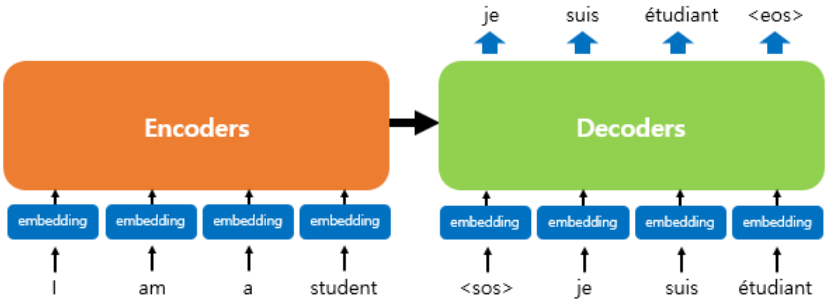
Transformer

Transformer

: 2017년 구글의 Attention is All You Need 논문에서 처음 발표된 모델. 기존의 seq2seq 모델을 개선하기 위해 어텐션을 RNN을 보정하기 위한 용도로서 사용하는 것이 아니라, 어텐션만으로 인코더와 디코더를 만들어낸 모델

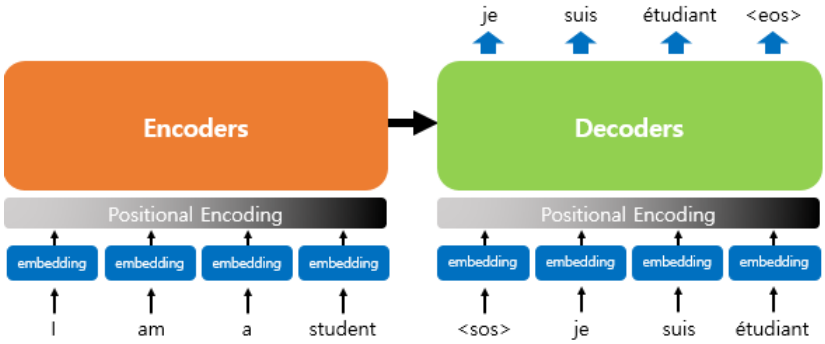
- 이전 seq2seq 구조: 인코더와 디코더에서 각각 하나의 RNN의 t개의 time step을 가지는 구조
- 트랜스포머: 인코더와 디코더 단위가 N개로 구성되는 구조. Attention is All You Need 논문에서는 각 6개씩 사용



N개의 인코더가 정보를 전달받아 N개의 디코더가 출력 결과를 만들어내는 구조

- 기존의 seq2seq처럼 <sos>를 입력으로 받아 <eos>가 나올때까지 연산을 진행 (인코더-디코더 구조)
- 트랜스포머는 각 단어의 임베딩 벡터를 입력받는 것이 아니라 임베딩 벡터에서 조정된 값을 입력받는다.

Positional Encoding



- RNN은 단어를 순차적으로 입력받음으로써 자연스레 각 단어의 위치 정보 (position information)를 가질 수 있지만, 트랜스포머는 다른 방식으로 위치 정보를 얻는다.
- 단어의 위치 정보를 얻기 위해 각 단어의 임베딩 벡터에 단어의 상대적 혹은 절대적 위치에 대한 정보를 더해주는 것을 **positional encoding**이라고 한다.

임베딩 벡터들이 입력으로 사용되기 전, 포지셔널 인코딩의 값이 더해진다

위치 정보를 가진 값을 만들기 위해 사용하는 함수들

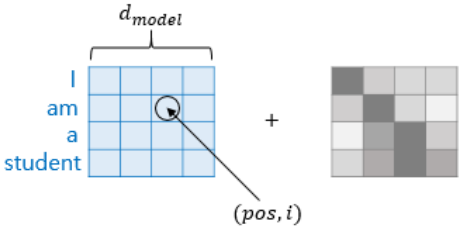
$$PE_{(pos,\ 2i)} = sin(pos/10000^{2i/d_{model}})$$

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

▼ Parameters

- pos=입력 문장에서의 임베딩 벡터의 위치
- i=임베딩 벡터 내의 차원의 인덱스
- d_{model}=트랜스포머의 모든 층의 출력 차원. (논문에서는 512의 값을 가짐)

- **Sinusoid funciton**인 사인 함수와 코사인 함수 의 값을 임베딩 벡터에 더해주면 단어의 순서 정보가 주어진다. 요동치는 값을 나타내는 사인/코사인 함수 특성상 모델이 단어의 상대적인 포지션을 더 쉽게 학습할 수 있게 된다고 함
  - 각 차원의 인덱스가 짝수인 경우, 즉 $(pos,\ 2i)$일 때는 사인 함수를 사용하고 홀수인 경우, 즉 $(pos,\ 2i+1)$일 때는 코사인 함수 사용
- 임베딩 벡터와 포지셔널 인코딩의 덧셈은 아래 그림처럼 임베딩 벡터가 모여 만들어진 문장 행렬과 포지셔널 인코딩 행렬의 덧셈 연산을 통해 이루어진다



- 포지셔널 인코딩 방법을 사용하면 같은 단어라고 하더라도 문장 내의 위치에 따라서 트랜스포머의 입력으로 들어가는 임베딩 벡터의 값이 달라진다. 즉 순서 정보가 고려된 임베딩 값을 입력값으로 받을 수 있게 된다

Transformer Model Architecture

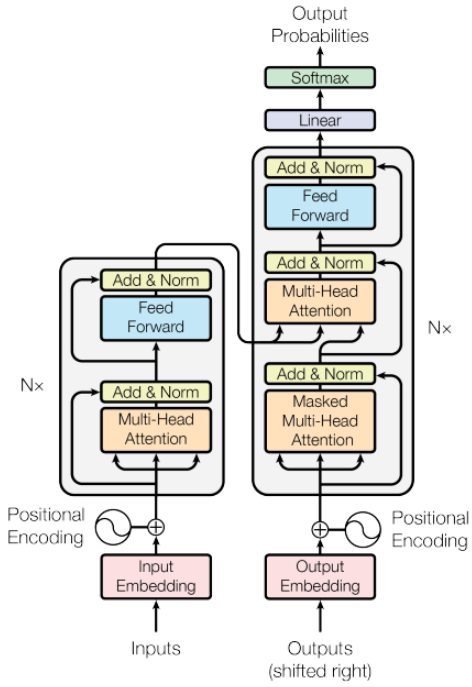


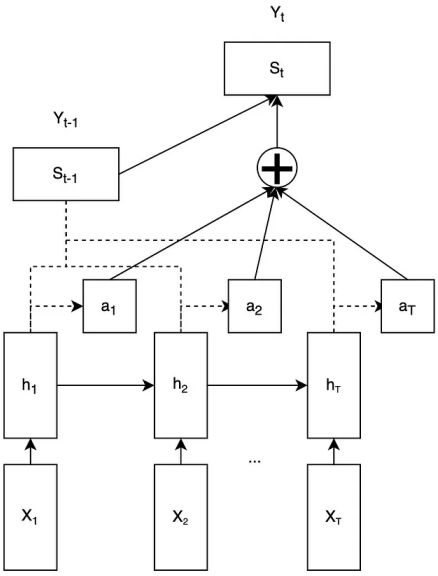
Figure 1: The Transformer - model architecture.

- 각 인코더 레이어는 두 가지의 서브층 1) multi-head self attention과 2) position-wise feed forward network로 구성되어 있다.
- 각 디코더 레이어는 3) masked multi-head attention 이 추가된 세 가지의 서브층으로 구성되어 있다

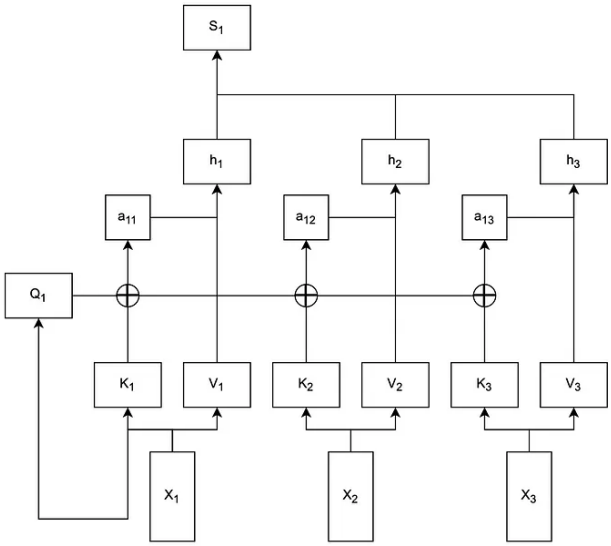
💡 어텐션 함수: 1) query에 대해 모든 key와의 유사도 구하기를 구하고 2) 해당 유사도를 키와 매핑된 각각의 value에 반영한 후 3) 반영된 값을 모두 더해 attention value 리턴

Self-Attention

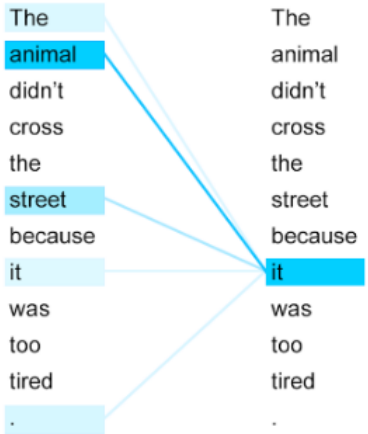
: 어텐션을 자기 자신에게 수행한다는 의미로, query/key/value가 모두 같은 입력 문장에서 생성되어 입력 시퀀스끼리 상호작용이 가능하다. Intra-attention이라고 불리기도 함.



아웃풋 출력시 인풋을 참고하는 기존 attention



같은 시퀀스 내에서 참고하는 self attention

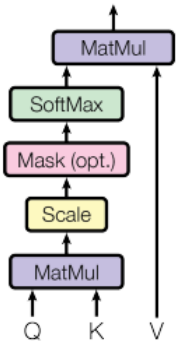


셀프 어텐션을 사용하면 위 문장에서 'it'이 'animal'에 해당할 확률이 높다는 것을 찾아낼 수 있다

Scaled Dot-Product Attention

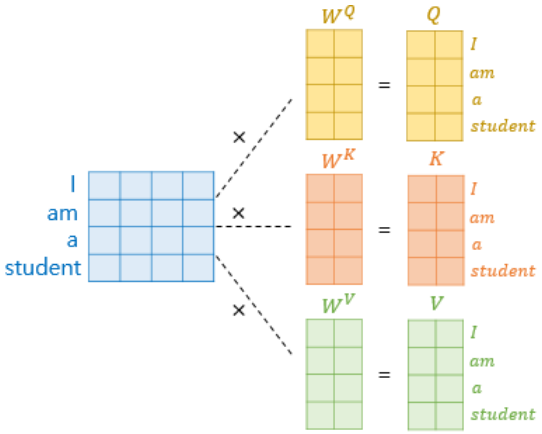
: 어텐션 함수로 가장 많이 사용되는 additive attention & dot-product 기법 중 dot-product attention에 값에 대한 스케일링을 추가하는 방법

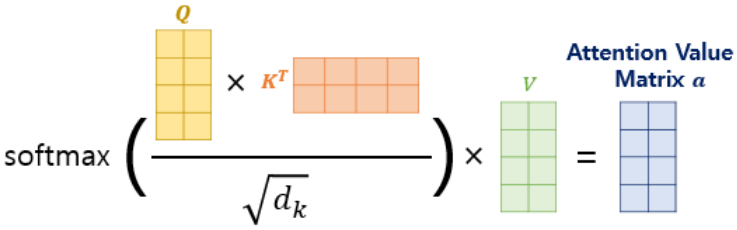
Scaled Dot-Product Attention



$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

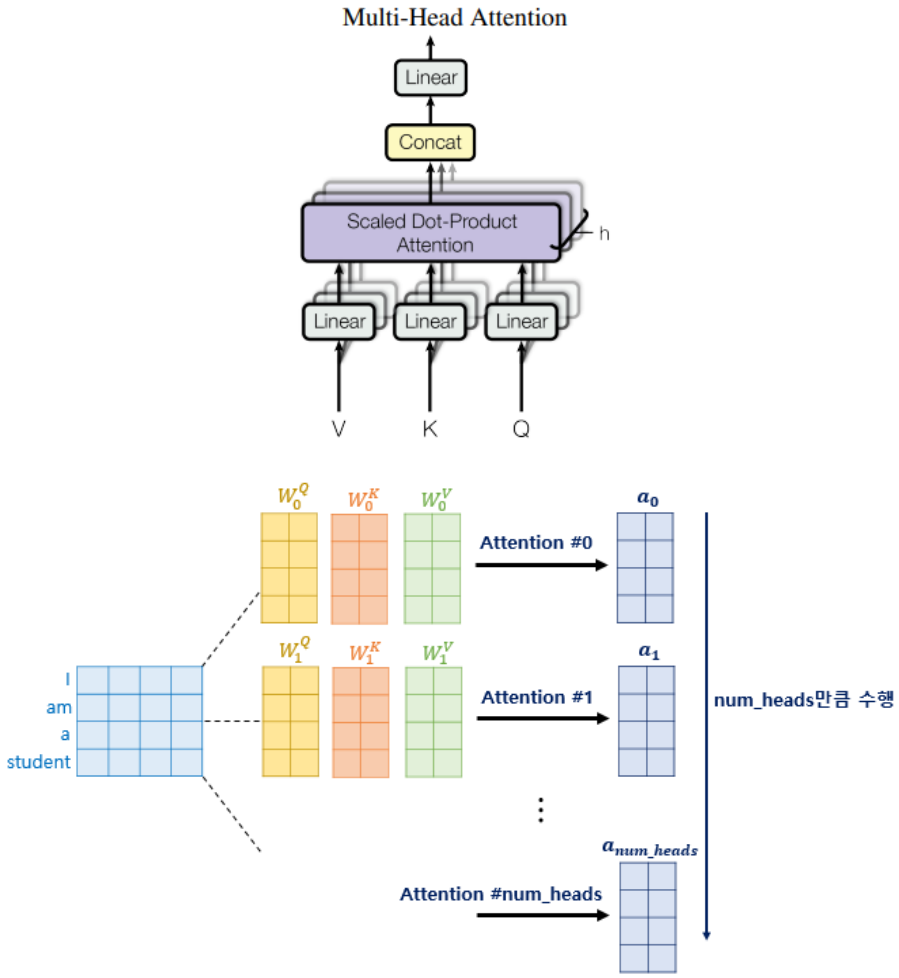
- d_k, 측 key의 차원이 커질수록 additive attention보다 dot product attention의 성능이 나빠지는 것을 보완하기 위해 scale ($\frac{1}{\sqrt{d_k}}$) 적용
- 먼저 문장 행렬에 가중치 행렬을 곱하여 Q행렬, K행렬, V행렬을 구한다. Q행렬과 전치된 K행렬을 곱해주어 스케일링을 적용하면 어텐션 스코어로 구성된 행렬이 만들어진다. 이 행렬에 소프트맥스 함수를 적용하면 어텐션 분포가 구해지는데 여기에 V행렬을 곱하면 각 단어의 어텐션 값을 모두 가지는 행렬이 결과로 나온다.





Multi-Head Attention

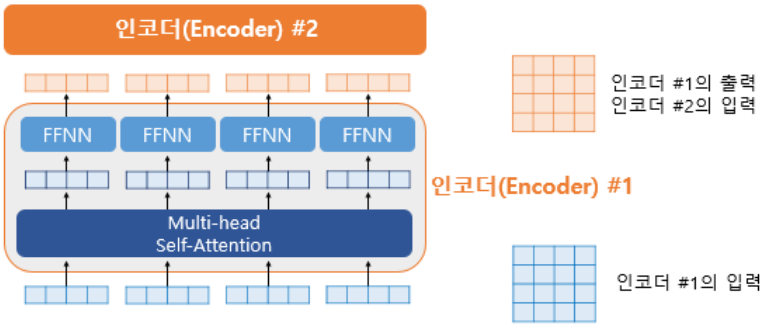
: 인코더의 첫번째 서브층 단계. 어텐션을 한번만 하는 것이 아닌, 여러번의 어텐션을 병렬로 사용하여 다시 하나로 합치는 방법. 논문에서는 병렬의 개수 (num_heads)를 8로 지정



- 어텐션을 병렬로 수행하면 다양한 시각에서 정보를 수집할 수 있다. 예를 들어 “The animal didn’t cross the road because it was too tired”라는 문장에서 it이 쿼리였다고 하면, it과 animal의 연관도를 높게 보는 어텐션 헤드가 있는 반면 it과 tired의 연관도를 높게 보는 헤드도 있을 것

Position-wise Feed Forward Neural Network

: 인코더와 디코더에서 공통적으로 가지고 있는 서브층. 은닉층이 1개이고 활성화 함수로 ReLU를 이용한 완전연결계층 (fully connected layer) 신경망을 이용함

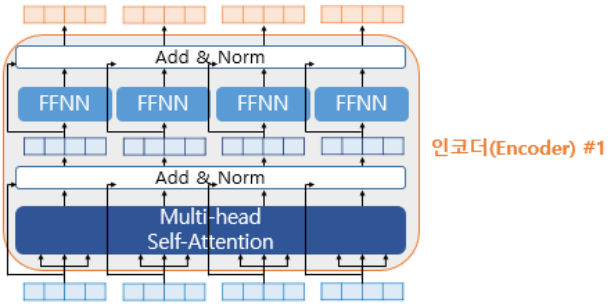


- 인코더의 입력은 첫번째 서브층인 멀티 헤드 어텐션 층을 지나 FFNN을 통과한다. 두번째 층을 지난 최종 출력은 인코더의 입력 크기를 보존하며 다음 인코더로 넘어간다

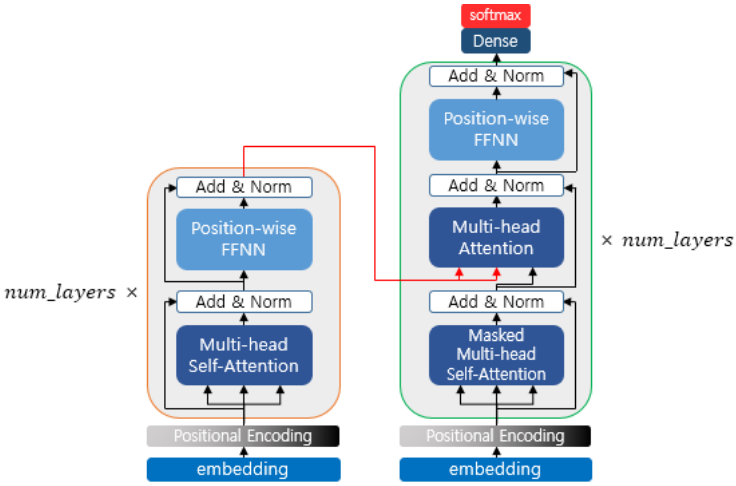
$$FFNN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

▼ Parameters

- x=멀티 헤드 어텐션의 결과로 나온 행렬
- W_1, W_2=가중치 행렬
- 인코더 내에서는 같은 값을 가지고, 인코더 층마다는 다른 값을 가짐



- 각 서브층을 지날 때 마다 입력과 출력을 더하는 **잔차 연결 (residual connection)**을 거친 후 **정규화 (layer normalization)** 연산 진행



- num_layers (인코더를 쌓은 횟수; 논문에서는 6) 만큼 연산을 거친 후 마지막 층의 인코더의 출력은 디코더로 전달된다

Masked Multi-Head Attention

: 디코더의 첫번째 서브층

- 이때 트랜스포머는 RNN 계열의 신경망과 달리 디코더가 문자 행렬을 입력으로 한꺼번에 받으므로, 현재 시점의 단어를 예측해야 하는데 미래 시점의 단어까지도 알 수 있는 현상이 발생한다. 이를 위해 “미리보기에 대한 마스크”인 **룩-어헤드 마스크 (look-ahead mask)** 를 도입했다.
- 현재 자기 자신보다 미래에 있는 단어는 참고하지 못하도록 어텐션 스코어 행렬에서 마스킹을 적용한다는 점만 빼면 인코더의 멀티 헤드 어텐션과 동일한 연산이 이루어짐

Conclusion

- 트랜스포머는 인코더 디코더 아키텍처의 recurrent layer를 멀티 헤드 셀프 어텐션으로 대체하여 어텐션 만으로 시퀀스 번역을 성공적으로 이루어낸다
- 트랜스포머를 이용하면 계산량을 줄이고 GPU를 이용한 병렬 계산의 혜택도 더 많이 누릴 수 있다. 그 결과 구글이 2016년에 발표한 RNN 기반 신경망 번역 시스템 (GNMT)보다 학습 시간을 대폭 줄이고 번역 품질을 끌어올릴 수 있었다
- 구글은 트랜스포머를 활용해 텍스트 뿐만 아니라 이미지, 비디오 등 다른 모달리티를 다루는 연구도 진행중
- 트랜스포머 연구가 시사하는 것 처럼 어텐션은 RNN을 대체할 수 있으므로 어텐션을 이용할 기회가 늘어날 것

참고

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017)

https://wikidocs.net/31379

https://medium.com/mlearning-ai/whats-the-difference-between-self-attention-and-attention-in-transformer-architecture-3780404382f3

밑바닥부터 시작하는 딥러닝2