[실습] 언어모델과 자연어처리 I

실습 제목

“다음 단어 확률과 Self-Attention을 직접 열어보기: 프롬프트 한 줄이 예측을 바꾼다”

실습 목표

1. 언어모델은 **“다음 토큰 확률분포”**를 만든다(Top-k 후보를 직접 확인).
2. 같은 문장이라도 **앞 문맥(프롬프트)**이 바뀌면 **다음 토큰 후보가 크게 바뀐다.**
3. 그 변화는 Transformer의 **Self-Attention이 “어떤 단어를 참고했는지”**로 일부 설명할 수 있다.

준비물(영상 자막/설명용)

- Google Colab(권장) 또는 로컬 파이썬
- `transformers`, `torch`
- 모델: `skt/kogpt2-base-v2` (한국어 예시가 잘 먹음)

실습 코드

> “코드를 다 이해하려고 하기보다, **출력 결과가 말해주는 것**을 보자”

```
# 1) 설치 (Colab 기준)
!pip -q install transformers torch

import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, AutoConfig

device = "cuda" if torch.cuda.is_available() else "cpu"
model_name = "skt/kogpt2-base-v2"

tokenizer = AutoTokenizer.from_pretrained(model_name)

# 모델 설정에서 attention 출력을 활성화
config = AutoConfig.from_pretrained(model_name, output_attentions=True)
model = AutoModelForCausalLM.from_pretrained(model_name, config=config).to(device)
model.eval()

# GPT류는 종종 pad_token이 없어서 설정해줌(없으면 generate에서 경고/에러 가능)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token


def topk_next_tokens(prompt: str, k: int = 5):
    inputs = tokenizer(prompt, return_tensors="pt").to(device)
```

```
    with torch.no_grad():
        out = model(**inputs)  # logits: (B, T, V)

    logits = out.logits[0, -1, :]                 # 마지막 위치에서 "다음 토큰" 예측
    probs = torch.softmax(logits, dim=-1)
    top = torch.topk(probs, k)

    print(f"\nPROMPT: {prompt}")
    print(f"Top-{k} next tokens:")
    for p, idx in zip(top.values, top.indices):
        tok = tokenizer.decode([idx.item()])
        print(f"  {tok!r:>12}  prob={p.item():.4f}")


def attention_peek(prompt: str, layer: int = -1, head: int = 0, topn: int = 8):
    """
    '마지막 입력 토큰'이 이전 토큰들을 얼마나 참고했는지(attention)를 상위 topn개만 보여줌.
    """
    inputs = tokenizer(prompt, return_tensors="pt").to(device)

    with torch.no_grad():
        out = model(**inputs, output_attentions=True, return_dict=True)

    # attentions: (num_layers, B, num_heads, T, T)
    att = out.attentions[layer][0, head, -1, :]   # 마지막 토큰 쿼리 -> 전체 토큰 키로의 가중치
    att = att.detach().cpu()

    ids = inputs["input_ids"][0].detach().cpu().tolist()
    toks = [tokenizer.decode([i]) for i in ids]

    pairs = list(enumerate(att.tolist()))
    pairs.sort(key=lambda x: x[1], reverse=True)

    print(f"\n[Attention peek] layer={layer}, head={head}")
    print("Prompt tokens:")
    print("".join(toks))

    print(f"\nTop-{topn} attended previous tokens (for the LAST input token):")
    for idx, w in pairs[:topn]:
        print(f"  pos={idx:>2} tok={toks[idx]!r:>10}  weight={w:.4f}")


# ===== 실습 1) 다음 토큰 후보 확인 =====
topk_next_tokens("대한민국의 수도는", k=8)

# ===== 실습 2) 문맥을 바꿔서 후보가 어떻게 달라지는지 비교 =====
topk_next_tokens("퀴즈: 대한민국의 수도는", k=8)
topk_next_tokens("여행 계획을 세우자. 대한민국의 수도는", k=8)
```

```
# ===== 실습 3) Self-Attention이 어디를 참고하는지 확인 =====
attention_peek("여행 계획을 세우자. 대한민국의 수도는", layer=-1, head=0, topn=10)
```