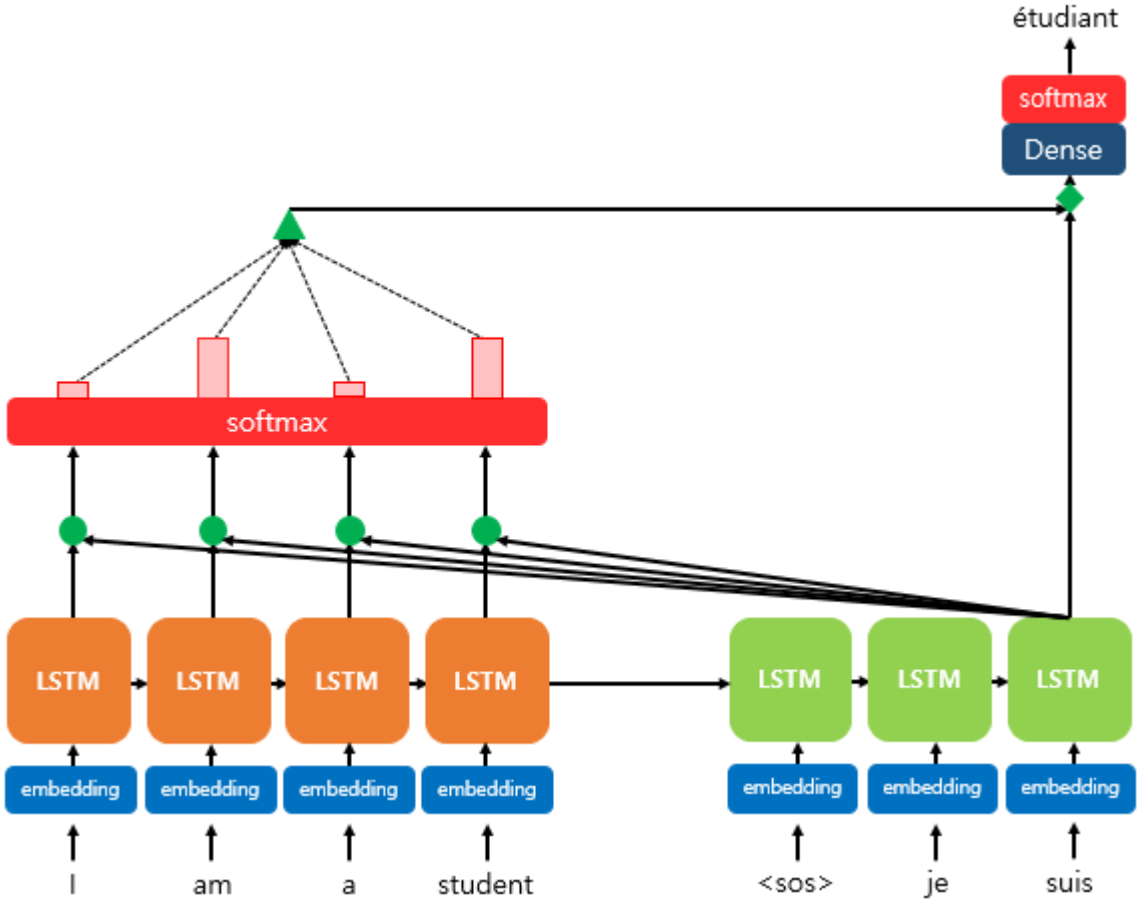
DATA
SCIENCE LAB

NLP Week 6
Transformers

23.02.25 / 9기 조의현

1. Review

RNN + Attention Mechanism



쿼리 (QUERY) : 분석의 대상이 되는 단어에 대한 가중치 벡터

➔ **현재 시점에서 분석하고자 하는 단어**

키 (KEY) : 각 단어가 쿼리에 해당하는 단어와의 연관정도

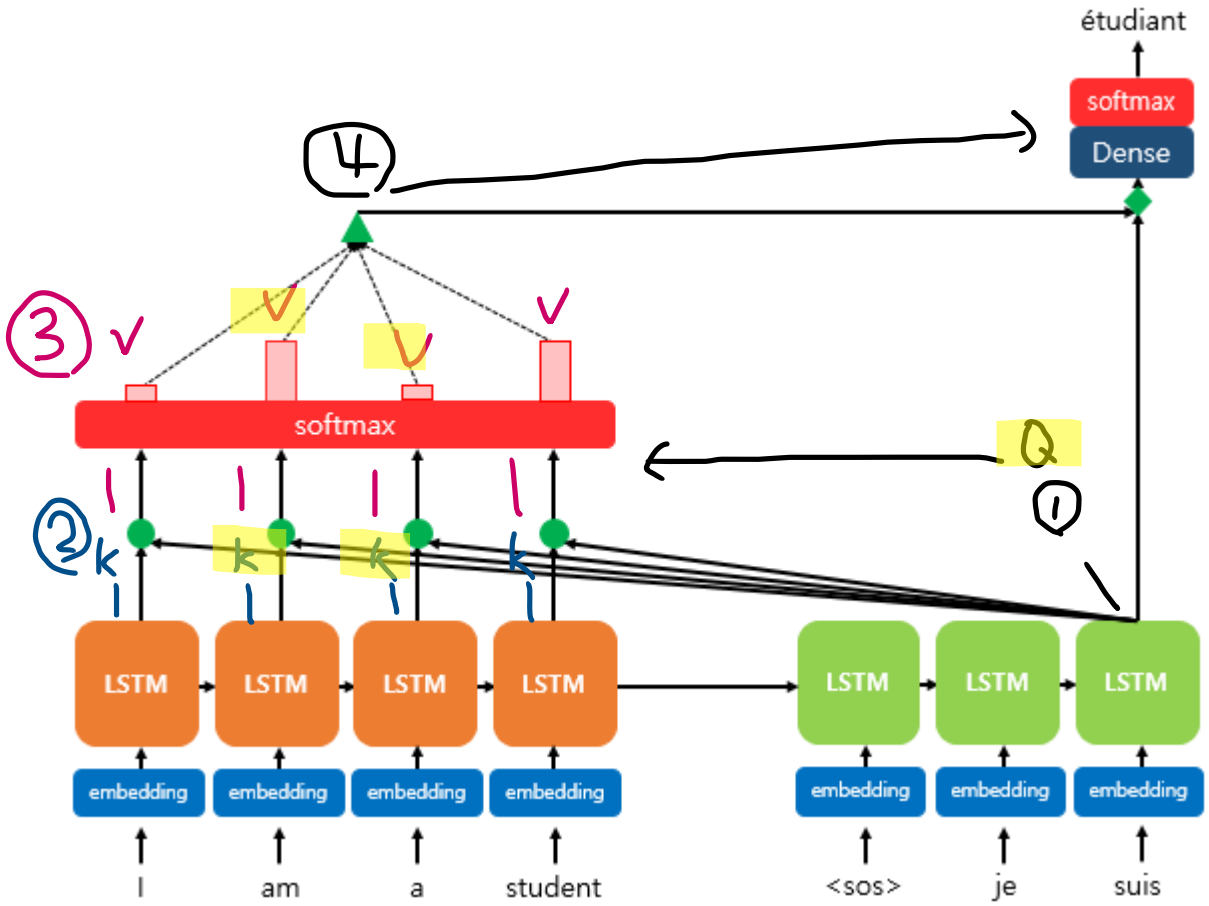
➔ **분석하고자 하는 단어와의 유사도를 나타낸 벡터**

밸류 (VALUE) : 키의 의미를 표현하는 가중치 벡터

➔ **문장에 있어 단어 (키)가 가지고 있는 중요도**

1. Review

RNN + Attention Mechanism



1. Decoder의 쿼리 (Q) 벡터와 Encoder의 키 (K) 벡터를 내적해 **어텐션 스코어 (Attention Score)**를 구합니다.

2. 어텐션 스코어를 **소프트맥스**하여 모든 값을 합치면 1이 되는 **어텐션 가중치(Attention Weight)**를 구합니다.

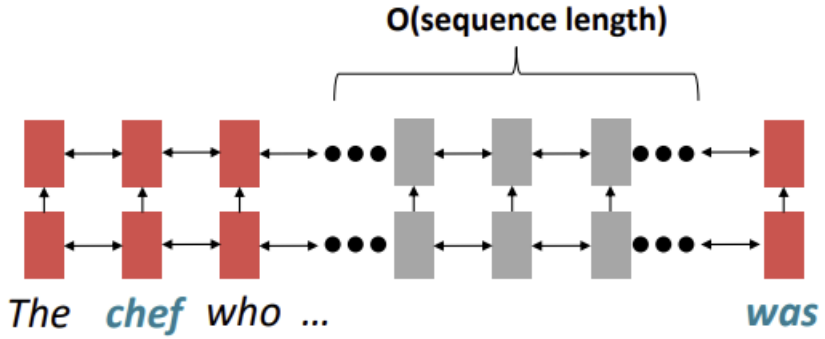
3. 어텐션 가중치와 각 단어의 밸류 (V) 벡터를 곱한다음, 모든 벡터의 가중합을 더해 **어텐션 값**을 구합니다.

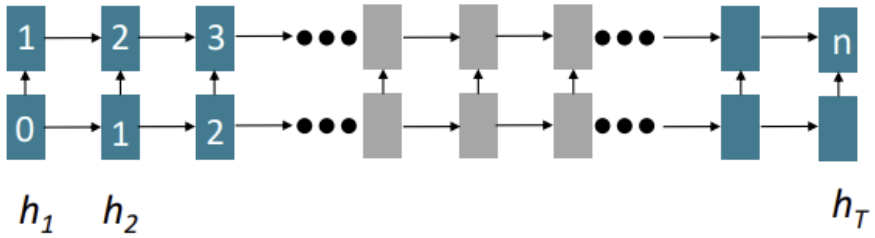
4. 구한 어텐션 값을 입력값으로 다음 단어를 예측합니다.

1. Review

Attention Mechanism - problems

Attention Mechanism은 기존 Recurrent Network (RNN, LSTM, GRU)에 Attention layer를 살짝 개입을 시킨 정도이기 때문에 Recurrent Network이 지닌 문제점을 가지고 있습니다.





Numbers indicate min # of steps before a state can be computed

1. 문장의 길이에 따라 연산 속도가 달라진다 : O(sequence length)
2. 순차적인 모델이기 때문에, GPU를 활용해 독립적인 연산을 진행하기 어렵다.

2. Transformer

트랜스포머 (Transformer)

기존 Encoder와 Decoder의 Recurrent Network을

전부 **Attention Mechanism**으로 대체합니다.

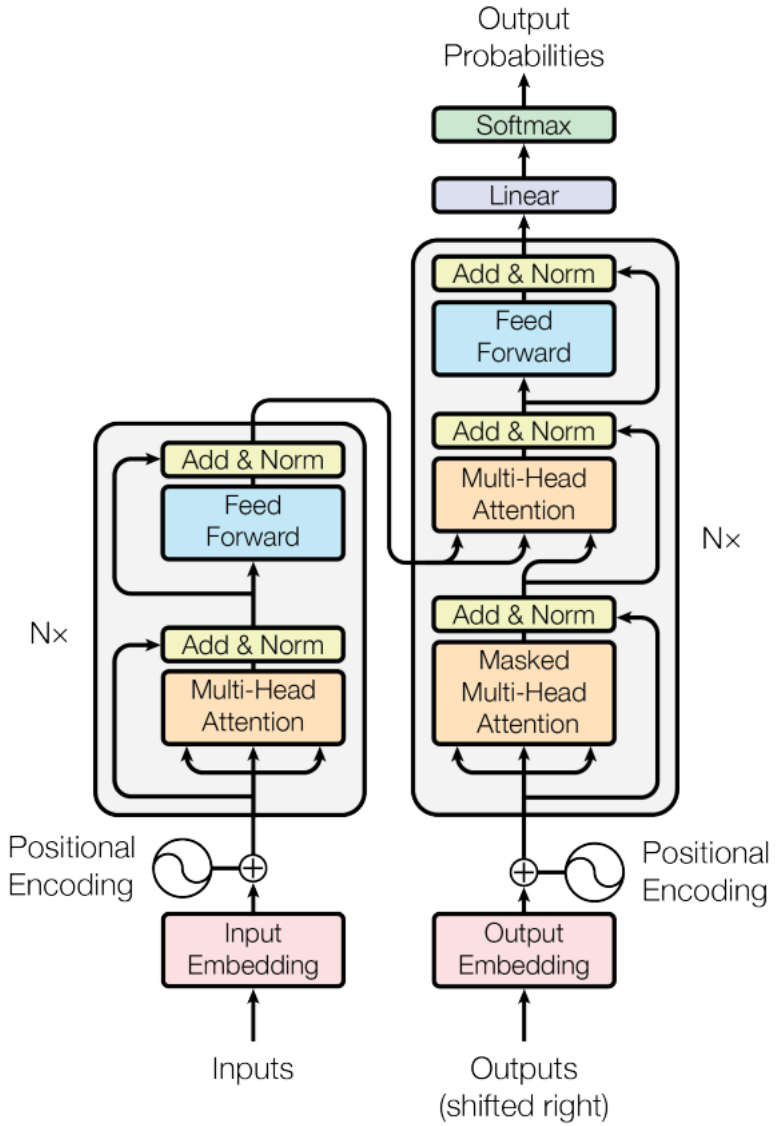
“Attention is all you need!”

순차적인 입력이 필요 없기 때문에

연산 속도가 O(1)으로 고정 + GPU 병렬 연산 가능

➔ 더 많은 파라미터를 더 빠른 시간에 학습이 가능하다

➔ ➔ 더 나은 결과를 산출할 수 있다.

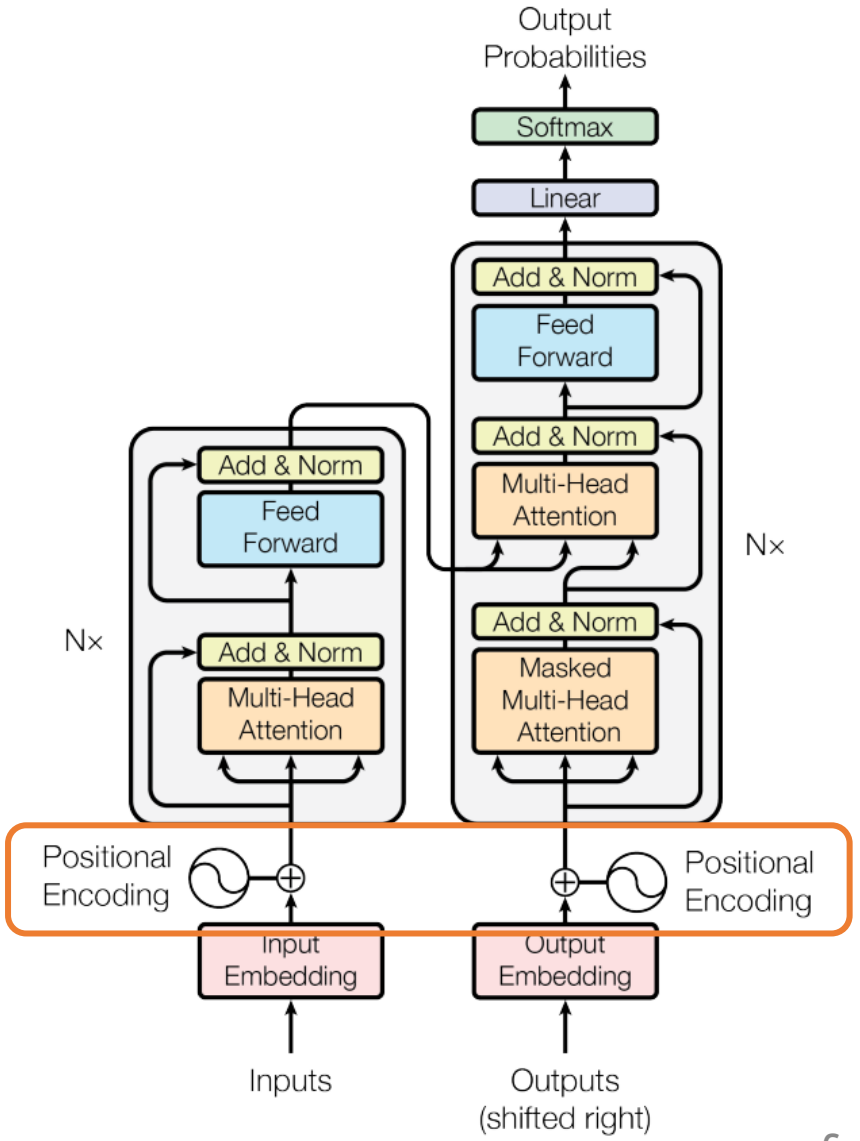


2. Transformer

Positional Encoding

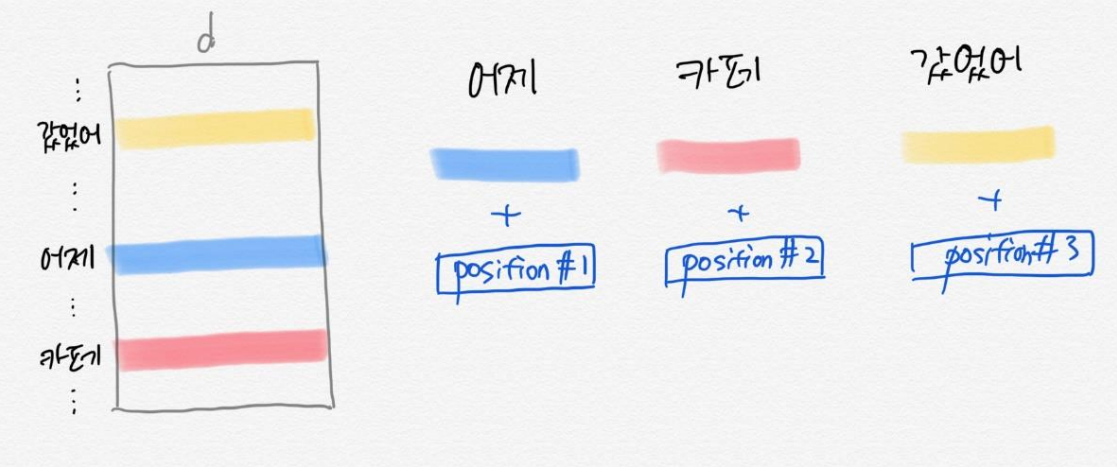
RNN 모델은 시퀀스 데이터의 순서 정보를
순서에 맞추어 차례대로 연산을 진행하는 방식으로 보존해
단어의 순서 정보는 간직했으나
연산 속도 O(sequence length)만큼의 손실이 있습니다.

하지만, 트랜스포머 모델은
모델에 입력하기 전 단어의 임베딩 벡터에 위치 정보를 더하는
포지셔널 인코딩(Positional Encoding) 단계를 거치기 때문에
Attenion layer에서 바로 연산하는 O(1)의 복합도를 가집니다.



2. Transformer

Positional Encoding



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

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

Pos = 문장에서 단어의 순서 (어제 = 1)

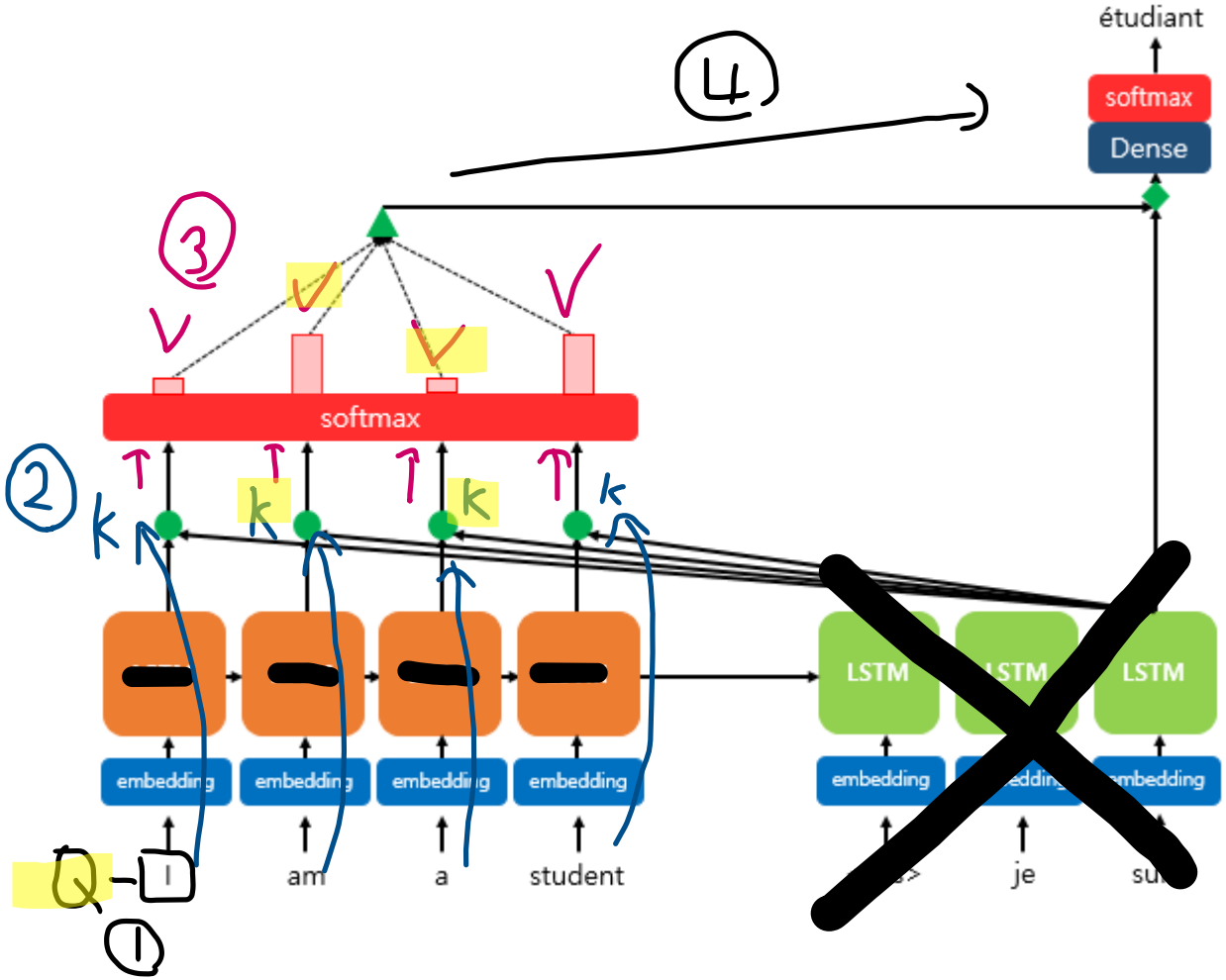
I = 전체 단어사전의 크기

문장 = “어제 카페 갔었어”

1. “어제” “카페” “갔었어” 단어 벡터를 사전 정의된 단어 사전에서 찾는다
2. 단어 사전에서 벡터를 가져와 고유한 위치 정보를 더한다.

2. Transformer

트랜스포머의 셀프 어텐션 (self-attention)



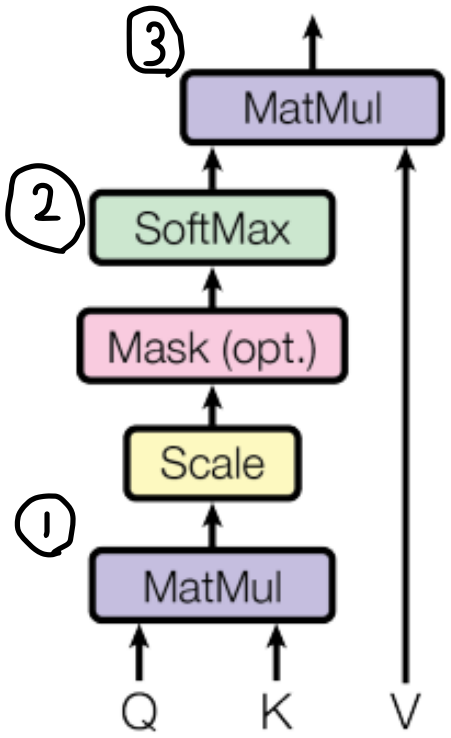
셀프 어텐션은 기존 Decoder에서 쿼리 벡터가 나오고 Encoder에서 키, 밸류 벡터를 계산하는 어텐션 모델과는 달리 자기 혼자 스스로 어텐션 연산을 수행합니다.

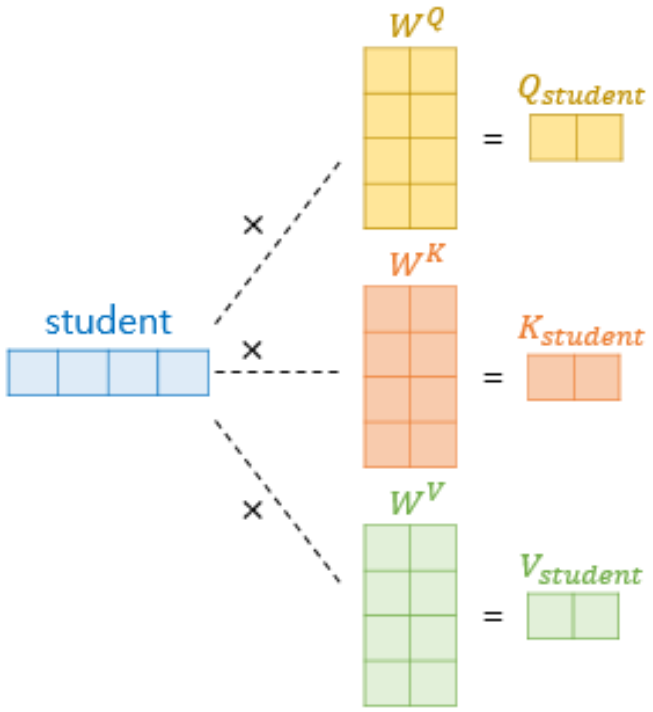
1. "I" 단어를 쿼리 (Q) 벡터로 지정합니다.
2. "I", "am", "a", "student"의 키 (K) 벡터와 내적을 진행해 어텐션 스코어를 구합니다.
3. 어텐션 가중치와 각 단어의 밸류 (V) 벡터를 곱한다음, 모든 벡터의 가중합을 더해 어텐션 값을 구합니다.
4. 구한 어텐션 값을 입력값으로 다음 단어를 예측합니다.

2. Transformer

Scaled Dot-Product Attention

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





쿼리, 키, 벡터를 외부 (디코더)에서 가져오지 않고

계산하는 방법 => 단어 벡터에 가중치 벡터를 곱한다!

X = 단어 ('student')을 숫자 벡터로 임베딩한 벡터

W^Q = 쿼리 벡터를 만드는 가중치 벡터

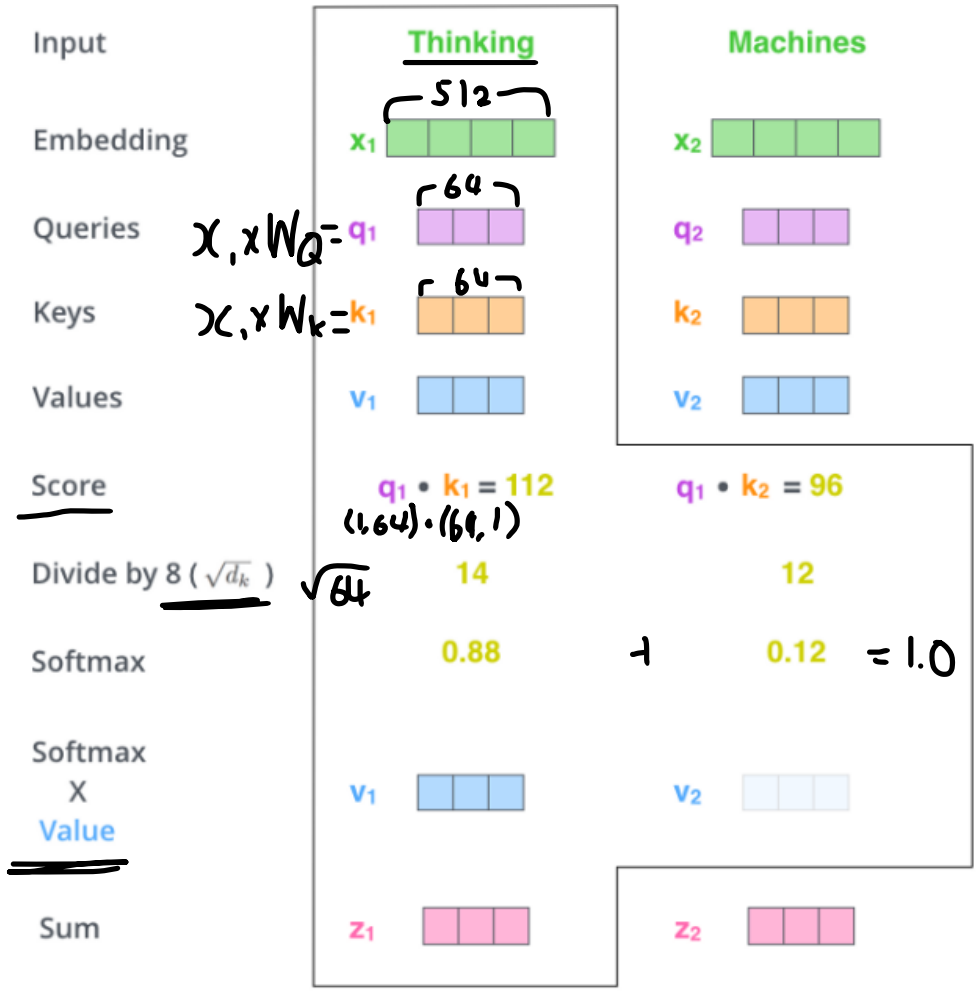
W^K = 키 벡터를 만드는 가중치 벡터

W^V = 밸류 벡터를 만드는 가중치 벡터

2. Transformer

Scaled Dot-Product Attention

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



문장 "Thinking Machines"에서 Thinking을 어텐션 할 시,

1. "Thinking"을 (1, N = 512)차원의 숫자벡터로 임베딩
2. W^Q, W^K, W^V 가중치 벡터와 곱해 Q, K, V을 만든다
3. 어텐션 스코어를 구한다 (Q와 K)의 내적
4. 어텐션 스코어를 $\sqrt{d_k}$으로 나누어 스케일링한다

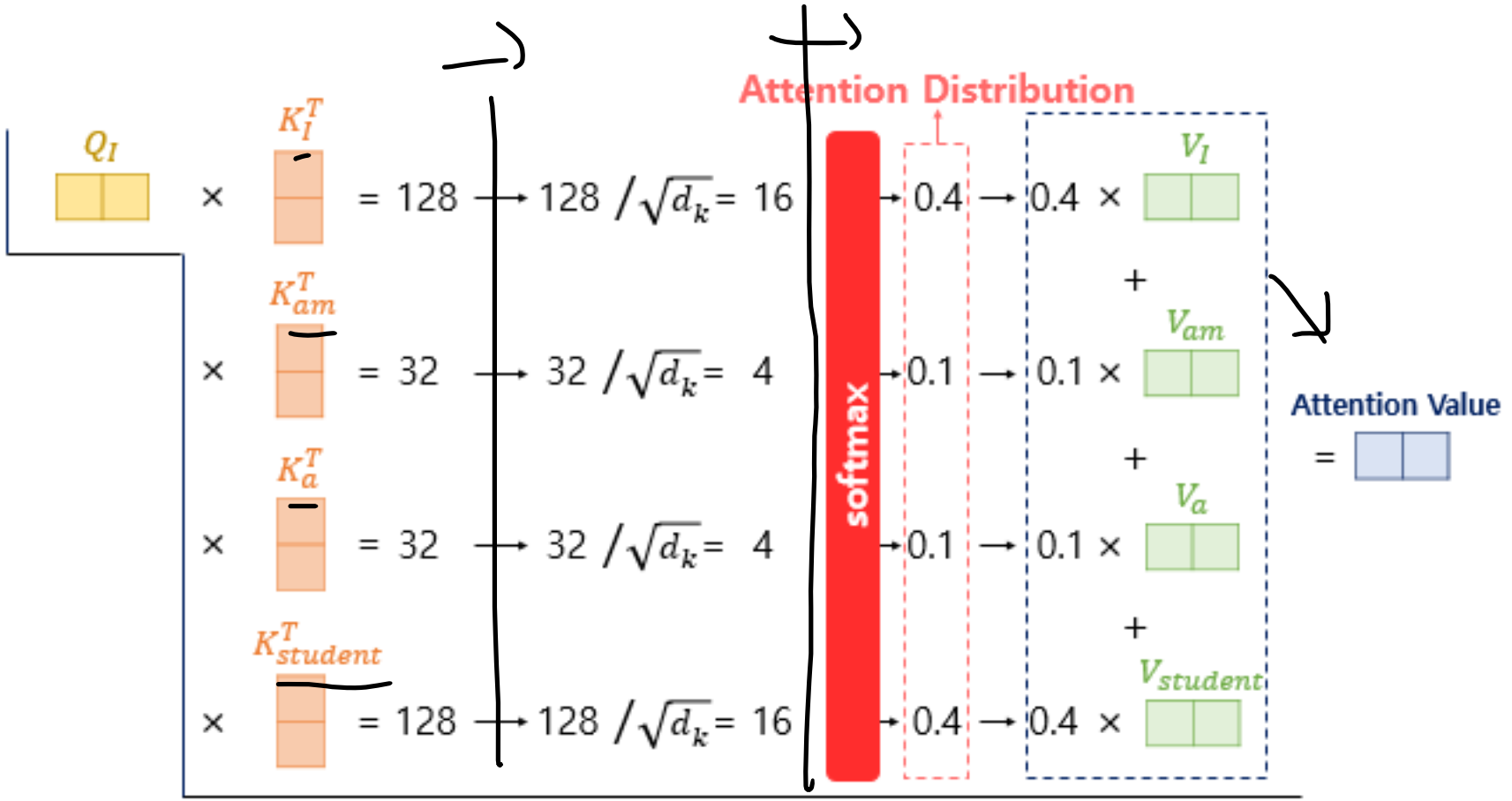
→ 내적의 결과물이 커지면 softmax 함수를 거쳤을 때 gradient vanishing 문제가 발생할 수 있기 때문!

5. Softmax값을 밸류 벡터와 곱해 어텐션 값을 구합니다.

2. Transformer

Scaled Dot-Product Attention

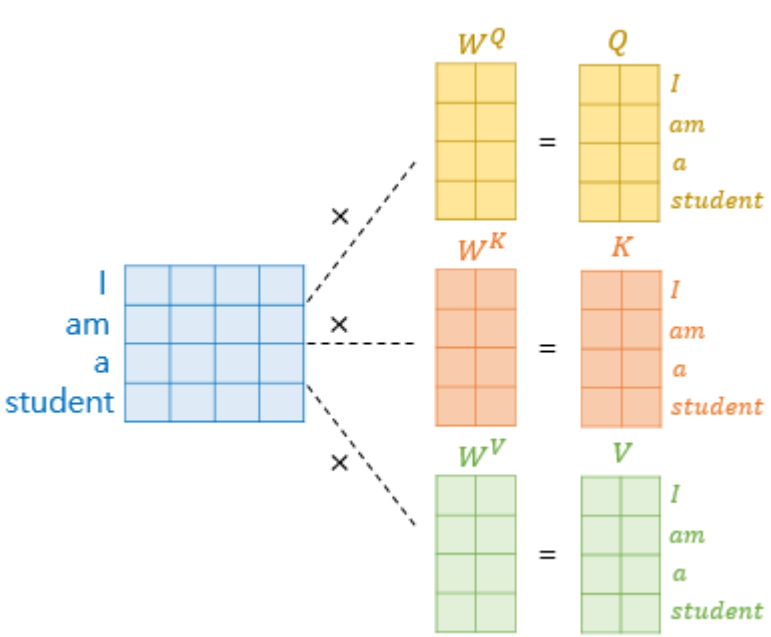
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

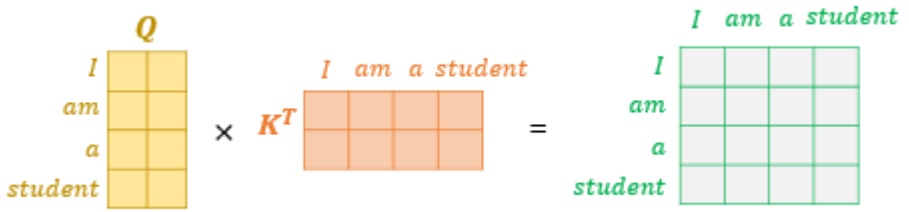


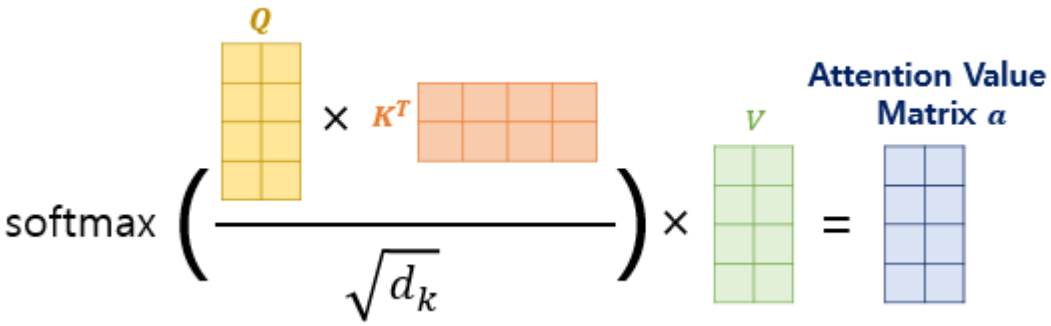
2. Transformer

Scaled Dot-Product Attention

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



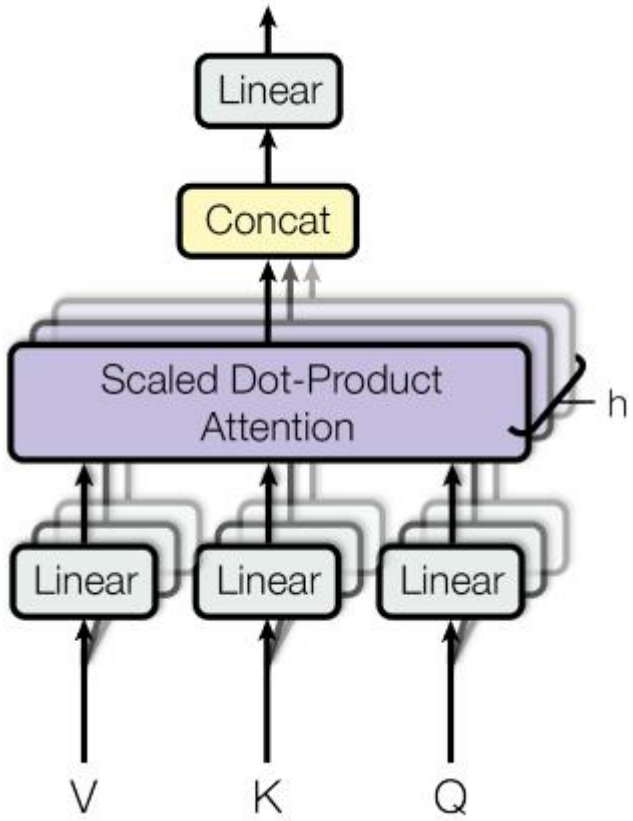




2. Transformer

Multi-Head Attention

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



트랜스포머 모델은 여러 개의 self-attention layer를 num_heads만큼 병렬 연결한 Multi-Head Attention을 사용합니다.

Self-attention에 비해 multi-head attention은 num_heads만큼의 어텐션 연산을 더 많이 수행했기 때문에,

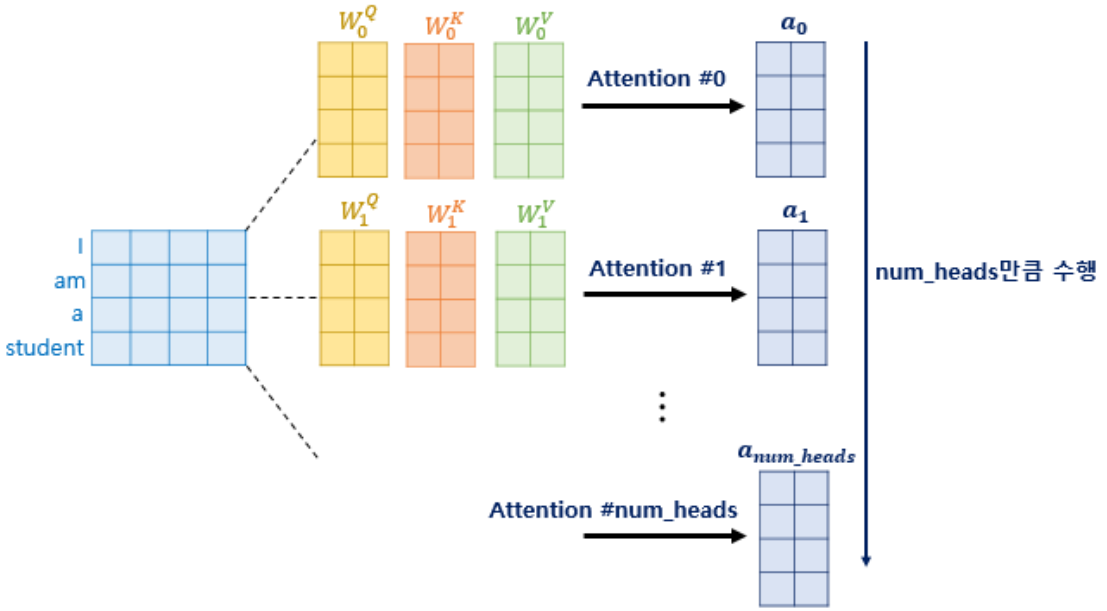
1. 한 가지 결과만 산출하는 self-attention과는 달리 multi-head attention은 num_heads만큼의 결과가 연산되기 때문에 더 다양한 결과를 만들 수 있고, 해당 결과들을 종합한 앙상블 학습을 진행할 수 있습니다.

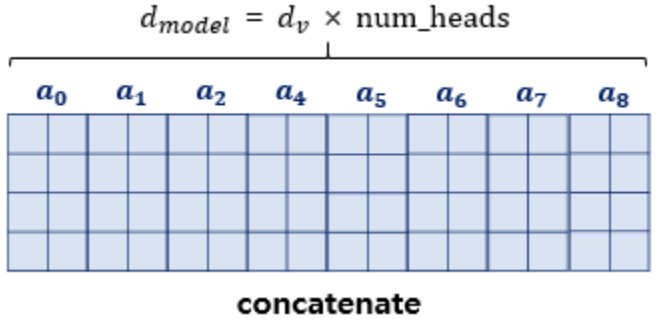
2. Multi-Head attention은 self-attention보다 N배 더 많이 연산하는 것이 아니라, N개의 layer를 전부 각각의 GPU에 할당할 수 있습니다 -> 연산속도 변화 X

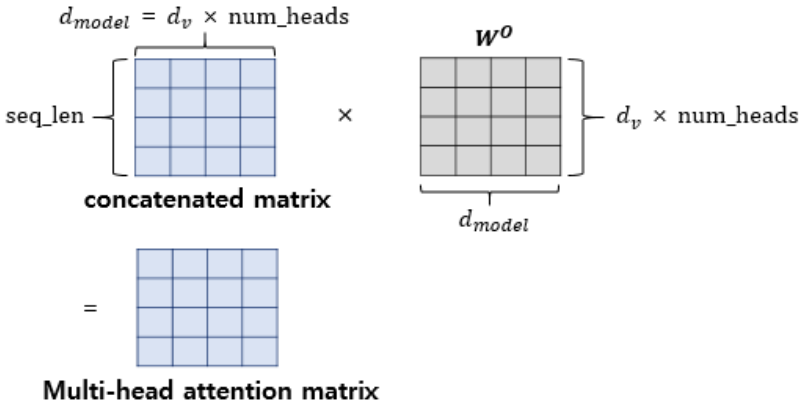
2. Transformer

Multi-Head Attention

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

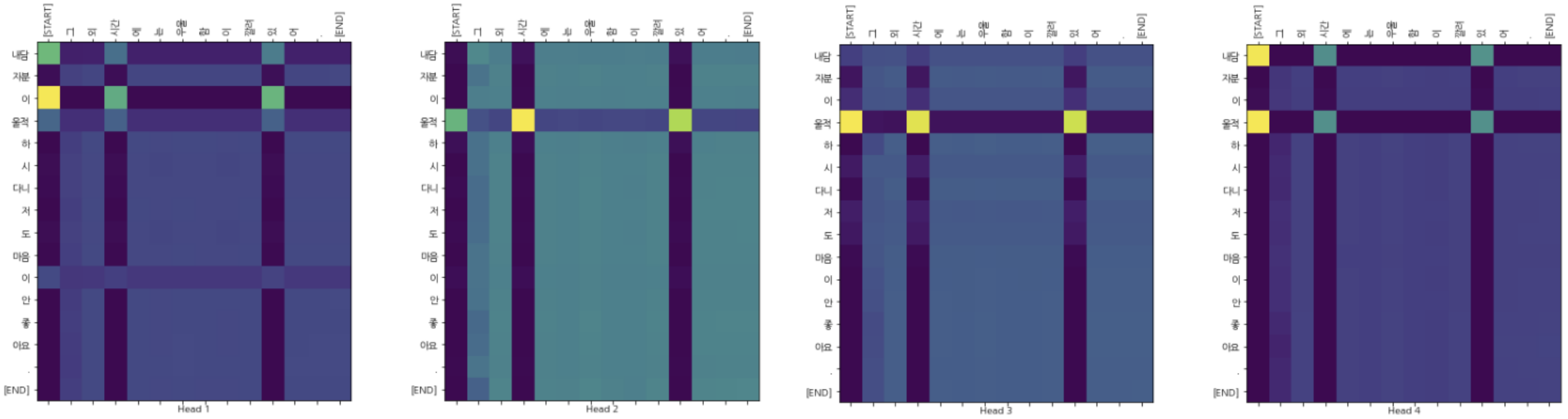






2. Transformer

Multi-Head Attention



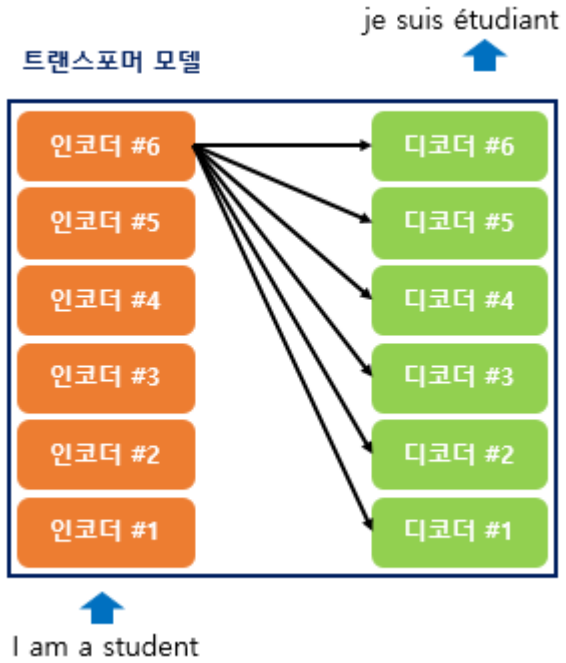
2. Transformer

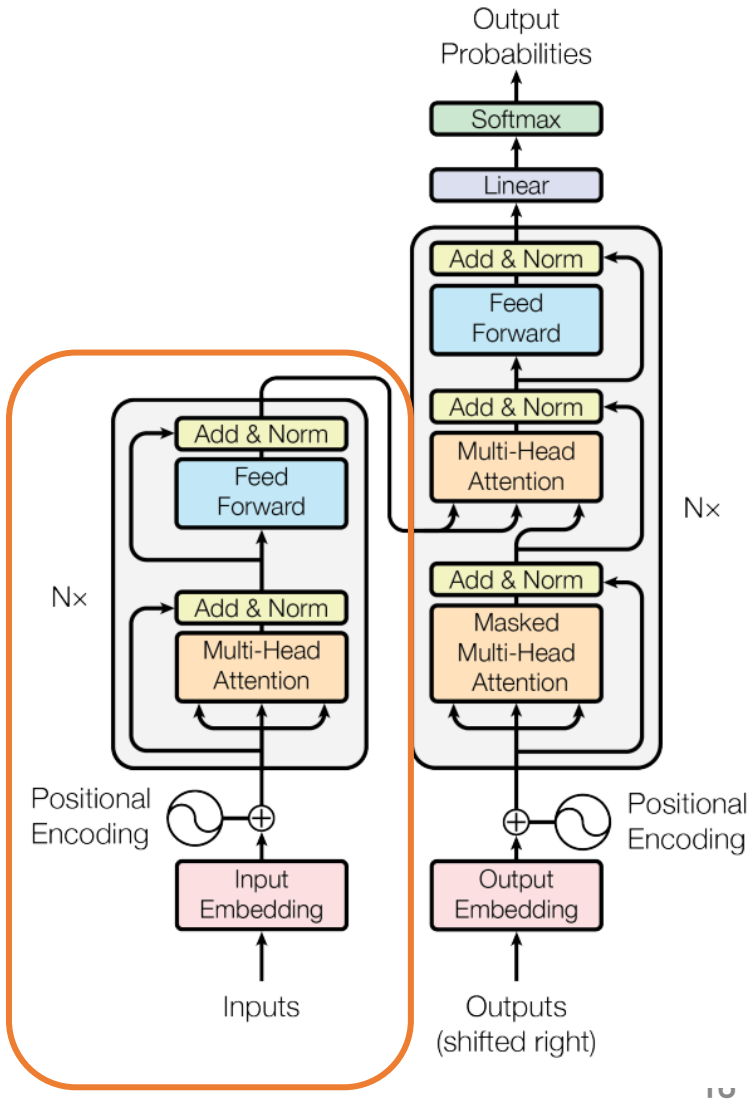
Encoder

트랜스포머의 Encoder 층은 N개의 동일한 Encoder 층으로 이루어져 있습니다.

1 ~ N – 1 Encoder 층의 output은 그 다음 Encoder 층의 input으로 입력됩니다.

N 번째 Encoder 층의 output은 1 ~ N개의 Decoder 층에 동일하게 분산됩니다.





2. Transformer

Encoder

Encoder 층은 2개의 Sub-layer를 가집니다.

Multi-head self-attention : Encoder로 입력된 단어 벡터의 쿼리, 키, 밸류를 받아 어텐션 병렬 연산을 수행합니다.

Feed Forward Network : 하나의 neural network층을 통과시킵니다.

$$FFN(X) = \max(0, xW1 + b1)W2 + b2$$

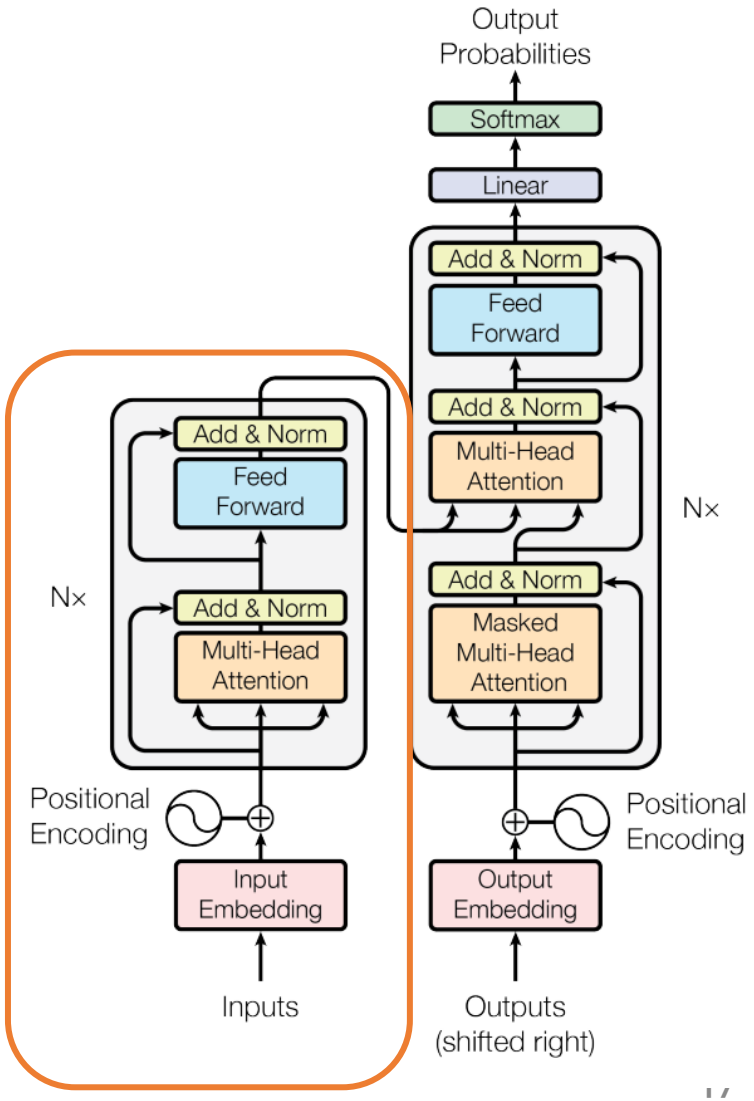
Residual connection : Attention을 수행하기 전 벡터와 한 벡터를 더해주는 과정

Why? ➔ Vanishing gradient problem 방지

Output of the layer : LayerNorm(x + Sublayer(x))

x + Sublayer(x) -> residual connection

LayerNorm -> 레이어의 output 벡터를 mean : 0, std :1인 정규분포로 정규화

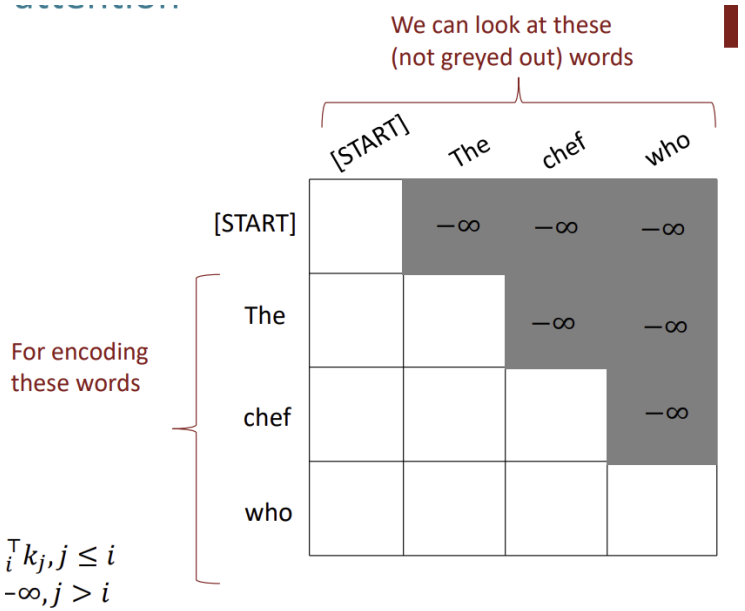


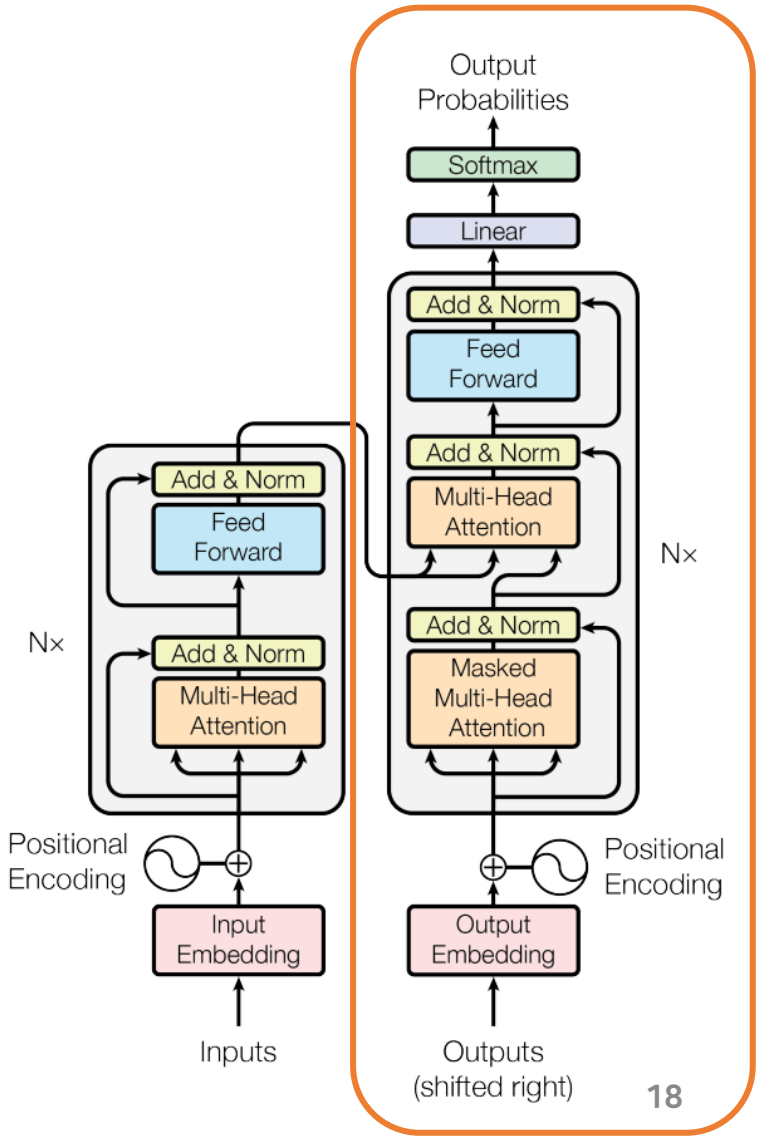
2. Transformer

Decoder

Encoder 층은 3개의 Sub-layer를 가집니다.

Masked Multi-head self attention: Decoder가 문장을 예측할 때 단어 이후의 문장을 참고하는 현상을 방지하기 위해, 해당 단어 이전의 문맥만 학습하게 제한을 둡니다.



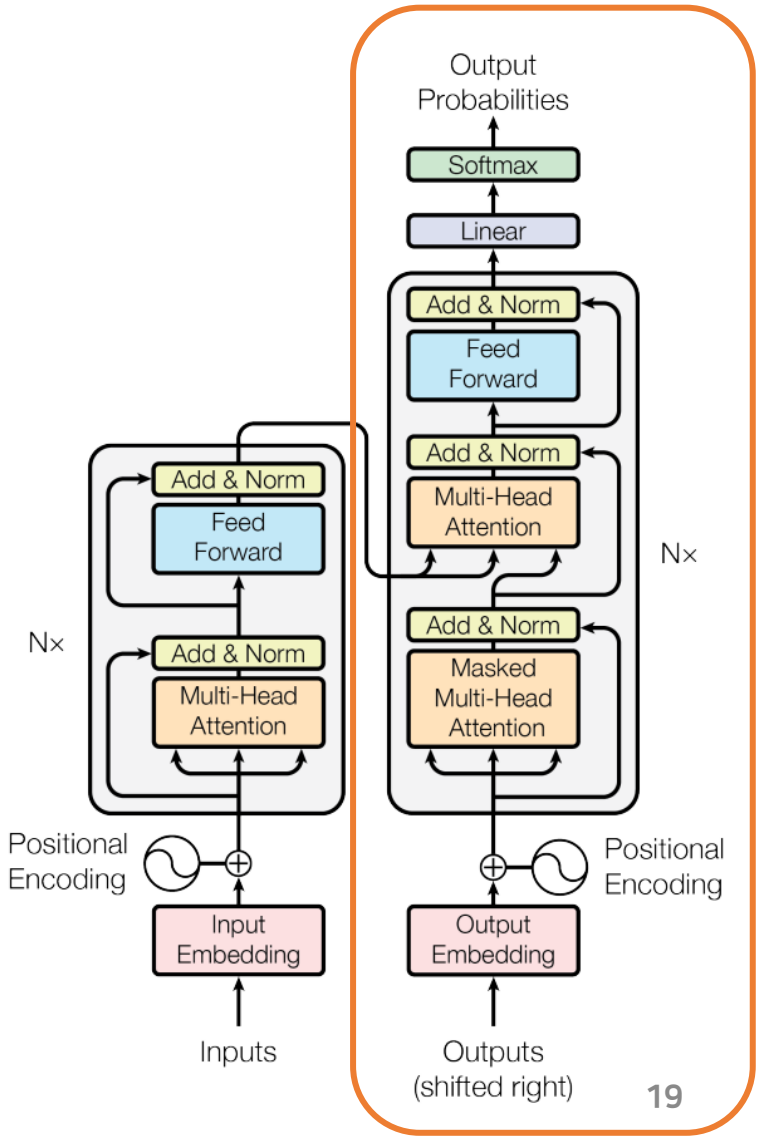


2. Transformer

Decoder

Multi-head attention over the output of the encoder:

앞선 RNN + attention 모델에서 attention layer와 마찬가지로,
Encoder의 마지막 N번째 layer에서 벡터의 키(K)와 밸류(V) 벡터를 가져와
Decoder의 쿼리(Q) 벡터와 연산을 진행해
어텐션 값을 출력합니다.



3. Results

Great Results with Transformers

First, Machine Translation from the original Transformers paper!

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**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$

3. Results

Great Results with Transformers

Next, document generation!

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

The old standard

Transformers all the way down.

DATA SCIENCE LAB

발표자 @@@ 010-1234-5678
E-mail: @@@@gmail.com