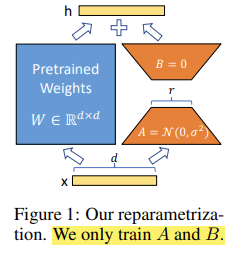
**LoRA 핵심**

기존의 weights 대신 새로운 파라미터를 이용해서

동일한 성능을 더 적은 파라미터로 튜닝할 수 있는 방법론

= LLM은 임베딩을 받아 선형 변환을 하고 임베딩을 뱉는다는 점에 착안해, 이미 사전 학습이 완료된 LLM의 파라미터는 업데이트하지 않고 downstream task에 필요한 파라미터 업데이트, 즉 변화량만 낮은 차원의 bottle-neck으로 계산해 순전파 시 더해주기만 하자

1.메모리 효율성

2.사전 훈련된 가중치 보존

3.훈련된 가중치의 이식성

4.Attention Layer와의 통합

**lora\_config = LoraConfig(**

**r=8,**

**lora\_alpha=16,**

**target\_modules=["q\_proj", "v\_proj"],**

**lora\_dropout=0.05,**

**bias="none",**

**task\_type="CAUSAL\_LM"**

**)**

**model.gradient\_checkpointing\_enable()** # Gradient Checkpointing을 활성화하여 훈련시 메모리를 절약

**model = prepare\_model\_for\_kbit\_training(model)** # k-비트 훈련을 위한 모델 준비

**model = get\_peft\_model(model, lora\_config)** # LoRA 방식으로 모델 래핑

### **LoRA 주요 파라미터**

| parameters | type | definition |
| --- | --- | --- |
| r(rank) | int | * LoRA의 주의(attention) 차원(랭크)을 설정 = original weight matrix를 얼마나 줄일거냐에 대한 계수 * 작을수록 파라미터가 적어져 메모리 효율성⬆️but 표현력 제한됨 * 클 수록 복잡한 패턴 학습가능 but 학습 속도 느려짐 * 기본값: 8, 일반적으로 2,4,6,18 등 사용 |
| target\_modules | list[str]/str | * 어떤 레이어에 LoRa 어댑터를 적용할지를 결정 * 주로 성능에 큰 영향을 주는 핵심 모듈에만 적용 (query, key, value) * 기본값: None * 종류:   + q\_proj (Query Projection Layer)   + v\_proj (Value Projection Layer)   + o\_proj (Output Projection)   + gate\_proj (Gating Projection)   + "up\_proj" (Upsampling Projection)   + "down\_proj" (Downsampling Projection) |
| lora\_alpha | int | * 학습률을 조정하는 파라미터 (스케일링 값: lora\_alpha/r) * 기본값: 8, 일반적으로 r과 동일값 사용 |
| lora\_dropout | float | * LoRA레이어의 드롭아웃 확률 설정해 과적합 방지 * 데이터셋이 클 때 낮게 설정, 작을때 높게 설정 * 기본값: 0.0, 일반적으로 0.1~0.2 |
| fan\_in\_fan\_out | bool | * True: 가중치 행렬이 전치되어야 할 때 * 학습 초기 loss가 정상적으로 감소하지 않는다면 이 파라미터를 확인 |
| bias |  | * 편향 학습 여부 설정 (대부분의 경우 학습x) * 기본값: None |
| task\_type | str | * 수행하려는 작업의 유형을 지정 * 종류: CAUSAL\_LM(텍스트 생성 모델에 사용), |
| use\_dora | bool | * DoRA 사용 여부 (설명은 아래 참고) |

+ QLoRA

기존의 LoRA에 새로운 quantization을 더한 형태 = frozen PLM의 가중치가 '4비트로 양자화'됨

quant\_config = BitsAndBytesConfig(

load\_in\_4bit=True, # 모델 가중치를 4비트로 로드

bnb\_4bit\_quant\_type="nf4", # 양자화 유형으로 nf4 사용한다.

bnb\_4bit\_compute\_dtype=torch\_dtype, # 양자화를 위한 컴퓨팅 타입은 직전에 정의 했던 torch\_dtype으로 지정

bnb\_4bit\_use\_double\_quant=False, # 이중 양자화는 사용하지 않는다.

)

+ DoRA

* Full Fine-Tuning을 진행할 때 일어나는 일을 분석한 결과에서 힌트를 얻어서, 사전 훈련된 가중치를 ‘크기’ 요소와 ‘방향’ 요소로 분리하고, 이 두 부분을 별도로 파인튜닝하는 것
* 가중치의 ‘방향’ 요소를 업데이트하는데 LoRA를 사용해서 효율성을 유지하면서도, 모델의 전반적인 학습 능력을 향상시켜서 결과적인 성능이 Full Fine-Tuning에 가깝게 나오도록 함

[Ref](https://turingpost.co.kr/p/dora-qlora-qdora) [github](https://github.com/huggingface/peft/pull/1474)

**Training 하이퍼파라미터**

**training\_args = TrainingArguments(**

**output\_dir="./results",**

**overwrite\_output\_dir=True,**

**per\_device\_train\_batch\_size=4,**

**gradient\_accumulation\_steps=4,**

**num\_train\_epochs=20,**

**learning\_rate=2e-5,**

**lr\_scheduler\_type="cosine",**

**warmup\_steps=50,**

**weight\_decay=0.03,**

**fp16=True,**

**logging\_steps=10,**

**save\_steps=50,**

**save\_total\_limit=2,**

**dataloader\_num\_workers=2,**

**report\_to="none",**

**max\_steps=200,**

**)**

| **parameters** | **type** | **definition** |
| --- | --- | --- |
| output\_dir=’경로’ | str | * 훈련 결과(모델 가중치 등)를 저장할 경로 설정 |
| learning\_rate | float | * 학습률(파라미터를 얼마나 빠르게 업데이트 할지)를 결정 * 클 수록 빠르게 학습하지만 모델이 불안정해짐 * 기본값: 5e-5 |
| lr\_scheduler |  | * 훈련중 learning rate를 조절 (linear/cosine이 보편적) * Constant: 고정 * linear: 선형, 학습 데이터가 크고 에포크 수가 적을때 잘 작동 * cosine: 코사인형, 긴 학습과정에 효과적 |
| num\_train\_epochs | int | * 전체 학습 데이터셋 반복 횟수 설정 * 기본값: 3 |
| per\_device\_train\_batch\_size/per\_device\_eval\_batch\_size | int | * 각 GPU/CPU에서 사용할 배치 크기 설정 * **클 수록 훈련시간이 짧아지며 모델 성능이 높**아지지만 과적합 가능성 ⬆️ * **작을수록** 훈련시간이 길어지지만 **메모리 효율성 ⬆️** * 기본값: 8, 큰 모델 학습시 주로 4~8로 설정 |
| gradient\_accumulation\_steps | int | * 여러 배치에 계산된 그래디언트를 누적해 실제 가중치 업데이트를 수행할 빈도를 지정 * GPU 메모리 부족할 때 유용 * 기본값: 1, 배치크기가 8이면 4를 주로 사용 |
| otim=’옵티마이저’ | str | * 사용할 옵티마이저 지정 * 기본값: adamw\_hf |
| max\_steps | int | * 모델이 학습할 총 트레이닝 스텝의 수를 지정 * num\_train\_epochs와 같이 사용할 시 둘 중 하나라도 조건이 충족되면 학습을 종료 (예: step이 1000이고 epoch가 3일때 3에포크를 달성하는 순간 학습종료) * 작을수록 훈련시간이 짧아짐 * 기본적으로 1000~2000로 설정해야 충분히 학습 |
| save\_steps | int | * 지정한 주기마다 모델을 기록하고 저장 * 기본값: 500 |
| save\_strategy | str | * 모델 가중치를 저장할 전략을 설정 * “no” : 모델 저장x * “epoch”: 각 에포크 끝날때마다 저장 |
| save\_total\_limit | int | * 최대 몇 개의 체크포인트를 유지할 지 설정 |
| logging\_steps | int | * 지정한 스텝(주기)마다 로그를 기록하고 저장 * 기본값: 500 |
| eval\_steps | int | * 지정한 주기마다 평가를 수행 * 기본값: 500 |
| warmup\_steps | int | * Learning rate를 얼마나 천천히 올릴지 설정 * 너무 짧으면 초기 학습이 불안정, 길면 학습시간 길어짐 * 미세 조정이 필요할 때 유용 * warmup\_ratio와 동시에 설정x * 주로 큰 데이터셋: 500, 작은 데이터셋 50~200로 설정 |
| warmup\_ratio | float | * 학습을 워밍업에 사용할 비율 * 데이터셋 크기가 커도 비율은 일정하게 유지 * ((데이터크기/배치크기)\*에포크수)\*0.1또는 0.06 * 데이터 크기가 크면 0.1 |
| weight\_decay | float | * 모델 파라미터 업데이트시 학습데이터에 과적합 방지 = 모델 복잡도 🔽 * 기본값: 0 |
| fp16=True/False | bool | * 16비트 부동소수점 정밀도를 사용할지 여부 * GPU메모리 사용량 줄이고 학습속도 높이지만 모든 모델에 적용가능한건 x |
| bf16=True/False | bool | * BF16연산 사용할지 여부 지정 * GPU메모리 사용량 줄이고 학습속도 높이지만 모든 모델에 적용가능한건 x |
| max\_grad\_norm | float | * 그래디언트의 폭발을 방지 * 기본값: 1.0 |
| use\_cpu=True/False | bool | * CPU사용해서 훈련할지 여부 설정 * GPU없거나 작은 모델일 때 유용 |
| do\_eval | bool | * 평가를 수행할지 여부를 결정 * 기본값: False |
| load\_best\_model\_at\_end | bool | * 학습이 끝난 후 최고 성능을 보인 모델을 자동으로 load할지 여부를 결정 * 기본값: False (가장 마지막에 저장된 모델이 load됨) |
| metric\_for\_best\_model | optional | * 최고 성능을 판단할 평가 기준을 설정 * accuracy, eval\_loss 등 |
| group\_by\_length | bool | * 입력 시퀀스의 길이에 따라 배치를 그룹화할지 결정 * 메모리 최적화 가능 * 기본값:False |

#### **파라미터 튜닝 가이드:**

1. 학습이 불안정할 때

Learning rate를 줄이고 warmup\_ratio를 늘림

1. 과적합이 발생할 때

Epochs를 줄이고 weight\_decay를 늘림

1. GPU가 작을 때

per\_device\_train\_batch\_size와 gradient\_accumulation\_steps를 줄임

#### **학습 - SFTTrainer**

Trainer vs SFTTrainer

- **Trainer** : 지도 학습 작업에서 모델을 처음부터 훈련시키는 데 사용, 큰 데이터 셋 필요

- **SFTTrainer** : PEFT (Parameter Efficient Fine-Tuning) 와 패킹 최적화와 같은 기술을 사용하여 효율적으로 메모리 사용, 작은 데이터셋과 짧은 훈련 시간으로도 유사하거나 더 나은 정확도를 달성, PLMs Fine-Tuning에 최적화

**trainer = SFTTrainer(**

**model=model,**

**train\_dataset=data,** # 모델 할당 (llama3)

**peft\_config=lora\_config,** # peft 할당 - LoRA

**dataset\_text\_field="data",** # 데이터셋에서 훈련에 사용할 텍스트가 저장된 필드명 지정

**tokenizer=tokenizer,** # 토크나이저 할당

**args=training\_args,** # 하이퍼 파라미터 할당

**max\_seq\_length='NONE',** # 한 문장의 최대 길이를 설정. NONE은 기본값

**packing=False,** # Packing(문장 합치기) 여부. True면 짧은 문장들을 하나로 합쳐서 메모리 효율성을 높이고 / False의 경우 각 문장을 개별적으로 처리

**)**

**model.config.use\_cache = False** # 캐시 설정. Training 시에는 False로 메모리 절약

**trainer.train()**

### **답변 생성 파라미터 (model.generate())**

**PROMPT = '''You are a helpful AI assistant. Please answer the user's questions kindly. 당신은 유능한 AI 어시스턴트 입니다. 사용자의 질문에 대해 친절하게 답변해주세요.'''**

**instruction = '''게임 시작 시 각 플레이어는 몇 장의 카드를 받나요?'''**

# 챗봇 대화 형식 입력 구성

**messages = [**

**{"role": "system", "content": f"{PROMPT}"},**

**{"role": "user", "content": f"{instruction}"}**

**]**

# 메시지를 토큰화하여 모델이 사용할 수 있는 입력 형태로 변환

**input\_ids = tokenizer.apply\_chat\_template(**

**messages,**

**add\_generation\_prompt=True,** # 변환 결과 뒤에 <|assistant|>를 추가해 모델이 그 이후부터 답변을 생성할 수 있도록

**return\_tensors="pt"** # 변환된 입력을 PyTorch 텐서 형태로 반환

**).to(model.device)** # 변환된 input\_ids를 모델이 실행되는 디바이스(GPU 또는 CPU) 에 맞게 이동

# 모델이 답변을 종료할 때 사용해야 할 토큰 ID 목록

**terminators = [**

**Tokenizer.eos\_token\_id,** # 모델의 종료 토큰 <|endoftext|> 을 ID로 변환하여 저장.

**tokenizer.convert\_tokens\_to\_ids("<|eot\_id|>")** # 특정 챗봇 모델의 종료 토큰 <|eot\_id|> 을 ID로 변환하여 저장

**]**

**outputs = model.generate(**

**input\_ids,**

**max\_new\_tokens=256,**

**eos\_token\_id=terminators,**

**do\_sample=True,**

**temperature=0.6,**

**top\_p=0.9,**

**repetition\_penalty = 1.1**

**)**

**print(tokenizer.decode(outputs[0][input\_ids.shape[-1]:], skip\_special\_tokens=True))**

| **parameters** | **type** | **definition** |
| --- | --- | --- |
| max\_new\_token | int | * 답변의 길이를 제한 |
| do\_sample | bool | * 확률적 샘플링을 활성화 * 모델이 항상 가장 높은 확률의 토큰만 선택하는 대신, 다양한 응답을 생성할 수 있게 해줌 |
| temperature | float | * 생성된 텍스트의 무작위성을 조절 * 0에 가까울수록 더 안정적이고 예측 가능한 출력(보수적) * 1에 가까울수록 더 창의적이고 다양한 출력 |
| top\_p | float | * 확률들의 누적합이 특정 값(p)에 도달할 때까지 가장 높은 확률의 단어들만 고려 * 1에 가까울수록 더 많은 단어가 선택 후보가 되어 다양하고 창의적인 출력 * 0에 가까울수록 더 안정적이고 예측 가능한 출력(보수적) |
| repetition\_penalty | float | * 모델이 같은 내용을 반복하는 것을 억제 * 보통 1~1.5미만으로 설정 |
| length\_penalty | float | * 답변 길이의 대한 페널티를 설정 * 1이상으로 설정시 긴 응답을 선호 |
| use\_cache | bool | * 생성 속도를 높임 * 속도는 빨라지지만 메모리 사용량이 증가 * 긴 텍스트를 생성할 때 유용 |

파인튜닝에 필요한 라이브러리

* accelerate
* Peft (QLoRA)
* Bitsandbytes
* Transformers: 다양한 사전 학습된 모델 로드, 텍스트 토크나이징(숫자로 변환), pipeline, Trainer 등
* Trl: 강화학습 기반 언어모델 파인튜닝 기술
* Datasets: 추후 Huggingface에 데이터셋 업로드 후 처리 또는 json파일 업로드한 후 datasets 형식으로 변환하는 용도

모델(Bllossom) :

* <https://blog.naver.com/se2n/223443729640>
* <https://unfinishedgod.netlify.app/2024/05/24/python/?fbclid=IwZXh0bgNhZW0CMTEAAR2-pGRONDIFLylVHBk393oOX0ajUZ17tyu1TVNzJ8pGAANphq4be8svXIg_aem_AaMDSVV7eLmIRquEGZ4hHbxtHmG4PU1BqNCXU1i9zCXdAR9R0Yz3qsIgupYDWbgxVd_crfrrswVpoQYlHdGjUg52>
* <https://huggingface.co/datasets/beomi/KoAlpaca-v1.1a>
* <https://x2bee.tistory.com/278> : training arguments parameter 설명
* <https://chlduswns99.tistory.com/50> : 파인튜닝 예시
* <https://velog.io/@hyeon0321/LoRA%EC%99%80-Quantization%EC%9D%84-%EC%82%AC%EC%9A%A9%ED%95%9C-%ED%95%9C%EA%B5%AD%EC%96%B4-%EA%B8%88%EC%9C%B5-%EB%8D%B0%EC%9D%B4%ED%84%B0%EC%85%8B-%ED%8C%8C%EC%9D%B8%ED%8A%9C%EB%8B%9D> : 금융데이터셋 파인튜닝 예시
* <https://blog.naver.com/se2n/223573762670> : ollama사용해서 GPT생성
* <https://medium.com/@abonia/running-ollama-in-google-colab-free-tier-545609258453> : ollama를 colab에서 돌리는 법