[참고자료] LLM Efficiency tutorial

- Flash Attention 적용 및 팁
- LoRA 적용 및 변형된 LLM 구조 이해
- Quantization
- training
- inference 다양성 (w/ seed)
- prompt engineering

/home/work/intext/LLM_Efficiency_Tutorial에 자료들이 있습니다.

폴더 내에 있는 ipynb 파일을 기준으로 진행되며, 본 컨플은 사진 + 다른 추가 팁들을 비교/기록 하기 위해 참고용으로 작성되었습니다.

본 tutorial은 mistral로 위의 내용을 테스트 해보는 code reivew 격의 성격을 띄고 있고, tutorial을 문제없이 이해하신다면 왠만한 파생 efficiency github code를 이해하는데 많은 도움이 있을 것 같습니다.

1. Flash Attention 2

체감상 정말 중요한 method

FlashAttention-2 is currently supported for the following architectures:

- Bark
- Bart
- DistilBert
- GPTBigCode
- GPTNeo
- GPTNeoX
- Falcon
- Llama
- Llava
- VipLlava
- MBart
- Mistral
- Mixtral
- OPT
- Phi
- Qwen2
- Whisper

- Flash Attention 설명(hugginface): https://huggingface.co/docs/text-generation-inference/conceptual/flash_attention
- Flash Attention 사용 방법 설명(huggingface): https://huggingface.co/docs/transformers/perf_infer_gpu_one
- Flash Attention version update 현황: https://github.com/Dao-AILab/flash-attention?tab=readme-ov-file#changelog

2. LoRA and QLoRA

자세한 적용 방법은 efficiency_tutorial.ipynb 참고

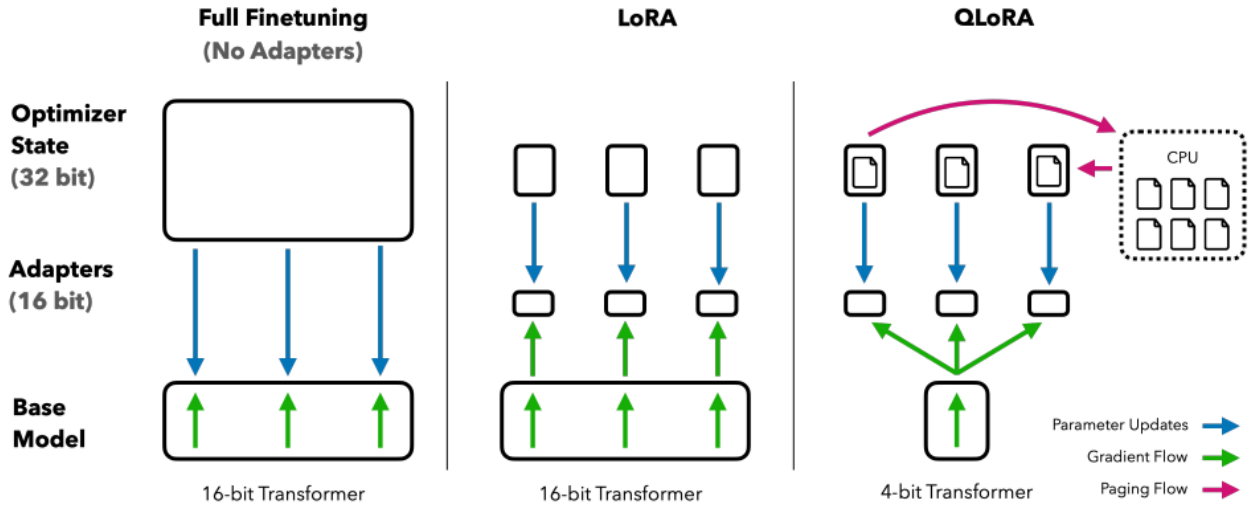

Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

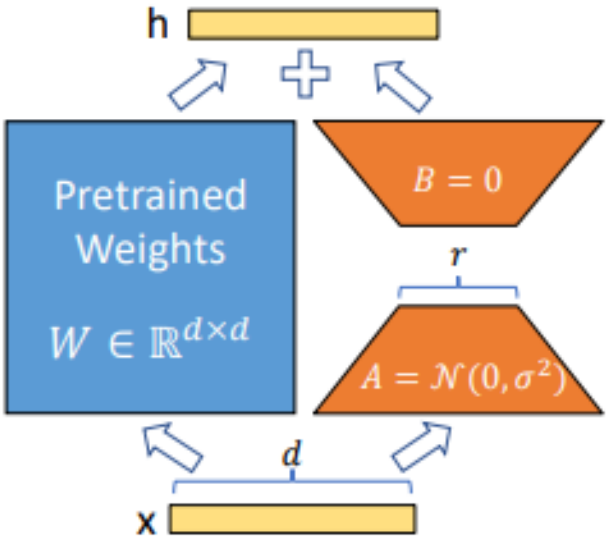

Figure 1: Our reparametrization. We only train A and B.

3. DPO

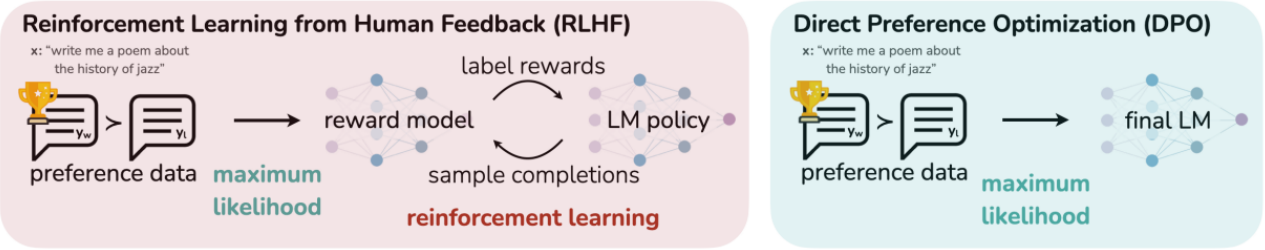

Figure 1: **DPO optimizes for human preferences while avoiding reinforcement learning.** Existing methods for fine-tuning language models with human feedback first fit a reward model to a dataset of prompts and human preferences over pairs of responses, and then use RL to find a policy that maximizes the learned reward. In contrast, DPO directly optimizes for the policy best satisfying the preferences with a simple classification objective, without an explicit reward function or RL.

alpaca format이랑 dpo format 차이가 있음.

→ data preprocessing 할 때 유의.

alpaca format	dpo format

{
{ 'instruction': '~',
'input': '~',
'output': '~' } x n개
}
	{ "prompt": ['~~' x n개],
	"chosen": ['~~' x n개],
	"rejected": ['~~' x n개] }

```
dpo_dataset_dict = {
    "prompt": [
        "hello",
        "how are you",
        "What is your name?",
        "What is your name?",
        "Which is the best programming language?",
        "Which is the best programming language?",
        "Which is the best programming language?",
    ],
    "chosen": [
        "hi nice to meet you",
        "I am fine",
        "My name is Mary",
        "My name is Mary",
        "Python",
        "Python",
        "Java",
    ],
    "rejected": [
        "leave me alone",
        "I am not fine",
        "Whats it to you?",
        "I dont have a name",
        "Javascript",
        "C++",
        "C++",
    ],
}
```

4. inference 다양성

Transforemr library 기준 LLM inference 하는데는 크게 2가지 방법이 사용 됨.

pipeline이 좀 더 high(?) level이라 기능성인 측면이 있지만, 그만큼 제한 사항도 따르는 편.

아래는 예시

pipeline

```python
def infer_score(instruction="", input_text=""):
    prompt = prompter.generate_prompt(instruction, input_text)
    #print(prompt)
    output = pipe(prompt,
     temperature=0.1,  # 온도
     top_k=40,         # 상위 k개 선택
     top_p=0.75,       # 확률 누적 상위 p
     do_sample=False,  # 샘플링 사용 여부
     num_beams=4,      # 빔 탐색의 빔 개수
     repetition_penalty=1.4,  # 반복 패널티
     eos_token_id=2,   # 문장 종료 토큰 ID
     num_return_sequences=1,  # 반환할 시퀀스의 수
     max_new_tokens=1)      # 생성할 새 토큰의 최대 개수
    s = output[0]["generated_text"]
    result = prompter.get_response(s)

    return result
```

generate() method

```python
# 최적화 해야 하는 부분
output = model.generate(
    **inputs,
    max_new_tokens=max_gen_len,  # 생성할 새 토큰의 최대 개
    temperature=0.1,             # 온도
    top_k=40,                    # 상위 k개 선택
    top_p=0.75,                  # 확률 누적 상위 p
    do_sample=False,             # 샘플링 사용 여부
    num_beams=1,                 # 빔 탐색의 빔 개수 -> str
    repetition_penalty=1.4,      # 반복 패널티
    eos_token_id=2,              # 문장 종료 토큰 ID
    num_return_sequences=1,      # 반환할 시퀀스의 수
    use_cache=use_cache,          # 캐시 사용 여부
    #streamer=streamer,
)

generated_text = tokenizer.decode(output[0], skip_special_
```

무엇보다 중요한건 seed 설정.

seed 설정을 어떻게 하느냐에 따라 답변 퀄리티 자체가 많이 달라지는데, seed도 optimization이 필요한 부분

➝ 가장 좋은 접근은 참고한 open source(github) inference 예시를 참조하거나, model structure(llama, gpt-neox or etc.)에 맞게 참조해서 하는 것도 좋은 선택.

5. prompt engineering

굳이 제가 언급하지 않아도 프롬프트 엔지니어링은 매우 중요합니다.

general한 것 말고(one-shot, few-shot 등), 현업에서 의미가 있었던 것들

- 주관식 QA와 score 묶어서 in-context learning ➝ 성능 향상 (주관식/객관식 따로 따로 보다 훨씬 괜찮아진 성능)
- raw data 정제의 필요성(정제되지 않은 현실 데이터들이 너무 제각각인 경우가 있음, data format 문제가 아님) ➝ 이 때 GPT 같은 것을 활용해 raw 데이터를 정제 시켜주니 성능 향상