

# Transformer

일시: 2025년 8월 19일

제출자: 1반 윤소현

## 목차

1. Transformer의 구성
2. Query, Key, Value
3. Self-Attention
4. Multi-Head Attention
5. 정규화, 잔차 연결, Feed Forward Layer, Positional Encoding
6. 디코더에서의 Masked Self-Attention
7. 디코더에서의 Cross-attention
8. Linear Layer와 Softmax
9. Transformer의 학습
10. Transformer의 추론(디코딩)
11. 학습과 추론

### #1 Transformer의 구성

Transformer는 Encoder, Decoder로 구성되고 각각 Input/Output은 다음과 같습니다.

Encoder (문장 길이  $n$ , 임베딩 차원  $d$ 일때)

- Input은 Word Embedding 벡터(size  $n \times d$ )
- Output은 문맥 표현 벡터(size  $n \times d$ )입니다.

Decoder (현재까지 생성된 단어 길이  $k$ , Vocabulary Size  $v$ 일때)

- Input은 문맥 표현 벡터(Encoder Output, size  $n \times d$ )과 이전 시점 출력 토큰 임베딩 벡터 (size  $k \times d$ )
- Output은 다음 단어 후보들에 대한 확률 분포(size  $1 \times v$ )입니다.

### #2 Query, Key, Value

- 인코더에 입력으로 들어온 Word Embedding 벡터의 크기가  $n \times d$  이면 여기에 가중치 행렬을 곱해서 차원을 변형합니다. 예를 들어  $d \times d_k$  크기의 가중치 행렬을 곱해주면 입력은  $n \times d_k$  크기로 변환되어 원래  $d$  차원이었던 단어 임베딩 벡터가  $d_k$  차원 표현으로 바뀌게 됩니다.
- 선형 변환으로 차원을 바꾼 뒤 Query, Key, Value 벡터로 나눕니다.
  - Query는 “내가 누구를 참고할지, 어디에 집중할지”, Key는 “Query가 참고할 수 있는 정보”, Value는 “실제로 전달될 정보” 입니다.

### #3 Self-Attention

Self-Attention에서 토큰이 자기 자신과 다른 모든 토큰들 사이의 관련성을 계산하는 법은 다음과 같습니다.

1. Query와 Key를 내적해  $n \times n$  크기의 score 행렬을 만듭니다. 행렬에서  $(i, j)$  위치의 값은  $i$ 번째 토큰이  $j$ 번째 토큰을 얼마나 주목해야 하는지 score인데, 내적 값이 크면 유사성이 높다는 뜻이고 주목해야 할 대상이라는 뜻입니다 (이처럼 토큰들 사이의 관련성을 자기 자신 안에서 계산하기 때문에 Self-Attention이라고합니다).
2. 내적 값이 너무 커질 경우 특정 항목만 지나치게 강조될 수 있으므로 score를  $d_k$ 의 제곱근으로 나누어 스케일링하는 과정을 통해 값의 분산이 안정화해서 학습을 안정화합니다.

3. Softmax 함수를 적용해 각 행이 합이 1이 되도록 확률 분포로 바꿉니다 이렇게 변환된 값이 Attention Score로, 각 토큰이 다른 토큰을 얼마나 참고할지를 확률 형태로 표현한 것입니다.
4. Value 벡터와 이 Attention Score를 곱합니다 즉 원래의 정보(Value)를 점수에 비례해 가중합한 새로운 벡터를 만듭니다. 이 결과는 원래 토큰 벡터를 업데이트한 것과 같습니다 즉, 각 토큰이 문맥 속에서 어떤 다른 토큰과 얼마나 연결되어 있는지를 반영해 업데이트된 새로운 벡터(문맥 벡터)가 생성됩니다.

#### #4 Multi-Head Attention

##### Multi-Head 필요성

- Self-Attention을 한 번만 거치면, 특정한 기준(맥락, 의미, 어휘적 유사성 등)에서만 관계를 포착할 수 있습니다. Multi-Head Attention은 이런 Self-Attention을 여러 개 병렬로 실행해서 서로 다른 관점에서 입력을 바라볼 수 있도록 합니다 예를 들어 어떤 헤드는 단어의 순서적 맥락에 집중할 수 있고, 또 다른 헤드는 의미적 유사성에 주목할 수 있으며, 또 다른 헤드는 특정 어휘 패턴을 따라가며 관계를 봅니다. 이렇게 여러 헤드가 만들어내는 다양한 관점을 합치면 일종의 앙상블처럼 작동해서 모델은 훨씬 다차원의 표현을 생성하게 됩니다.

##### 학습 과정

1. 각 헤드는 Q, K, V를 각각 독립적인 가중치 행렬로 변환합니다 따라서 같은 입력이라도 헤드마다 Q, K, V가 달라지고 그 결과로 나온 Attention Output도 서로 다릅니다. 즉 헤드별로 서로 다른 방식으로 "무엇을 주목할지"를 학습합니다.
2. 각 헤드의 Attention Output은  $n \times d_k$  와 같은 크기의 행렬로 나오는데 각 헤드의 출력들을 옆으로 이어붙이는 방식으로 결합합니다(Concat).
3. 단순히 붙인 결과는 각 헤드의 특징이 분리된 채로 남아 있어 모델이 이를 자연스럽게 활용하기 어려우므로 이어붙인 벡터를 다시 한 번 선형 변환해서 하나의 통합된 표현으로 만듭니다. 이렇게 하면 맥락 정보, 의미 정보, 어휘 정보 등 다양한 관점의 결과가 하나의 일관된 벡터 공간 안에서 재표현되어 이후 레이어들이 이 표현을 자연스럽게 사용할 수 있습니다.

#### #5 정규화, 잔차 연결, Feed Forward Layer, Positional Encoding

##### 정규화(Normalization)

- 정규화는 각 토큰 벡터 차원별 평균과 분산을 정규화해서 입력 분포가 일정하게 유지되도록 만듭니다. 이렇게 하면 학습이 빠르고 안정적이 되며, 그래디언트 소실이나 폭주를 막을 수 있습니다.

##### 잔차 연결(Residual Connection)

- 연산 과정에서 원래 입력 정보를 보존하기 위해 사용됩니다. 어텐션 결과만 계속 쌓아가면 초기 입력의 정보가 소실될 수 있으며 이를 방지하기 위해 원래 입력을 연산 결과에 더해주는 방식으로 정보 흐름을 유지합니다.

##### Feed Forward Layer

- 어텐션만으로는 선형 결합만 수행되므로 모델의 표현력이 부족할 수 있습니다 딥러닝의 핵심은 비선형성을 주입하는 것인데, 이를 위해 Activation Function를 사용해 입력 표현을 더 고차원으로 바꿉니다.

##### Positional Encoding

- 어텐션 메커니즘은 모든 토큰을 동시에 바라보기 때문에 토큰의 순서를 직접적으로 알 수 없습니다 예를 들어 "나는 밥을 먹었다"와 "밥이 나를 먹었다"는 순서가 바뀌면 의미가 완전히 달라지지만, 어텐션만 사용하면 두 문장을 구분하기 어렵습니다. 이를 해결하기 위해 입력 임베딩에 순서 정보를 더해주는 것이 Positional Encoding입니다. 위치 정보를 정수로 추가하면 값의 범위가 커져서 다루기 힘들기 때문에 sin과 cos 함수를 이용해 주기적인 패턴으로 위치를 표현합니다. 이렇게 하면 어떤 위치든 간결하게 표현할 수 있고, 모델은 순서를 반영한 연산을 할 수 있습니다.

#### #6 디코더에서의 Masked Self-Attention

##### 마스킹(masking)

- 트랜스포머의 어텐션 메커니즘은 모든 단어가 한꺼번에 보일 때 서로 간의 문맥을 파악하는 구조지만 디코더는 문장을 생성할 때 미래 단어까지 동시에 볼 수 없도록 제한해야 합니다. 예를 들어 "I study AI hard"라는 문장을 받을 때 "I study"까지 입력이 주어졌다면, 그 시점에서 "AI hard"라는 단어들은 아직 주어지지 않은 정보이므로 모델이 참고하면 안 됩니다.
- 이를 위해 마스킹 과정을 거칩니다. 마스킹에서 중요한 연산은 Q와 K의 내적을 통해 어텐션 score matrix를 만드는 것입니다. 그런데 마스킹이 적용되면 아직 주어지지 않은 단어는 스코어 계산에서 제외됩니다 따라서 어텐션 스코어 행렬에는 주어진 토큰

큰까지만 반영됩니다.

## #7 디코더에서의 Cross-attention

- 디코더의 Encoder-Decoder Attention 층은 디코더의 현재 hidden state가 인코더가 출력한 문맥 표현 벡터를 반영하여 어떤 위치(단어)에 주의를 기울일지를 알려주는 역할을 합니다 (현재 hidden state: 이전 시점 출력 토큰 임베딩들이 Masked Self-Attention을 거쳐 얻어진 중간 표현 벡터).

## #8 Linear Layer와 Softmax

- Decoder가 만든 벡터(= 이전 시점 출력 토큰 임베딩이 디코더의 Self-Attention → Cross-Attention → FFN을 거쳐 얻어진 벡터)는 Linear Layer를 통해 모델이 학습한 전체 어휘 집합과 연결된 로짓 벡터가 됩니다. 만약 어휘가 10,000개라면, 로짓 벡터는 10,000차원의 벡터가 됩니다. 벡터의 각 원소는 해당 단어가 정답일 가능성을 점수 형태로 표현합니다. 로짓 벡터의 값들은 실수라 확률로 해석할 수 없으므로 Softmax 함수를 적용하여 모든 값의 합이 1이 되도록 만들고 변환된 결과는 "각 단어가 다음에 나올 확률 분포"가 됩니다. 이를 기반으로 가장 높은 확률을 가진 단어가 선택되어 실제 출력 단어가 됩니다. 이 과정이 반복되면서 한 단어씩 생성해가며 최종 문장을 만들어냅니다.

## #9 Transformer의 학습

### 학습 목표

- Output Vocabulary는 모델이 출력할 가능한 후보 단어들의 목록입니다.
- 정답 벡터는 Output Vocabulary의 크기만큼 차원을 가진 벡터로 정답 단어 위치만 1이고 나머지는 0인 벡터입니다.
- 모델 출력(예측)은 Linear + Softmax를 거쳐 나온 확률 분포로써 모든 값의 합이 1인 벡터입니다.
- 정답은 "답인 한칸만 1인 벡터", 예측은 "모든 값의 합이 1인 분포(벡터)"입니다. 두 벡터를 얼마나 가깝게 만들 것인가가 학습 목표입니다.

### Cross-Entropy + 역전파

- 두 벡터를 얼마나 가깝게 만들 것인가가 학습 목표인데 "가깝다/멀다"의 정도를 Cross-Entropy로 측정합니다. Cross-Entropy는 확률 분포에서 정답 칸의 확률값 크기에 따라 손실을 부여합니다. 정답 칸의 확률을 p라고 하면 손실은  $-\log p$ 가 됩니다. 모델이 정답칸 확률을 0.9로 주면  $-\log(0.9)$ 라는 작은 손실을 받고, 0.1로 주면  $-\log(0.1)$ 이라는 큰 손실을 받습니다. 이 손실은 역전파 과정에서 모델의 파라미터에 대해 손실 함수의 기울기를 계산하는 형태로 전파되고 정답 클래스의 확률이 커지도록 가중치를 조정합니다.

## #10 Transformer의 추론(디코딩)

- 학습이 끝나고 모델을 실제로 쓰는 단계에서는 정답이 없으므로 디코더가 스스로 다음 단어를 선택해가며 문장을 생성합니다. Decoder가 생성한 벡터(= 이전 시점 출력 토큰 임베딩이 디코더의 Self-Attention → Cross-Attention → FFN을 거쳐 얻어진 벡터)는 Linear Layer를 거쳐 '해당 단어가 정답일 가능성' 벡터(로짓 벡터)가 되고 Softmax 함수를 통해 '다음 단어 후보들'에 대해 얼마나 가능성이 높다고 판단하는지 정보'인 확률 분포가 됩니다.

### Greedy Decoding

- 생성된 확률 분포에서 확률이 가장 큰 단어를 선택해 출력하는 것이 Greedy Decoding입니다.

### Beam Search

- 모델이 문장을 만들기위한 첫 단어를 뽑을 때, Greedy Decoding이 확률이 가장 큰 단어 1개를 선택한다면 Beam Search에서는 확률이 높은 단어 k개를 후보로 두고, 각 후보로 시작하는 k개 시퀀스(후보 문장)를 선택합니다.
- 두 번째 단어를 뽑을 땐 k개 시퀀스마다 두 번째 단어를 추가하고 각 시퀀스의 누적 로그 확률을 계산, 확률이 높은 k개의 시퀀스만 남기고 나머지는 제거하여 후보는 k개를 유지합니다. 이러한 절차를 반복하여 최종적으로 완성된 시퀀스들 중에서 누적 확률이 가장 높은 시퀀스를 최종 출력으로 선택합니다. 빔 크기 k를 키우면 더 많은 후보를 선택함으로써 더 다양한 문장 경로를 탐색할 수 있지만 계산량이 늘어납니다. 빔 크기를 1로 줄이면 그리디 방식이 됩니다.

## #11 학습과 추론

#### 학습과 추론의 차이

- 학습 때는 정답이 주어지므로 모델의 출력 확률 분포와 정답 벡터를 비교하여 오차를 계산할 수 있고, 추론 때는 정답이 없으므로 모델이 스스로 다음 단어를 선택해야 하는 환경입니다.
- 학습 때는 출력인 예측 분포와 정답을 비교하여 Cross-Entropy를 계산하고 가중치를 수정하며 추론 때는 확률 분포에서 Greedy 또는 Beam Search 등으로 다음 단어를 선택합니다.