Attention is all you need

Transformer

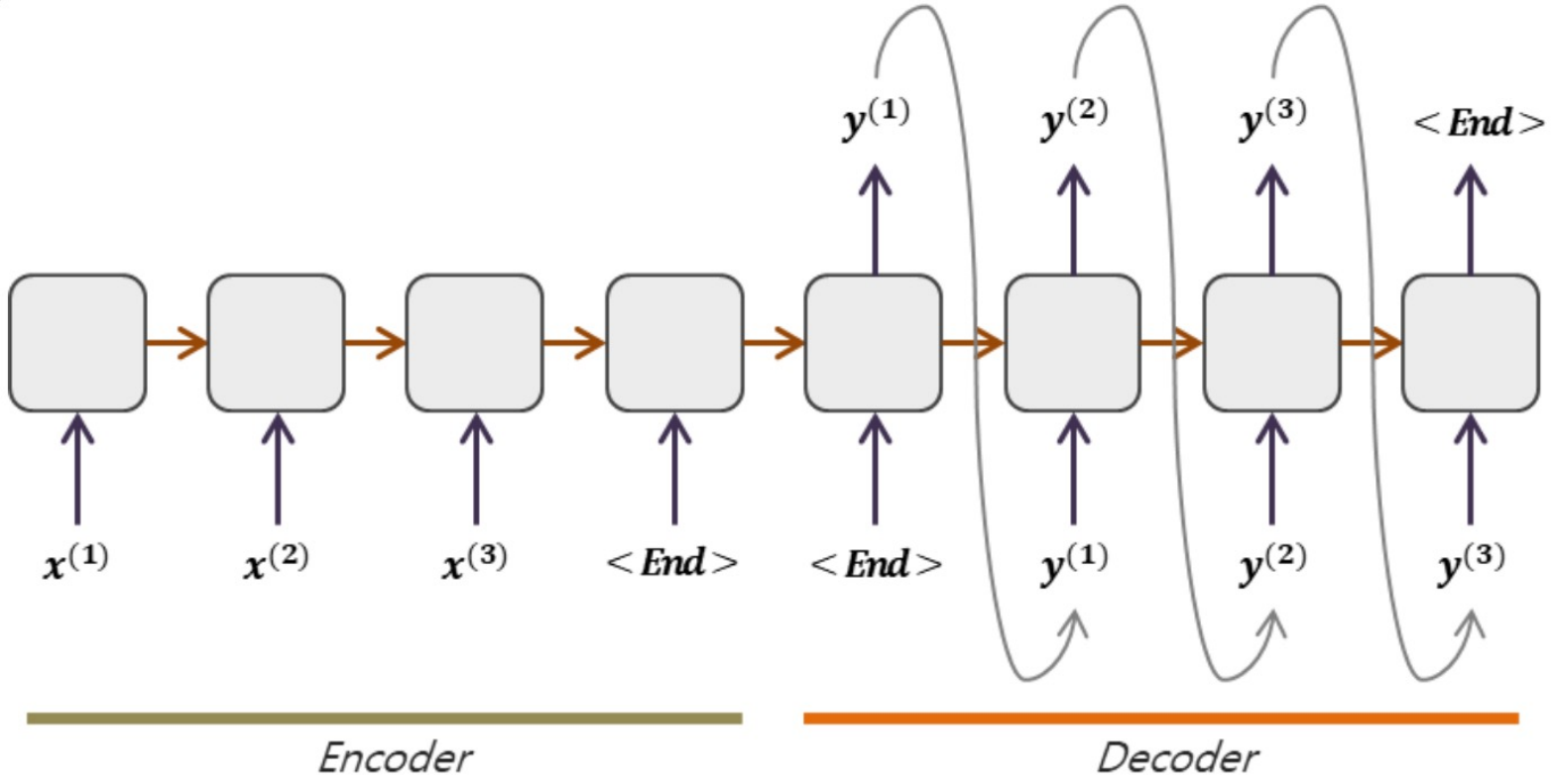
https://arxiv.org/abs/1706.03762.pdf

이정우

Introduction & Background

Transformer 이전의 모델들의 문제점

- Long-term dependency problem
- Parallelization



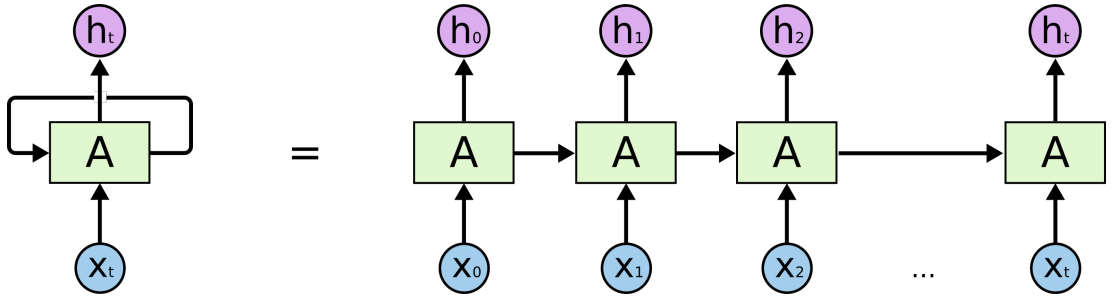
Introduction & Background

Transformer 이전의 모델들의 문제점

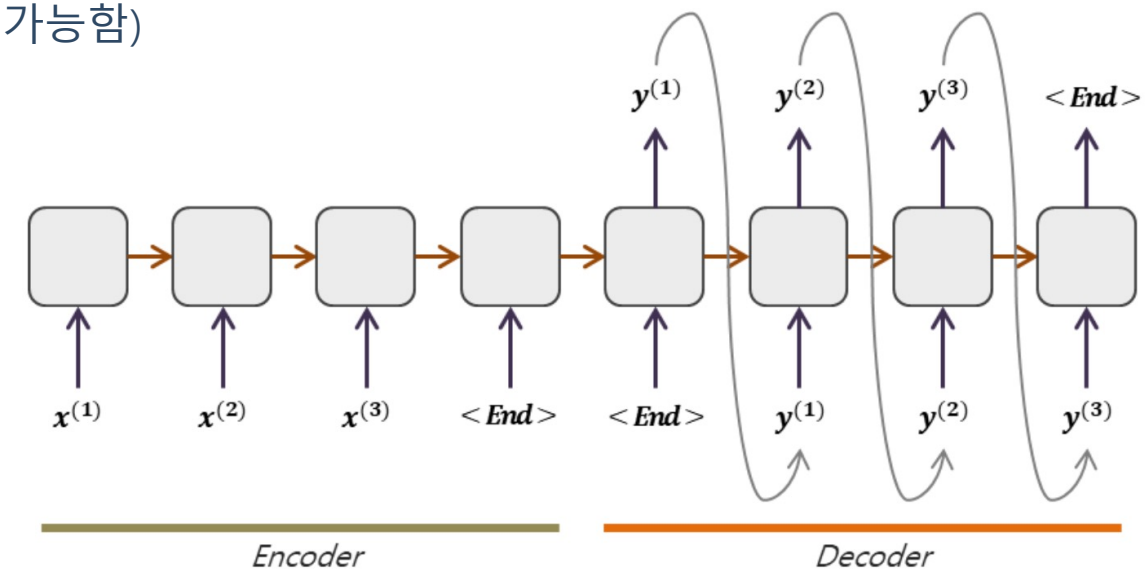
- Long-term dependency problem

  (고정길이 벡터이므로 Sequence 길어지면 Long-term dependency 처리 능력 저하)

- Parallelization (순서 정보를 기억해야 하므로 병렬 계산 불가능함)



RNN



Seq2Seq

Introduction & Background

Attention

- **Attention**은 단어의 의미처럼 특정 정보에 가중치를 부여하여 좀 더 주의를 기울이는 것이다.

- 예를 들어 Model이 수행해야 하는 task가 번역이라고 가정한다.

  source는 영어이고 target은 한국어이다.

  "I like stray cat." 라는 문장과 대응되는 ***"나는 길고양이 좋아해."*** 라는 문장이 있다 했을 때

  Model이 길고양이라는 token을 decode 할 때, source에서 가장 중요한 것은 **stray cat** 이다.

- 이때, source의 모든 token이 비슷한 중요도를 갖는 것 보다는

  stray cat이 더 큰 중요도(가중치)를 가지면 된다.

  더 큰 중요도(가중치)를 갖게 만드는 방법이 바로 **Attention**이다.

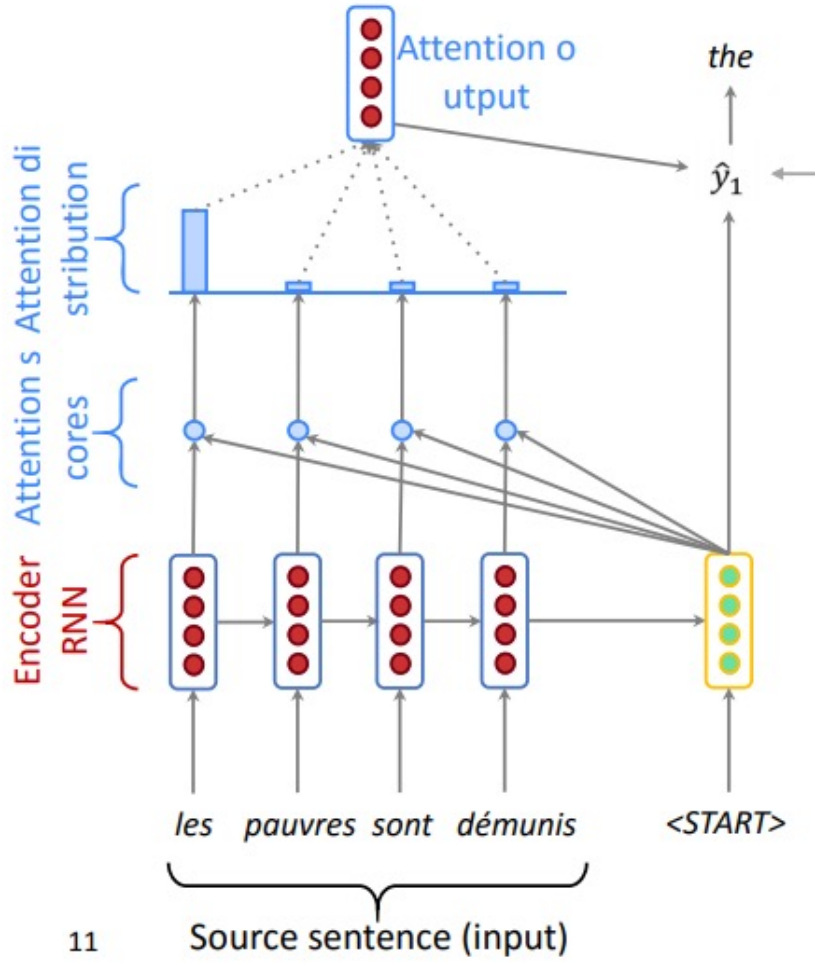
Introduction & Background

Attention

$$e_{ij} = a(s_{i-1}, h_j)$$

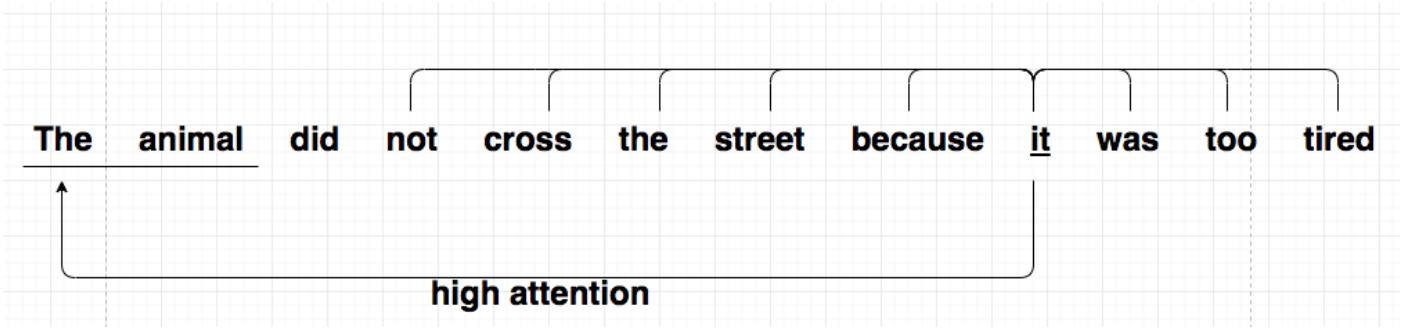
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

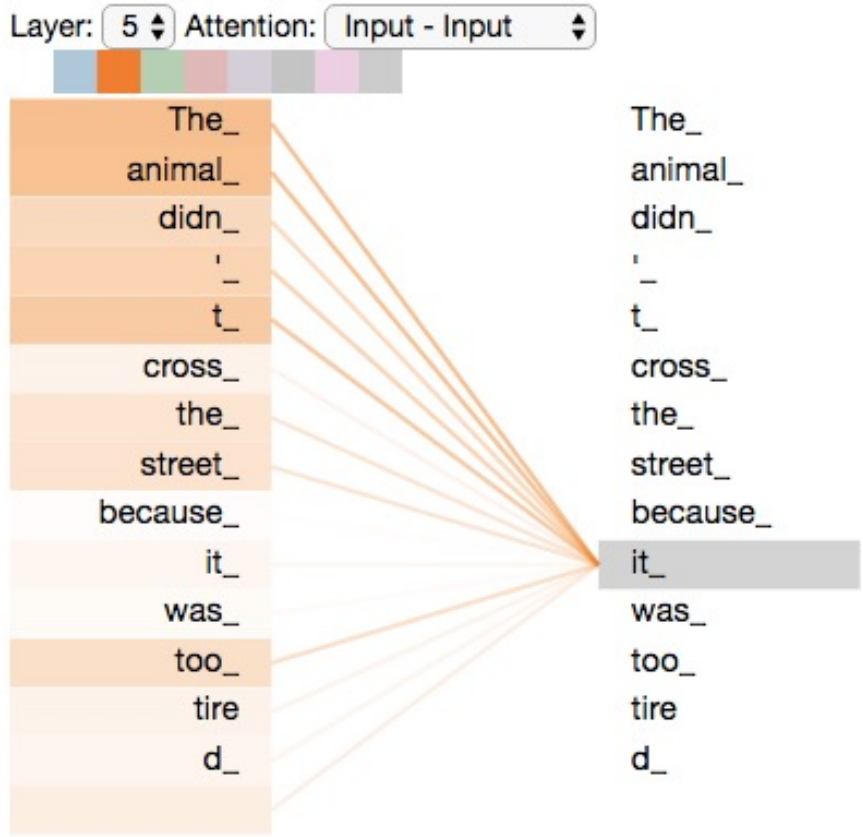
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$



Introduction & Background

Self-Attention

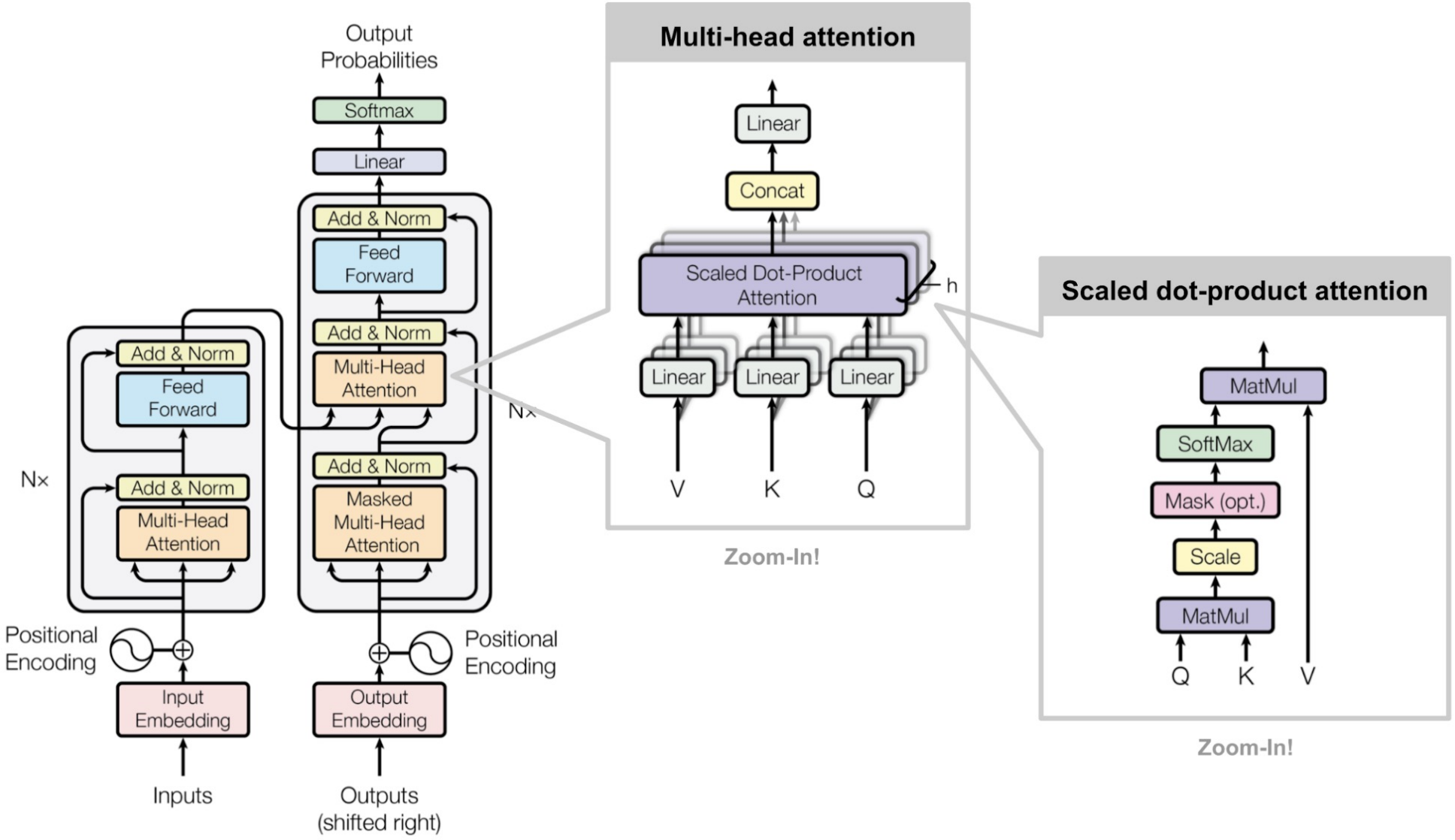




- 특정 문장이 있을 때 자기 자신의 문장 스스로에게 **Attention**을 수행해 학습하는 것
- 하나의 시퀀스가 있을 때 그 시퀀스에 포함된 서로 다른 위치 정보가 서로가 서로에게 가중치를 부여하게 만들어 하나의 시퀀스에 대한 representation을 효과적으로 학습, 표현

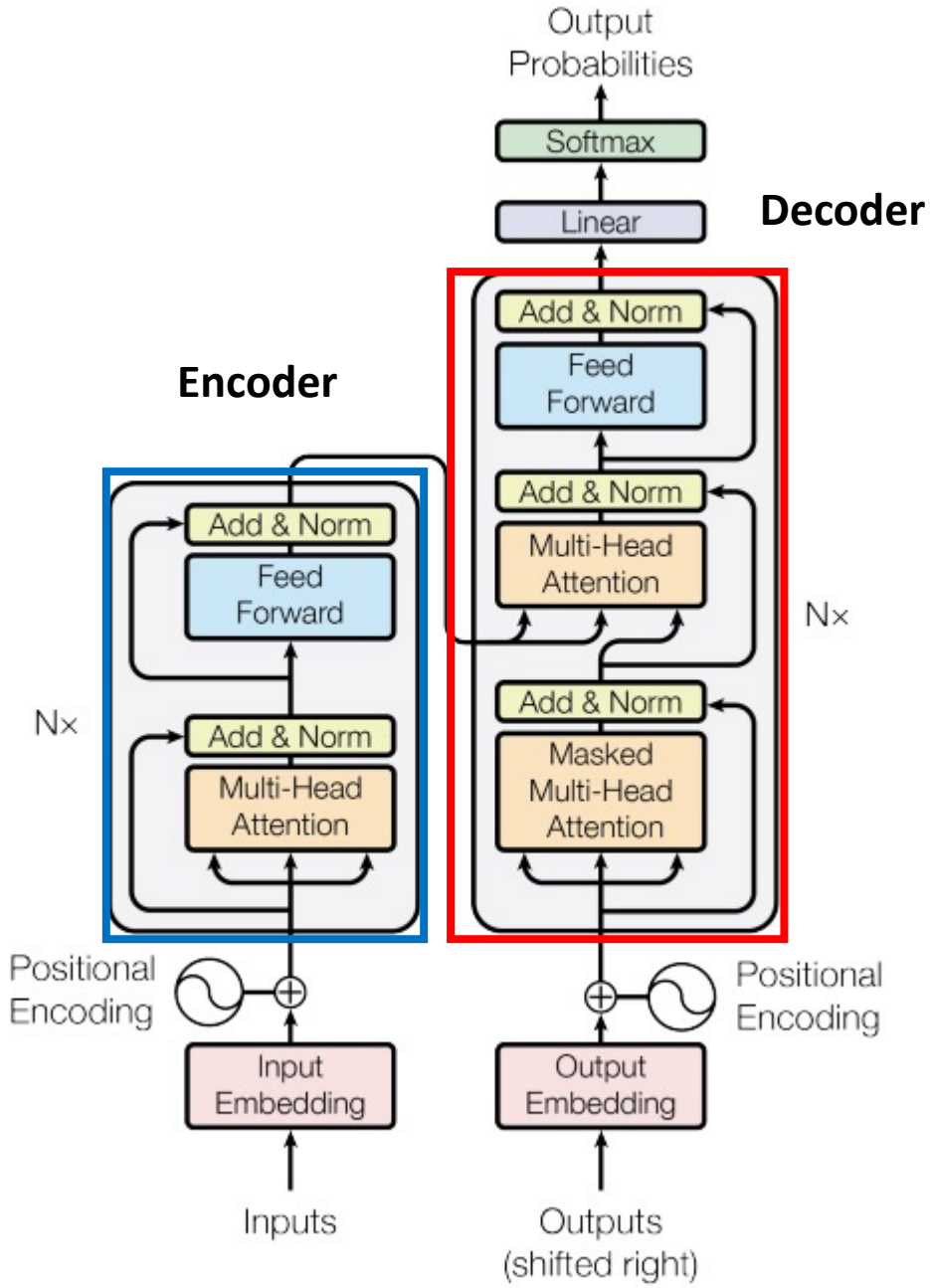
  예) I am a teacher → 4개의 단어는 서로에게 **Attention** 수행, 가중치 부여

Model Architecture



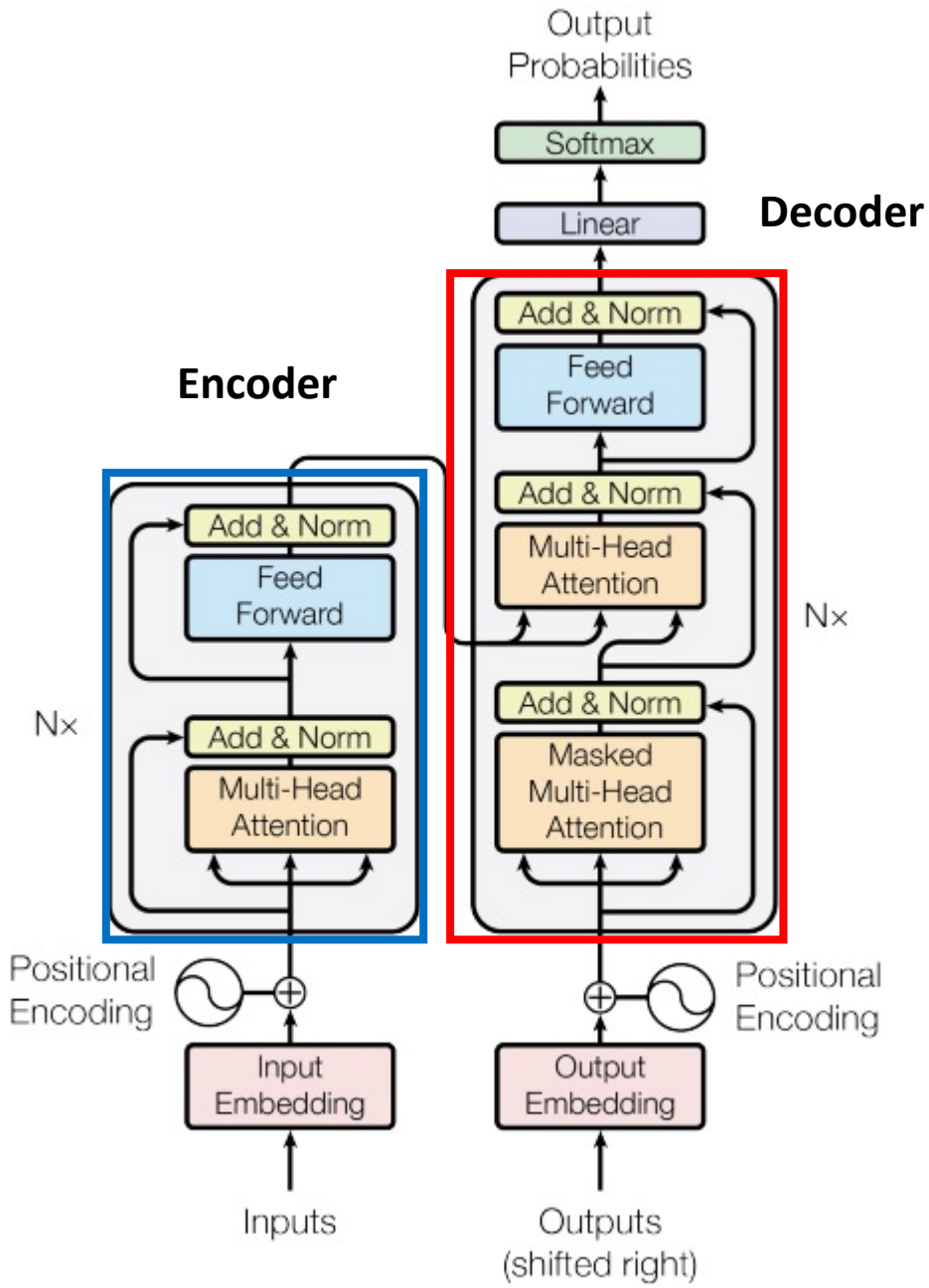
Model Architecture

- RNN 이나 CNN을 사용하지 않음
- 기존에 잘되던 모델은 RNN에 attention을 얹은 형태
- RNN Encoder-Decoder 번역모델의 단점 보완하고자 나옴
- 중요한 부분에 더 집중(attention weight)
- RNN이나 CNN을 쓰는게 아니라
  아예 성능 좋은 Attention으로 대체

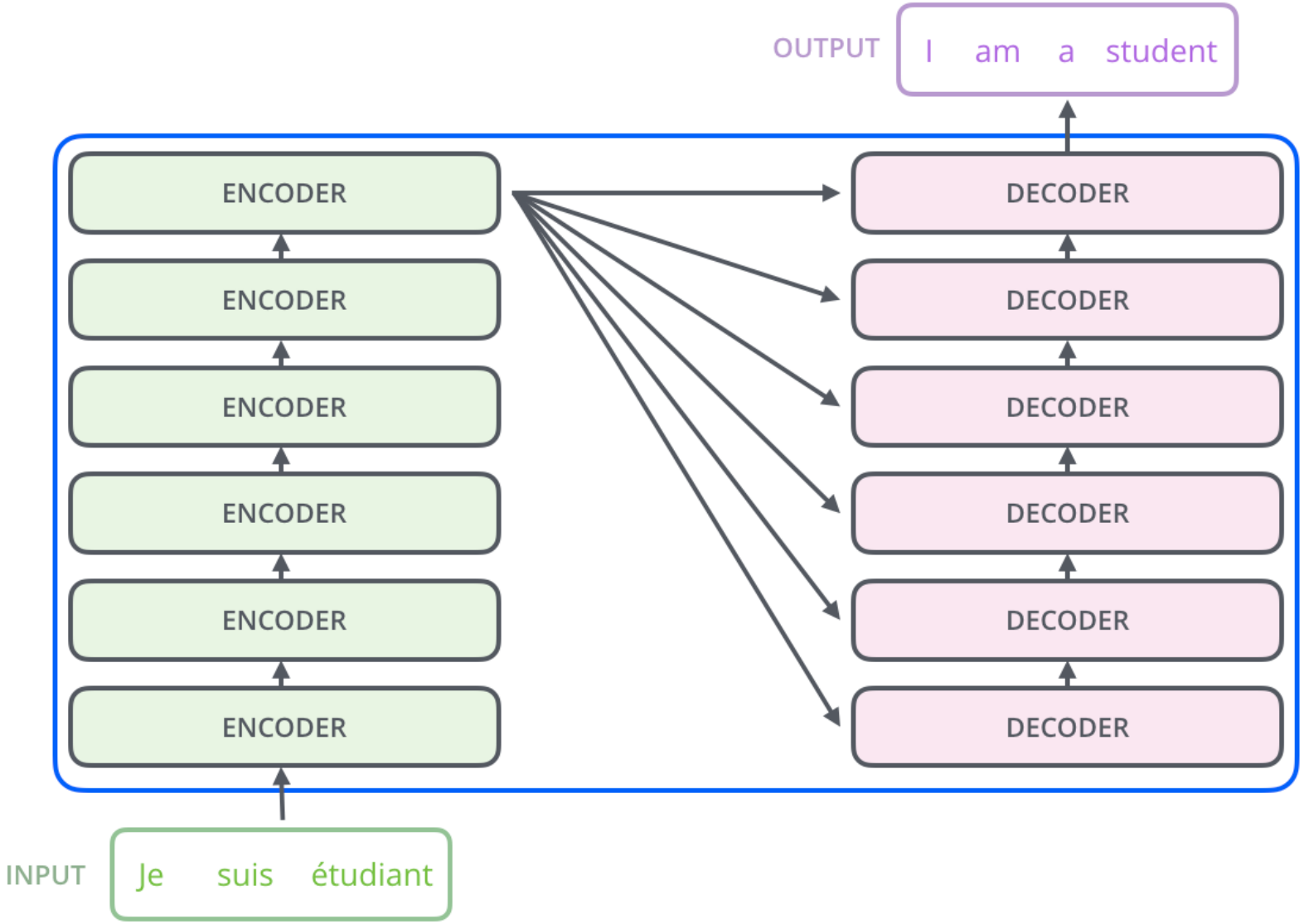


Model Architecture

- RNN 이나 CNN을 사용하지 않음
- Encoder와 Decoder로 구성
- Encoder : 2개의 sublayer
- Decoder : 3개의 sublayer
- N개의 동일한 layer를 stack 구조로 쌓아 올린 형태

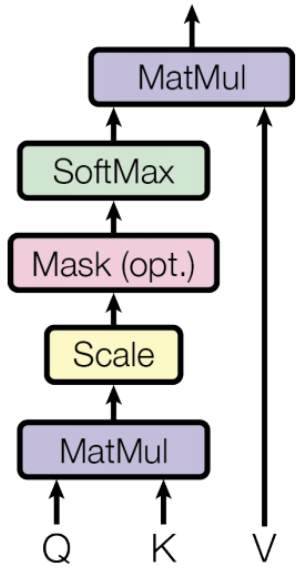


Model Architecture



Model Architecture

Attention - Scaled Dot-Product Attention



Q : query , K : key, V : value

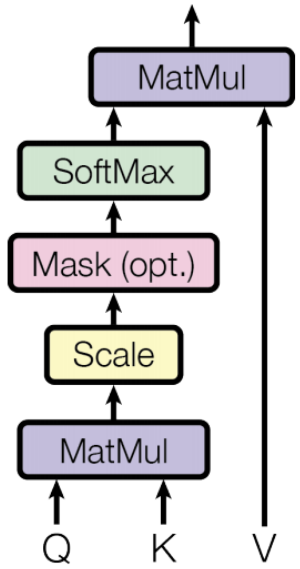
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

d_k = key / value vector의 dimension

- Query(Q), Key(K), Value(V)를 입력 받는다.
- Q, K의 dot production를 통해 attention weight를 구한다.

Model Architecture

Attention - Scaled Dot-Product Attention



Q : query , K : key, V : value

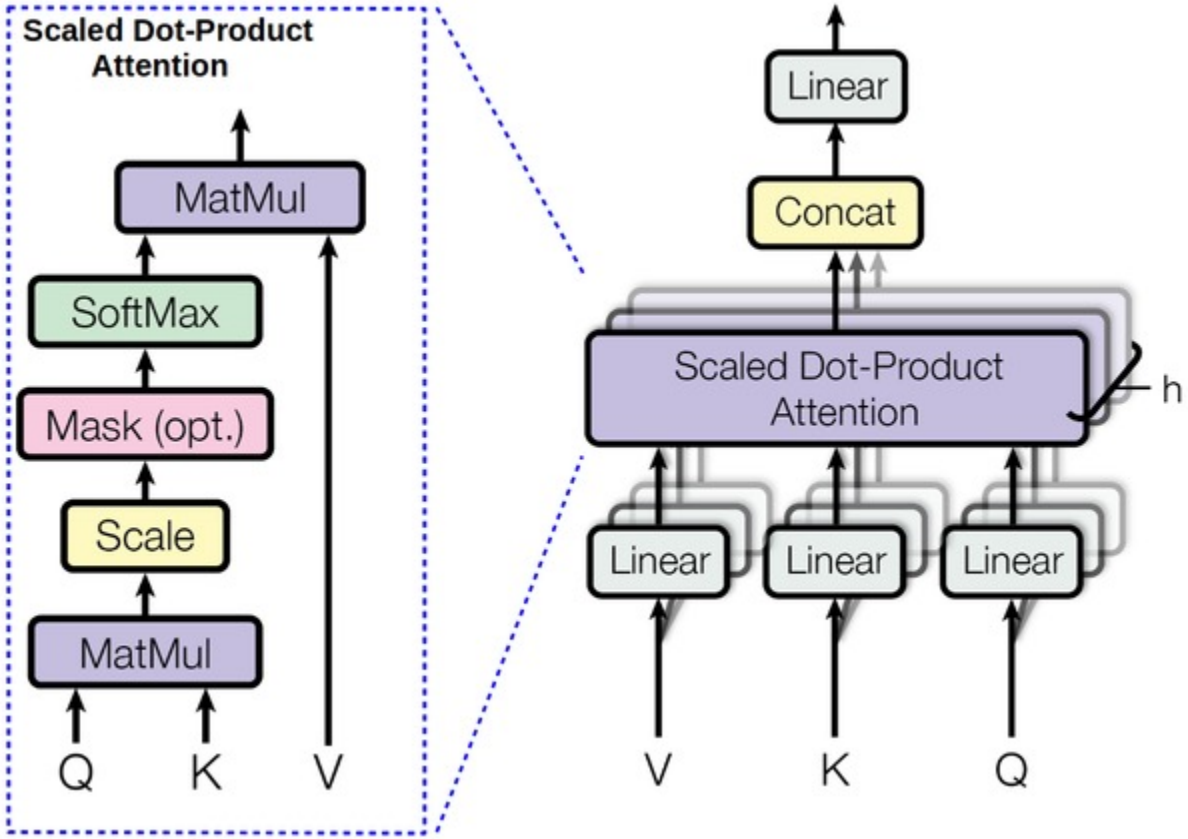
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

d_k = key / value vector의 dimension

➡ d_k가 커질수록 내적 값도 커져 softmax의 값이 편중될 위험이 있으므로 scaling을 진행

Model Architecture

Multi-Head Attention



- Multi-Head Attention은
  여러 개의 Scaled Dot-Product Attention으로 이루어진다.
- 논문에서는 8개의 Scaled Dot-Product Attention
- 전체 모델이 512차원 8로 나누면 64차원
- 각 Scaled Dot-Product Attention은 64차원
- 이렇게 병렬처리한다

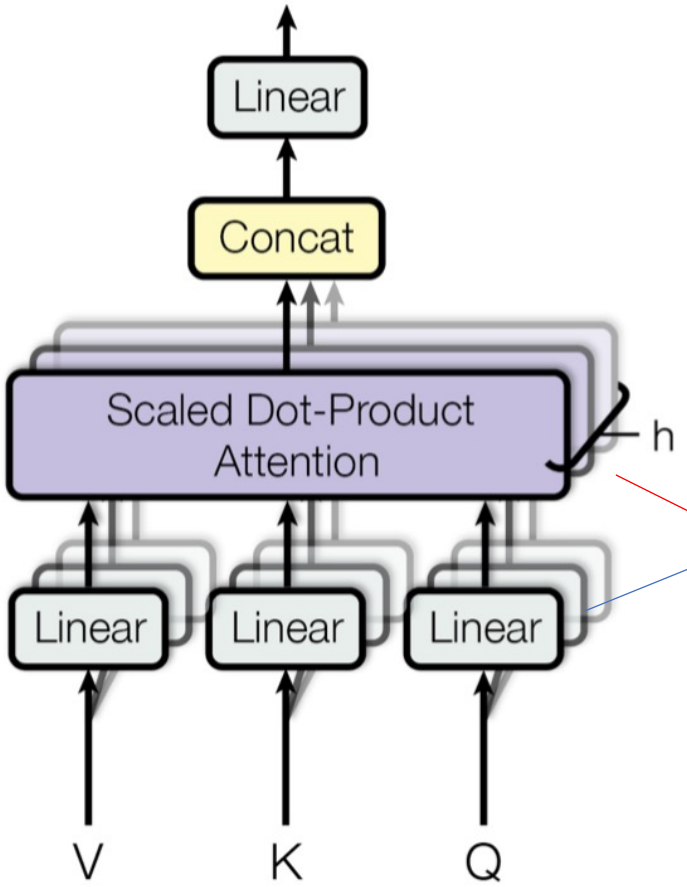
Model Architecture

Multi-Head Attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

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

$$d_k = d_v = d_\text{model}/h = 64$$



$$QW_i^Q = [d_Q \text{x} d_{model}] \text{x} [d_{model} \text{x} d_k] = [d_Q \text{x} d_k]$$

$$KW_i^K = [d_K \text{x} d_{model}] \text{x} [d_{model} \text{x} d_k] = [d_K \text{x} d_k]$$

$$VW_i^V = [d_V \text{x} d_{model}] \text{x} [d_{model} \text{x} d_v] = [d_V \text{x} d_v]$$

$$\downarrow$$

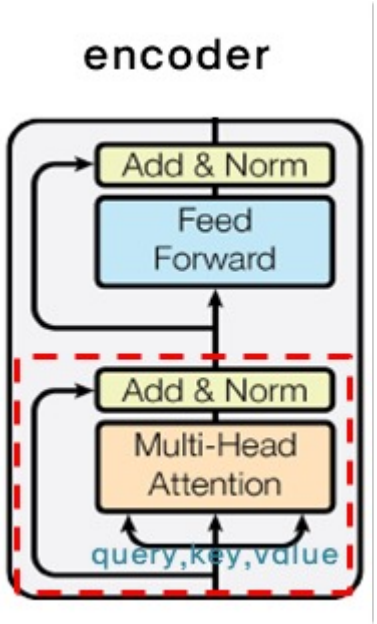
$$Attention(QW_i^Q, KW_i^K, VW_i^V) = [d_V \text{x} d_v]$$

$$\downarrow$$

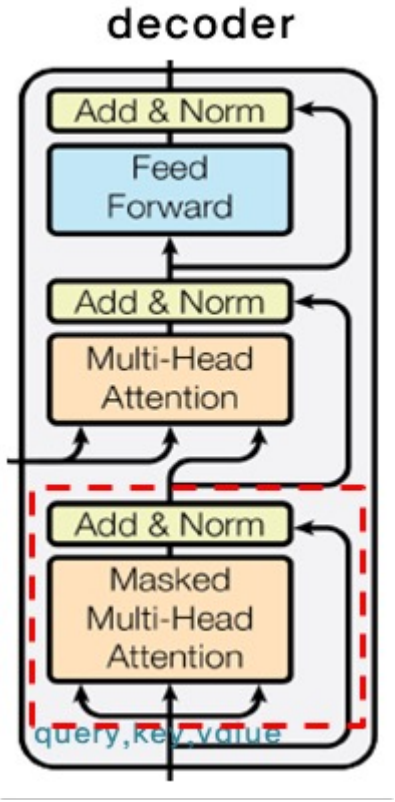
$$Concat(QW_i^Q, KW_i^K, VW_i^V)W^O = [d_V \text{x} hd_v] \text{x} [hd_v \text{x} d_{model}] = [d_V \text{x} d_{model}]$$

Model Architecture

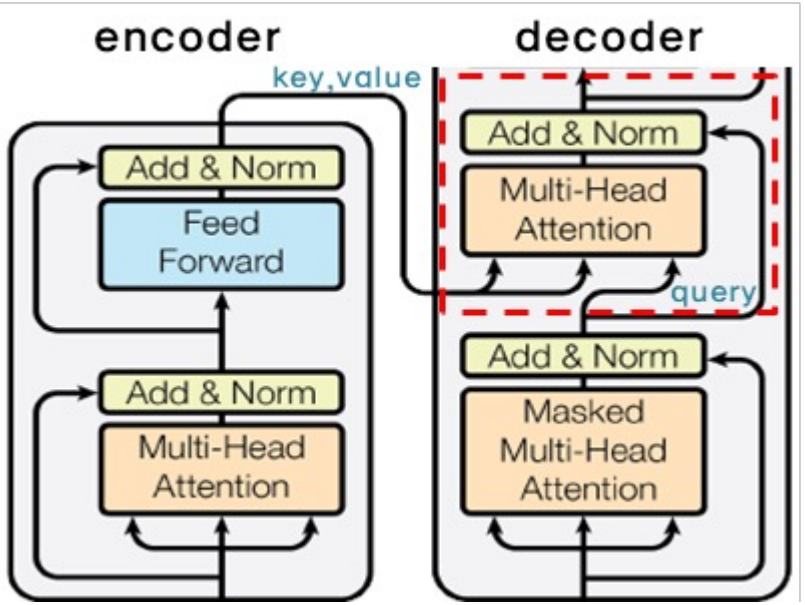
Applications of Attention in Model



Encoder Self-Attetnion



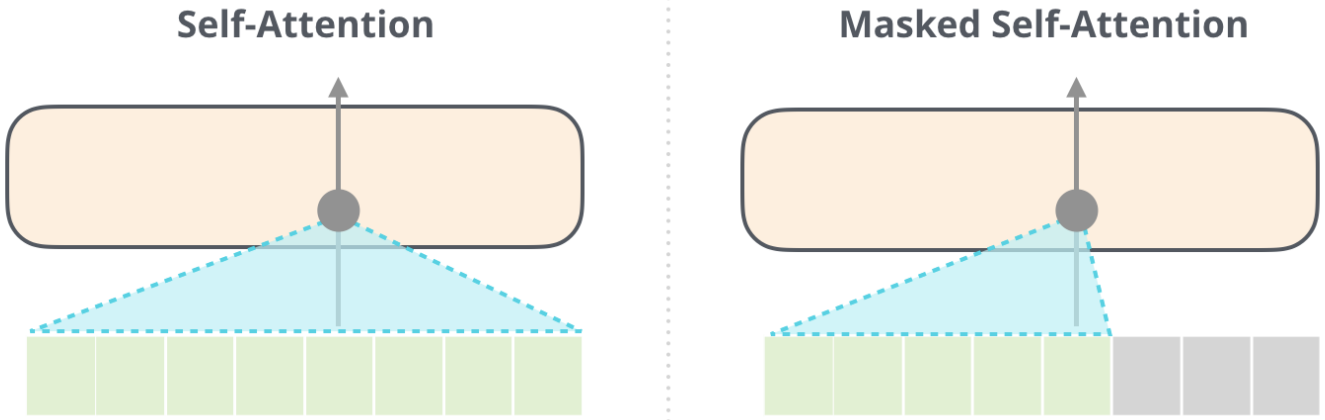
Masked Self-Attetnion

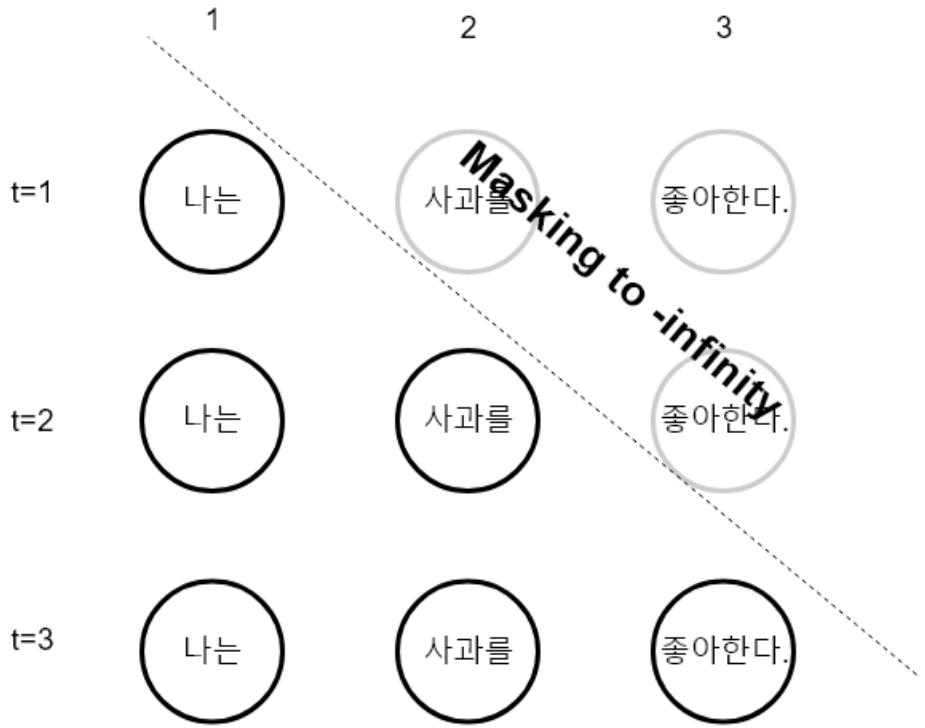


Encoder-Decoder Attetnion

Model Architecture

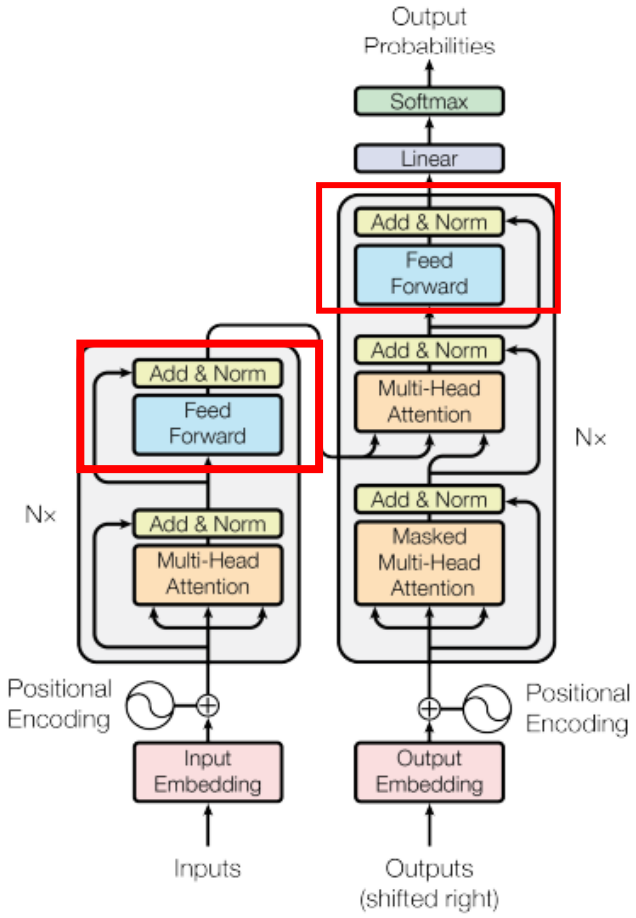
Masked Attention





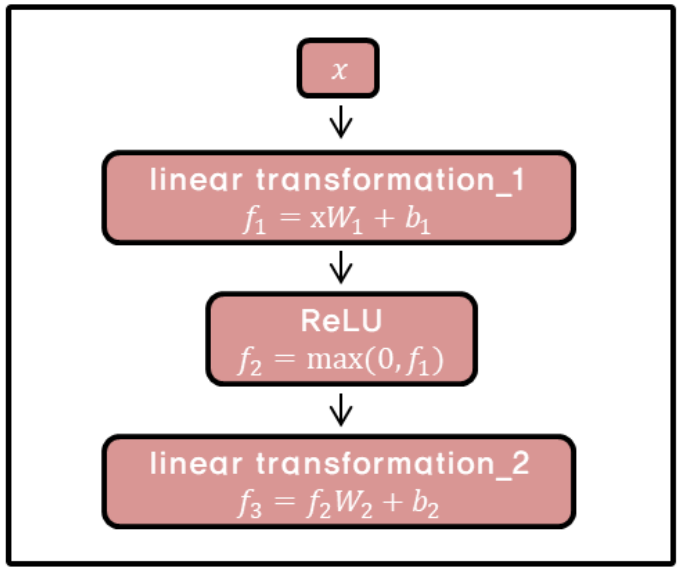
Model Architecture

Position-wise Feed-Forward Networks



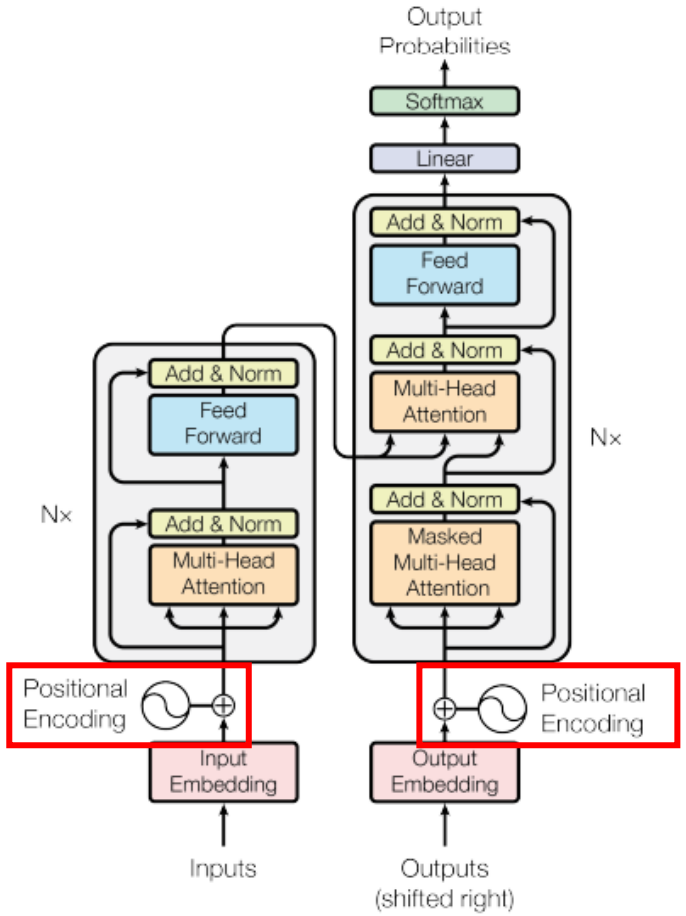
$$\mathrm{FFN}(x) = \underline{\max(0, xW_1 + b_1)}W_2 + b_2$$

ReLU



Model Architecture

Positional Encoding



- RNN을 사용하지 않기 때문에 위치 정보가 결여
- 주기 함수를 활용한 공식을 사용
- Embedding vector에 Positional vector를 더해주는 방식

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

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

Why Self-Attention

- Layer마다 필요로 하는 총 연산 량이 줄어든다.
- 병렬처리가 가능한 연산이 늘어난다.
- 먼 거리에 있는 시퀀스를 잘 학습할 수 있다.
- Attention을 사용하면 모델의 동작을 해석하기 쉬워진다.

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Training

- **Dataset :**
  - WMT 2014 English-German dataset(4.5M쌍의 문장, 37000 vocab)
  - WMT 2014 English-French dataset(36M쌍의 문장, 32000 vocab)
- **Batch size :** 25000
- **Hardware :** 8개의 P100 GPU
- **Schedule :**
  - Base Model : 12시간=10만 step × 0.4초/step,
  - Big Model : 36시간=30만 step

Training

- **Optimizer :**
  - Adam optimizer
- **Regularization :**
  - Residual Dropout
  - Layer Smooth

Results

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

Results

	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

Results

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

Conclusion

- RNN, CNN을 사용하지 않고, Attention만 사용하여 Sequential data를 처리할 수 있는 모델 제안
- 다른 모델들 보다 빠른 학습 속도와 좋은 성능을 보임
- Parallelization 이 가능해졌다
- 기계 번역 뿐만 아니라 큰 입출력 값을 가지는 분야에도 적용될 것이 기대됨