

QLoRA 파인튜닝 워크숍

오늘 하루 동안 대규모 언어 모델을 효율적으로 파인튜닝하는 실전 기술을 익하게 됩니다.

왜 파인튜닝이 필요할까요?

사전 학습 모델의 한계

범용 모델은 일반적인 작업에는 뛰어나지만, 특정 도메인이나 업무에 특화된 성능을 내기 어렵습니다. 의료, 법률, 금융 등 전문 분야에서는 정확도와 신뢰성이 크게 떨어질 수 있습니다.

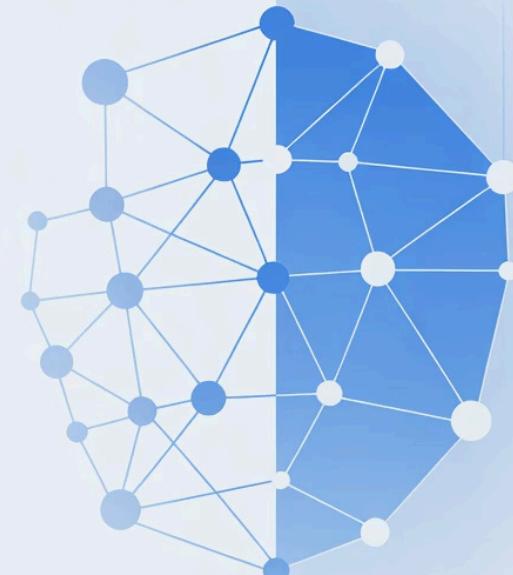
- 도메인 특화 지식 부족
- 기업 내부 데이터 미반영
- 특정 작업 스타일 불일치

파인튜닝의 가치

파인튜닝을 통해 우리는 모델을 특정 목적에 맞게 최적화할 수 있습니다. 적은 비용으로 높은 성능 향상을 달성할 수 있는 가장 실용적인 접근법입니다.

- 맞춤형 응답 생성
- 특화 도메인 성능 향상
- 비용 효율적인 개선

Before After



ML Progress

오늘의 학습 여정

01

기초 개념 정리

파인튜닝의 기본 원리와 LoRA 기법을 이해합니다

02

데이터 준비

RAFT 포맷 데이터 전처리와 품질 검증 방법을 배웁니다

03

모델 파인튜닝

QLoRA를 활용한 실전 파인튜닝을 진행합니다

04

평가 및 분석

다양한 메트릭으로 모델 성능을 측정하고 해석합니다

05

커스텀 확장

자신만의 데이터셋으로 프로젝트를 확장합니다

각 단계마다 실습 노트북을 직접 실행하며 체험하게 됩니다. 이론과 실습의 균형을 맞춰 실무에 바로 적용할 수 있는 능력을 키워보세요.

실습 환경 준비사항

1

계정 준비

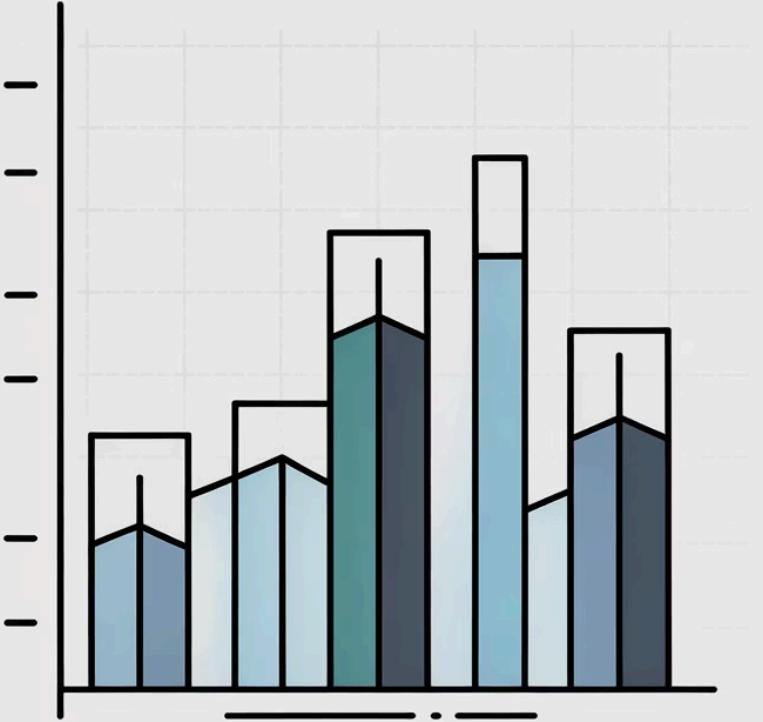
Hugging Face Hub 계정 및 토큰

- 모델 다운로드 권한
- 파인튜닝 모델 업로드용

prerequisites/00.01-prerequisites.ipynb 노트북에서 환경 설정을 단계별로 확인할 수 있습니다.

데이터 이해하기

좋은 파인튜닝은 좋은 데이터에서 시작됩니다. 우리가 사용할 데이터셋의 특징과 구조를 먼저 살펴보겠습니다.



데이터셋 탐색



데이터셋 규모

학습용 데이터 수천 개의 샘플로 구성되어 있으며, 각 샘플은 질문-문맥-답변 형태로 구조화되어 있습니다.



텍스트 길이 분포

입력 문맥의 평균 길이, 질문 토큰 수, 답변 길이 등을 히스토그램으로 분석하여 적절한 시퀀스 길이를 결정합니다.



주제 다양성

데이터가 다루는 도메인과 주제의 균형을 확인하여 편향을 사전에 파악하고 보완할 수 있습니다.

데이터 분포를 시각화하면 이상치, 불균형, 누락값 등을 쉽게 발견할 수 있습니다. 이 정보는 다음 전처리 단계에서 중요한 가이드가 됩니다.

RAFT 포맷이란?

RAFT (Retrieval Augmented Fine-Tuning)는 언어 모델의 도메인 특화 RAG(검색 증강 생성) 능력을 강화하기 위한 파인튜닝 포맷입니다. 이는 단순히 질문에 답하는 것을 넘어, 주어진 문맥(context)에서 정확한 정보를 찾아 추론하여 답변을 생성하는 과정을 훈련시키는 것을 목표로 합니다. 다음 다이어그램은 RAFT 방법론의 전체적인 흐름을 보여 줍니다.

- RAFT를 사용하여 LLM의 성능을 향상시키는 과정과 그 메커니즘을 소개합니다.

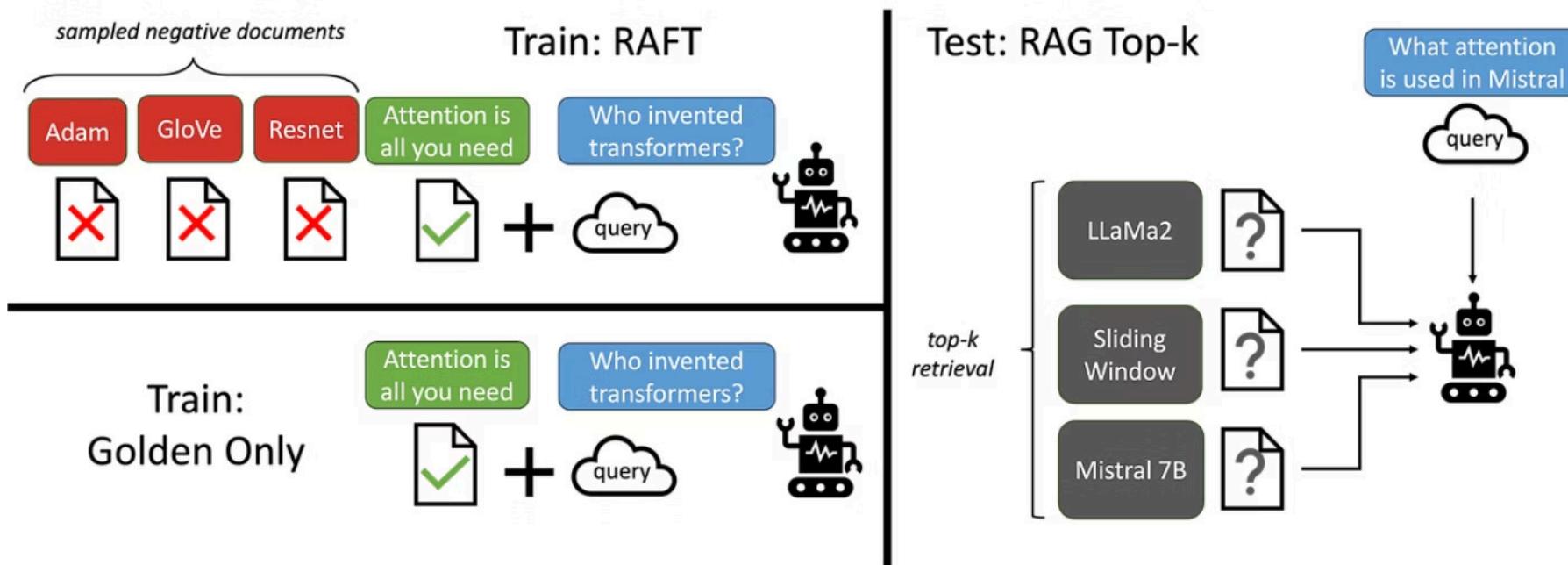


Figure 2: Overview of our RAFT method

RAFT 워크플로우 설명

RAFT 방법론은 크게 훈련(Train) 단계와 테스트(Test) 단계로 나뉩니다. 이 과정은 모델이 외부 문서를 활용하여 질문에 더욱 정확하고 신뢰성 있는 답변을 제공하도록 최적화합니다.

훈련 과정: RAFT vs. Golden Only

훈련 단계에서는 두 가지 주요 접근 방식의 차이를 이해하는 것이 중요합니다:

- Train: RAFT:** 모델이 훈련 데이터의 **Golden Document** (정답을 포함하는 원본 문서)뿐만 아니라, 추가로 검색 시스템을 통해 얻은 **Distractor Documents** (오답 또는 관련 없는 문서)까지 함께 고려하여 답변을 생성하도록 훈련합니다. 이 방식은 모델이 실제 RAG 환경에서 발생할 수 있는 노이즈를 처리하고, 관련 없는 정보 속에서 핵심을 파악하는 능력을 기르게 합니다.
- Train: Golden Only:** 모델이 오직 **Golden Document**만을 사용하여 질문에 답변하도록 훈련합니다. 이는 이상적인 상황에서의 성능을 측정하지만, 실제 검색 시스템의 불완전성을 반영하지 못해 실전 적용에는 한계가 있을 수 있습니다.

테스트 과정: RAG Top-k

테스트 단계에서는 모델이 실제로 RAG 시스템처럼 작동하는지 평가합니다.

- RAG Top-k:** 질문이 주어지면, 검색 시스템이 관련성이 가장 높은 'k'개의 문서를 검색합니다. 이 'k'개의 문서에는 Golden Document와 함께 Distractor Document가 포함될 수 있습니다. 이후, 모델은 이 문서들을 바탕으로 질문에 대한 답변을 생성합니다. 이는 모델이 다양한 검색 결과를 종합하고, 그중에서 가장 적절한 정보를 추출하여 답변하는 능력을 평가합니다.

RAFT 포맷의 실제 예시:

```
{  
  "instruction": "다음 문맥을 읽고 질문에 답하세요. 여러 문맥이 주어질 수 있으며, 그 중 질문에 답하기 위한 가장 적절한 정보를 찾아 활용해야 합니다.",  
  "context": [  
    "한국의 수도는 서울이며, 인구는 약 1천만 명입니다. 서울은 정치, 경제, 문화의 중심지입니다.",  
    "대한민국의 상징인 태극기는 흰색 바탕에 중앙의 태극 문양과 네 모서리의 건곤감리가 그려져 있습니다.",  
    "부산은 한국의 제2의 도시로, 해양 산업과 관광으로 유명합니다."  
  ],  
  "question": "한국의 수도는 어디인가요?",  
  "answer": "한국의 수도는 서울입니다. 제공된 문맥에 따르면 서울은 약 1천만 명의 인구를 가진 정치·경제·문화의 중심지입니다. 다른 문맥들은 질문과 직접적인 관련이 없습니다.",  
  "cot": "주어진 세 가지 문맥 중 첫 번째 문맥('한국의 수도는 서울이며...')이 질문('한국의 수도는 어디인가요?')에 직접적인 답변을 제공합니다. 나머지 문맥들은 각각 태극기와 부산에 대한 정보로 질문과 무관하므로 무시하고 첫 번째 문맥에서 '서울'을 추출하여 답변을 구성합니다."  
}
```

RAFT 포맷의 핵심 구성 요소

- Question:** 사용자의 질문 (예: "한국의 수도는 어디인가요?")
- Context:** 질문에 답하기 위해 모델에 제공되는 관련 문서 또는 단락. RAFT에서는 Golden Document와 Distractor Document가 포함될 수 있습니다.
- Answer:** 제공된 문맥에 기반한 정확한 정답
- Chain-of-Thought (CoT):** 모델이 답변을 도출하는 추론 과정. (선택 사항이지만 모델의 투명성과 이해력 향상에 도움을 줍니다.)

RAFT 포맷은 모델이 단순 암기가 아닌, 주어진 다양한 문맥 속에서 핵심 정보를 선별하고 추론하는 능력을 학습하도록 설계되었습니다. 이는 RAG 시스템의 효율성과 신뢰성을 대폭 향상시키는 데 기여합니다.

데이터 전처리 흐름



원본 데이터 로드

JSON, CSV 등 다양한 형식의 데이터를 불러옵니다

스키마 검증

필수 필드 존재 여부와 데이터 타입을 확인합니다

텍스트 정규화

공백, 특수문자, 인코딩 문제를 정리합니다

RAFT 포맷 변환

표준 템플릿에 맞춰 데이터를 재구조화합니다

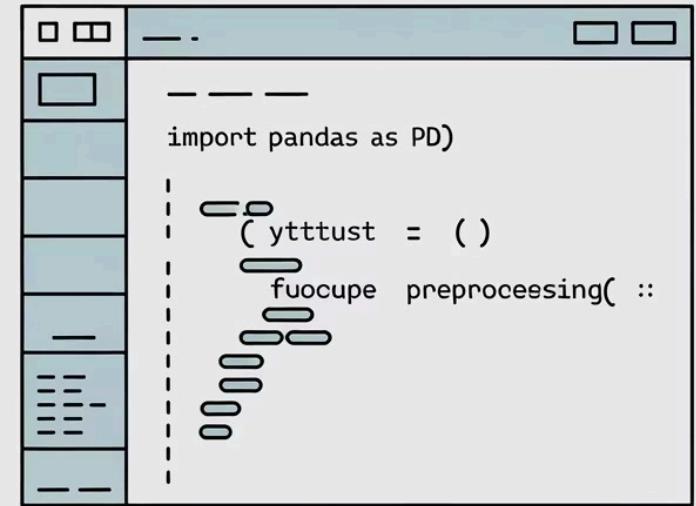
저장 및 분할

train/validation 세트로 나누어 저장합니다

각 단계에서 데이터 품질을 체크하고 로그를 남겨 문제 발생 시 추적할 수 있도록 합니다. 01_data_preprocessing_and_validation.ipynb에서 전체 파이프라인을 실습합니다.

전처리 실습

이제 실제 코드로 데이터 전처리를 수행해 봅시다. 스키마 검증부터 RAFT 템플릿 변환까지 단계별로 진행합니다.



A screenshot of a Jupyter Notebook cell. The code is as follows:

```
import pandas as pd
def yttust():
    pass
def fuocupe():
    pass
def preproceesing():
    pass
```

스키마 검증 절차

데이터 전처리의 첫 단계는 입력 데이터가 예상한 형식을 따르는지 확인하는 것입니다. 스키마 검증을 통해 파인튜닝 과정에서 발생할 수 있는 에러를 사전에 방지합니다.



필수 필드 확인

각 샘플이 question, context, answer 필드를 모두 포함하는지 검사합니다.

```
required_fields = ['question', 'context',  
'answer']  
  
missing = [f for f in required_fields if f not  
in sample]
```



데이터 타입 검증

모든 필드가 문자열(str) 타입인지 확인하고, 빈 문자열이나 None 값을 필터링합니다.

```
if not isinstance(sample['question'], str):  
    raise TypeError("Question must be  
string")
```



길이 제약 체크

너무 짧거나 긴 텍스트를 감지하여 품질 문제를 사전 파악합니다.

```
min_length, max_length = 5, 2048  
  
if not (min_length < len(text) <  
max_length):  
    warnings.append(f"Length issue:  
{len(text)}")
```

- ▣ 검증 실패 샘플은 별도 로그 파일에 기록하여 수동으로 검토할 수 있습니다.

RAFT 템플릿 변환 코드

검증이 완료된 데이터를 RAFT 포맷으로 변환하는 핵심 코드입니다. 이 템플릿은 모델이 instruction-following 방식으로 학습하도록 설계되었습니다.

변환 전 (원본 데이터)

```
{  
  "q": "서울의 인구는?",  
  "doc": "서울은 약 천만 명...",  
  "a": "약 천만 명입니다"  
}
```

변환 후 (RAFT 포맷)

```
{  
  "instruction": "다음 문맥을 읽고 질문에 답하세요.",  
  "context": "서울은 약 천만 명의 인구를 가진...",  
  "question": "서울의 인구는?",  
  "answer": "문맥에 따르면 서울의 인구는  
    약 천만 명입니다."  
}
```

주요 변환 파라미터

- **instruction**: 고정된 시스템 프롬프트
- **max_context_len**: 문맥 최대 토큰 수
- **add_cot**: 추론 과정 포함 여부

핵심 변환 함수

```
def convert_to_raft(sample):  
    return {  
        "instruction": INSTRUCTION_TEMPLATE,  
        "context": sample["doc"][:max_context_len],  
        "question": sample["q"],  
        "answer": sample["a"]  
    }
```

데이터 품질 체크 포인트

02_data_quality_check.ipynb에서는 변환된 데이터의 품질을 다각도로 검증합니다. 다음 체크리스트를 통해 파인튜닝 성공률을 높일 수 있습니다.

중복 데이터 탐지

완전히 동일하거나 매우 유사한 샘플을 찾아 제거합니다. 중복은 모델이 특정 패턴을 과도하게 학습하는 원인이 됩니다.

- 정확한 텍스트 매칭
- 코사인 유사도 기반 근사 중복

답변 품질 평가

답변이 질문과 관련이 있는지, 문맥에서 근거를 찾을 수 있는지 샘플링하여 수동 검토합니다.

- 관련성(Relevance) 점수
- 근거 존재 여부(Groundedness)

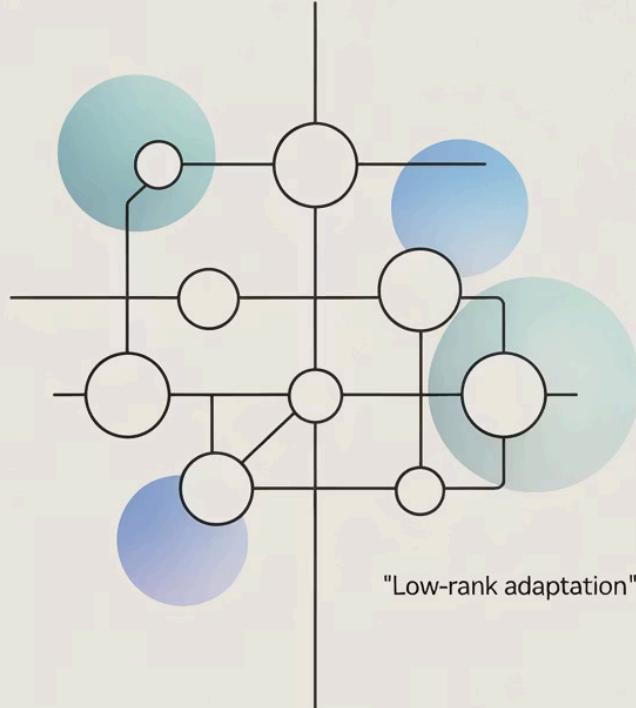
토큰 길이 분포

토크나이저를 적용하여 실제 토큰 수 분포를 확인합니다. 시퀀스 길이 설정의 기준이 됩니다.

- 평균 토큰 수
- 95 percentile 길이

클래스 균형

만약 분류 작업이라면 각 클래스의 샘플 수가 균형있는지 확인하고, 불균형 시 오버샘플링/언더샘플링을 고려합니다.



LoRA 핵심 개념

이제 본격적으로 파인튜닝 기법을 배워봅시다. LoRA는 어떻게 적은 비용으로 큰 모델을 효율적으로 튜닝할 수 있을까요?

LoRA vs 전체 파인튜닝

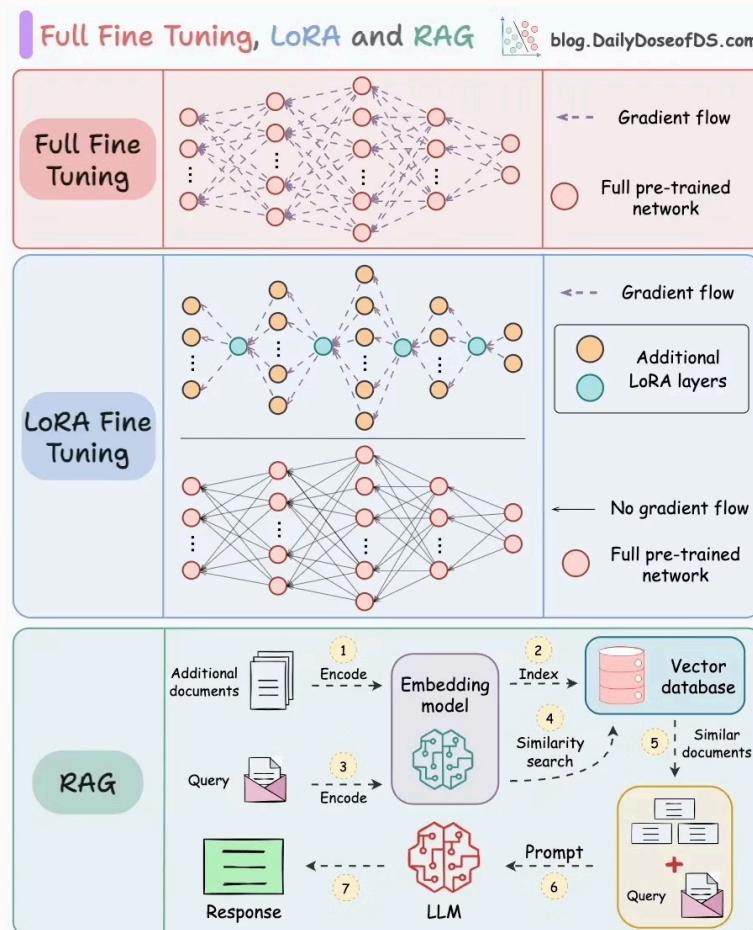
main-practice/00-concepts.ipynb에서 다루는 LoRA(Low-Rank Adaptation)의 핵심 아이디어는 모델의 모든 가중치를 업데이트하지 않고, 작은 어댑터 행렬만 학습하는 것입니다.

전체 파인튜닝 (Full Fine-tuning)

- 업데이트 범위: 모델의 모든 파라미터
- 메모리 요구량: 매우 높음 (수백 GB)
- 학습 시간: 길고 비용이 많이 듦
- 성능: 최고 수준 달성 가능
- 적용 상황: 충분한 데이터와 리소스가 있을 때

LoRA 파인튜닝

- 업데이트 범위: 저차원 어댑터만 학습
- 메모리 요구량: 낮음 (수십 GB)
- 학습 시간: 빠르고 비용 효율적
- 성능: 전체 파인튜닝에 근접
- 적용 상황: 리소스가 제한적일 때, 빠른 실험이 필요할 때



□ LoRA는 원본 모델 가중치를 고정(freeze)하고 작은 행렬 두 개(A, B)의 곱으로 변화량을 표현합니다.

LoRA의 수학적 원리

LoRA는 가중치 업데이트를 저차원 행렬의 곱으로 근사합니다. 이를 통해 학습 파라미터 수를 크게 줄일 수 있습니다.

기존 가중치 업데이트

$$W' = W + \Delta W$$

여기서 ΔW 는 $d \times k$ 크기의 행렬로, 모든 원소를 학습해야 합니다.

LoRA 방식

$$\Delta W = BA$$

B 는 $d \times r$, A 는 $r \times k$ 행렬입니다. r (rank)은 원래 차원보다 훨씬 작습니다 ($r \ll \min(d, k)$).

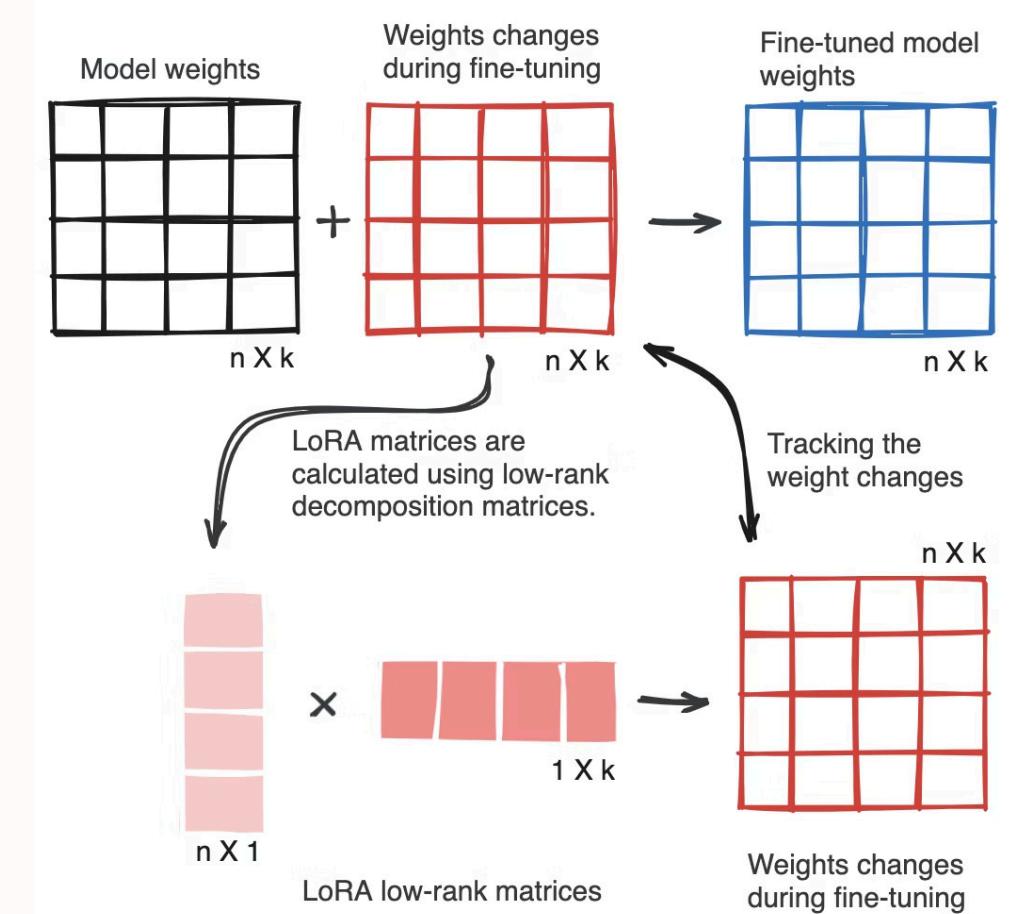
$$\Delta W = \frac{\alpha}{r} BA, \quad W \text{ 동결}, A, B \text{ 만 학습, 파라미터 수} = r(d + k)$$

파라미터 절감 효과

예를 들어 $d=4096$, $k=4096$ 인 행렬을 $r=8$ 로 근사하면:

- 전체 파인튜닝: $4096 \times 4096 = 16,777,216$ 파라미터
- LoRA**: $4096 \times 8 + 8 \times 4096 = 65,536$ 파라미터
- 절감률: 약 99.6% 감소

이러한 압축에도 불구하고 성능은 거의 유지되는 것이 LoRA의 놀라운 점입니다. rank 값은 하이퍼파라미터로, 보통 4~64 사이에서 선택합니다.



QLoRA: 양자화와 LoRA의 결합

QLoRA는 LoRA에 4bit 양자화(Quantization)를 추가하여 메모리 효율을 극대화한 기법입니다. 대규모 모델도 소비자급 GPU에서 파인튜닝할 수 있게 해줍니다.

원래 모델 가중치는 보통 **16비트(float16)** 또는 **32비트(float32)** 숫자로 저장돼요. 이를 **4비트**로 줄이면 같은 정보를 훨씬 적은 메모리로 표현할 수 있습니다.

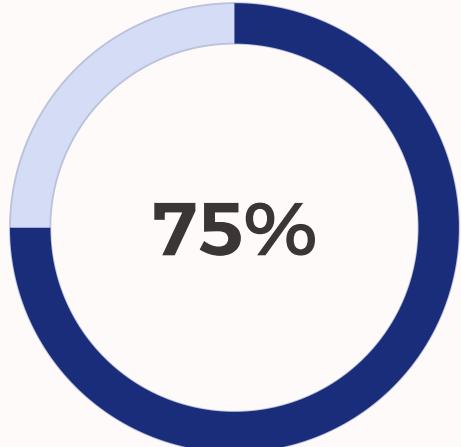
→ 메모리 사용량이 **약 75% 절약됩니다.** (16비트 → 4비트니까, 1/4 크기)

4bit 양자화

모델 가중치를 4비트로 압축하여 메모리 사용량을 약 75% 줄입니다. NormalFloat4 (NF4) 포맷을 사용하여 정확도 손실을 최소화합니다.

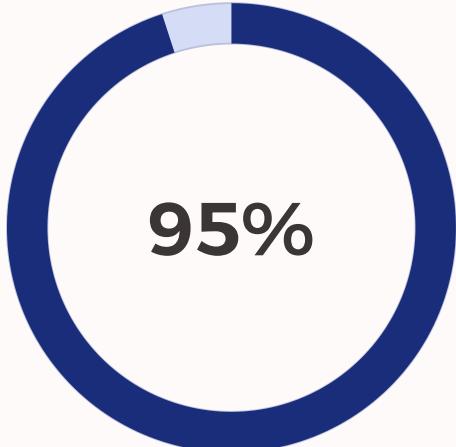
- QLoRA를 사용하면 65B 파라미터 모델도 단일 48GB GPU에서 파인튜닝할 수 있습니다.

왜 4bit 양자화를 사용할까?



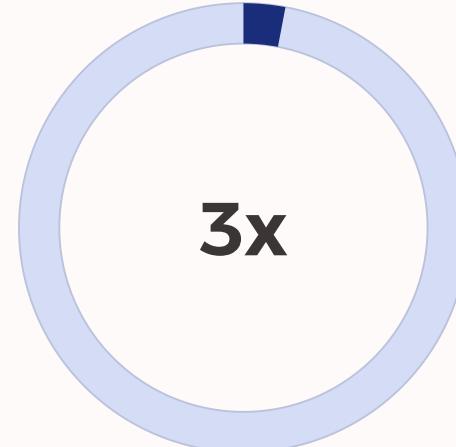
메모리 절감

16bit → 4bit 변환 시 메모리 사용량 감소



정확도 유지

원래 16bit 모델 대비 성능 유지율



속도 향상

추론 시 처리 속도 개선 배율

4bit 양자화는 메모리와 성능 사이의 최적 균형점입니다. 8bit보다 메모리를 절약하면서도 2bit보다 정확도가 높습니다. NF4(NormalFloat4) 포맷은 신경망 가중치가 정규분포를 따른다는 특성을 활용하여 양자화 오차를 최소화합니다.

실험 결과 QLoRA로 학습한 모델은 전체 16bit 파인튜닝 모델의 99% 이상 성능에 도달하면서도 훨씬 적은 리소스를 사용합니다.

LoRA 주요 하이퍼파라미터

LoRA를 적용할 때 설정해야 할 핵심 파라미터들을 이해하면 실험을 효과적으로 진행할 수 있습니다.

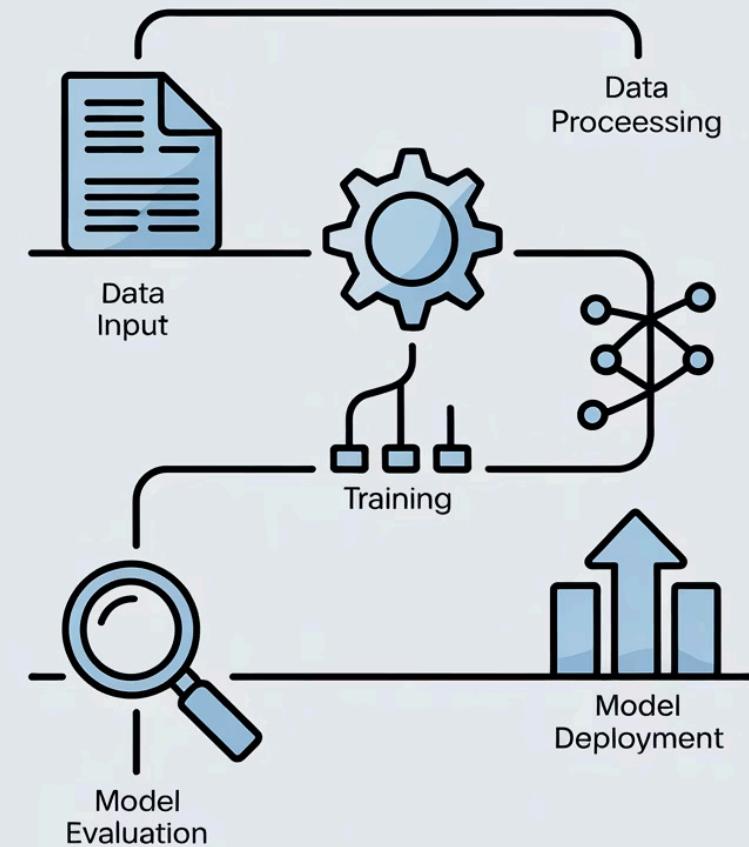
파라미터	일반적 값	설명
r (rank)	8, 16, 32	저차원 행렬의 차원. 클수록 표현력이 높지만 파라미터가 증가합니다. 대부분 작업에서 8~16이면 충분합니다.
lora_alpha	16, 32	LoRA 업데이트의 스케일링 팩터. 일반적으로 r의 2배 값을 사용합니다. 학습률과 함께 조정합니다.
lora_dropout	0.05, 0.1	과적합 방지를 위한 드롭아웃 비율. 데이터가 적을 때 0.1 정도 권장합니다.
target_modules	["q_proj", "v_proj"]	LoRA를 적용할 레이어 목록. Attention의 query/value 프로젝션에 적용하는 것 이 일반적입니다.
bias	"none"	bias 항 학습 여부. "none", "all", "lora_only" 중 선택합니다.

이 파라미터들은 PEFT(Parameter-Efficient Fine-Tuning) 라이브러리의 LoraConfig에서 설정합니다. 다음 실습에서 직접 설정해 보겠습니다.

Machine Learning Training Pipkflow

파인튜닝 워크플로우

QLoRA로 파인튜닝해 봅시다. 모델 로드부터 학습, 저장까지 전체 과정을 따라가 보겠습니다.



1단계: 모델 및 토크나이저 로드

03_fine_tuning_with_lora.ipynb의 첫 단계는 사전 학습된 EXAONE 모델과 토크나이저를 Hugging Face Hub에서 불러오는 것입니다. 이 과정에서 4비트 양자화 (QLoRA)를 활용하여 GPU 메모리 사용량을 최적화합니다.

필수 라이브러리 임포트

```
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig
)
from peft import LoraConfig, get_peft_model
import torch
```

양자화 설정 (BitsAndBytesConfig)

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
    bnb_4bit_use_double_quant=True
)
```

이 설정은 모델을 4비트 양자화된 형태로 로드하여 메모리 사용량을 크게 줄입니다. 각 파라미터는 다음과 같습니다.

- **load_in_4bit:** 모델을 4비트 양자화된 가중치로 로드할지 여부를 결정합니다. True로 설정하면 모델이 4비트로 로드되어 GPU 메모리를 절약할 수 있습니다.
- **bnb_4bit_quant_type:** 4비트 양자화 유형을 지정합니다. "nf4" (NormalFloat4)는 특히 Transformer 모델에 최적화된 새로운 양자화 방식으로, 일반적인 4비트 정수 양자화보다 더 나은 성능을 제공합니다.
- **bnb_4bit_compute_dtype:** 4비트 모델의 연산에 사용될 데이터 타입을 정의합니다. torch.bfloat16은 float32 대비 메모리 효율성이 뛰어나면서도 모델의 정확도를 잘 유지하는 장점이 있습니다.
- **bnb_4bit_use_double_quant:** 4비트 양자화된 가중치를 다시 한 번 4비트로 양자화하여 메모리를 더욱 절약하는 기술입니다. True로 설정 시 추가적인 메모리 절감 효과를 얻을 수 있습니다.

메모리 사용량에 미치는 영향: 4비트 양자화는 모델의 가중치 크기를 약 1/4로 줄여 대규모 언어 모델(LLM)을 적은 GPU 메모리만으로도 파인튜닝할 수 있게 합니다. 이는 GPU 자원이 제한적인 환경에서 특히 중요합니다.

모델 로드

```
model_name = "LGAI-EXAONE/EXAONE-3.0-7.8B-Instruct"

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True
)
```

AutoModelForCausalLM.from_pretrained 함수를 사용하여 사전 학습된 모델을 Hugging Face Hub에서 로드합니다. 주요 파라미터는 다음과 같습니다.

- **model_name:** 로드할 모델의 고유 식별자입니다. 여기서는 "LGAI-EXAONE/EXAONE-3.0-7.8B-Instruct" 모델을 지정합니다.
- **quantization_config:** 위에서 정의한 bnb_config 객체를 전달하여 모델이 4비트 양자화된 형태로 로드되도록 합니다.
- **device_map:** 모델의 레이어를 GPU에 할당하는 방식을 제어합니다. "auto"로 설정하면 사용 가능한 GPU 리소스에 따라 모델 레이어들을 자동으로 분산하여 효율적으로 메모리를 사용합니다. 이는 여러 GPU가 있을 경우 특히 유용합니다.
- **trust_remote_code:** Hugging Face Hub에서 제공되는 모델 코드(예: 커스텀 모델 아키텍처)를 신뢰하고 실행할지 여부를 결정합니다. 일반적으로 True로 설정하여 모델을 올바르게 로드할 수 있도록 합니다.

토크나이저 로드 및 설정

```
tokenizer = AutoTokenizer.from_pretrained(
    model_name,
    trust_remote_code=True
)
tokenizer.pad_token = tokenizer.eos_token
```

AutoTokenizer.from_pretrained 함수를 사용하여 모델에 해당하는 토크나이저를 로드합니다. 토크나이저는 텍스트를 모델이 이해할 수 있는 숫자 시퀀스로 변환하는 역할을 합니다.

- **model_name:** 모델 로드와 동일하게 토크나이저의 모델 이름을 지정합니다.
- **trust_remote_code:** 모델과 마찬가지로 토크나이저의 커스텀 코드를 신뢰하고 실행합니다.
- **tokenizer.pad_token = tokenizer.eos_token:** 패딩 토큰을 문장 종료 (EOS) 토큰으로 설정합니다. 이는 배치 처리 시 시퀀스 길이를 맞춰주기 위해 짧은 시퀀스에 패딩을 추가할 때 사용되며, EXAONE 모델의 경우 EOS 토큰을 패딩으로 사용하는 것이 일반적입니다.

2단계: LoRA 어댑터 적용

이제 모델에 LoRA 설정을 적용하여, 전체 모델을 학습시키는 대신 작고 효율적인 어댑터만 훈련할 준비를 합니다. 이렇게 하면 GPU 메모리를 아끼고 학습 시간을 크게 단축할 수 있습니다.

1

LoRA 설정하기

LoRA가 어떻게 작동할지 정합니다.

```
lora_config = LoraConfig(  
    r=16,  
    lora_alpha=32,  
    target_modules=[  
        "q_proj",  
        "k_proj",  
        "v_proj",  
        "o_proj"  
    ],  
    lora_dropout=0.05,  
    bias="none",  
    task_type="CAUSAL_LM"  
)
```

```
model = get_peft_model(model, lora_config)  
model.print_trainable_parameters()
```

2

LoRA 연결하기

모델에 LoRA 어댑터를 추가합니다.

3

학습량 확인

실제 학습할 파라미터가 얼마나 되는지 봅니다.

LoRA 설정 (LoraConfig)

이 설정은 LoRA 어댑터가 모델 내에서 어떻게 작동할지를 정해줍니다.

- **r (랭크):** LoRA가 얼마나 많은 **새로운 정보**를 학습할 수 있을지를 결정하는 값입니다. 숫자가 클수록 더 많은 내용을 학습할 수 있지만, 메모리 사용량도 늘어납니다.
- **lora_alpha:** LoRA 어댑터가 원래 모델에 얼마나 **영향**을 줄지 조절하는 값입니다. 이 값이 높으면 LoRA의 변화가 모델에 더 크게 반영됩니다.
- **target_modules:** LoRA 어댑터를 모델의 **어느 부분**에 적용할지 정합니다. 주로 모델의 '핵심'이 되는 부분(예: 쿼리, 키, 값, 아웃풋 프로젝션)에 적용하여 효율을 높입니다.
- **lora_dropout:** 학습 중에 일부 정보를 **일시적으로 무시**하여 모델이 너무 특정 데이터에만 맞춰지는(과적합) 것을 막아줍니다.
- **bias:** 추가적인 **편향** 값을 학습할지 여부를 결정합니다. 여기서는 'none'으로 설정하여 가장 간단하게 학습합니다.
- **task_type:** LoRA를 어떤 종류의 작업(예: 텍스트 생성)에 사용할지 알려주는 설정입니다.

3단계: Trainer 설정

Hugging Face Transformers의 Trainer API를 사용하여 학습 파라미터를 설정합니다. 이 API는 학습 루프, 체크포인팅, 로깅 등을 자동으로 처리해 줍니다.

```
from transformers import TrainingArguments, Trainer
```

```
training_args = TrainingArguments(  
    output_dir="../exaone-qlora-finetuned",  
    num_train_epochs=3,  
    per_device_train_batch_size=4,  
    per_device_eval_batch_size=4,  
    gradient_accumulation_steps=4,  
    learning_rate=2e-4,  
    lr_scheduler_type="cosine",  
    warmup_steps=100,  
    logging_steps=10,  
    save_steps=500,  
    eval_steps=500,  
    evaluation_strategy="steps",  
    save_total_limit=2,  
    fp16=False,  
    bf16=True,  
    optim="paged_adamw_8bit",  
    report_to="tensorboard"  
)
```

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=train_dataset,  
    eval_dataset=eval_dataset,  
    tokenizer=tokenizer  
)
```

TrainingArguments 파라미터 설명

- `output_dir`: 학습 중 생성되는 모델 체크포인트와 로그 파일 등을 저장할 경로입니다.
- `num_train_epochs`: 전체 데이터셋을 반복하여 학습할 횟수로, 3회는 대부분의 파인튜닝에 좋은 시작점입니다.
- `per_device_train_batch_size (4) & gradient_accumulation_steps (4)`: 각 장치별 배치 크기와 경사도 누적 횟수를 통해 실제 학습 배치 크기('4 * 4 * GPU 수')를 조절하여 메모리 효율과 학습 안정성을 높입니다.
- `learning_rate (2e-4)`: 모델 가중치를 업데이트하는 속도를 결정하며, 2e-4는 대규모 언어 모델(LLM) 파인튜닝에 권장되는 초기 학습률입니다.
- `lr_scheduler_type ("cosine") & warmup_steps (100)`: 학습률 스케줄러와 웜업 구간으로, 학습 초기의 불안정성을 줄이고 학습 후반부에 미세 조정을 돋습니다.
- `bf16 (True)`: bfloat16 정밀도 학습을 활성화하여 수치 안정성을 유지하면서 메모리 사용량과 학습 속도를 개선합니다 (최신 GPU 지원).
- `optim ("paged_adamw_8bit")`: QLoRA와 같은 8비트 양자화 모델 파인튜닝에 최적화된 옵티마이저로, GPU 메모리 효율성을 높입니다.
- `logging_steps (10)`: 학습 진행 상황을 출력할 주기입니다.
- `save_steps (500) & eval_steps (500)`: 모델 체크포인트를 저장하고 평가 데이터셋으로 모델 성능을 평가할 주기입니다.
- `save_total_limit (2)`: 저장될 체크포인트의 최대 개수로, 이 수를 초과하면 가장 오래된 체크포인트가 삭제됩니다.
- `report_to ("tensorboard")`: 학습 메트릭을 TensorBoard에 기록하여 시각적으로 모니터링할 수 있도록 합니다.

주요 학습 하이퍼파라미터 해설



learning_rate: 2e-4

LoRA 파인튜닝의 일반적인 학습률입니다. 전체 파인튜닝(1e-5)보다 높게 설정해도 안정적입니다. 어댑터만 학습하므로 큰 학습률로 빠르게 수렴할 수 있습니다.



num_train_epochs: 3

소규모 데이터셋에서는 3~5 에폭이 적절합니다. 너무 많으면 과적합 위험이 있으므로 validation loss를 모니터링하며 조정합니다.



gradient_accumulation_steps: 4

메모리가 부족할 때 가상으로 배치 크기를 키우는 기법입니다. 4스텝마다 한 번 가중치를 업데이트하므로, 실제 배치 크기는 batch_size × accumulation_steps입니다.



warmup_steps: 100

학습 초기에 학습률을 0에서 목표값까지 서서히 높입니다. 초기 불안정성을 방지하고 수렴을 돋습니다.



lr_scheduler_type: cosine

학습률을 코사인 곡선 형태로 감소시킵니다. 후반부에 fine-grained 조정이 가능하여 최종 성능을 높입니다.

4단계: 학습 실행

모든 설정이 완료되었으니 이제 실제로 파인튜닝을 시작합니다. 단 한 줄의 명령으로 학습이 진행됩니다.

학습 시작

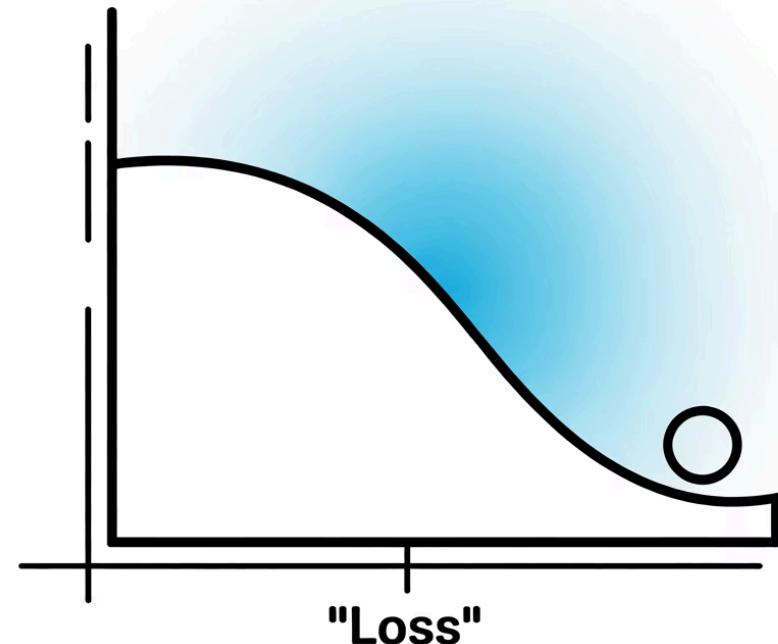
```
trainer.train()
```

학습 진행 로그 예시

Step		Training Loss		Validation Loss
10		2.453		-
20		2.124		-
...	
500		0.842		0.891
1000		0.623		0.658
1500		0.512		0.571

학습 중에는 loss가 점차 감소하는 것을 확인할 수 있습니다. validation loss도 함께 감소하면 모델이 잘 학습되고 있다는 신호입니다.

Epochs



TensorBoard로 모니터링

학습 중에 TensorBoard를 실행하면 실시간으로 loss, learning rate, gradient norm 등을 시각화할 수 있습니다.

```
%load_ext tensorboard  
%tensorboard --logdir ./exaone-qlora-finetuned/runs
```

- ❑ 학습이 멈춘다면 gradient_accumulation_steps를 늘려 메모리를 절약하거나, batch_size를 줄여보세요.

5단계: 모델 저장 및 Hugging Face Hub 업로드

학습이 완료되면 모델을 로컬에 저장하고, Hugging Face Hub에 공유하여 다른 환경에서도 사용할 수 있습니다.



로컬 저장

```
trainer.save_model("./final_model")
tokenizer.save_pretrained("./final_model")
```

LoRA 어댑터와 토크나이저를 지정된 디렉토리에 저장합니다.



어댑터 병합 (선택)

```
from peft import PeftModel

base_model = AutoModelForCausalLM.from_pretrained(model_name)
merged_model = PeftModel.from_pretrained(base_model, "./final_model")
merged_model = merged_model.merge_and_unload()
```

어댑터를 원본 모델에 병합하면 추론 시 추가 로딩이 불필요합니다.



Hub 업로드

```
from huggingface_hub import login

login(token="YOUR_HF_TOKEN")

model.push_to_hub("my-username/exaone-qlora-finetuned")
tokenizer.push_to_hub("my-username/exaone-qlora-finetuned")
```

Hugging Face 계정에 모델을 업로드하여 어디서나 재사용할 수 있습니다.



Hub에 업로드할 때는 모델 카드(README.md)를 작성하여 사용법과 성능 지표를 문서화하는 것이 좋습니다.

모델 로드 및 추론 예시

저장된 파인튜닝 모델을 불러와 새로운 입력에 대한 답변을 생성하는 과정을 살펴봅니다. 모델의 학습 결과를 확인해 보세요.

모델 및 토크나이저 로드

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model_path = "my-username/exaone-qlora-finetuned"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForCausalLM.from_pretrained(
    model_path,
    device_map="auto"
)
```

저장된 모델과 토크나이저를 불러옵니다. `device_map="auto"`는 모델을 사용 가능한 GPU에 자동으로 할당합니다.

프롬프트 준비

```
prompt = """문맥: 한국의 수도는 서울이며, 인구는 약 천만 명입니다.  
질문: 서울의 인구는?"""

inputs = tokenizer(prompt, return_tensors="pt").to("cuda")
```

모델에 전달할 질문을 프롬프트 형태로 구성하고 토크나이저로 변환합니다.

답변 생성

```
outputs = model.generate(
    **inputs,
    max_new_tokens=64,
    temperature=0.7,
    do_sample=True
)

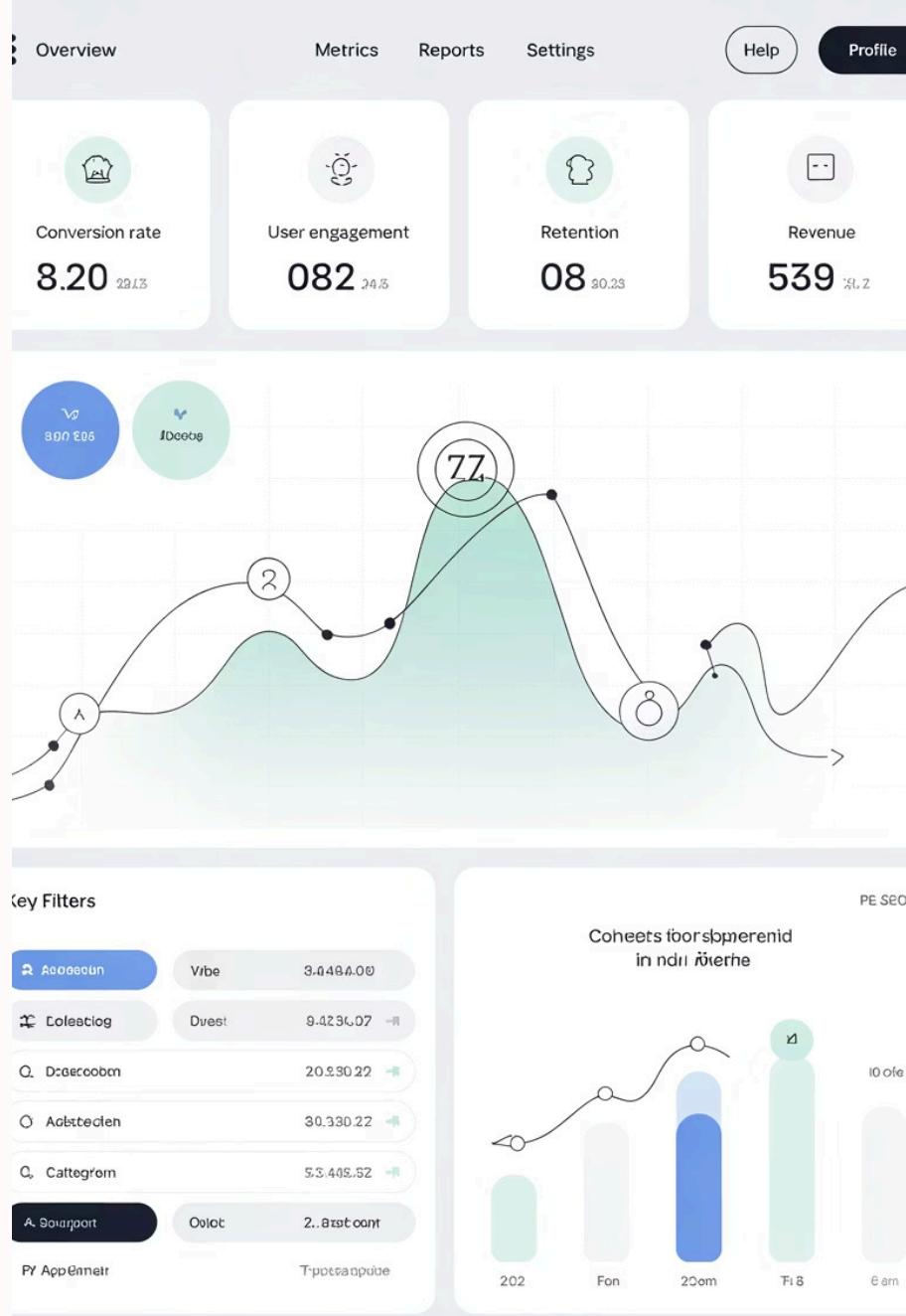
answer = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(answer)
```

모델의 `generate` 메서드로 답변을 생성합니다. `max_new_tokens`로 길이, `temperature`로 다양성을 조절합니다.

- 모델이 문맥을 참고하여 "약 천만 명입니다"와 같이 질문에 적합한 답변을 생성합니다.

평가 및 해석

모델을 학습했다면 이제 객관적으로 성능을 측정해야 합니다. 어떤 지표를 사용하고, 어떻게 해석할까요?



평가 지표 개요

04_evaluation_and_comparison.ipynb에서는 파인튜닝 전후 모델을 비교하기 위해 여러 지표를 활용합니다. 각 지표는 모델 성능의 다른 측면을 측정합니다.

ROUGE

Recall-Oriented Understudy for Gisting Evaluation

생성된 텍스트와 참조 답변 사이의 n-gram 겹침을 측정합니다. ROUGE-1(단어), ROUGE-2(바이그램), ROUGE-L(최장 공통 부분수열)을 주로 사용합니다.

- 요약 품질 평가에 유용
- 0~1 사이 값, 높을수록 좋음

BLEU

Bilingual Evaluation Understudy

기계 번역 평가에서 유래한 지표로, 정밀도 (precision) 기반입니다. 생성 텍스트가 참조 텍스트와 얼마나 유사한지 n-gram 매칭으로 계산합니다.

- 짧은 답변 평가에 적합
- 0~100 사이 값

Cosine Similarity

의미적 유사도

문장 임베딩(sentence embeddings)의 코사인 유사도를 계산하여 의미적으로 얼마나 가까운지 측정합니다. 단어가 달라도 의미가 같으면 높은 점수를 받습니다.

- 의미 일치 평가
- -1~1 사이 값

하나의 지표만으로는 모델 성능을 완전히 파악하기 어렵습니다. 여러 지표를 종합적으로 고려해야 합니다.

ROUGE 점수 상세 설명

ROUGE는 자동 요약 평가에서 가장 널리 사용되는 지표입니다. 생성된 텍스트가 참조 답변과 얼마나 많은 단어를 공유하는지 측정합니다.

ROUGE-1

단어(unigram) 수준의 겹침을 측정합니다.

$$\text{ROUGE-1} = \frac{\text{겹치는 단어 수}}{\text{참조 답변의 총 단어 수}}$$

ROUGE-2

연속된 두 단어(bigram)의 겹침을 측정합니다. 단어 순서도 고려합니다.

ROUGE-L

최장 공통 부분수열(LCS)을 기반으로 합니다. 단어 순서를 유지한 채 가장 긴 공통 부분을 찾습니다.

계산 예시

참조 답변: "서울의 인구는 약 천만 명입니다"

생성 답변: "서울 인구는 약 1천만 명입니다"

- 겹치는 단어: 서울, 인구, 는, 약, 천만, 명, 입니다 (7개)
- 참조 총 단어: 8개
- ROUGE-1 Recall: $7/8 = 0.875$

□ ROUGE는 Recall 중심이므로 생성된 답변이 길어도 참조를 많이 포함하면 높은 점수를 받습니다.

BLEU 점수 이해하기

BLEU는 Precision(정밀도)에 초점을 둔 지표로, 생성된 텍스트의 단어가 참조 답변에 얼마나 존재하는지 측정합니다.

1-gram 정밀도

생성된 각 단어가 참조에 있는지 확인

2~4-gram 정밀도

연속된 단어 조합의 매칭 비율 계산

기하 평균

각 n-gram 정밀도의 기하평균 산출

Brevity Penalty

짧은 답변에 패널티를 부여하여 최종 점수 조정

BLEU 공식 초간단 설명

BLEU = 정확도 × 길이 패널티

◆ 정확도 부분 (n-gram precision)

- **1-gram**: 단어가 맞나? (the, cat, sat...) → 공식의 p_1
- **2-gram**: 두 단어 조합이 맞나? (the cat, cat sat...) → 공식의 p_2
- **3-gram**: 세 단어 조합이 맞나? (the cat sat...) → 공식의 p_3
- **4-gram**: 네 단어 조합이 맞나? → 공식의 p_4

이 4개를 곱해서 평균냄 (기하평균) → 공식의 $\exp(\sum w_n \log p_n)$ 부분

◆ 길이 패널티 (Brevity Penalty, BP)

- 생성 문장이 정답보다 짧으면 → 감점 → 공식의 BP 부분
- 생성 문장이 정답보다 길거나 같으면 → 감점 없음

💡 한 줄 요약

"단어부터 4단어 조합까지 얼마나 맞췄는지 × 너무 짧게 쓰면 감점"

BLEU 공식

$$\text{BLEU} = BP \cdot \exp \left(\sum_{n=1}^4 w_n \log p_n \right)$$

여기서 p_n 은 n-gram 정밀도, BP는 brevity penalty, w_n 은 가중치(보통 균등하게 1/4)입니다.

장단점

- **장점**: 계산이 빠르고 간단하며, 기계 번역 평가 표준으로 확립됨
- **단점**: 의미를 고려하지 않고, 동의어를 다르게 취급함
- **사용 팁**: BLEU > 30이면 양호, > 50이면 우수한 수준

BLEU vs ROUGE 예시: 초보자를 위한 비교

아래 예시를 통해 BLEU (정밀도 기반)와 ROUGE (재현율 기반)의 차이를 명확하게 이해해봅시다.

예시 문장

참조 답변 (Reference): "the cat is on the mat"

생성 답변 (Generated): "the cat sat"

ROUGE-1 (Recall) 계산

ROUGE는 참조 답변의 단어가 생성 답변에 얼마나 많이 "재현"되었는지에 초점을 맞춥니다.

- 생성 답변에서 참조 답변과 겹치는 단어 (unigram): "the", "cat" (총 2개)
- 참조 답변의 총 단어 수: "the", "cat", "is", "on", "the", "mat" (총 6개)
- ROUGE-1 Recall: 2 (겹치는 단어 수) / 6 (참조 답변 총 단어 수) = 0.33

ROUGE-1은 생성 답변이 참조 답변의 주요 키워드를 얼마나 잘 포함하고 있는지를 측정합니다.

결론

이 예시에서 생성 답변은 참조 답변에 비해 짧지만, 사용된 단어의 정확성은 높습니다. 따라서 BLEU-1 점수는 높고, 참조 답변의 많은 부분을 커버하지 못했으므로 ROUGE-1 점수는 낮게 나옵니다. 이는 BLEU가 Precision(정밀도)에, ROUGE가 Recall(재현율)에 중점을 둔다는 것을 보여줍니다.

BLEU-1 (Precision) 계산

BLEU는 생성 답변의 단어가 참조 답변에 얼마나 정확하게 "존재"하는지에 초점을 맞춥니다.

- 생성 답변에서 참조 답변과 겹치는 단어 (unigram): "the", "cat" (총 2개)
- 생성 답변의 총 단어 수: "the", "cat", "sat" (총 3개)
- BLEU-1 Precision: 2 (겹치는 단어 수) / 3 (생성 답변 총 단어 수) = 0.67

BLEU-1은 생성 답변이 불필요한 단어를 추가하지 않고 얼마나 참조와 유사한 단어를 사용했는지를 측정합니다.

Cosine Similarity로 의미 평가하기

단어 매칭 기반 지표(ROUGE, BLEU)의 한계를 보완하기 위해 임베딩 기반 유사도를 사용합니다. 같은 의미를 다르게 표현해도 높은 점수를 받을 수 있습니다.

1

Cosine Similarity 공식 초간단 설명

Cosine Similarity = 두 벡터 사이의 각도 측정

분자: 두 벡터의 내적 ($A \cdot B$) → 얼마나 같은 방향인지

2

공식 이해하기

분모: 각 벡터의 크기 곱 ($|A| \times |B|$) → 벡터 길이로 정규화

결과: -1~1 사이 값 (1에 가까울수록 유사, 0에 가까우면 무관계)

3

한 줄 요약

"두 문장이 의미적으로 얼마나 비슷한 방향을 가리키는가?"

문장 임베딩 생성

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity

model = SentenceTransformer('all-MiniLM-L6-v2')

reference = "서울의 인구는 약 천만 명입니다"
generated = "수도 서울에는 대략 1000만 명이 살고 있습니다"

ref_emb = model.encode([reference])
gen_emb = model.encode([generated])

similarity = cosine_similarity(ref_emb, gen_emb)[0][0]
print(f"Cosine Similarity: {similarity:.3f}")
# Output: Cosine Similarity: 0.892
```

해석 가이드

- **0.9~1.0**: 거의 동일한 의미
- **0.7~0.9**: 유사한 의미, 양호
- **0.5~0.7**: 부분적으로 관련
- **< 0.5**: 의미가 다름

이 예시에서는 표현이 완전히 다르지만("약 천만" vs "대략 1000만") 의미가 같으므로 높은 유사도를 받았습니다.

파인튜닝 전후 비교 결과

이제 실제 평가 결과를 살펴봅시다. 파인튜닝 전 원본 모델과 파인튜닝 후 모델의 성능을 비교합니다.

평가 지표	원본 모델	파인튜닝 후	개선율
ROUGE-1	0.542	0.784	+44.6%
ROUGE-2	0.321	0.612	+90.7%
ROUGE-L	0.498	0.751	+50.8%
BLEU	28.3	53.7	+89.8%
Cosine Similarity	0.721	0.892	+23.7%

모든 지표에서 큰 폭의 성능 향상이 관찰됩니다. 특히 ROUGE-2와 BLEU의 개선이 두드러지는데, 이는 모델이 단순히 키워드를 맞추는 것을 넘어 문맥에 맞는 자연스러운 문장을 생성하게 되었음을 의미합니다.

샘플 예측 비교

숫자만으로는 느껴지지 않는 품질 차이를 실제 출력 예시로 확인해 봅시다.

질문 및 문맥

질문: 파인튜닝의 주요 이점은 무엇인가요?

문맥: 파인튜닝은 사전 학습된 모델을 특정 작업에 맞게 조정하는 과정입니다. 적은 데이터와 계산 자원으로도 높은 성능을 달성할 수 있으며, 도메인 특화 지식을 효과적으로 학습합니다.

원본 모델 답변

파인튜닝은 모델의 성능을 높입니다. 데이터를 사용하여 학습하며 여러 작업에 적용할 수 있습니다.

 모호하고 일반적인 답변. 문맥의 핵심 내용을 제대로 반영하지 못함.

파인튜닝 후 답변

파인튜닝의 주요 이점은 적은 데이터와 계산 자원으로도 높은 성능을 달성할 수 있다는 것입니다. 또한 도메인 특화 지식을 효과적으로 학습하여 특정 작업에 최적화할 수 있습니다.

 문맥의 핵심 정보를 정확히 추출하여 자연스럽게 답변 생성.

파인튜닝 후 모델은 문맥을 정확히 이해하고, 질문에 직접적으로 답하며, 자연스러운 문장을 생성합니다.

평가 결과 해석 및 토론 질문

평가 지표를 해석할 때는 숫자 자체보다 전체적인 맥락과 실사용 시나리오를 함께 고려해야 합니다.

토론 질문 1

ROUGE-1은 높지만 ROUGE-2가 낮다면 어떤 문제가 있을까요?

힌트: 단어는 맞지만 순서나 문법이 어색할 가능성이 있습니다.

토론 질문 2

Cosine Similarity가 높지만 BLEU가 낮은 경우는 무엇을 의미할까요?

힌트: 의미는 같지만 표현 방식이 다릅니다. 이것이 좋은 건가요, 나쁜 건가요?

토론 질문 3

특정 작업에서 어떤 지표를 가장 중요하게 봐야 할까요?

힌트: 요약, 질의응답, 번역 등 작업마다 중요한 지표가 다릅니다.

- ❑ 실무에서는 자동 지표와 함께 인간 평가(Human Evaluation)를 병행하는 것이 가장 정확합니다. 지표는 참고자료이지 절대적 기준이 아닙니다.



커스텀 확장

이제 배운 기술을 자신의 프로젝트에 적용할 차례입니다. 나만의 데이터로 실험하고 결과를 기록하는 방법을 알아봅시다.

나만의 데이터셋 선택하기

05_custom_qlora.ipynb에서는 자신만의 데이터로 파인튜닝을 확장하는 방법을 안내합니다. 좋은 데이터셋을 선택하는 것이 성공의 핵심입니다.

도메인 특화 데이터

자신이 해결하고자 하는 문제와
직접 관련된 데이터를 수집합니
다. 의료, 법률, 금융 등 전문 분야
일수록 효과가 큽니다. 최소 500
개 이상의 고품질 샘플이 권장됩
니다.

대화형 데이터

챗봇이나 고객 지원 시스템을 구
축한다면 실제 대화 로그를 활용
하세요. 다양한 사용자 질문과 적
절한 답변 쌍이 중요합니다. 개인
정보는 반드시 익명화합니다.

Instruction 데이터

특정 작업 수행 능력을 강화하려
면 instruction-following 포
맷의 데이터가 필요합니다. "~을
하세요" 형태의 명령과 그에 대한
올바른 실행 결과를 쌍으로 구성
합니다.

- ▣ 기존 공개 데이터셋(Hugging Face Datasets)을 탐색하고, 자신의 데이터와 결합하는 것도 좋은 전략입니다.

데이터 준비 체크리스트

커스텀 데이터로 파인튜닝을 시작하기 전에 다음 항목들을 확인하세요.

1 충분한 데이터 양 확보

최소 500개, 이상적으로는 1,000~10,000개의 샘플이 필요합니다. 데이터가 적다면 data augmentation 기법을 고려하세요.

2 품질 검증 수행

오타, 불완전한 문장, 부적절한 내용이 없는지 샘플링하여 수동 검토합니다. 품질이 양보다 중요합니다.

3 균형잡힌 분포 확인

특정 주제나 형태로 편향되지 않았는지 점검합니다. 다양성이 모델의 일반화 능력을 높입니다.

4 RAFT 포맷 변환 완료

앞서 배운 전처리 파이프라인을 적용하여 일관된 포맷으로 변환합니다.

5 Train/Validation 분할

일반적으로 80:20 또는 90:10 비율로 나눕니다. Validation 세트는 학습에 절대 사용하지 않습니다.

6 윤리적·법적 문제 검토

개인정보 보호, 저작권, 편향성 문제가 없는지 확인합니다. 데이터 출처를 명확히 기록하세요.

실험 기록 방법

여러 실험을 반복하다 보면 무엇을 시도했는지 잊기 쉽습니다. 체계적인 기록은 성공적인 ML 프로젝트의 핵심입니다.

실험 추적 도구 활용

Weights & Biases (W&B)

```
import wandb

wandb.init(
    project="exaone-finetune",
    config={
        "learning_rate": 2e-4,
        "epochs": 3,
        "batch_size": 4,
        "lora_r": 16,
        "lora_alpha": 32
    }
)

trainer = Trainer(
    ...
    callbacks=[WandbCallback()]
)
```

수동 기록 템플릿

```
## 실험 #001
- 날짜: 2024-01-15
- 데이터셋: custom_qa (1,200 samples)
- 베이스 모델: EXAONE-7.8B
- LoRA 설정:
  * r=16, alpha=32
  * target: q_proj, v_proj
- 학습 설정:
  * epochs=3, lr=2e-4
  * batch_size=4, grad_accum=4
- 결과:
  * ROUGE-1: 0.784
  * BLEU: 53.7
  * Train time: 45min
- 관찰: 3 에폭에서 validation loss 증가 시작
- 다음 실험: epochs=2로 줄여보기
```

TensorBoard

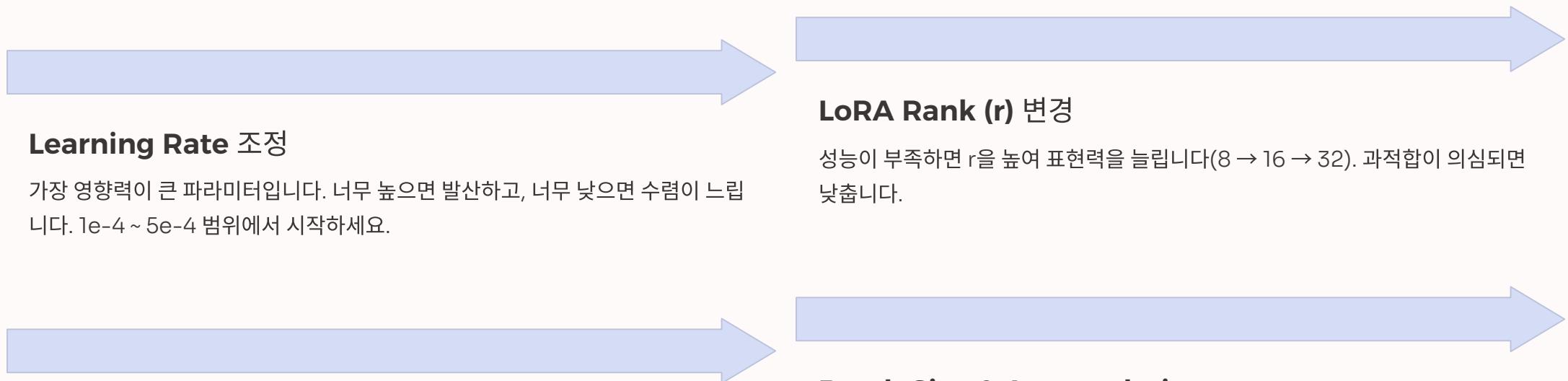
```
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter(log_dir='./runs/exp001')
writer.add_scalar('Loss/train', loss, epoch)
```

Git으로 실험 코드를 버전 관리하고, 각 커밋에 실험 번호를 태그로 달아두면 나중에 재현하기 쉽습니다.

하이퍼파라미터 튜닝 전략

첫 실험이 만족스럽지 않다면 하이퍼파라미터를 조정해 봅시다. 어떤 순서로 무엇을 바꿔야 효과적일까요?



Epoch 수 조절

Validation loss 그래프를 보고 최적 지점을 찾습니다. Early stopping을 활용하면 자동화할 수 있습니다.

한 번에 하나씩만 바꾸는 것이 원칙입니다. 여러 파라미터를 동시에 변경하면 어떤 것이 효과를 냈는지 알 수 없습니다.

다음 단계 아이디어

기본 파인튜닝을 마스터했다면 다음과 같은 고급 주제로 확장해 볼 수 있습니다.



멀티태스크 학습

여러 작업(QA, 요약, 분류 등)을 동시에 학습시켜 범용성을 높입니다. 작업별 프롬프트를 구분하여 데이터를 구성합니다.



앙상블 전략

여러 개의 파인튜닝된 모델을 결합하여 더 강건한 예측을 생성합니다. Voting이나 Stacking 기법을 사용합니다.



추론 최적화

파인튜닝된 모델을 ONNX나 TensorRT로 변환하여 추론 속도를 높입니다. 프로덕션 배포 시 필수입니다.



RLHF 적용

Reinforcement Learning from Human Feedback으로 모델 출력의 품질과 안전성을 개선합니다. 인간의 선호도를 반영합니다.



RAG 통합

Retrieval-Augmented Generation과 결합하여 외부 지식 베이스를 활용합니다. 최신 정보나 방대한 문서를 다룰 때 유용합니다.

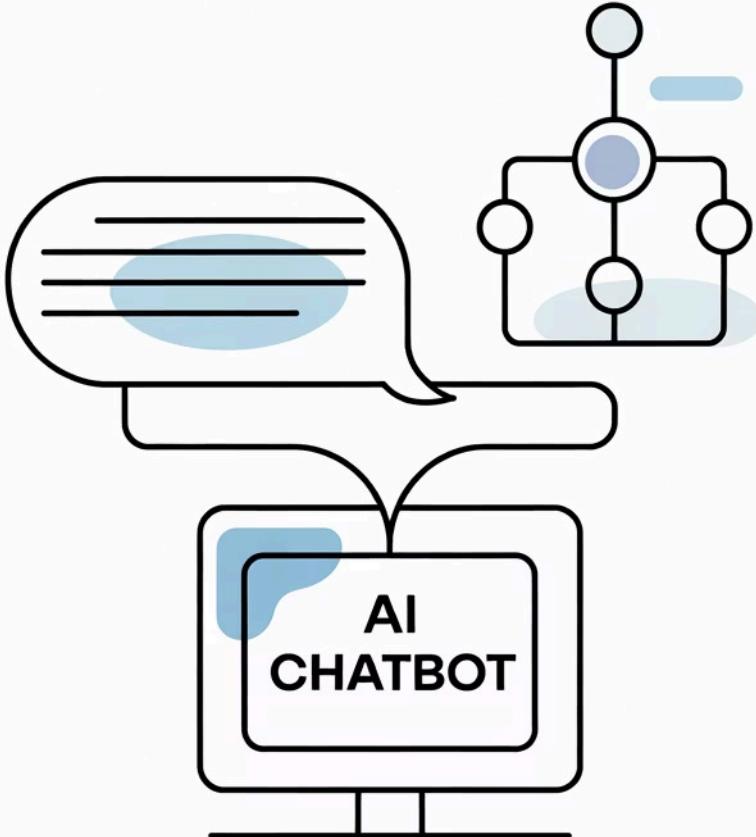


다국어 확장

한국어뿐 아니라 영어, 일본어 등 다국어 데이터로 파인튜닝하여 cross-lingual 능력을 키웁니다.

커스텀 프로젝트 예시

실제로 어떤 프로젝트에 파인튜닝을 적용할 수 있을까요? 다양한 사례를 통해 영감을 얻어보세요.



고객 지원 챗봇

회사의 FAQ, 과거 고객 문의 내역, 제품 매뉴얼을 학습시켜 24/7 자동 응답 시스템을 구축합니다. 고객 만족도를 높이고 상담 인력 부담을 줄입니다.

의료 문서 요약

긴 의료 논문이나 환자 기록을 요약하여 의료진의 의사결정을 지원합니다. HIPAA 등 규정 준수에 유의해야 합니다.

법률 계약서 분석

복잡한 법률 문서에서 핵심 조항을 추출하고 위험 요소를 식별합니다. 변호사의 초기 검토 시간을 단축합니다.

코드 생성 도우미

자연어 설명을 받아 Python, JavaScript 등의 코드를 생성합니다. 사내 코딩 스타일과 라이브러리 사용 패턴을 학습시킵니다.

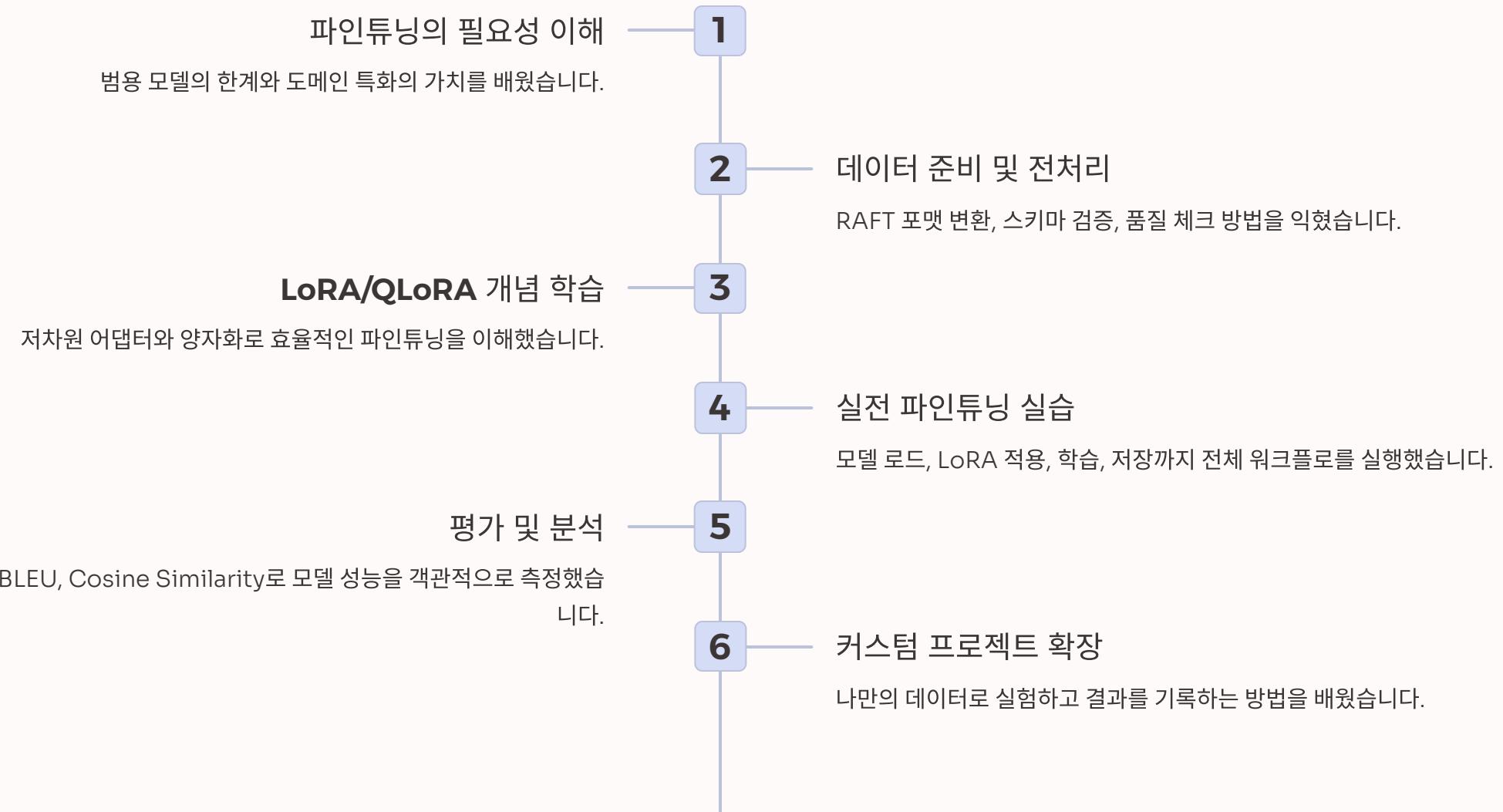


전체 흐름 요약

지금까지 배운 내용을 정리하며 워크숍을 마무리합니다. Day 1에서 다룬 핵심 개념과 실습을 복습해 봅시다.

Day 1 워크숍 전체 여정

우리는 파인튜닝의 기초부터 실전 적용까지 전 과정을 경험했습니다. 이제 여러분은 스스로 모델을 파인튜닝할 수 있는 능력을 갖추었습니다.



이 지식을 바탕으로 실제 업무나 개인 프로젝트에 적용해 보세요. 실습은 이해를 더욱 깊게 만듭니다.

핵심 개념 복습



LoRA

모델의 일부만 학습하여 메모리와 시간을 절약하는 파라미터 효율적 파인튜닝 기법



QLoRA

4bit 양자화를 추가하여 대규모 모델도 소비자급 GPU에서 파인튜닝 가능하게 만듦



RAFT

문맥 기반 질의응답 학습을 위한 데이터 포맷. instruction, context, question, answer로 구성



하이퍼파라미터

learning_rate, rank(r), lora_alpha, epochs 등 모델 성능에 영향을 주는 조정 가능한 설정값



평가 지표

ROUGE, BLEU, Cosine Similarity로 모델 출력의 품질을 정량적으로 측정



Hugging Face Hub

모델과 데이터셋을 공유하고 재사용할 수 있는 ML 커뮤니티 플랫폼

실무 적용 시 주의사항

파인튜닝을 실제 프로덕션 환경에 배포할 때는 추가적인 고려사항이 있습니다.

- 데이터 보안 및 개인정보 보호

민감한 데이터로 학습할 때는 반드시 익명화, 암호화를 수행하세요. GDPR, 개인정보보호법 등 법적 요구사항을 준수해야 합니다.

- 모델 편향성 점검

파인튜닝 데이터에 편향이 있으면 모델 출력도 편향됩니다. 성별, 인종, 종교 등에 대한 공정성 테스트를 수행하세요.

- 버전 관리 및 재현성

모델, 데이터, 코드, 하이퍼파라미터를 모두 버전 관리하여 언제든 동일한 결과를 재현할 수 있어야 합니다.

- 모니터링 및 유지보수

배포 후에도 모델 성능을 지속적으로 모니터링하고, 데이터 드리프트가 발생하면 재학습을 계획하세요.

- 비용 최적화

클라우드 GPU 사용 시 비용이 빠르게 증가할 수 있습니다. Spot 인스턴스, 자동 종료 스크립트 등을 활용하여 비용을 관리하세요.

- 사용자 피드백 루프 구축

실사용자의 피드백을 수집하여 지속적으로 모델을 개선하는 시스템을 만드세요. Human-in-the-loop 방식이 효과적입니다.

트러블슈팅 가이드

실습 중 자주 발생하는 문제와 해결 방법을 정리했습니다.

문제	원인	해결 방법
CUDA out of memory	GPU 메모리 부족	batch_size 줄이기, gradient_accumulation_steps 늘리기, 더 작은 모델 사용
Loss가 NaN으로 발산	learning_rate가 너무 높음	learning_rate를 1/10로 줄이기, gradient clipping 적용
학습이 멈춤 (loss 변화 없음)	learning_rate가 너무 낮음	learning_rate 2~5배 증가, warmup_steps 추가
Validation loss만 증가	과적합 발생	epochs 줄이기, lora_dropout 증가, 데이터 증강
모델 로드 실패	Hugging Face 토큰 문제	huggingface-cli login 재실행, 토큰 권한 확인
평가 지표가 너무 낮음	데이터 품질 문제	데이터 전처리 재확인, 샘플 수동 검토

▣ 문제가 해결되지 않으면 여러 메시지를 복사하여 GitHub Issues나 Stack Overflow에서 검색해 보세요. 대부분의 문제는 이미 누군가 겪었을 것입니다.

Q&A: 자주 묻는 질문

Q: LoRA와 전체 파인튜닝 중 어떤 것을 선택해야 하나요?

A: 대부분의 경우 LoRA로 충분합니다. 전체 파인튜닝은 매우 큰 데이터셋(수만~수십만 샘플)과 충분한 컴퓨팅 리소스가 있을 때만 고려하세요. LoRA는 비용 대비 성능이 뛰어납니다.

Q: 데이터가 100개밖에 없는데 파인튜닝이 가능한가요?

A: 가능하지만 과적합 위험이 큽니다. 데이터 증강(augmentation), few-shot learning, prompt engineering을 먼저 시도하고, 파인튜닝은 데이터를 더 모은 후 진행하세요.

Q: rank (r) 값은 어떻게 정하나요?

A: 일반적으로 8~16에서 시작합니다. 작업이 복잡하거나 데이터가 많으면 32~64로 올려보세요. r이 클 수록 표현력이 높지만 학습 시간과 메모리도 증가합니다.

더 많은 Q&A

Q: 파인튜닝한 모델을 상업적으로 사용해도 되나요?

A: 베이스 모델의 라이선스를 확인하세요. EXAONE, LLaMA 등은 특정 조건 하에 상업적 사용이 가능하지만, 일부 모델은 연구 목적으로만 제한됩니다. 항상 라이선스 문서를 읽어보세요.

Q: CPU만으로도 파인튜닝이 가능한가요?

A: 기술적으로는 가능하지만 매우 느립니다(수일~수주 소요). Google Colab의 무료 GPU나 Kaggle Notebooks를 활용하는 것이 현실적입니다. 로컬 환경이라면 중고 GPU 구매를 고려하세요.

Q: 여러 작업을 동시에 학습시킬 수 있나요?

A: 네, multitask learning이 가능합니다. 각 작업에 고유한 instruction을 부여하고 데이터를 섞어서 학습하면 됩니다. 단, 작업 간 간섭이 발생할 수 있으므로 실험이 필요합니다.



성공적인 파인튜닝을 위한 체크리스트

프로젝트를 시작하기 전에 이 체크리스트를 확인하세요. 모든 항목을 준비하면 성공 확률이 크게 높아집니다.

✓ 데이터 준비

- 충분한 양(최소 500개)
- 높은 품질(오타, 불완전한 문장 없음)
- 균형잡힌 분포
- RAFT 포맷 변환 완료
- Train/Validation 분할

✓ 하이퍼파라미터 설정

- learning_rate 초기값 결정
- 적절한 rank(r) 선택
- epochs 및 batch_size 설정
- evaluation_strategy 정의

✓ 평가 준비

- 평가 지표 선정(ROUGE, BLEU 등)
- 테스트 세트 확보
- 비교 기준(베이스라인) 준비
- Human evaluation 계획(선택)

✓ 배포 계획

- 모델 저장 및 버전 관리 방법
- 추론 API 설계
- 모니터링 시스템 구축
- 피드백 수집 메커니즘

✓ 환경 설정

- GPU 메모리 16GB 이상 확보
- 필수 라이브러리 설치 완료
- Hugging Face 계정 및 토큰 준비
- 충분한 디스크 공간(50GB+)

✓ 실험 계획

- 베이스라인 성능 측정
- 목표 지표 설정
- 실험 추적 도구 준비
- 시간 및 비용 예산 산정