



# Lecture 9. Attention Mechanism & Transformers

김수한, 주연우

# 목차

#01 Limitations of RNN-based models

#02 Self-attention

#03 Transformer

- 1) Positional Encoding
- 2) 인코더의 서브층
- 3) 인코더에 사용하는 기법
- 4) 디코더의 첫번째 서브층

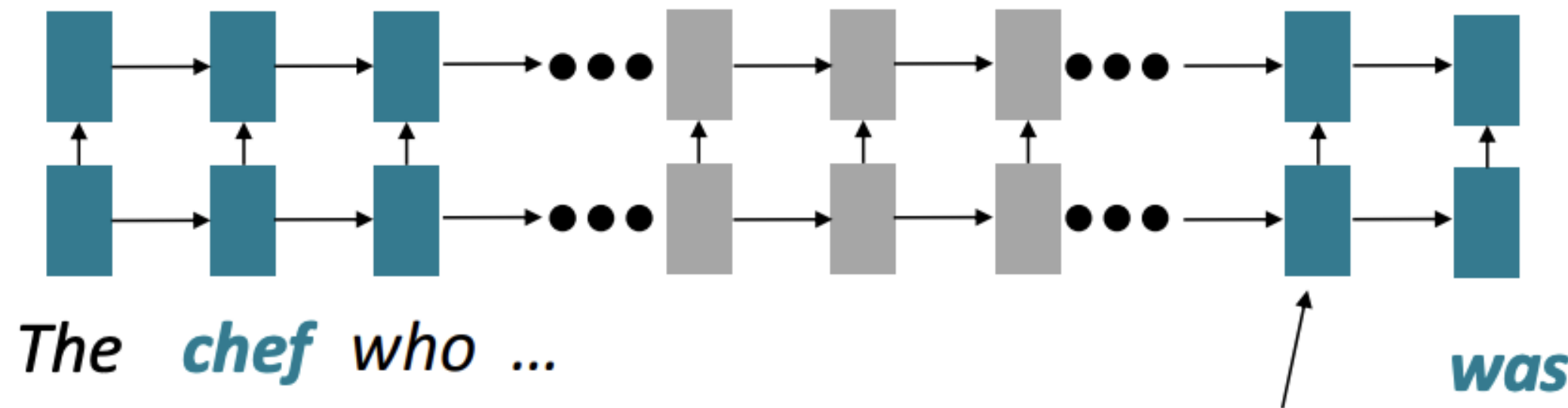


# 1. Limitations of RNN-based models



# 1. Limitations of RNN-based models

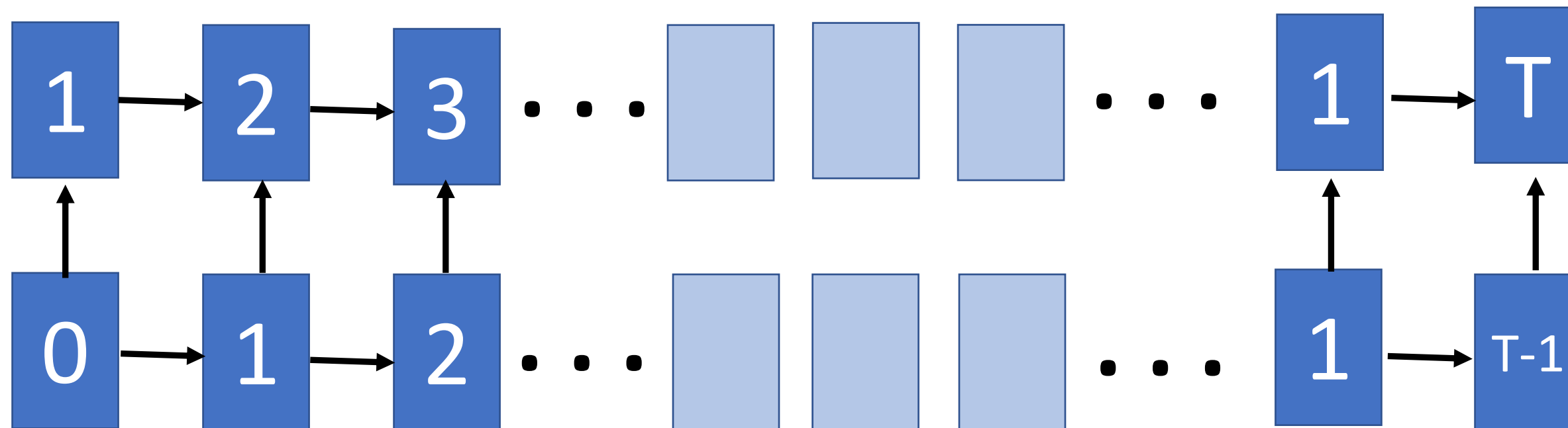
## #1 Linear interaction distance



Info of **chef** has gone through  $O(\text{sequence length})$  many layers!

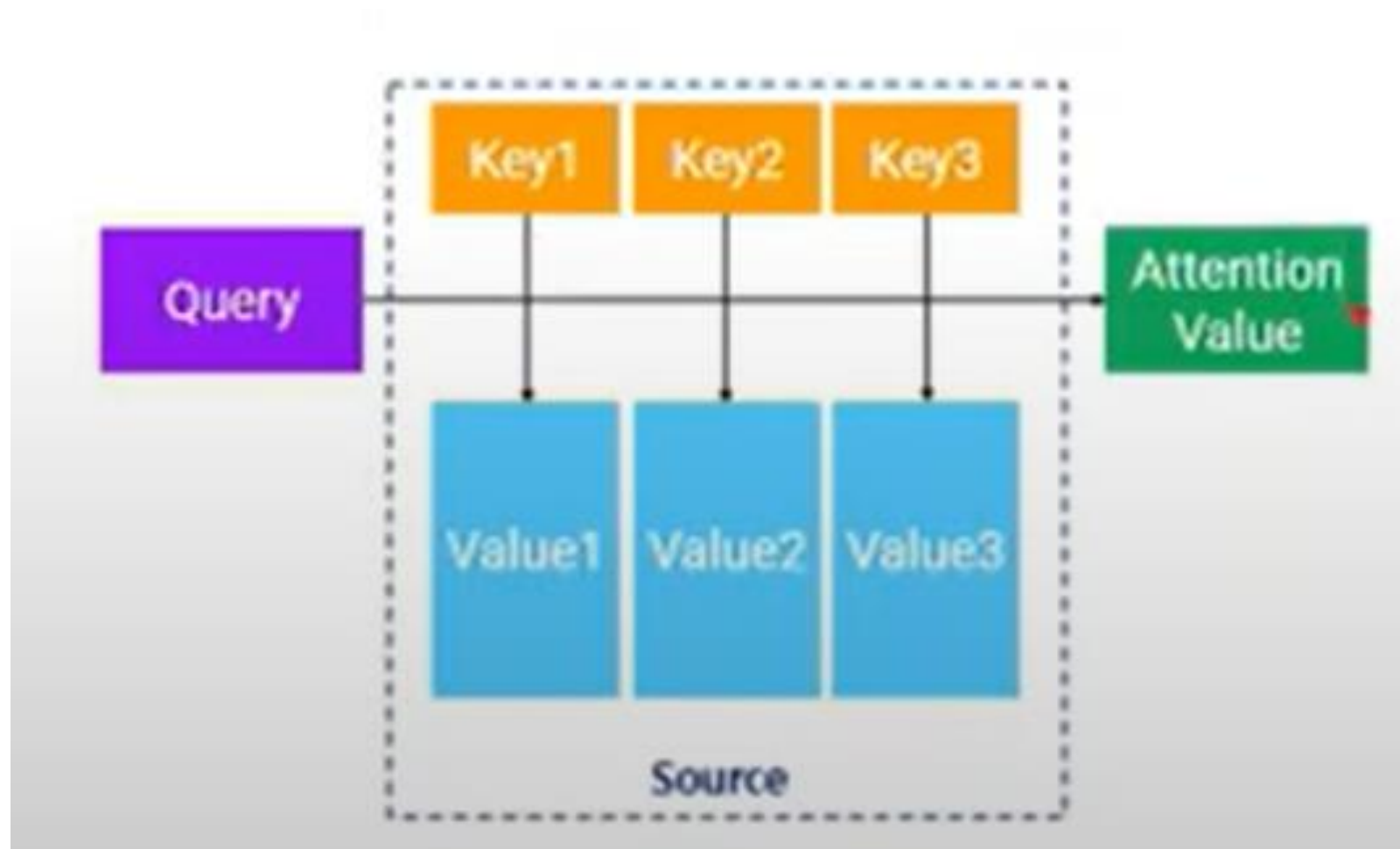
# 1. Limitations of RNN-based models

## #2 Lack of parallelizability



# 1. Limitations of RNN-based models

## #3 Attention



Query: 기준

- query와 더 관련성 있는 candidate에 가중치를 주어 attention value 도출
- attention은 각 단어의 표현을 query로 처리하여 value의 집합에 접근하거나 정보를 통합

## 2. Self-attention



# 2. Self-attention

## #1 Self-attention

**Query**: t시점에서 decoder의 hidden state

**Keys**: 모든 시점에서 encoder의 hidden state

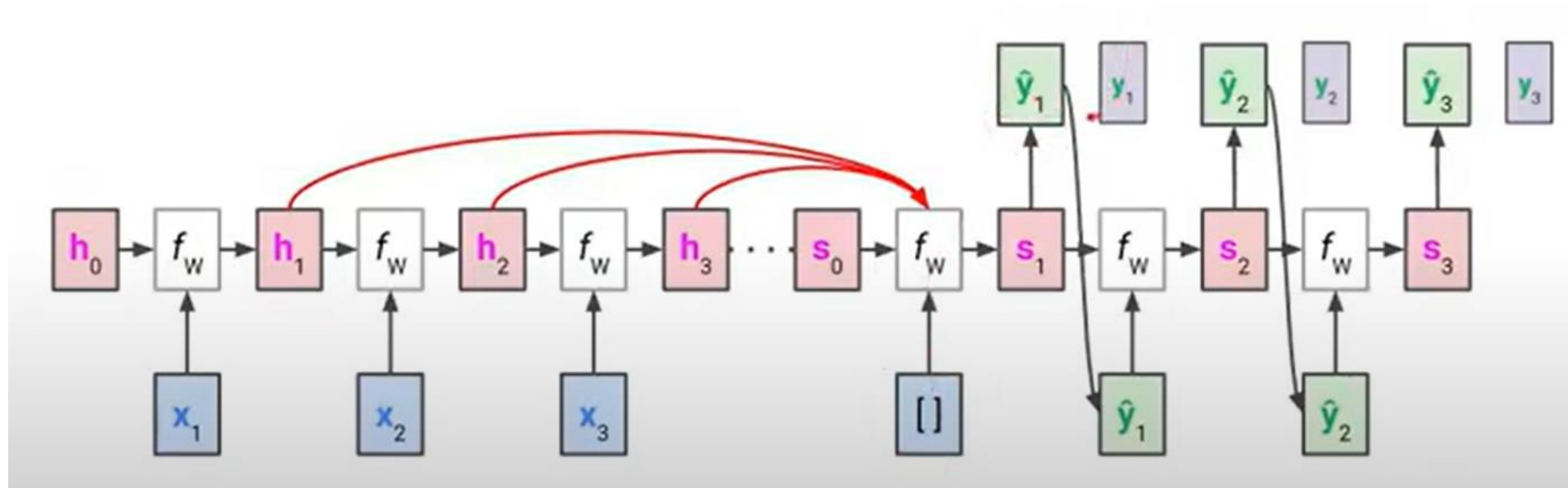
**Values**: 모든 시점에서 encoder의 hidden state

Self-attention은 Query, Key, Value가 서로 같을 때를 의미,  
즉 attention을 자기자신에게 수행한다는 의미



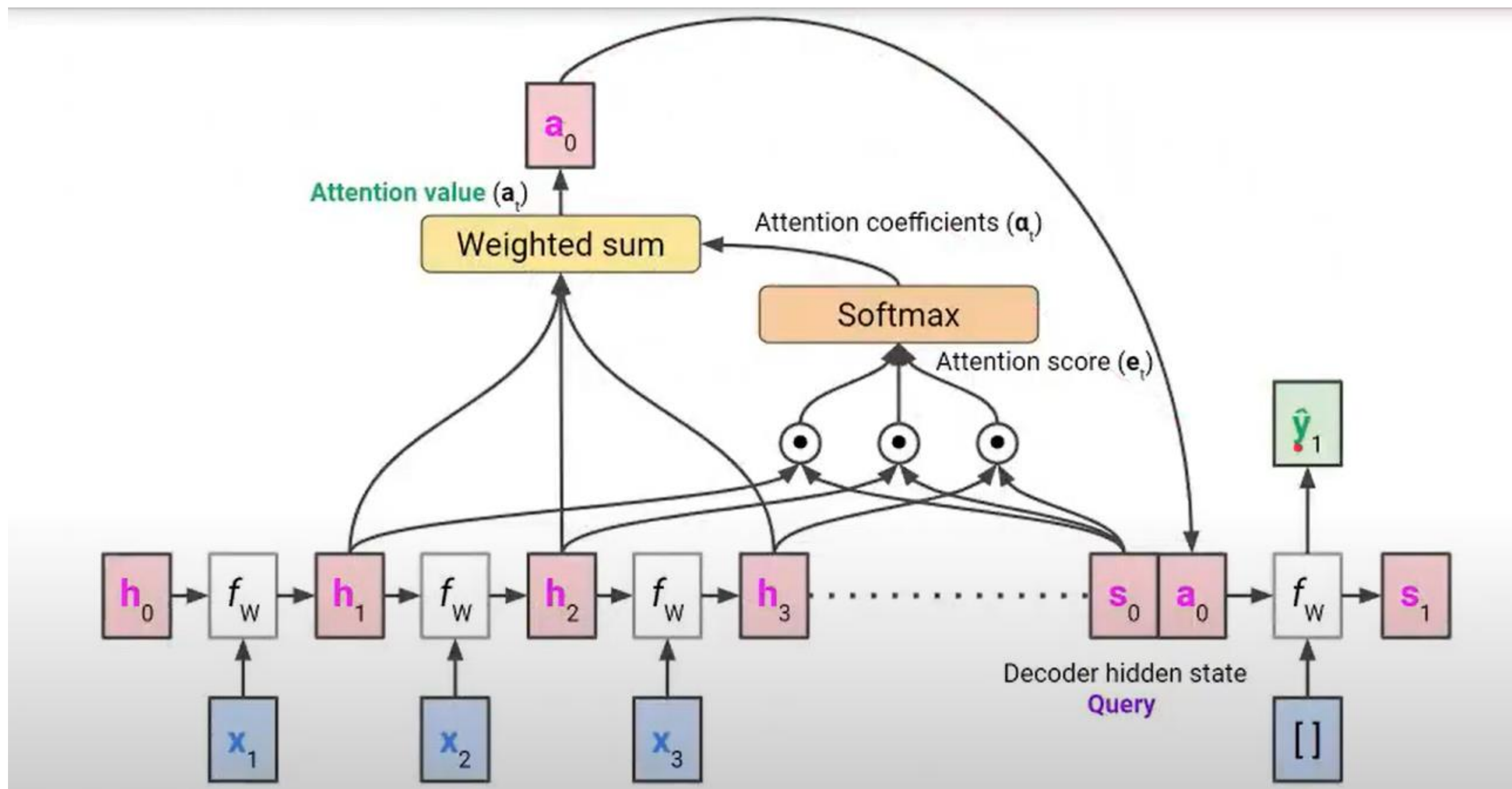
# 2. Self-attention

## #1 Self-attention



# 2. Self-attention

## #2 Attention의 구조



# 2. Self-attention

## #2 Attention의 구조

Start with encoder hidden states:  $\mathbf{h}_1, \dots, \mathbf{h}_T \in \mathbb{R}^h$

Decoder state at time step  $t$ :  $\mathbf{s}_t \in \mathbb{R}^h$

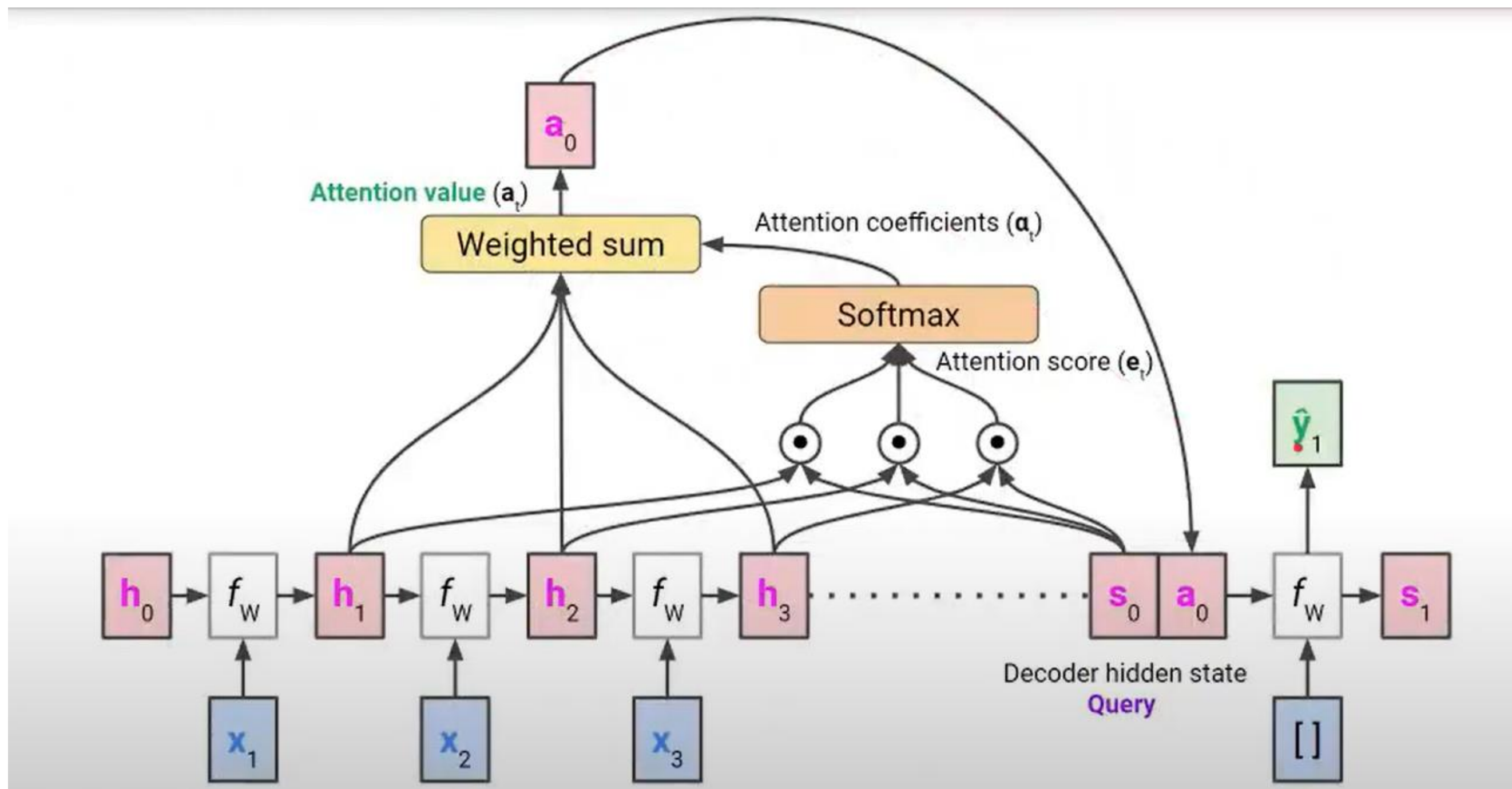
Compute the attention scores for this time step  $t$ :  $\mathbf{e}_t = [\mathbf{s}_t^\top \mathbf{h}_1, \dots, \mathbf{s}_t^\top \mathbf{h}_T] \in \mathbb{R}^T$

Take softmax to get attention coefficients for this time step  $t$ :

$$\alpha_t = \text{softmax}(\mathbf{e}_t) = \frac{\exp\{\alpha_t\}}{\sum_{\tau} \exp\{\alpha_{\tau}\}} \in \mathbb{R}^T$$

# 2. Self-attention

## #2 Attention의 구조



# 2. Self-attention

## #2 Attention의 구조

Take wighted sum of the encoder hidden states according to the attention coefficients to get the attention value:

$$\mathbf{a}_t = \sum_{i=1}^T [\alpha_t]_i \mathbf{h}_i \in \mathbb{R}^h$$

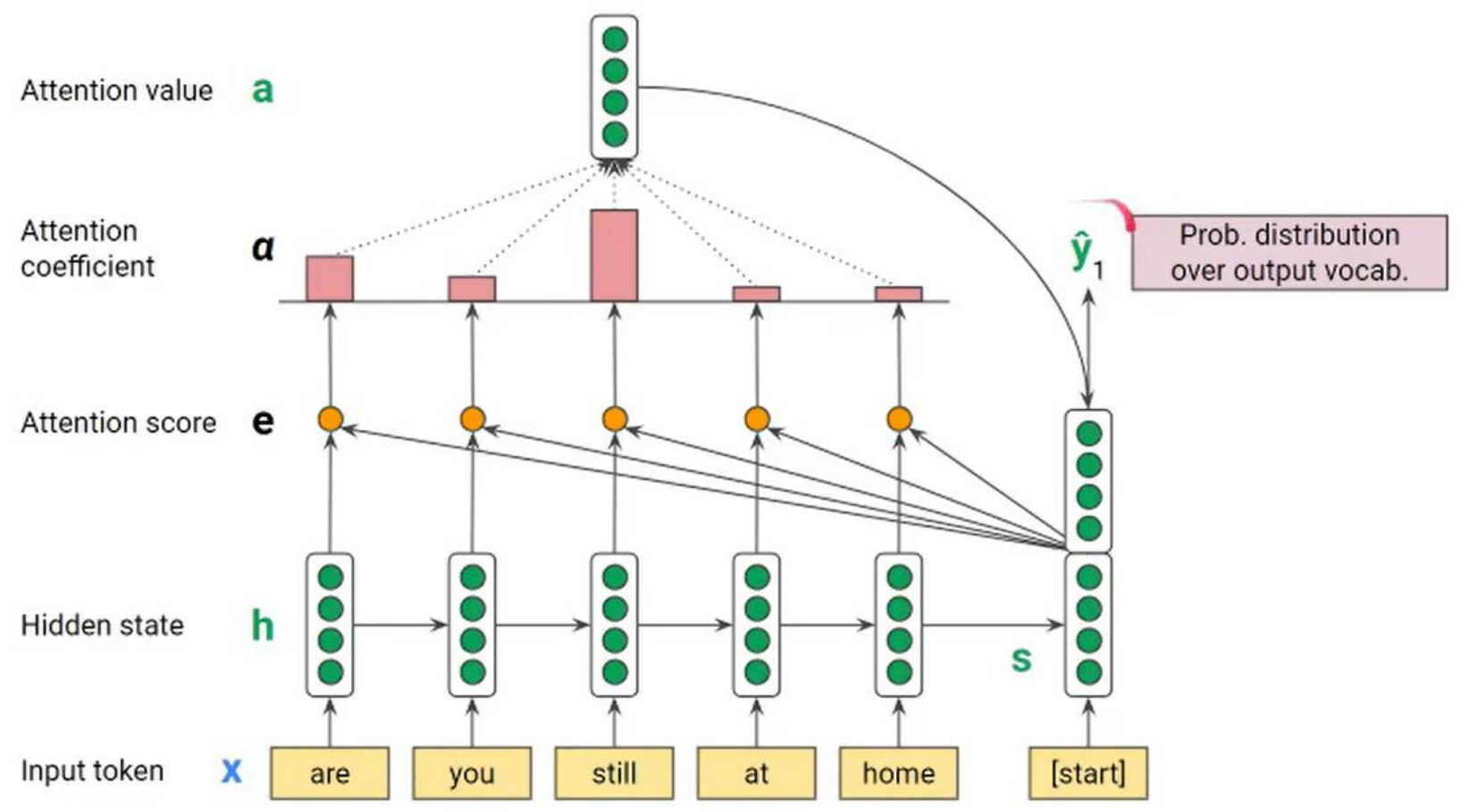
Concatenate the attention value with the decoder hidden state and proceed as in the vanilla seq2seq model:

$$[\mathbf{a}_t; \mathbf{s}_t] \in \mathbb{R}^{2h}$$

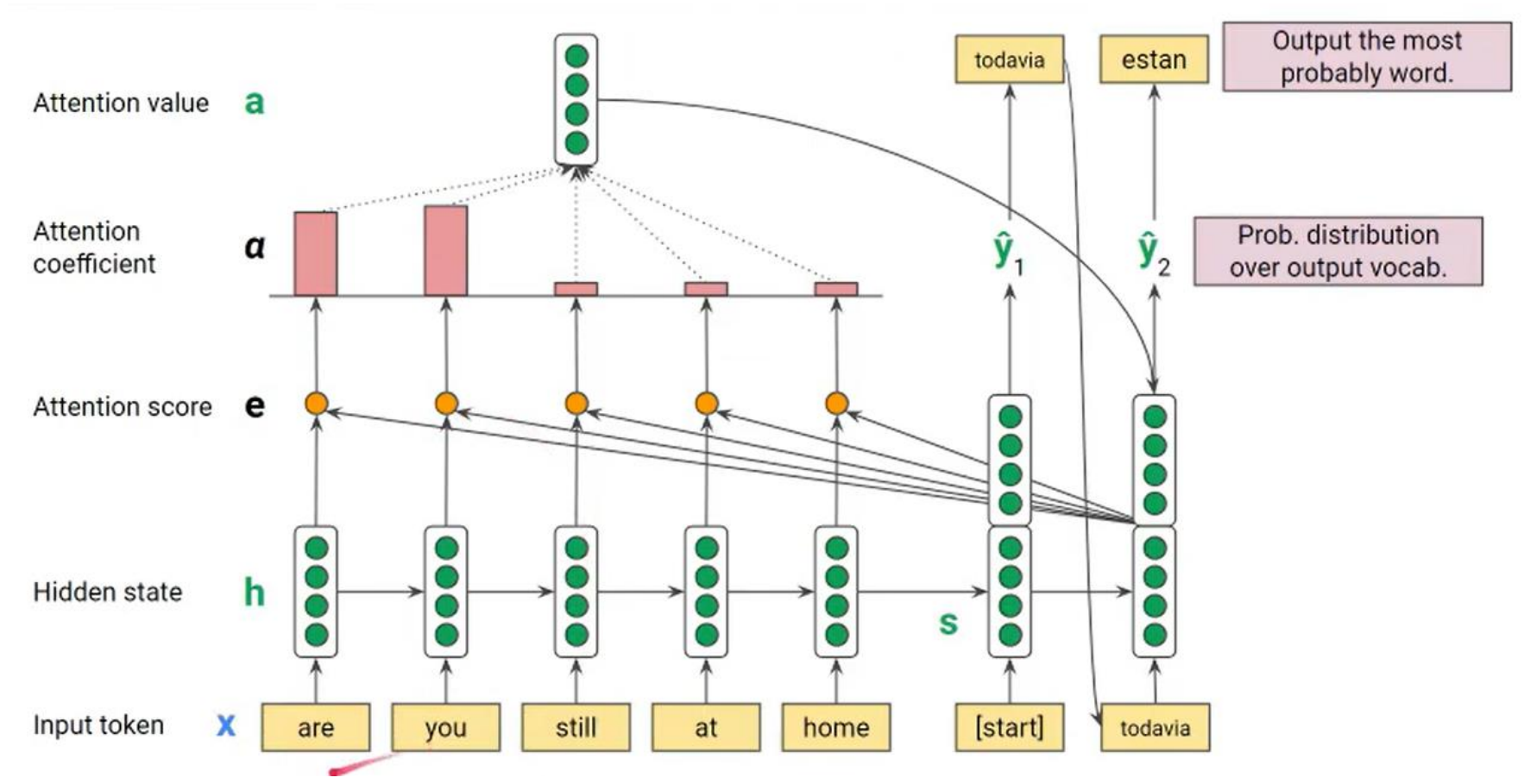


# 2. Self-attention

## #2 Attention의 구조



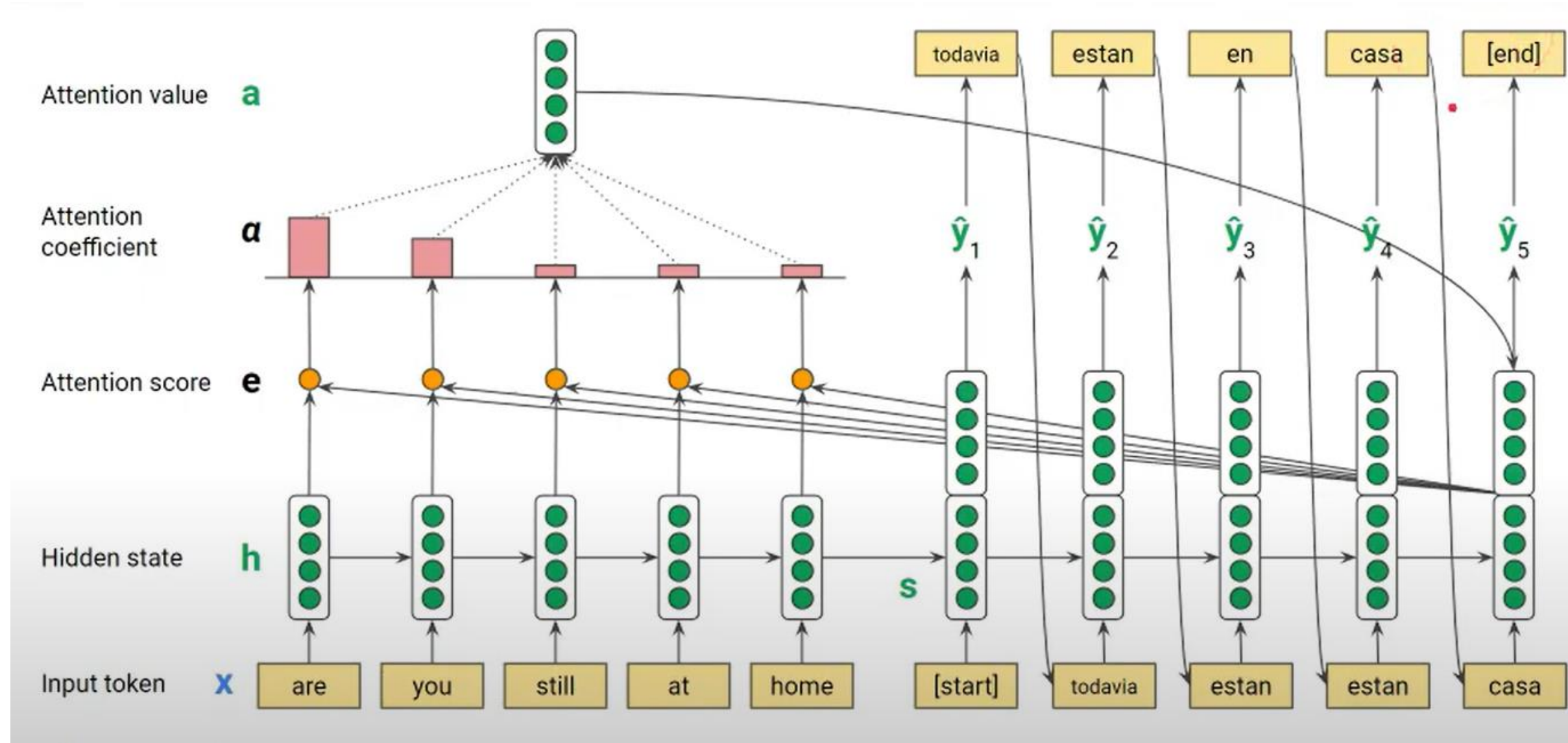
[1<sup>st</sup> step]



[2<sup>nd</sup> step]

# 2. Self-attention

## #2 Attention의 구조



-어느 단어에 집중해야 하는지 학습을 통해 가중치 부여

# 2. Self-attention

## #3 Barriers

sequence order가 없다 → self-attention은 순서 정보를 만들지 않기 때문

가중 평균을 이용하기 때문에 attention만으로 non-linear를 구현할 수 없다

Sequence 예측 시 미리 결과를 알지 못하도록 해야함



Input에 position representations 추가

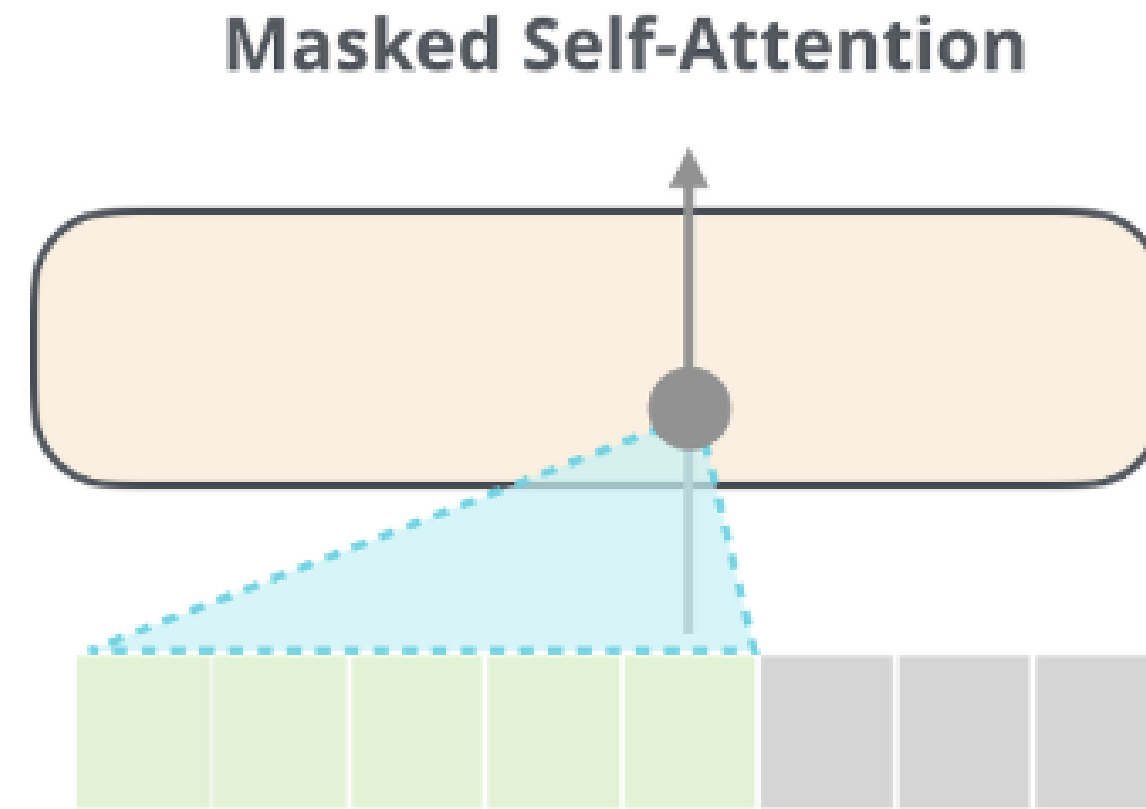
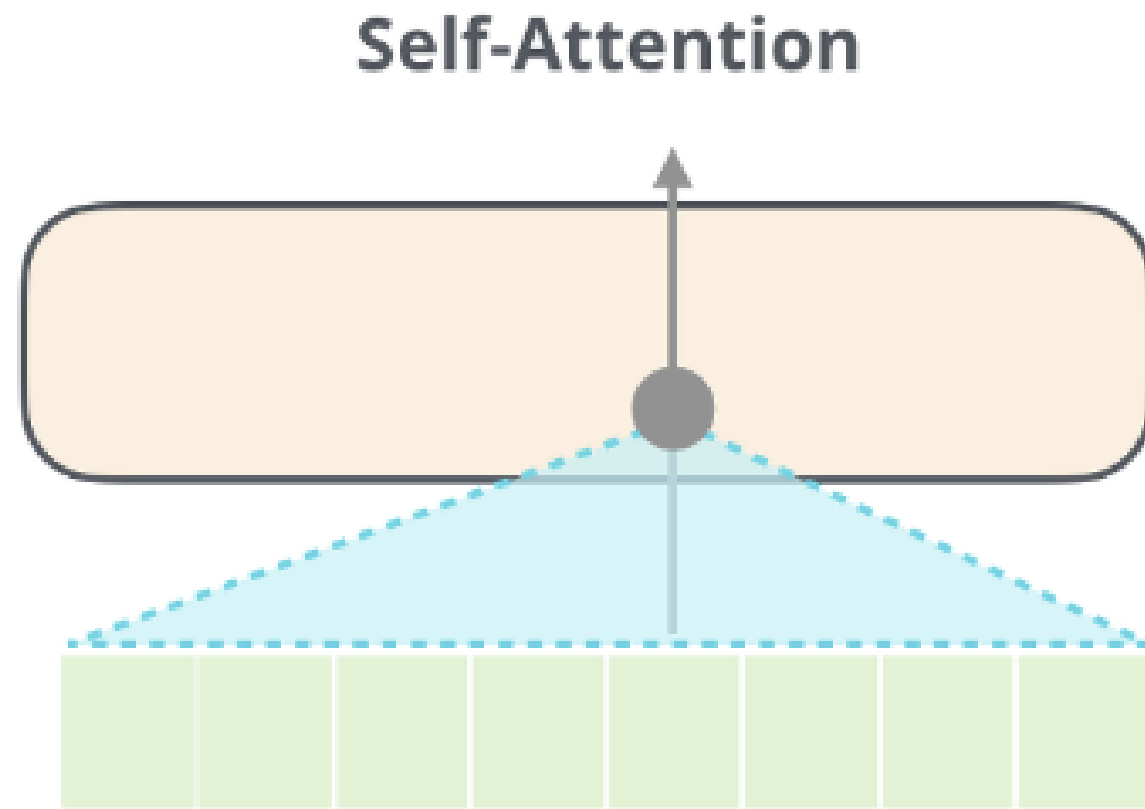
각 self-attention의 output에 동일한 feedforward network 적용

self-attention의 input에 masking한다



# 2. Self-attention

## #4 Masking



- 타깃 단어 뒤에 위치한 단어가 self-attention에 영향을 주지 않도록 가린다
- 병렬화를 가능하게 하기 위해 future words에 대한 attention을  $-\infty$ 로 설정하여 해당 attention을 masking한다.

# 2. Self-attention

## #5 Necessities for a self-attention building block

Self-attention: the basis of the method

Position representations: self-attention은 입력의 순서가 지정되지 않았기에 시퀀스에 position representations를 지정한다.

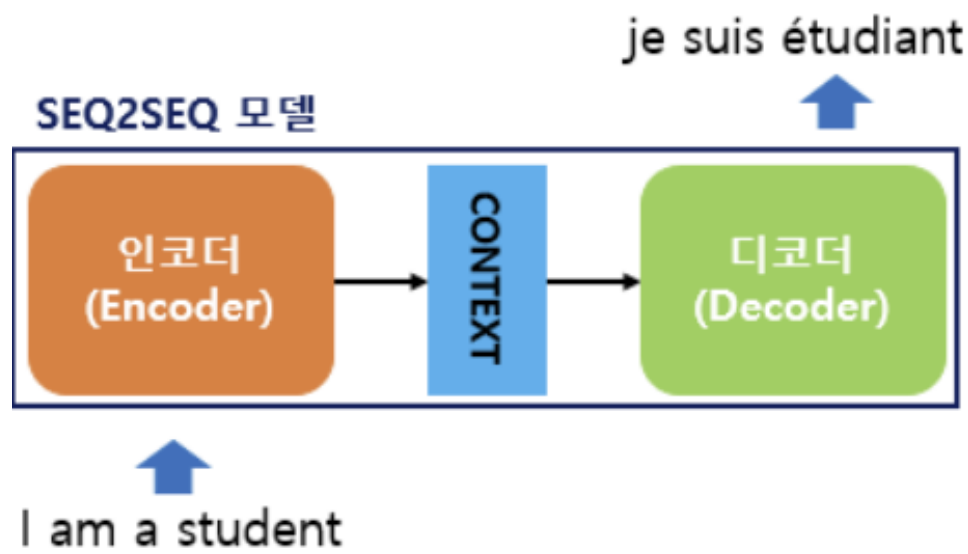
Nonlinearities: output of the self-attention block  
feed-forward network에 의해 구현

Masking: 미래의 값을 미리 알지 않고 병렬화하기 위해 사용  
미래의 값이 과거로 유출되지 않도록 하는 역할

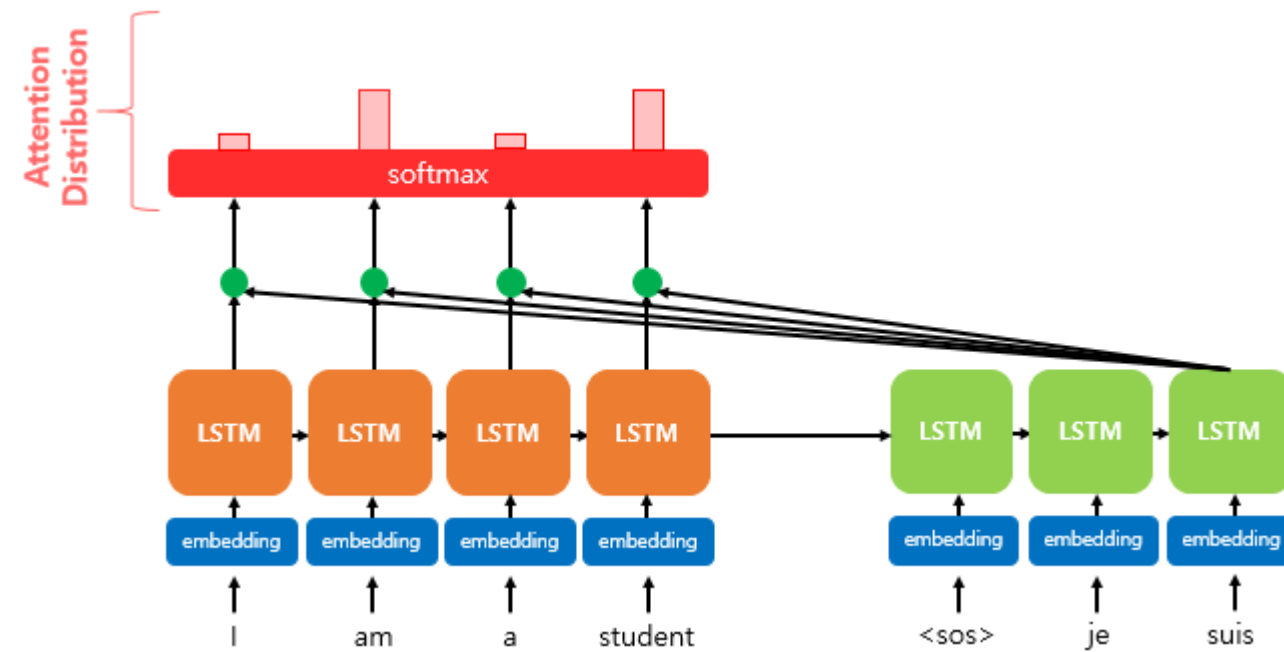
# 3. Transformer



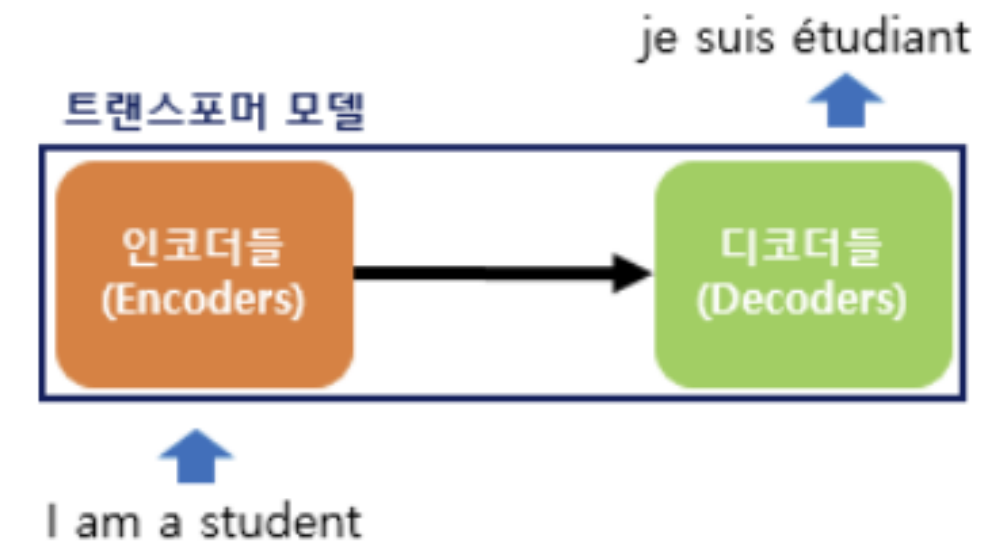
# 기존 seq2seq 모델의 한계와 transformer



기존의 바닐라 sequence to sequence 모델은 source sentence에 대해서 포착한 모든 정보를 encoder의 last hidden state에서 decoder로 전달 하는데, 이 과정에서 Q. information Bottleneck이 발생할 수 있다.



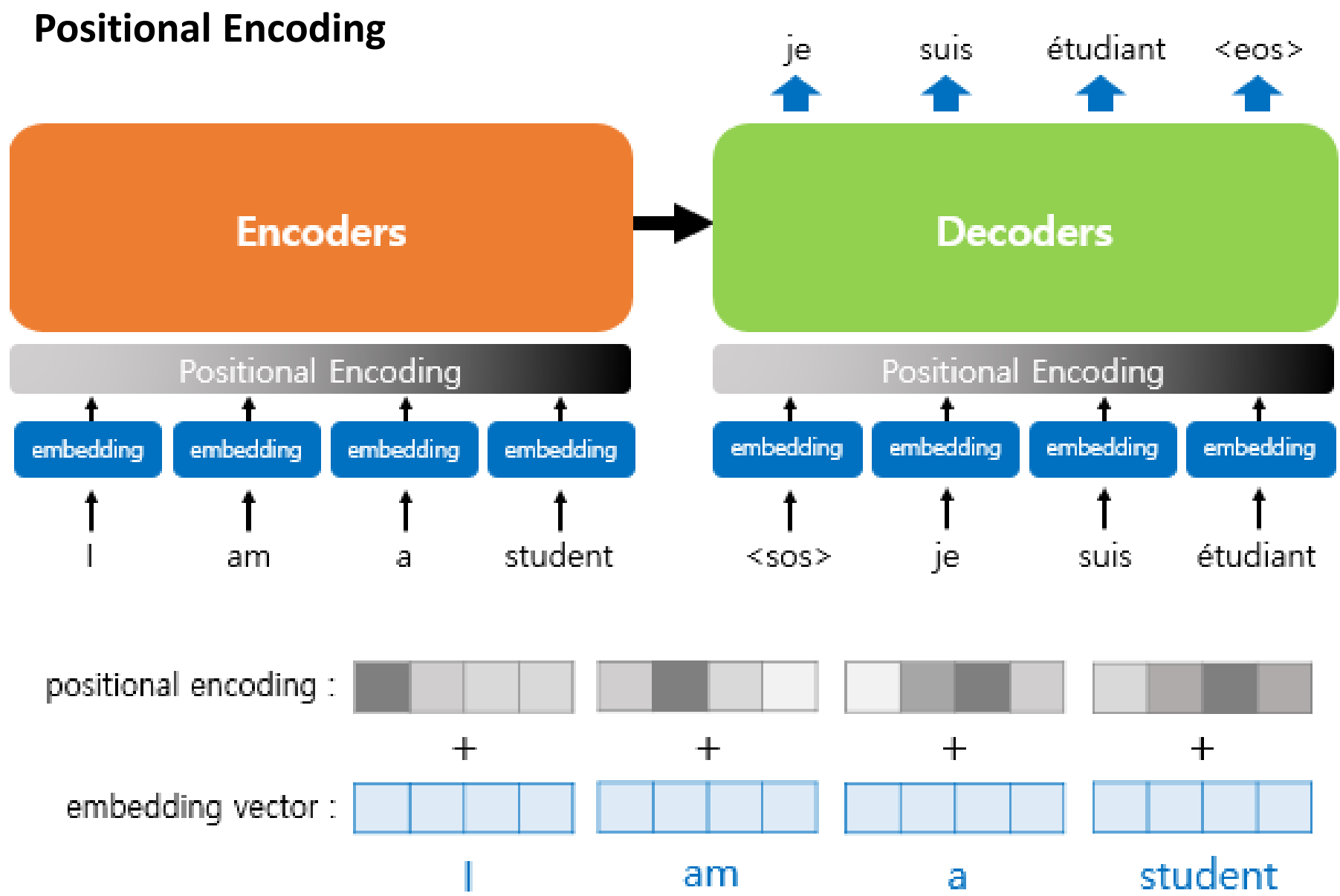
A. Attention을 사용하여 decoder의 매 스텝마다 source sentence의 특정 부분에 집중하기 위해 encoder와 직접적인 연결을 사용함으로써 문제점을 개선할 수 있다.



트랜스포머는 RNN을 사용하지 않지만 기존의 seq2seq의 인코더-디코더 구조를 유지하면서 building block만 트랜스포머로 변한 모습.

; 기존의 seq2seq가 인코더와 디코더에서 하나의 RNN이 t개의 시점을 가지는 구조였던 것과 달리, 트랜스포머에서는 인코더와 디코더의 단위가 N개로 구성되는 구조.

# Positional Encoding



; 단어 입력을 순차적으로 받는 방식이 아니므로 단어의 임베딩 벡터에 위치정보들을 더하여 단어의 위치정보를 얻는다.

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

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

;포지셔널 인코딩 값들을 얻기 위해  
sin,cos함수가 결합한 함수(sinusoidal)를 사용

Pos : 입력 문장에서의 임베딩 벡터의 위치

i : 임베딩 벡터 내의 차원의 인덱스  
; 인덱스가 짝수인 경우 (2i) 사인 함수의 값을,  
홀수인 경우 (2i+1) 코사인 함수의 값을 사용

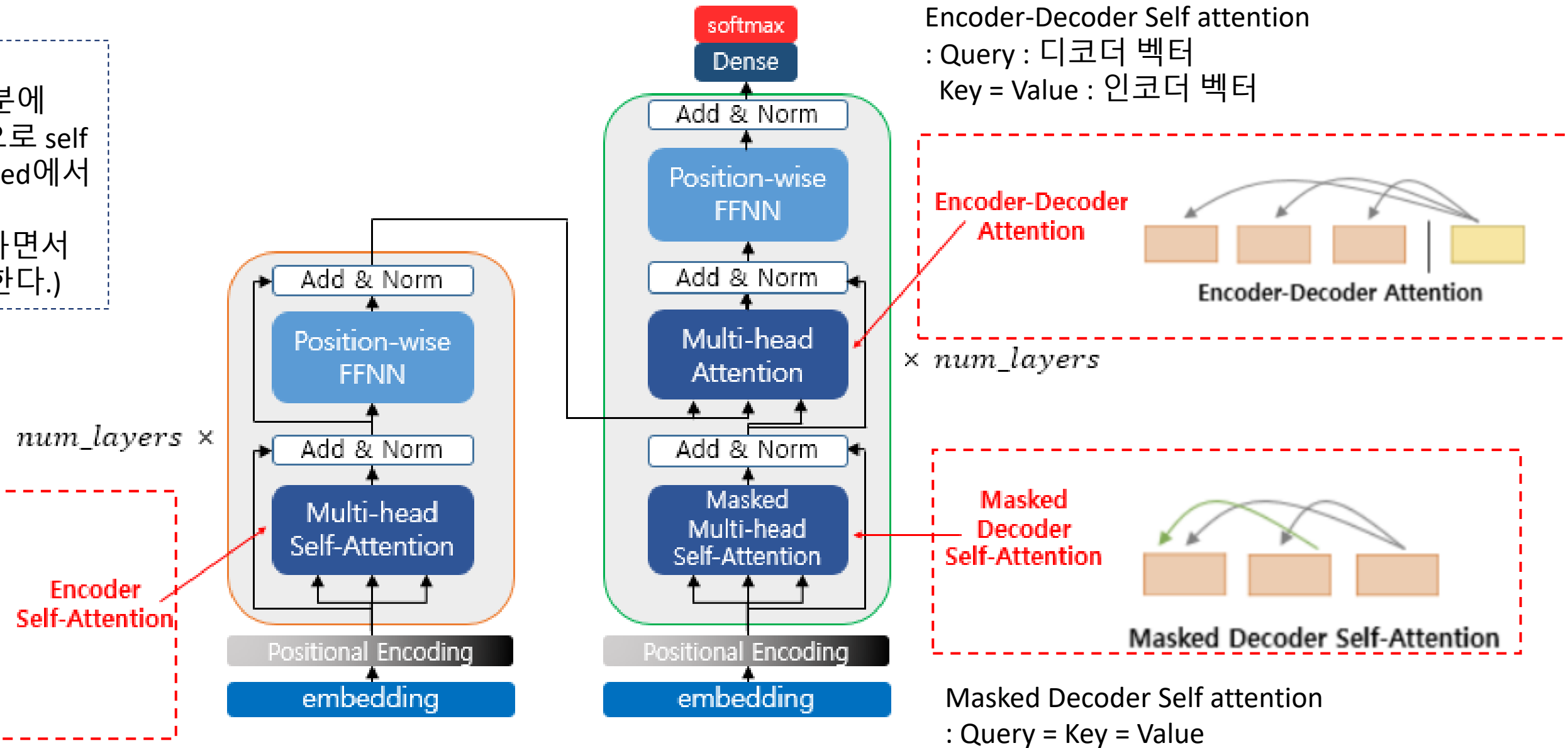
$d_{model}$ : 트랜스포머의 모든 층의 출력 차원을 의미하는  
트랜스포머의 하이퍼파라미터

# Encoder architecture

Multi-head attention  
; 트랜스포머가 한번에 여러 부분에 집중할 수 있도록 해주는 방법으로 self attention의 연산의 여러 개의 head에서 수행한다.  
(다른 시각으로 정보들을 수집하면서 다양한 관점을 가질 수 있도록 한다.)

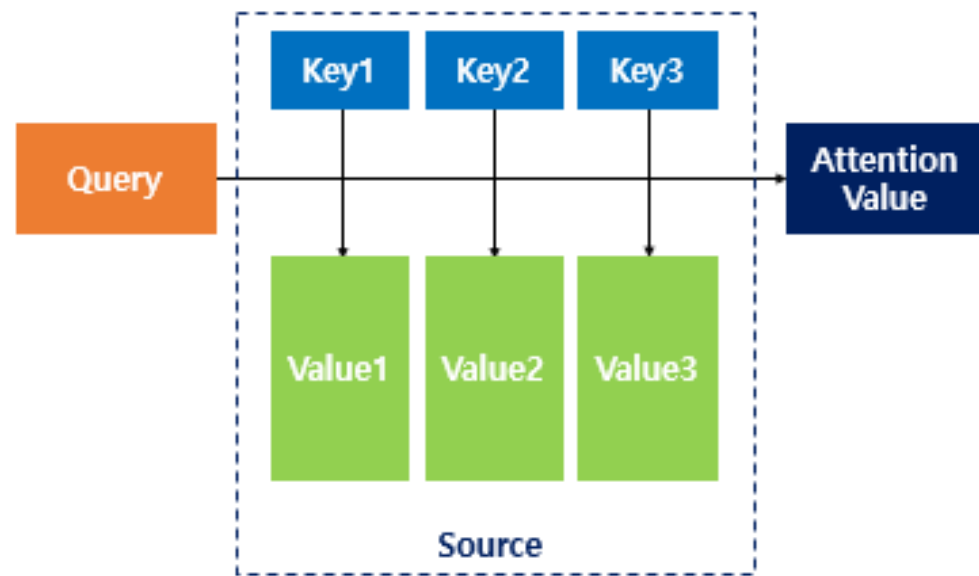


Self attention : Query, Key, Value가 동일한 경우  
Encoder Self attention  
:Query = Key = Value(벡터의 값이 같다는 것이 아니라 벡터의 출처가 같다는 의미)



인코더  
⊃ 피드포워드 네트워크, self attention 네트워크  
  
디코더  
⊃ 피드포워드 네트워크, self attention 네트워크, Encoder-Decoder attention 네트워크

# 인코더의 첫번째 서브층 -1)self-attention



Attention: 디코더의 매 스텝마다 인코더와 직접적인 연결을 사용하여 source sentence의 특정 부분에 집중하는 방법  
→ 한 문장안에서 attention을 계산하는 방법 = self attention

Query :  
t시점의 디코더 셀에서의 은닉상태

Key:  
모든 시점의 인코더 셀의 은닉상태

Value:  
모든 시점의 인코더 셀의 은닉 상태

① 주어진 query에 대해 모든 key와의 유사도를 구한다.

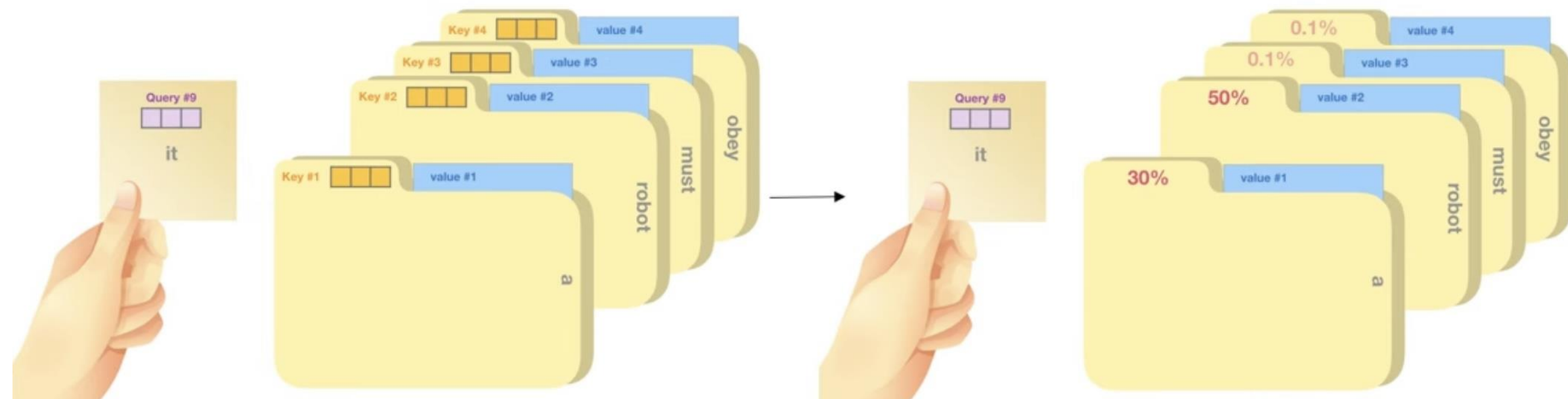
② 이를 가중치로 하여 key와 매핑되어있는 각각의 value에 반영한다.

③ 유사도가 반영된 value를 모두 가중합하여 리턴한다.

•**Query**: The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token *we're currently processing*.

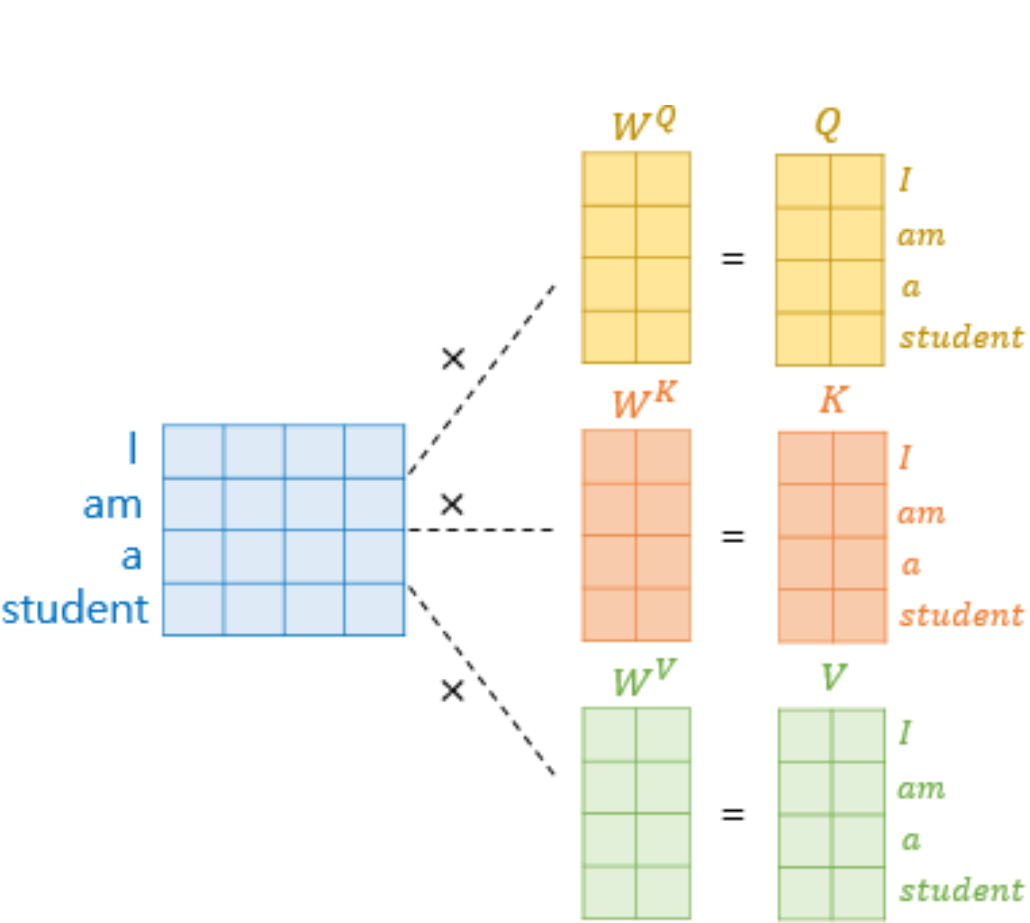
•**Key**: Key vectors are like labels for all the words in the segment. They're what we match against in our search for relevant words.

•**Value**: Value vectors are actual word representations, once we've scored how relevant each word is, these are the values we add up to represent the current word.

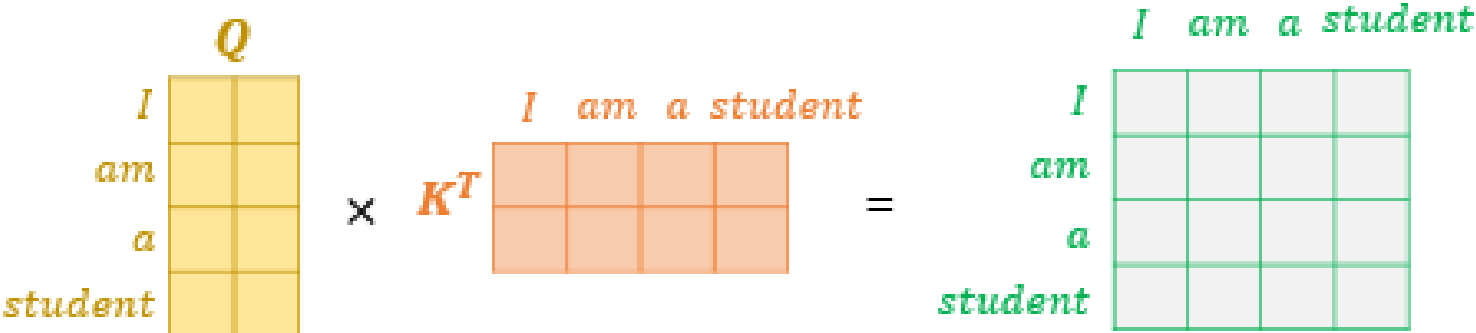


# 인코더의 첫번째 서브층 -1)self-attention

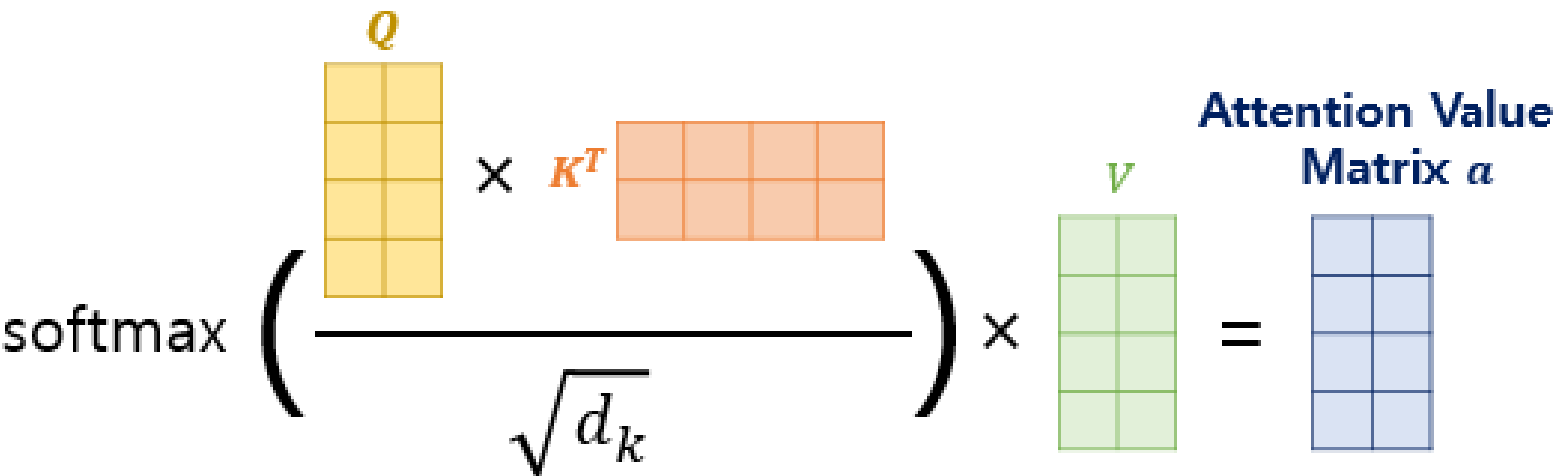
① 문장 행렬에 가중치 행렬을 곱하여 Q행렬, K행렬, V행렬을 구한다.



② 어텐션 스코어 행렬을 구하여 소프트맥스 함수를 사용한 후 V행렬을 곱하여 어텐션 값 행렬을 구한다.



Q행렬을 k행렬을 전치한 행렬과 곱해주면, Q벡터와 k벡터의 내적이 각 행렬의 원소가 되는 행렬을 구할 수 있다.

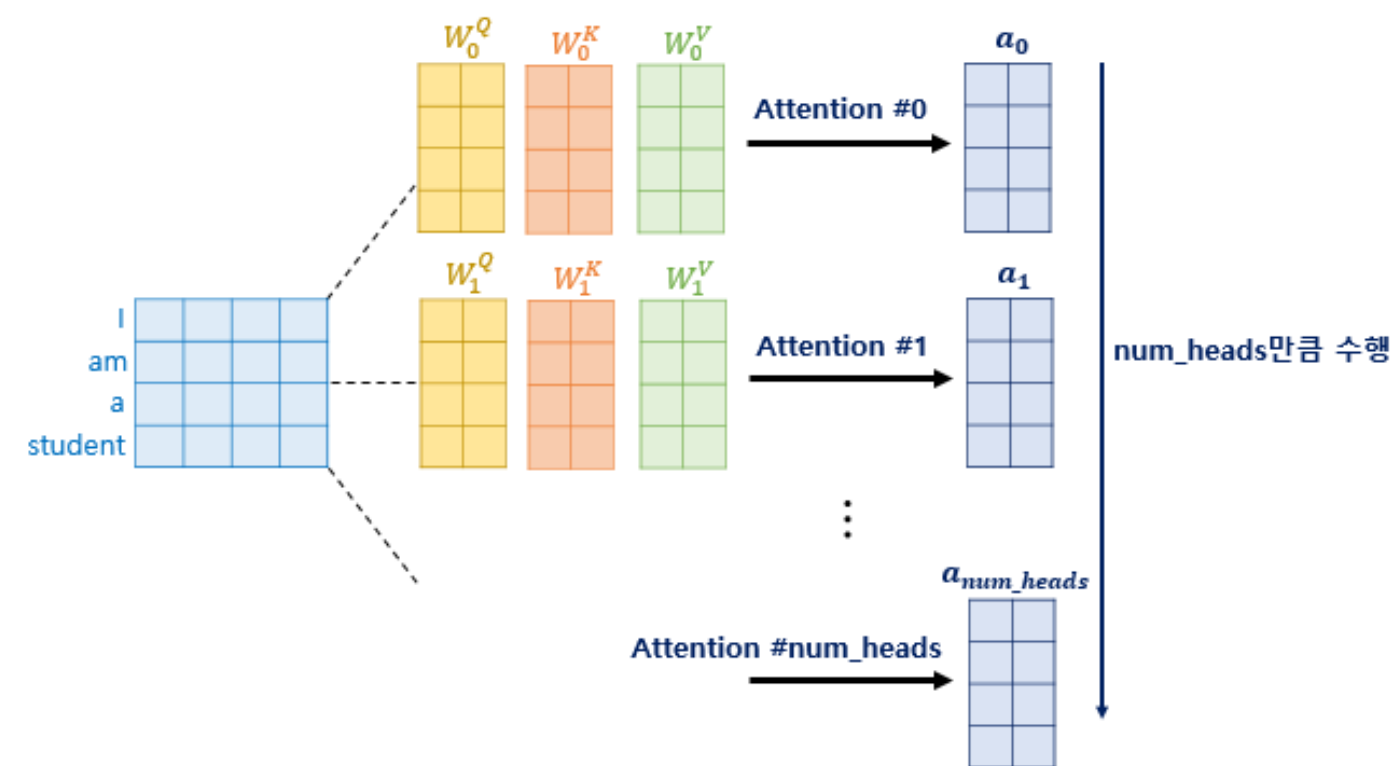


행렬의 값에 전체적으로  $\sqrt{d_k}$  를 나누어주어 각 행과 열이 어텐션 스코어를 가지는 행렬을 구한 후 이를 소프트맥스 함수에 넣은 후 v행렬을 곱하여 어텐션 값 행렬을 구한다.



# 인코더의 첫번째 서브층 -1)self-attention

①  $D_{model}/num\_heads$ 의 차원을 가지는 Q, K, V에 대해  $num\_heads$ 개의 병렬 어텐션을 수행한다.

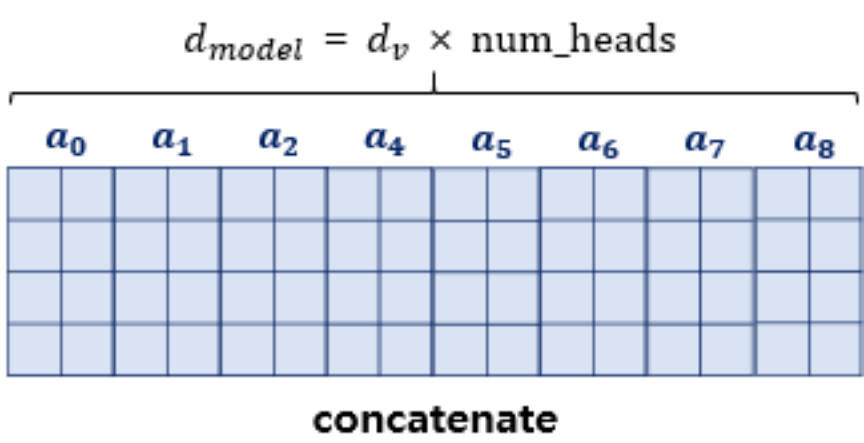


;이때, 각각의 어텐션 값 행렬을 어텐션 헤드라 하고,  $W^Q, W^K, W^V$ 의 값은 어텐션 헤드마다 다르다.

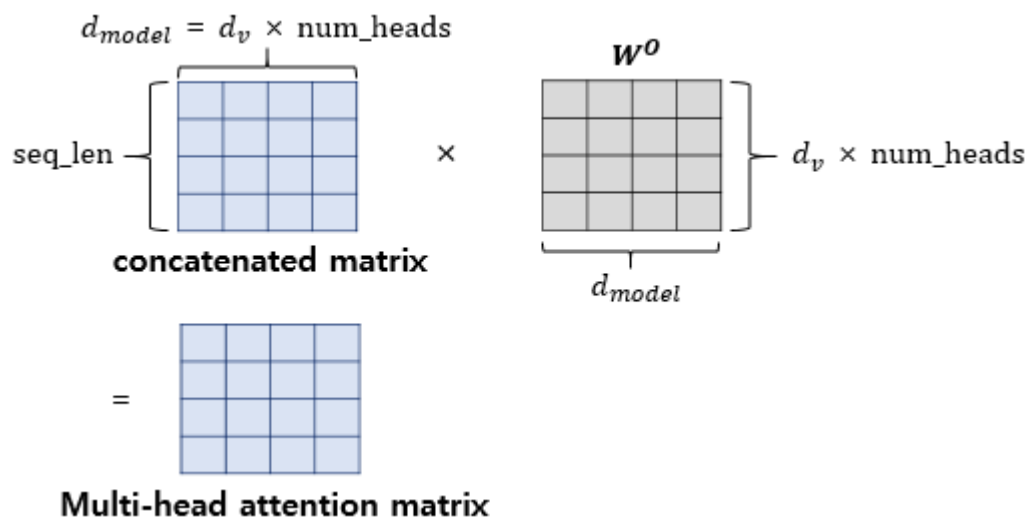
; 어텐션을 병렬로 수행하여 다양한 시각으로 정보를 수집할 수 있게된다.

② 병렬 어텐션을 수행한 후 스케일드 닷 프로덕트 어텐션을 수행한다.

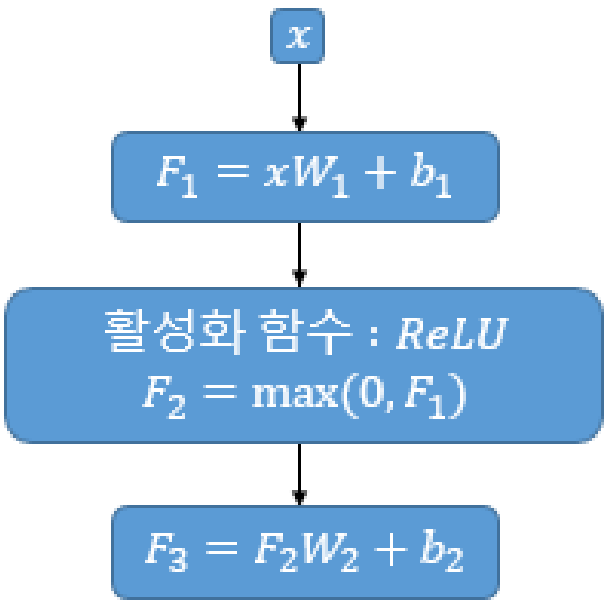
③ 나눠졌던 어텐션 헤드를 연결한다.



④ 어텐션 헤드를 모두 연결한 행렬에 가중치 행렬  $W^O$  을 곱하여 멀티-헤드 어텐션 행렬을 구한다.



# 인코더의 두번째 서브층 -2)position-wise FFNN



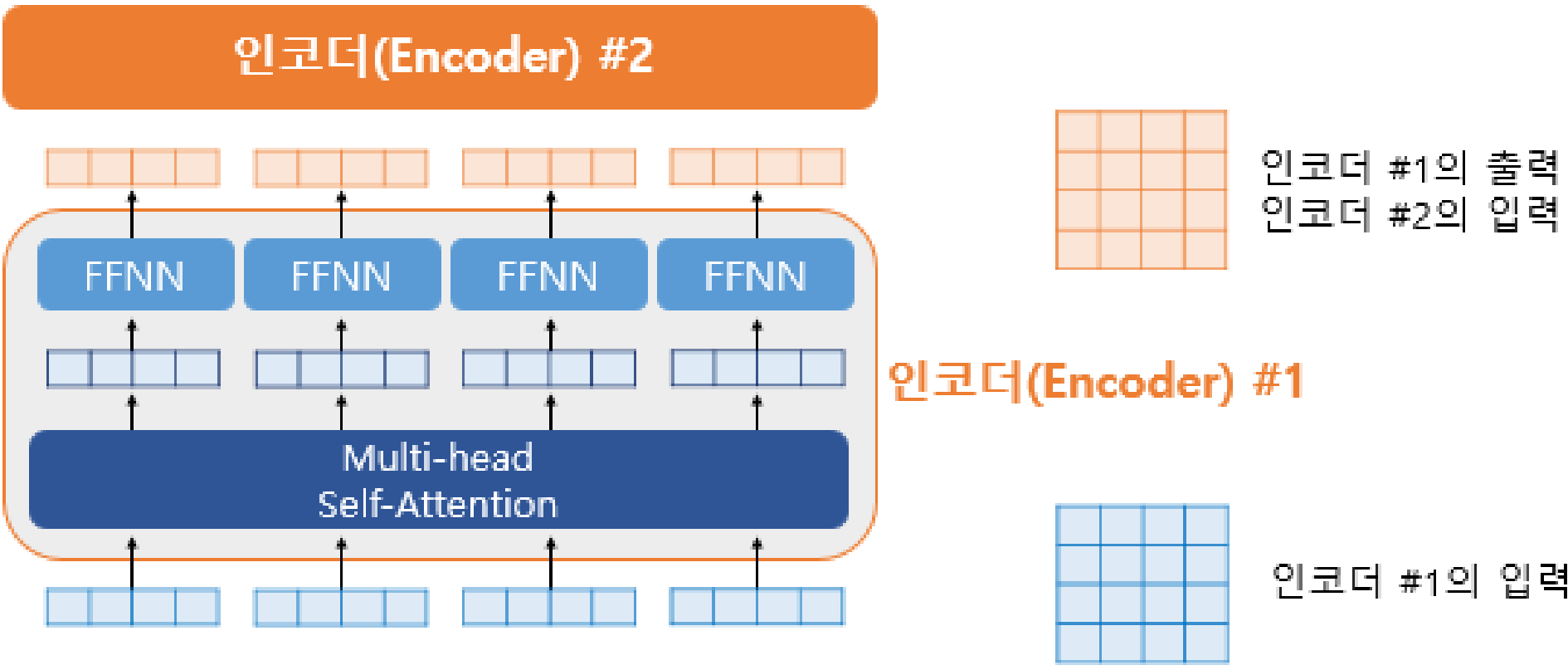
$$FFNN(x) = MAX(0, xW_1 + b_1)W_2 + b_2$$

x : 멀티 헤드 어텐션의 결과로 나온 행렬

W1, b1, W2, b2는 하나의 인코더 층 내에서는 다른 문장, 다른 단어마다 정확하게 동일하게 상용되지만 인코더 층마다 다른 값을 가진다.

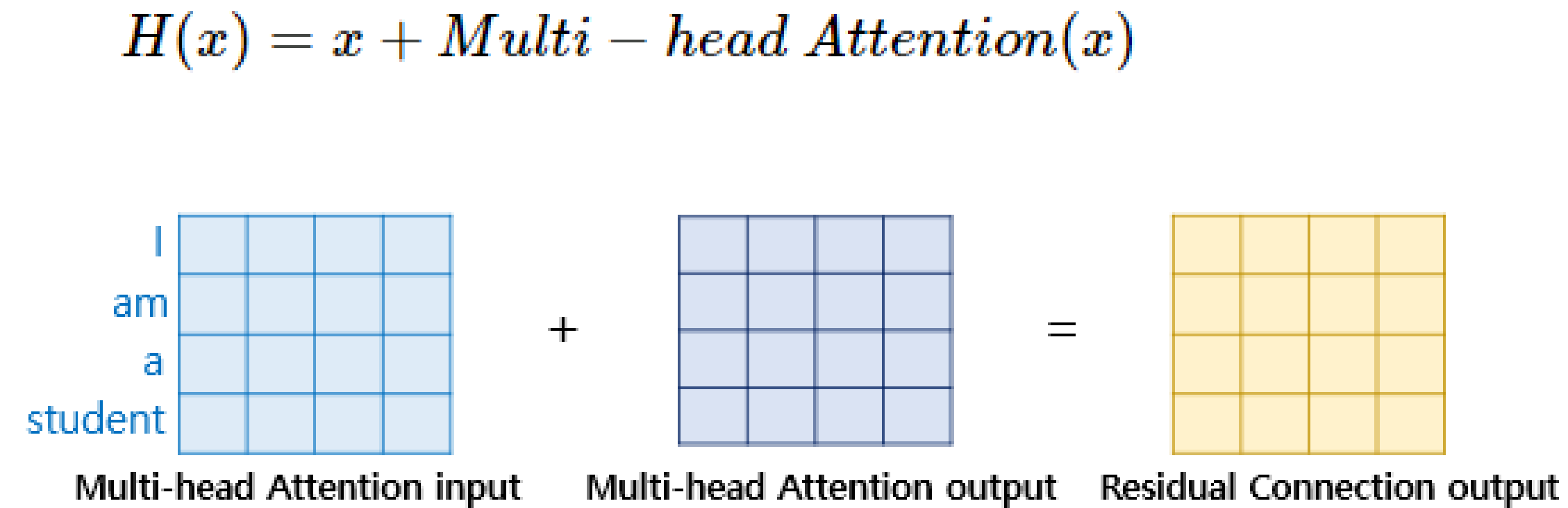
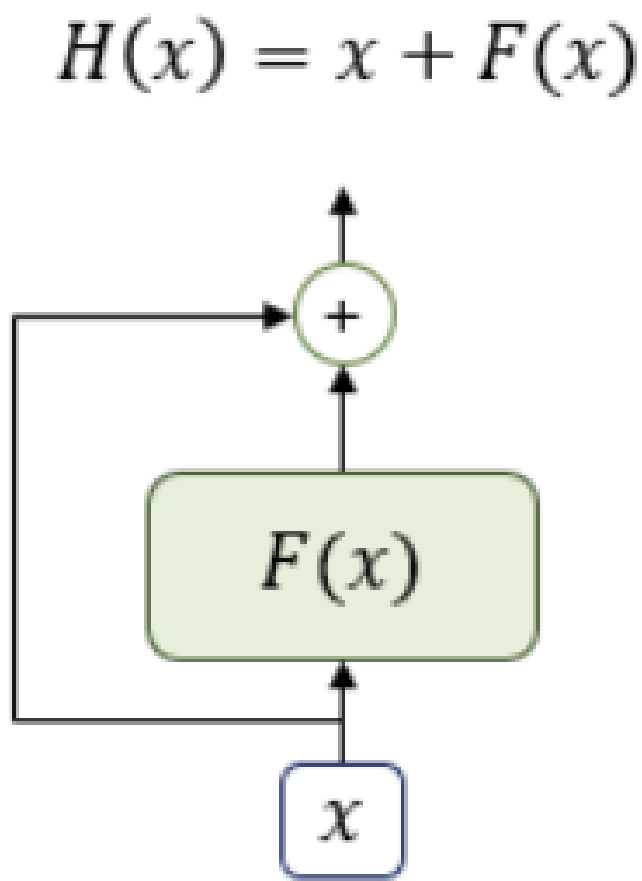
;오른쪽 그림에서 좌측은 인코더의 입력을 벡터 단위로 봤을 때, 각 벡터들이 멀티 헤드 어텐션 층 (인코더 내 첫번째 서브층)을 지나 FFNN을 통과하는 것을 보여준다. (실제로는 행렬로 연산됨)

;하나의 인코더 층을 지난 행렬은 다음 인코더 층으로 전달되고, 다음 층에서도 동일한 인코더 연산이 반복된다.



# 인코더에 사용하는 기법- 1) 잔차 연결(Residual connection)

## 1) 잔차연결



- : 서브층의 입력과 출력을 더하는 것
- ;이전 시점보다 얼마나 변화했는지를 학습한다고 해석할 수도 있다.
- ;기울기를 smoothing 해주는 효과가 있어 local min 빠지지 않도록 해준다.

;F(X)는 트랜스포머의 서브층의 해당하는데 , 서브층의 입력과 출력은 동일한 차원을 갖고 있으므로 덧셈연산이 가능하므로, 서브층이 멀티 헤드 언텐션이라 할 때 H(X)는  $x + \text{Multi-head attention}(x)$ 로 표현할 수 있다.

## 인코더에 사용하는 기법- 2) 층 정규화(layer normalization)

### 2) 층 정규화(layer normalization)

- Layer 내에서 하나의 input sample  $x$ 에 대해서 모든 feature에 대한 평균과 분산을 구해 normalization

Let  $x \in \mathbb{R}^d$  be an individual (word) vector in the model.

Let  $\mu = \sum_{j=1}^d x_j$ ; this is the mean;  $\mu \in \mathbb{R}$ .

Let  $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^d (x_j - \mu)^2}$ ; this is the standard deviation;  $\sigma \in \mathbb{R}$ .


Let  $\gamma \in \mathbb{R}^d$  and  $\beta \in \mathbb{R}^d$  be learned “gain” and “bias” parameters. (Can omit!)

$$\text{output} = \frac{x - \mu}{\sqrt{\sigma} + \epsilon} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

Layer Normalization

	batch		
			
	1	3	6
	2	2	2
	0	1	5
	4	6	1
	5	2	3
	1	0	1
mean	2	3	3
std	2	2	2

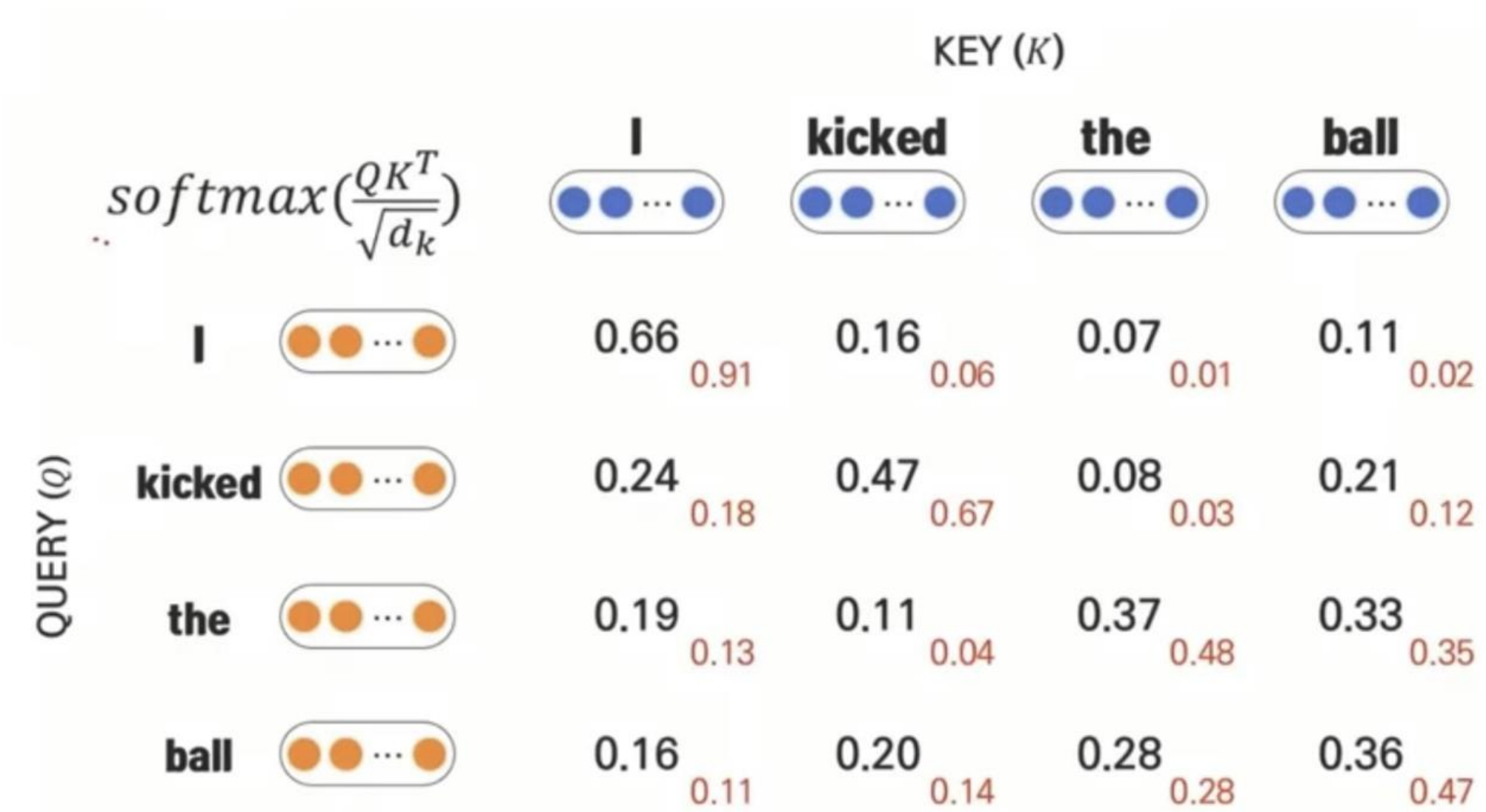
Same for all feature dimensions

# 인코더에 사용하는 기법-3) Scaled dot product

## 3) Scaled dot product

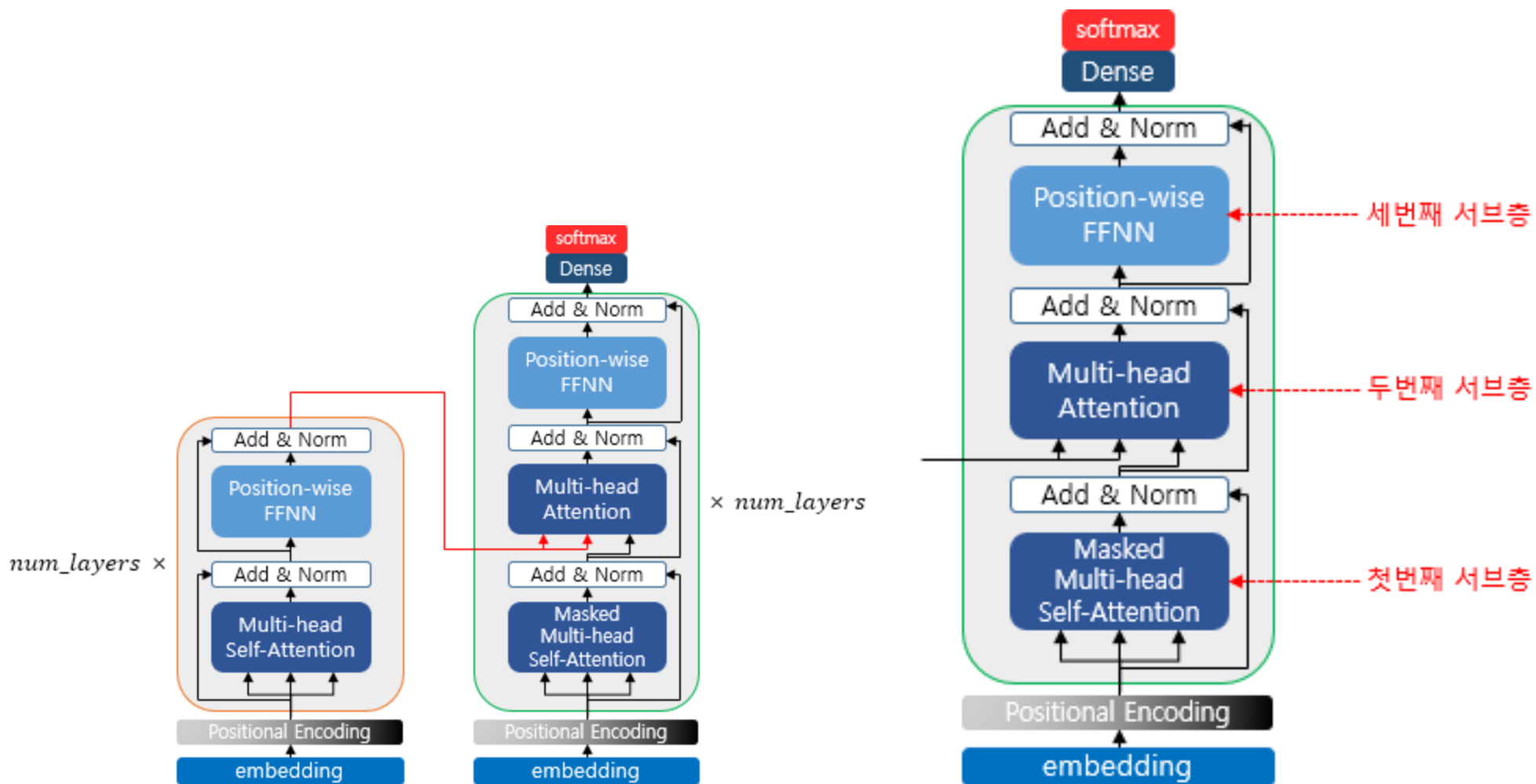
- Attention score 좀 더 다양한 vector들에게 분배

$$\text{output}_\ell = \text{softmax}(XQ_\ell K_\ell^\top X^\top) * XV_\ell \longrightarrow \text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$



# 디코더의 첫번째 서브층 -1)self-attention, look-ahead mask

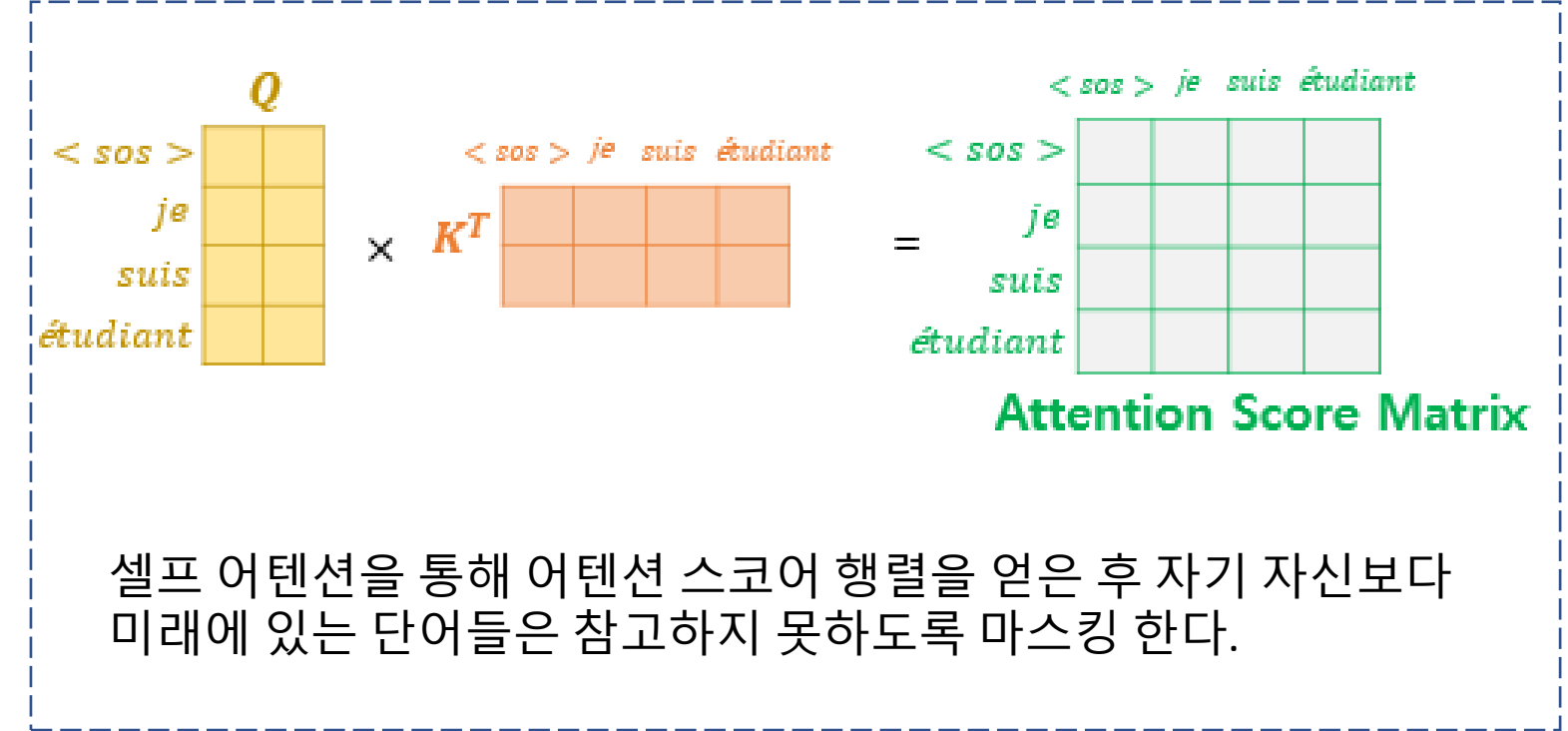
## 디코더의 첫번째 서브층 -1) self-attention, look-ahead mask



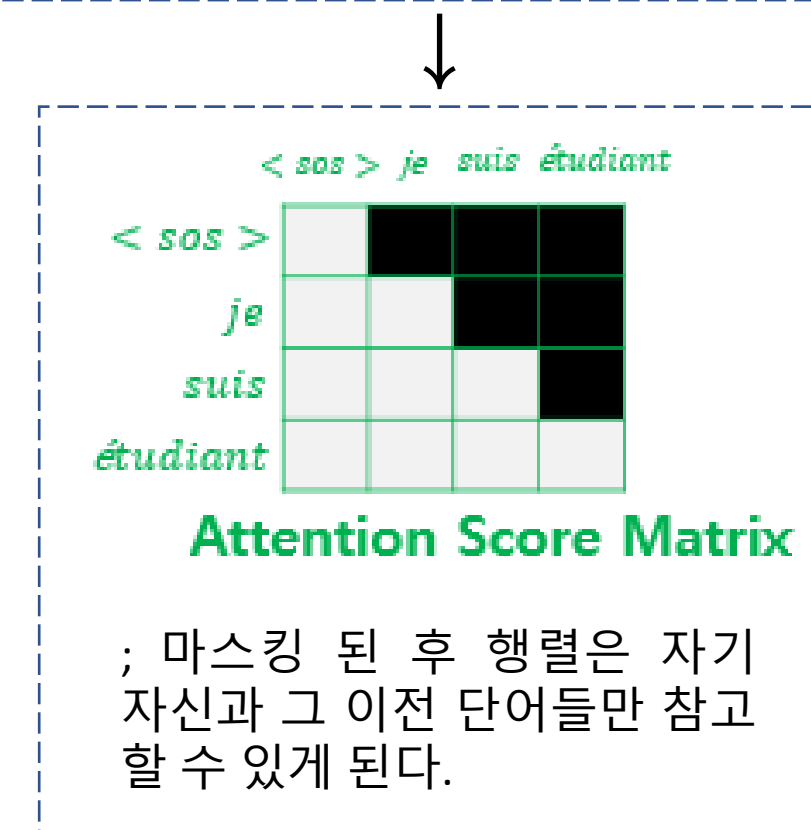
; 인코더는 총 num-analysis만큼의 층 연산을 순차적으로 한 후 마지막 층의 인코더의 출력을 디코더에게 전달한다.

; Multi-head self-attention층(디코더의 첫번째 서브층)은 인코더의 첫번째 서브층과 동일한 연산을 수행하며, 어텐션 스코어 행렬에서 마스킹을 적용한다.

; look-ahead mask 는 디코더의 첫번째 서브층에서 이루어진다.



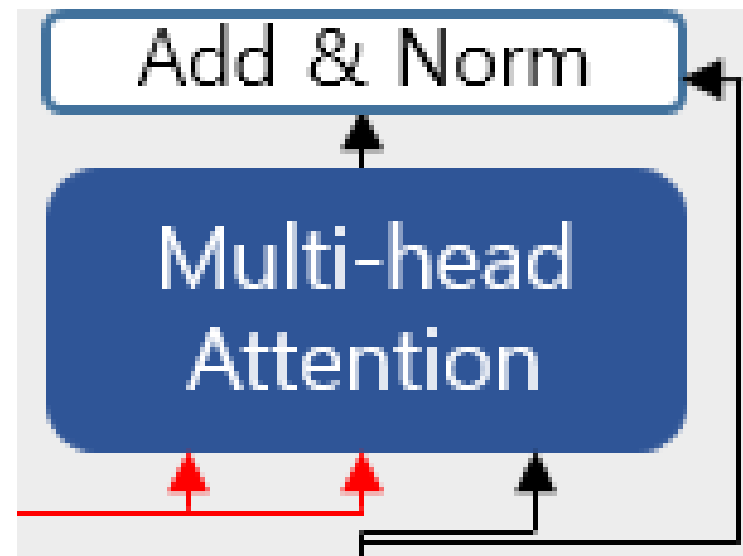
셀프 어텐션을 통해 어텐션 스코어 행렬을 얻은 후 자기 자신보다 미래에 있는 단어들은 참고하지 못하도록 마스킹 한다.



; 마스킹 된 후 행렬은 자기 자신과 그 이전 단어들만 참고할 수 있게 된다.

## 디코더의 두번째 서브층 -2)encoder-decoder attention

디코더의 두번째 서브층 -2)encoder-decoder attention



$$\begin{matrix} & Q \\ & \begin{matrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{matrix} \\ \begin{matrix} < sos > \\ je \\ suis \\ \acute{e}tudiant \end{matrix} \end{matrix} \times K^T \begin{matrix} I \text{ am a student} \\ \begin{matrix} & & & & \\ & & & & \\ & & & & \\ & & & & \end{matrix} \end{matrix} = \begin{matrix} & I \text{ am a student} \\ \begin{matrix} < sos > \\ je \\ suis \\ \acute{e}tudiant \end{matrix} & \begin{matrix} \begin{matrix} & & & & \\ & & & & \\ & & & & \\ & & & & \end{matrix} \end{matrix} \end{matrix}$$

**Attention Score Matrix**

; 디코더의 두번째 서브층은 인코더로부터 2개의 화살표를 받는데, 이는 각각 Key와 Value를 의미한다.

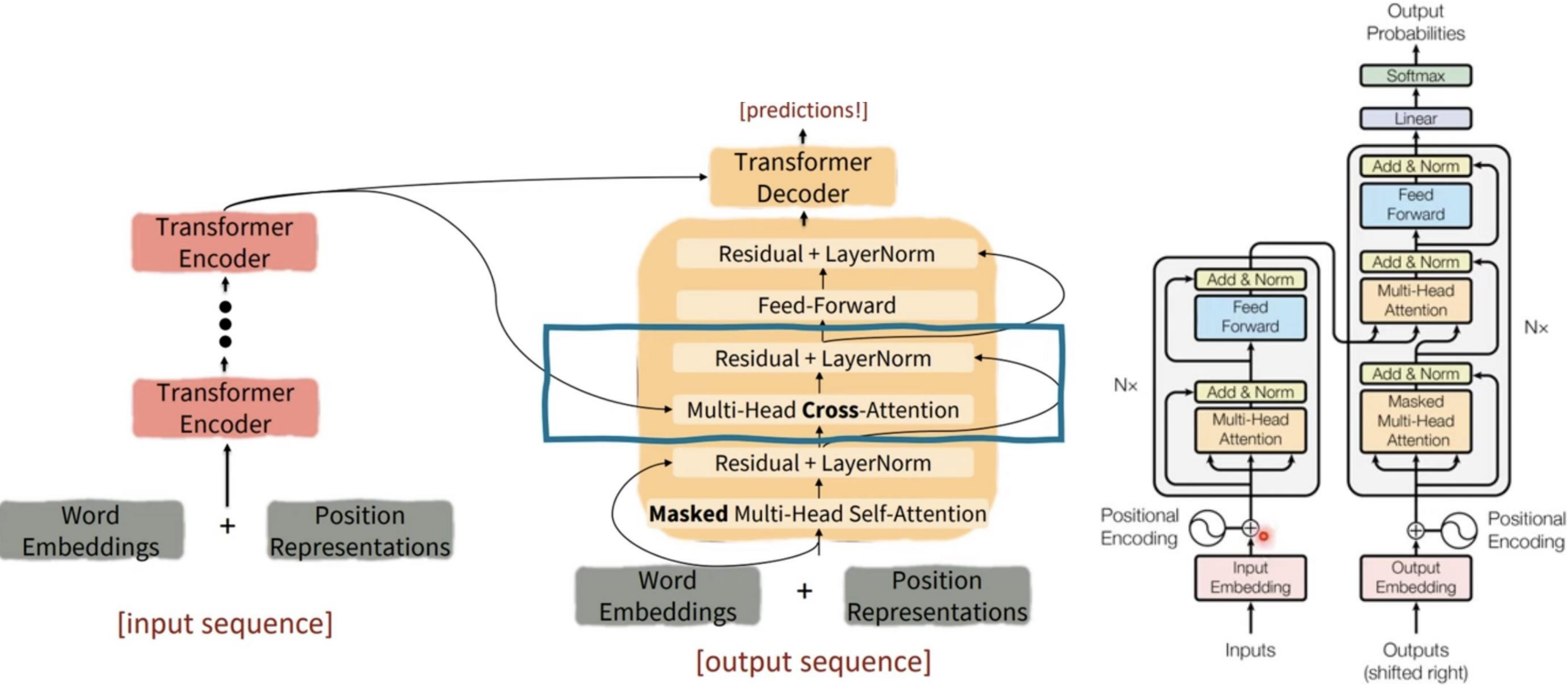
; Key와 Value는 인코더의 마지막 층에서 온 행렬로부터, Query는 디코더의 첫번째 서브층의 결과 행렬로부터 얻는다.

; Query가 디코더 행렬, Key가 인코더 행렬일 때 Q행렬에 k가 전치된 행렬을 곱하여 어텐션 스코어 행렬을 구한다.

; 그 외 멀티 헤드 어텐션을 수행하는 과정은 다른 어텐션과 동일하다.



# Encoder & Decoder detail





# THANK YOU

출처: [\[DSBA\] CS224n 2021 Study | #09 Self- Attention and Transformers | - YouTube](#)  
1) 트랜스포머(Transformer) - 딥 러닝을 이용한 자연어 처리 입문 ([wikidocs.net](#))

