Attention is all you need

논문 리뷰

윤태우

Attention

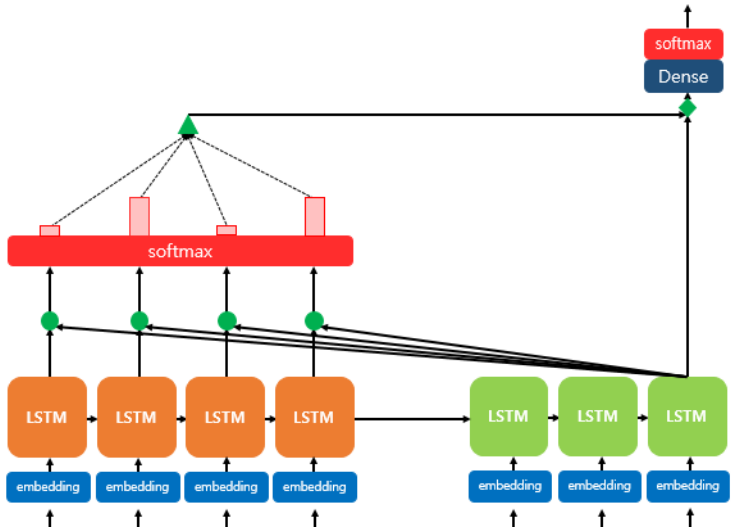
기본 아이디어는 디코더에서 출력 단어를 예측하는 매 시점마다 인코더에서의 전체 입력 문장을 한번 더 참고한다는 것입니다.

인코더에서 소프트맥스 함수를 통과하면, 각 단어가 예측에 미치는 영향을 합이 1인 수들로 각각 나타내어 집니다. 이 값을 디코더의 출력과 결합(concatenate) 하여

최종 예측을 하게 됩니다.

(즉, 예측 해야 할 단어와 연관이 있는 부분을

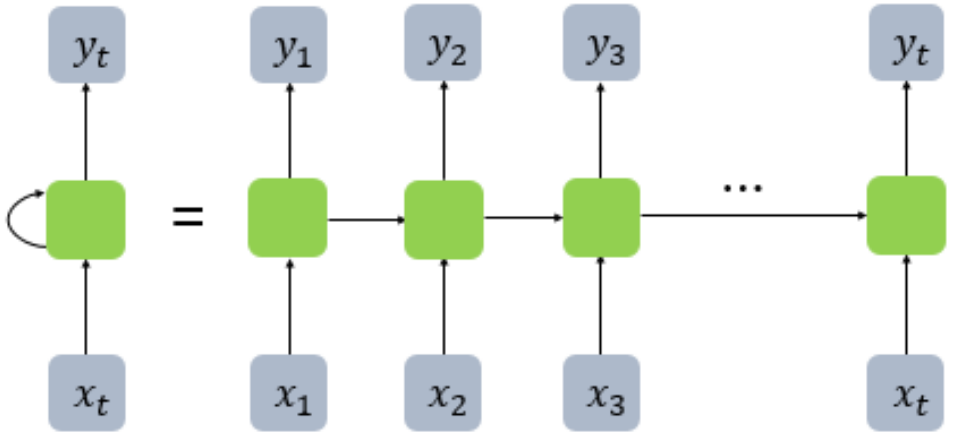
좀 더 집중적으로 참고하게 됩니다)



Transformer 모델은 논문 제목 attention is all you need 에서 알 수 있듯 attention만으로 구현한 모델입니다.

1. Introdution

Recurrent 모델들은 언어 모델링, 기계 번역 등과 같은 언어 변환문제에서 많은 연구와 노력으로 정립되어 왔습니다. 그러나 이 모델들이 hidden state를 계산하기 위해서는 t-1,t-2,t-3,...등 많은 hidden state가 필요했습니다. 때문에 기존 RNN 구조에서는 병렬화가 불가능했죠. Computation 과정에서 전 단계의 정보가 필요하니까요.



https://wikidocs.net/60690

1. Introdution

또한 인코더가 입력 시퀀스를 하나의 벡터로 압축하는 과정에서 입력 시퀀스의 정보가 일부 손실된다는 단점이 있었고, 이를 보정하기 위해 어텐션이 사용되었습니다. 어텐션은 위치 정보에 관계없이 종속성을 모델링할 수 있습니다.

논문에서는 인풋과 아웃풋 사이의 글로벌 의존성을 끌어내기 위해 어텐션 메커니즘에 전적으로 의존했고, 결과적으로 Transformer는 훨씬 더 많은 병렬화가 가능하게 되었습니다.

2. Background

ByteNet, ConvS2S 같은 모델들이 고안됐으나, 이것들은 거리가 먼 값들 사이의 의존성을 학습려면 전대수적으로, 선형적으로 연산량이 늘어난다는 한계가 있었습니다.

이에 반해, Transformer는 계산량을 매우 줄일 수 있었으나 attetion-weighted posiotion들을 평균 내버리는 바람에 effective 해상도를 희생하게 되었습니다.

이를 극복하기 위해서 논문에서는 Multi-Head Attention을 사용했습니다.

3. Model architecture

Transformer model은 RNN 또는 convolution을 사용하지 않고 전적으로 self-attention 메커니즘으로 동작하는 최초의 모델입니다.

transformer는 RNN을 사용하지 않지만 기존의 seq2seq과 같이 인코더-디코더 구조를 유지하고 있습니다.

오른쪽 그림이 transformer의 전체적인 구조입니다.

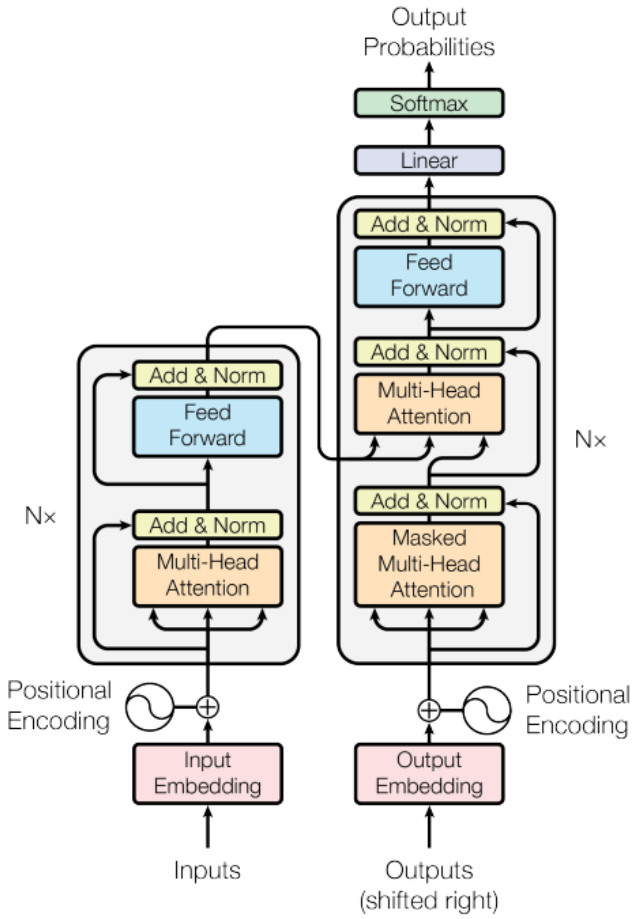


Figure 1: The Transformer - model architecture.

3. Model architecture(Encoder)

N은 인코더와 디코더를 하나의 층으로 생각 했을 때,

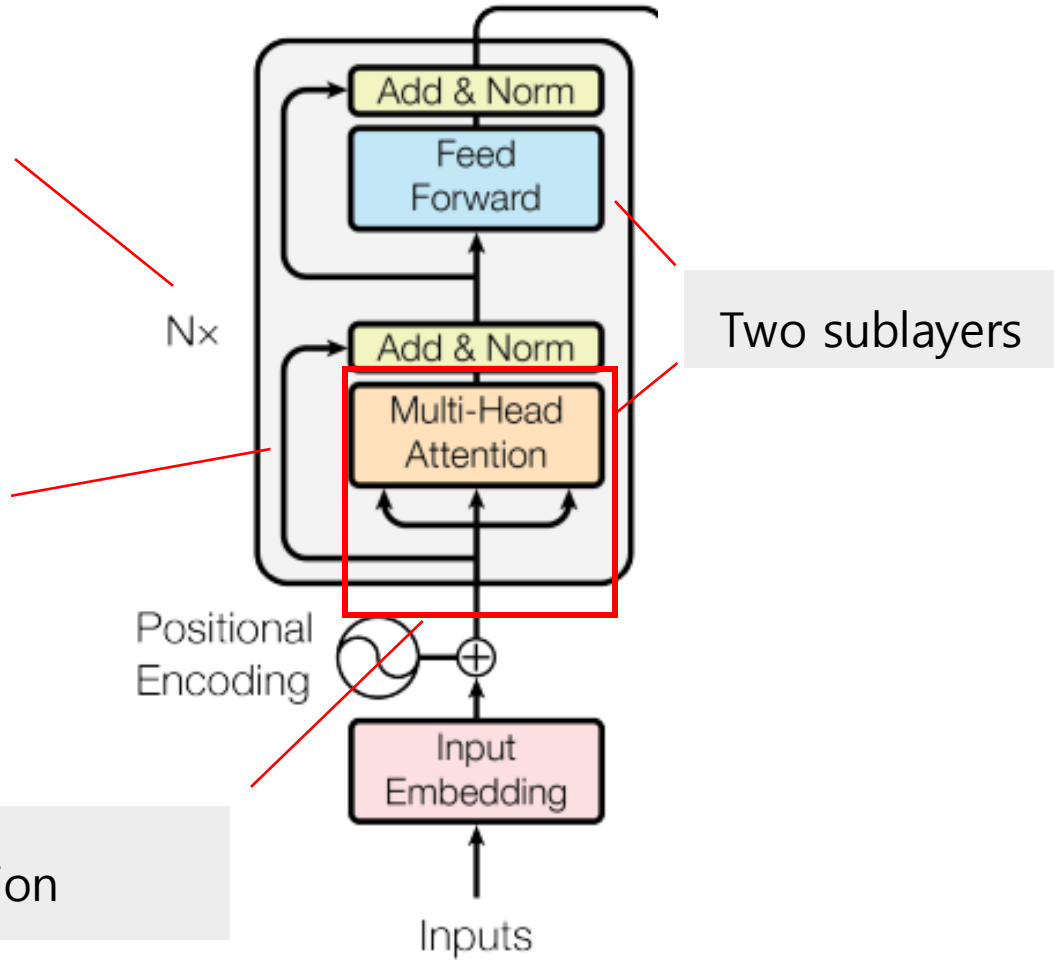
그 층의 개수 입니다. (N = 6 이면 6개의 인코더-디코더)

* 논문에서 N = 6

잔차(잔차 연결을 용이하게 하기 위해 각 sublayer들은 512의 입력과 출력을 가집니다.

* d_model : 입력과 출력의 크기

Muti-head self-attention

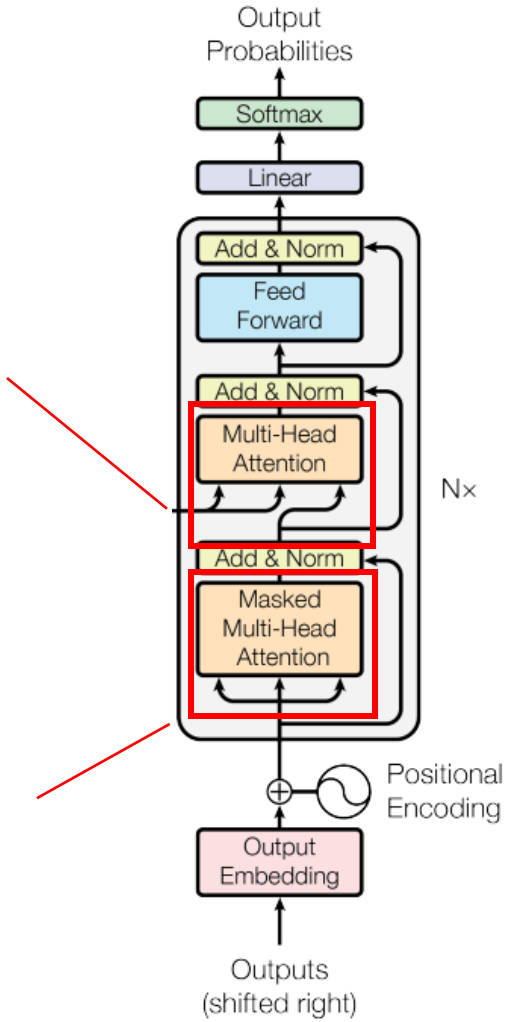


3. Model architecture(Decoder)

인접하지 않은 이전 값에 주의를

기울이지 않기 위해 encoder의 softmax 함수를 거친 값들을 attention 합니다.

Muti-head self-attention



Three sublayers

N = 6

잔차 (d_model = 512)

3. Model architecture(attention)

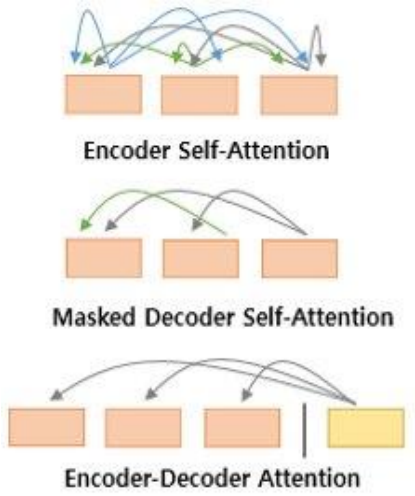
어텐션 함수는 key-value 쌍으로 이루어진 딕셔너리를 통해 정의합니다.

여기서 key, value, query 값은 모두 벡터입니다.

Output은 values에 가중치 합으로 계산되며, 각 가중치 값은 상응하는 key값에 query의 compatibility function에 의해 계산됩니다.

3. Model architecture(attention)

구조적으로 보면, encoder에 attention 하나, 디코더에 두개가 있는데, 각 attention은 오른쪽 그림과 같이 나타낼 수 있습니다.



Self-attention은 본질적으로 query, key, value가 모두 동일한 경우를 말합니다.
그러나 세번째 attention의 경우 query가 디코더의 백터인 반면, key,value가 인코더의 벡터이므로 self-attention이 아닙니다.

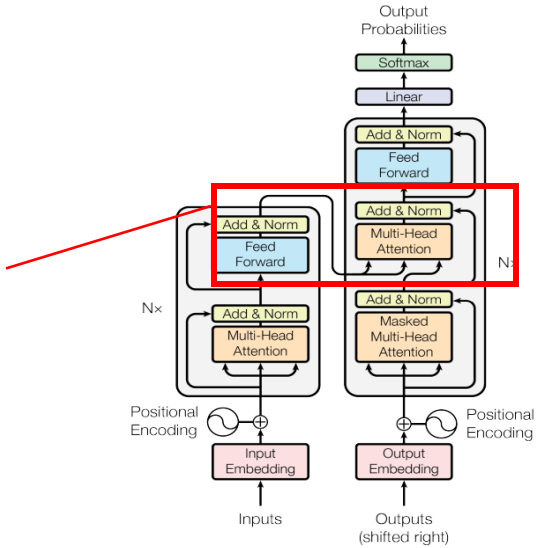
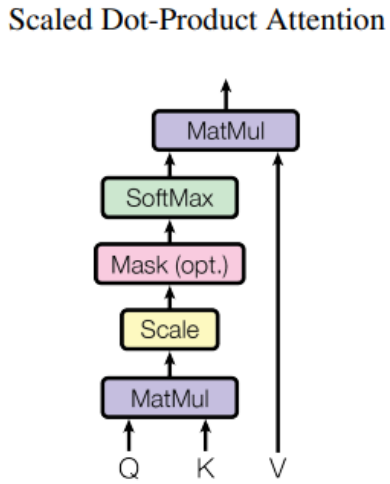


Figure 1: The Transformer - model architecture.

3. Model architecture(attention)

- Scaled Dot-Product Attention

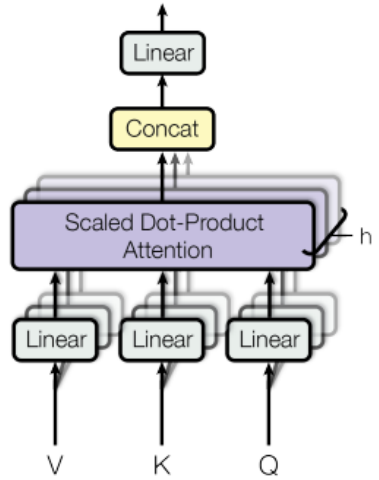
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



- Multi-Head Attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

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 $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.



4. Applications of Attention in our Model

- Position-wise Feed-Forward Networks

FFN은 encoder layer마다 독립적이지만 동일한 방식으로 적용되는데 두 번의 linear transformation과 활성화 함수 ReLU(:max 부분)를 거치게 됩니다. 다만 이때 각 layer마다 서로 다른 parameter를 사용합니다.

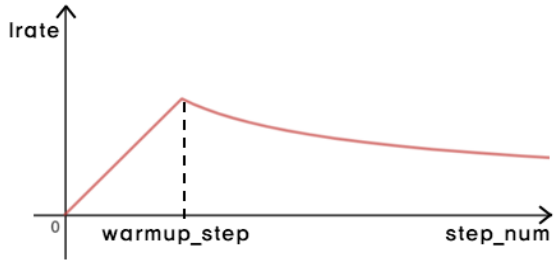
- Embeddings and Softmax

learned embedding을 이용해서 input 토큰과 output 토큰을 벡터로 바꾸고, 학습된 linear transformation 및 softmax 함수를 이용해서 decoder output을 예측된 다음 토큰의 확률을 계산하는데 이용했습니다.

5. Training

- optimizer

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$



- BELU score

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

6. Result

- Variations on the Transformer architecture

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

이상 transformer model의 주요한 아이디어 위주로 리뷰를 해 보았습니다.