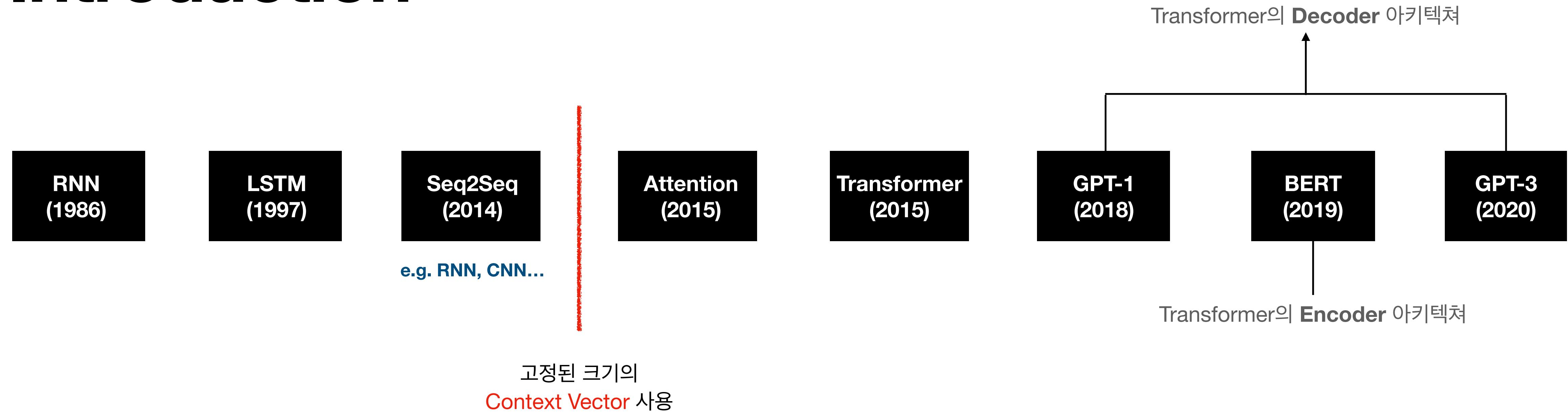


# **Attention is all you need**

**Younjung Kang**

# Introduction

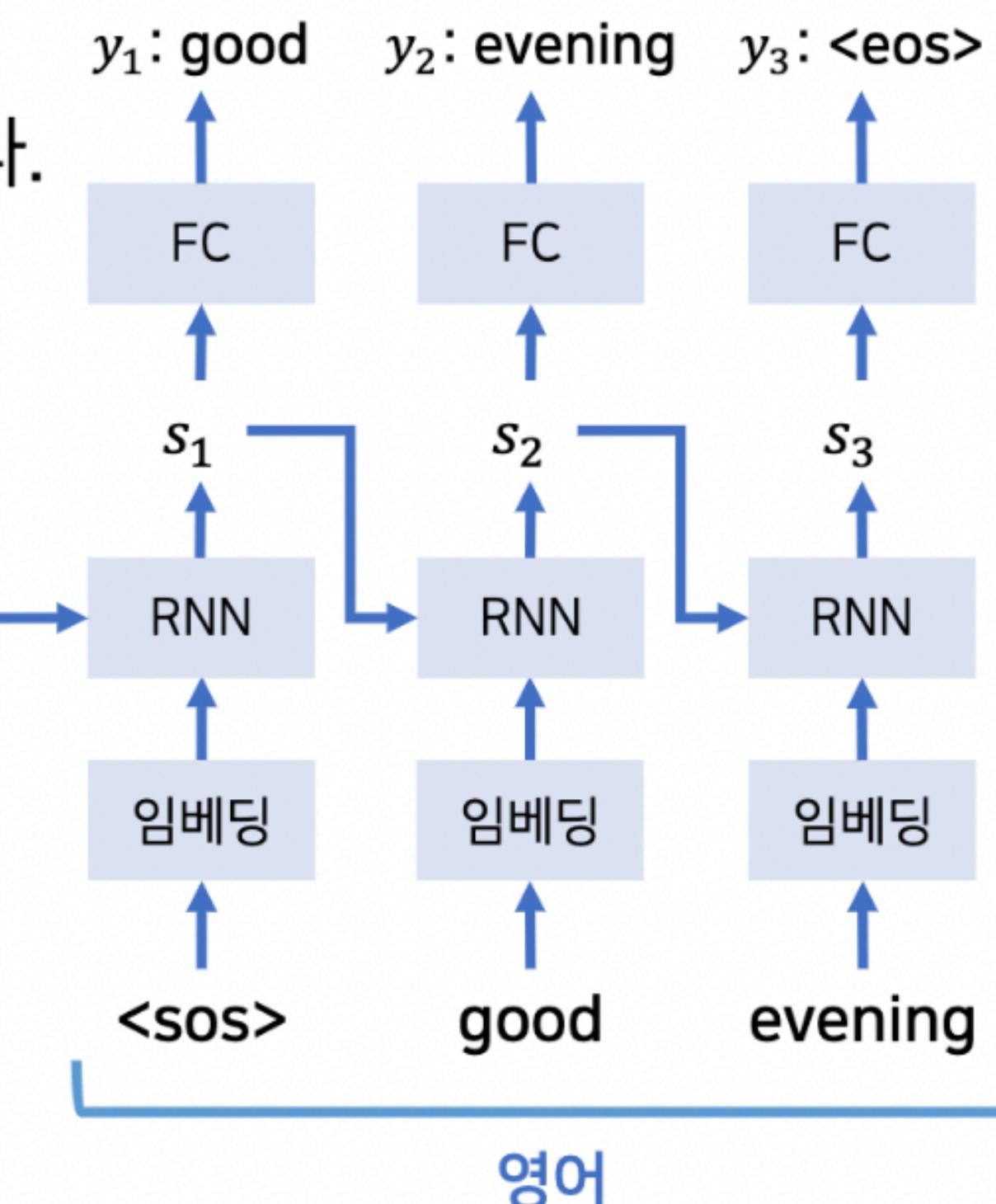
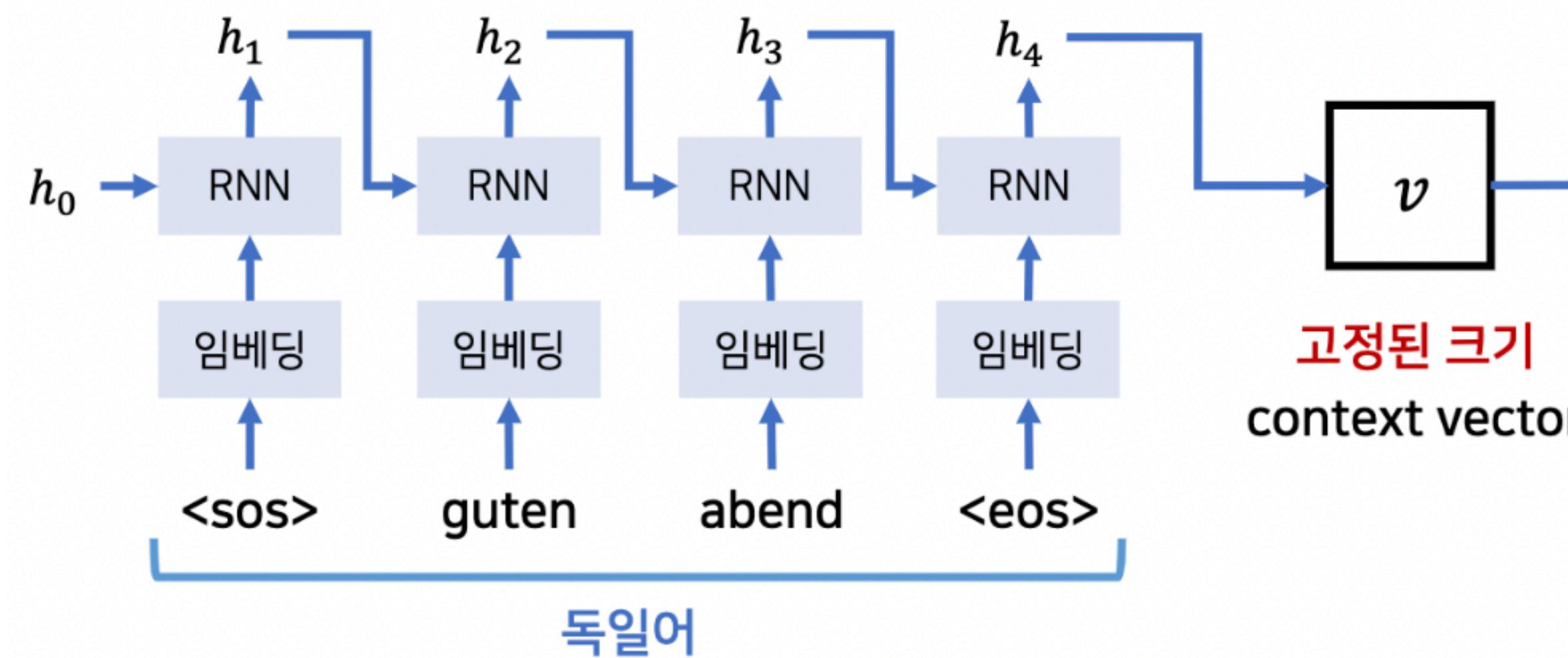


Self-Attention 모델의 등장 이전,  
대부분의 자연어 처리는 Seq2Seq(즉, Encoder-Decoder) 구조를 가지는 RNN, CNN 등의 모델로 구현되었다.  
초기에는 RNN 모델만을 주로 이용하다가, Attention이 추가된 RNN 계열의 모델을 이용하게 된다.

하지만 이 모델들에게는 한계점이 있었고, 이를 극복하는 Self-Attention 기법인 Transformer 모델이 해당 논문을 통해 처음으로 제안된다.

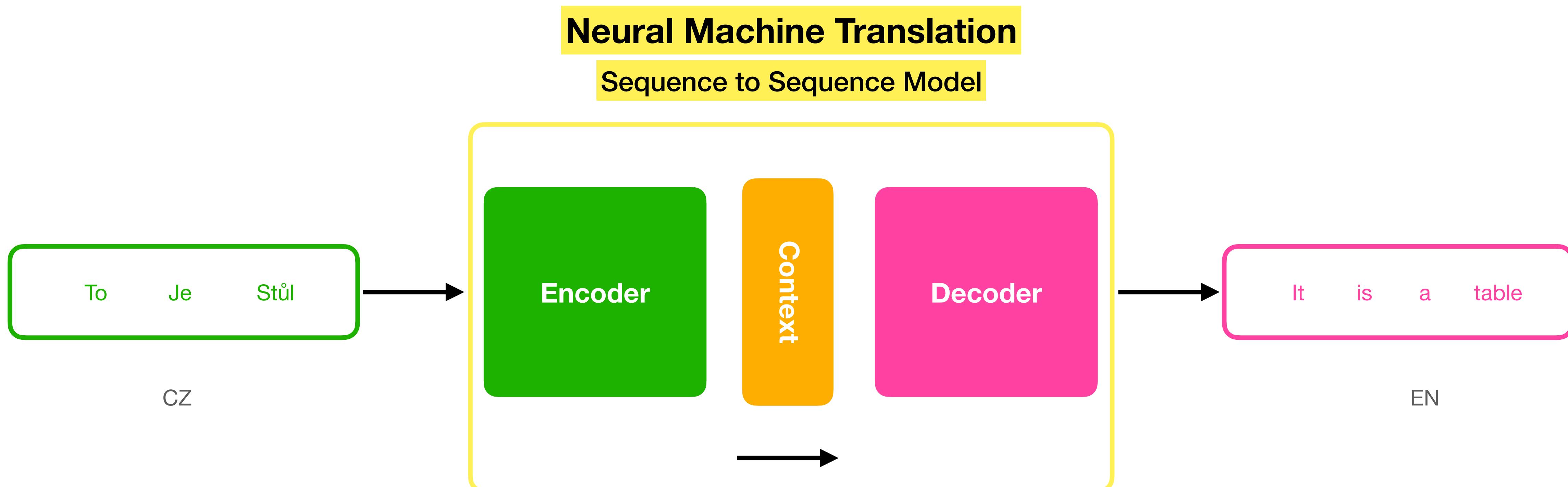
# Recurrent neural networks(RNN)

- context vector  $v$ 에 소스 문장의 정보를 압축합니다.
  - 병목(bottleneck)이 발생하여 성능 하락의 원인이 됩니다.



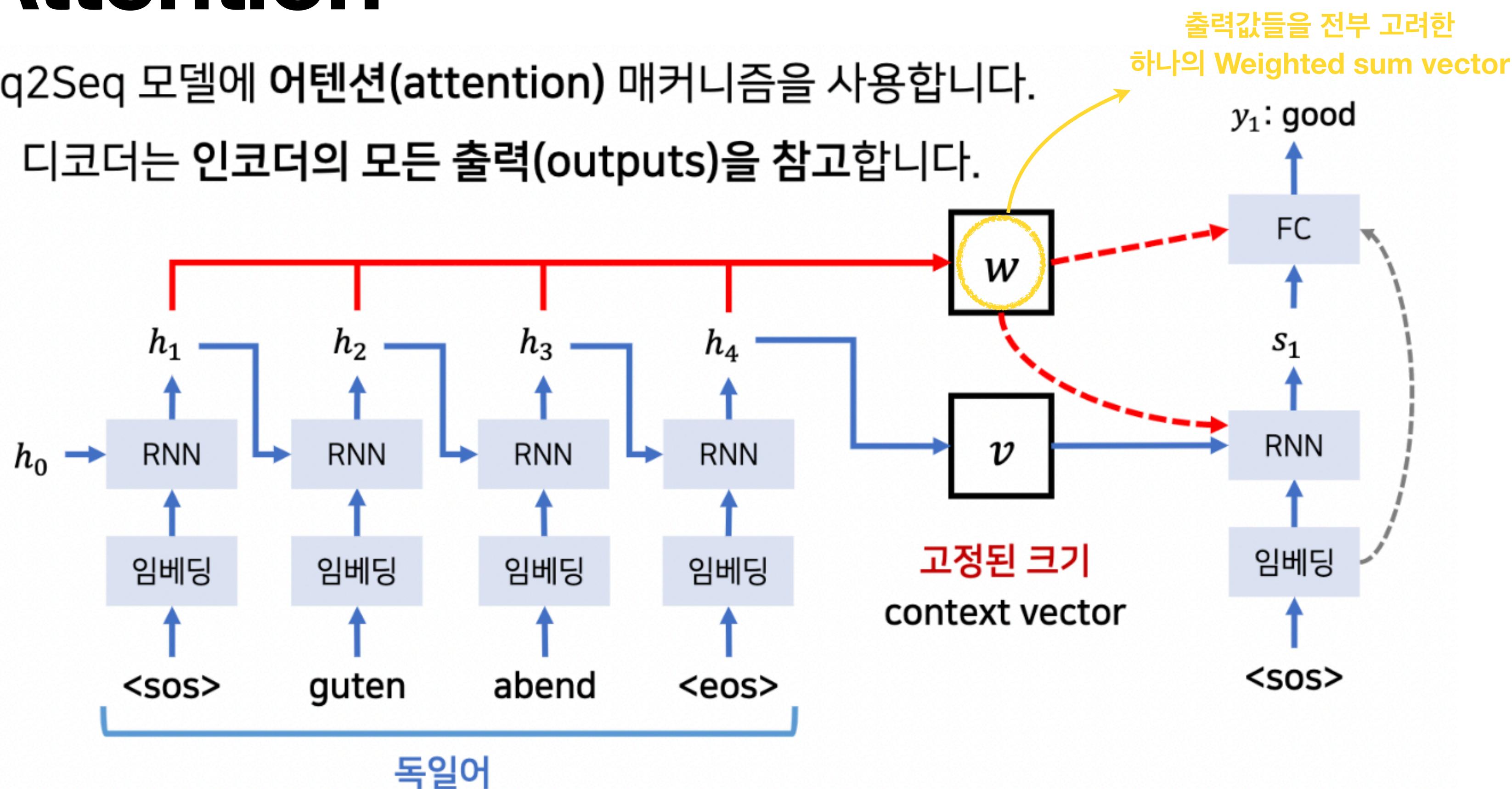
- 이전 상태에 종속적
- 병렬 처리 불가능
- 속도 느림
- Context vector 사이즈 고정 + 고정된 크기의 벡터에 모든 입력 정보 압축 -> 정보 손실 발생

# Recurrent neural networks(RNN)



# RNN + Attention

- Seq2Seq 모델에 어텐션(attention) 매커니즘을 사용합니다.
  - 디코더는 인코더의 모든 출력(outputs)을 참고합니다.



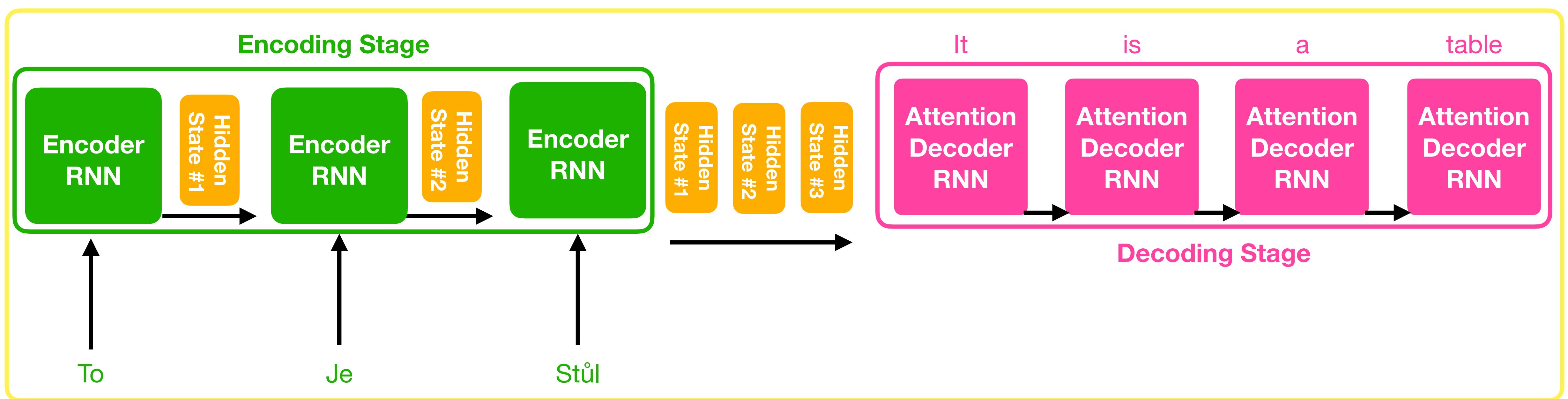
출처: [딥러닝 기계 번역] Transformer: Attention Is All You Need (꼼꼼한 딥러닝 논문 리뷰와 코드 실습), 동빈나(2020)

하나의 context vector를 사용하여 전체를 출력하지 않고,  
각 출력마다의 encoder 결과값들을 전부 참고한 Weighted sum vector( $w$ )를 이용한다.

# RNN + Attention

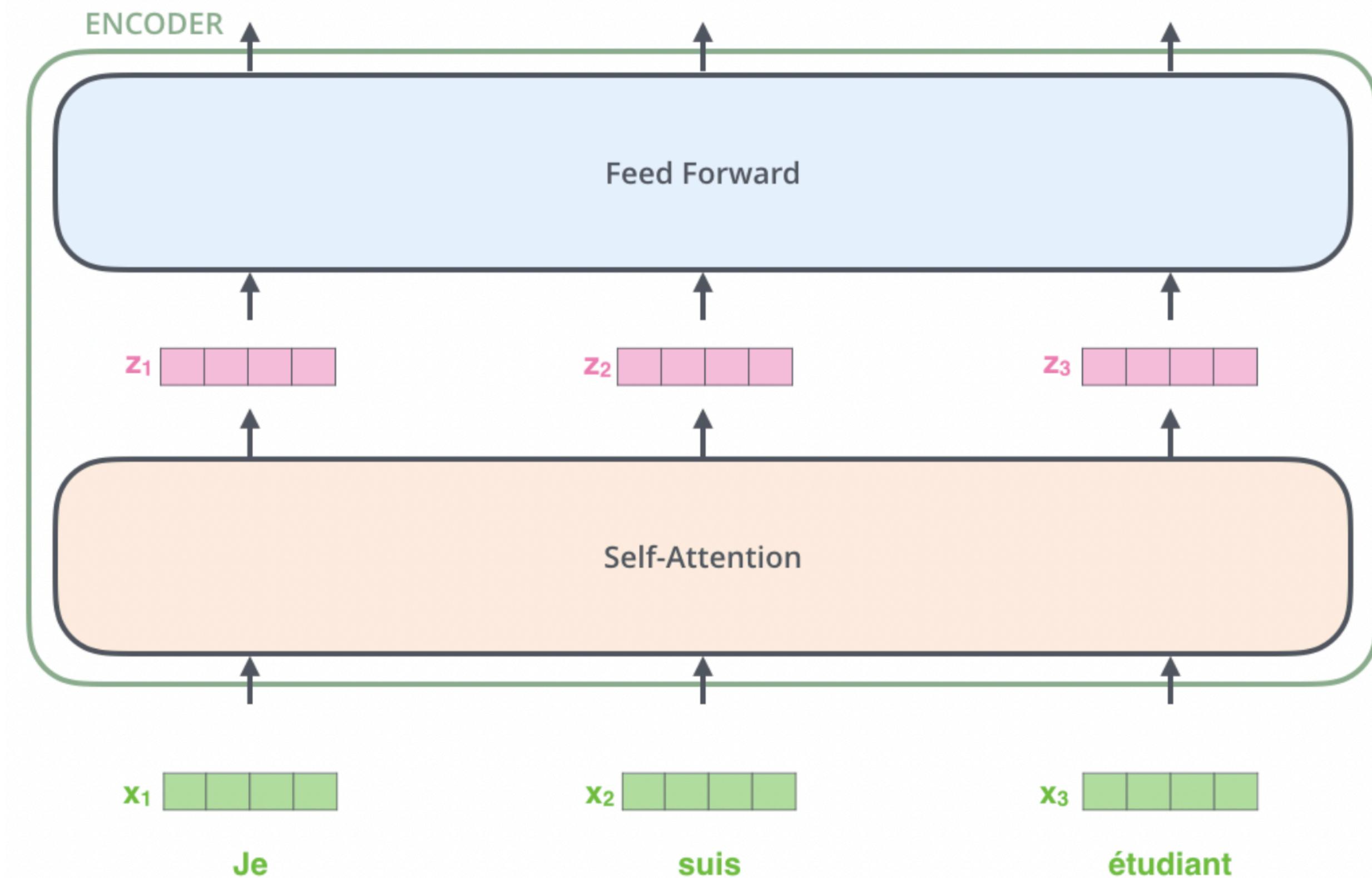
## Neural Machine Translation

### Sequence to Sequence Model with Attention



해당 시점에서 예측해야 할 단어와 연관이 있는 입력 단어 부분을 조금 더 집중해서 볼 수 있다.  
-> 하지만 여전히 RNN 계열의 모델을 사용하기 때문에 속도가 느리다는 단점이 존재한다.

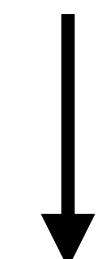
# Transformer



RNN, CNN 구조를 탈피한 self-attention 메커니즘을 제안!

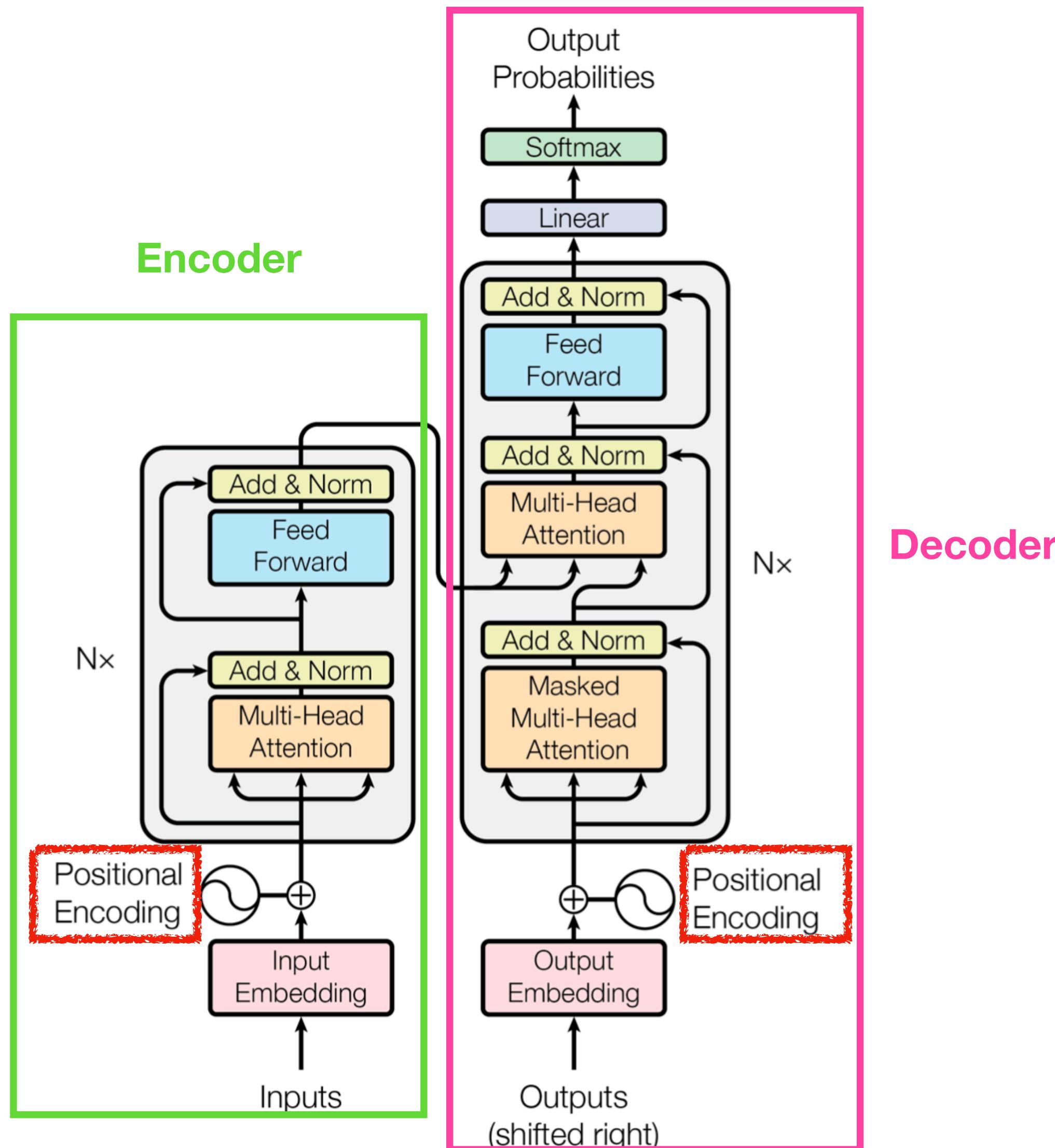


RNN 구조의 가장 큰 문제였던 순차적 계산(병렬 처리 불가능)을 행렬곱을 사용해 한 번에 처리할 수 있게 하였다.



즉, 기존의 encoder-decoder 구조는 그대로 유지하되, RNN구조는 탈피했기 때문에 **병렬처리가 가능해지고, 시간이 단축된다.**

# Transformer



- 인코더-디코더로 구성
- Attention 과정을 여러 레이어에서 반복
- **Position Encoding**으로 행렬을 이용해 입력 데이터의 순서가 섞인 문제를 해결
- 모델은 크게 1. Self-attention, 2. Position-wise feed forward network의 두 가지 구조를 합한 것과 같다.
- 마지막 인코더 레이어의 출력이 모든 디코더 레이어에 입력된다.

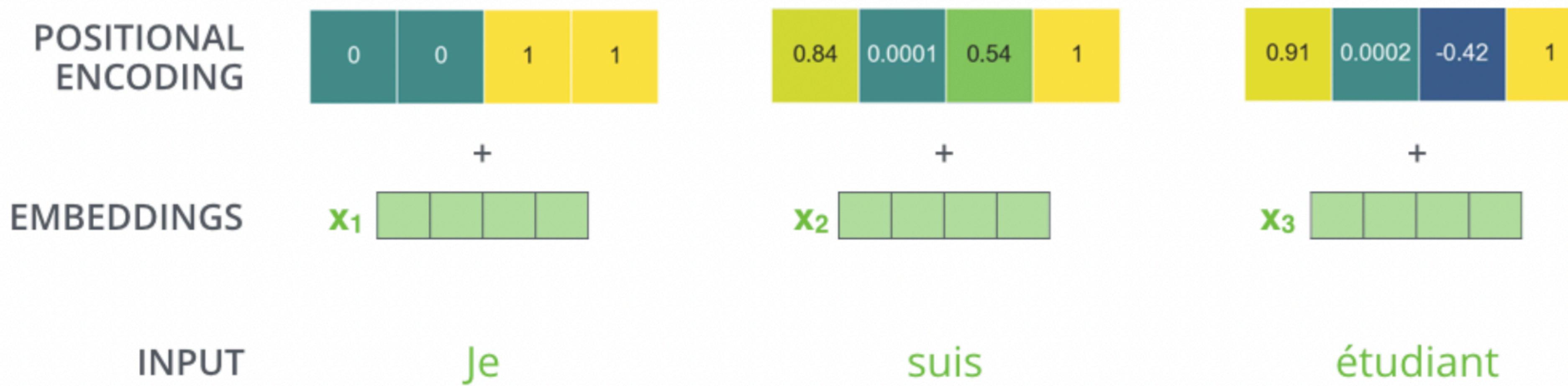
Figure 1: The Transformer - model architecture.

# Positional Encoding

-> 문장 내 각각의 단어들의 순서에 대한 정보를 알려주기 위해 사용

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

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

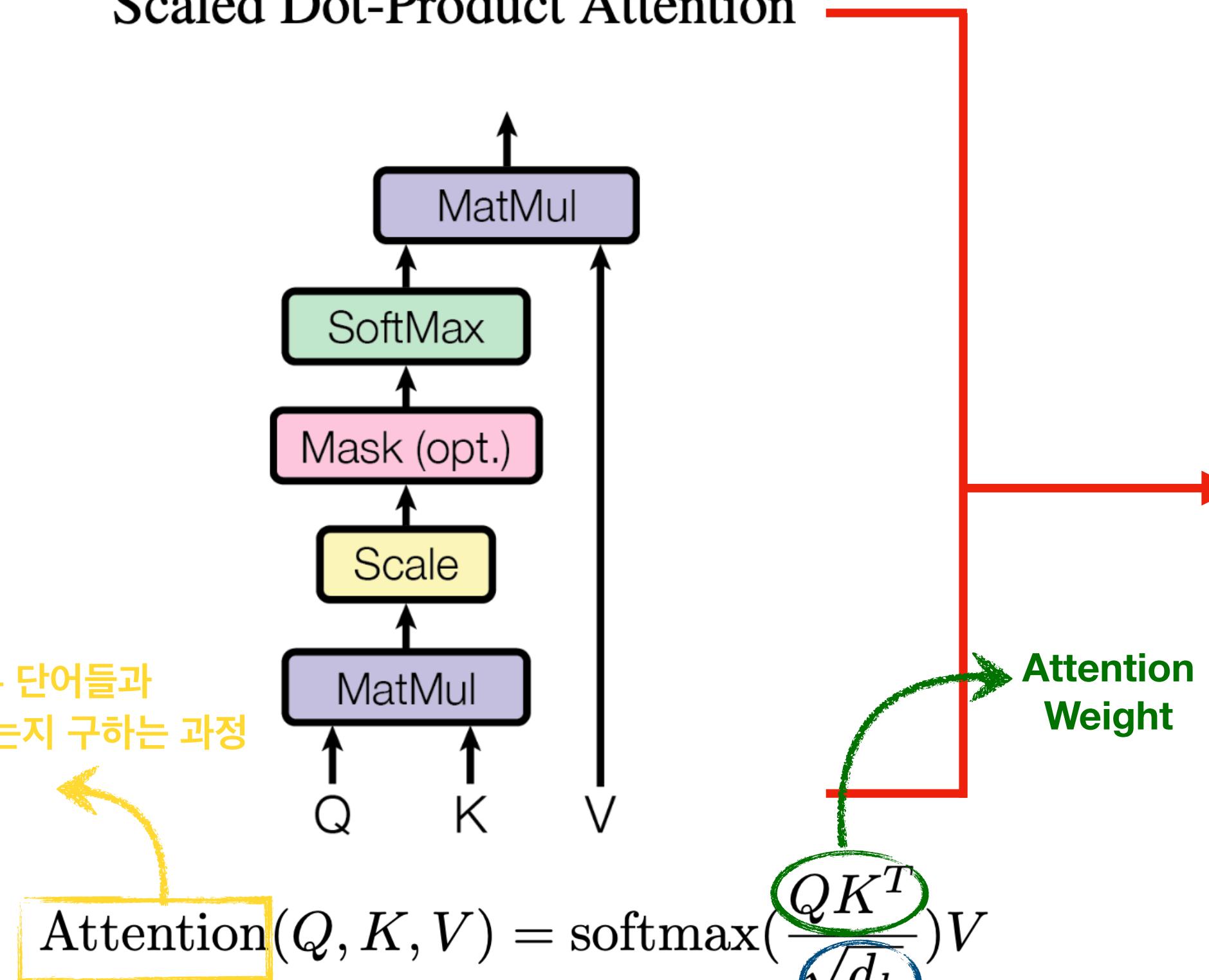


- Position encoding된 값은 초기 임베딩된 단어벡터와 함께 합산하여 초기 입력으로 사용된다.
- Position encoding된 벡터 크기와 embedding vector크기가 같기 때문에 더하여 인풋값을 출력할 수 있다.

# 1. Multi-head self-attention

Scaled Dot-Product Attention

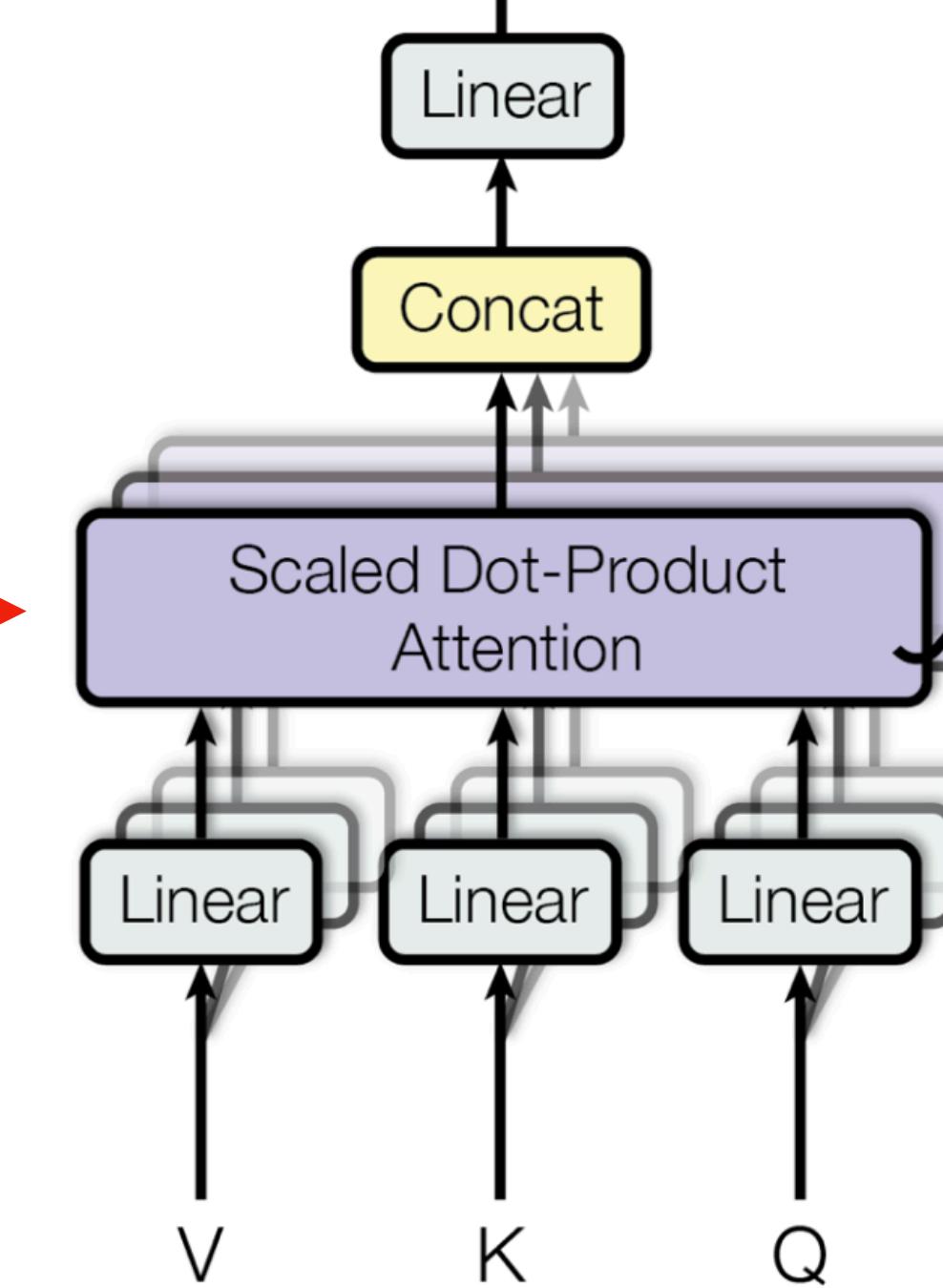
어떤 단어가 다른 단어들과  
어떠한 연관성을 가지는지 구하는 과정



- ▶ 어텐션을 위한 세 가지 입력 요소
  - 쿼리(Query) : 주체
  - 키(Key) : 대상
  - 값(Value)

Multi-Head Attention

: Scaled Dot-Product Attention을  
병렬 처리하기 위해 여러 개 사용한 것



: head의 개수

(e.g. head = 8인 경우,  
기존의 scaled dot-product attention을 동시 수행하기  
위해 기존의 512차원의 각 단어 벡터를 8로 나누어  
64차원의 Query, Key, Value 벡터로 바꾸어 attention을  
수행한다.)

Dimension

-> Gradient vanishing 문제를 막기 위함.

# 1. Multi-head self-attention

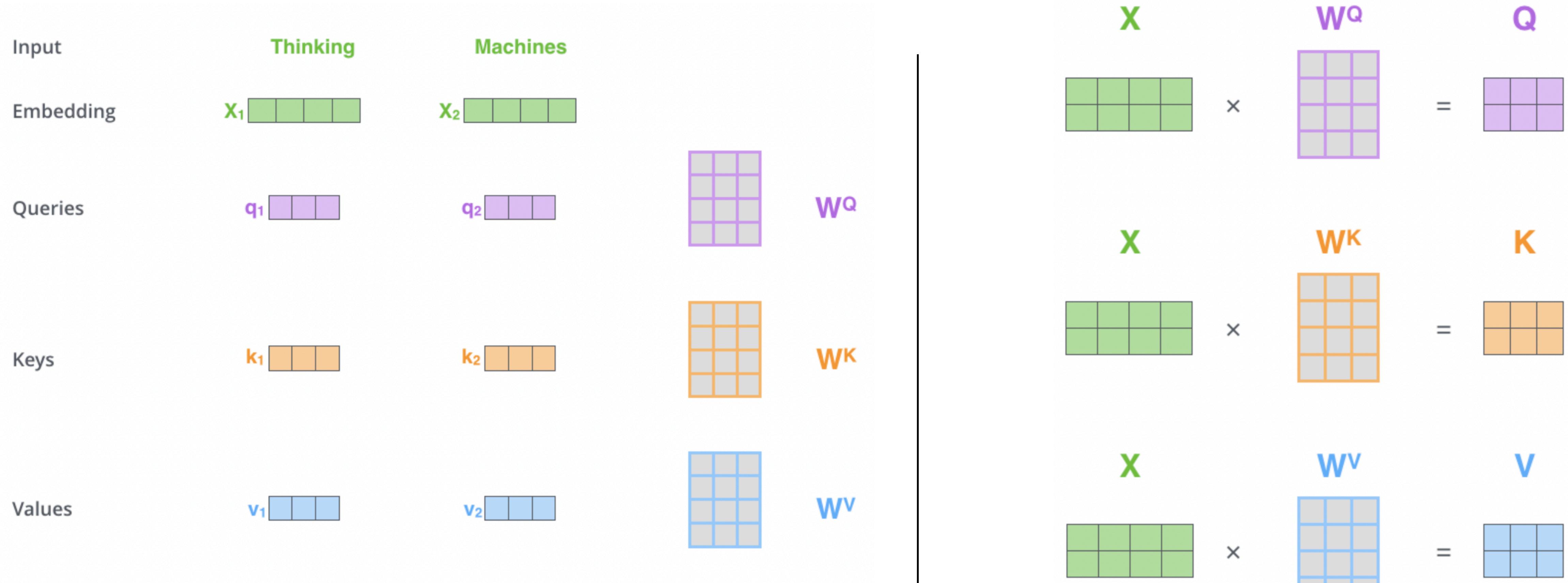
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

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

하나의 MultiHead는 작은 head(dot-product attention)들의 합.

$QW_i^Q$ ,  $KW_i^K$ ,  $VW_i^V$ 는 기존의 512차원의 입력 단어 벡터를 64차원으로 나눠주기 위한 가중치 행렬.

# 1. Multi-head self-attention -> Matrix Calculation

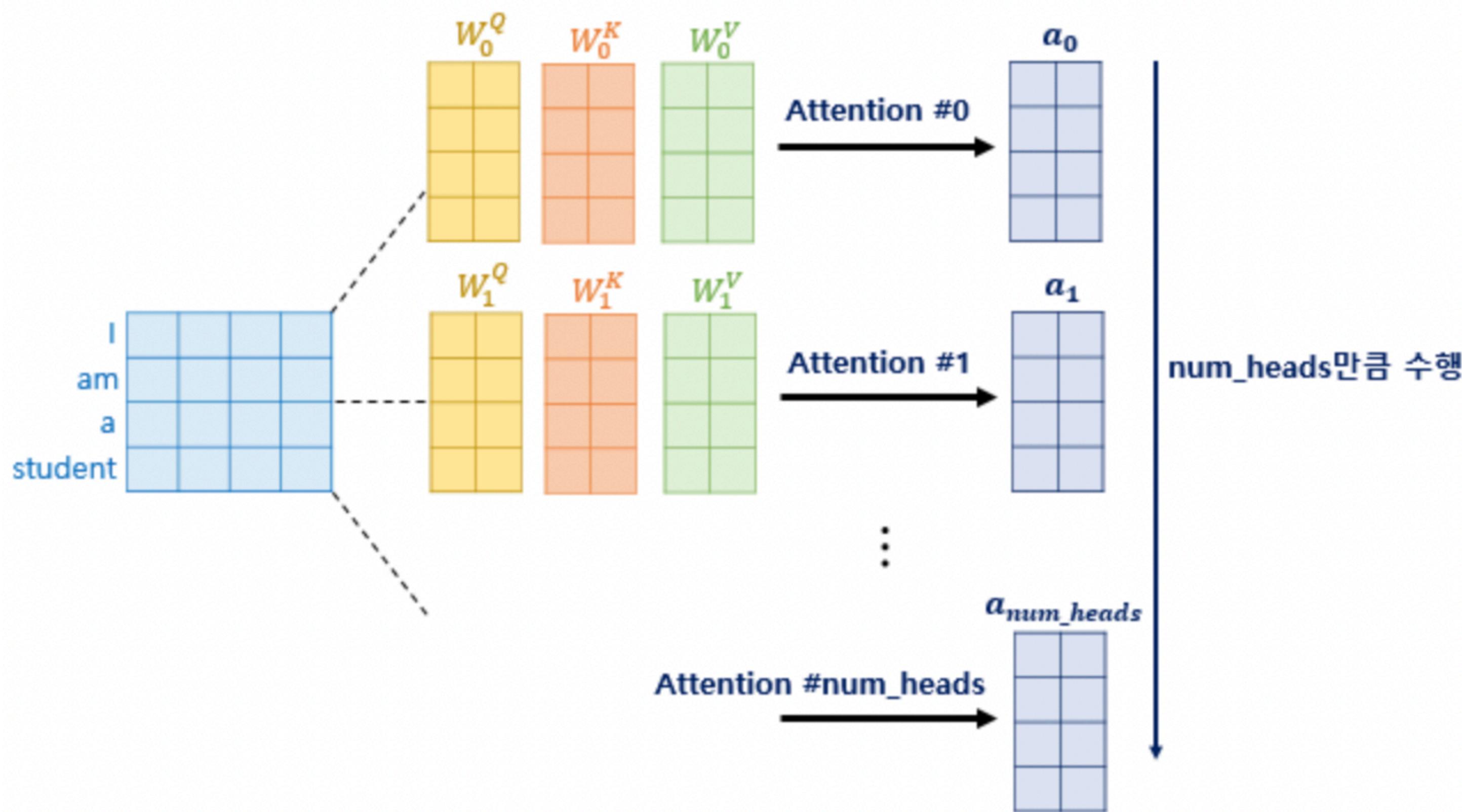


-> 4차원의 단어 벡터가 들어오면 각각의 3x4의 가중치 행렬과 곱해져 각각의 Query, Key, Value는 3차원의 벡터가 된다.

이렇게 벡터 크기를 조정해주는 이유는 초반에 언급한 것처럼 병렬처리를 하기 위함이다.

위 그림은 병렬처리하는 것을 보여주는 예이다.  
 왼쪽 그림에서 단어는 1개씩 떨어져있지만,  
 위의 그림  $X$ 를 보면 단어가 2개 붙어있는 것을 확인할 수 있다.  
 이렇게 병렬 처리를 하는 것이다.

# 1. Multi-head self-attention

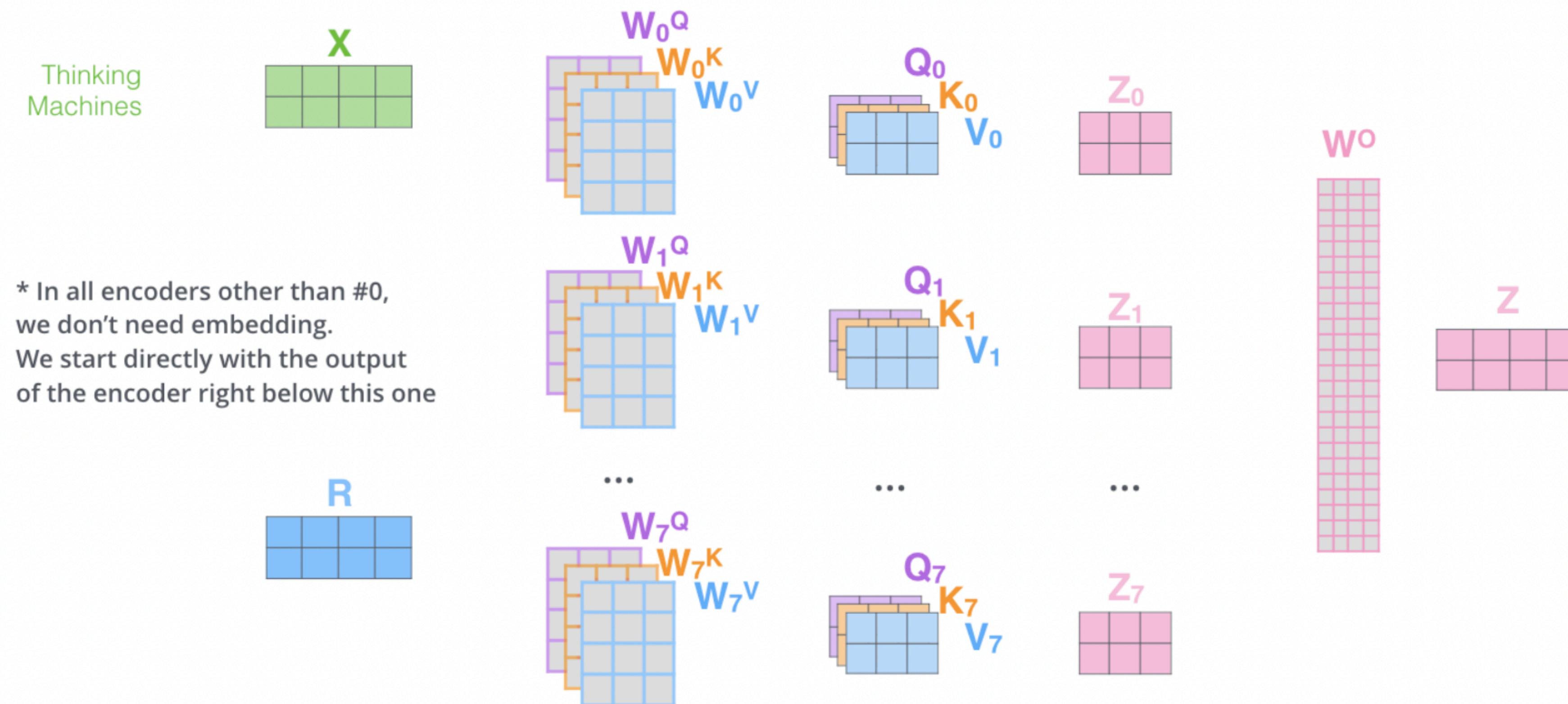


I, am, a, student 각각의 단어가 하나의 행렬을 이루고,  
가중치 행렬과 곱해진다. 그 다음, 여러 개의 헤드를 두고 attention을  
수행한다.  
(\*참고로 각각의 헤드값은 다르다.)

이 논문에서는 한 번 attention을 하는 것 보다 여러 번의 attention을  
병렬로 수행하는 것이 더 효과적이라고 판단하였다.  
Attention을 여러 번 수행한다는 것은 다양한 시각으로 정보를 수집한  
다는 것이며, 이는 더 정확하고 의미있는 결과를 도출할 것이다.

# 1. Multi-head self-attention

- 1) This is our input sentence\*  $X$
- 2) We embed each word\*  $R$
- 3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices  $W_0^Q, W_0^K, W_0^V$
- 4) Calculate attention using the resulting  $Q/K/V$  matrices
- 5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer



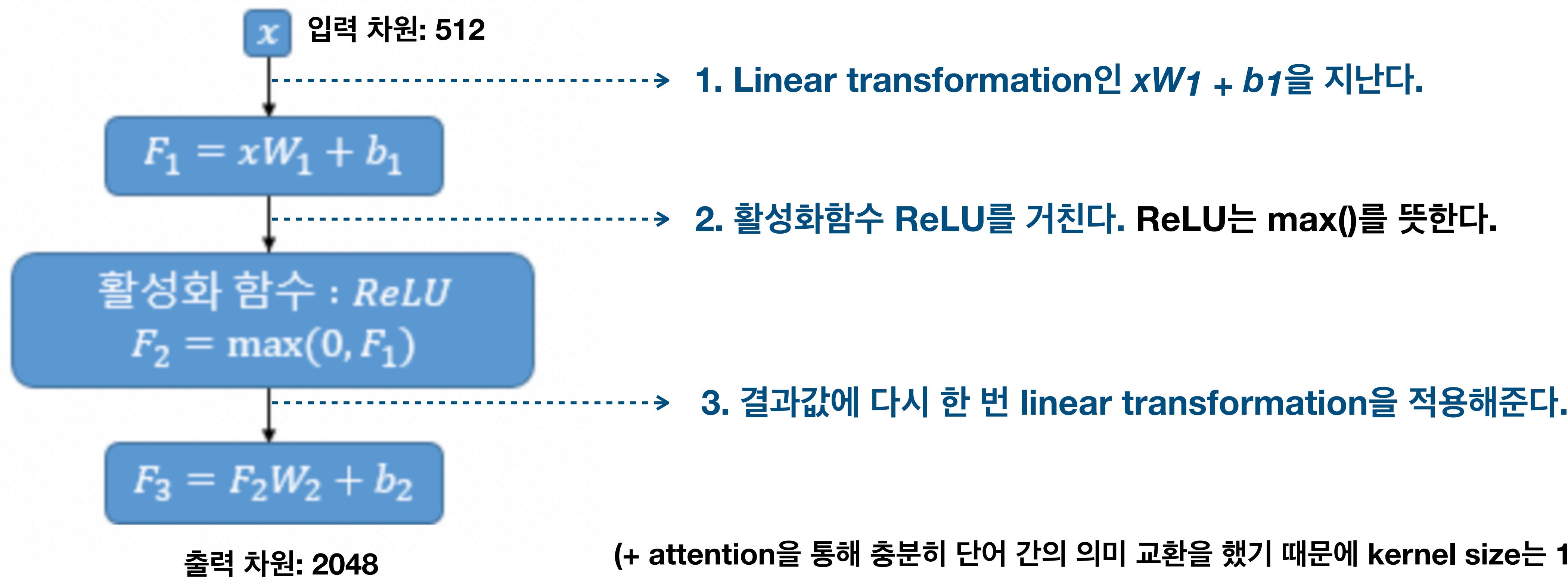
-> 각 head의 attention이 끝나면, 모든 결과값들은 하나의 벡터로 concatenate된다.

즉,  $Z_0 \sim Z_7$  은 하나의 벡터로 concatenate되고, 가중치 행렬  $W^O$ 와 곱해져 벡터 사이즈를 조정해준다.

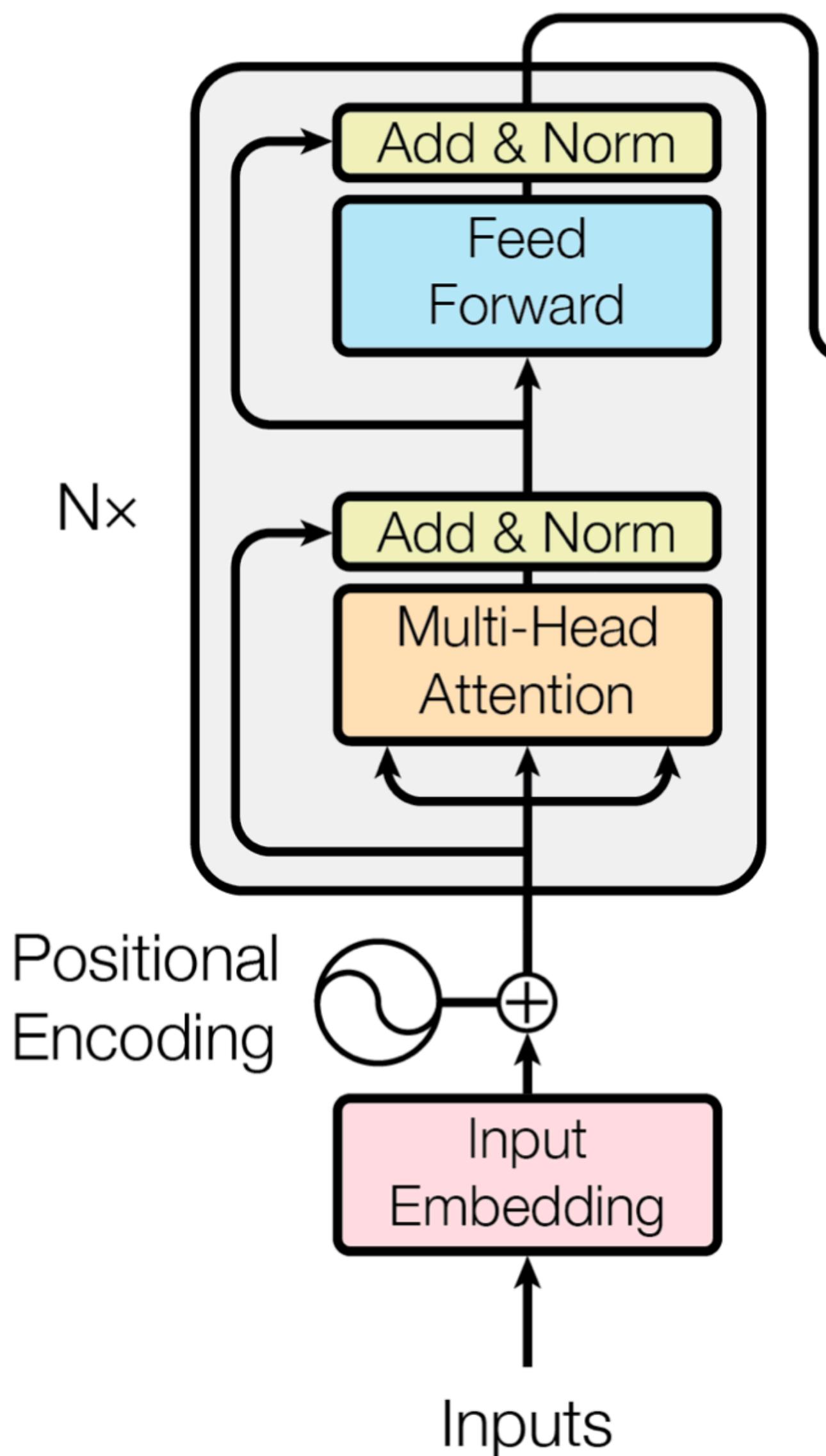
## 2. Position-wise feed forward network

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

파라미터  $W, b$ 는 모든 단어(position)마다 같은 값을 사용한다. 즉, 하나의 파라미터로 모든 단어들을 계산한다.  
같은 인코더 내에서는 같은 파라미터를 사용하지만, 인코더가 달라지면 다른 파라미터 값을 갖는다.

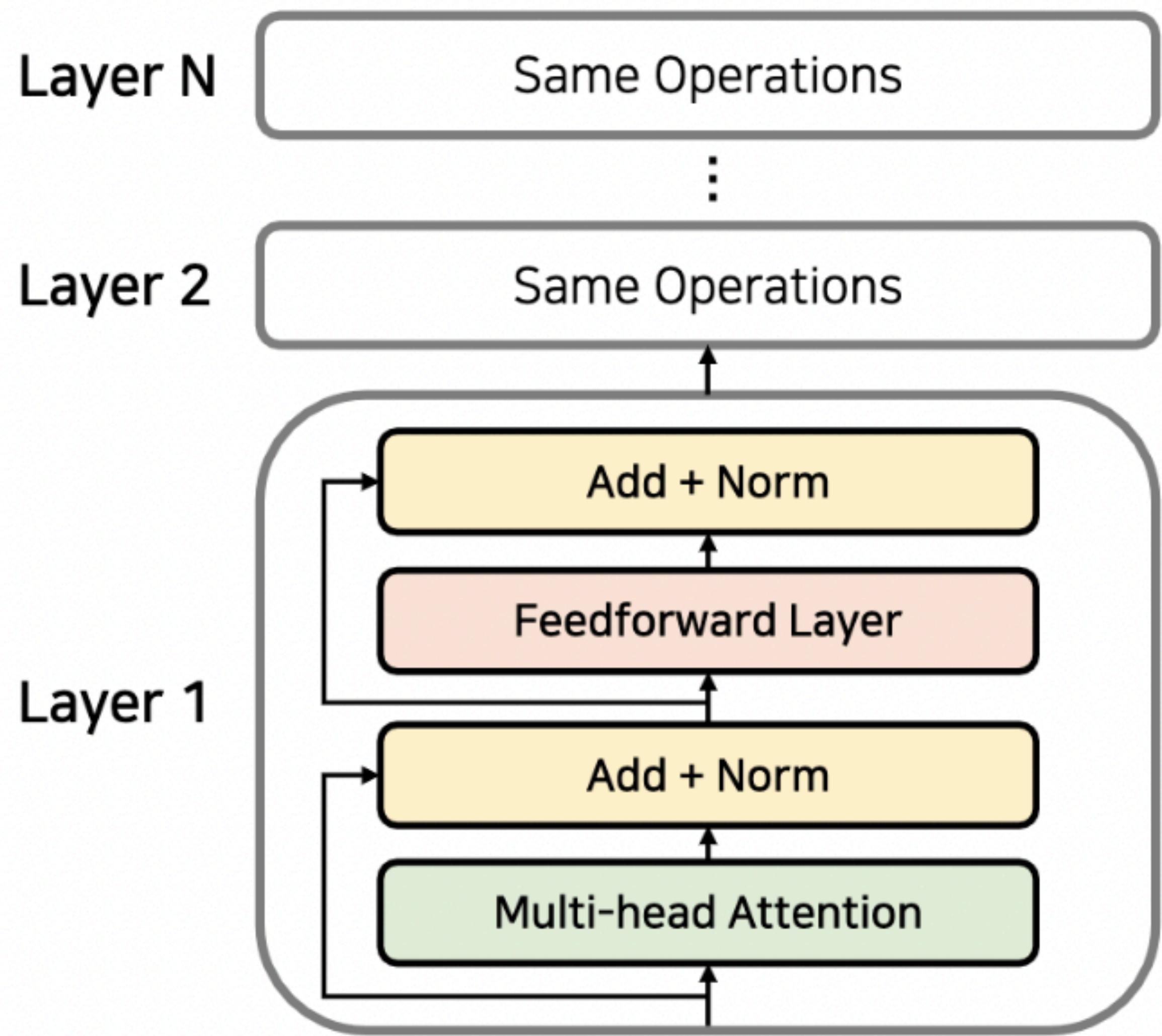


# Encoder



- ▶ 크게 2가지의 **sub-layers**가 존재
  1. **Multi-head self-attention**
  2. **Position-wise fully connected feed-forward network**
- ▶ **Residual connection**  
: 레이어를 건너뛰어 잔여된 부분만 학습시키는 것  
-> 전체 네트워크는 기존 정보를 입력받고, 추가적으로 잔여된 부분만 학습하도록 만듦으로써 전반적인 학습 난이도를 낮추고, 초기 모델 수렴 속도가 높혀 Gradient vanishing 위험도를 낮춘다. 이는 학습 효율성을 높일 수 있는 좋은 방법이다.

# Encoder

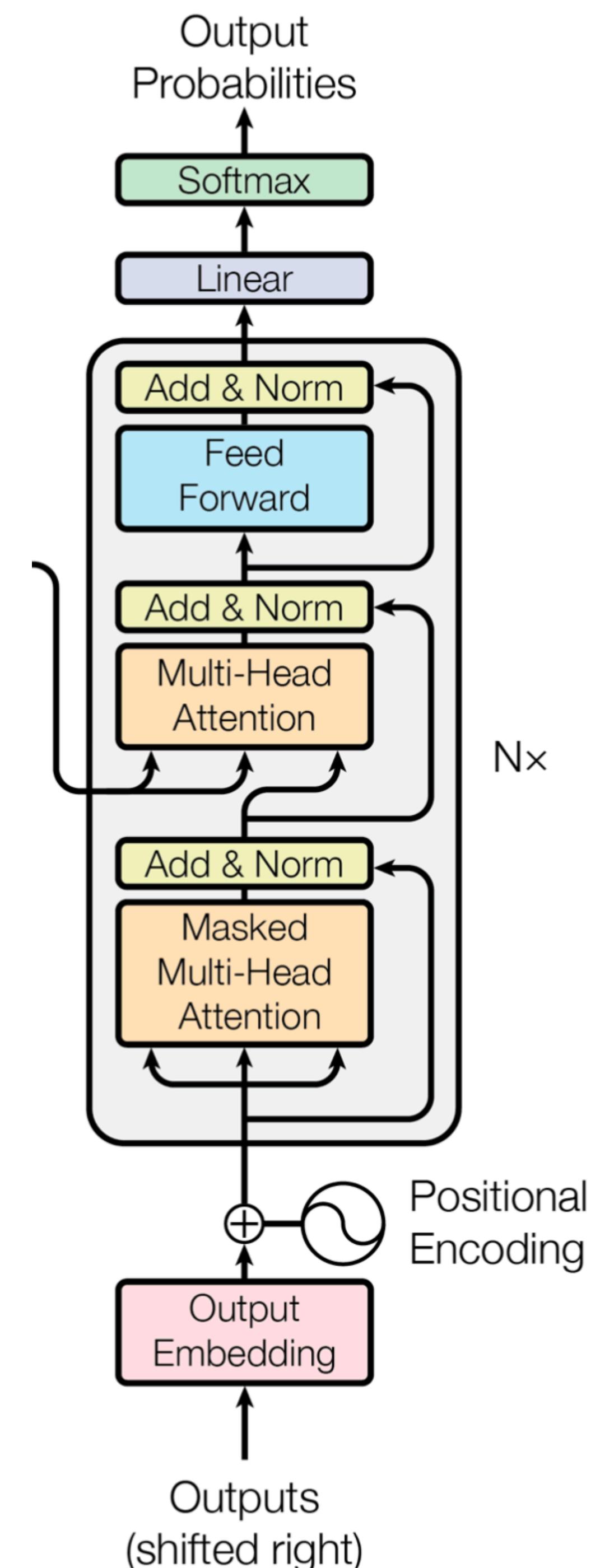


초기 임베딩된 단어들이 인코더로 들어오면,  
1. Multi-head Attention에 의해 attention이 가해진다.  
2. 정규화 단계를 수행한다. 이때 residual이 이용된다.  
3. Feed forward 과정을 거친다.  
4. 정규화 단계를 수행한다. 이때 residual이 이용된다.  
이 과정을 N번 반복한다.

- 어텐션(Attention)과 정규화(Normalization)과정을 반복한다.
- 각 레이어는 서로 다른 파라미터를 가진다.
- 입력되는 값과 출력되는 값의 dimension은 동일하다.

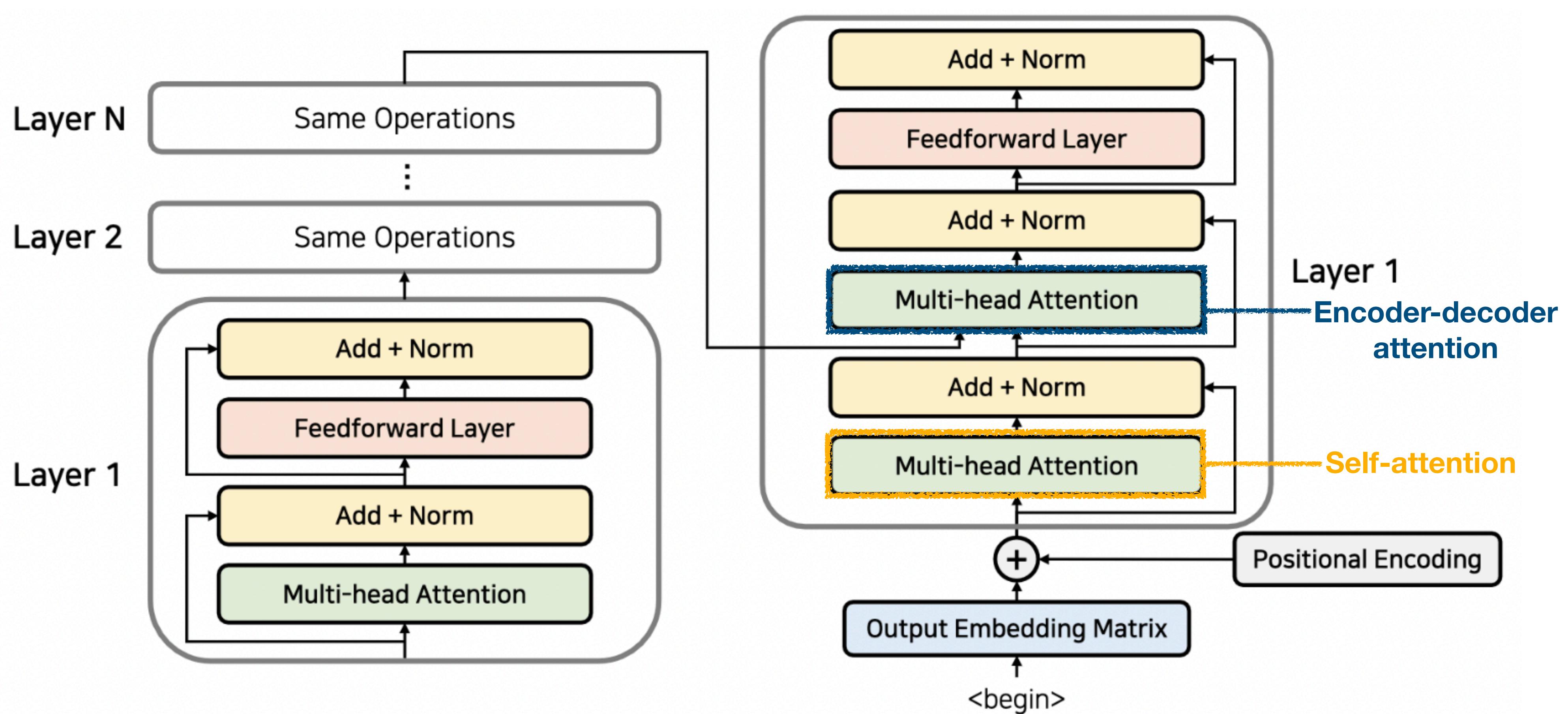
# Decoder

- ▶ 크게 3가지의 sub-layers가 존재
  1. **Masked Multi-head self-attention**
  2. **Multi-head self-attention**
  3. **Position-wise fully connected feed-forward network**
- ▶ **Masked Multi-head self-attention**  
: 앞에 나왔던 단어들만 참고해서 연산하는 attention layer이다.  
이를 위해서는 현재의 단어보다 뒤쪽 단어들에 대한 masking 작업이 필요하다.  
참고하지 않을 부분을  $-\infty$ 로 할당해줌으로써 softmax의 output이 0으로 수렴하게 만든다.



❖ 입력되는 값과 출력되는 값의 dimension을 동일하게 만들어서 각각의 디코더 레이어를 여러 번 중첩해서 사용할 수 있다.

# Decoder



입력값이 들어온 후, 여러 개의 인코더 레이어를 반복해서 가장 마지막 인코더 레이어에서 나온 출력값이 디코더 레이어로 들어간다.

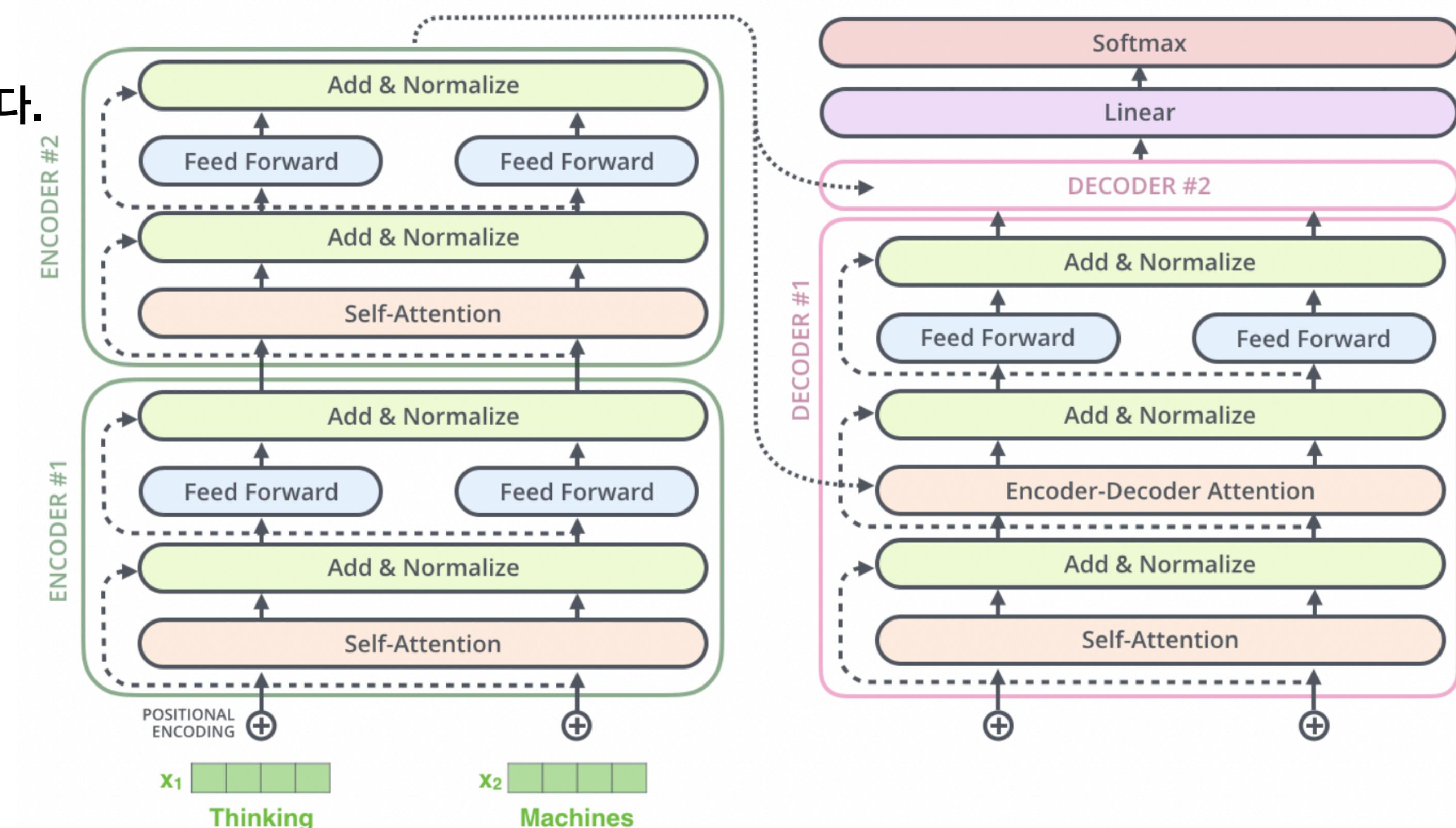
하나의 디코더 레이어에서는 두 개의 attention을 사용한다. 첫번째 attention은 self-attention으로, 인코더 파트와 마찬가지로 각각의 단어들이 서로에게 어떠한 가중치를 갖는지를 구하는지 만들어, 출력되고 있는 문장에 대한 전반적인 표현을 학습한다.

두번째 attention인 encoder-decoder attention은 인코더에 대한 정보를 어텐션할 수 있도록 만든다.  
즉, 인코더의 출력정보를 받아와 사용하는 것이다.

# Decoder

✓ Decoder는 Encoder의 결과값을 받아 단어 예측을 시작한다.

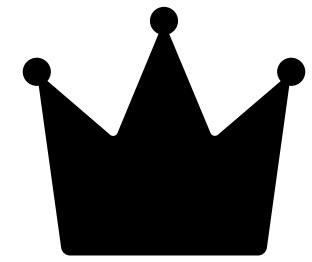
1. 초기 <start>토큰은 masked self-attention 과정을 거친다.
  2. 정규화 단계를 수행한다. 이때 residual이 이용된다.
  3. 입력받은 encoder의 결과값과 함께 encoder-decoder attention을 거친다.
  4. 정규화 단계를 수행한다. 이때 residual이 이용된다.
  5. Feed forward 과정을 거친다.
  6. 정규화 단계를 수행한다. 이때 residual이 이용된다.
- 이 과정을 총 N번 반복한다.
7. 이후 Fully-connected layer와 softmax를 통해 단어를 예측한다.



# Conclusion

**“Attention is all you need.”**

기존의 RNN계열의 모델을 탈피하고자 병렬 처리가 가능한 Transformer가 제안되었다.  
Transformer는 병렬 처리가 가능하기 때문에 빠르고 정확하게 텍스트를 처리할 수 있다.



**Thank you**