DDA5001 Machine Learning

Large Language Models (Part II)

Xiao Li

School of Data Science
The Chinese University of Hong Kong, Shenzhen

Pre-training

Post-training

Reasoning Models

Pre-training Overview

Overview:

- Pre-training is to train an LLM from scratch by optimizing the pre-training objective on a large corpus (i.e., training dataset).
- This looks like a standard training procedure. However, it contains so many non-trivial engineering steps, huge computing resources, and man power resources due to truly large-scale feature.

We quickly go through: data, model architecture, scaling law, training setting for pre-training.

We will take Llama 3 developed by Meta as example.

Pre-training Dataset

Llama 3's pre-training uses 15T carefully crafted data from the internet.

The processing of data includes:

- Safety filtering.
- Text extraction and cleaning: extract high quality text from HTML.
- De-duplication.
- Quality filtering.
- Code and math reasoning data.
- Multilingual data: 176 languages.

Data mix:

- 50% general knowledge tokens.
- 25% mathematical and reasoning tokens.
- 17% code tokens.
- 8% multilingual tokens.

Model Architecture

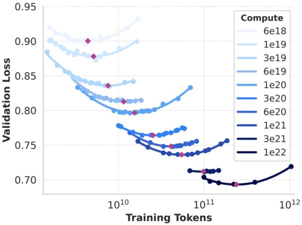
- Llama 3 uses a standard, dense (decoder) Transformer architecture.
- Use group query attention with 8 key-value heads to accelerate inference speed.
- Vocabulary is of size 128K, i.e., V in the final layer logistic regression classifier has dimension 128K.
- Llama 3 has 8B, 70B, 405B models (trainable parameters of the transformer).

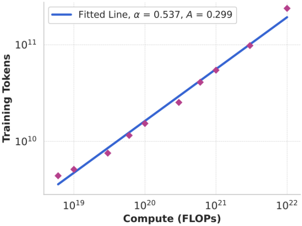
Overview of the key hyperparameters of Llama 3:

	8B	**70B**	**405B**
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	6,144	12,288	20,480
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ($\theta = 500,000$)		

Scaling Law

- Scaling law is to predict the training loss of (**??**) based on training tokens and model size trade-off, given a fixed compute budget (measured by FLOPS).





- The left figure uses quadratic function to do fitting.
- The right figure takes the red compute-optimal points in left to fit a power law relationship:

$$N^*(C) = A \cdot C^{\alpha}.$$

Here, C is compute FLOPS (x-axis) and N^* is the optimal training tokens (y-axis). This also indicates the model size.

Training Recipe

Llama 3-405B is pre-trained using

- AdamW optimizer.
- 8×10^{-5} initial lr, and use cosine decay to 8×10^{-7}.
- It uses an increasing batch size to stabilize training process. The batch size for computing the stochastic gradient is starting from 4M, to 8M, and finally to 16M.
- It also increases the sequence length n, starting from 4096 to final 128K sequence length / context window, in order to learn long context understanding.

Pre-training

Post-training

Reasoning Models

SFT: Supervised Finetuning

- A pre-trained LLM can be regarded as having general knowledge.
- Sometimes, we need to adapt an LLM to a specific domain.
- Additionally, pre-trained model is not capable of understanding human instructions. We can adapt to follow human instructions.

$\rightsquigarrow$ The need of supervised finetuning (SFT), or sometimes simply called finetuning.

- SFT has exactly the same learning problem formulation as in pre-training. The major difference is that it is done on a much smaller finetuning dataset, compared to the pre-traning dataset.
- SFT also has the question-answer pair $(\boldsymbol{x}, \boldsymbol{y})$, in which the question $\boldsymbol{x}$ does not count into loss; recall the loss mask in your final project part II.
- Our final project part II is a SFT task, where the Math500 dataset is a mathematical dataset used to teach pre-trained model to have more mathematical solving ability.

Training Problem Formulation of SFT

Consider the finetuning data pairs $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$:

- $\boldsymbol{x}$ is the question, like a math question.
- $\boldsymbol{y}$ is the answer action to the question.

The SFT loss has exactly the same formulation as that of the pre-training loss:

$$\widehat{\boldsymbol{\theta}} \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \;\; \mathcal{L}(\boldsymbol{\theta}) = \sum_{i \in \mathcal{D}} \sum_{j=1}^{m} -\log\left(\mathbb{P}_{\boldsymbol{\theta}}[y_{i,j} | \boldsymbol{x}_i, \boldsymbol{y}_{i,1:j-1}]\right).$$

The difference is that $\mathcal{D}$ is very small and we do not count $\boldsymbol{x}$ in the loss using a loss mask.

- Our project part II is a format of post-training with SFT on math data, which can reasonably improve the mathematical performance of a very small model.

Memory Analysis for Finetuning LLMs

To fientune an LLM with M billion parameters using AdamW optimizer and mixed precision training approach, one needs to store:

- Float16 model: $2M$ GB
- Float32 model: $4M$ GB
- Float32 gradient: $4M$ GB
- Float32 momentum: $4M$ GB
- Float32 second moment: $4M$ GB
- Activation values.

In total, one needs at least $18M$ GB GPU RAM, plus additional memory for storing activations.

Examples:

- For Llama 3-8B, one needs at least 144GB GPU RAM, necessitating $2\times$ A100-80GB.
- For Llama 3-70B, $8\times$ A100-80GB is even not enough...
- This is why we can only tune a very small Qwen3 model.

PO: Preference Optimization of LLMs

- Preference optimization (PO) is to teach LLMs to prefer one answer over another, i.e., learning to have a preference.

PO is to fit the following kind of dataset

$$\{(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_l)\},$$

where $\boldsymbol{x}$ is the prompt, $\boldsymbol{y}_w$ is the preferred answer over $\boldsymbol{y}_l$.

After PO step, we expect

$$\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{y}_w|\boldsymbol{x}) > \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{y}_l|\boldsymbol{x}).$$

That is, the finetuned model prefers $\boldsymbol{y}_w$ over $\boldsymbol{y}_l$.

DPO: Direct Preference Optimization

- One very elegant way to performing PO is the DPO.
- DPO is based on BT model to measure preference.

DPO seeks to solve the following problem

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \left\{ -\log \sigma \left(\beta \log \left(\frac{\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{y}_w|\boldsymbol{x})}{\mathbb{P}_{ref}(\boldsymbol{y}_w|\boldsymbol{x})} \right) - \beta \log \left(\frac{\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{y}_l|\boldsymbol{x})}{\mathbb{P}_{ref}(\boldsymbol{y}_l|\boldsymbol{x})} \right) \right) \right\}.$$

- $\mathbb{P}_{ref}$ is the SFTed reference model, σ is the sigmoid (logistic) function.

Interpretation: DPO tries to max $\boldsymbol{y}_w$'s generation probability, while simultaneously min $\boldsymbol{y}_l$'s generation probability.

- Though DPO is widely used and has strong performance, it does not do "unlearning" properly. $\rightsquigarrow$We still have much to investigate in the future.

Pre-training

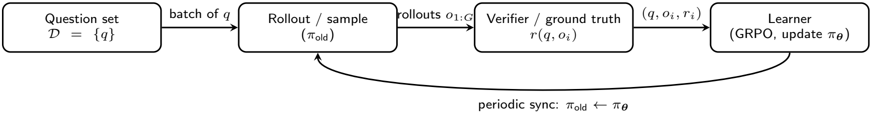
Post-training

Reasoning Models

LLM Reasoning Background

- Strong reasoning LLMs:
  - OpenAI o1 family, DeepSeek-R1, etc.
- Math reasoning is a key testbed:
  - Math500, AIME24/25, AMC, etc.
- Two main post-training paradigms:
  - Reinforcement learning with Verifiable Rewards (RLVR) such as GRPO, using verifiable reward signals.
  - Supervised finetuning (SFT) on reasoning traces generated by strong reasoning models like R-1.

RLVR and its Self-Training Framework



RLVR (GRPO) is one of such instances:

For prompt q and group of rollouts $\{o_i\}_{i=1}^{G} \sim \pi_{\text{old}}(\cdot \mid q)$, maximize

$$\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\min\Big(\rho_i A_i,\ \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon)A_i\Big)\right]$$

where,

$$\rho_i = \frac{\pi_{\boldsymbol{\theta}}(o_i \mid q)}{\pi_{\text{old}}(o_i \mid q)} \quad \text{(importance)}, \qquad A_i = \frac{r(q, o_i) - \mu_r}{\sigma_r} \quad \text{(advantage)},$$

$$\mu_r = \frac{1}{G}\sum_{j=1}^{G} r(q, o_j), \qquad \sigma_r = \text{std}\Big(\{r(q, o_j)\}_{j=1}^{G}\Big),$$

and $r(q, o_i)$ is the reward (ground truth answer) of rollout o_i to question q.

Understanding RLVR (GRPO)

- RLVR (GRPO) uses self-generated data for improving its math ability.
- The training signal comes from the outside verifier, which gives a reward.

Consider GRPO without clipping. Mathematical intuition:

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \; (\text{positive } A_i)\pi_{\boldsymbol{\theta}}(o_i \mid q) + (\text{negative } A_j)\pi_{\boldsymbol{\theta}}(o_j \mid q).$$

- Fit correct answer, while against wrong answer.

Pass@k

- In reasoning models, we often allow the model to produce multiple attempts for the same problem.
- This is a form related to test-time scaling.

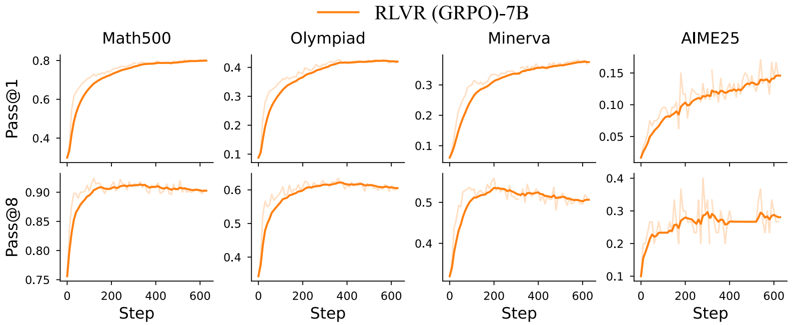
Pass@k definition:

- For each question, sample k independent answers from the model.
- Pass@k is the probability that at least one of these k answers is correct.
- Pass@1 is the usual top-1 accuracy.

Interpretation:

- Basically, allow answer more than one times to the same question, aiming to get at least one correct answer from the k answers. $\rightsquigarrow$ Test-time scaling.

GRPO on Qwen2.5-Math



- Pass@1 and pass@8 over training steps for Qwen2.5-Math-7B.
- GRPO shows improvements on mathematical reasoning ability.

We will see in final project part III how test-time scaling can help improve mathematical reasoning.

The End.

Special thanks to

your Participation ;-)