

# 17. Transformer

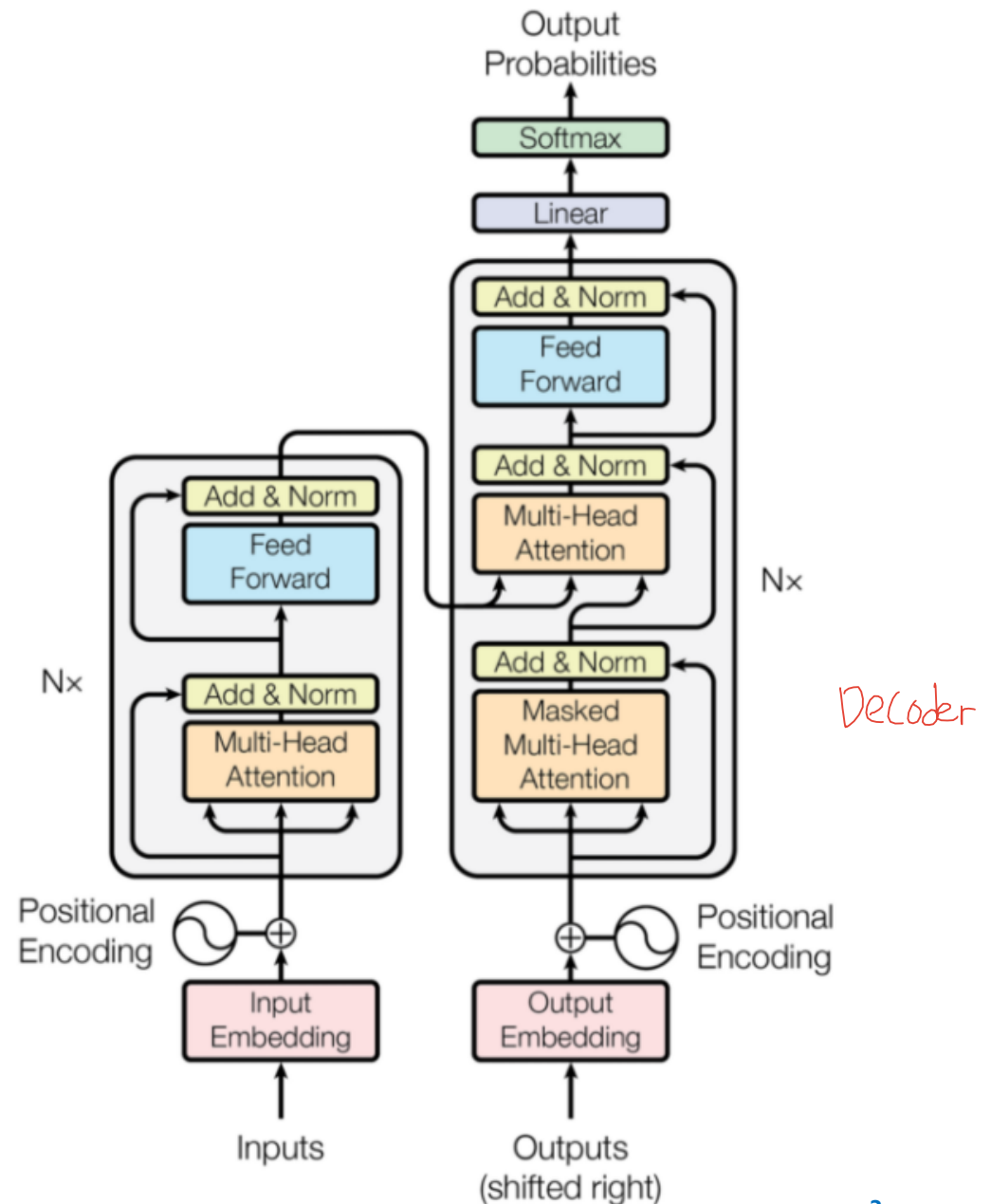
# Transformer

- RNN 구조를 사용하지 않고 문장 전체를 동시에 처리하는 방식
- Attention, residual connection 등 핵심적인 기술들을 사용함
- Encoder-decoder 구조로 되어 있음
- Encoder에서는 자체 문장 분석을 통해 self-attention을 계산: Decoder로부터의 정보 유입이 없음

# Transformer

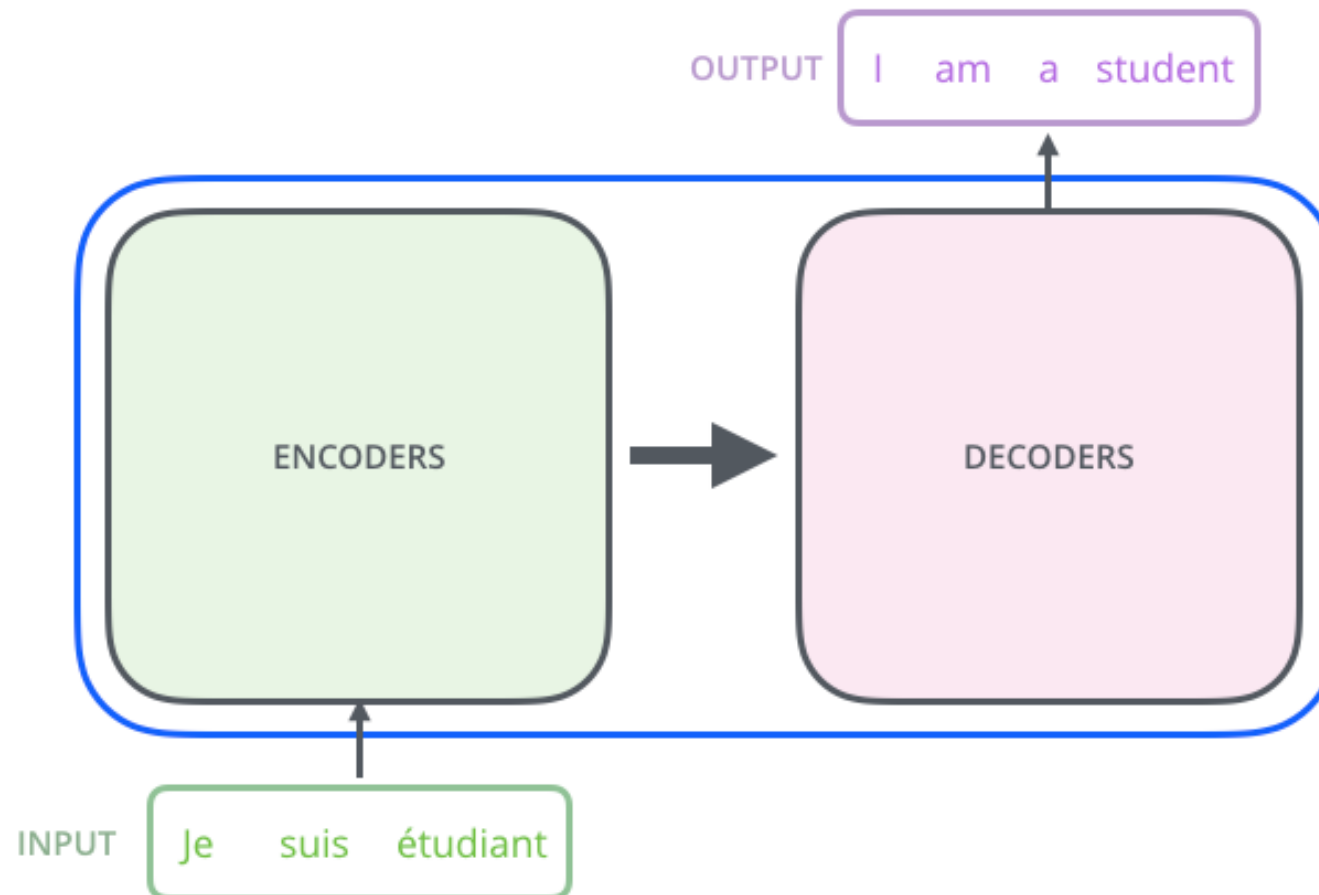
- 2017년에 발표된 Google 연구자들의 논문
- RNN 구조에 비해 기계번역 등에서 성능이 우수함

A. Vaswani *et al*, [Attention is all you need](#), *Proc. NIPS*, 2017.



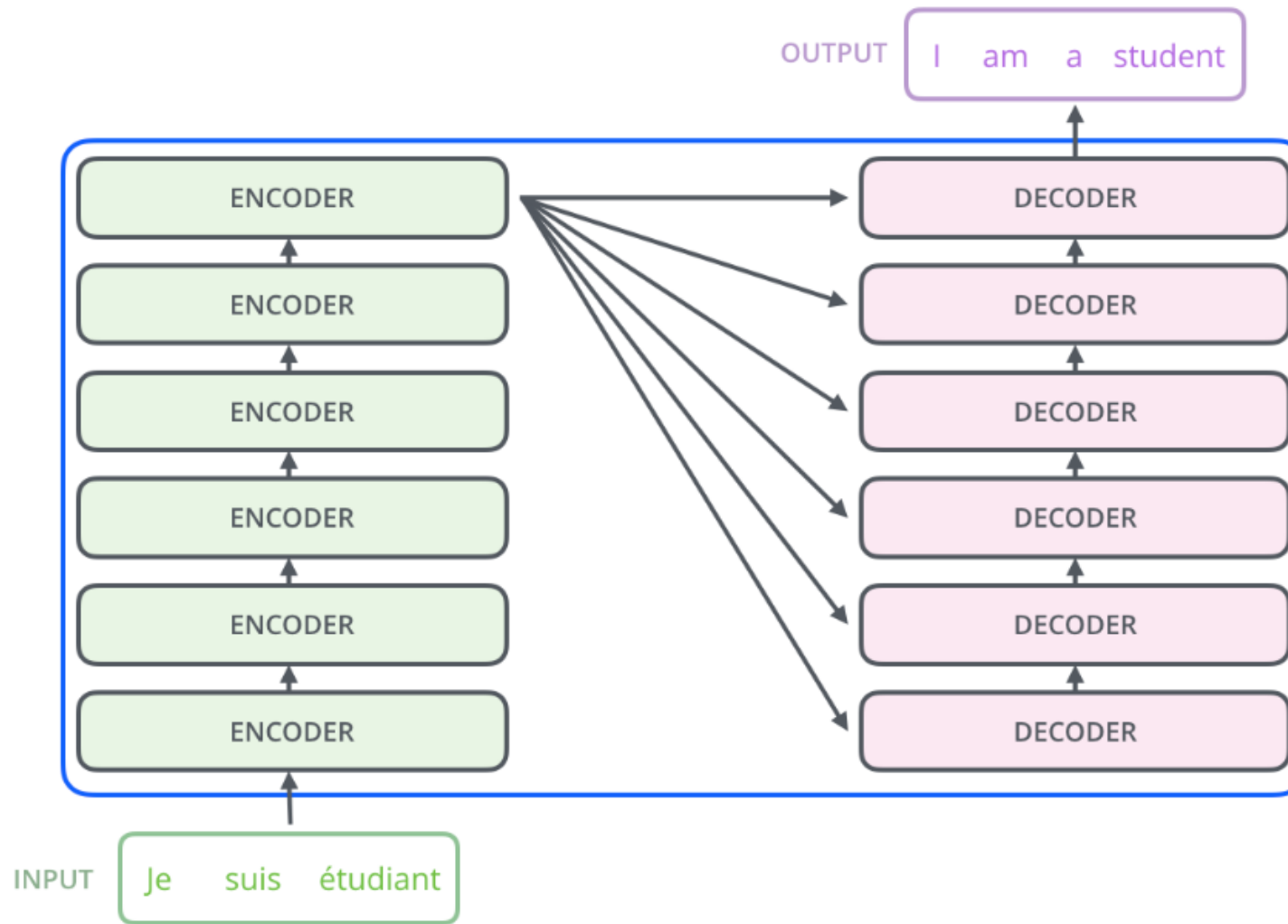
# Encoder-decoder 구조

- Sequence-to-sequence 구조와 같이 encoder-decoder로 구성



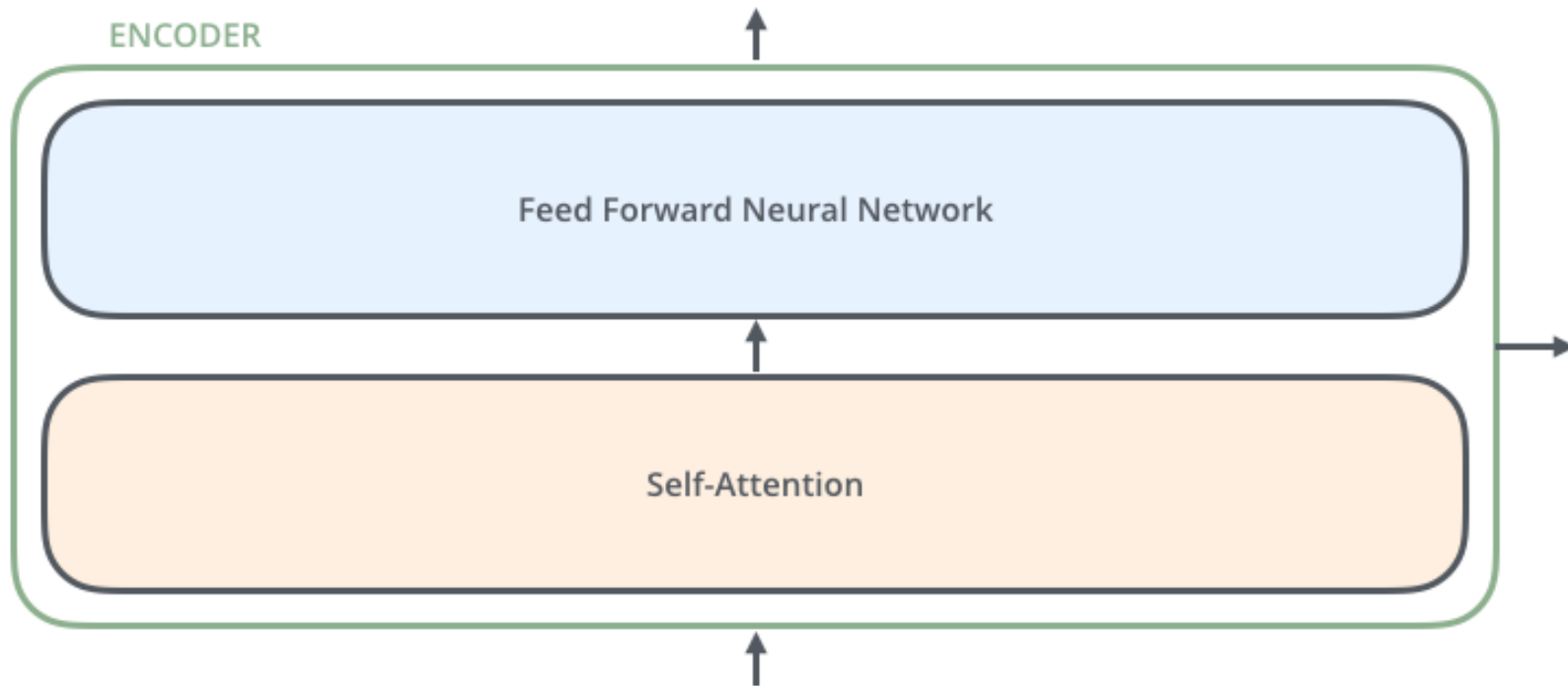
# 레이어 구조

- 인코더-디코더는 각각 6층의 동일한 레이어로 구성



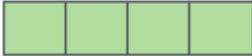
# 레이어 구성

- 각 레이어는 self-attention과 Feed forward 신경망으로 구성

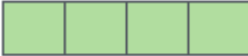


# Input Embedding

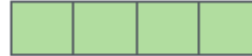
- 입력 단어는 512 자리의 embedding 벡터로 변환됨
- RNN과 달리 여기서는 문장 전체가 한꺼번에 입력됨

$x_1$  

Je

$x_2$  

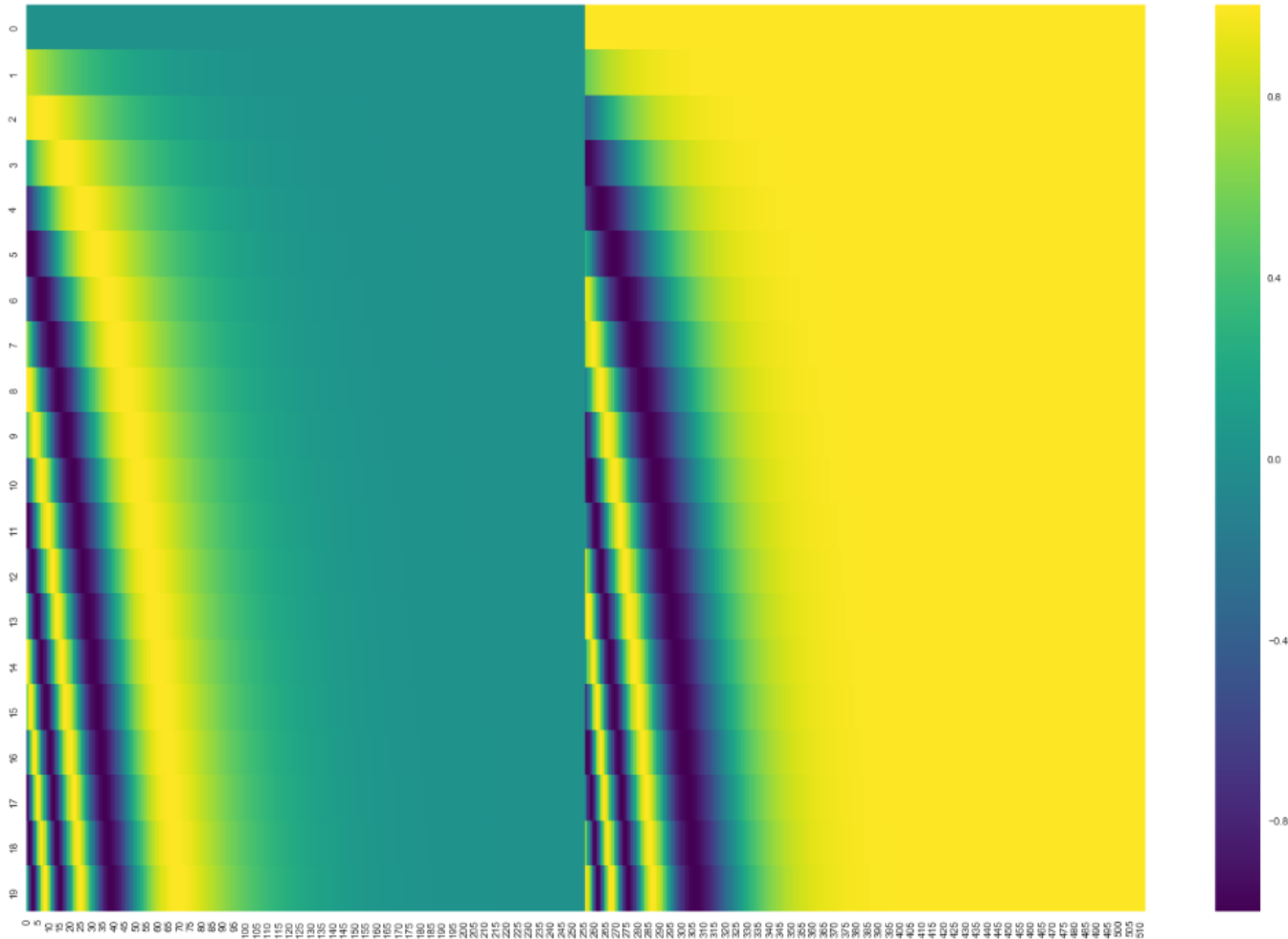
suis

$x_3$  

étudiant

# Positional Encoding

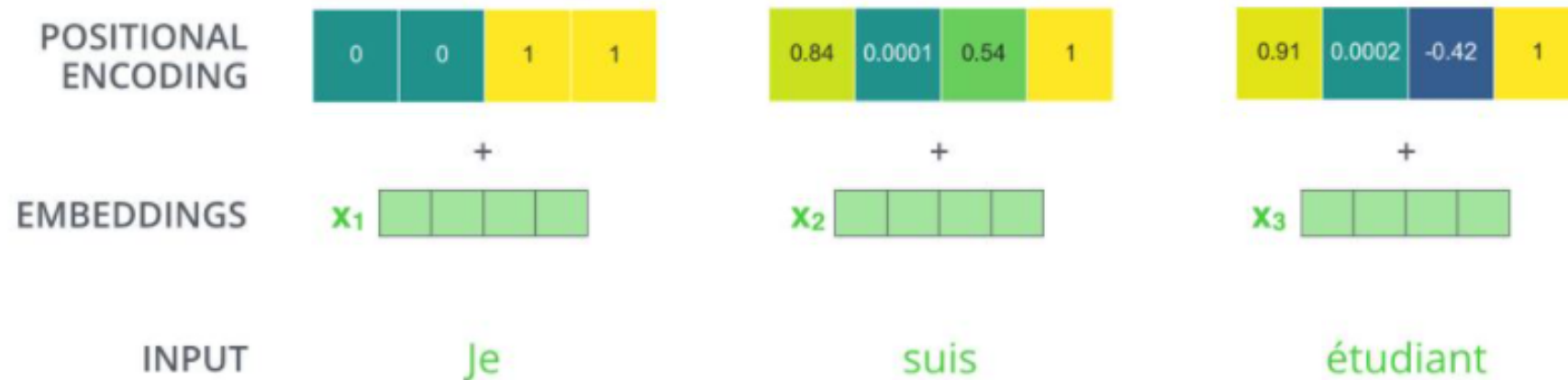
- RNN과 달리 여기서는 문장 전체가 한꺼번에 입력되므로 단어들의 위치 정보를 Positional Encoding으로 추가함
- 위치에 따라 값이 다른 position vector를 이용





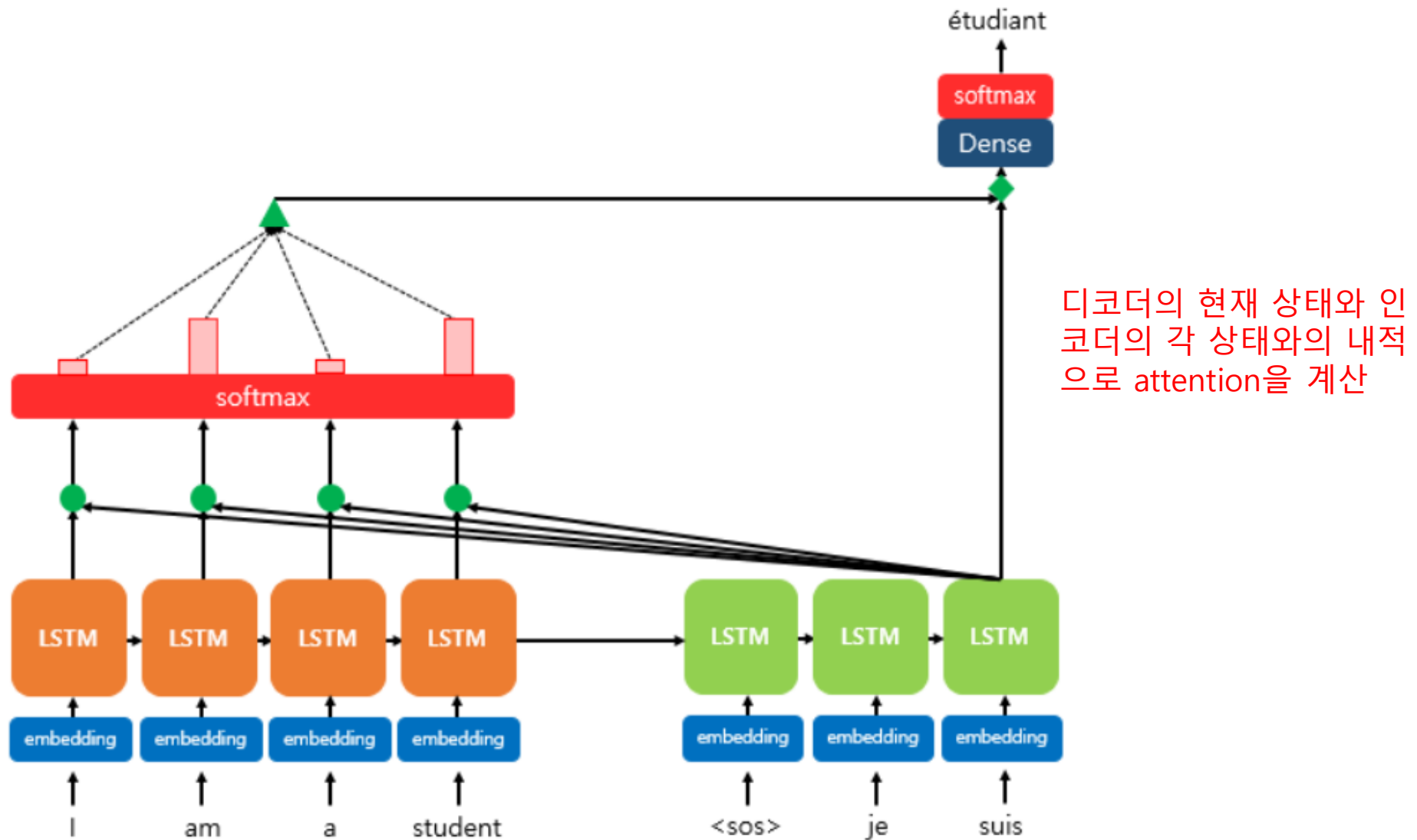
# Position vector

- Position vector는 embedding vector와 같은 크기로 구성
- 아래 그림은 벡터의 크기가 4일 때의 사례임



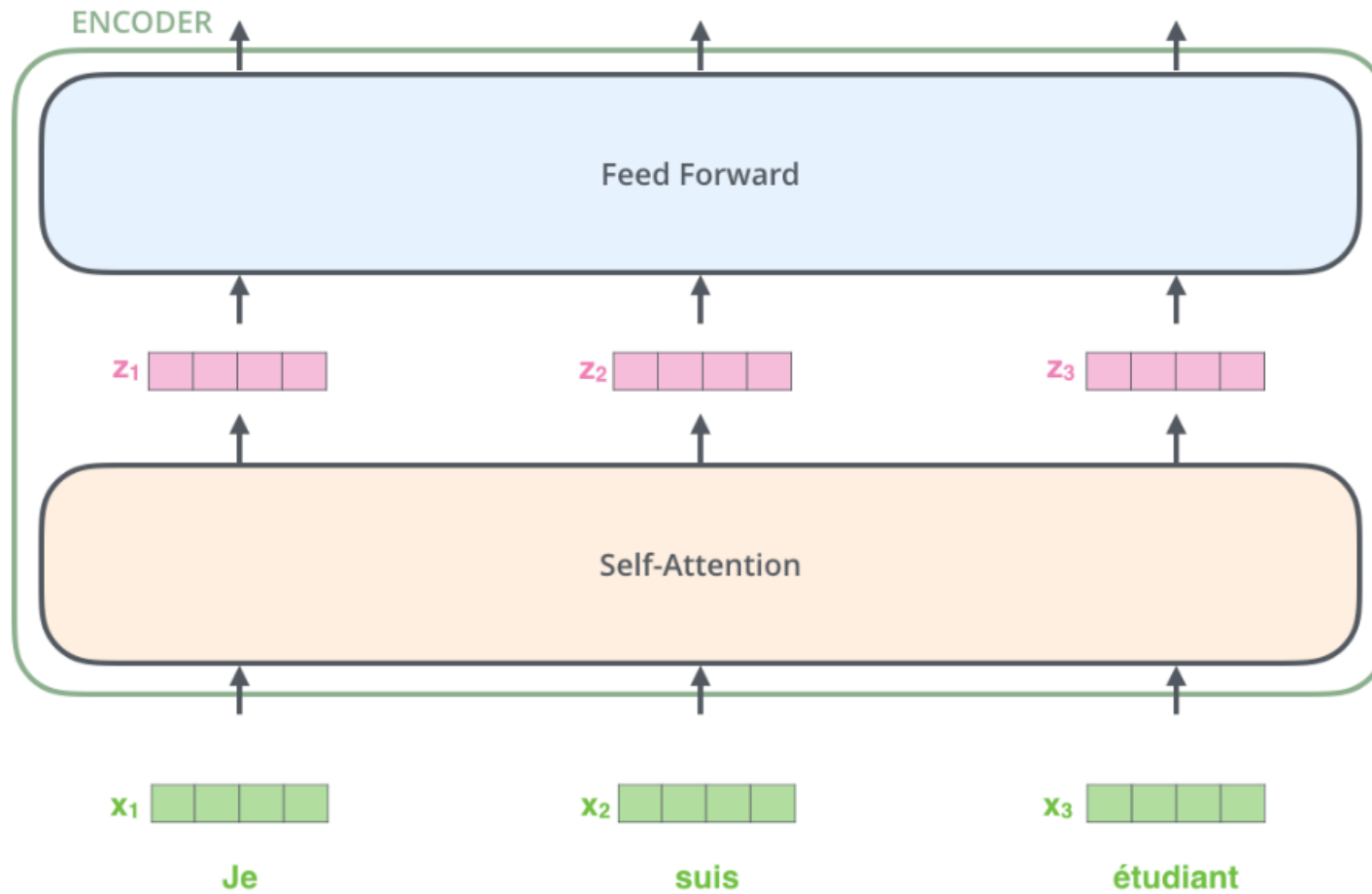
# RNN에서의 Attention 계산(복습)

- Decoder 상태를 encoder로 보내서 attention을 계산



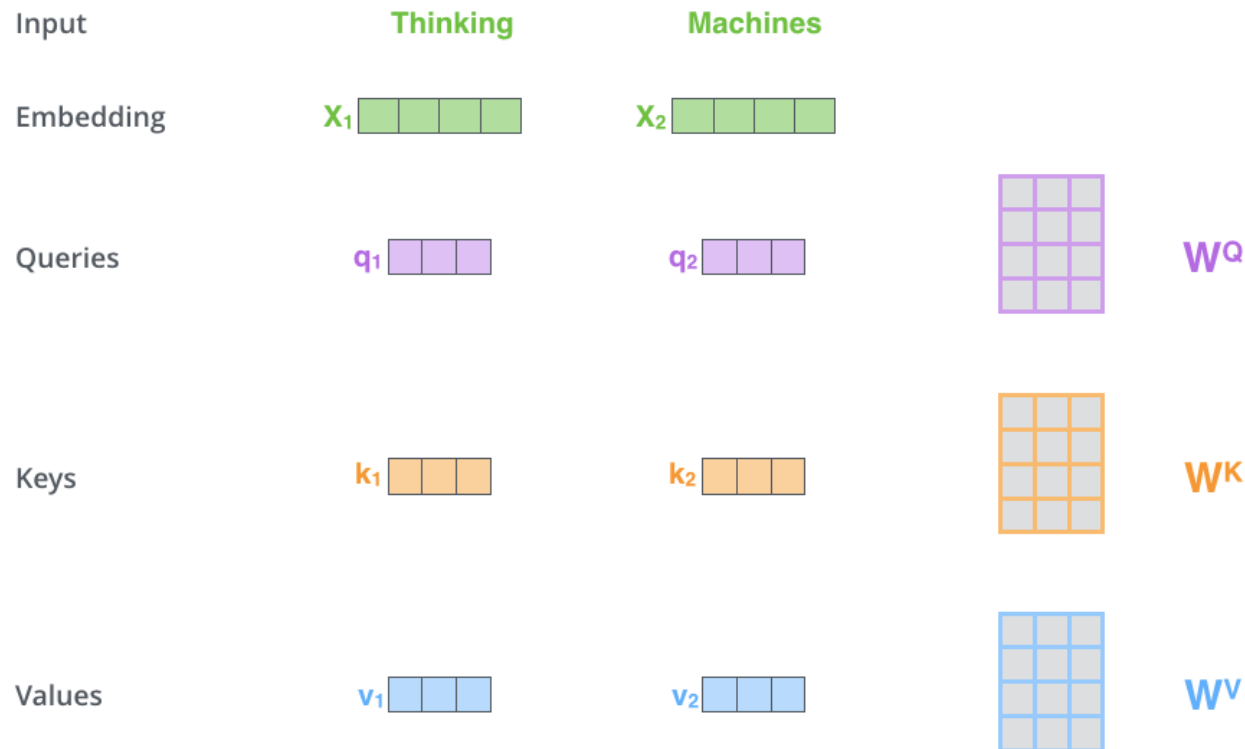
# Self-Attention

- 문장내 단어간의 attention이 먼저 계산됨.
- RNN attention과 달리 decoder에서 들어오는 정보는 없음



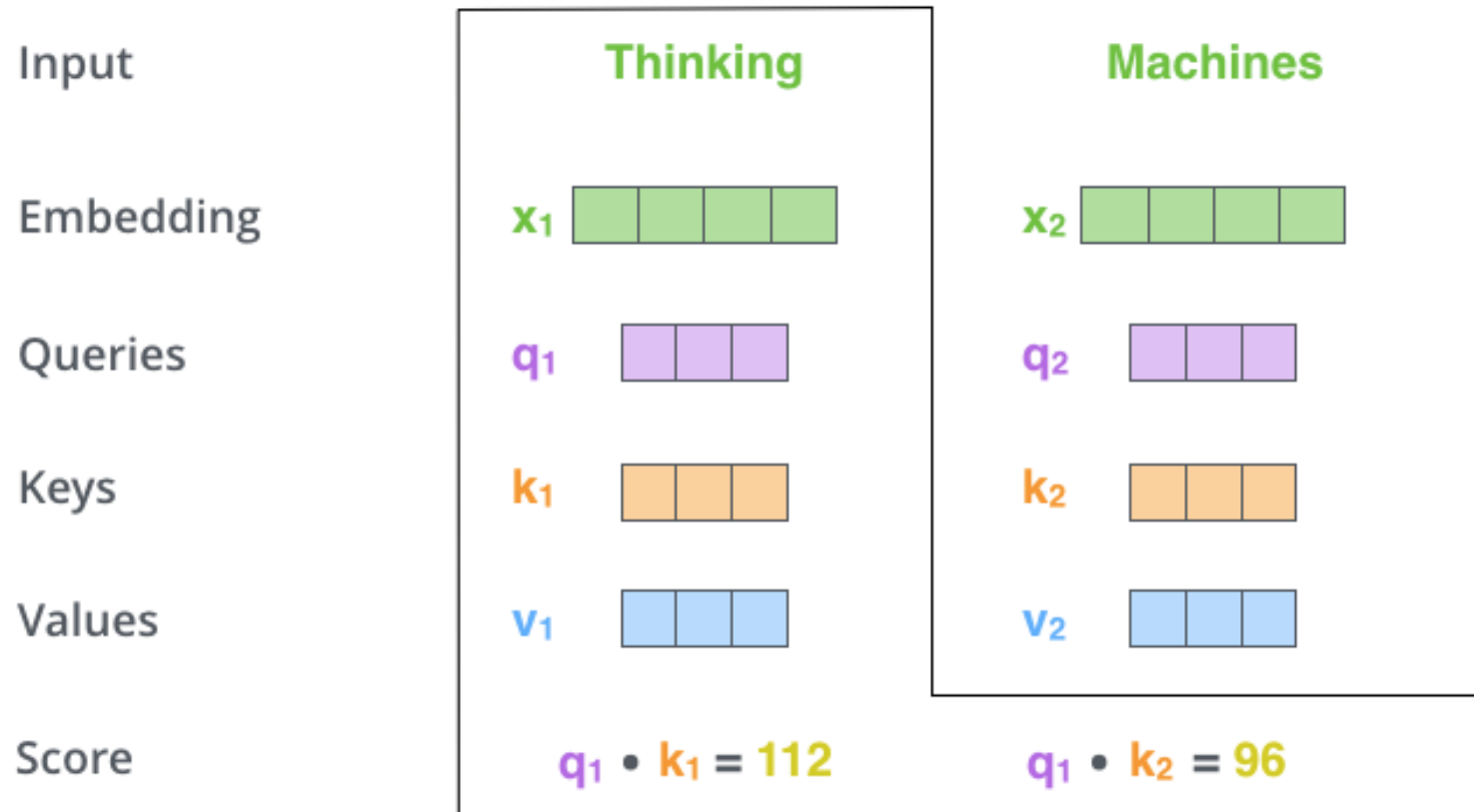
# Self-Attention 계산

- 각 입력 단어에 대해 Q, K, V 성분을 계산한 다음, 단어간의 관련성을 attention으로 계산: “Thinking Machines”는 입력 문장이라고 가정
- 아래 그림에서  $\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q$  등으로 계산.  $\mathbf{W}$  행렬은 훈련시켜서 구함



# Self-Attention 계산 과정

- Scaled dot product로 attention을 계산



# Attention 계산

$$\text{score function}(\mathbf{q}, \mathbf{k}) = \mathbf{q} \cdot \mathbf{k} / \sqrt{d_k}$$

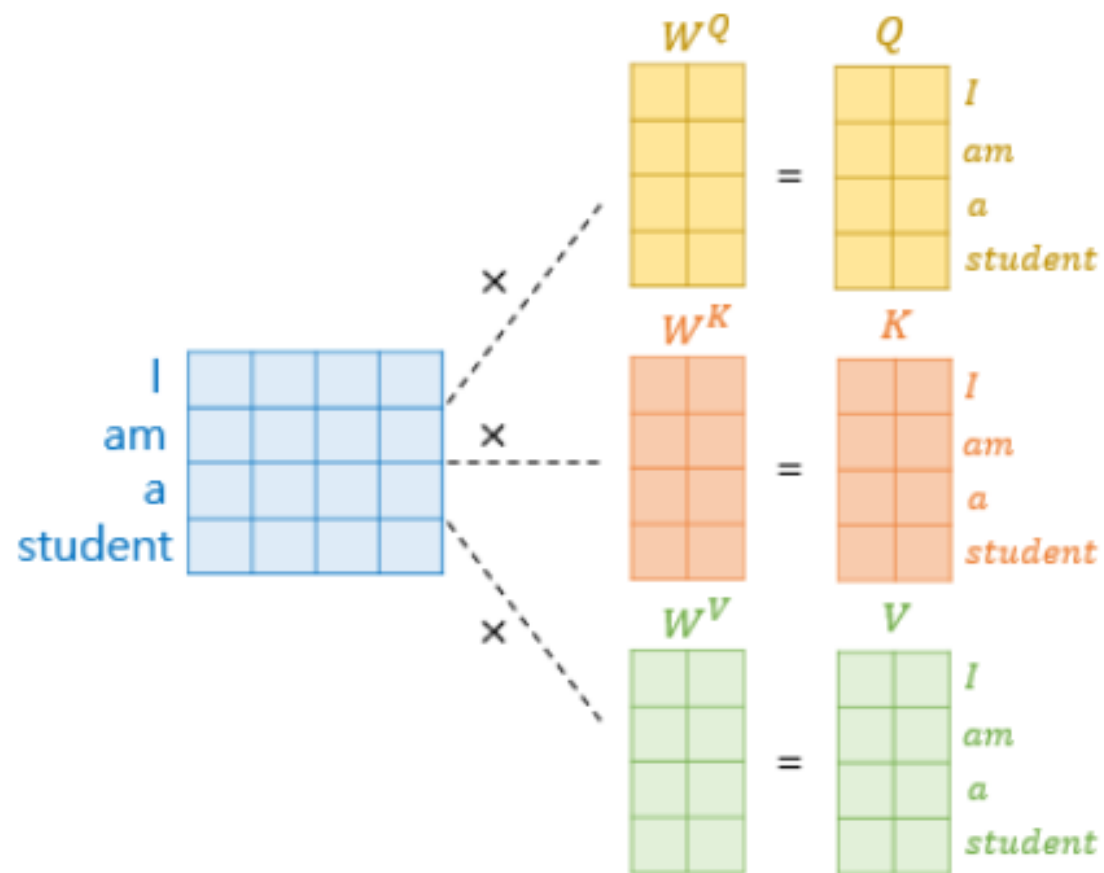
The diagram illustrates the calculation of attention scores for a query  $Q_I$  against four different keys. The query  $Q_I$  is represented by a yellow rectangle. The keys are represented by orange rectangles and are labeled  $K_I^T$ ,  $K_{am}^T$ ,  $K_a^T$ , and  $K_{student}^T$ . The calculations are as follows:

- $Q_I \times K_I^T = 128 \rightarrow 128 / \sqrt{d_k} = 16$
- $Q_I \times K_{am}^T = 32 \rightarrow 32 / \sqrt{d_k} = 4$
- $Q_I \times K_a^T = 32 \rightarrow 32 / \sqrt{d_k} = 4$
- $Q_I \times K_{student}^T = 128 \rightarrow 128 / \sqrt{d_k} = 16$

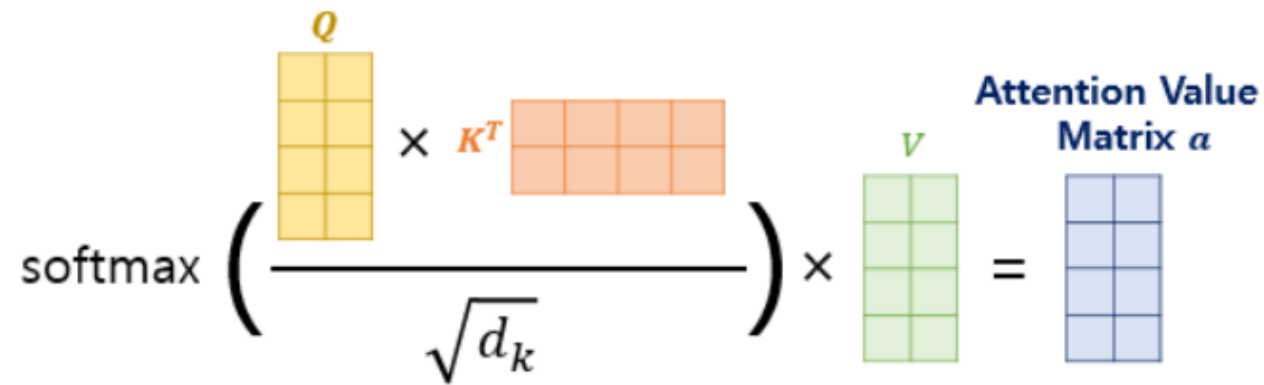
The results 16, 4, 4, and 16 are grouped by a dashed green box and labeled **Attention Score**.

# Matrix attention 계산

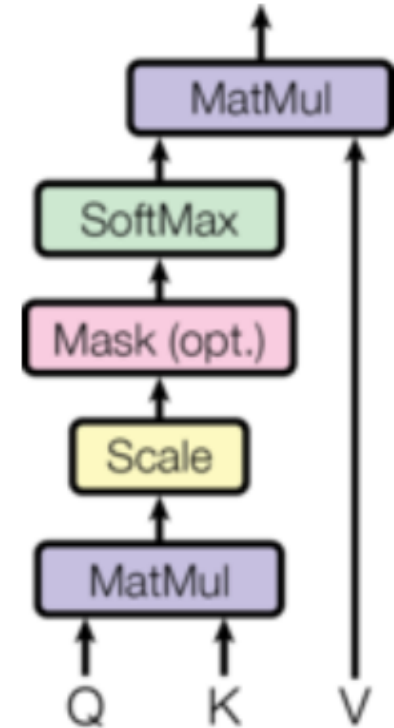
- Attention은 행렬 연산으로 계산할 수 있음



# Attention 계산



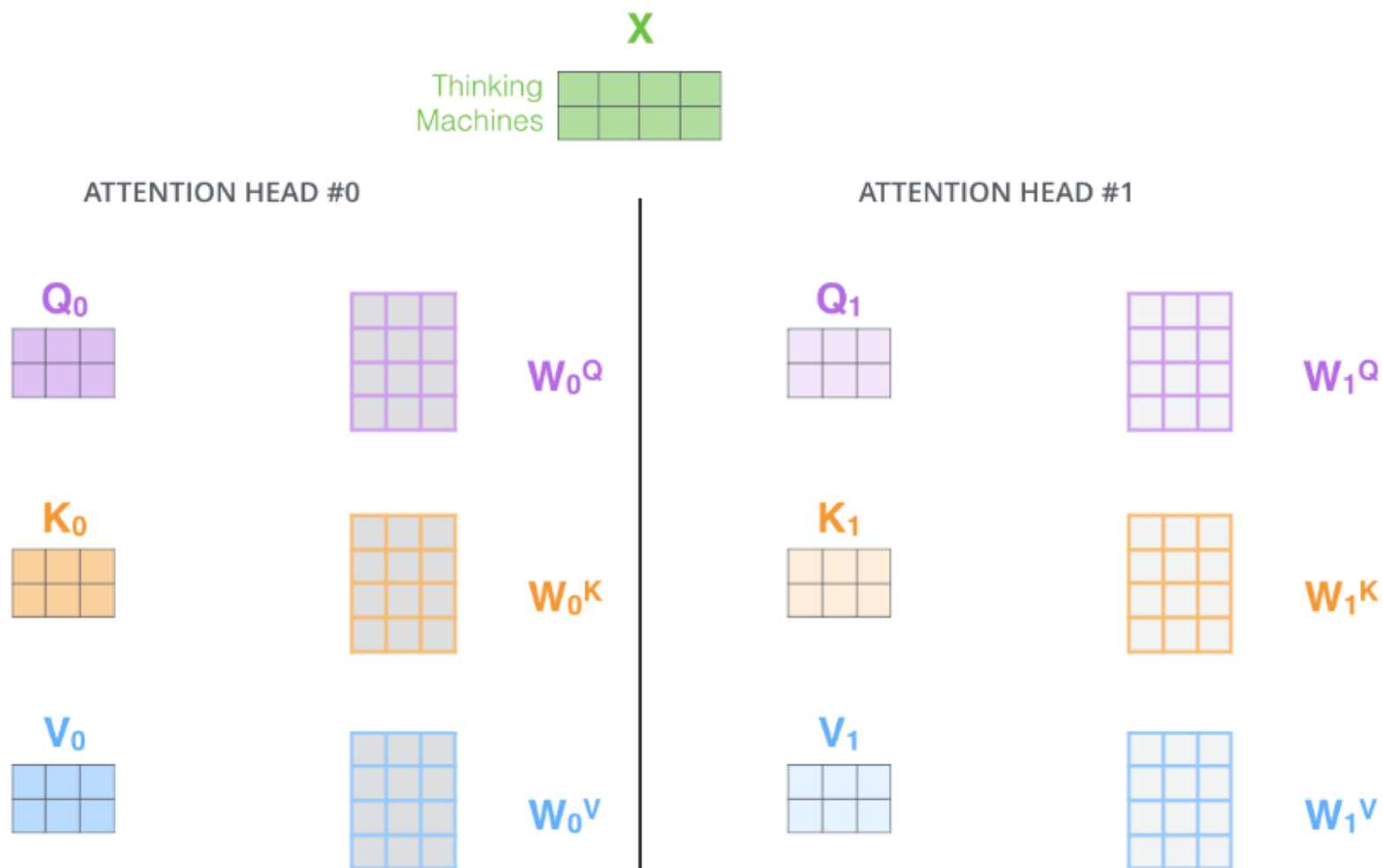
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$





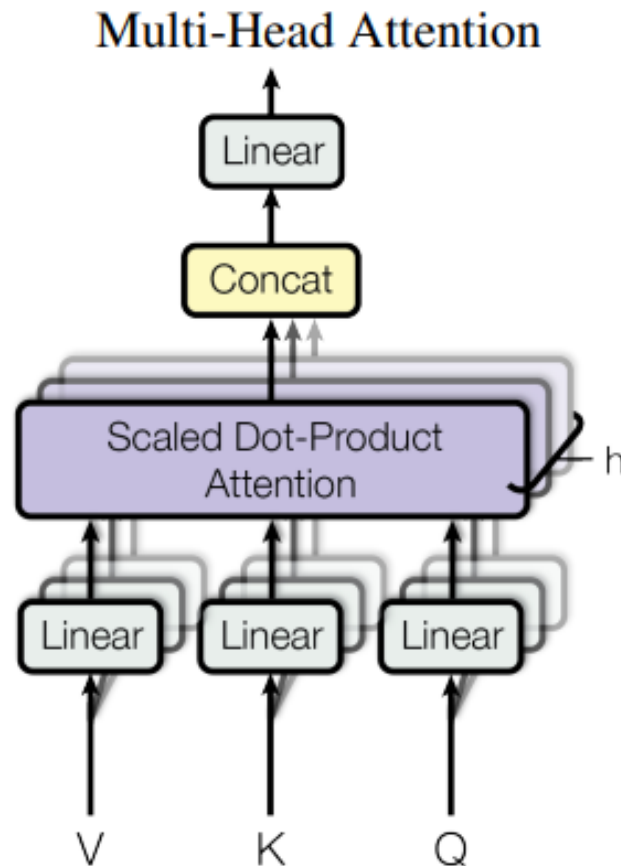
# Multi-head attention

- 문장 내의 정보를 충분히 나타내기 위해 attention 정보를 8개로 확장



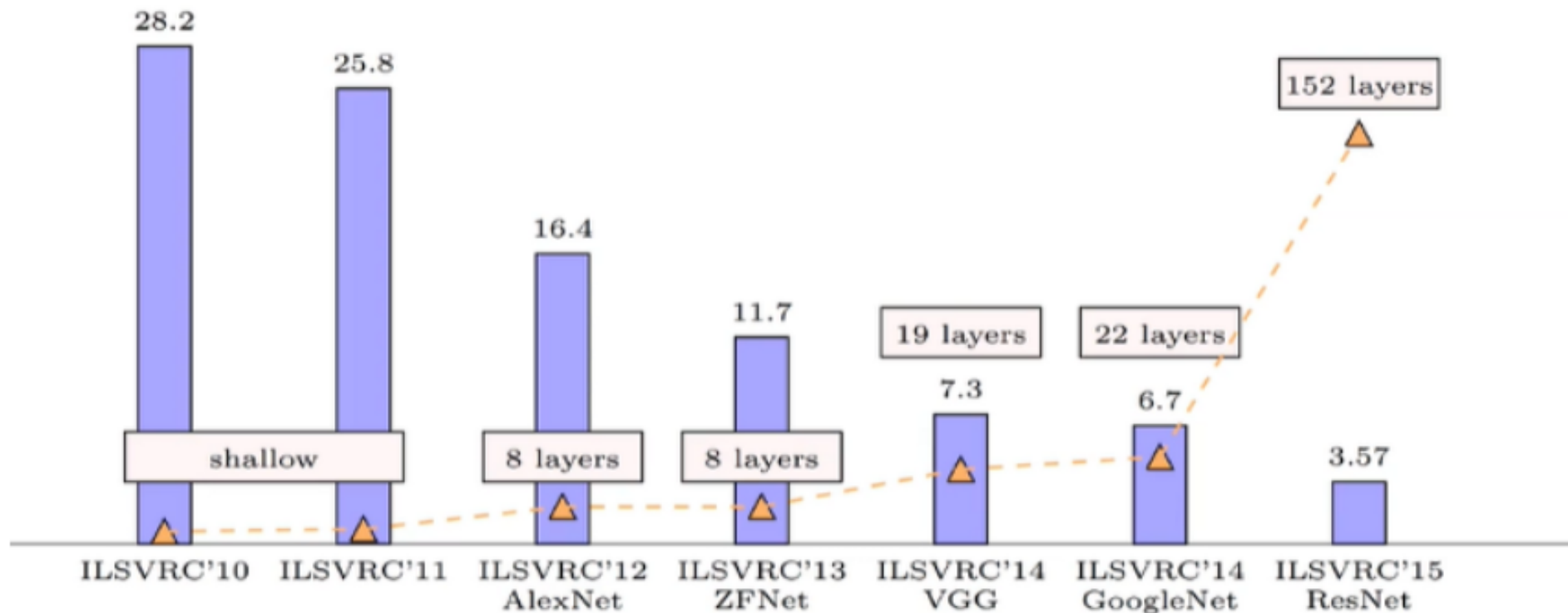
# Multi-head attention 표현

- 각 벡터를 concatenate 하여 정보를 표현



# Residual Connection

- 1,000개 클래스 영상에 대한 인식 알고리즘 경연대회인 ILSVRC의 2015년 우승 알고리즘: **ResNet(Residual Network)**

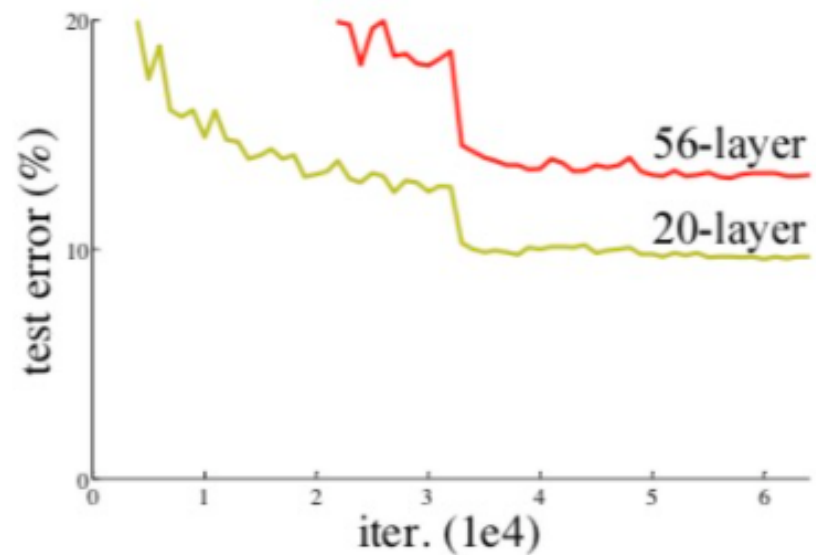
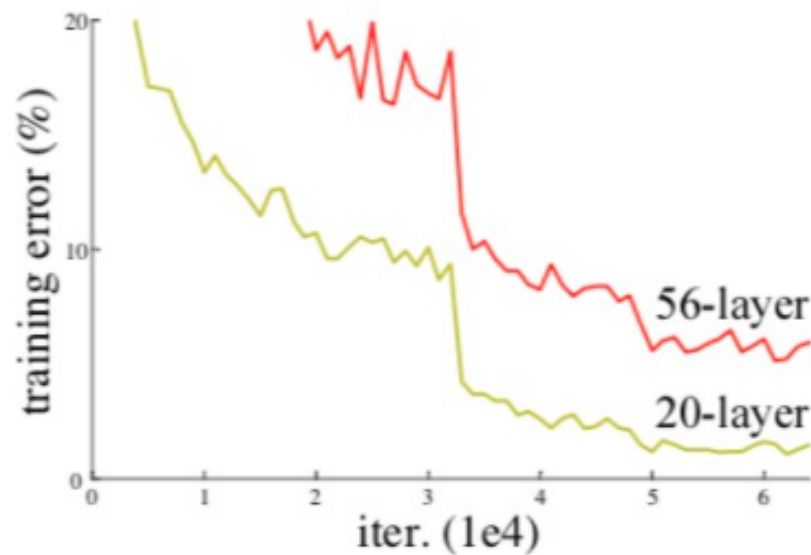


**ILSVRC 우승 알고리즘의 오차율**

# 이전의 Deep Network 한계

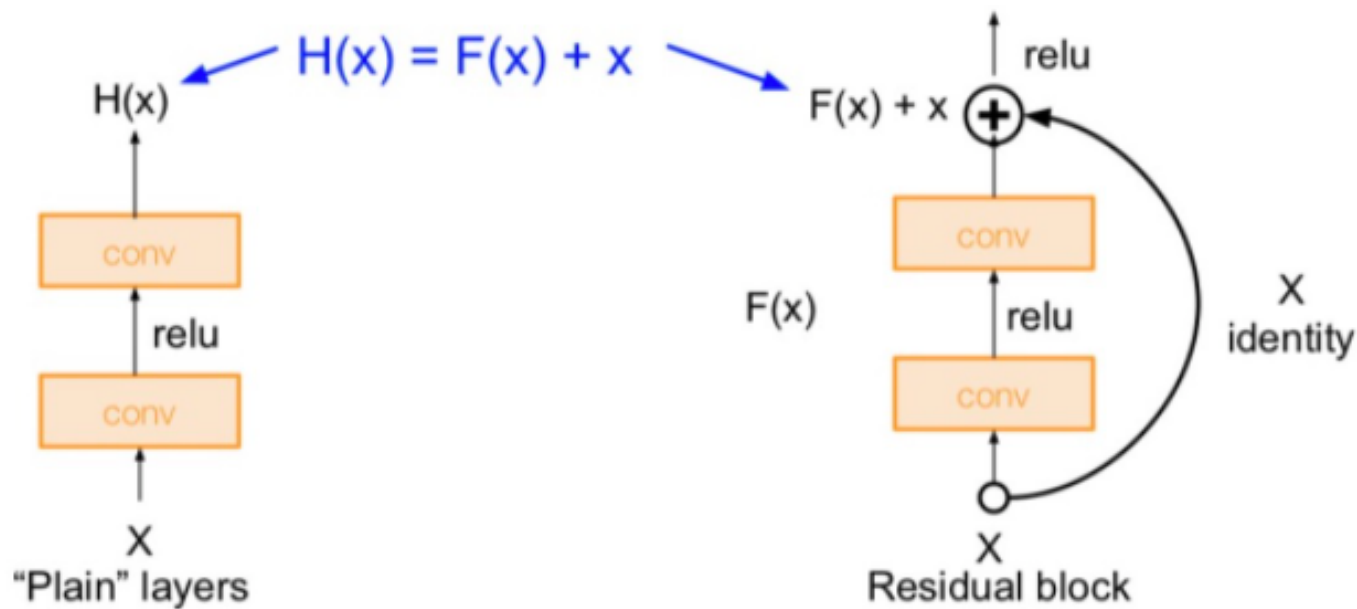
- 레이어 수가 늘어나면 오히려 오차가 증가하는 현상이 발생:

**Gradient vanishing/exploding 현상의 영향**



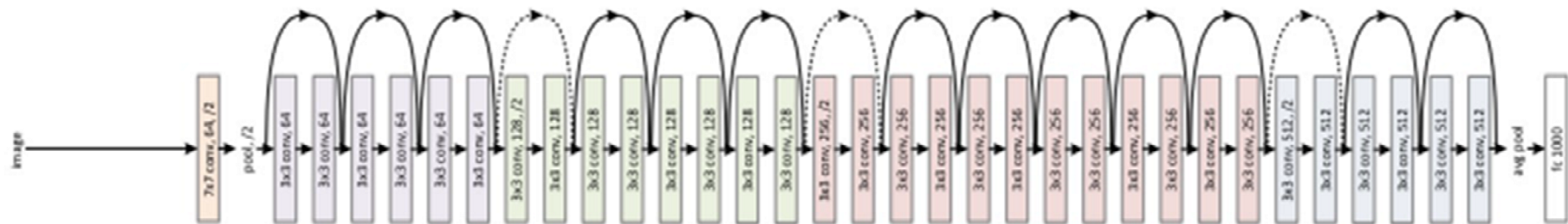
# Residual Connection

- 중간 레이어를 건너뛰어 층을 연결하는 방식
- 신경망 레이어를 이전보다 깊게 쌓을 수 있음



# ResNet 구조

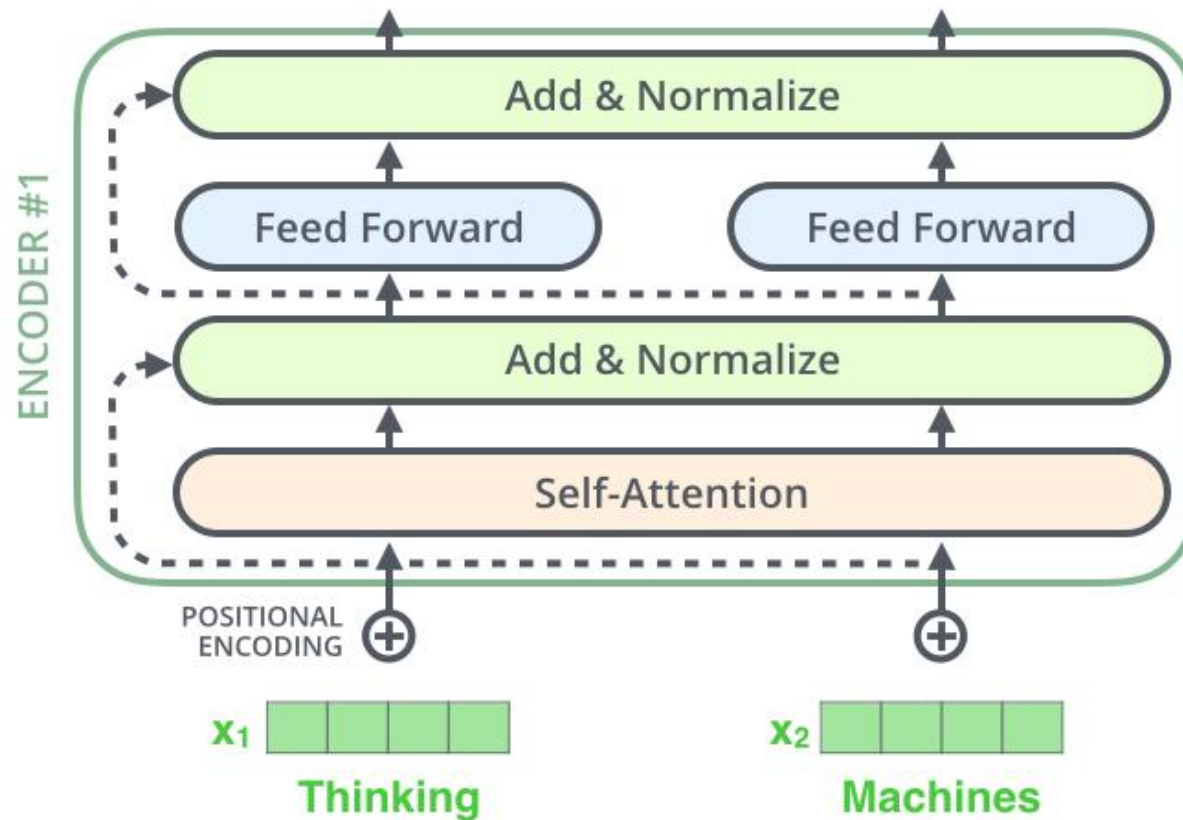
- 아래 구조는 34층을 쌓은 형태임
- ResNet에서는 152층까지 구축했음



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

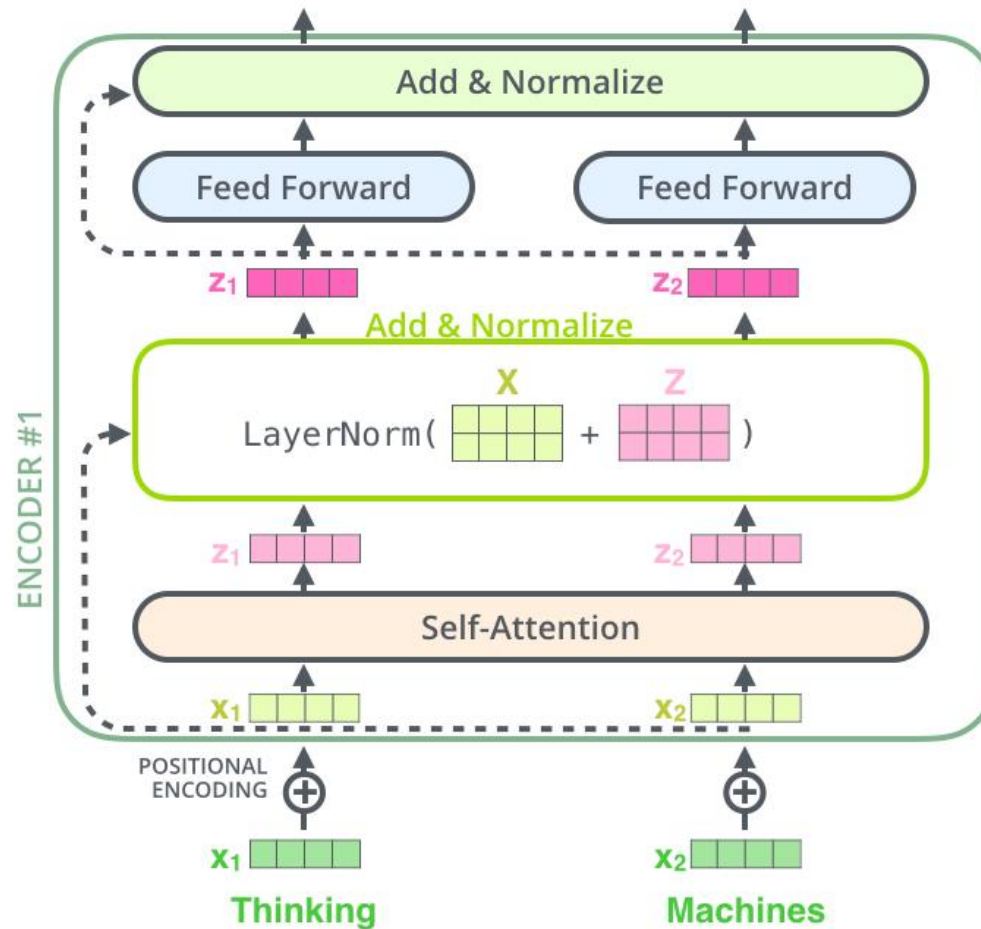
# Transformer Residual Connection

- Attention과 Feed Forward 망에서 residual connection을 사용



# Normalization

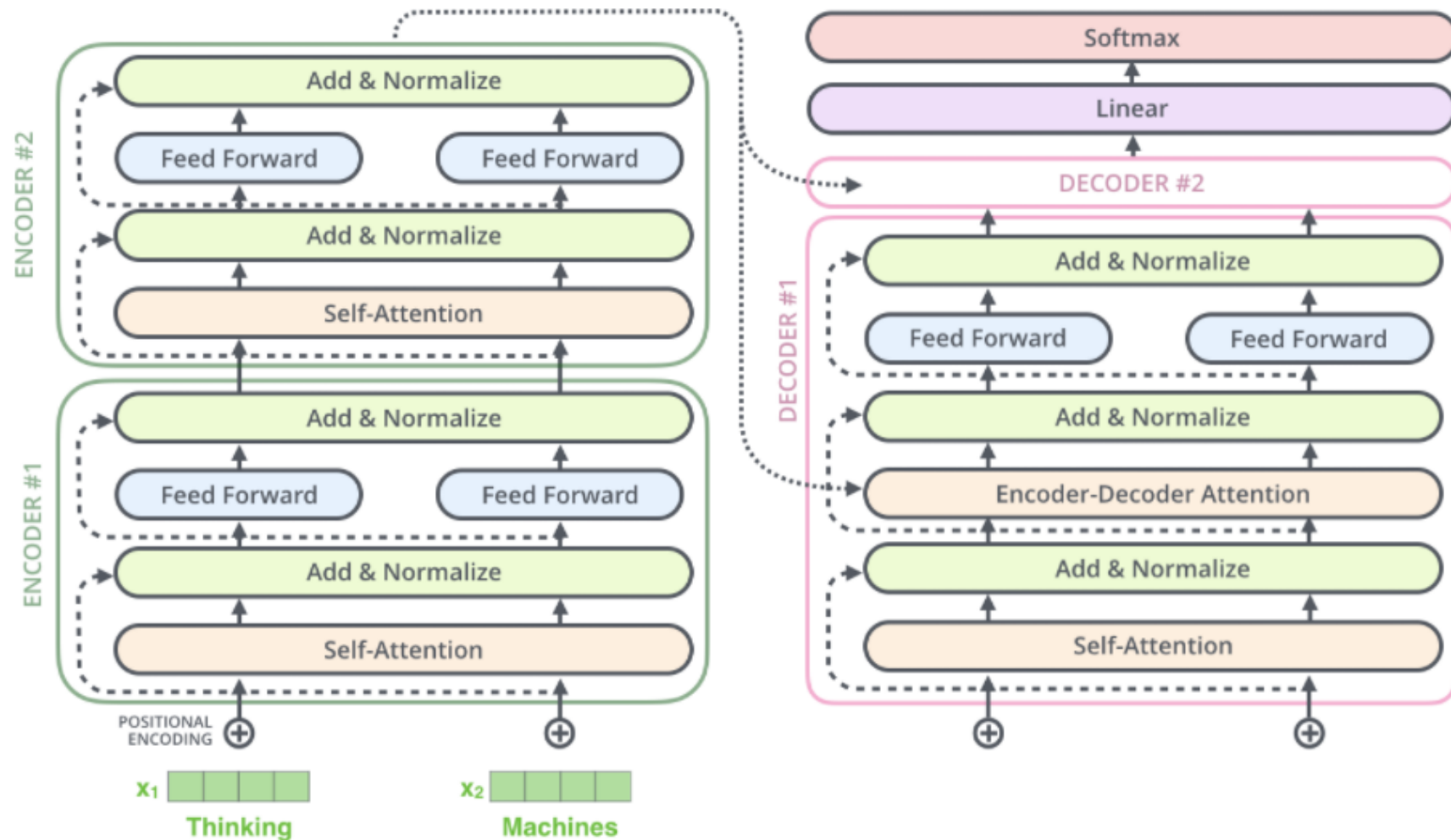
- 훈련의 안정성을 높이기 위해 데이터를 정규화하는 방식 중 하나
- Self-attention 입력과 출력을 더함





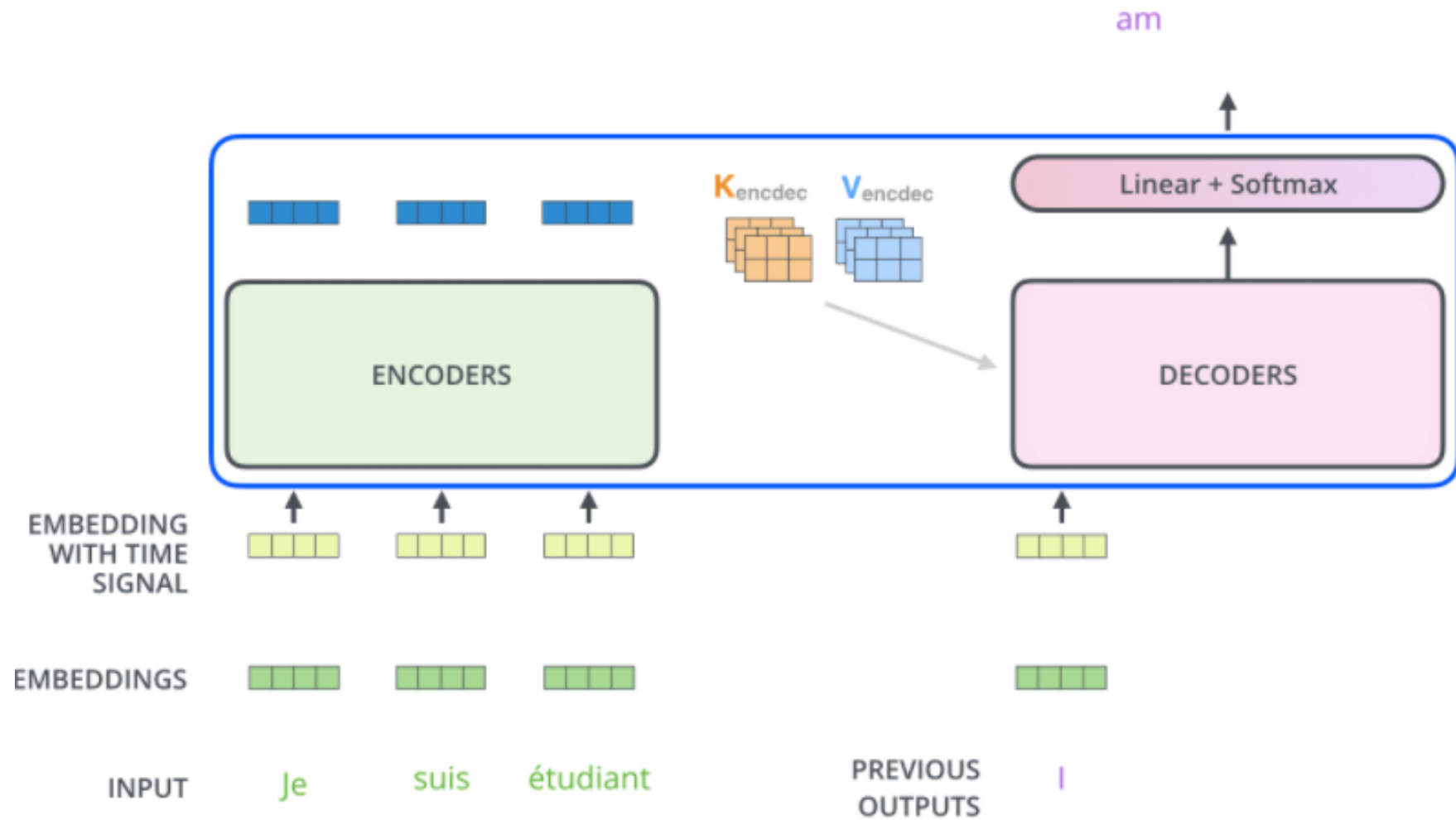
# Decoder 구조

- Attention 정보를 encoder에서 받음



# Decoder 구조

- Encoder에서 K와 V 정보가 전달됨



# Decoder 출력단

- softmax 단을 거쳐 출력 단어가 생성됨

Which word in our vocabulary  
is associated with this index?

am

Get the index of the cell  
with the highest value  
(argmax)

log\_probs



Softmax

logits



Linear

Decoder stack output



# Transformer 번역기 성능

- RNN 방식에 비해 적은 양의 연산을 통해 개선된 성능을 보여줌

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	<b><math>3.3 \cdot 10^{18}</math></b>	
Transformer (big)	<b>28.4</b>	<b>41.0</b>	$2.3 \cdot 10^{19}$	

# Transformer 번역 프로그램

- Tensorflow 사이트에서 포르투갈어-영어 번역 프로그램을 볼 수 있음
  - Transformer model for language understanding: [Transformer model for language understanding | TensorFlow Core](#)
- pytorch 버전의 한국어-영어 번역 프로그램
  - GitHub - kh-kim/simple-nmt: This repo contains a simple source code for advanced neural machine translation based on sequence-to-sequence.
  - <https://github.com/kh-kim/simple-nmt>