7장. 모델 가볍게 만들기

LLM을 활용한 실전 AI 애플리케이션 개발

실습 발표: 니콜

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ 필요한 라이브러리 import

```
!pip install transformers==4.40.1 accelerate==0.30.0 bitsandbytes==0.43.1 auto-gptq==0.7.1 autoawq==0.2.5 optimum==1.19.1 -qqq
```

```
import transformers
import accelerate
import bitsandbytes
import auto_gptq
import awq
```

- BitsAndBytes, GPTQ, AWQ 양자화 방식을 구현하는 데 필요한 라이브러리

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ BitsAndBytes 양자화 모델 불러오기

```python
from transformers import AutoModelForCausalLM, BitsAndBytesConfig

# 8비트 양자화 모델 불러오기
bnb_config_8bit = BitsAndBytesConfig(load_in_8bit=True)
model_8bit = AutoModelForCausalLM.from_pretrained("facebook/opt-350m", quantization_config=bnb_config_8bit)

# 4비트 양자화 모델 불러오기
bnb_config_4bit = BitsAndBytesConfig(load_in_4bit=True,
                                     bnb_4bit_quant_type="nf4")

model_4bit = AutoModelForCausalLM.from_pretrained("facebook/opt-350m",
                                                  low_cpu_mem_usage=True,
                                                  quantization_config=bnb_config_4bit)
```

- 모델을 불러올 때 양자화 설정을 전달
  - 8비트 양자화: load-in_8bit, bnb_8bit_quant_type 설정
  - 4비트 양자화: load-in_4bit, bnb_4bit_quant_type 설정

- 모델을 불러오면서 바로 양자화 가능

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ BitsAndBytes 양자화 모델 비교하기

```python
import time

def evaluate_model_performance(model, input_text):
    start_time = time.time()
    inputs = tokenizer(input_text, return_tensors="pt")
    outputs = model.generate(**inputs)
    end_time = time.time()
    return end_time - start_time

input_text = "This is a test sentence."

# 모델 평가
time_8bit = evaluate_model_performance(model_8bit, input_text)
time_4bit = evaluate_model_performance(model_4bit, input_text)

print(f"8비트 모델 추론 시간: {time_8bit:.4f}초")
print(f"4비트 모델 추론 시간: {time_4bit:.4f}초")
```

```
8비트 모델 추론 시간: 0.8124초
4비트 모델 추론 시간: 0.5129초
```

- 추론 시간 비교:
  동일한 텍스트를 입력하여 두 모델의 예측 시간을 비교
- 4비트 모델이 8비트 모델보다 약 37% 더 빠르게 추론을 수행함
- 4비트 모델은 메모리와 연산 자원을 더 적게 사용하기 때문에 대체로 처리 속도가 빠르지만, 그만큼 데이터 표현 범위가 줄어들어 성능 면에서 약간의 정확도 손실이 발생
- 따라서 속도와 메모리 효율을 우선시한다면 4비트 양자화 모델이 적합

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ GPTQ 양자화 수행 코드

```python
from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig

model_id = "facebook/opt-125m"
tokenizer = AutoTokenizer.from_pretrained(model_id)
quantization_config = GPTQConfig(bits=4, dataset = "c4", tokenizer=tokenizer)

model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto", quantization_config=quantization_config)
```

- 양자화 설정을 전달
  - GPTQConfig 클래스에 bits, dataset, tokenizer를 설정

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ GPTQ 양자화된 모델 불러오기

```python
from transformers import AutoModelForCausalLM
model = AutoModelForCausalLM.from_pretrained("TheBloke/zephyr-7B-beta-GPTQ",
                                             device_map="auto",
                                             trust_remote_code=False,
                                             revision="main")
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ AWQ 양자화 모델 불러오기

```python
from awq import AutoAWQForCausalLM
from transformers import AutoTokenizer

model_name_or_path = "TheBloke/zephyr-7B-beta-AWQ"
tokenizer = AutoTokenizer.from_pretrained(model_name_or_path, trust_remote_code=False)
model = AutoAWQForCausalLM.from_quantized(model_name_or_path, fuse_layers=True, trust_remote_code=False, safetensors=True)
```

- AWQ는 모델의 활성화 값 분포를 통해 중요한 파라미터를 결정하고 양자화를 수행
- from_quantized는 양자화된 모델을 불러옴
- trust_remote_code를 False로 보안 설정을 하고, safetensors=True로 안전한 텐서 형식을 사용합니다.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

감사합니다