Transformer 구조 조사 및 분석

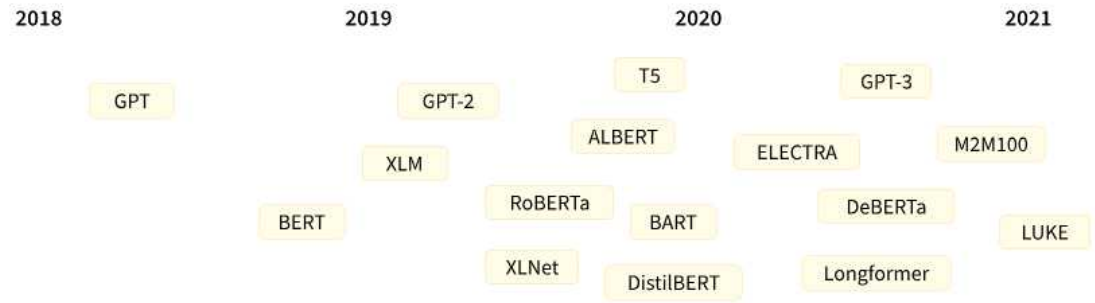
과목명 : 딥러닝 프레임워크
학번 : 202004245, 202184028
이름 : 한지은, 성윤주
제출일 : 2024.06.04

I. 서론

트랜스포머(Transformer)는 2017년 구글이 발표한 "Attention is all you need" 논문에서 소개된 모델이다. 이 모델은 기존의 seq2seq 구조인 Encoder-Decoder를 따르면서도 어텐션(Attention) 메커니즘만을 사용하여 구현되었다. RNN을 사용하지 않고 Encoder-Decoder 구조를 설계하였지만, 번역 성능 면에서 RNN보다 더 우수한 결과를 보여주었다.

트랜스포머 모델의 중요성을 알기 위해 먼저 기존의 seq2seq를 상기해보면, seq2seq 모델은 Encoder-Decoder 구조로 구성돼 있다. 여기서 인코더(Encoder)는 입력 시퀀스를 하나의 벡터 표현으로 압축하고, 디코더(Decoder)는 이 벡터 표현을 통해서 출력 시퀀스를 만들어냈다. 하지만 이러한 구조는 인코더가 입력 시퀀스를 하나의 벡터로 압축하는 과정에서 입력 시퀀스의 정보가 일부 손실되었고, 입력값을 순차적으로 하나씩 받기 때문에 필연적으로 계산 속도가 느리다는 단점이 발생했다.

트랜스포머 모델을 사용하면 RNN을 사용하지 않아 연산 효율도 좋아지고, 뿐만 아니라 성능도 더욱 좋아진다. 어텐션 메커니즘을 사용하여 병렬 처리가 가능해지면서, 긴 시퀀스 데이터를 더 빠르게 처리할 수 있기 때문이다. 또한 셀프 어텐션(self-attention) 메커니즘을 사용하여 각 단어의 중요도를 동적으로 결정하기 때문에 문맥을 더 잘 이해하고 반영할 수 있게 하여 자연어 처리 작업에서 더 정확한 결과를 도출할 수 있다. 특히 긴 문장이나 복잡한 문맥에서도 우수한 성능을 보인다. 트랜스포머 모델이 중요한 점엔 확장성도 있는데, 예를 들어 BERT, GPT, T5 등 다양한 변형 모델들이 등장하면서 각각의 모델이 특정 작업에서 최적화된 성능을 발휘할 수 있고 다양한 형태의 데이터 처리에 매우 유연하게 대응할 수 있다.

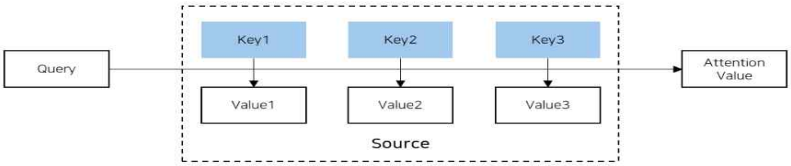


GPT 시리즈와 같은 모델들은 트랜스포머의 디코더를 분리하여 독자적인 모델로 발전시킨 것으로, 다음 단어를 예측하는데 뛰어나 주로 대화형 AI, 자동 소설 작성, 기사 작성 같은 텍스트 생성 작업에 활용된다. BERT와 같은 인코더 구조의 모델들은 문서 내에서 질문에 대한 정확한 답변을 찾아내는 데 매우 뛰어난 성능을 보인다.

트랜스포머는 NLP 분야 외에도 기계의 시각에 해당하는 부분을 연구하는 Computer Vision(CV) 분야에서도 사용되고 있다. 특히, ViT와 같은 모델은 이미지 분류, 객체 검출, 이미지 생성 및 세그멘테이션 등 다양한 응용 분야에서 우수한 성능을 보인다.

II. 이론적 배경

트랜스포머의 이론적 배경을 이해하려면 먼저 Self-Attention에 대해 알아야 한다. Self-Attention은 문장에서 단어들의 연관성을 파악하기 위해 입력 시퀀스 내의 각 단어와 다른 모든 단어와의 관계를 고려하여 가중치를 부여하는 방식이다. 쿼리(Query), 키(Key), 밸류(Value)라는 세 가지 벡터가 존재한다. Self-Attention은 Query, Key, Value 세 요소 간의 문맥적 관계를 추출하는 과정이며, 입력 벡터 시퀀스에 쿼리(Q), 키(K), 밸류(V)를 생성하는 행렬(W)을 각각 곱하는 방식으로 이루어진다. (Q = X * WQ, K = X * WK, V = X * WV) 이때 Query, Key, Value의 초기 입력 벡터는 동일하다.



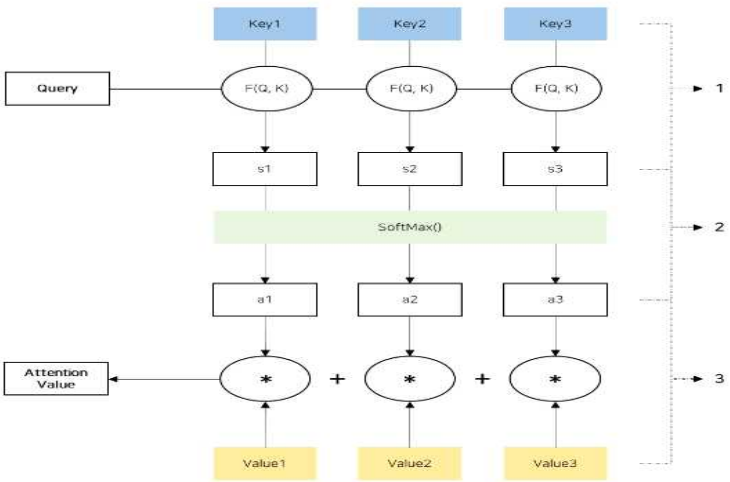
이 과정은 인코더와 디코더 블록 모두에서 수행되며, 세 행렬은 태스크를 가장 잘 수행하는 방향으로 업데이트된다. 예를 들어, "주어-동사-목적어" 구조의 문장이 있을 때 인공지능은 학습 과정에서 문장을 개별 단어로 분리하고, 그 단어가 주어인지 동사인지를 구분하기 위해 중요도를 측정하여 강조한다. 이때 사용되는 Self-Attention Function은 입력 시퀀스의 각 단어에 대한 가중치를 계산하는 함수로, 각 단어의 중요도를 측정하여 출력에 반영한다.

Self-Attention에서 Query는 현재 출력 단어를 나타내는 벡터(t 시점의 디코더 셀에서의 은닉 상태)이고, Key와 Value는 입력 시퀀스의 각 단어에 대응하는 벡터(모든 시점의 인코더 셀의 은닉 상태)이다. Attention Function은 다음과 같은 순서로 계산된다.

(1). Query 벡터와 Key 벡터 간의 유사도(Attention Score)를 측정한다.

(2). 유사도를 소프트맥스 함수를 통해 정규화하여 각 단어의 가중치(인코더의 각 시점의 정보 중요도를 나타내는 지표)를 계산한다.

(3). (2)의 가중치를 Value 벡터와 곱한 후 모든 인코더 시점을 합산하여 Attention 값을 구한다.



위 과정에서 입력 시퀀스가 "The cat sat on the mat"이라면, Self-Attention 메커니즘은

"cat"이라는 단어가 시퀀스의 다른 모든 단어("The", "sat", "on", "the", "mat")와 어떻게 상호작용하는지를 평가한다. "cat"의 최종 표현은 이 상호작용을 바탕으로 가중 평균된 값이 된다. Self-Attention 메커니즘 덕분에 트랜스포머 모델은 입력 시퀀스의 모든 부분 간의 상호작용을 고려하여 더 깊고 정교한 문맥 이해를 가능하게 한다.

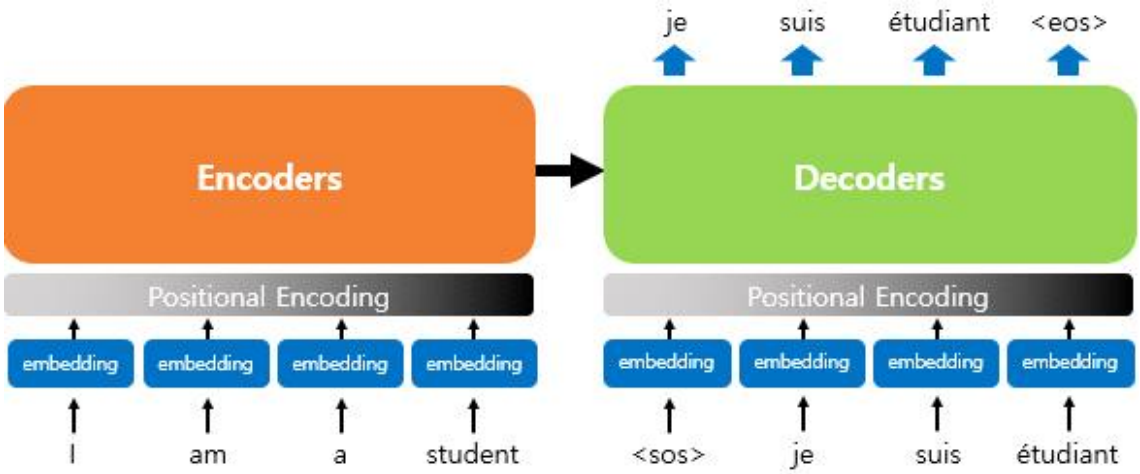
다음으로는 Positional Encoding이다. 위에 언급했듯 트랜스포머는 어텐션 메커니즘을 사용하여 긴 시퀀스 데이터를 병렬 처리로 빠르게 처리할 수 있다. 하지만 RNN과 LSTM과 다르게 입력 순서가 단어 순서라는 정보를 보장하지 않는다. 즉 트랜스포머의 경우 시퀀스가 병렬로 한번에 입력되기에 단어의 순서에 대한 정보가 사라지고, 단어의 위치 정보를 별도로 넣어줘야 한다.

단어의 위치 정보가 중요한 이유는 다음과 같다.

1. Although I did not get 95 in last TOEFL, I could get in the Ph.D program.
2. Although I did get 95 in last TOEFL, I could not get in the Ph.D program.

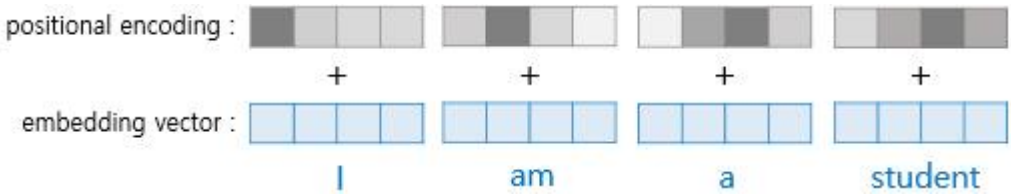
위 두 문장을 해석해보면, 1번 문장은 '지난 토플시험에서 95점을 못 받았지만, 박사과정에 입학할 수 있었다.'이고 2번 문장은 '지난 토플시험에서 95점을 받았지만, 박사과정에 입학하지 못했다.'가 된다. not이라는 단어의 위치 차이로 인해 두 문장의 뜻이 완전히 달라져버린 것을 알 수 있다. 이와 같이 문장 내의 정확한 단어 위치를 알 수 없다면 문장의 뜻이 완전히 달라지는 문제가 발생할 수밖에 없다. 따라서 각각의 단어 벡터에 Positional Encoding을 통해 얻은 위치 정보를 더해줘야 한다.

이 때 모든 위치 임베딩은 입력 시퀀스의 각 위치에 대해 고유하고 일관된 임베딩 값을 가져야 한다. 예를 들어 'I love dogs', 'You like cats' 이라는 두 문장이 있을 때 각 단어의 위치를 인식하기 위해 첫 번째 단어에는 동일한 위치 임베딩을, 두 번째 단어에는 또 다른 동일한 위치 임베딩을 부여한다. 따라서 I와 You는 서로 다른 단어이지만, 둘 다 첫 번째 위치에 있기 때문에 동일한 위치 임베딩 값을 가지게 된다. 마찬가지로 love와 like도 서로 다른 단어이지만 두 번째 위치에 있기 때문에 동일한 위치 임베딩 값을 가진다. 또 모든 위치 임베딩 값의 크기가 너무 크면 안된다. 위치값이 너무 커지면 단어 간의 상관관계 및 의미를 유추할 수 있는 의미정보값이 상대적으로 작아지게 되고, 어텐션 Layer에서 학습 및 훈련이 제대로 되지 않을 수 있다.



위 그림은 입력으로 사용되는 임베딩 벡터들이 트랜스포머의 입력으로 사용되기 전에 포지셔널 인코딩 값이 더해지는 과정을 보여준다. 임베딩 벡터가 인코더의 입력으로 사용되기 전,

포지셔널 인코딩 값이 더해지는 과정을 시각화하면 아래와 같다.

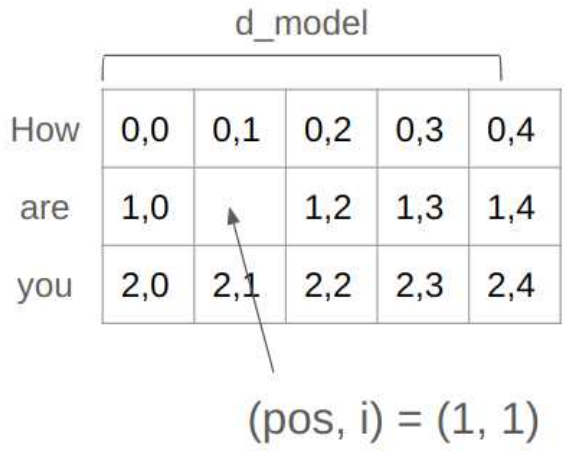


트랜스포머는 위치 정보를 가진 값(위치 임베딩 벡터)을 만들기 위해서 아래 두 개의 함수를 사용한다.

$$PE_{(pos,\,2i)} = sin(pos/10000^{2i/d_{model}})$$

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

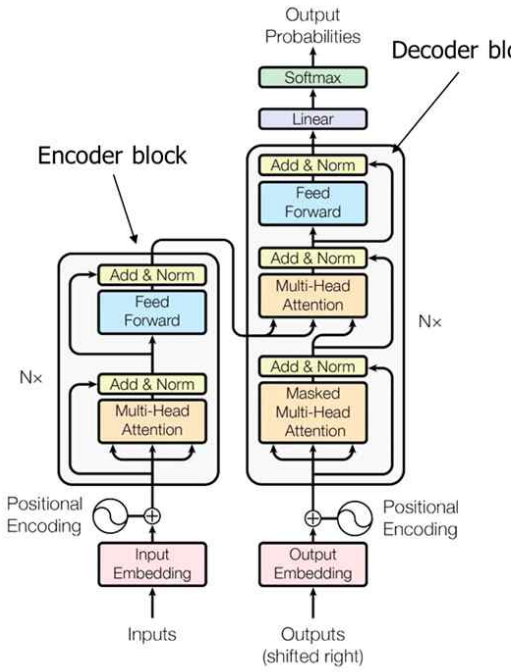
일정한 패턴을 반복하는 sin함수와 cos함수는 주기적인 패턴을 생성하므로 위치 인코딩 값들이 서로 다른 주기를 가지게 된다. 이것은 각 차원이 서로 다른 위치 정보를 인코딩하는 데 도움이 된다. pos는 입력 문장에서 임베딩 벡터의 위치를 나타내며, i는 임베딩 벡터 내의 차원의 인덱스를 의미한다. 위 식에 따르면 임베딩 벡터 내의 각 차원의 인덱스가 짝수인 경우에는 사인 함수의 값을 사용하고 홀수인 경우에는 코사인 함수의 값을 사용한다.
또한 위의 식에서 d_{model}는 트랜스포머의 모든 층의 출력 차원을 의미하는 트랜스포머의 하이퍼파라미터이다.



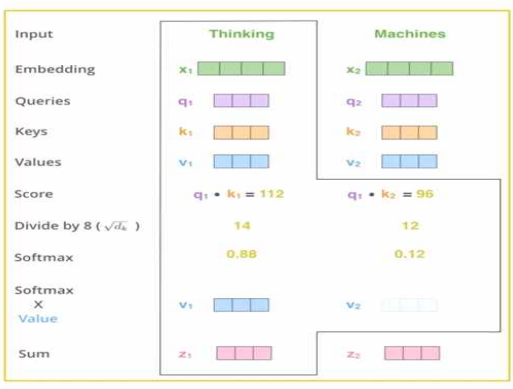
위와 같은 포지셔널 인코딩 방법을 사용하면 순서 정보가 보존되는데, 예를 들어 각 임베딩 벡터에 포지셔널 인코딩의 값을 더하면 같은 단어라고 하더라도 문장 내의 위치에 따라서 트랜스포머의 입력으로 들어가는 임베딩 벡터의 값이 달라진다. 이에 따라 트랜스포머의 입력은 순서 정보가 고려된 임베딩 벡터가 된다.

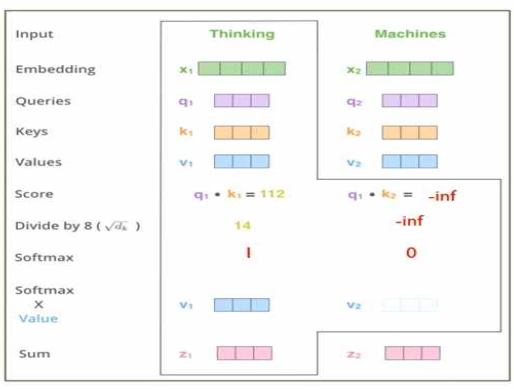
III. Transformer 구조 분석

Transformer의 두 주요 구성 요소인 인코더와 디코더의 구조와 역할에 대해 알아보겠다.

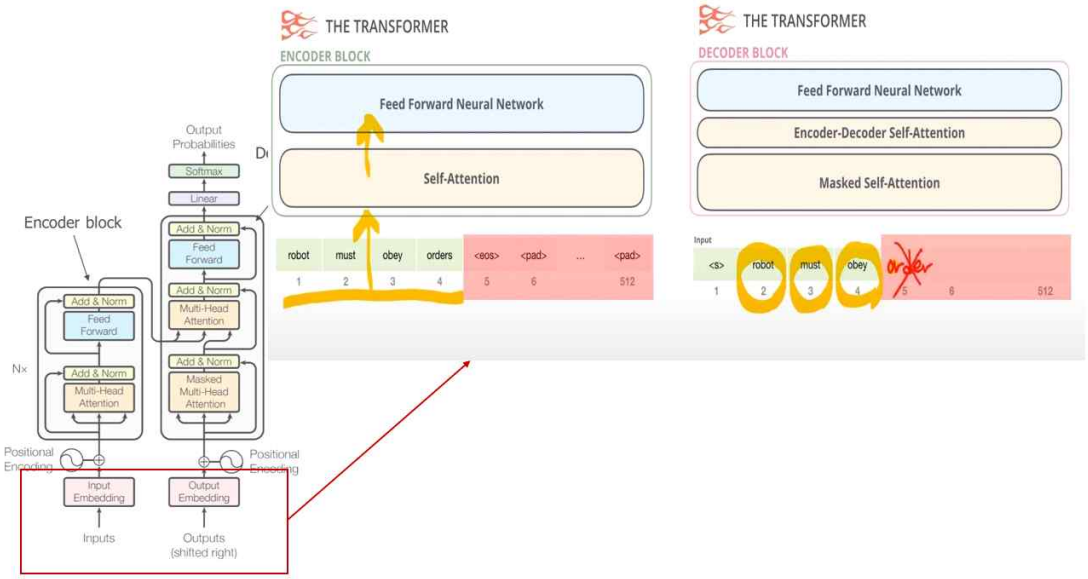


인코더는 먼저 입력값을 받아들이고 이를 임베딩한 후 포지션 인코딩을 추가한다. 그 다음, 멀티 헤드 어텐션을 거쳐 각각의 어텐션 아웃풋을 생성한다. 이후, 이러한 아웃풋에 원래 입력값을 더하고(Residual Connections) 이를 정규화한다(Layer Normalization). 그런 다음, feed forward를 적용하고 다시 한 번 Residual Connections와 Layer Normalization을 적용하여 이 과정을 반복한다. 디코더는 인코더와 유사하나, 중요한 차이가 있다. 멀티헤드 어텐션에서는 자신보다 뒤에 나온 단어들을 고려하지만, 디코더에서는 이러한 단어들을 보지 않도록 마스킹된다. 디코더는 자신보다 먼저 있는 토큰의 어텐션 스코어만을 볼 수 있다. 마스크 어텐션은 이러한 마스킹을 수행하며, 이후에는 인코더와 유사하게 Residual Connections 및 Layer Normalization을 거친다. 인코더와 디코더의 출력값 사이에서는 멀티 어텐션을 수행한다. 이후, Residual Connections와 Layer Normalization을 거쳐 새로운 벡터를 생성하고 Softmax로 정규화된 어텐션 가중치를 얻는다. 이를 통해 입력 벡터를 어텐션 가중치로 변환하여 해당하는 단어를 찾아낼 수 있다.





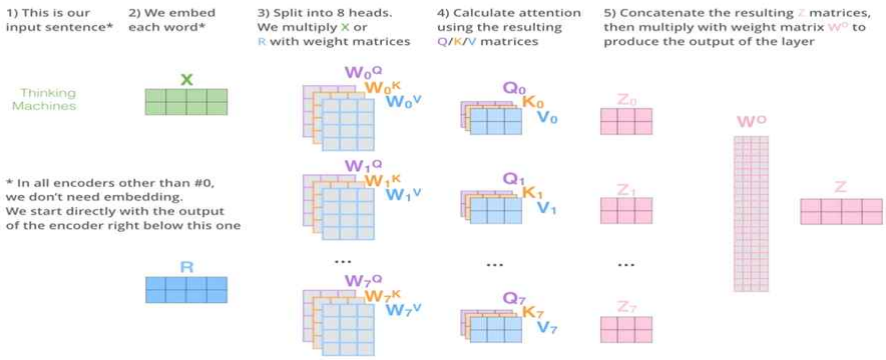
인코더는 여러 개의 인코더 스택으로 구성되어 있다. 논문에서는 6개의 인코더 스택이 사용된다. 인코더는 입력 시퀀스를 처리하는데, 모든 시퀀스를 한 번에 처리하는 언마스크(unmasked) 방식을 사용한다. 디코더는 인코더와 유사한 구조를 가지고 있지만, 순서에 따라 생성하는 과정에서 특정 단어를 예측하기 위해 이전에 생성된 단어를 참조해야 하는 특징이 있다. 이러한 특징으로 인해 디코더에서는 마스킹(masking)이 사용된다. 마스킹은 이후 단어를 미리 보지 않도록 하여 적절한 예측을 할 수 있도록 도와준다.



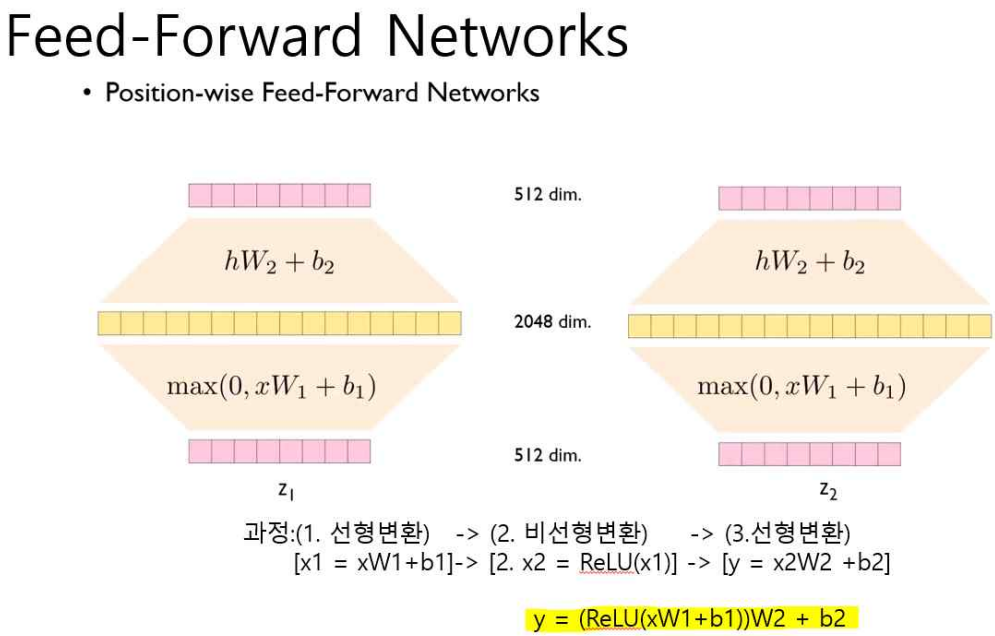
인코더에서 4개의 토큰을 처리할 때, 각 토큰에 대한 정보를 계산하는 셀프 어텐션 레이어를 사용한다. 이 레이어는 주어진 입력 시퀀스 내의 다른 단어들 간의 상호작용을 계산하여 각 단어의 중요성을 판단한다. 이후 Feed Forward 네트워크를 통해 각 단어에 대한 추가적인 처리를 수행하고, 이를 통해 디코더에서 단어를 생성한다. 만약 4번째 단어를 생성한다면, 이전 단어들을 마스킹하여 현재 단어 생성에 영향을 주지 않도록 합니다. 이 과정은 masked self-attention으로 알려져 있다.

디코더는 셀프 어텐션을 수행한 후, 그 결과를 이용하여 인코더에서 제공된 정보와의 어텐션을 계산한다. 이후, 이러한 정보를 결합하여 최종적으로 Feed Forward 네트워크를 통해 단어를 생성한다.

- **Multi-headed attention**



Multi-Head Attention은 여러 어텐션 헤드를 동시에 사용하여 입력 시퀀스의 서로 다른 부분에 주의를 기울일 수 있다. 예를 들어, 어떤 헤드는 문법적 관계에 주의를 기울일 수 있고, 다른 헤드는 문맥적 의미에 더 주의를 기울일 수 있다. 이를 통해 모델은 다양한 학습을 수행할 수 있으며, 병렬로 계산되는 여러 어텐션 헤드는 더 많은 정보를 동시에 처리할 수 있도록 해주어 긴 문맥을 처리하는 데 유리하다. 또한, 단일 헤드에서 발생할 수 있는 편향이나 과적합을 줄이는 데에도 도움이 된다. 이러한 이유들로 인해 Multi-Head Attention은 효율적인 정보 처리와 모델의 성능 향상에 기여한다.



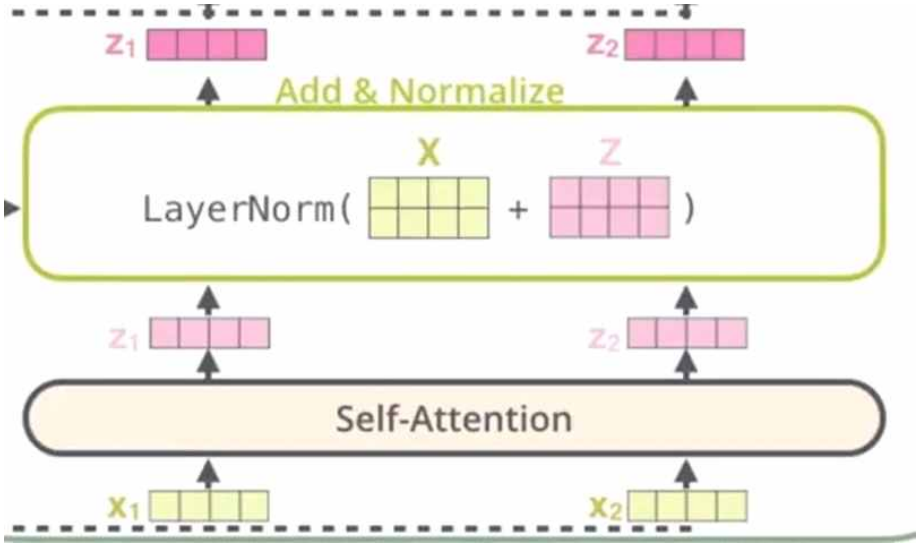
Feed-Forward Networks는 각각의 포지션에 대해 개별적으로 적용된다. Transformer 모델은 일반적으로 512개의 토큰을 사용하며, 이러한 토큰을 처리하기 위해 초기 512차원 벡터가 Feed-Forward Networks를 거치면서 2048차원의 히든 벡터로 표현된다. 이때, 선형 변환을 통해 입력 벡터를 먼저 2048차원으로 변환하고, 그 다음 비선형성을 추가하기 위해 ReLU 함수를 적용한다. 마지막으로, 다시 512차원으로 차원을 축소하기 위해 또 다른 선형 변환을 적용한다. Feed-Forward Networks의 과정은 다음과 같다.

선형 변환: 입력 벡터에 가중치 행렬과 편향을 곱하여 새로운 고차원 표현을 얻는다. (x1 = xW1+b1)

비선형 변환: ReLU 함수를 적용하여 비선형성을 추가한다.(x2 = ReLU(x1))

다시 선형 변환: 이전 단계의 결과에 다시 가중치 행렬과 편향을 곱하여 최종 출력을 얻는다.(y = x2W2 +b2) 최종 식은 y = (ReLU(xW1+b1))W2 + b2로 표현된다.

인코더에서의 Feed-Forward Networks는 입력 시퀀스를 고차원 표현 벡터로 변환하는 과정에서 복잡한 패턴을 학습하고, 중요한 특성을 추출하여 강화하며, 비선형성을 추가하여 정교한 표현을 학습한다. 디코더에서의 Feed-Forward Networks는 인코더로부터 받은 컨텍스트 벡터와 현재까지 생성된 출력 시퀀스를 기반으로 복잡한 패턴을 학습하고, 중요한 정보를 통합하여 최종 출력 시퀀스를 생성하며, 비선형성을 추가하여 더 의미 있는 출력을 만든다.



마지막으로 Residual Connections와 Layer Normalization은 모델의 학습을 돕는 두 가지 중요한 기술이다. Residual Connections(잔차 연결)은 입력 정보를 다음 레이어에 직접 전달하여 기울기 소실 문제를 완화한다. 이는 셀프어텐션 등의 레이어를 통과하면서 발생할 수 있는 기울기 소실 문제를 해결하여, 역전파 시 기울기가 더 잘 전달되게 한다. f'(x)+1 처럼 전달이 되지 못할 작은 값이 나오더라도 1만큼은 흘려주게 되므로 학습에서 상당히 유리하게 해준다. 잔차 연결을 통해 레이어 스택을 깊게 쌓을 수 있으며, 이는 모델의 표현력을 향상시키고, 더 깊은 네트워크를 구성할 수 있게 한다. Layer Normalization은 잔차 연결 결과에 대해 층별 정규화를 적용하여, 각 레이어의 출력 분포를 정규화한다. 이를 통해 내부 공변량 변화(Internal Covariate Shift)를 줄이고, 모델의 학습 안정성을 향상시킨다. 또한 배치 크기에 관계없이 각 레이어의 출력을 정규화할 수 있어 작은 배치나 배치 크기가 1인 온라인 학습에도 유리하다. 이는 학습 속도를 향상시키고, 더 빠르고 안정적인 수렴을 도와준다. 이러한 두 가지 기술을 결합한 "Add & Normalize"는 모델의 학습을 안정화시키고, 더 효율적으로 학습할 수 있도록 도와준다.

- 참고문헌

1. Attention Is All You Need, Ashish Vaswani, et al., 2017
2. https://jalammar.github.io/illustrated-gpt2/, The Illustrated GPT-2 (Visualizing Transformer Language Models)
3. https://huggingface.co/learn/nlp-course/chapter1/4, 허깅페이스 NLP Course
4. https://www.youtube.com/watch?v=Yk1tV_cXMMU, Transformer
5. https://ffighting.net/deep-learning-paper-review/language-model/transformer/, [17′ NIPS] Transformer : Attention Is All You Need