

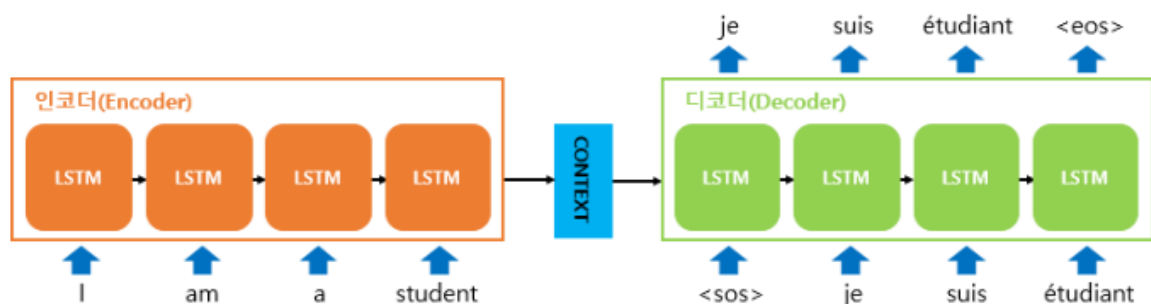
Transformer

Attention is all you need, 2017 논문에서 나온 모델

기존의 Seq2Seq 구조인 encoder-decoder 형식을 따름

특이점 : Attention 만으로 모델을 구현

1. Attention 배경 설명



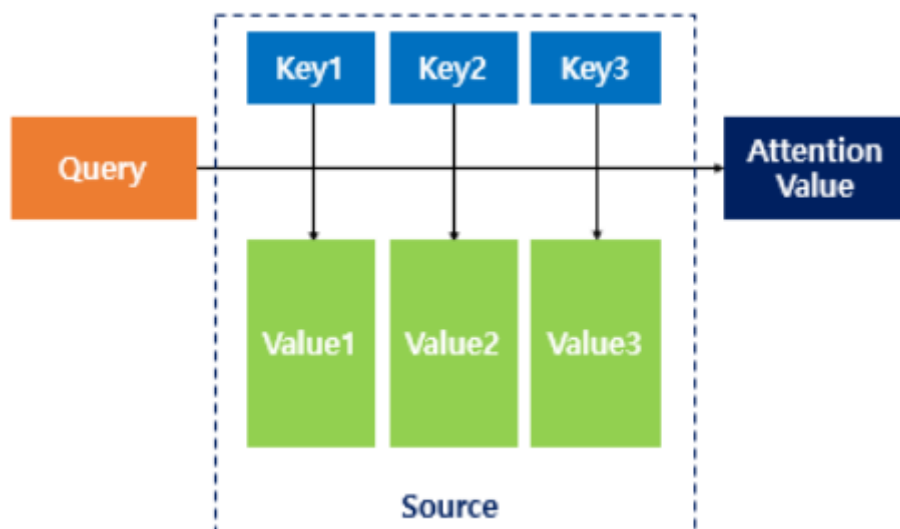
본 그림의 기존의 번역 Task를 수행하는 Encoder-Decoder, 즉 Seq2Seq 모델 중 하나임.

기존 방식의 문제점

1. Context Vector에 많은 Load가 실리게 됨. (정보의 손실이 발생함)
2. Vanishing gradient 문제 발생

이를 해결하기 위해 Attention concept이 등장함

Attention

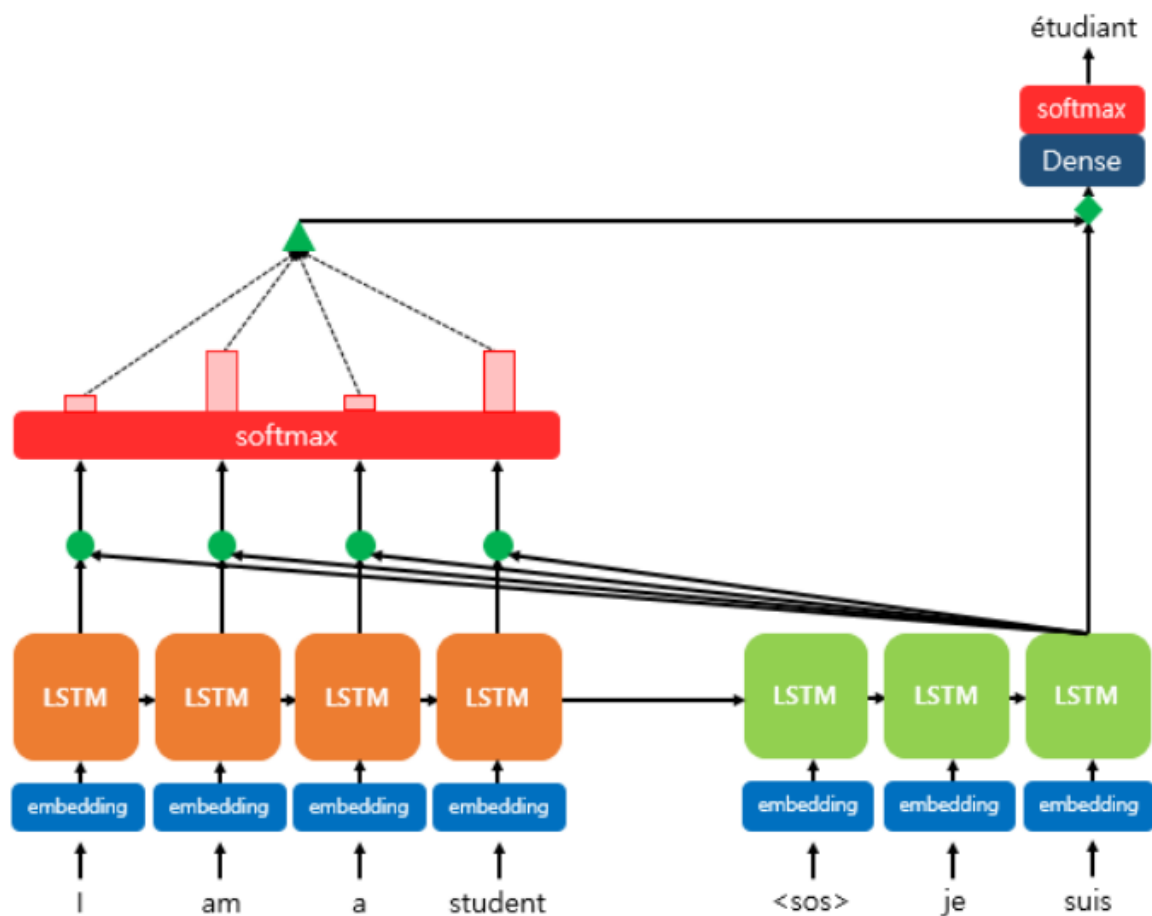


$\text{Attention}(Q, K, V) = \text{Attention Value}$

Query와 Key 간의 유사도를 계산한 후에 이를 Value에 반영해서, 이들을 다 더해서 Attention value를 계산해냄.

여러 가지 종류의 Attention이 있지만, Dot Product Attention을 위주로 설명하겠습니다.

Dot Product Attention



기존의 seq2seq 모델과 동일한 형태에 Attention 개념이 들어가게 된 것임.

Attention이란 Decoder의 각 step의 hidden state를 Query로 두고, Encoder의 각 step의 output을 Key, Value로 둔다

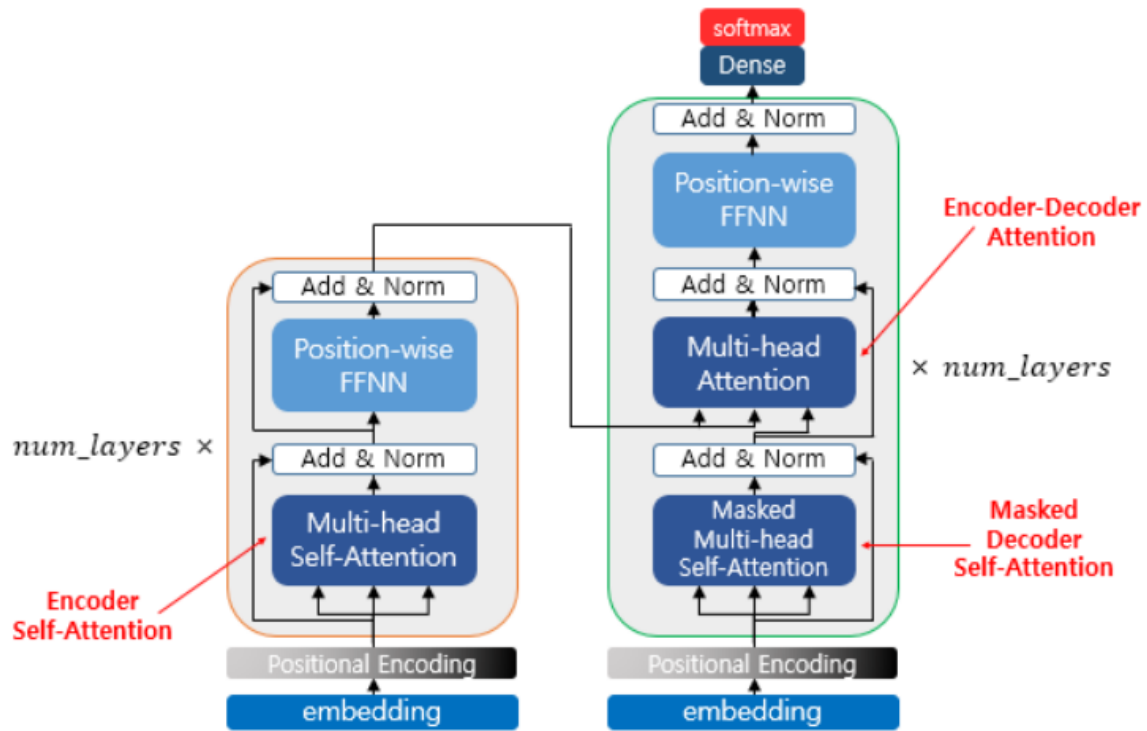
식으로 표현하면 Attention value = $\text{Softmax}(Q \cdot K^T) \cdot K$

이를 의미적으로 풀어내면, suis를 예로 들어보자. suis와 encoder의 값들(영어) 간의 유사도(Attention score)를 계산한 후에, 이에 softmax를 취한다.(Attention Distribution) 그리고 이 값들을 각각에 대응되는 Value 값에 곱한 후에 이를 다 더한다. 이렇게 되면 suis와 유사한 영어 단어의 유사도를 반영한 영어 단어들의 합으로 결과값을 산출하게 됨.

이렇게 Attention을 활용하게 되면 Context vector에 많은 Load가 실리는 문제를 해결할 수 있고, vanishing gradient에 관해서는 Attention value는 먼 거리에 있는 값들에 대해서도 동일하게 적용될 수 있기 때문에 보완해낼 수 있다.

Transformer는 이러한 Attention 만을 가지고 Seq2Seq 모델을 구현해보자라는 것이다

2. Transformer



Transformer의 핵심은, **Self Attention**과 **Multi head attention**이라고 할 수 있다.

Attention

Transformer는 Attention만으로 구성된 Seq2Seq 모델이라고 할 수 있다.

기존의 Seq2Seq 모델의 경우 RNN+Attention구조였는데, 이러한 RNN의 경우엔 Auto regressive 성질을 지니고 있기 때문에 병렬처리가 힘들다. 이렇게 되면 GPU 등을 효율적으로 활용할 수 없기 때문에 Attention 만으로 모델을 만들어보자라고 해서 나온 것이 Transformer이다.

본 논문에서 나오는 Attention의 종류는 3가지이다.

1. Self Attention (Encoder) -> BERT
2. Encoder-Decoder Attention
3. Masked Self Attention (Decoder) -> GPT2,3

1. Self Attention은 말 그대로, Query, Key, Value가 동일한 source에서 나온 값이다.

ex) 나는 학교에 간다

	1	2	3	4	5
나는					
학교에					
간다					

	나는	학교에	간다
1			
2			
3			
4			
5			

	나는	학교에	간다
나는			
학교에			
간다			

	1	2	3	4	5
나는					
학교에					
간다					

softmax ↓

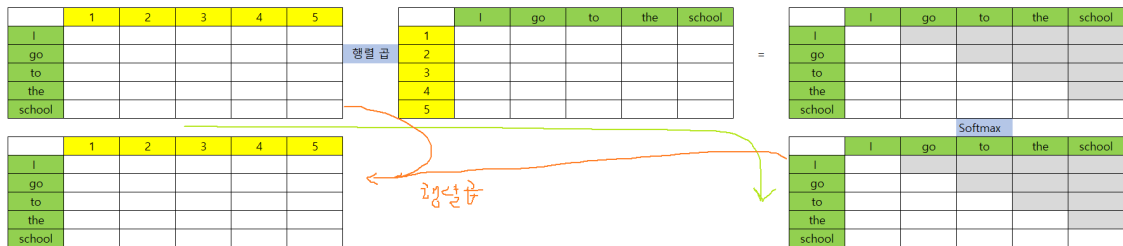
임베딩 차원을 5라고 한다면, 여기에서 Self Attention 중에서 Attention을 scaled dot product로 한다면 $\text{Softmax}(\mathbf{QK.T})\mathbf{V}$ 에서 $\mathbf{QK.T}$ 의 결과가 맨 우측의 값이 된다.

이제 여기에 Softmax를 취하고 V와 행렬 곱을 해주면, 결과가 나오게 된다.

의미론 적으로 본다면, 각 Token들이 자기 자신을 포함한 다른 Token 간의 유사도를 반영한 임베딩 벡터의 합이라고 할 수 있겠다.

2. Encoder - Decoder Attention은 Query가 Decoder의 Output이고, Key, Value가 Encoder의 Output인 기존 RNN+Attention에서 Attention이 했던 역할이라고 하면 된다. 즉 Decoder에서 output을 내기 위해서 Encoder 부분을 참조한다는 의미로 보시면 될 것 같다.

3. Masked Self Attention은 Decoder 부분에서 쓰인다.



예시, I go to the school

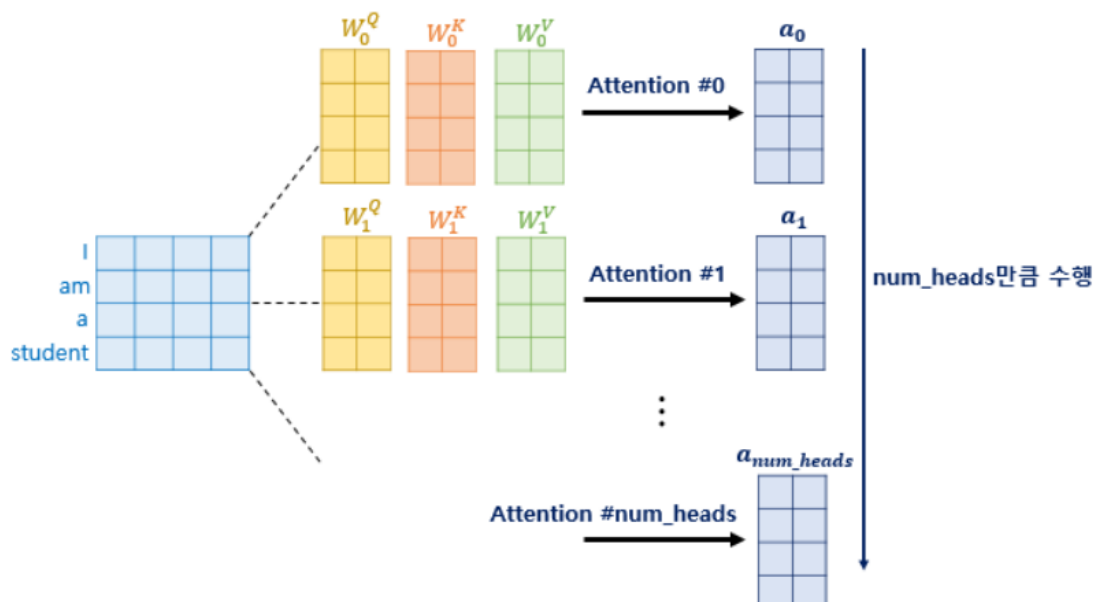
실제로 decoder에서 I에 해당되는 token을 넣었을 때에 I는 뒤에 따라오는 문장에 대한 정보는 없다. 이렇기 때문에 mask를 씌우게 된다. mask는 즉, 다음에 올 문장에 대한 정보를 못보게 하기 위함이다.

첨언

BERT는 Transformer의 Encoder 부분으로만을 활용해서 word representation을 구성한 것이고, GPT는 Transformer의 Decoder 부분으로만 활용해서 word representation을 구성한 것이다.

Multi Head Attention

Multi Head Attention은 다양한 관점에서 Attention 작업을 수행하는 것으로서,



num_heads만큼 수행을 하게 됩니다. 그림을 보게 되시면 a0,a1, .. a_num_heads만큼 attention value가 나오게 됩니다. 이제 이를 concat 해서 최종적인 attention value를 산출하게 됩니다.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

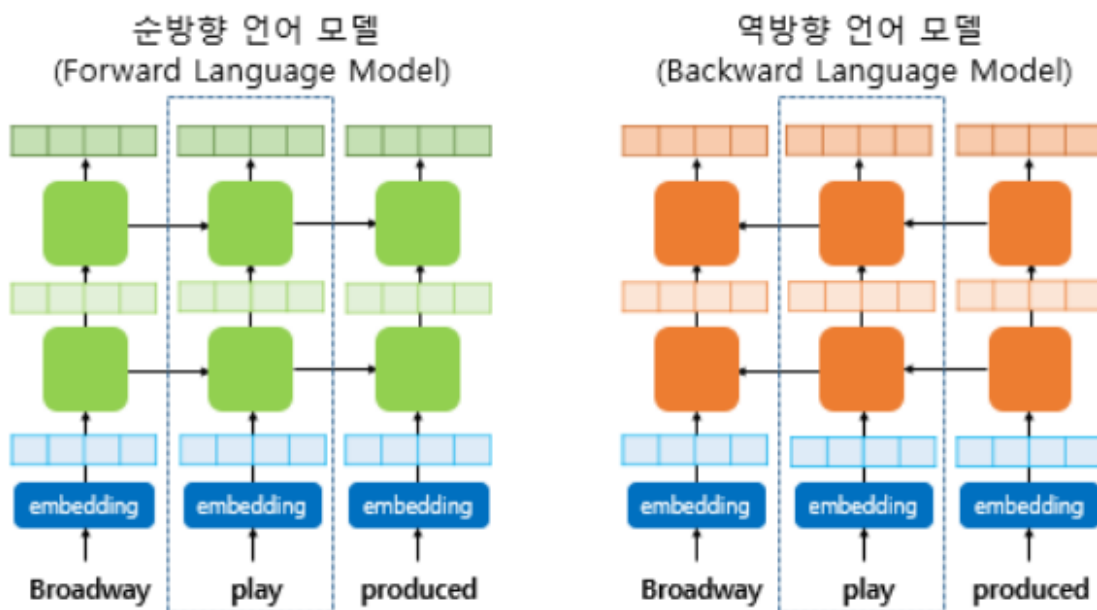
Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

추가

Word Representation -> 최근에는 Contextual Representation

Language modeling : 다음에 올 Token을 예측

$$P(w_1, w_2, \dots, w_n) = P(w_1) * P(w_2|w_1) \dots P(w_n|w_{n-1}, w_{n-2}, \dots, w_1)$$



최근에 나오는 Contextual Representation은 네모로 표시된 부분을 활용하는 방식으로 word representation을 한다.

- ELMO의 경우, 각 층 별로 concatenation을 진행하고, 각 층별로 가중합을 한다. 이제 여기에 scalar를 곱해서 하나의 word representation을 완성함.
- BERT,GPT의 경우 , Embedding part가 3가지로 구성됨 - token embedding / positional embedding / segment embedding