Homework 3: Foundation Model and Generative Model

Deep Learning (84100343-0)
Autumn 2024
Tsinghua University

Important notes:

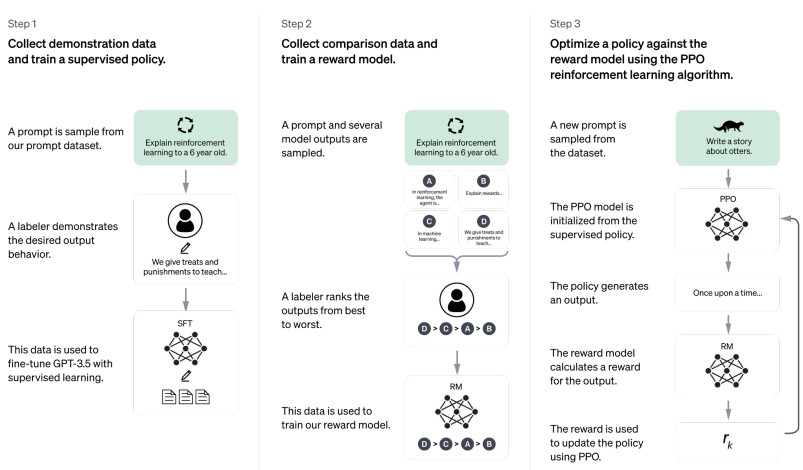
- This homework contains two parts: Foundation Models (FM) [**50pts**] and Generative Models (GM) [**50pts**], and will account for **20pts** in your final score.
- Please carefully read the guidelines for submission in Sec 3. DO NOT submit your checkpoint files, result files, or data. Your report should be concise and NO MORE THAN 10 pages.

1 Part One: Foundation Model

Foundation models have emerged as a pivotal domain within deep learning, and have benefited from increasing data amount and computational resources fed into scalable architectures such as Transformers [14]. Large language models like ChatGPT, as the most representative achievement in fundation model research, are further getting envolved within newly proposed techniques (instruct tuning, RLHF, etc.) based on the GPT-style next-token-prediction framework.

In this assignment, you will develop your own (perhaps not sufficiently "large") language model by further training of base model GPT-2 [9]. Following techniques of ChatGPT, you will work on two sub-tasks:

1. Supervised fine-tuning (SFT) [15]: Train language models to imitate Q&A policies.
2. Preference alignment: Train the language model to align with human prefenernce. This is traditionally achieved by Reinforcement learning with human feedback (RLHF) [8], but direct preference optimization (DPO) [10] is an alternative especially when data and time is limited.



Deep Learning (Autumn 2024), Tsinghua University.

1.1 Self-Attention

Self-attention modules are of the most significant mechanisms in Transformers [14]. Denote $X \in \mathbb{R}^{T\times h}$ as input, with T as the number of tokens and h as the hidden dimension, the calculation of a self-attention module is defined as

$$Q = XW_Q, K = XW_K, V = XW_V \in \mathbb{R}^{T\times d}, \tag{1}$$

$$A = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}} + M\right) \in \mathbb{R}^{T\times T}, \tag{2}$$

$$O = AV \in \mathbb{R}^{T\times d}. \tag{3}$$

Here W_Q, W_K, W_V are parameters of the self-attention module, and d is the vector dimention. Here M refers to the mask matrix, where visible pairs refer to elements 0, and invisible pairs refer to significantly small values (such as `float("-inf")`). Multi-head self-attention is an improvement which splits Q, K, V into multiple matrices calculated independently.

Your task: (5 points) Complete the code in `multi_head_self_attention` in `attention.py`. Then run `attention.py` to evaluate your answer. The program will output relative errors w/o and w/ masks, respectively. Unless any implementation error, the relative error should not exceed 10^{-6}.

1.2 Supervised Fine-Tuning (SFT)

Supervised fine-tuning [15] aims to tune a pre-trained model using the following next-token-prediction objective

$$\mathcal{L}_{\text{LM}}(x) = \sum_{i=1}^{T} -\log P(x_i|x_{1:i-1}). \tag{4}$$

While for practical convenience, one directly lets $y_{1:T} = x_{2:T+1}$ and adds proper special tokens (such as `<BOS>`, `<EOS>` and padding tokens[1]) when processing data, and the objective[2] further becomes

$$\mathcal{L}_{\text{LM}}(x, y) = \sum_{i=1}^{T} -\log P(y_i|x_{1:i}), \tag{5}$$

which is learned with a gpt-style causal transformer.

Your task:

- **(5 points)** Complete the code in `gpt.py` to define a causal mask.
- **(5 points)** Download GPT-2 pre-trained weights and data from `https://cloud.tsinghua.edu.cn/f/dc0ff3454fff44c9b626/?dl=1`. Place the unzipped two folders (`checkpoints/` and `data/`) under the main folder (`fm/`). Then run `test.py`, and show the generated results.
- **(10 points)** Train the model through `train_sft.py`. Report the curve of the training loss, and finally run `test.py` (you should specify the directory of fine-tuned model) and show the generated results.

1.3 Direct Preference Optimization (DPO)

While RLHF [8] learns a reward model and trains language models through reinforcement learning algorithms (such as PPO [12]), DPO [10] directly minimize the following objective with prompt x, winning response y_w, losing response y_l as input:

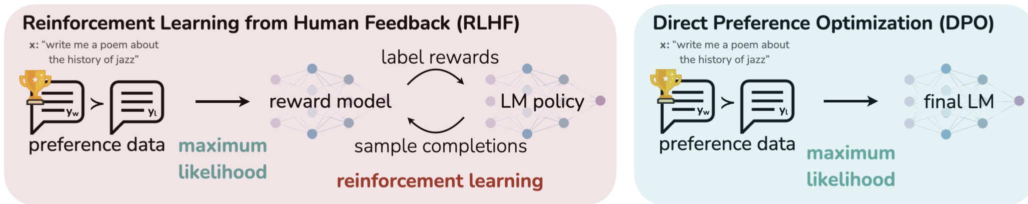
$$\mathcal{L}_{\text{DPO}}(x, y_w, y_l) = -\log\sigma\left(\beta\log\frac{p_{\text{DPO}}(y_w|x)}{p_{\text{SFT}}(y_w|x)} - \beta\log\frac{p_{\text{DPO}}(y_l|x)}{p_{\text{SFT}}(y_l|x)}\right) \tag{6}$$

The "correctness" that the model prefers y_w than y_l could be further derived by

$$\beta\log\frac{p_{\text{DPO}}(y_w|x)}{p_{\text{SFT}}(y_w|x)} - \beta\log\frac{p_{\text{DPO}}(y_l|x)}{p_{\text{SFT}}(y_l|x)} > 0 \tag{7}$$

[1] Referred to "`attention_mask`" in code, where details could be ignored in your task.

[2] Referred to Line 106 in `trainer.py`.



Your task:

- **(5 points)** Prove that

$$\beta \log \frac{p_{\text{DPO}}(y_w|x)}{p_{\text{SFT}}(y_w|x)} - \beta \log \frac{p_{\text{DPO}}(y_l|x)}{p_{\text{SFT}}(y_l|x)} \tag{8}$$

$$= \beta \log \frac{p_{\text{DPO}}(x, y_w)}{p_{\text{SFT}}(x, y_w)} - \beta \log \frac{p_{\text{DPO}}(x, y_l)}{p_{\text{SFT}}(x, y_l)}. \tag{9}$$

  Remark: This indicates that, it suffices to define the concatenation $c_w = [x, y_w]$ and use the log-probability

$$\log p_{\text{DPO}}(x, y_w) = \log P(c_w) = \sum_{i=1}^{T} \log P\big((c_w)_i | (c_w)_{1:i-1}\big) \tag{10}$$

  similarly calculated as Eq. (4). The implementation of log-probability is further provided in `get_log_p` function in `gpt.py`.

- **(15 points)** Implement the shared step in `trainers.py` to compute the DPO loss and accuracy. Then run `train_dpo.py` to train your model from last sub-problem. Select proper hyper-parameters, report the curve of (training and validation) losses and accuracies. Finally, run `test.py` and show the generated results.
  Hint: It is recommended to use relatively small (such as 10^{-6}) learning rate, but it is acceptable if your model failed to generate plausible responses. Simply make sure your code is correct.

1.4 Enhancement of Language Models

There exist many aspects to further enhance your language model, including algorithm, hardware, hyper-parameter tuning, etc. You could choose one of your interest.

Your task: **(5 points)** Choose one of the following tasks:

- Efficient fine-tuning algorithm: Try low-rank adaptation (LoRA) [4] in your language model. You can refer to `loralib`[3] by replacing the linear layers with `lora.Linear`. Re-train the model and report your experiment results.
- Efficient hardware-focused training strategies: Try one parallel strategies such as Distributed Data Parallel (DDP) and Fully Sharded Data Parallel (FSDP). Re-train the model and report your experiment results.
- Hyper-parameter tuning: Provide an analysis of the supervised fine-tuning (SFT) phase by comparing at least 3 numbers of training steps such as 0%, 50%, 100% (or data amount, whatever you want), and report your experiment results.

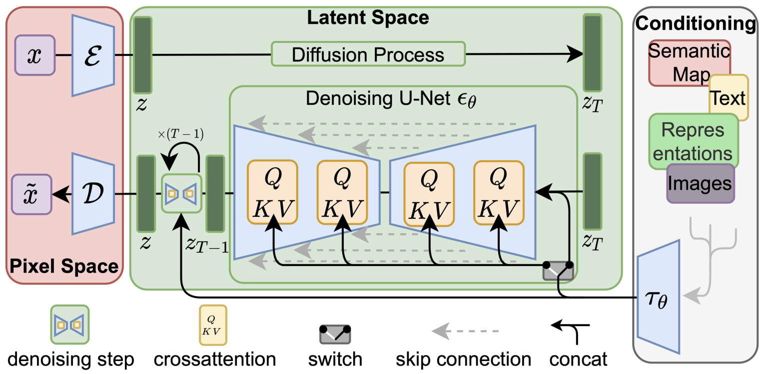
2 Part Two: Generative Model

Generative models aim to build a complex distribution through noise-data mapping with deep neural networks. Recently, the study of generative models arises researchers' interest, especially in field of image generation and cross-modality generation. By incorporating various generative learning

[3] `https://github.com/microsoft/LoRA`

algorithms (VAE [6], GAN [1], Diffusion Models [3]) and effective network architectures, it is possible to generate high-resolution images.

In this assignment, you will develop a Latent Diffusion Model (LDM [11]) for image synthesis on MNIST and CIFAR-10. The latent diffusion model first learns a VAE model to encode the raw image into latent spaces for efficiency, which might require a discriminator from GAN as auxilarity. The model then learns a denoiser in the latent space for generation.



2.1 Variational Auto-Encoder (VAE)

VAE [6] originally aims to learn a decoder as a generative model with the latent variable satisfying the Gaussian prior distribution $\mathcal{N}(0, I)$. To learn such a model, VAE incorporates a variational encoder to approximate the posterior distribution of the latent variable with samples observed. Denote the encoder as ϕ which outputs two tensors μ, σ[4], i.e., $(\mu, \sigma) = \phi(x)$, and

$$z \sim q_\phi(z|x) = \mathcal{N}(z; \mu, \exp(\sigma)), \tag{11}$$

which is further implemented by re-parameterization trick

$$z = \mu + \exp\left(\frac{1}{2}\sigma\right) \odot \epsilon, \epsilon \sim \mathcal{N}(\epsilon; 0, I) \tag{12}$$

Denote the decoder as ψ, and the VAE is trained with the following ELBO-based objective

$$\mathcal{L}_{\text{VAE}}(x) = \mathbb{E}_{z\sim q_\phi(z|x)}[-\log p_\psi(x|z)] + \text{KL}\big(q(z|x)||p(z)\big) \tag{13}$$

$$= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\|x - p_\psi(x|z)\|_2^2\right] + \lambda\text{KL}\big(q(z|x)||p(z)\big), \tag{14}$$

where $p(z) = \mathcal{N}(0, I)$, and λ are hyper-parameters to tune.

Your task:

- **(10 points)** Directly write down the closed form of $\text{KL}(q(z|x)||p(z))$, where $q(z|x) = \mathcal{N}(z; \mu, \exp(\sigma))$ and $p(z) = \mathcal{N}(0, I)$. Then complete the code in `loss.py` to implement the KL-divergence loss.
- **(10 points)** Complete the code in `model.py` to implement the forward process of VAE. You should implement the re-parameterization trick for gradient availability. Then train your VAE on MNIST and CIFAR-10 by running `train_vae_mnist.py` and `train_vae.py`, respectively. Tune $\lambda \in \{1, 10^{-3}, 10^{-6}\}$ and report your results by showcases.

  Hint: In the saved `.png` files, the 3 columns of images correspond to groundtruth, reconstruction and samples, respectively. Normally, your VAE might fail to generate (not reconstruct) samples on CIFAR-10.

[4]The σ here refers to "log-variance" for numerical stability.

2.2 Generative Adversarial Network (GAN)

In case that VAE's reconstruction is not precise and has artifacts, one solution is to introduce the discriminator from GAN [1] for adversarial training. Unlike the generator in GAN to generate samples from random noise, in our task, the discriminator treats the reconstruction $\hat{x}$ as fake samples and the groundtruth x as real samples, trained with the following hinge loss[5]

$$\mathcal{L}_D(x, \hat{x}) = \frac{1}{2}\Big(\max\left(0, 1 - D(x)\right) + \max\left(0, 1 + D(\hat{x})\right) \Big). \tag{15}$$

The GAN loss for the VAE is

$$\mathcal{L}_G(\hat{x}) = -D(\hat{x}). \tag{16}$$

Your task: (10 points) Implement GAN losses in `loss.py`, then tune the hyper-parameter `gan_weight` and `gan_loss_start` to improve VAE training. Report your experiment results on CIFAR-10 by showcases.

2.3 Denoising Diffusion Probablistic Models (DDPM)

In DDPM [3], a denoiser is learned to distinguish the noise from disturbed samples $z_t = \sqrt{\bar{\alpha}} z_0 + \sqrt{1 - \bar{\alpha}}\epsilon$, i.e.,

$$\mathcal{L}_{\text{DDPM}}(z) = \mathbb{E}_{t \in \mathcal{U}[1,T], \epsilon \in \mathcal{N}(0,I)} \left[\left\| f_\theta\left(\sqrt{\bar{\alpha}} z_0 + \sqrt{1 - \bar{\alpha}}\epsilon \right) - \epsilon \right\|_2^2 \right] \tag{17}$$

During the sampling process, DDPM starts from $x_T \sim \mathcal{N}(x_t; 0, I)$ and iteratively sample from $p_\theta(x_{t-1}|x_t)$ defined by the denoiser f_θ. In latent diffusion models (LDM), the DDPM is trained on the latent $z = q_\phi(x)$.

Your task: (15 points) Complete the code of functions `q_sample` and `sample` to implement the training and sampling process for DDPM, respectively. Then load the VAE model with the best reconstruction performance and train your diffusion model by running `train_ldm.py`. Report the showcases.

Hint: Tuning diffusion models is difficult, **but** it is acceptable if your model generates blurred, distorted or noisy samples. Simply make sure your code is correct.

2.4 Enhancement of LDMs

There exist many aspects to further enhance your latent diffusion models, including sampling acceleration, generation frameworks, quantitive analysis, etc. You could choose one of your interest.

Your task: (5 points) Choose one of the following tasks:

- Sampling acceleration: Implement DDIM [13] for sampling acceleration. Report showcases under various number of sampling steps.
- Generation framework: Implement flow-based generative model [7] for better generation. Compare DDPM with flow-based models and report the showcases.
- Quantitive analysis: Implement Fréchet Inception Distance [2] (FID) and report the FID value for your models.

3 Submit Format

Submit your *code and report* as an Archive (zip or tar). Your code should only contain `.py` files, not checkpoint files, result files, or data. The report is supposed to cover your **question answering**, the model **technical details**, **experimental results**, and necessary **references**. The length of your report is restricted to 10 pages.

[5] Practically, we adopt PatchGAN [5], where the discriminator outputs feature map instead of a value, and the loss is averaged spatially.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[2] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[4] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[5] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[6] Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[7] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[8] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[9] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[10] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

[11] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[13] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[14] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[15] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.