Attention Is All You Need

나보영

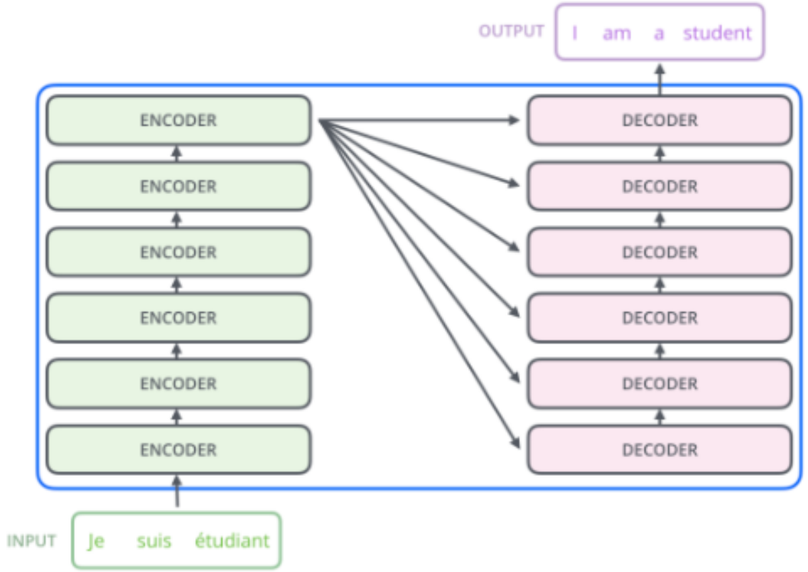
기존 모델 한계

- RNN 기반 모델의 경우 출력이 이전 상태와 현재의 입력에 의존 -> 병렬 처리 제한
- long short-term 학습 제한 -> 순차적 처리 인하여 입력이 길어지게 되면 초반에 넣었던 입력이 약해짐
- Attention mechanisms에서 여전히 RNN 또는 CNN 사용 -> 기존의 순차적 모델 구조를 완전히 대체 X

따라서

- attention mechanism을 전적으로 의존 -> 병렬 처리 O , 속도 향상

트랜스포머 구조



N개의 인코더와 디코더 사용(논문에서는 6개)

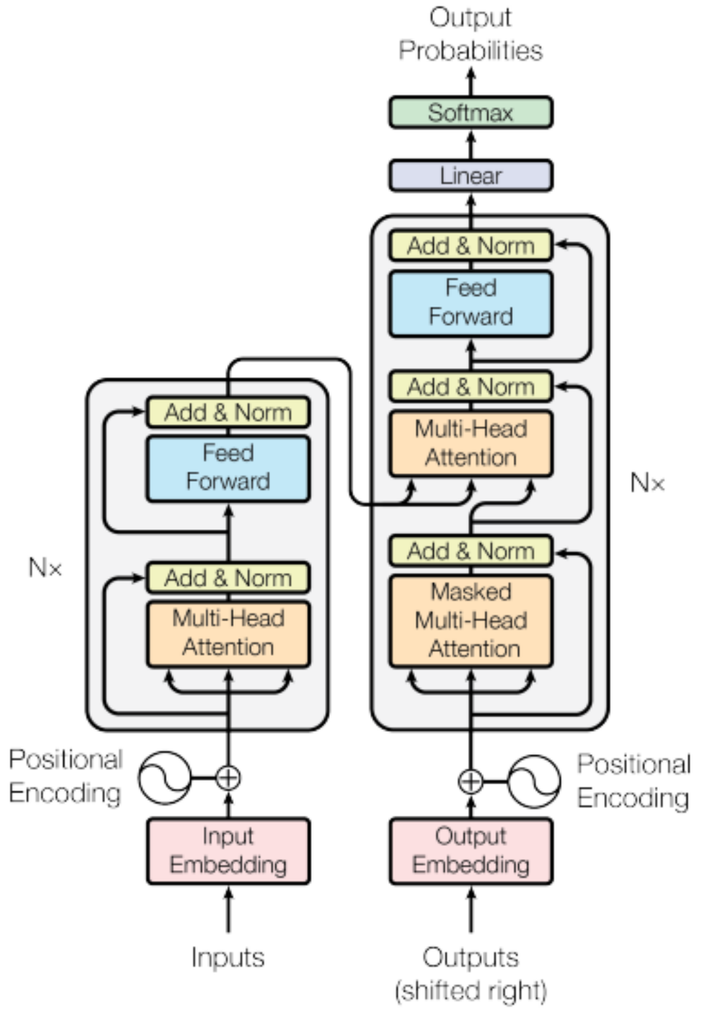
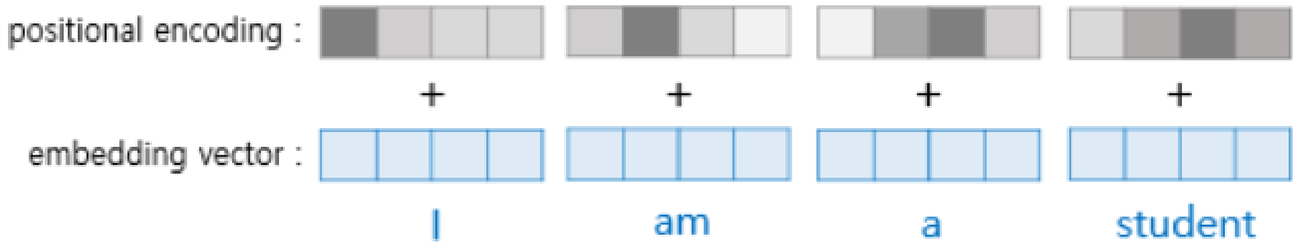


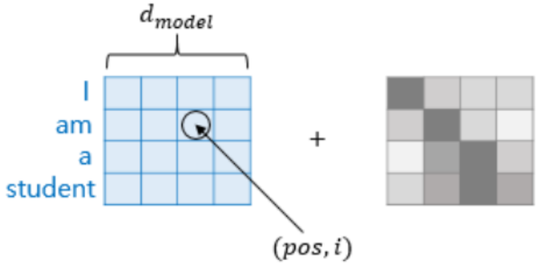
Figure 1: The Transformer - model architecture.

Positional Encoding(위치 인코딩)

RNN의 경우 단어의 입력을 순차적으로 받기 때문에 이로 인해 순서 정보를 얻을 수 있었다!

BUT 트랜스 포머의 경우 병렬도 문장의 단어를 입력 받기 때문에 위치 정보 X
따라서 각 단어의 인베딩 벡터에 위치 정보를 더하여 모델의 입력에 사용하여 이를 해결한다.

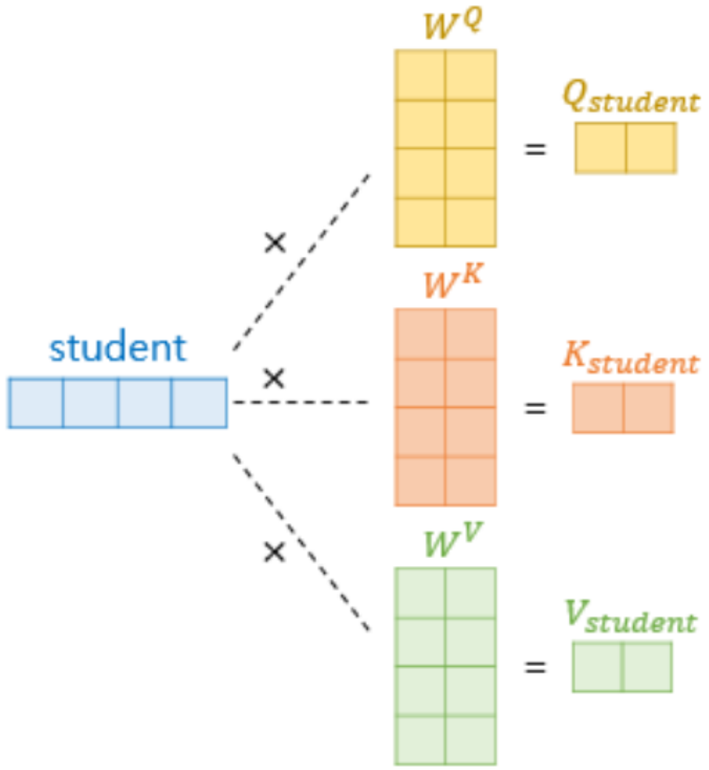




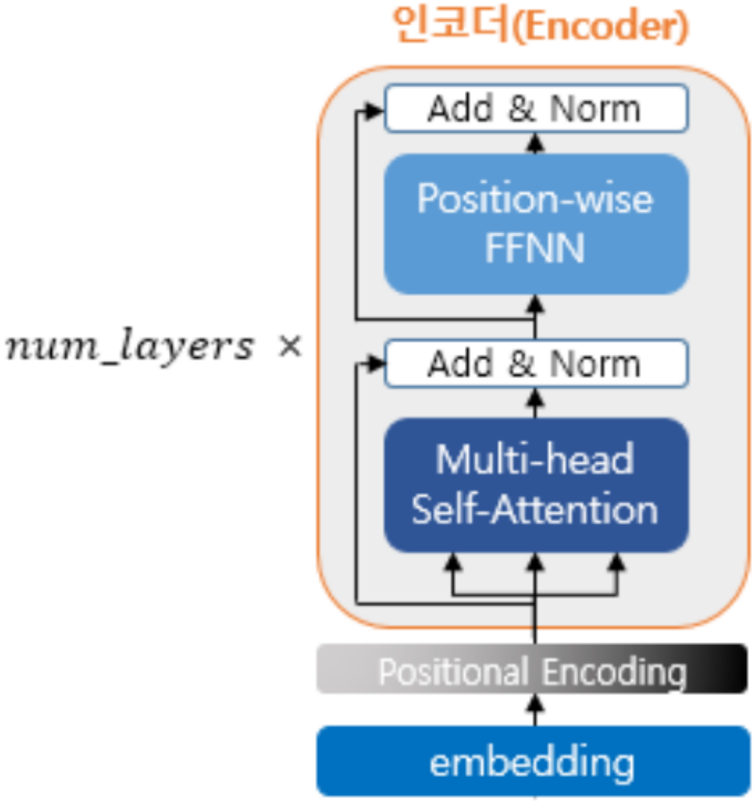
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

멀티 헤드 어텐션 - 셀프어텐션

1. Q, K, V의 벡터 구하기(이때 차원은 64, 입력 단어 벡터 차원보다 적게!)



인코더

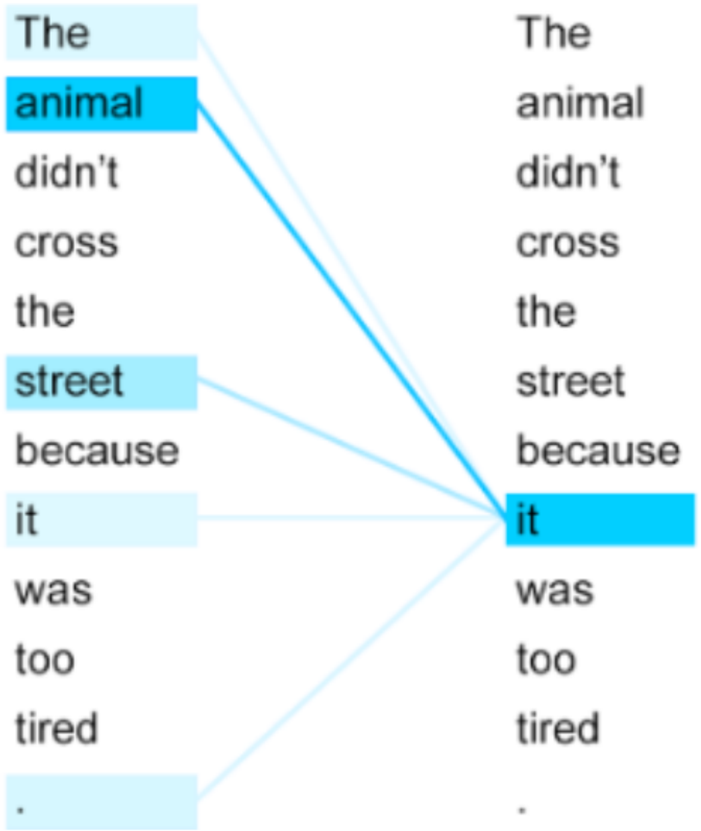


멀티 헤드 어텐션

피드 포워드 네트워크

2가지 요소로 작동

멀티 헤드 어텐션 - 셀프어텐션



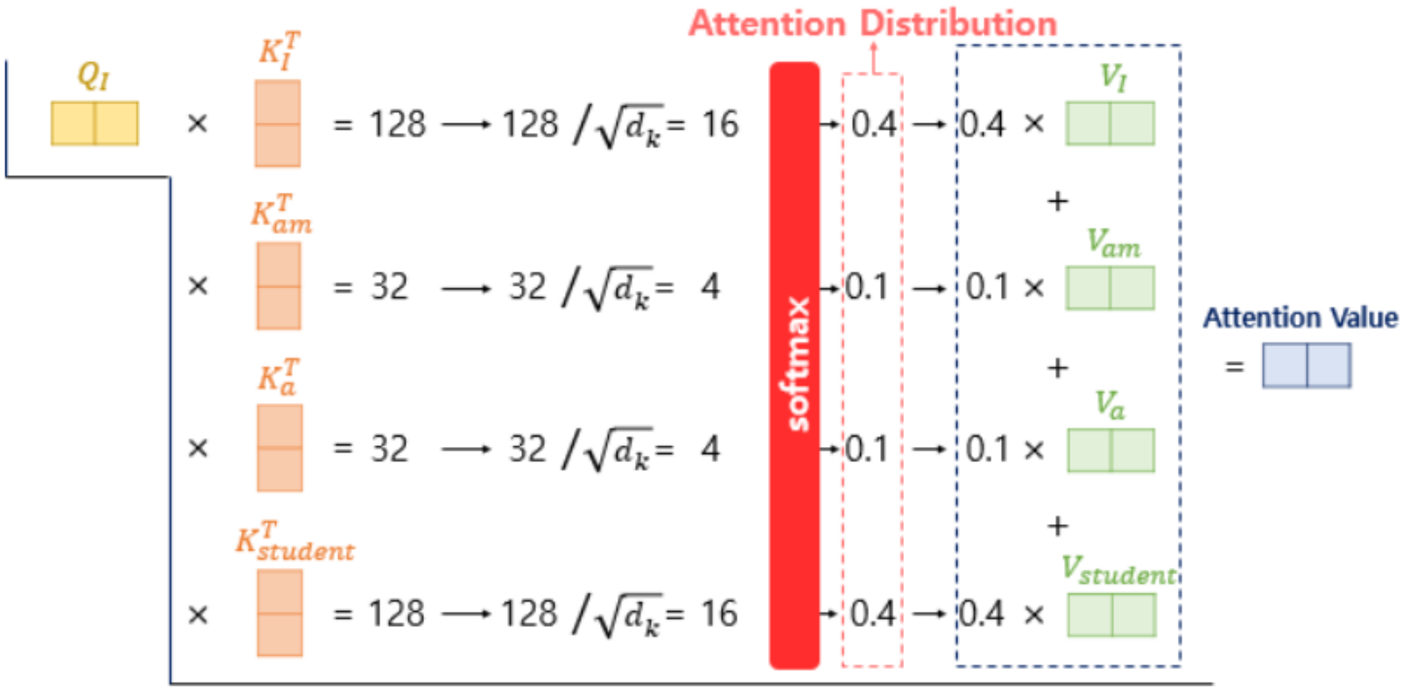
단어들끼리 유사도를 구함

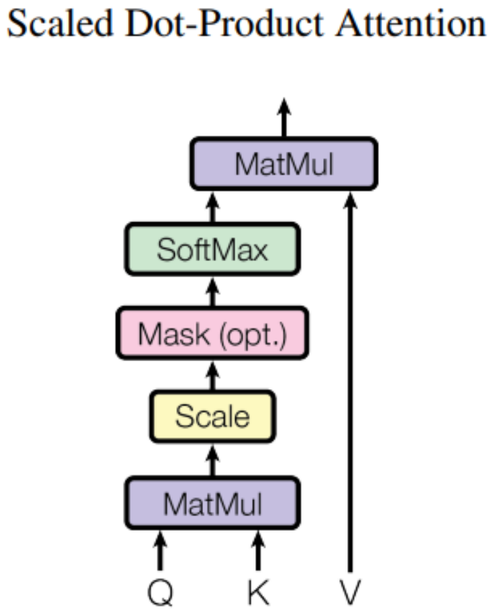
멀티 헤드 어텐션 - 셀프어텐션

2. $score(q, k) = q \times k / \sqrt{n}$ 구하기

3. 2에서 구해준 유사도 값에 소프트맥스 함수를 사용하여 정규화 -> 스토어 행렬

4. V벡터와 가중합 -> 어텐션 값(Attention Value)



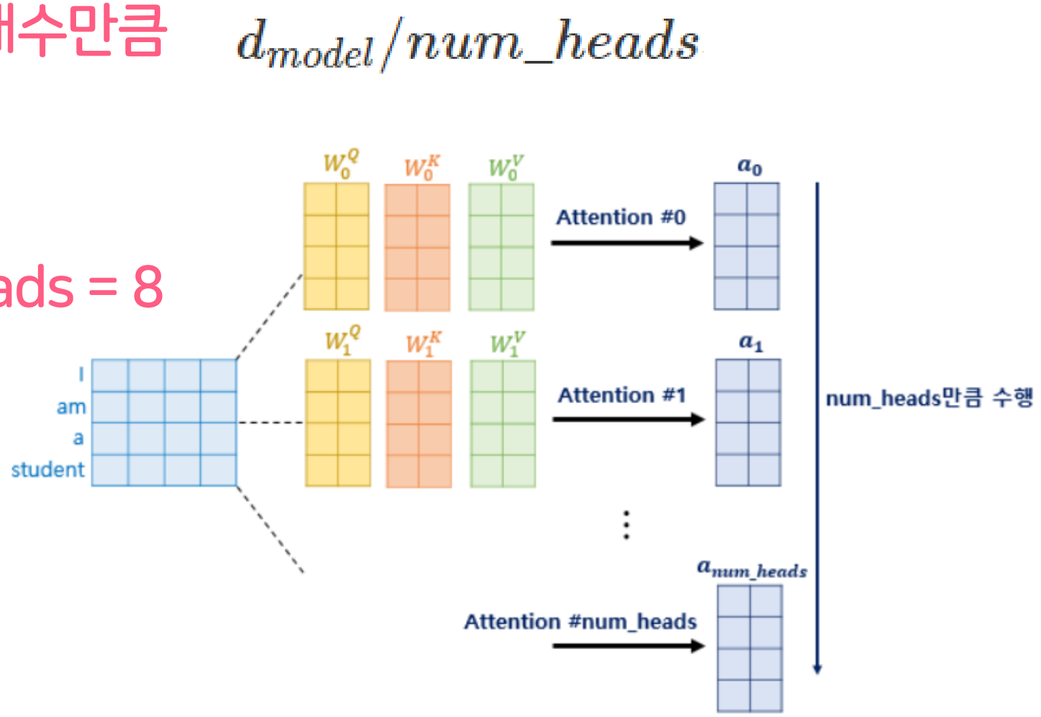


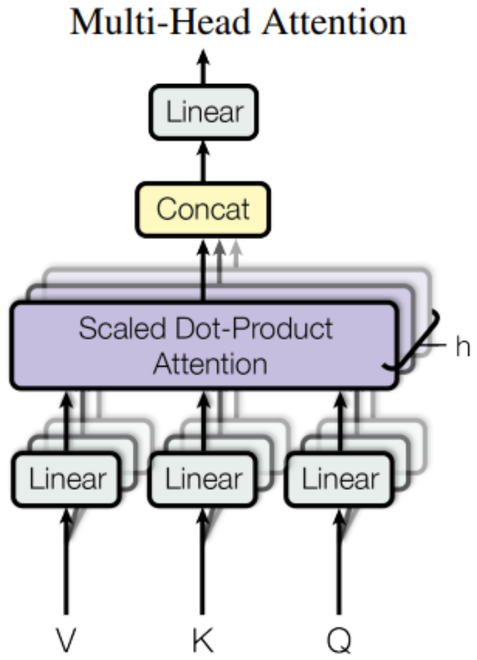
멀티 헤드 어텐션 (multi-head self-attention)

스코어 행렬의 값이 잘 계산 되지 않았을 경우 문장의 의미가 잘못 해석 될 수 있다!
따라서 정확도를 높이기 위해 멀티 헤드 어텐션을 사용한다.(다른 시각으로 정보를 수집하기 때문, 어텐션헤드마다 가중치 행렬이 다르다!)

입력 단어 벡터 차원을 멀티 헤드 개수만큼 나눈 값을 Q,V, K의 차원으로 사용

논문에서는 512 차원에 num_heads = 8
따라서 64차원으로 change



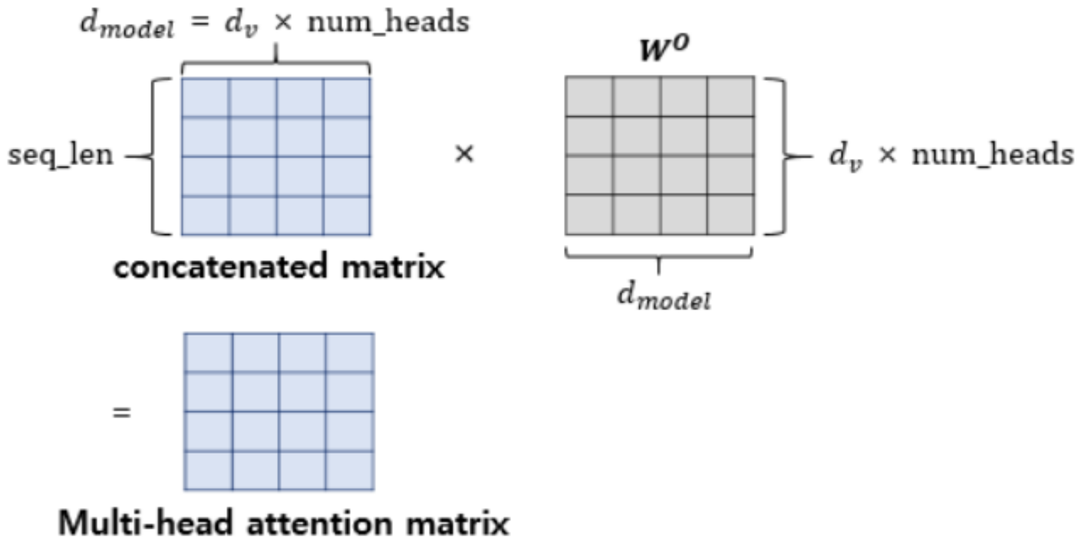


*num_heads 하이퍼파라미터

멀티 헤드 어텐션

병렬적으로 attention 수행 후
어텐션 헤드를 연결(concatenate)한 후 새로운 가중치 행렬 W^O

을 곱해준다!!
이때 최종 멀테 헤드 어텐션의 크기 입력으로 들어온 행렬의 크기와 똑같다.

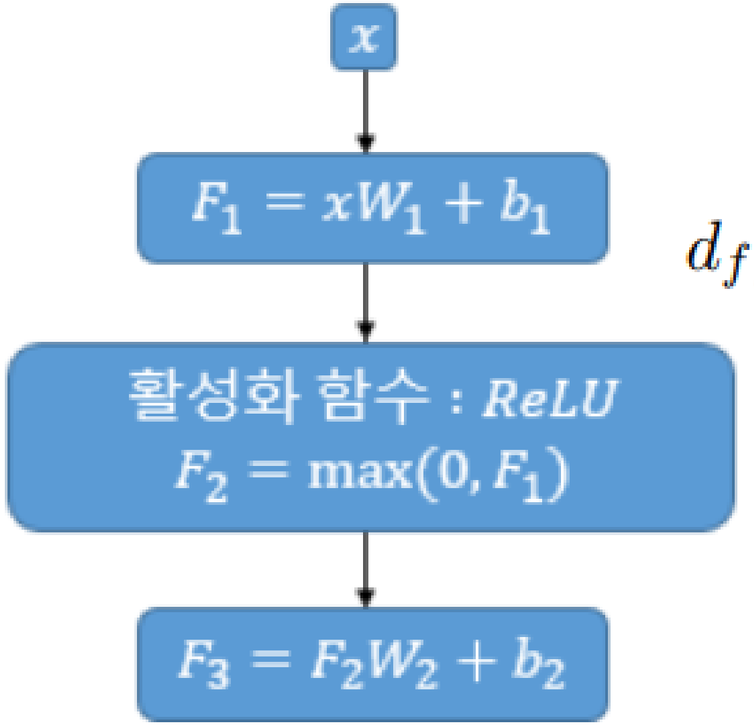


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

피드 포워드 네트워크(feed-forward network)

2개의 선형 변화과 그 사이에 ReLU

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$



$d_{ff} = 2048.$

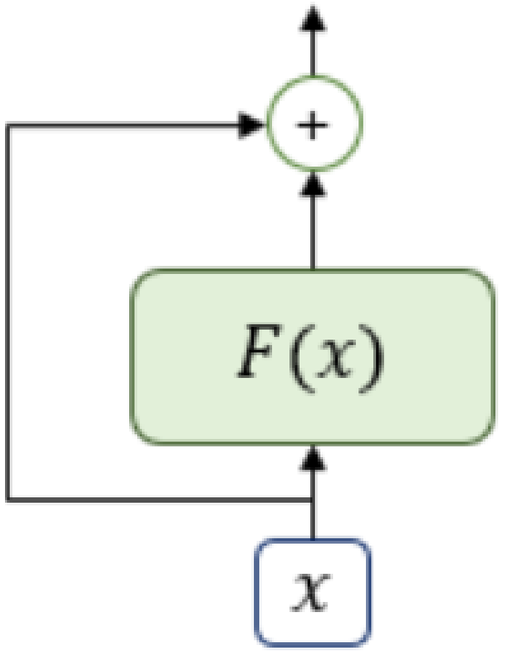
```
outputs = tf.keras.layers.Dense(units=dff, activation='relu')(attention)
outputs = tf.keras.layers.Dense(units=d_model)(outputs)
```

잔차연결(residual connection)

서브레이어의 입력과 출력을 더하는것
-> 정보 손실 방지 최소화, 경사 소실 문제 완화, 학습과정 안정화

$$H(x) = x + F(x)$$

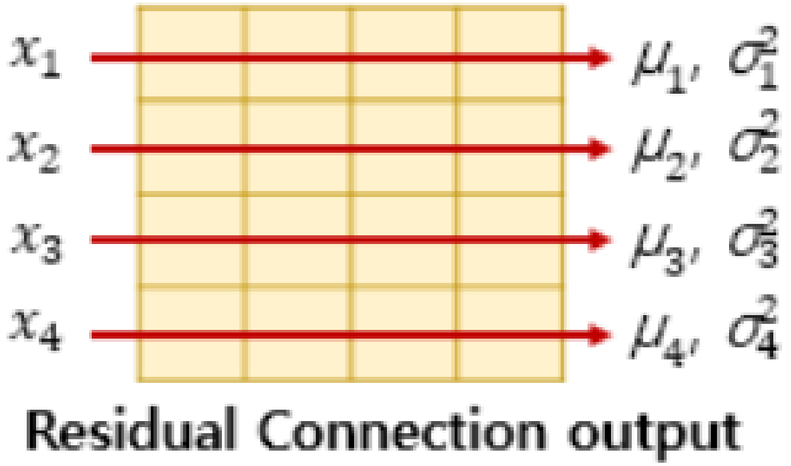
잔차연결을 하기 위해서 모든 서브 레이어의 출력과
임베딩 레이어의 차원을 512로!



norm - 정규화

1) 평균과 분산을 통한 정규화

$$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$



2) 감마와 베타를 도입

$$ln_i = \gamma \hat{x}_i + \beta = LayerNorm(x_i)$$

γ

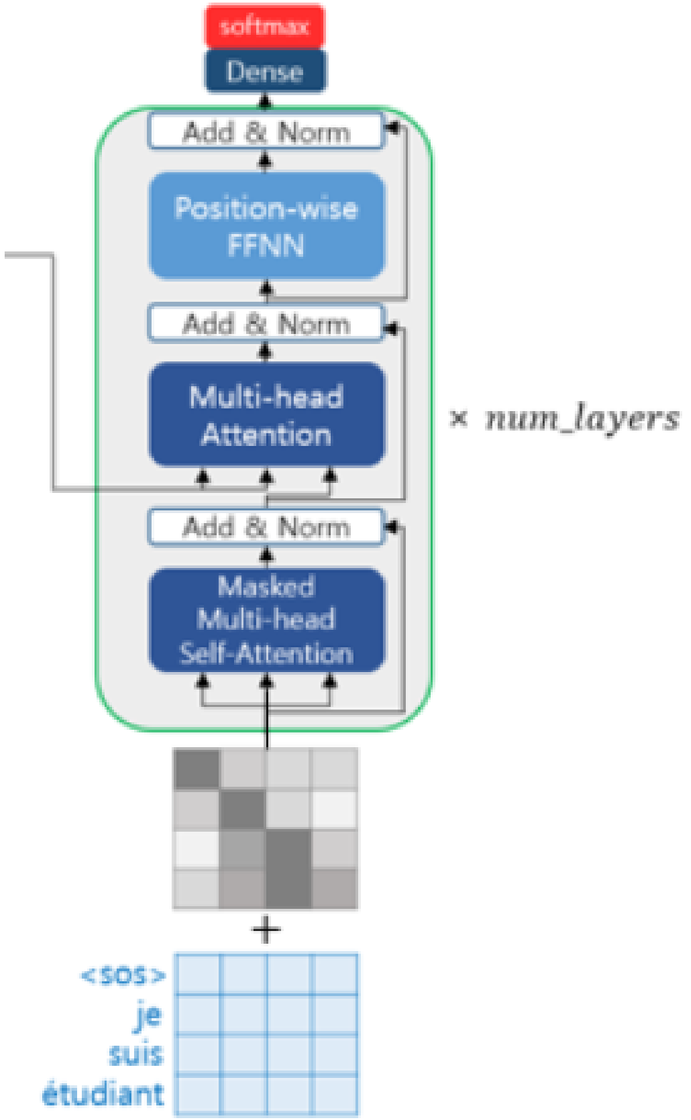
1	1	1	1

β

0	0	0	0

디코더

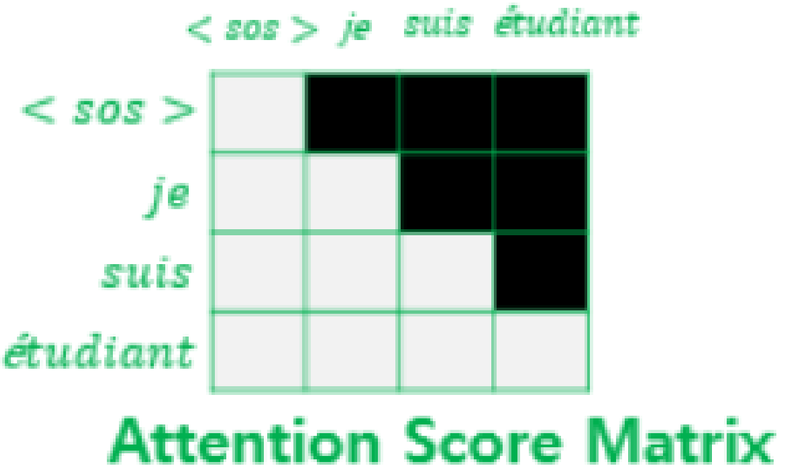
1. 마스크된 멀티 헤드 어텐션
2. 멀티 헤드 어텐션
3. 피드포워드



마스크된 멀티 헤드 어텐션

position i 에 대한 예측이 i 보다 적은 위치의 출력 단어들에 만 의존하도록
그외 부분은 인코더의 멀티 헤드 어텐션과 동일

-> 디코더의 입력을 이전 단계에서 생성한 단어만을 넣기 때문!
즉 현재 입력을 주어진 단어만 연관성을 고려해야하는데
인코드의 출력값에는 전체 단어가있으므로 이를 마스킹해줘야 한다.

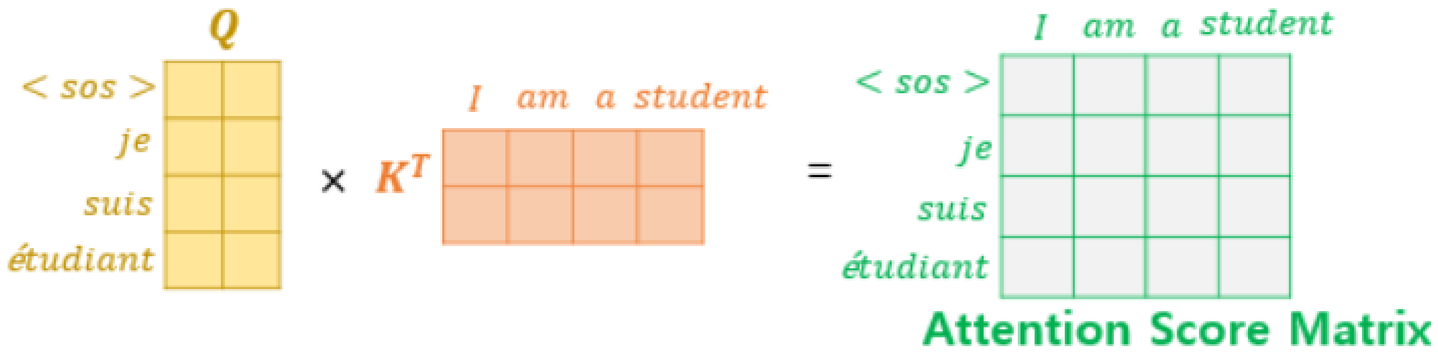


멀티 헤드 어텐션

쿼리 행렬 Q의 디코더에서 얻은 즉 이전 서브레이어의 출력값을 사용하여 생성
K, V의 경우 인코더의 출력값 즉 인코더 표현값을 활용하여 생성

-> 이는 디코더는 결국 타깃 문장(출력 문장)을 만든것이기 때문에 타깃 문장에 대한 값을 참고해야한다.
이때 타깃 문장 값이 이전 서브레이어의 출력값에 담겨 있기 때문에 이를 참조

K, V의 경우 생성한 타깃 문장의 단어가 입력 문장에서 어떤 단어와 유사한지 확인하도록 한다.



선형과 소프트맥스 레이어

1. 여러개의 디코더를 통과한 후 마지막 디코더의 출력 값을 선형 레이어에 전달
- 이때 선형레이어의 차원은 VOCAB의 크기

2. 1에서 출력값을 소프트맥스 함수에 넣어 확률값으로 나타내 준다.
- 이를 통해 타깃 문장의 다음 단어 예측

실험

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**	$2.3 \cdot 10^{19}$	

실험

	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

실험

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

Q&A - 키 벡터 차원의 제곱근값으로 나누는 이유?

- 내적 값이 큰 값을 가지게 되면 softmax function가 작은 그래디언트 값을 가지므로 그래디언트 소실 문제가 발생 할 수 있으므로 이를 완화하기 위해
- 특히 키 벡터 차원이 커질 수록 내적값이 더 커지므로 이에 비례하여 값들이 적절한 범위 내에 위치하도록 조정해주기 위해 키벡터 차원의 제곱근으로 나누어줌 (+ 실험을 통해 효과적임이 입증)

참고자료

트랜스포머(Transformer) 간단히 이해하기
https://moondol-ai.tistory.com/461

구글 BERT의 정석, 한빛미디어, 수다르산 라비찬디란

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (pp. 5998-6008).

딥 러닝을 이용한 자연어 처리 입문
https://wikidocs.net/31379