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

MDS5210 Final Project · 2023 Fall
Large Language Models — A Case Study from GPT-2
Due: (23:59), December 24, 2023.

Contents

1 Introduction

Large Language Models (LLMs) are a transformative development in the field of artificial intelligence and machine learning, having a profound impact on numerous applications including text generation, translation, summarization, and question-answering, among others [1]. These models, which are trained on vast amounts of text data, have the ability to generate human-like text that can be remarkably coherent, insightful, and contextually nuanced. The power of LLMs comes from their pre-training process, which involves learning from vast amounts of text data. Note that the name "GPT" is the acronym of "Generative Pre-trained Transformer". During the pre-training phase, these models learn to predict the next word in a sentence, given the context of the preceding words. This process allows them to learn grammar, facts about the world, reasoning abilities, and even some level of common sense.

The motivation to studying LLMs stems from their ability to understand and generate language in a way that is impressively similar to human beings. They can capture the context of a conversation, respond to prompts with relevant and coherent answers, and even generate creative writing. This capacity for natural language understanding and generation opens up a myriad of possibilities for applications in various domains, from customer service and digital assistants to content creation and beyond. One of the most well-known LLMs is ChatGPT (with GPT-3.5 and GPT-4 as its cores), developed by OpenAI, has demonstrated remarkable capabilities in understanding and generating text. It is widely utilized. Other notable LLMs include Llama family from Meta [6]. Nowadays, LLMs represent a significant advancement in the field of artificial intelligence, offering a powerful tool for natural language understanding and generation. The development and refinement of these models hold immense promise for a wide range of applications and are a compelling motivation for further research and industry development in this area.

However, LLMs are not without their challenges. For instance, they require substantial computational resources for pre-training, making it impossible to pre-train a LLMs for a small group. Fortunately, one can *fine-tune* a released LLMs for certain downstream tasks, which considerably reduces the requirement on the computational resources. In this project, we aim to fine-tune an early version of GPT, i.e., *GPT-2* [4].

2 Prerequisite Knowledge

2.1 Prompts

In the context of language models, a prompt is the initial input that the model receives to which it responds by generating subsequent text. It serves as a starting point for the model's output, guiding the direction and context of the generated response. Prompts can be as short as a single word or as long as multiple sentences. The quality and specificity of a prompt can significantly influence the quality, relevance, and accuracy of a language model's output. In essence, a prompt acts as the seed for the text generation process, shaping the model's understanding of the task at hand and the desired output. You may simply regard prompts as the input texts, e.g.,

```
Shenzhen is a great city.
```

is a prompt.

2.2 Tokenization

The input texts will be transformed into *tokens* before feeding into model. The step is called *tokenization*. The translator between the input text and the output token is called *tokenizer*. Modern NLP models apply subword unit tokenization technique. Here is an example:

```
Input: Shenzhen is a great city.
Subword split: S hen zhen  is  a  great  city .
Encoded tokens: 50, 831, 46732, 318, 257, 1049, 1748, 13
```

Since we will finetune GPT-2, we use GPT-2's tokenizer as well. It consists of 50257 tokens in total.

2.3 Large language models (LLMs)

The GPT-2 model is an auto-regressive model (a.k.a. causal model), which is also true for more contemporary LLMs like GPT-4. Let f_θ be our LLMs and θ be the model parameters. Let x be a text of length T, with x_i being the i-th element of x. Let X be the text generated by the model, which is a random variable. The probability for model f_θ to generate x can be decomposed as the multiplication of a set of conditional probabilities:

$$
\begin{aligned}
& p(X = x|f_\theta) \\
=& p(X_1 = x_1)p(X_{\{2:T\}} = x_{\{2:T\}}|x_1, f_\theta) \\
=& p(X_1 = x_1)p(X_2 = x_2|x_1, f_\theta)p(X_{\{3:T\}} = x_{\{3:T\}}|x_1, x_2, f_\theta) \\
=& p(X_1 = x_1)p(X_2 = x_2|x_1, f_\theta)\cdots p(X_i = x_i|x_1, \cdots, x_{i-1}, f_\theta)p(X_{\{i+1:T\}} = x_{\{i+1:T\}}|x_1, \cdots, x_i, f_\theta)
\end{aligned}
$$

Hence, LLMs generate text in the following manner:

(1) Receiving your input prompt (which may be a question or a story you would like the model to do the continuation) to the LLMs.

(2) The model generates the logits of shape (B, T, C); see our tutorials for details about the logits[1].

(3) Select the logits at the last position of the sequence[2], which is of size (B, C). Apply softmax to the tensor and get the probability distribution of the next word.

(4) Sample the next token according to the probability distribution.

(5) Append the newly generated token to the input, repeat the aforementioned steps, until the maxmimum length / or other stopping criterion is reached.

Intuitively, you can regard LLMs as a sequential content generator. Based on your prompt, it generates the most probable next word and augment this generated word to the up-to-date sentence, and repeat such a processing. In terms of the techniques used to find the most probable next word, LLMs reduce such a problem to a multi-class classification problem and utilize the logistic regression technique for achieving this classification task.

[1] B: batch size, T: sequence length, C: vocabulary size.

[2] Use `logits[:, -1, :]` to fetch the tensor

2.4 Fine-tuning

Fine-tuning comes after pre-training. During this process, the model is further trained on a narrower, specific dataset related to the task it will perform. For example, if the task is to generate medical advice, the model is fine-tuned using medical texts. This additional, specific training helps the model to generate better, more relevant responses when presented with prompts related to its task. The fine-tuning process can be perfectly understood from an optimization perspective: Regard the pre-trained model parameters as the initial point point θ^0, and then repeat exactly the pre-training processing with your algorithm starting from θ^0 *on a much-much-much smaller fine-tuning dataset.*

2.4.1 Why fine-tuning?

The most prominent reason for studying fine-tuning is described in the previous introduction section. Namely, it is almost impossible for a small group to pre-train their own LLMs due to the requirement on the huge computing resources. Apart from that, there are other reasons for studying fine-tuning. One promising reason is that the pre-trained model itself is purely a text completion / continuation machine and its generation may not align optimally with specialized tasks. One failure example is shown below:

```
Input: What is the tallest building in the world?
Output: What is the tallest building in the world in 2020? What is the tallest
building in the world in 2021?
```

We can see that the model does not answer the question. Instead, it just do text completion following the input question. Fine-tuning aims to solve this issue by adapt the pretrained model to perform specific tasks. In this project, we will build a dialogue model by fine-tuning on GPT-2 (Generative Pre-trained Transformer-2) to generate contextual relevant and informative response to human prompts.

2.4.2 Supervised fine-tuning (SFT)

The most fundamental technique for fine-tuning is called supervised fine-tuning (SFT). It is basically the same as the supervised learning we learned in lecture and assignments, except that the label becomes a target token instead of classification label. Consider the previous example:

```
Input: Shenzhen is a great city.
Subword split: S hen zhen  is  a  great  city .
Encoded tokens: 50, 831, 46732, 318, 257, 1049, 1748, 13
```

In SFT, we forward the token 50 into our model and let it to fit the label 831. Then, we feed tokens [50, 831] into the model and fit the label 46732. We continue repeating the procedure until the last token is reached. When implementing the model, the forward process can be done simultaneously, yielding logits of shape (B, T, C). The loss is calculated simultaneously as well. Similar to logistic

regression, we will use the cross entropy loss:

$$-\log\left(\frac{\exp(\text{logits}_l)}{\sum_{i=1}^{C}\exp(\text{logits}_i)}\right).$$

The SFT loss for one input is the average loss over every sequence position. Please refer to our tutorials for details about this process.

3 Dataset

We will use the Anthropic's "HH-RLHF" dataset[3] for our project. In particular, we will use the "helpful-base" division of the dataset. It consists of 43835 samples for training dataset and 2354 samples for test dataset. Each sample has one "chosen" dialogue and one "rejected" dialogue where the agent answer to the same question. The dataset was collected in a way that for "chosen" dialogue, the agent tends to provide more helpful and informative response than "rejected" dialogue. A typical example is shown below:

```
# Chosen:
    Human: Have any new technological advances been made in regards to electricity
        within the past few years?

    Assistant: One thing that's recently gained a lot of attention is the potential of
         electricity to help with bone repair and regeneration.  Some research suggests
         that electricity could be used as an "artificial stimulus", essentially
        mimicking the action of biological bone growth and bone formation. This
        research is still in its early stages and requires further study, but could be
        especially interesting in the context of bone loss and treatment, in
        conjunction with potential new treatments to Osteoporosis.  You may also find
        this video about the research interesting:  https://www.youtube.com/watch?v=1
        r6U39KAcbQ


# Rejected:
    Human: Have any new technological advances been made in regards to electricity
        within the past few years?

    Assistant: Let me think.. oh yes, there have been some!
```

We will use the "chosen" samples to fine-tune our model in a supervised manner, forming our SFT dataset.

4 Project Components

4.1 Base task: SFT [60 points]

The base task of our project is to implement SFT for GPT-2 based on the "HH-RLHF" dataset, to facilitate its dialogue ability with human. You may use any optimizer embedded in PyTorch. You

[3] https://arxiv.org/pdf/2204.05862.pdf

need to implement the following items:

(A) **Plot the curve of training error and test error.** Since the dataset is relatively large, evaluating across the whole dataset is time consuming. To compensate the issue, you may estimate the loss by sampling a portion of data independently from the dataset. A recommended size for the estimation is 200.[4]

(B) **Evaluate the dialogue ability.** Evaluate the model's dialogue ability during training. The most straightforward way is to evaluate the text generation by yourself. However, it may be extremely costly as data scale becomes large. An alternative approach is to use well-trained model to perform the auto-evaluation. For instance, you may use "OpenAssistant/reward-model-deberta-v3-large-v2"[5] to judge the dialogue ability, in which you just need to input your generated dialogue to this model, it will output a score. ChatGPT (either GPT3.5-turbo or GPT4-turbo) is a better candidate model to assist in evaluating the dialogue ability, while it is too expensive to use.

(C) **Summarize your findings.** Briefly summarize and discuss your observations. For instance, you may play with your fine-tuned LLMs, by asking some prompts and compare the model's answer with that before fine-tuning. You may also study the overfitting issue (if any) through the fine-tuning. Any other phenomena you observed and you think interesting to report.

Data snooping: Never use the test dataset to tune anything of training, since in this way you will have overfitting, resulting in a bad quality model.

4.2 Explorations [40 points]

We also have exploration directions detailed below. Each of them is of **20 points**. Even if you accomplish more than two explorations, **at most 40 points** in total can be earned (we certainly encourage you to explore more along these directions).

(a) **Impact of optimizer.** Compare the convergence using different optimizers such as SGD, SGD with momentum, SGD with Nesterov's acceleration, and Adam / AdamW. Compare the GPU memory consumption of different optimizers, and explain how the memory is utilized[6].

(b) **SFT dataset quality.** As discussed and illustrated in [6, Section 3.1], the quality of the fine-tuning dataset is all we need for training a high quality dialogue model rather than the number of data samples. Explore the effect of data quality on the performance of fine-tuned models, by selecting some high quality data samples (this reduces the dataset size). You may use "OpenAssistant/reward-model-deberta-v3-large-v2" to help select high quality data samples. You can try to reduce the dataset size to 35000, 30000, 25000, 20000, 15000, and 10000, and then train your model on the carefully chosen but smaller dataset. Report and summarize

[4] You may want to use WandB to trace your loss and GPU memory usage during the training; see `https://docs.wandb.ai/quickstart` for a quick tutorial.

[5] See `https://huggingface.co/OpenAssistant/reward-model-deberta-v3-large-v2` for the usage of the model.

[6] You may refer to https://medium.com/@Shrishml/breaking-the-gmemory-barrier-how-zero-revolutionizes-large-language-model-training-8e00d2e2f30a on the memory usage.

weather the newly fine-tuned models have worse, equal, or better dialogue ability than the one trained the whole SFT dataset.

(c) **Parameter Efficient Fine-tuning.** So far, our fine-tuning has been full parameters training, which updates all the parameters in θ using the SFT dataset. The method LoRA [3] significantly reduces the training parameters by seeking for a low-rank update of part of the weight matrices; see our slides on low-rank matrix factorization in the PCA session. LoRA can reduce the training parameters up to 1%. However, it may suffer from performance degradation when dealing with complex tasks, due to the limited search space. To this end, we compare the performance of LoRA and full-parameter fine-tuning. In particular, you are asked to implement the LoRA algorithm[7]. Then, compare the LoRA trained model and full-parameter tuned model in terms of train and test loss and dialogue quality. You also need to report the performance of the model trained with different LoRA rank.

(d) **Fine-tuning from human feedback through DPO.** Other than SFT, fine-tuning LLMs includes one additional phase called fine-tuning from human feedback (HF), which has been shown to improve the model's capability to downstream tasks. In this phase, the model is trained based on a preference dataset, where each sample consists of two answers to the same question, with one answer being more preferred than the other. Indeed, the whole Anthropic's "HH-RLHF" dataset shown in Section 3 is a dataset for HF, in which each data point includes a preferred (chosen) sample and a less preferred (rejected) sample. Roughly speaking, the HF phase increases the probability of generating chosen answer and decreases the probability of rejected answer. In this part, you are asked to implement a simple yet efficient HF algorithm called direct preference optimization (DPO)[8] [5], and compare its dialogue performance with the previous SFT-only tuned model.

(e) **Scaling effect.** Scaling up language model has been shown to predictably improve performance on a wide range of downstream tasks. Here we explore scaling effect with respect to model size, as measured by the number of model parameters. Specifically, we focus on GPT-2 model class of different sizes: GPT-2 (124M), GPT-2 Medium (355M), GPT-2 Large (774M), GPT-2 XL (1.5B). You are asked to examine how the model scale affects the performance. To complete this task, you need to seek for computing resources by yourself. A GPU with at least 24 GB memory is required. You may use shorter sequence length (block size) to reduce the memory consumption.

(f) **Optimal model scale under fixed computation budget.** While the larger model size often exhibits better performance, it requires more computation cost for each training step. An important question in industry is: Given a fixed computation budget, how should one choose the model size to attain the minimum loss?

We use floating-point operation (FLOP) to denote the computation budget, which is a function of model parameters N and number of tokens used for training D[9]. It can be roughly estimated by $\text{FLOP}(N, D) = 6ND$. You are asked to fix a range of reasonable FLOP budgets, and find

[7] In fact, the project code has already included LoRA implementation. You only need to set `cfg.lora_rank` to be positive in order to use LoRA. However, you need to make sure that you understand the code; see `https://github.com/microsoft/LoRA/tree/main`.

[8] See `https://zhuanlan.zhihu.com/p/642569664` for a interpretation.

[9] See Appendix F of [2] for a more precise explanation and computation.

the best model among aforementioned GPT-2 models in part (e). Briefly summarize your results in a table.

(g) **Memory-efficient optimizer.** The parameter number of LLMs are huge, imposing bottleneck on GPU memory. One straightforward way is to change the optimizer from Adam/AdamW to stochastic gradient descent (SGD). However, this will inevitably slow down the convergence. This part asks you to propose a memory-efficient optimizer that significantly reduces the memory usage, while maintaining the optimization convergence speed. One possible direction is to extend / mimic LoRA in part (c). Note that our goal is to find a memory-efficient method with fast convergence, so that small memory GPU can fine-tune large models, increasing computational time is totally acceptable.

5 How to Start?

5.1 Code Structure

If you have installed Git[10], clone the code by running

```
git clone https://github.com/Ledzy/MDS5210-23fall.git
```

Otherwise, go to `https://github.com/Ledzy/MDS5210-23fall`, click `Code` and select `Download ZIP`.

The file structures looks like this:

```
/src
    configs.py  #configuration of models
    dataset.py  #definition of datasets
    evaluate.py  #GPT4 evaluation API
    gpt.py  #definition of models
    prepare_sft_dataset.py  #script for downloading datasets
    requirements.txt  #package requirements
    tokenizer.py  #definition of tokenizer
    train_sft.py  #training script
    trainers.py  #file containing training details
Kaggle_training.md #Instruction on using Kaggle GPU
main.ipynb #script that is actually executed in Kaggle
dataset-metadata-template.json #template for creating dataset-metadata.json
kernel-metadata-template.json #template for creating kernel-metadata.json
```

You will mainly work with `train_sft.py` and `trainers.py`.

The file `train_sft.py` have prepared the SFT dataset loading, as well as necessary configurations. You may adjust input of `get_configs` to change the model configuration. You can add the argument `cfg.lora_rank` to apply LoRA-based finetuning[11]; see `gpt.py` for the detailed implementation of

[10] `https://git-scm.com/`

[11] `https://github.com/microsoft/LoRA/tree/main`.

LoRA. You will complete your main training code in `trainers.py`, specifically in `fit` function of `SFTTrainer`. You may refer to our tutorial 3 for the implementation.

5.2 Training with GPU

Training large language model often involves huge amount of tensor operations, making GPU an almost indispensable computation resource.

You have a powerful GPU with at least 15 GB GPU RAM. Great! you can run all the experiments locally. Download the dataset from `https://www.kaggle.com/datasets/jonery/finetuning-dataset` and put `sft_train.json` and `sft_test.json` in `src/` directory.

You do not have such a GPU. You have to use the GPU provided by the Kaggle platform. Kaggle offers 30 hours of free P100 GPU usage per week for every account, which should be enough for the project. We have created a step-by-step guidance on how to use Kaggle's computation resource; see `Kaggle.md`. In this case, you do not need to download dataset locally as it can be accessible in Kaggle by specifying `dataset_sources` in `kernel-metadata.json`; see `Kaggle_training.md` for more information.

Here are some suggestions for efficient code development:

- **Prioritize your job**. Completeness is much more important than being perfect.
- Use your local machine to develop your code. Submit the code to Kaggle only when everything is ready.
- Save your logs during the training. It is highly suggested to use Wandb[12] to save the plots and text generations.
- Start with `batch_size=1` when developing your code in order to avoid out of memory error.

6 Project Submission

Each group should submit a `zip` file which includes

- **Report**. It should be written using the latex template provided in Blackboard. The main body of the report should be concise and contains **no more than 6 pages** excluding references and appendix. You may put additional results into the appendix.
- **Code**. Please include a `readme` file that explains the submitted code, explaining the code structure and how to execute the code.

The report should cover:

- What is this project about (a few paragraphs).
- Designs and descriptions of your methods.

[12] https://docs.wandb.ai/tutorials

- Experiment results and your observations in SFT.
- Experiment results and your observations in exploration topics.
- Any other contents you think is meaningful to include.

Responsibilities of each group member: Please clearly indicate the responsibilities and contributions of each group member in the footnote of the first page. A group member with zero contribution cannot earn the score.

Flexibility of our project: Please note that this is a project, not a homework. There is no standard answer. The goal is to encourage you to explore as much as you can. The result may be favorable or unfavorable; both are acceptable once you provide observations and possible explanations.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[3] 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.

[4] 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.

[5] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

[6] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.