7장(~GPT)

날짜	@2025년 5월 21일 → 2025년 5월 27일
선택	DL세션 과제
주차	12주차
진행 상태	완료

7장 트랜스포머

💭 트랜스포머(Transformer)

: 딥러닝 모델 중 하나로 순환 신경망이나 합성곱 신경망과 달리 어텐션 메커니즘(Attention Mechanism)만을 이용해 시퀀스 임베딩을 표현함.

- 인코더와 디코더 간의 상호작용으로 입력 시퀀스의 중요한 부분에 초점을 맞추어 문맥을 이해하고적절한 출력을 생성
- 어텐션 메커니즘은 인코더와 디코더 단어 사이의 상관관계를 계산하여 중요한 정보에 집중 → 시퀀스의 각 단어가 출력 시퀀스의 어떤 단어와 괄녈있는지 파악
- 순환 신경망 모델보다 빠름
- 병렬처리가 가능해 대규모 데이터셋트에 높은 성능을 보임
- 문장의 길이가 길어지더라도 성능이 유지됨

⇒ 자연어 처리 분야에서 가장 인기 있는 모델

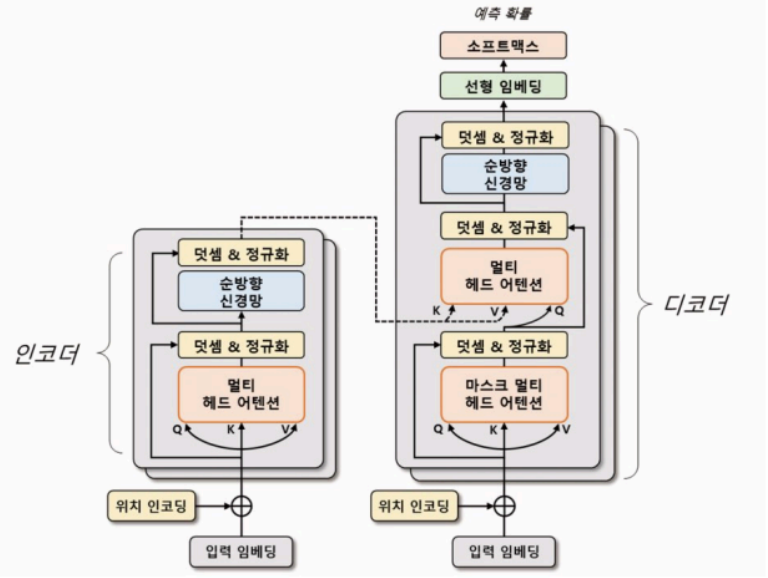


그림 7.2 트랜스포머 모델 구조(Q: 쿼리, K: 키, V: 값)

- 트랜스포머의 인코더와 디코더는 두 부분으로 구성되며, 각각 N개의 트랜스포머 블록으로 구성됨
- 트랜스포머 블록은 멀티 헤드 어텐션과 순방향으로 이루어짐

💭 **멀티 헤드 어텐션**

입력 시퀀스에서 **쿼리(Query), 키(Key), 값(Value) 벡터를 정의**해 입력 시퀀스들의 관계를 **셀프 어텐션(Self-Attention)하는 벡터 표현 방법**

- 쿼리와 각 키의 유사도를 계산 → 가중치로 사용하여 벡터를 합산
- 이렇게 계산된 어텐션 행렬은 입력 시퀀스 각 단어의 임베딩 벡터를 대체한다.

💭 **순방향 신경망**

산출된 임베딩 벡터를 고도화하기 위해 사용

여러 개의 선형 계층으로 구성

앞선 순방향 신경망 구조와 동일하게 입력 벡터에 가중치를 곱하고, 편향을 더하고, 활성화 함수를 적용

학습된 가중치들은 입력 시퀀스의 각 단어의 의미를 잘 파악할 수 있는 방식으로 갱신됨

트렌스포머에서는 **입력 시퀀스 데이터를 소스(Source)와 타깃(Target) 데이터**로 나눠 처리함.

(ex) 영어를 한글로 번역하는 경우, 생성하는 언어인 한글을 타깃 데이터, 참조하는 언어인 영어를 소스 데이터로 정의

인코더는 소스 시퀀스 데이터를 위**치 인코딩(Positional Encoding)된 입력 임베딩으로 표현**해 트랜스포머 블록의 출력 벡터를 생성

디코더는 인코더와 유사하게 트랜스포머 블록으로 구성되어 있지만, **마스크 멀티 헤드 어텐션(Masked Multi-Head Attention)을 사용해 타깃 시퀀스 데이터**를 순차적으로 생성시킴. 이때 인코더의 출력 벡터 정보를 참조함

최종적으로 생성된 디코더 출력 벡터는 **선형 임베딩으로 재표현**되어 이미지나 자연어 모델에 활용됨

입력 임베딩과 위치 인코딩

트랜스포머 모델에서 입력 시퀀스의 각 단어는 임베딩처리 되어 벡터 형태로 변환된다.

- 입력 시퀀스를 병렬 구조로 처리 → 단어의 순서 정보 X → 위치 정보를 임베딩 벡터에 추가해 단어의 순서 정보를 반영

  ⇒ 위치 인코딩

💭 위치 인코딩

입력 시퀀스의 순서 정보를 모델에 전달

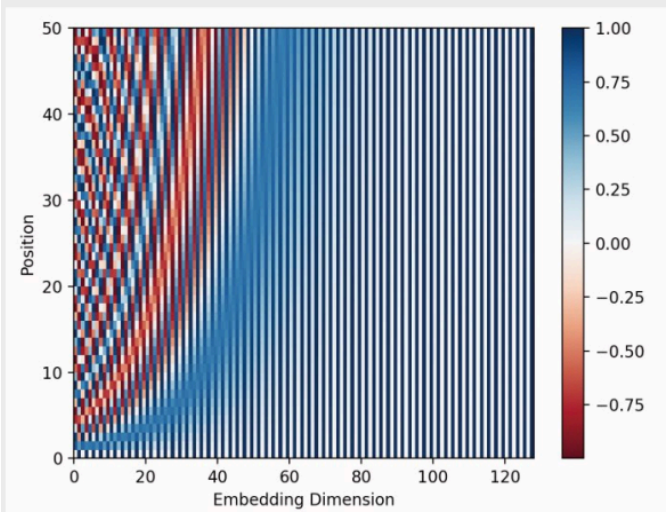
- 위치 인코딩 벡터는 sin 함수&cos 함수를 사용해 생성됨: 각 토큰의 위치를 각도로 표현
- 임베딩 벡터 + 위치 정보 ⇒ 최종 입력 벡터

수식 7.1 위치 인코딩

$$PE(pos, 2i) = \sin(pos/10000^{2i/d_{model}})$$
$$PE(pos, 2i+1) = \cos(pos/10000^{2i/d_{model}})$$

- pos: 입력 시퀀스에서 해당 단어의 위치
- i: 임베딩 벡터의 차원 인덱스
  - 짝수 sin, 홀수 cos
- 각도 정보로 변환하기 위한 스케일링 인자로 각 위치에 대한 주기적인 신호를 생성/2

```python
# 7.1 위치 인코딩
import math
import torch
from torch import nn
from matplotlib import pyplot as plt

class PositionalEncoding(nn.Module):
  # 임베딩 차원 & 최대 시퀀스를 입력받음
  def __init__(self, d_model, max_len, dropout=0.1):
    super().__init__()
    self.dropout = nn.Dropout(p=dropout)

    # sin cos 함수로 위치 인코딩을 계산
    position = torch.arange(max_len).unsqueeze(1)
    div_term = torch.exp(
        torch.arange(0, d_model, 2)*(-math.log(10000.0)/d_model)
    )
    pe = torch.zeros(max_len, 1, d_model)
    pe[:, 0, 0::2] = torch.sin(position * div_term)
    pe[:, 0, 1::2] = torch.cos(position * div_term)
    self.register_buffer("pe",pe)

  def forward(self, x):
    x = x + self.pe[: x.size(0)]
    return self.dropout(x)

encoding = PositionalEncoding(d_model=128, max_len=50)

plt.pcolormesh(encoding.pe.numpy().squeeze(),cmap="RdBu")
plt.xlabel("Embedding Dimension")
plt.xlim((0,128))
plt.colorbar()
plt.show()
```



🧩 특수 토큰

💭 **특수 토큰**

: 입력 시퀀스의 시작과 끝을 나타내거나 마스킹(Masking) 영역으로 사용

- 모델이 입력된 시퀀스의 시작/끝 인식
- 일부 입력을 무시할 수 있음

1. <BOS>: Beginning of Sentence, 문장의 시작
2. <EOS>: End of Sentence, 문장의 끝
3. <UNK>: Unknown, 어휘 사전에 없는 단어
4. <PAD>: Padding, 모든 문장을 일정한 길이로 맞추기 위함

→ 이렇게 생성된 문장 토큰 배열을 어휘 사전에 등장하는 위치에 원-핫 인코딩으로 표현

- 임베딩으로 변환되는 과정은 Word2Vec과 동일
  - 어휘사전의 크기 V, 입력 임베딩 차원 d
  - '트랜스포머의'의 원-핫 벡터는 [1,V]의 크기를 가짐
  - 임베딩 행렬[V,d]에 의해 [1,d] 크기의 벡터로 변환됨

  ⇒ N개의 문장이 최대 S개의 토큰 길이를 가질 때, [N, S, V] 크기의 원-핫 벡터 텐서는 [N, S, d] 크기의 임베딩 텐서로 변환됨 → 입력으로 사용되며, 트랜스포머의 모든 계층에서 공유되는 텐서로 사용됨

🔍 트렌스포머 인코더

💭 입력 시퀀스를 받아 → 여러 개의 계층으로 구성된 인코더 계층을 거쳐 연산을 수행

- 각 계층은 멀티 헤드 어텐션 & 순환 신경망으로 구성
- 입력 데이터에 대한 정보를 추출 → 다음 계층으로 전달
- 위치 임베딩 벡터를 입력 벡터에 더해줌
- 디코더 계층으로 전달

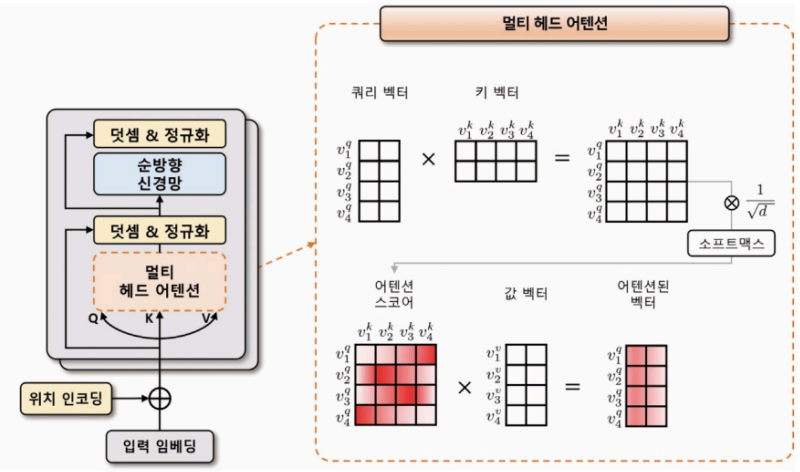


그림 7.5 트랜스포머 인코더 블록 연산 과정

<트랜스포머 인코더 블록 연산 과정>

1. 트랜스포머 인코더는 위치 인코딩이 적용된 소스 데이터의 입력 임베딩을 입력 받음.
2. 멀티 헤드 어텐션

   (1) 입력 텐서 차원 [N, S, d]일 때, 입력 임베딩은 선형 변환을 통해 3개의 임베딩 벡터 쿼리(Q), 키(K), 값(V) 벡터를 정의

💭 **쿼리 벡터(V^q)**: 현재 시점에서 참조하고자 하는 정보의 위치를 나타내는 벡터

- 인코더의 각 시점마다 생성됨
- 현재 시점에서 질문이 되는 벡터로, 이 벡터를 기준으로 다른 시점의 정보를 참조함

키 벡터(V^k): 쿼리 벡터와 비교되는 대상으로 쿼리 벡터를 제외한 입력 시퀀스에서 탐색되는 벡터

- 인코더의 각 시점에서 생성됨

값 벡터(V^v): 쿼리 벡터와 키 벡터로 생성된 어텐션 스코어를 얼마나 반영할지 설정하는 가중치 역할

이러한 쿼리, 키, 값 벡터는 초기에는 비슷한 값으로 가지지만, 모델 학습을 통해 의도한 의미의 값을 가짐

- 쿼리와 키 벡터의 연관성은 내적 연산으로 [N, S, S] 어텐션 스코어 맵을 사용함

수식 7.2 어텐션 스코어 계산식

$$score(v^q, v^k) = \text{softmax}\left(\frac{(v^q)^T \cdot v^k}{\sqrt{d}}\right)$$

(2) 셀프 어텐션 과정에서는 쿼리, 키, 값 벡터를 내적해 어텐션 스코어를 구하고 이 스코어값에 벡터 차원의 제곱근만큼 나눠 보정함

*벡터 차원이 커질 때 스코어 값이 같이 커지는 문제를 완화하기 위해 적용

(3) 보정된 어텐션 스코어를 softmax 함수로 확률적으로 재표현 → 값 벡터와 내적하여 셀프 어텐션된 벡터 생성 → 반복하여 셀프 어텐션된 스코어 맵 생성

3. 멀티 헤드(Multi-Head)는 이러한 셀프 어텐션을 여러 번 수행해 여러 개의 헤드를 만듦

   각각의 헤드가 독립적으로 어텐션 수행 → 결과를 합침

   (1) 입력 [N, S, d] 텐서에 k개의 셀프 어텐션 벡터를 생성하고자 한다면 헤드에 대한 차원 축을 생성해 [N, k, S, d/k] 텐서를 구성함: k개의 셀프 어텐션된 [N, S, d/k] 텐서를 의미

(2) k개의 셀프 어텐션 벡터는 임베딩 차원 축으로 다시 병합(concatenation)되어 [N, S, d]의 형태로 출력됨

4. 덧셈 & 정규화

   (1) 멀티 헤드 어텐션을 통과하기 이전 입력값 [N, S, d]텐서와 통과한 이후의 출력값 [N, S, d] 텐서를 더함으로써 기울기 소실을 완화함

   (2) 이 값에 임베딩 차원 축으로 정규화하는 계층 정규화를 적용함

5. 순방향 신경망

   a. 선형 임베딩과 ReLU로 이루어진 인공 신경망

   b. 1차원 합성곱

   두 가지 방법 중 하나를 적용할 수 있음.

6. 덧셈 & 정규화

트랜스포머 인코더는 여러 개의 트랜스포머 인코더 블록으로 구성됨 → 이전 출력 벡터는 다음 블록의 입력으로 전달되어 점차 입력 시퀀스의 정보가 추상화됨. → 디코더에 사용됨

⇒ 인코더와 디코더가 서로 정보를 공유!

🔎 트랜스포머 디코더

- 입력: 위치 인코딩이 적용된 타깃 데이터의 입력 임베딩
- 인코더의 멀티 헤드 어텐션 모듈은 인과성(Casuality)을 반영한 **마스크 멀티 헤드 어텐션 모듈**로 대체됨
  - 맵을 계산할 때 첫 번째 쿼리 벡터가 첫 번째 키 벡터만 바라보게 마스크를 씌움
  - 두 번째 쿼리 벡터는 첫 번째&두 번째 키 벡터를 바라보게 마스크를 씌움

  → 마스크를 적용하면 셀프 어텐션에서 현재 위치 이전의 단어들만 참조 → 인과성이 보장됨

  - 마스크 영역에 수치적으로 굉장히 작은 값인 -inf 마스크를 더해줌 → 해당 영역의 어텐션 스코어 값이 0에 가까워짐
  - softmax 계산 시 어텐션의 가중치가 0 → 마스크 영역 이전에 있는 입력 토큰들에 대한 정보를 참조하지 않게 됨.

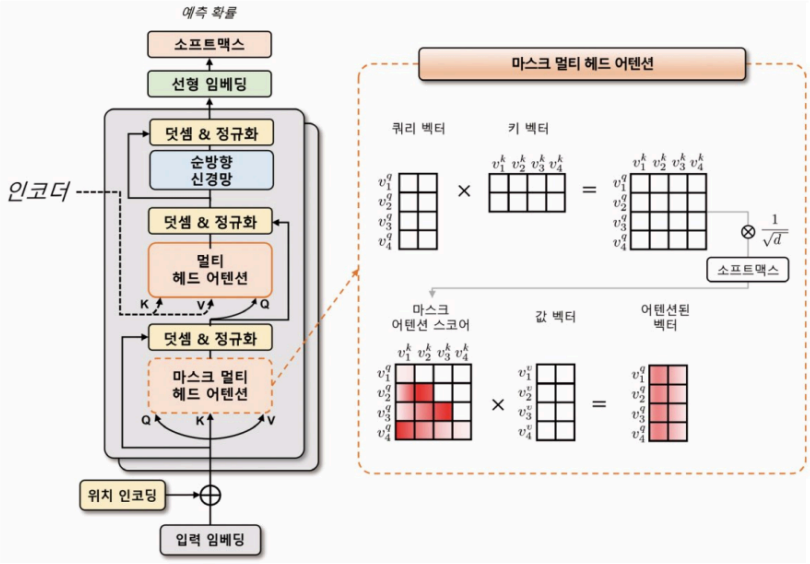


그림 7.6 트랜스포머 디코더 블록 연산 과정

<트랜스포머 디코더 블록 연산 과정>

1. 디코더의 멀티 헤드 어텐션
    a. 타깃 데이터: 쿼리 벡터(입력 임베딩 + 위치 인코딩)
    b. 인코더의 소스 데이터: 키와 값 벡터
2. 쿼리 벡터, 키 벡터, 값 벡터 → 어텐션 스코어 맵 계산
3. softmax 함수로 어텐션 가중치 계산
4. 어텐션 가중치와 값 벡터를 가중합 → 멀티 헤드 어텐션의 출력 벡터

- 인코더의 멀티 헤드 어텐션과 동일하지만, 마스킹이 적용된다는 차이점이 존재
- 인코더처럼 여러 개의 트랜스포머 디코더 블록으로 구성됨
- 이전 블록의 산출물이 다음 디코더 블록의 입력으로 전달됨:이전 시점 정보를 현 시점에서 활용
- 마지막 디코더 블록의 출력 텐서 [N, S, d]에 선형 변환 & softmax를 적용해 타깃 시퀀스 위치마다 예측 확률을 계산가능

⇒ 디코더는 타깃 데이터를 추론할 때 토큰 또는 단어를 순차적으로 생성시키는 모델

빈 값을 의미하는 PAD 토큰을 사용하여 입력 토큰을 순차적으로 나타냄

트랜스포머 클래스

```
transformer = torch.nn.Transformer(
    d_model = 512, # 트랜스포머 모델의 입력/출력 차원의 크기 정의
    nhead = 8, # 멀티 헤드 어텐션의 헤드 개수, 많을수록 병렬 처리 능력 & 매개변수 수
    num_encoder_layers = 6, # 인코더 계층 수
    num_decoder_layers = 6, # 디코더 계층 수
    dim_feedforward = 2048. # 순방향 신경망의 은닉층의 크기
    dropout = 0.1,
    activation = torch.nn.functional.relu,
    layer_norm_eps = 1e-05, # 계층 정규화 수행 시 분모에 더해지는 입실론 값
```

트랜스포머 순방향 메서드

```
output = transformer.forward(
    src,
    tgt,
    src_mask = None,
    tgt_mask = None,
    memory_mask = None,
    src_key_padding_mask = None,
    tgt_key_padding_mask = None,
    memory_key_padding_mask = None,
)
```

- 소스(src)와 타깃(tgt)은 인코더와 디코더에 대한 시퀀스
  - [소스(타깃) 시퀀스 길이, 배치 크기, 임베딩 차원] 형태의 데이터를 입력 받음
- 소스 마스크 & 타깃 마스크:소스와 타깃 시퀀스의 마스크
  - [소스(타깃) 시퀀스 길이, 시퀀스 길이] 형태의 데이터를 입력 받음
  - 마스크 값이 0이라면 해당 위치에서는 모든 입력 단어가 동일한 가중치를 갖고 어텐션이 수행됨
  - 1이라면 모든 입력 단어의 가중치가 0으로 설정 → 어텐션 연산이 수행되지 않음
  - -inf라면 해당 위치에서는 어텐션 결과에 0으로 가중치 부여 → 마스킹된 위치의 정보를 모델이 무시

    - +inf 라면 모든 입력 단어에 무한대 가중치가 부여돼 어텐션 결과가 해당위치에 대한 정보만으로 구성됨, 일반적으로 적용X
- 메모리 마스크: 인코더 출력의 마스크로 [타깃 시퀀스 길이, 소스 시퀀스 길이]의 형태를 가지며 메모리 마스크의 값이 0인 위치에서 연산 수행X
- 소스, 타깃, 메모리 키 패딩 마스크: 소스, 타깃, 메모리 시퀀스에 대한 패딩 마스크
    - [배치 크기, 소스(타깃) 시퀀스 길이] 형태의 데이터를 입력받으며, 메모리 키 패딩 마스크는 소스 키 패딩 마스크와 동일한 형태의 데이터를 입력 받음
    - 키 패딩 마스크: 입력 시퀀스에서 패딩 톡큰이 위치한 부분을 가리키는 이진 마스크로, 패딩 토큰이 실제 의미를 가지지 않는 것으로 간주되어 해당 위치의 어텐션 연산 결과에 대한 가중치를 0으로 만듦
- 순방향 메서드는 인스턴스의 설정과 입력 시퀀스를 통해 타깃 시퀀스의 임베딩 텐서를 반환하며 [타깃 시퀀스 길이, 배치 크기, 임베딩 차원]을 반환

🔮GPT(Generative Pre-trained Transformer)

💭 **GPT 모델은 트랜스포머의 디코더를 여러 층 쌓아 만든 모델 → ** 자연어 생성, 기계 번역, 질의응답, 요약 등 다양한 자연어 처리 작업에 사용

- 트랜스포머의 인코더는 입력 문장의 특징을 추출하는데 초점
- 트랜스포머의 디코더는 입력 문장과 출력 문장 간의 관계를 이해하고 출력 문장을 생성하는 데 초점
- GPT-1(2018), GPT-2(2019), GPT-3(2020) 버전이 갱신될수록 더 많은 트랜스포머 계층과 데이터를 활용해 학습
- 많은 양의 매개변수를 줄이기 위해 RLHF(Reinforcement Learning from Human Feedback)강화학습 방식을 도입해 GPT-3.5를 학습함
    - 인간이 모델이 생성한 결괌루을 평가하고, 모델이 좋은 결과물을 생성하면 보상을 줌
    - 2020 ChatGPT 서비스 공개
    - 2023 3월 GPT-4 모델 발표 : 이미지 데이터 처리 가능

GPT-1

: 트랜스포머 구조를 바탕으로 한 단방향(Unindirectional) 언어 모델로 현재 위치에서 이전 단어들에 대한 정보만을 참고해 다음 단어를 예측

- 단어들을 차례대로 입력 → 다음단어 예측: 계산량이 줄어들어 속도가 빠르고, 모델 크기가 작아질 수 있음
- 트랜스포머 구조 중 디코더만 사용
  - → 매개변수 수 줄임 & 성능 높임
  - 다중헤드 어텐션 사용 X

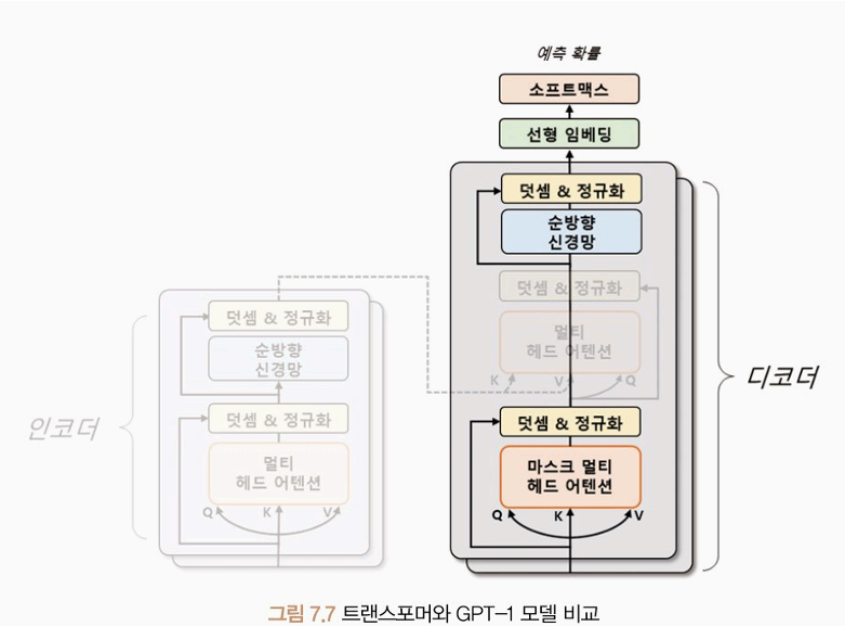


그림 7.7 트랜스포머와 GPT-1 모델 비교

- 인코더-디코더 간 어텐션 연산 제외
- 4.5GB의 BookCorpus 데이터세트로 언어 모델 사전 학습: 입력 문장을 임의의 위치까지 보여주고, 뒤에 이어지는 문장을 모델이 자동으로 예측하게 함
  - 별도의 레이블이 필요X → 비지도 학습

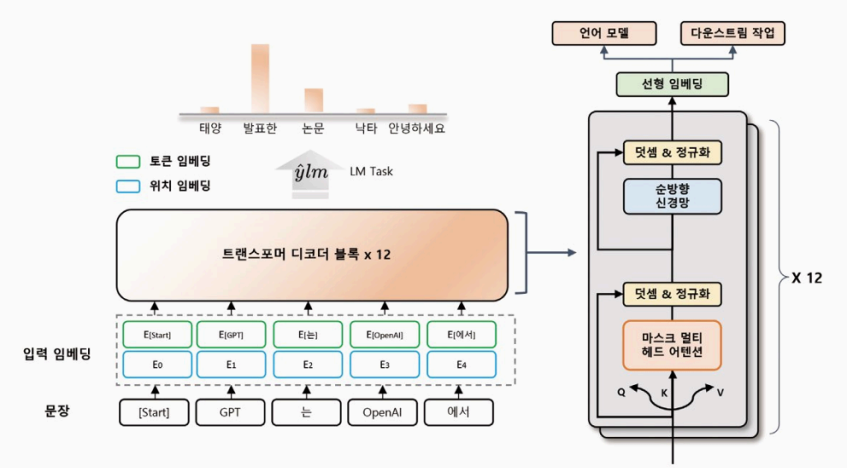


그림 7.8 GPT-1의 사전 학습 작업 구조

토큰 임베딩 + 위치 임베딩 → 입력 임베딩 계산

- 토큰 임베딩: 각 토큰을 고정된 차원의 벡터로 매핑
- 위치 임베딩: 입력 문장에서 각 토큰의 위치 정보를 나타내는 백터를 매핑

💭 보조학습

: 미세 조정을 위한 다운스트림 작업에서도 입출력 구조를 바꾸지 않고 언어모델을 **보조학습(Auxiliary Learning)**해 사용할 수 있음

- 주어진 모델에 다른 작은 부가적인 학습을 동시에 수행하는 것
- 다운 스트림 작업에서는 일반적인 언어 모델 학습 시 사용되는 손실함수 외에 다운스트림 작업에서 필요한 추가 손실함수가 필요함 → 언어 모델 개선 & 다운스트림 작업 목표 달성 가능
- 다운 스트림작업에서도 특수토큰이 사용됨
  - <start>: 문장의 시작 의미
  - <delim>: 두 문장의 경계 의미
  - <extract>: 문장의 마지막 의미

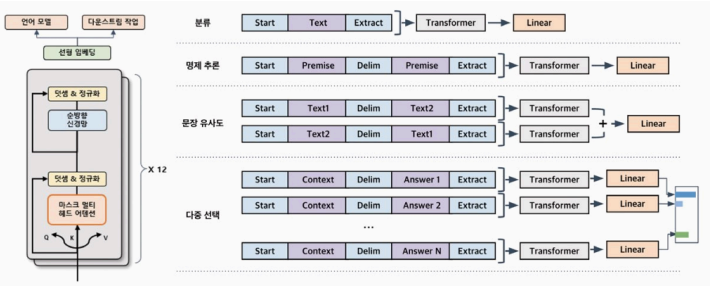


그림 7.9 GPT-1의 다운스트림 작업 구조

GPT-2

: 1024의 길이까지 입력 처리 가능 ↔ GPT-1(512문장)

- 48개의 디코더 계층 사용하여 복잡한 패턴 학습 가능 ↔ GPT-1(12개)
- 15억개의 매개변수 ↔ GPT-1(1200만 개)
- 40GB 데이터 사용하여 사전 학습 → 자연스러운 문장 생성 능력
- 제로-샷 학습 가능: 별도의 미세 조정 없이 다운스트림작업에서도 사용할 수 있게 사전 학습 과정에서 특정한 형식의 데이터를 입력함

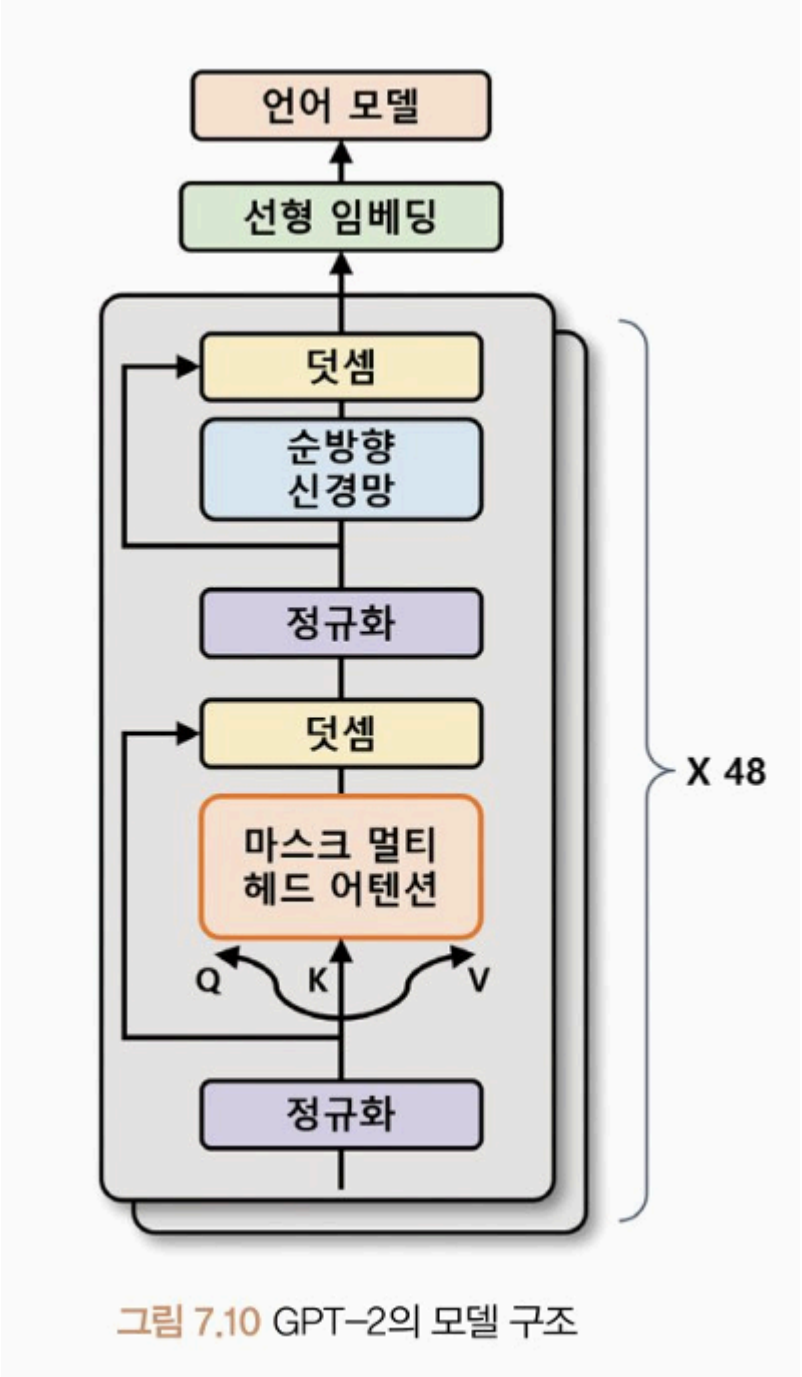


그림 7.10 GPT-2의 모델 구조

GPT-3

GPT-2와 거의 동일한 모델 구조

GPT-2보다 약 116배 많은 매개변수

- 96개의 헤드를 가진 멀티 헤드 어텐션
  - 연산량을 줄이기 위해 희소 어텐션 + 일반 어텐션 섞어 사용함
- 디코더 층 96개 사용

- 토큰 임베딩의 크기 1600 → 12888로 증가
- 최대 2048개의 토큰까지 입력가능하여 긴 문장 처리 능력 향상
- 45TB에 해당하는 방대한 양의 데이터세트를 이용하여 모델 학습
- 프롬프트(Prompt) 형식으로 작업 수행
- 자연어 처리 작업에 대한 일반화 능력이 높지만, 사람과 비교할 때 인간적인 이해력과 상식적인 추론이 부족함
- 결과물에 대한 검토와 검증 과정이 중요

GPT 3.5

: GPT-3의 모델 구조를 그대로 따르면서도, RLHF를 도입해 모델의 매개변수를 줄이고 자연스러움을 높인 특징을 가짐

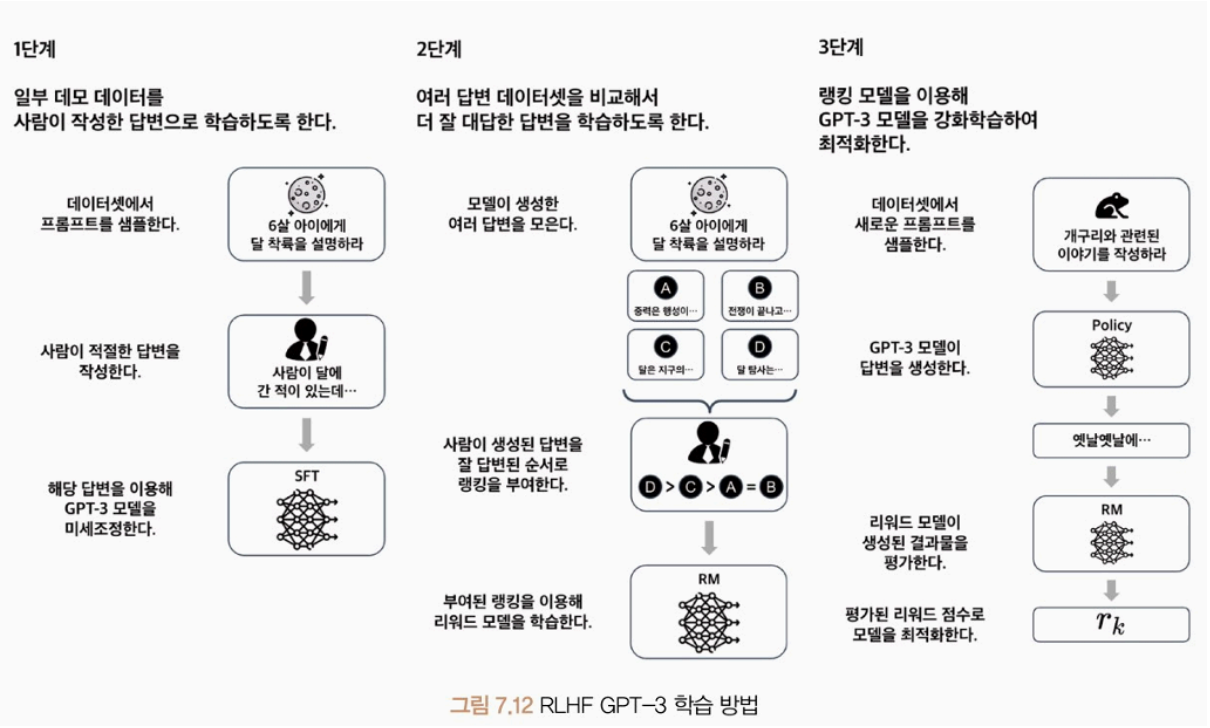


그림 7.12 RLHF GPT-3 학습 방법

- RLHF: 데이터세트에서 임의의 프롬포트를 가져와 사람이 직접 적절한 담을 작성 → 모델이 생성한 문장과 함께 평가해 보상을 주거나 주지 않음 → **지도 미세 조정 (Supervised Fine-Tuning, SFT)**
- 전체 프롬포트에 사람이 답변을 다는 것은 어려우므로 GPT 모델이 얼마나 잘 답변했는지 평가하는 보상(Reward) 모델을 학습함
- GPT 모델은 시드(seed)에 따라 다른 답변을 생성하므로 내용이 서로 다른 여러 개의 답변이 생성될 수 있음.

- RLHF 과정에서 생성된 답변과 랭킹 정보를 활용하여 리워드 모델이 학습되며, 더 자연스러운 문장 생성이 가능해짐
- GPT-4는 높은 성능 향상을 보엿지만, 언어 모델이 일부 입력에 대해 현실적이지 않은 결과를 생성하는 환각(Hallucination) 현상이 아직 존재함
    - 모델이 인간과 같은 추론과 판단 능력을 갖추지 못하고 부적절하거나 오류가 많은 답변을 제시하는 현상을 말함

GPT-4

이전 GPT 모델과 달리, 텍스트 데이터뿐만 아니라 이미지 데이터도 인식 가능한 멀티모달(Multimodal) 모델

- GOT-3.5의 최대 여덟 배의 토큰을 입력 받을 수 있음
- 25000개의 단어 처리 가능
- GPT-4의 모델 매개변수 수나 학습 데이터 및 방법은 공개되지 않았지만, GhatGPT Plus, 마이크로 소프트 Bing을 통해 모델 사용 가능