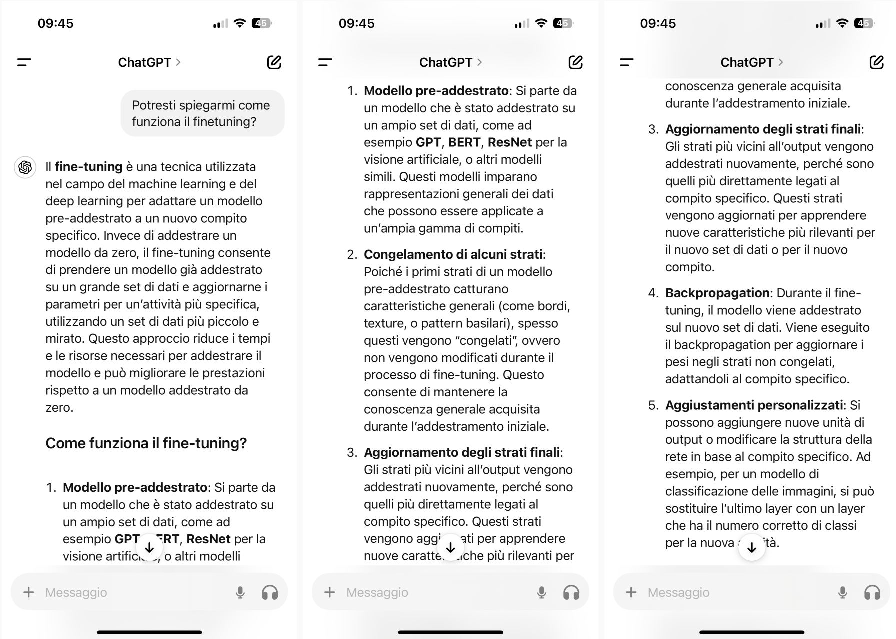
Novembre 2025

Note

- Praticamente impossibile pre-addestrare un LLM (a meno che tu non sia Google, OpenAI, Mistral, Anthropic e pochissimi altri)
- Utilizzare un LLM per scopi personali è un conto, adottarli in un contesto *business* significa scontrarsi con problematiche di natura etica, legale ed economica
- Gli LLM saranno sempre più bravi a modellare la comprensione del linguaggio. . .

. . . ma come usarli per *task* specifici o con conoscenza che a loro manca?!

Note



09:45
ChatGPT
Potresti spiegarmi come funziona il finetuning?
Il **fine-tuning** è una tecnica utilizzata nel campo del machine learning e del deep learning per adattare un modello pre-addestrato a un nuovo compito specifico. Invece di addestrare un modello da zero, il fine-tuning consente di prendere un modello già addestrato su un grande set di dati e aggiornarne i parametri per un'attività più specifica, utilizzando un set di dati più piccolo e mirato. Questo approccio riduce i tempi e le risorse necessari per addestrare il modello e può migliorare le prestazioni rispetto a un modello addestrato da zero.
Come funziona il fine-tuning?
1. **Modello pre-addestrato**: Si parte da un modello che è stato addestrato su un ampio set di dati, come ad esempio **GPT**, **BERT**, **ResNet** per la visione artificiale, o altri modelli simili. Questi modelli imparano rappresentazioni generali dei dati che possono essere applicate a un'ampia gamma di compiti.
2. **Congelamento di alcuni strati**: Poiché i primi strati di un modello pre-addestrato catturano caratteristiche generali (come bordi, texture, o pattern basilari), spesso questi vengono "congelati", ovvero non vengono modificati durante il processo di fine-tuning. Questo consente di mantenere la conoscenza generale acquisita durante l'addestramento iniziale.
3. **Aggiornamento degli strati finali**: Gli strati più vicini all'output vengono addestrati nuovamente, perché sono quelli più direttamente legati al compito specifico. Questi strati vengono aggiornati per apprendere nuove caratteristiche più rilevanti per il nuovo set di dati o per il nuovo compito.
4. **Backpropagation**: Durante il fine-tuning, il modello viene addestrato sul nuovo set di dati. Viene eseguito il backpropagation per aggiornare i pesi negli strati non congelati, adattandoli al compito specifico.
5. **Aggiustamenti personalizzati**: Si possono aggiungere nuove unità di output o modificare la struttura della rete in base al compito specifico. Ad esempio, per un modello di classificazione delle immagini, si può sostituire l'ultimo layer con un layer che ha il numero corretto di classi per la nuova
Messaggio

Note

- *Full learning*
  - Creare una architettura neurale da zero. . .
  - . . . oppure scegliere una architettura in letteratura (per i meno sadici)
  - Addestramento da zero (a partire da pesi e *bias random*)
- *Transfer learning*
  - Sfruttare una rete neurale giá addestrata su un altro insieme di dati di addestramento
  - Modificare solo alcuni strati (solitamente gli ultimi) per addestrare la rete per i propri scopi

Computer Vision	***Full learning***	***Transfer learning***
Numero dati addestramento	10^3–10^6	10^2
Computazione	Intensiva (GPU)	Media (CPU–GPU)
Tempo di addestramento	Giorni–settimane	Ore–giorni
Accuratezza del modello	Alta	Variabile

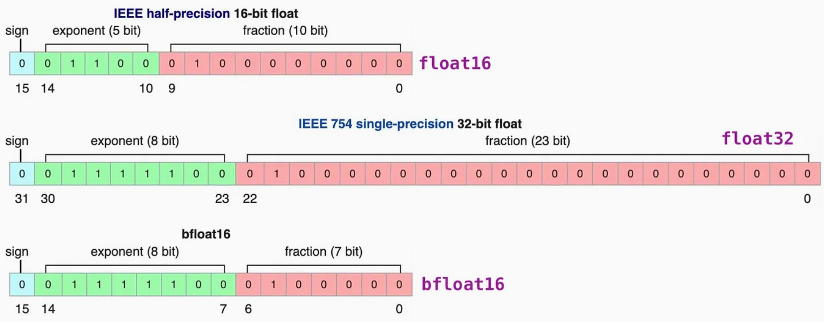
Note

- ChatGPT parla del *fine-tuning* per LLM come sinonimo di *transfer learning*
- Si parla di *full fine-tuning* quando nessuno strato viene congelato
- Diverse tecniche per quanto riguarda il “*transfer tuning*”
  - *Multi-Level Fine-Tuning* (MLFT)
  - *Parameter Efficient Fine-Tuning* (PEFT)
  - *Low-Rank Adaptation* (LoRA)
  - Adapter Training
  - . . .

Note

Quantization
Ottimizzazione per *full fine-tuning*

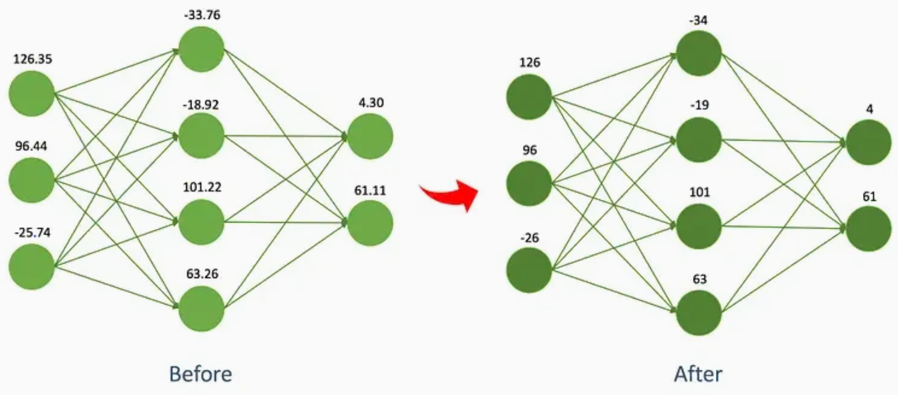
- Possibile eseguire *full fine-tuning* su LLM pre-addestrate di dimensioni “contenute”. . .
- . . . facendo attenzione a non esagerare. . .
- . . . e possibilmente, ottimizzando il processo decidendo la precisione dei parametri del modello (ovvero, *quantizzandolo*)



Quantizzazione
©Cerebras

- Quantizzazione a `float16` e `bfloat16` usati maggiormente per addestramento
- `bfloat16` generalmente preferito a `float16`
- Addestramento a `float32` riservato alle *big companies*
- Quantizzazioni inferiori disponibili (`int8`, `int4`), ma consigliate solo per inferenza

Note



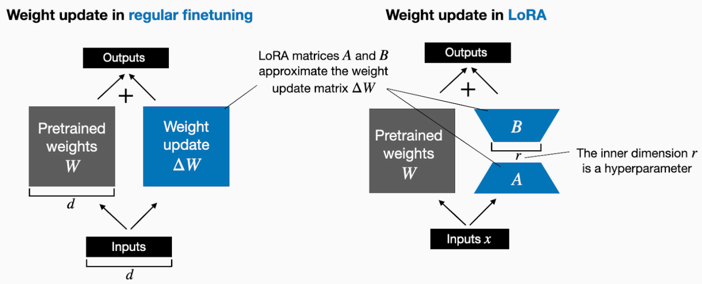
Quantizzazione
©jeremy jouvance@GoPenAI

- A favore del peso del modello in memoria e ai tempi di addestramento. . .
- . . . a scapito dell'accuratezza in fase di inferenza
- *Tradeoff* fra risorse e prestazioni dettato anche dalla strategia adottata
  - Quantizzazione terminato il processo di addestramento (Post Training Quantization – PTQ)
  - Quantizzazione durante l'addestramento (*Quantization-Aware Training (QAT)*)

Note

Edward Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weigh Chen. *LoRA. Low-Rank Adaptation of Large Language Models*. **ArXiV,** 2021.

- Variazione di PEFT
- Si riaddestrano solo un sottoinsieme ΔW di parametri esplicitamente selezionati (una sorta di *transfer learning*). . .
- . . . mantenendo fissi tutti gli altri W
- Fissato un parametro r (rango), LoRA approssima ΔW come prodotto scalare di due matrici più piccole



LoRA
©Sebastian Raschka@Ahead of AI

Note

Prodotto scalare

Il *prodotto scalare* di matrice $(n \times r)$ $A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1r} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nr} \end{bmatrix}$ e matrice $(r \times m)$

$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ \vdots & \vdots & \vdots & \vdots \\ b_{r1} & b_{r2} & \dots & b_{rm} \end{bmatrix}$ é la matrice $(n \times m)$

$$A \cdot B = \begin{bmatrix} a_{11} * b_{11} + \dots + a_{1r} * b_{r1} & \dots & a_{11} * b_{1m} + \dots + a_{1r} * b_{rm} \\ \vdots & \vdots & \vdots \\ a_{n1} * b_{11} + \dots + a_{nr} * b_{r1} & \dots & a_{n1} * b_{1m} + \dots + a_{nr} * b_{rm} \end{bmatrix}$$

- In pratica, fissato r, LoRA computa A e B tale per cui $\Delta W = A \cdot B$
- Ma perché é così potente?!

Note

Esempio LoRA

$$\Delta W = \begin{bmatrix} 5 & 1 & -1 & 3 & 4 \\ 15 & 3 & -3 & 9 & 12 \\ 35 & 7 & -7 & 21 & 28 \\ -20 & -4 & 4 & -12 & -16 \\ 10 & 2 & -2 & 6 & 8 \end{bmatrix} \xRightarrow{\text{LoRA}(r\,=\,1)} A = \begin{bmatrix} 1 \\ 3 \\ 7 \\ -4 \\ 2 \end{bmatrix}, B = \begin{bmatrix} 5 & 1 & -1 & 3 & 4 \end{bmatrix}$$

- Numero parametri da salvare?
  - ΔW: 25
  - A e B (totale): 10
  - Risparmio spazio: 40%
- La *backpropagation* opera direttamente sulle rappresentazioni di A e B!
- Metodo di approssimazione, quindi a scapito della *accuracy* del modello

Note

QLoRA
Cosa sarà mai. . .?

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, Luke Zettlemoyer.
QLoRA: Efficient Finetuning of Quantised LLMs.
ArXiV, 2023.

- Decomposizione LoRA + *Quantization*. . .
- . . . tutto qui.

Note