[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

> 영상에서는 “코드를 다 이해하려고 하기보다, **출력 결과가 말해주는 것**을 보자” 톤이 좋습니다.

```
# 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)
```

5분 강의 대본(타임코드 + 화면 지시 + 멘트)

아래 대본은 **영상 제작용**으로, "말"과 "화면에서 하는 행동"을 같이 적었습니다.

0:00–0:20 오프닝(실습 목적)

화면: 슬라이드 1장(제목) → 바로 Colab로 전환
멘트:

> "2주차 핵심이 세 가지였죠. **다음 단어 예측**, 그걸 가능하게 하는 **Transformer의 Self-Attention**, 그리고 그 위에서 발전해온 **GPT 계열**.
> 이번 5분 실습은, 이걸 '개념'이 아니라 **출력으로 눈으로 확인**하는 게 목표입니다."

0:20–0:50 환경 준비(모델 로드)

화면: Colab에서 설치/모델 로드 코드 실행
멘트:

> "오픈소스 GPT를 하나 불러오겠습니다. 오늘은 한국어 예시가 잘 보이는 **KoGPT2**를 쓸게요.
> 중요한 건 모델이 엄청 크냐가 아니라, **언어모델이 실제로 뭘 계산하는지**를 확인하는 겁니다."

0:50–1:40 실습 1) "다음 단어 예측"을 확률로 보기

화면: `topk_next_tokens("대한민국의 수도는", k=8)` 실행 결과
멘트:

> "지금 출력에 **Top-k next tokens**가 보이죠.
> 언어모델은 '정답 한 개'를 뽑는 게 아니라, 매 순간 **다음 토큰 전체에 대한 확률분포**를 만듭니다.
> 여기서 확률이 높은 토큰이 우리가 말하는 '다음 단어 후보'예요.
> 프롬프트 엔지니어링 관점에서는, 결국 우리가 하는 일은 이 **확률분포를 원하는 방향으로 기울이는 것**입니다."

포인트(짧게 강조):

- "정답"이 아니라 "분포"
- top-1만 보지 말고 top-5/top-10을 보면 모델의 '망설임'이 보임

1:40–2:40 실습 2) 문맥을 바꾸면 예측이 바뀐다(프롬프트의 힘)

화면: 아래 2개를 연달아 실행

- `"퀴즈: 대한민국의 수도는"`

- `"여행 계획을 세우자. 대한민국의 수도는"`

멘트:

> “이번엔 프롬프트를 살짝 바꿔볼게요. 의미는 비슷한데, 앞 문맥이 달라졌죠.
> 결과를 보면 Top-k 후보와 확률이 달라집니다.
> 이게 바로 ‘언어모델은 문맥 기반 다음 토큰 예측기’라는 말의 실체예요.
> 같은 질문이라도 앞에 ‘퀴즈’, ‘여행 계획’ 같은 힌트를 붙이면 모델은 **다음에 나올 말의 스타일과 주제**를 바꿉니다.”

영상에서 바로 던질 미션(짧게):

> “여러분도 프롬프트 앞에 ‘정답만 말해줘’, ‘이유를 한 줄로’, ‘초등학생에게 설명해줘’를 붙여서 Top-k가 어떻게 바뀌는지 확인해보세요.”

2:40–4:10 실습 3) Self-Attention “어디를 보고 있나” 살짝 엿보기

화면: `attention_peek(...)` 실행 결과(Top attended tokens 출력)
멘트:

> “이제 Transformer의 핵심인 **Self-Attention**을 직접 보겠습니다.
> 엄밀히는 레이어와 헤드가 엄청 많지만, 오늘은 ‘느낌’만 잡으면 돼요.
> 출력은 이런 의미예요: **마지막 입력 토큰이 이전 토큰들을 얼마나 참고했는지**를 가중치로 보여준 겁니다.
> 가중치가 큰 토큰들이, 모델이 다음 토큰을 예측하기 위해 특히 강하게 참고한 단서라고 볼 수 있어요.”

개념 연결 멘트(중요):

> “그러니까 ‘프롬프트를 잘 쓰는 것’은, 결국 모델이 **주의(attention)를 어디에 주게 만들지 설계하는 것**입니다.
> 긴 프롬프트에서 핵심 정보가 묻히면, 모델이 중요한 토큰을 덜 참고할 수도 있겠죠.”

4:10–5:00 GPT 계열 발전을 “실습 관점”으로 정리(짧은 마무리)

화면: 짧은 슬라이드 1장(텍스트 3줄) 또는 말로 정리
멘트:

> “마지막으로 GPT 발전을 실습 관점에서 한 줄로 정리하면 이렇습니다.
> 첫째, **규모(데이터/파라미터/학습)**가 커질수록 다음 토큰 예측이 더 안정적이고, 문맥을 더 잘 씁니다.
> 둘째, 기본 GPT는 원래 ‘문장 이어쓰기’에 강한데, 지금 우리가 쓰는 ChatGPT류는 여기에 **지시를 잘 따르게 만드는 튜닝(Instruction/RLHF 등)**이 더해져서 ‘대화’가 가능해졌습니다.
> 셋째, 그래서 프롬프트 엔지니어링은 **기본 예측기(언어모델) + 튜닝된 행동(챗봇)** 둘 다를 이해해야 훨씬 잘 됩니다.”

클로징 미션(10초):

> “오늘 실습 과제: 프롬프트를 3개만 바꿔서 **Top-5 후보**와 **Attention 상위 토큰**이 어떻게 달라지는지 캡처해보세요. 그게 다음 주 프롬프트 설계로 바로 연결됩니다.”

영상에 바로 넣을 “실습 과제(학생 제출용)” 3줄 버전

1. 프롬프트 3개를 만들기(같은 질문, 다른 문맥): 예) “퀴즈: …”, “친구에게 설명: …”, “한 문장 답: …”
2. 각 프롬프트에 대해 **Top-5 다음 토큰 후보와 확률**을 캡처
3. 그중 1개 프롬프트에 대해 **attention_peek 결과(상위 5~10개 토큰)** 캡처 후, “왜 저 토큰을 봤을까?” 한 줄 코멘트