모듈 불러오기 + 데이터셋

**라이브러리 설치**

**!pip install --upgrade pip**

**!pip install -U accelerate==0.29.3 peft==0.10.0 bitsandbytes==0.43.1 transformers==4.40.0 trl==0.8.6 datasets==2.19.0**

**모듈 import**

**import torch**

**from datasets import load\_dataset**

**from transformers import (BitsAndBytesConfig,**

**AutoTokenizer,**

**AutoModelForCausalLM,**

**TrainingArguments)**

**from peft import (LoraConfig,**

**get\_peft\_model,**

**prepare\_model\_for\_kbit\_training)**

**from trl import SFTTrainer**

**모델 가져오기**

**import huggingface\_hub**

**huggingface\_hub.login('hf\_ikmwWunAbXOJZXMOQSoUNvYiBLKzyiXHrB')**

**base\_model = 'MLP-KTLim/llama-3-Korean-Bllossom-8B'**

>> llama3 한국어 파인튜닝된 모델 사용 - Bllossom

—------------------------------------------------------------------------------------------------------------------------------------

**토크나이저 로드**

**tokenizer = AutoTokenizer.from\_pretrained(base\_model) # base model에 맞는 토크나이저 로드**

**tokenizer.pad\_token = tokenizer.eos\_token**

**padding(pad) 토큰**은 입력 시퀀스의 길이를 맞추기 위해 추가하는 토큰. 긴 문장을 짧은 문장과 맞추기 위해 사용

**eos 토큰**은 end-of-sequence(시퀀스 종료)를 나타내는 토큰. 문장의 끝을 모델에게 알리는 역할

**=> "<EOS>"를 패딩 및 시퀀스 종료 역할로 사용**

**데이터셋 로드**

**import json**

**import pandas as pd**

**with open('bang\_questions.json', 'r', encoding='utf-8') as f:**

**data = json.load(f)**

<Dataset> 객체 형태로 변환된 데이터셋 사용해야함

**from datasets import Dataset**

**dataset = pd.DataFrame(data) # list 형태의 dict인 경우**

\*\* Dict 형태 List 데이터셋인 경우 dict 로 -> Dataset으로 변경하는 과정 필요

**# pandas DataFrame-> dictionary 리스트**

**data\_dict = {**

**'question': [item['question'] for item in data],**

**'answer': [item['answer'] for item in data]**

**}**

**# dict -> Dataset 형식으로 변환**

**dataset = Dataset.from\_dict(data\_dict)**

**데이터셋 전처리**

Q&A 데이터를 챗봇 학습용 데이터 형식으로 변환 ( 모델 입력으로 사용 가능하게 만들기 )

**def preprocess(data):**

**return data.map( # 대화 형식으로 변환**

**lambda x: {**

**'data': tokenizer.apply\_chat\_template(**

**[{"role": "system", "content": x['question']},**

**{"role": "assistant", "content": x['answer']}],**

**add\_generation\_prompt=False, tokenize=False, return\_tensors="pt"**

**)**

**}**

**).map(lambda samples: tokenizer(samples["data"]), batched=True) # 토큰화**

**data = preprocess(dataset)**

1. ***apply\_chat\_template()*** 로 대화 형식 데이터 생성

*role : “system”*으로 질문 -> dataset의 *‘question’*

*role : “assistant”*로 답변 -> dataset의 *‘answer’* 로 설정

2. ***tokenizer.apply\_chat\_template()*** 적용

hugging face 토크나이저로 대화 형식 텍스트를 만들지만, *tokenize = False*로 토큰화는 안함.

*return\_tensors = ‘pt’* 로 PyTorch 텐서로 반환

3. **첫 번째 *.map()*** : 각 샘플을 대화 형식으로 가공

dataset의 각 샘플(x)을 { “data” : formatted\_text } 형태로 변환

4. **두 번째 *.map()*** : 최종 **토큰화**

tokenizer(samples[“data”])를 실행해 토큰화.

batched=True로 한 번에 여러 샘플 변환하여 속도 향상

/예시



**양자화**

4비트 양자화로 메모리 효율성을 높임.

**bnb\_config = BitsAndBytesConfig(**

**load\_in\_4bit=True, # 모델의 가중치를 4비트 정밀도로 양자화**

**bnb\_4bit\_compute\_dtype=torch.float16, # 16비트 부동소수점 형식으로 계산을 수행**

**bnb\_4bit\_use\_double\_quant=True, # 이중 양자화 옵션**

**bnb\_4bit\_quant\_type="nf4" # 양자화 방식 - nf4**

**)**

**model = AutoModelForCausalLM.from\_pretrained(**

**base\_model,**

**quantization\_config = bnb\_config, # 위에서 설정한 양자화**

**device\_map = 'auto', # 자동으로 모델이 여러 GPU에 분산되어 로딩**

**low\_cpu\_mem\_usage=True # CPU 메모리 사용량을 최소화하는 옵션**

**)**