# Transformer

# Attention is All you need

- 기계 번역을 위해 탄생
- 인코더-디코더 구조를 여전히 유지.

- RNN의 경우, 각 step 별로 연산하므로 병렬 연산 불가.
- 많은 어텐션 메커니즘이 RNN과 함께 사용됨.

- **이 모델은 병렬 연산을 추구하며 RNN을 사용 안함!!!!!!**

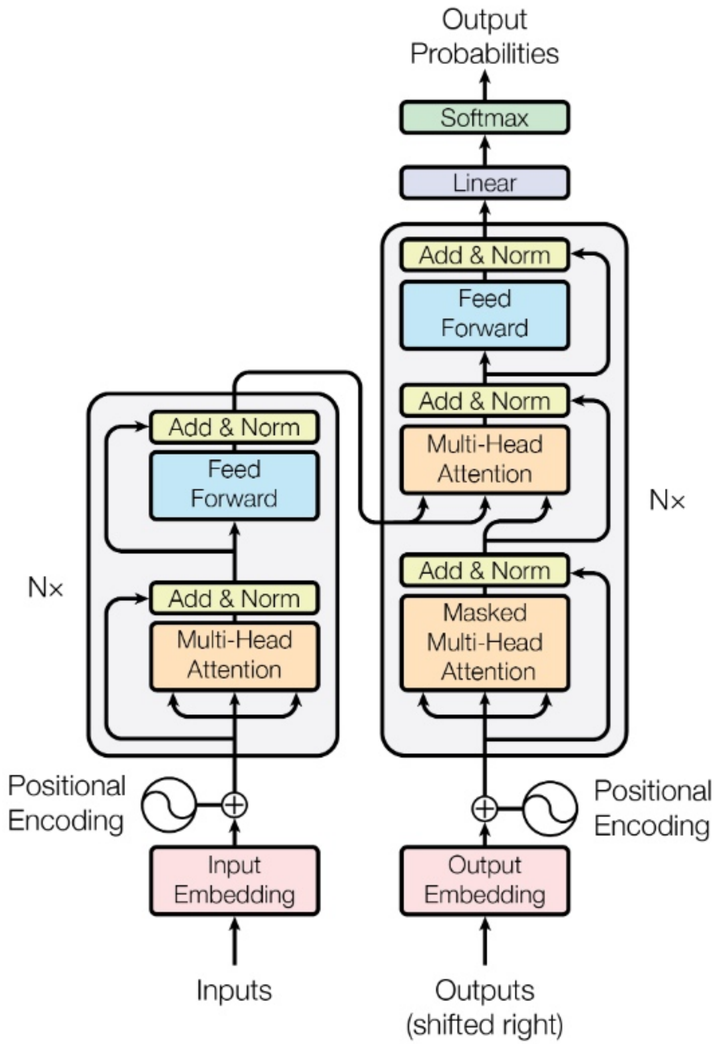


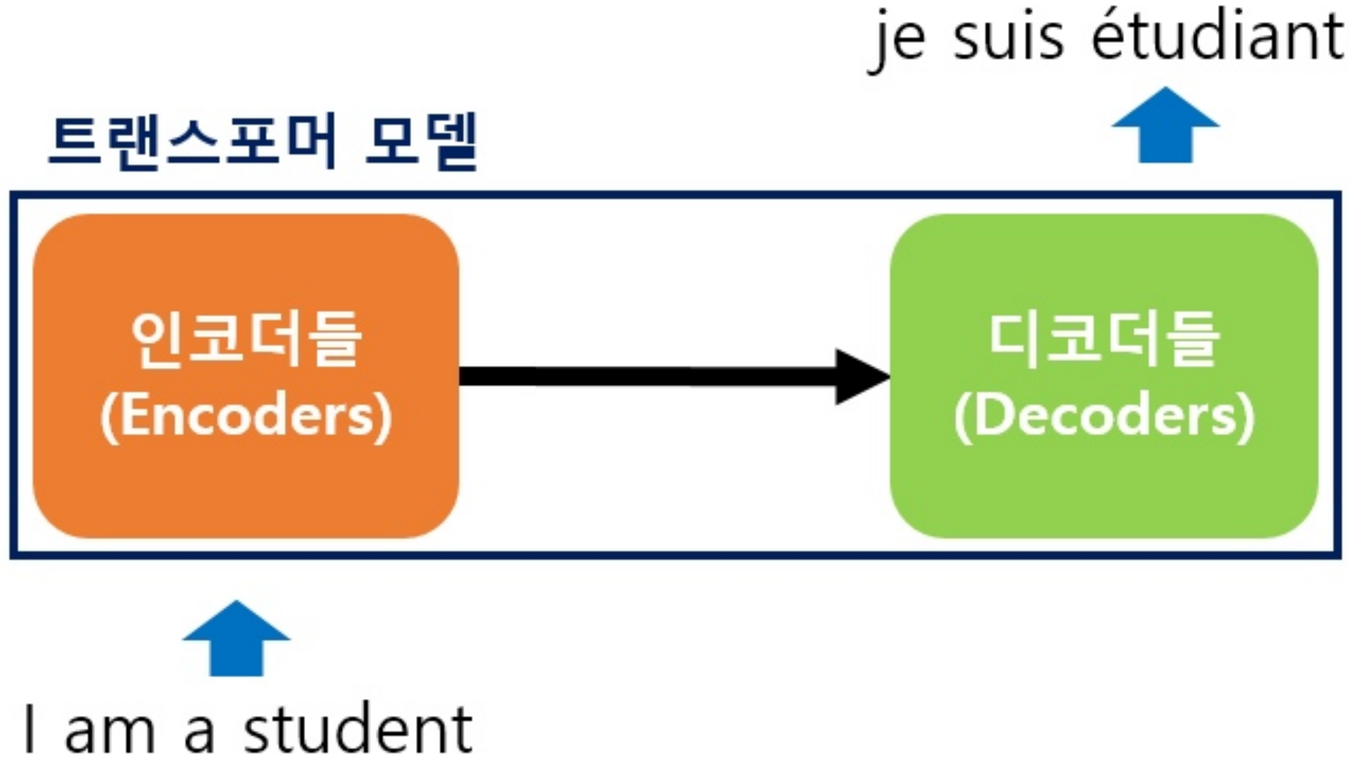

Figure 1: The Transformer - model architecture.

# Transformer

- 트랜스포머는 기본적으로 기계 번역을 위해 제안된 모델.
- 번역하고자 하는 문장을 입력하면, 번역 문장이 출력.

# Transformer

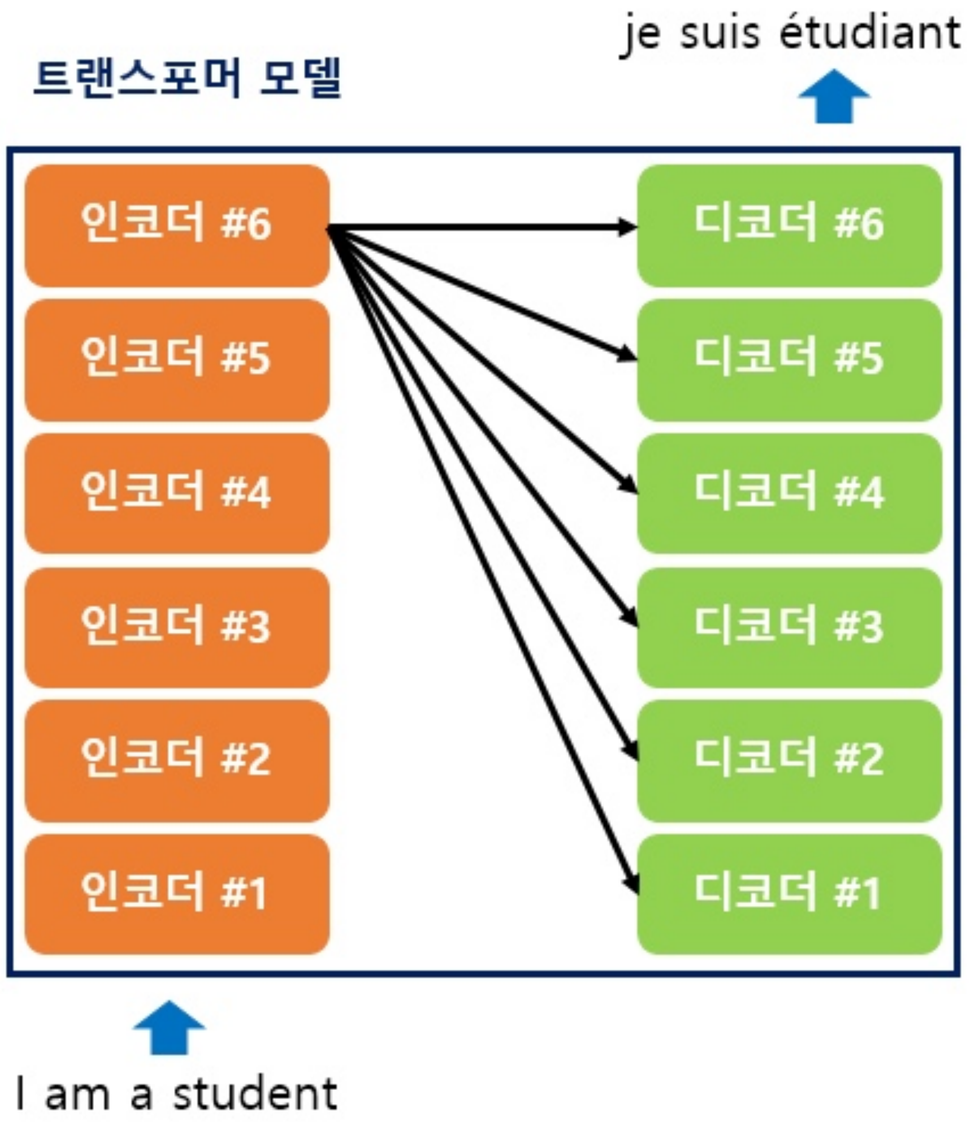
- 트랜스포머는 기본적으로 기계 번역을 위해 제안된 모델.
- 번역하고자 하는 문장을 입력하면, 번역 문장이 출력
- 인코더-디코더 구조



# Transformer

- 트랜스포머는 기본적으로 기계 번역을 위해 제안된 모델.
- 번역하고자 하는 문장을 입력하면, 번역 문장이 출력
- 인코더-디코더 구조

- 인코더-디코더 블록이 N개 존재
- 논문의 경우 각 6개.

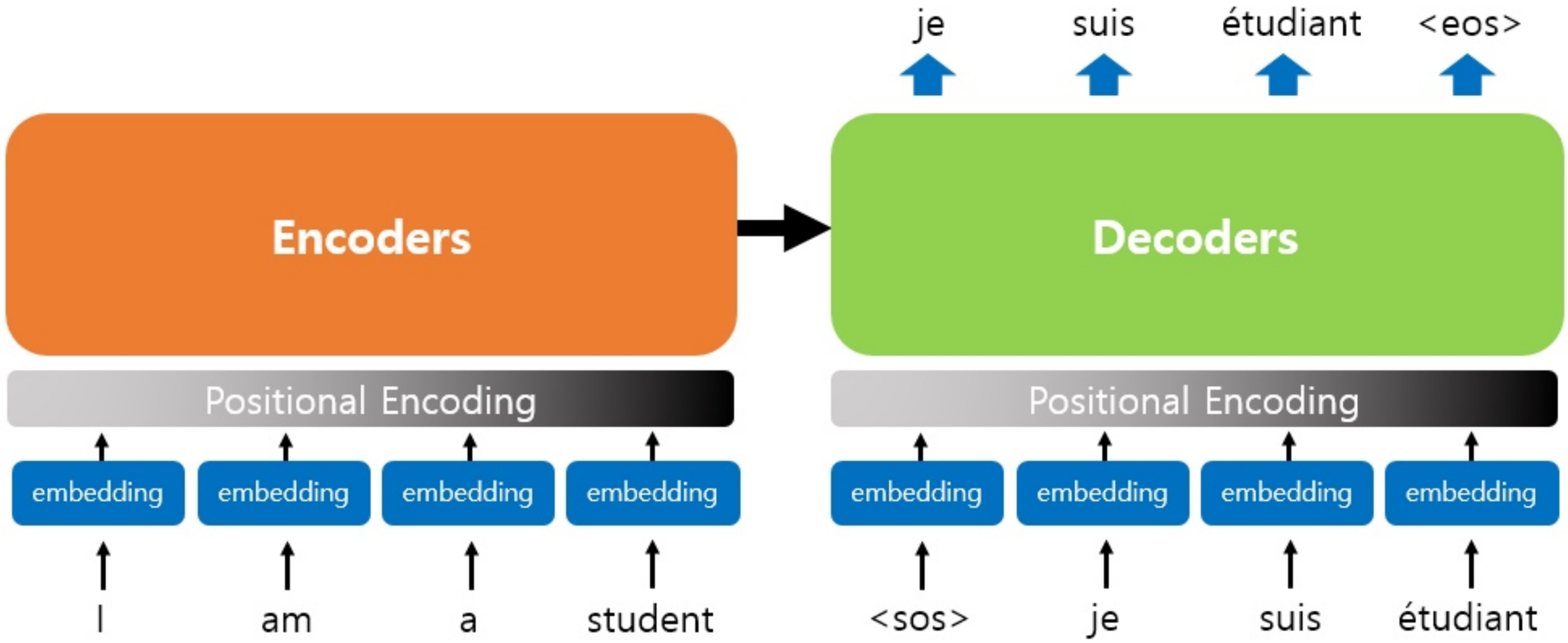


# Transformer

트랜스포머도 다른 딥 러닝 모델과 마찬가지로 **Embedding layer**를 사용

Embedding layer를 통해서 얻은 임베딩 벡터를 인코더와 디코더의 입력으로 한다.

임베딩 벡터에 **Positional Encoding**이라는 과정을 거친 후에 입력으로 한다.

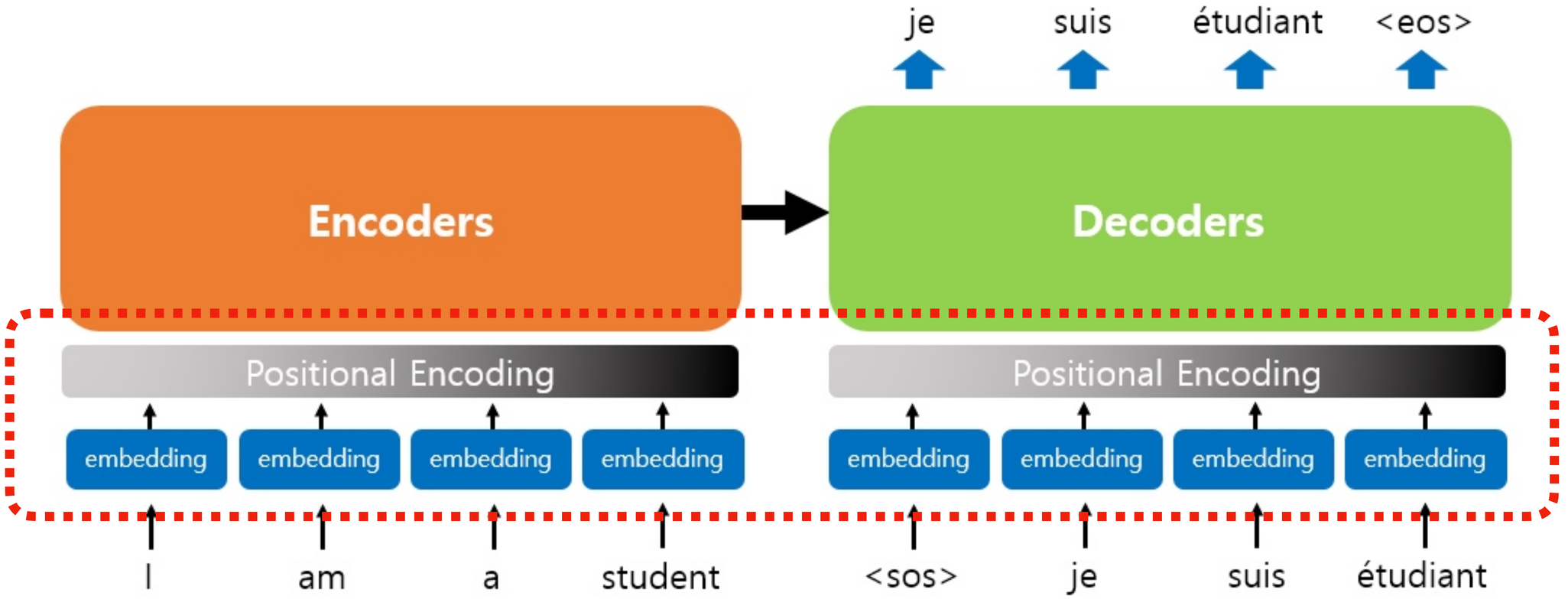


# Transformer

트랜스포머도 다른 딥 러닝 모델과 마찬가지로 **Embedding layer**를 사용

Embedding layer를 통해서 얻은 임베딩 벡터를 인코더와 디코더의 입력으로 한다.

임베딩 벡터에 **Positional Encoding**이라는 과정을 거친 후에 입력으로 한다.



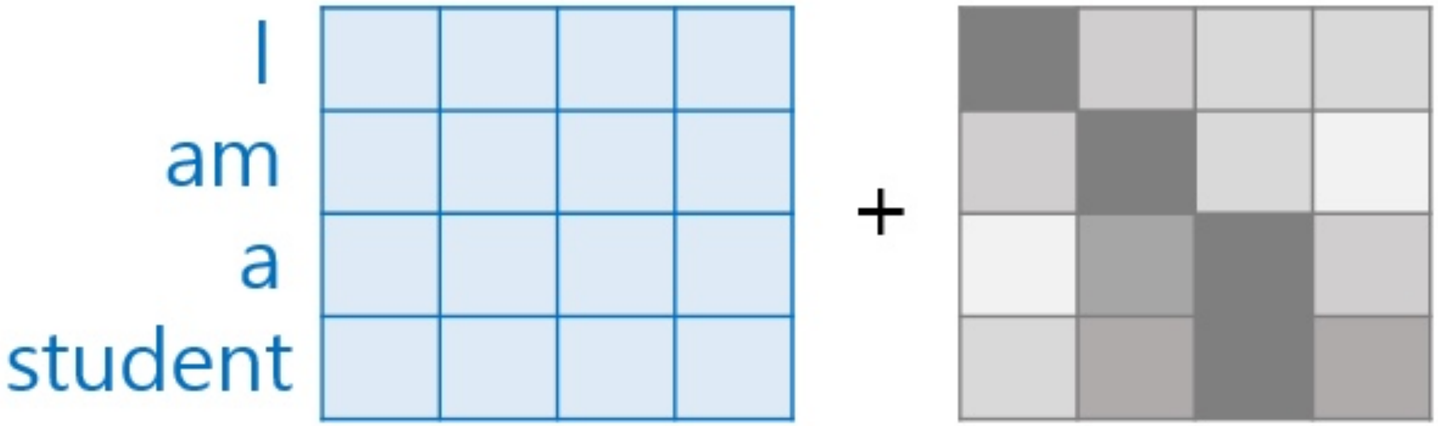
# Transformer

트랜스포머도 다른 딥 러닝 모델과 마찬가지로 Embedding layer를 사용

Embedding layer를 통해서 얻은 임베딩 벡터를 인코더와 디코더의 입력으로 한다.

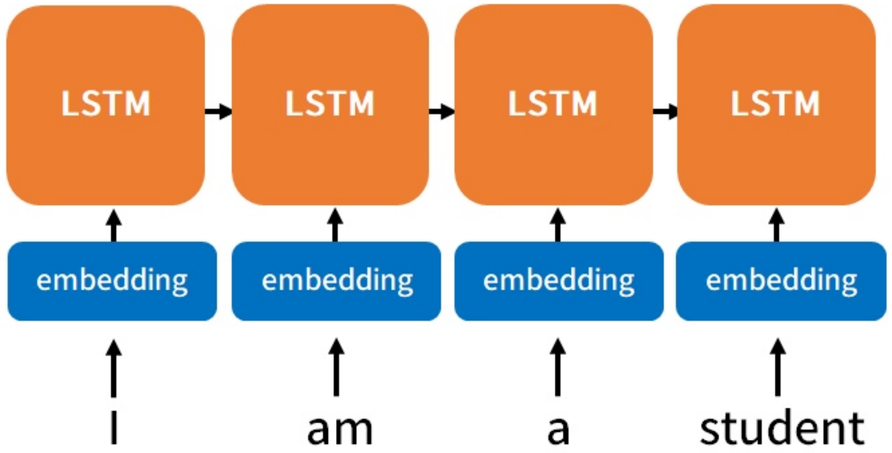
임베딩 벡터에 Positional Encoding이라는 과정을 거친 후에 입력으로 한다.

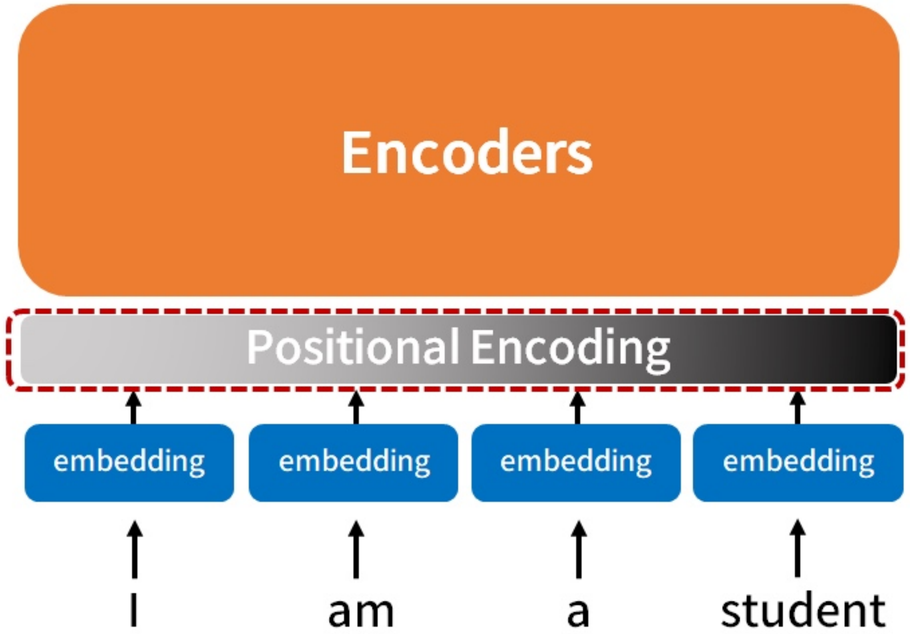
이를 행렬 연산으로 이해해본다면 다음과 같이 이해할 수 있다.



# Transformer: Positional Encoding

- RNN이 자연어 처리에서 유용했었던 이유는 단어 입력을 순차적으로 받으므로 각 단어의 위치 정보(position information)을 가질 수 있었기 때문
- 하지만 트랜스포머의 경우, 입력을 병렬로 받으므로 위치 정보를 더해줄 필요가 있음.



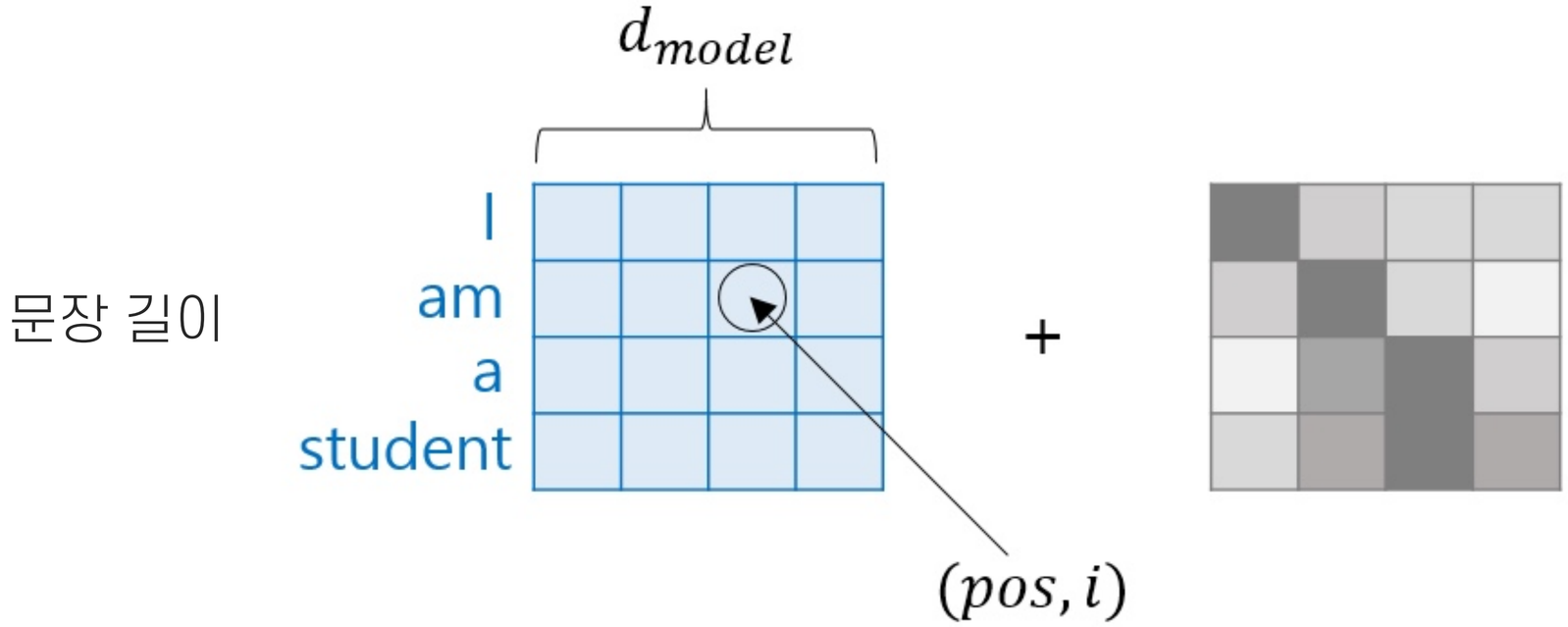



# Transformer: Positional Encoding

$pos$ 는 입력 문장에서의 임베딩 벡터의 위치

$i$ 는 임베딩 벡터의 차원

$d_{model}$ 은 트랜스포머의 입, 출력 차원. 또한 임베딩 벡터의 차원이기도 하다.



# Transformer: Positional Encoding

Positional Encoding 행렬을 만들기 위해서 트랜스포머는 아래와 같은 수식을 사용.

사인 함수와 코사인 함수의 그래프를 통해서 위치에 따라 다른 정보를 더한다.

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

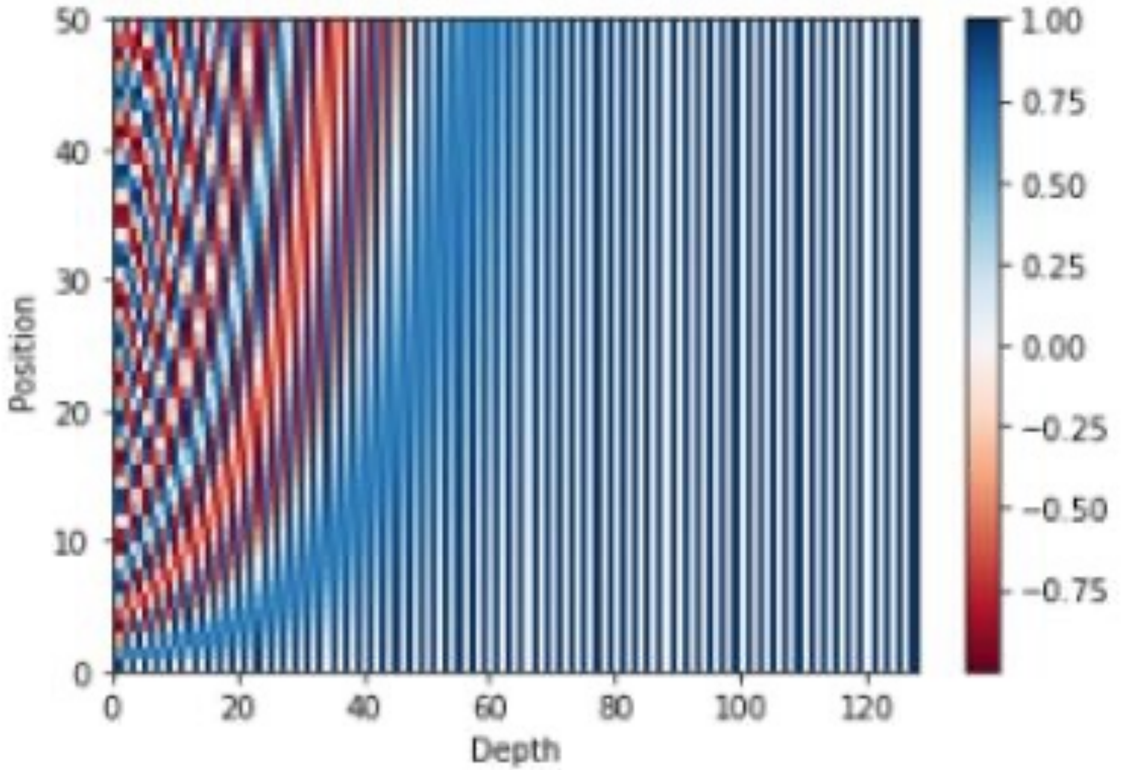
$$PE_{(pos,\ 2i+1)} = cos(pos/10000^{2i/d_{model}})$$

# Transformer: Positional Encoding

- 다음의 그림은 문장 길이 50, 임베딩 벡터의 차원이 128일 경우의 Positional Encoding 행렬

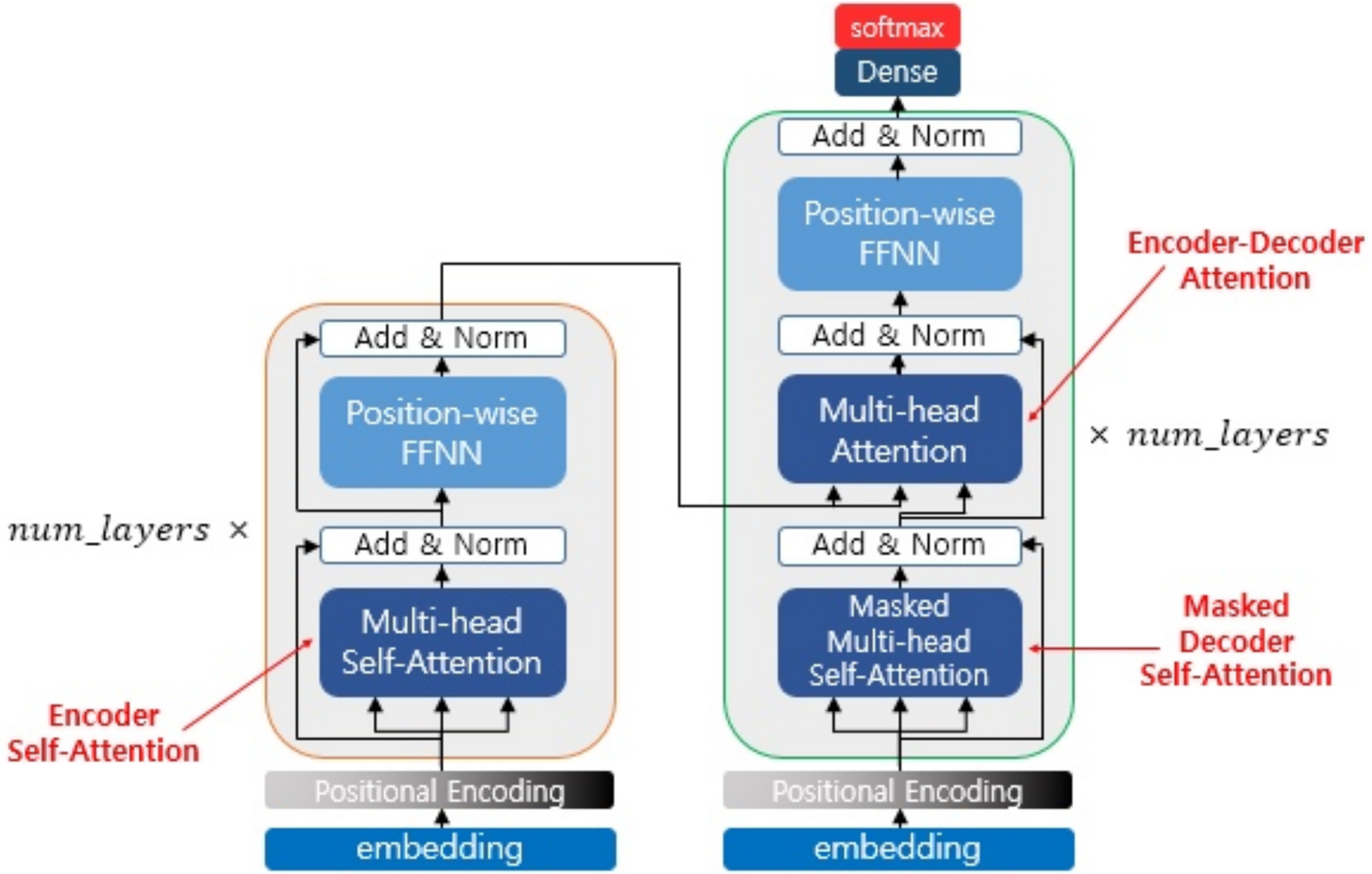
$d_{model}$

문장 길이

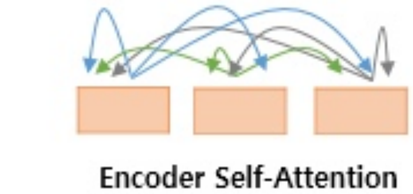


# Transformer: Attention Mechanism

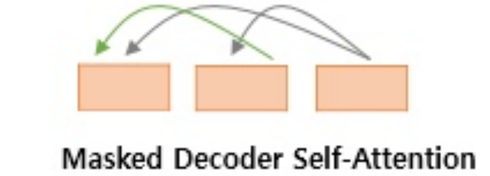
트랜스포머는 총 세 종류의 어텐션이 존재.



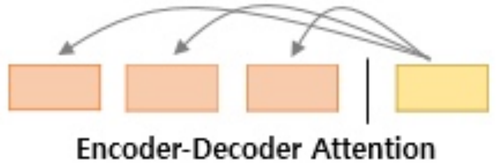

Q, K, V이 **같은** 곳에 있는 경우에는 Self-Attention이라고 부른다.




Q, K, V 가 같은 곳에




Q, K, V 가 같은 곳에

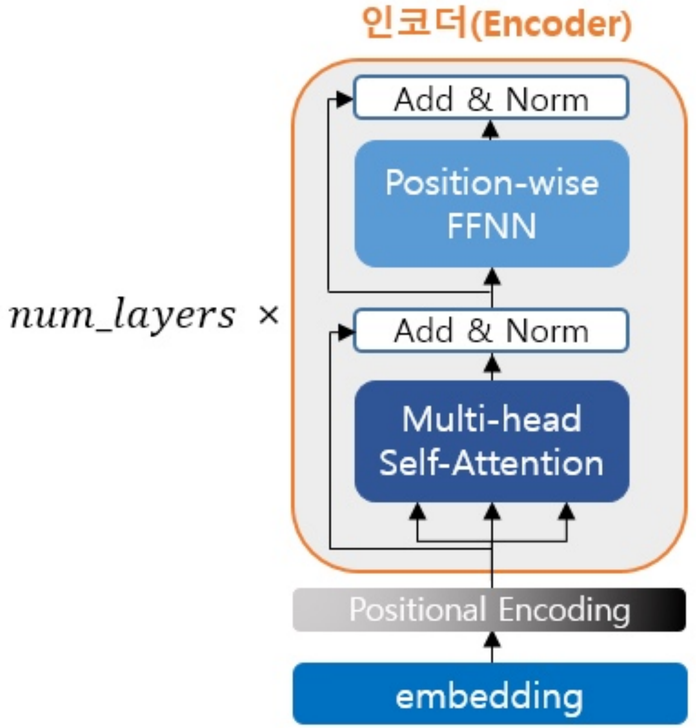



Q, K, V 가 다른 곳에

# Transformer Encoder

# Transformer Encoder

- 인코더를 1개의 층(layer)으로 생각하였을 때, 논문에서는 총 6개의 layer가 존재.
- num_layers= 6
- 인코더 내부에는 총 2개의 서브층이 존재.




**두번째 서브층(sublayer) : 포지션 와이즈 피드 포워드 신경망**

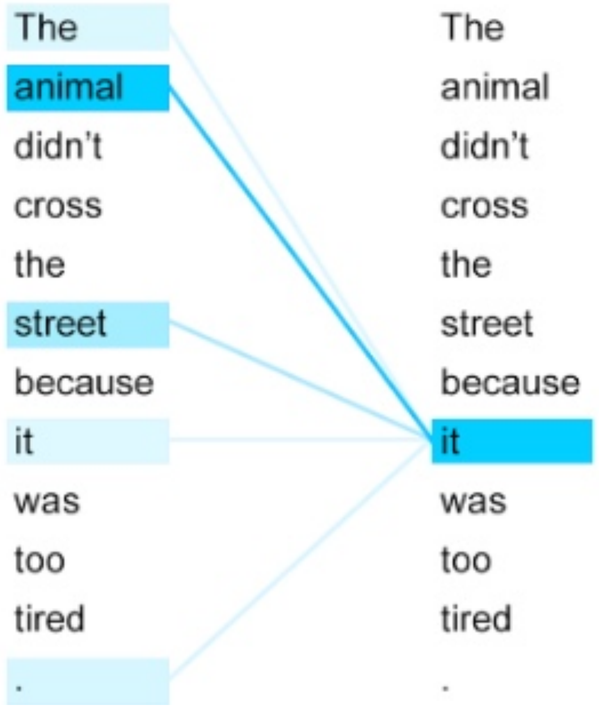
**첫번째 서브층(sublayer): 셀프 어텐션**

# Transformer Encoder: Self-Attention

**그 동물은 길을 건너지 않았다. 왜냐하면 그것은 너무 피곤하였기 때문이다.**

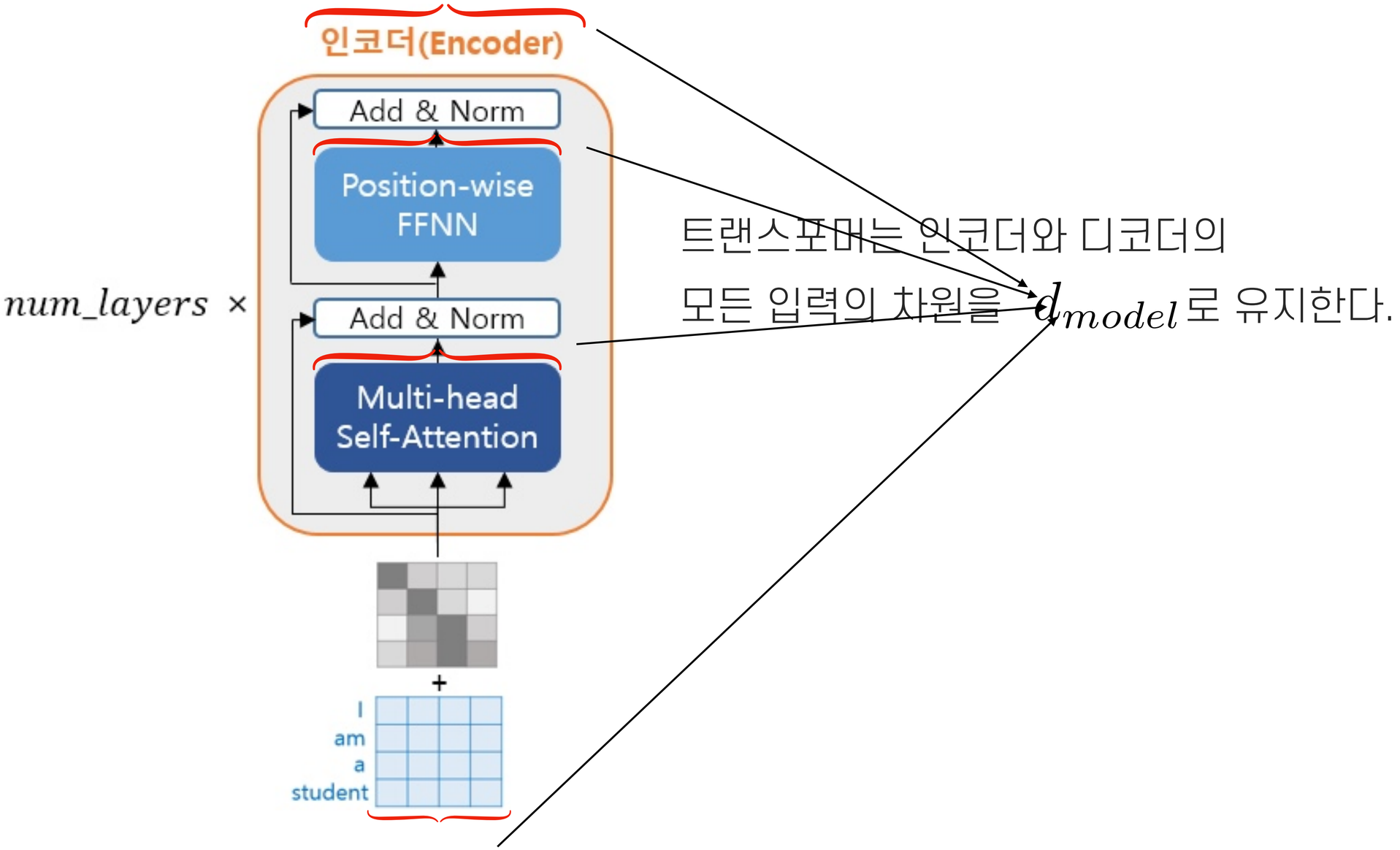
여기서 그것(it)에 해당하는 것은 과연 길(street)일까, 동물(animal)일까?

인간에게는 굉장히 쉬운 문제지만 기계에게는 그렇지 않음.




**셀프 어텐션은 입력 문장 내의 단어들끼리 유사도를 구하므로서 그것(it)이 동물(animal)과 연관되었을 확률이 높다는 것을 찾아낸다.**

# Transformer Encoder: Self-Attention

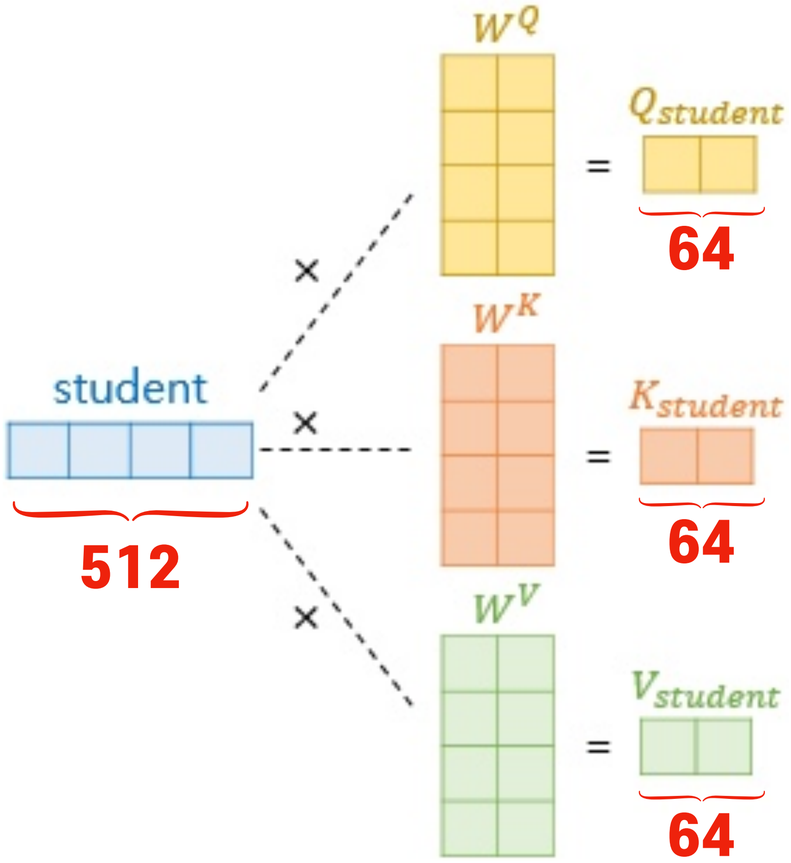


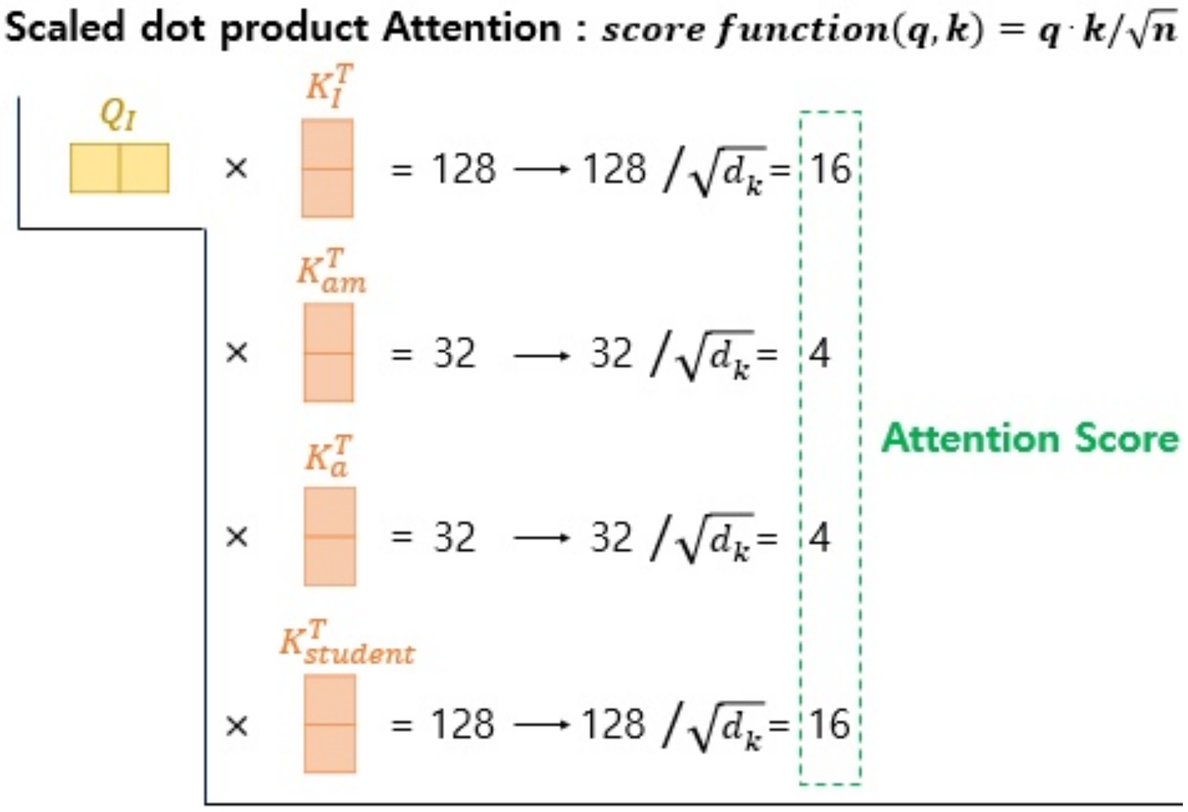

트랜스포머는 인코더와 디코더의 모든 입력의 차원을 $d_{model}$로 유지한다.

# Transformer Encoder: Self-Attention

셀프 어텐션은 입력 벡터에서 가중치 행렬 곱으로 Q, K, V 벡터를 얻고 수행.

Scale Dot Product 어텐션을 통해 각각의 Q벡터가 각각의 K벡터에 대해서 스코어 연산.

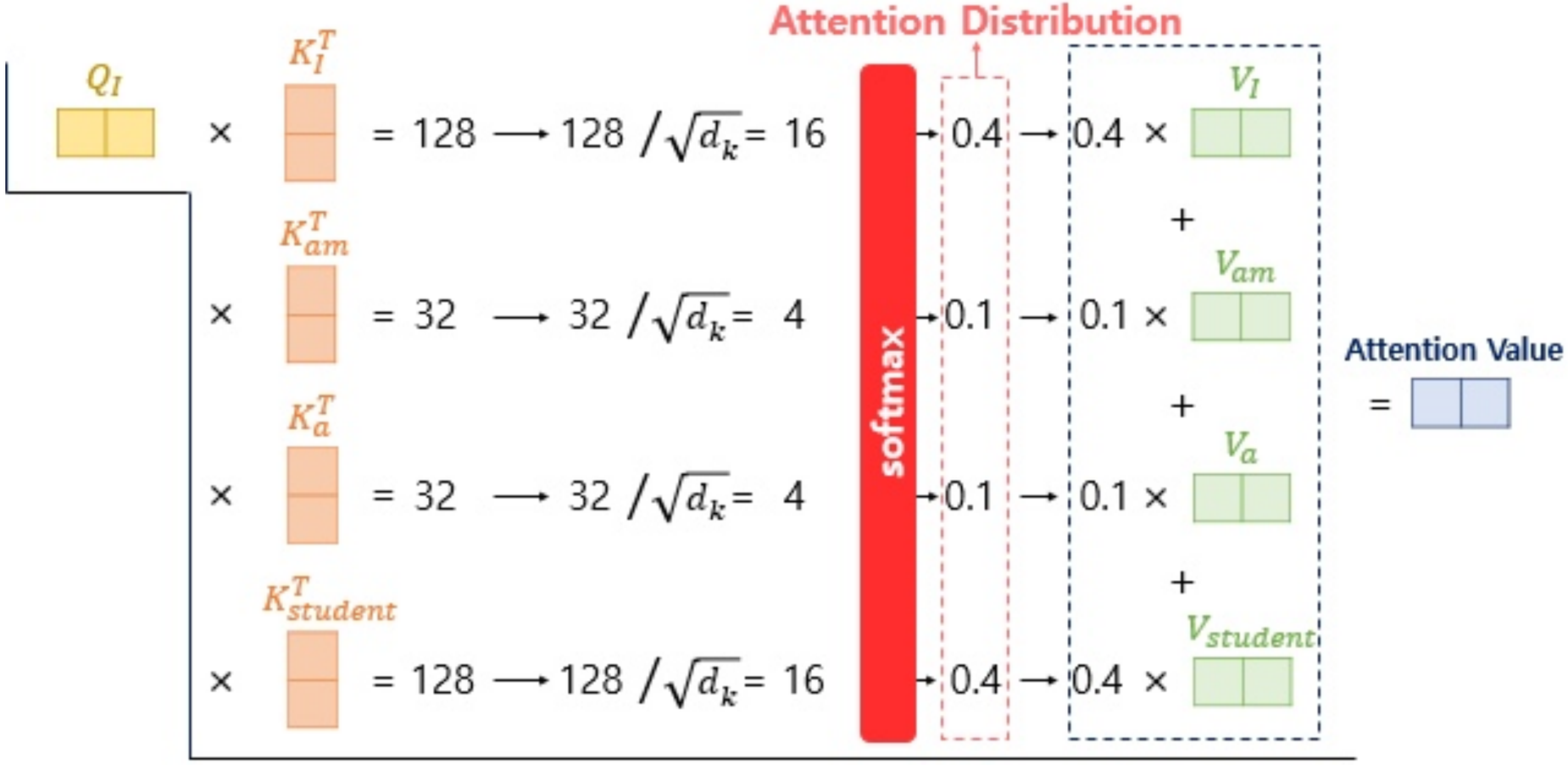


# Transformer Encoder: Self-Attention

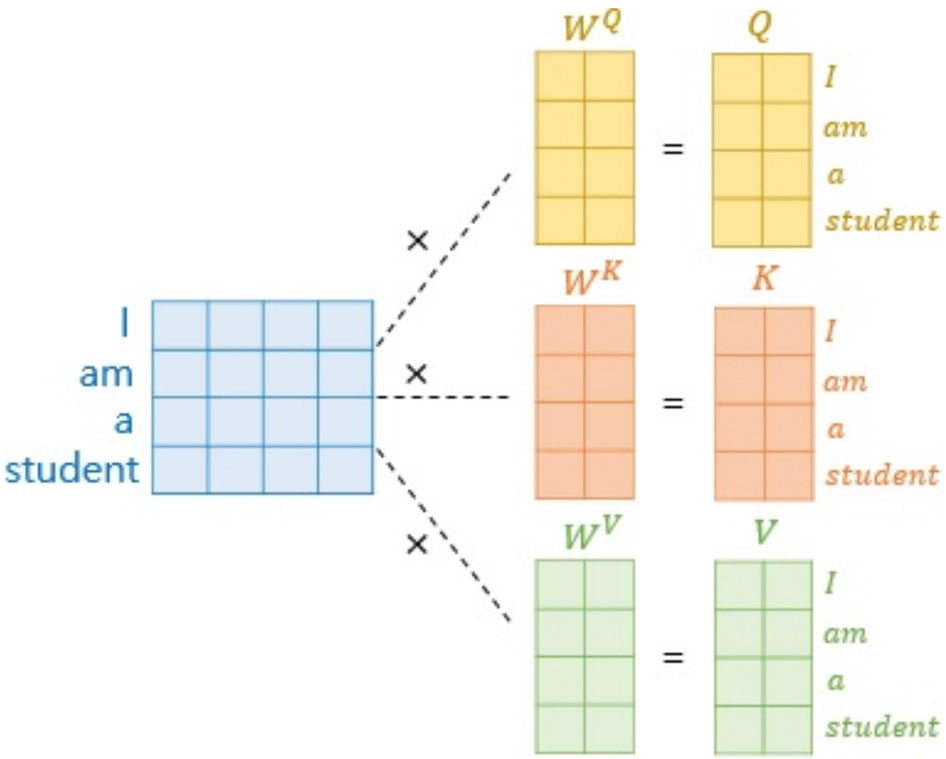
셀프 어텐션은 입력 벡터에서 가중치 행렬 곱으로 Q, K, V 벡터를 얻고 수행.

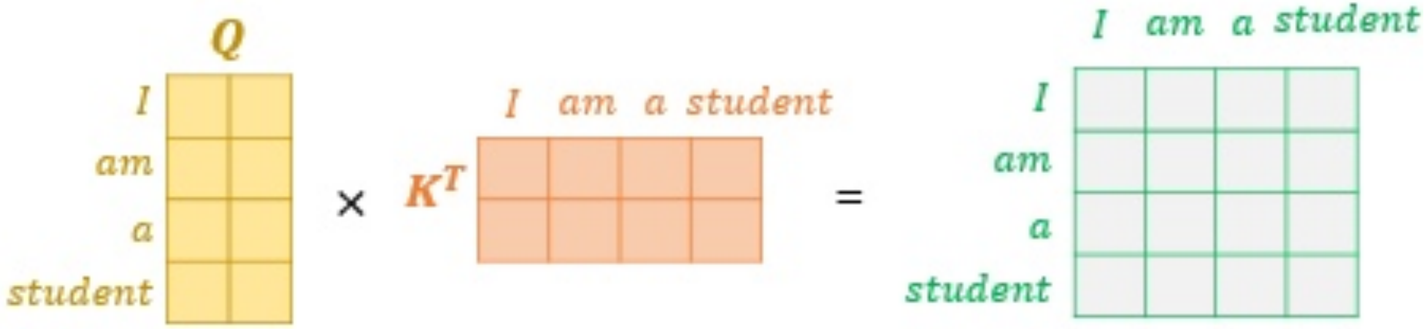
Scale Dot Product 어텐션을 통해 각각의 Q벡터가 각각의 K벡터에 대해서 스코어 연산.



# Transformer Encoder: Self-Attention

실제 연산 시에는 벡터 간 연산이 아니라 행렬 연산으로 한 번에 땅! 이루어진다.

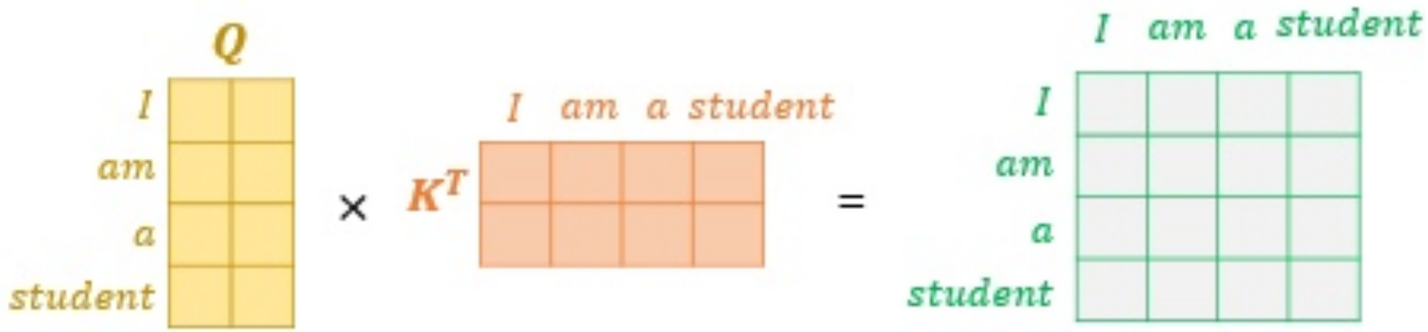


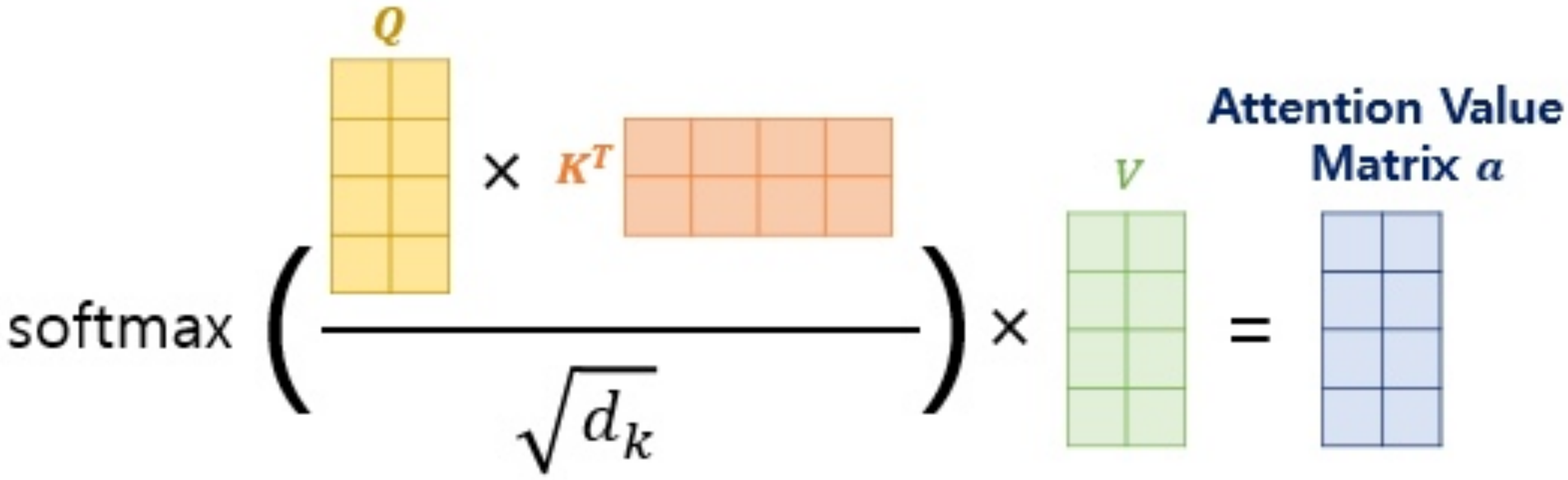




위 행렬에 특정 값을 나누어주면 **어텐션 스코어 행렬**

# Transformer Encoder: Self-Attention

실제 연산 시에는 벡터 간 연산이 아니라 행렬 연산으로 한 번에 땅! 이루어진다.



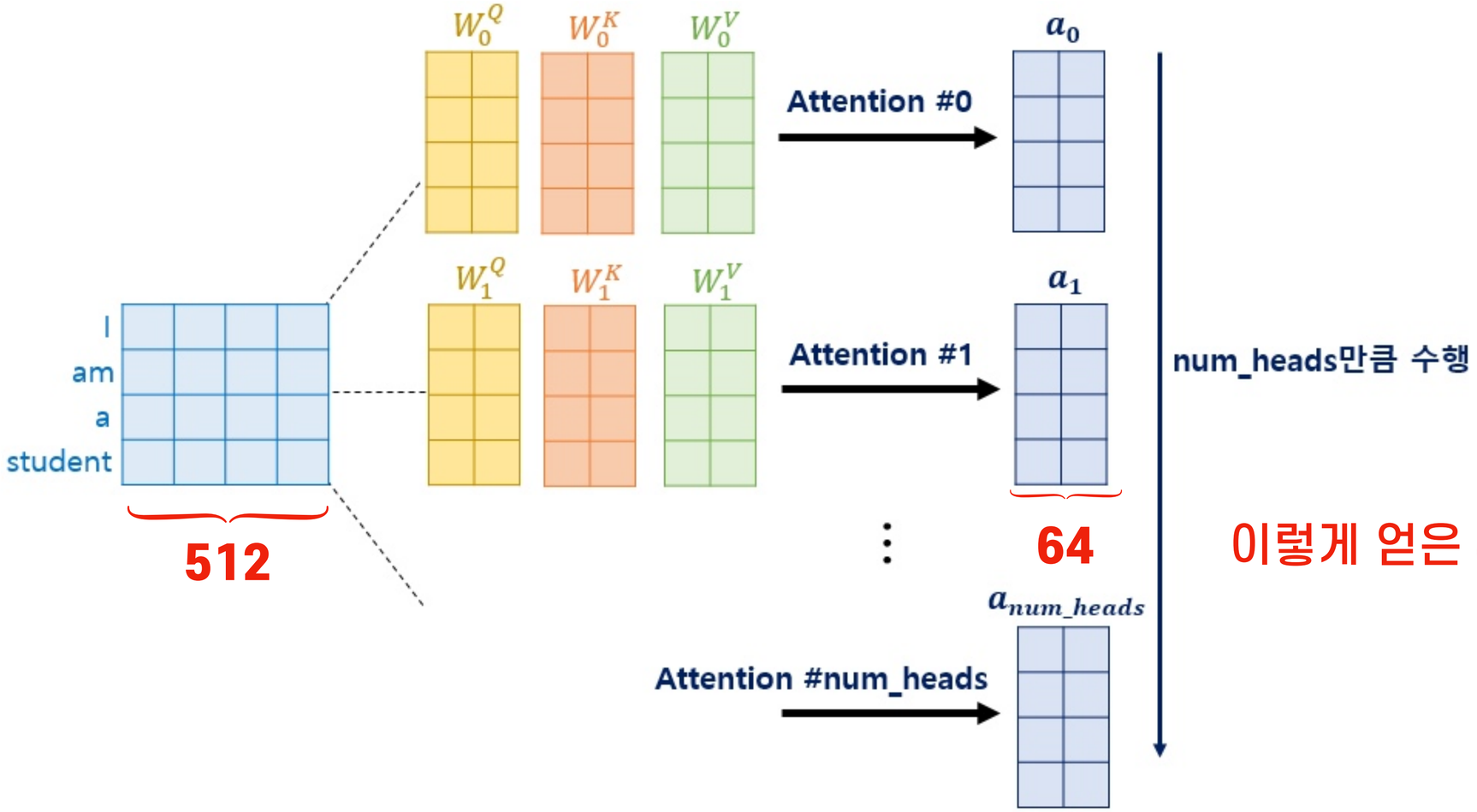




셀프 어텐션의 최종 결과

# Transformer Encoder: Self-Attention

트랜스포머는 어텐션에 대해서 병렬적으로 수행한다는 특징을 가지고 있다.

이를 집단 지성, 다수의 머리를 이용한다고 하여 Multi-Head Attention이라고 부른다.




$d_{model}$=512

num_heads = 8

$d_{model}$ / num_heads = 64

이렇게 얻은 8개의 결과를 다시 전부 연결한다

64*8 = ...

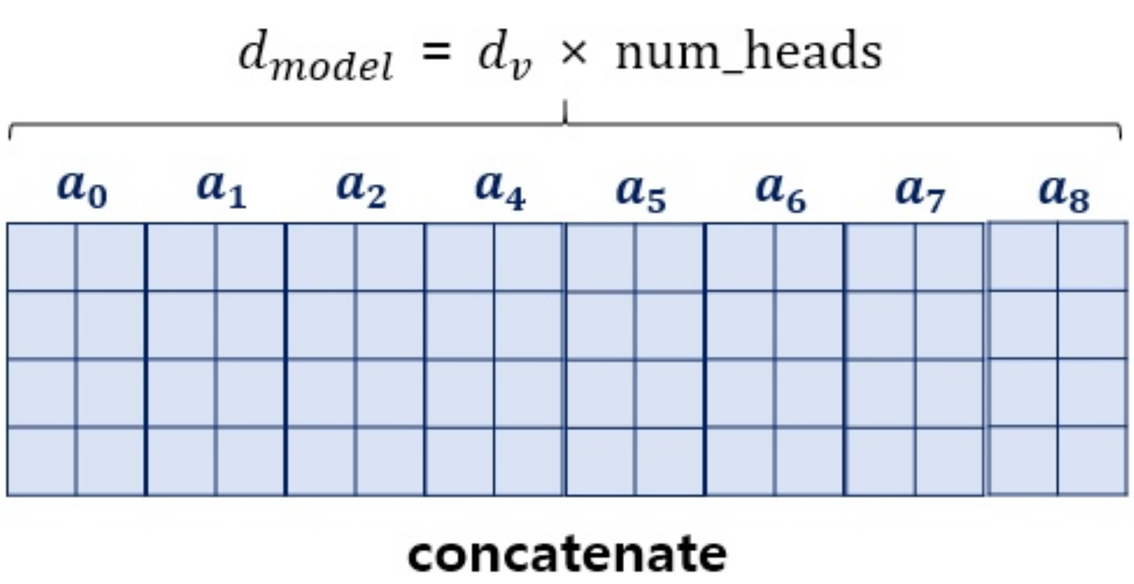
# Multi-head Attention

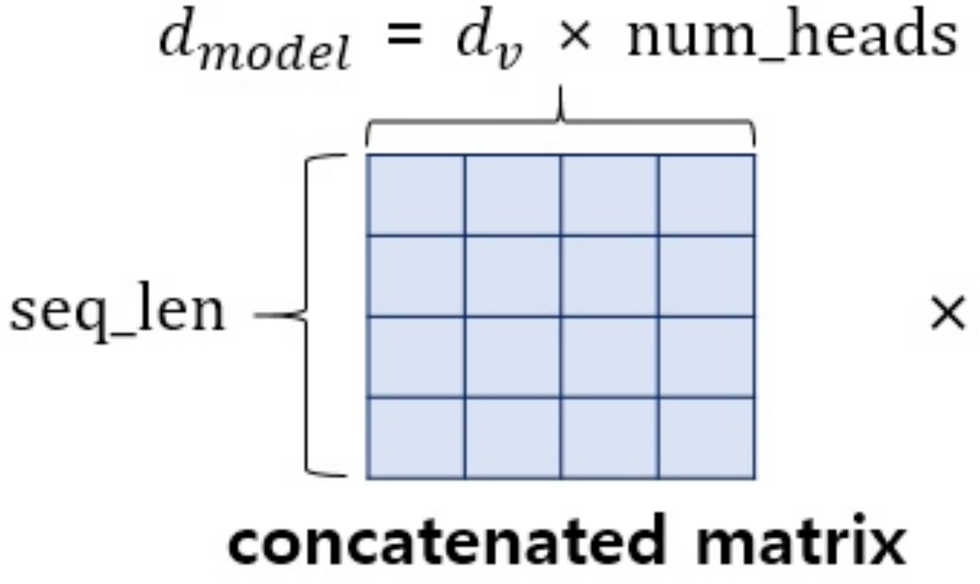
트랜스포머는 어텐션에 대해서 병렬적으로 수행한다는 특징을 가지고 있다.

이를 집단 지성, 다수의 머리를 이용한다고 하여 Multi-Head Attention이라고 부른다.

병렬적으로 수행한 어텐션은 마지막에 전부 연결한다.

**64 * 8 = 512**



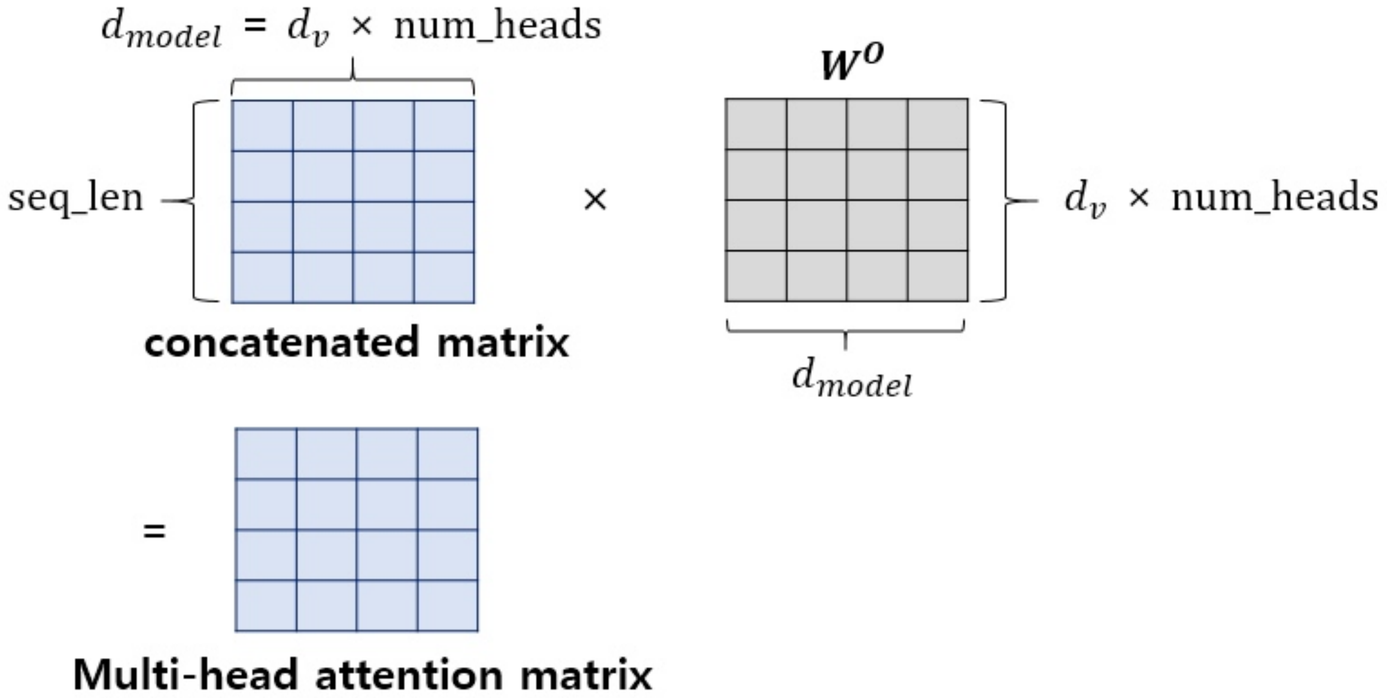
# Multi-head Attention

트랜스포머는 어텐션에 대해서 병렬적으로 수행한다는 특징을 가지고 있다.

이를 집단 지성, 다수의 머리를 이용한다고 하여 Multi-Head Attention이라고 부른다.

병렬적으로 수행한 어텐션은 마지막에 전부 연결한다.

연결한 행렬에 마지막으로 가중치 행렬을 한 번 더 곱해준다.



# Multi-head Attention

논문에서는 지금까지의 과정을 아래의 수식으로 정리한다.

Attention(Q, K, V)는 Attention Value를 얻는 어텐션 함수

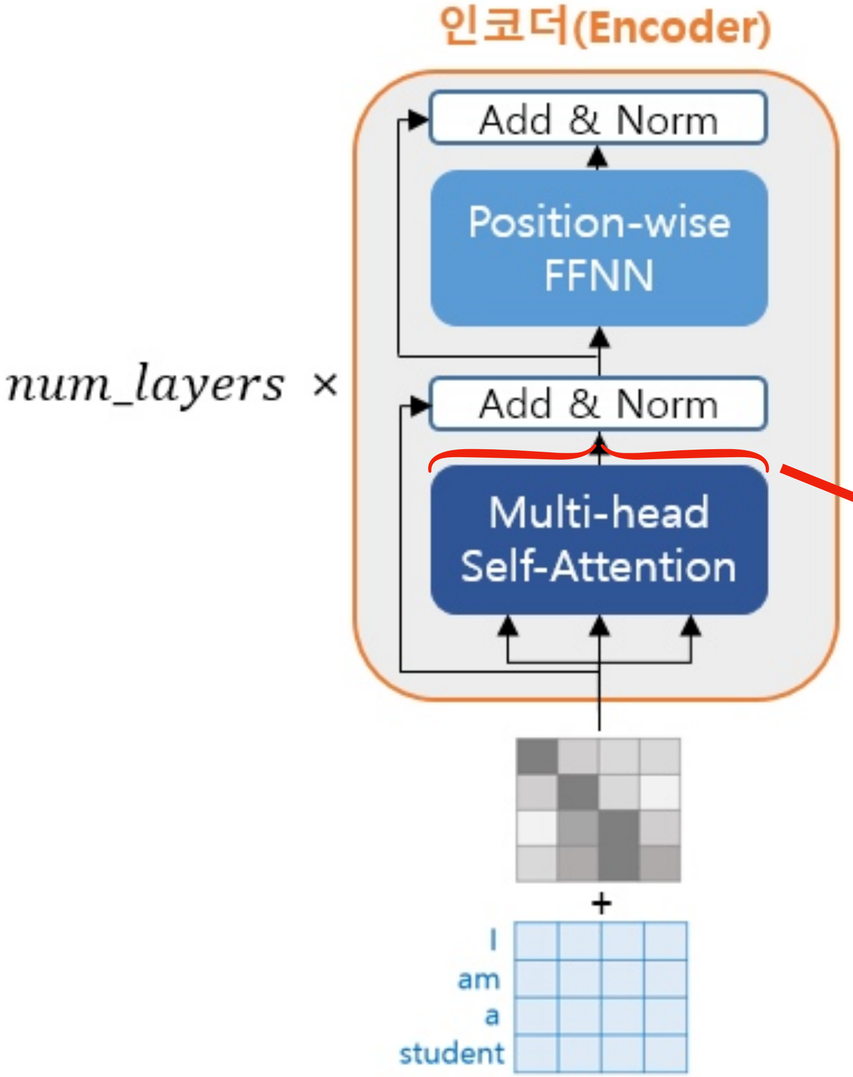
h는 head 개수,  i는 1번째 head를 의미한다.

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

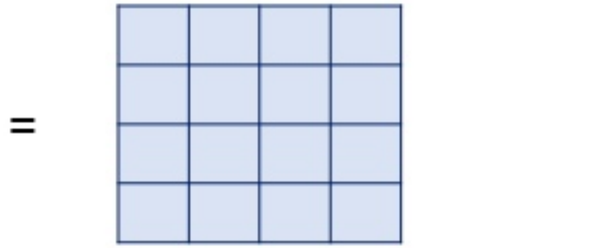
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

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

# Multi-head Attention

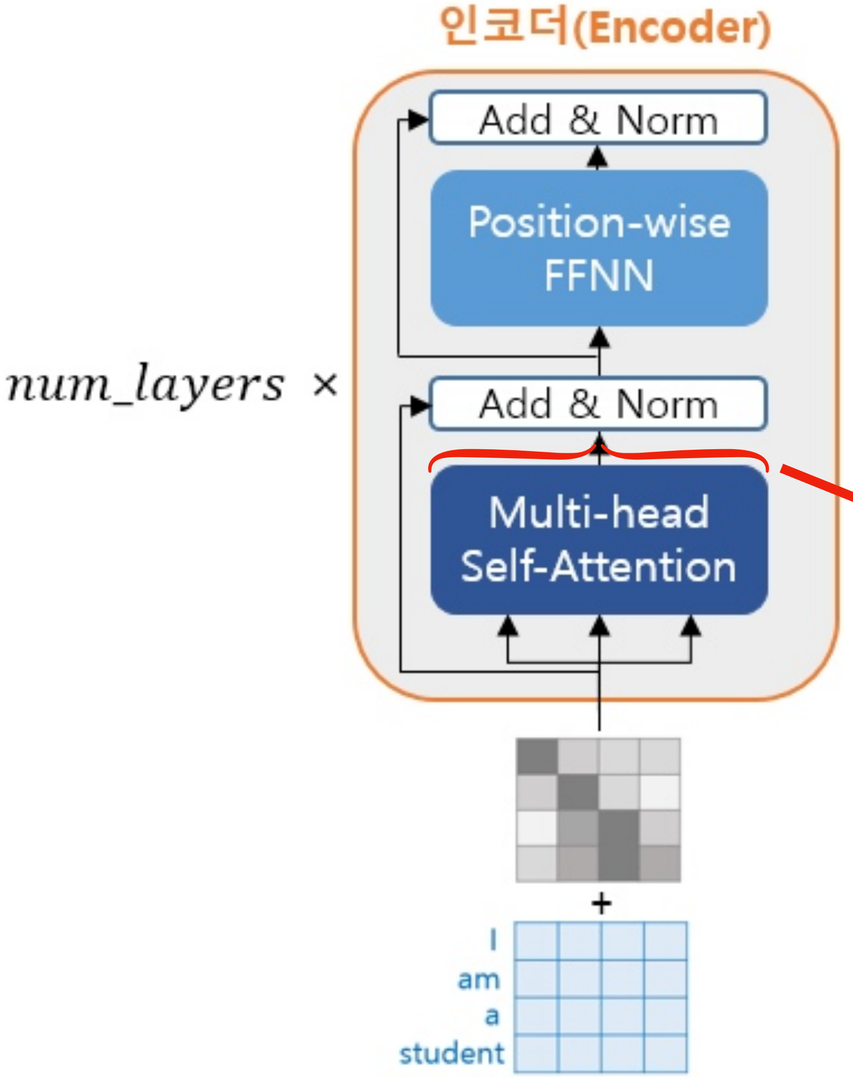



트랜스포머는 인코더와 디코더의
모든 입력의 차원을 $d_{model}$로 유지한다.

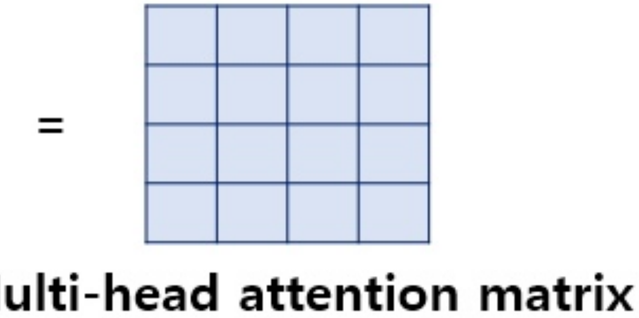



**Multi-head attention matrix**

# Multi-head Attention




트랜스포머는 인코더와 디코더의
모든 입력의 차원을 $d_{model}$로 유지한다.




**Multi-head attention matrix**

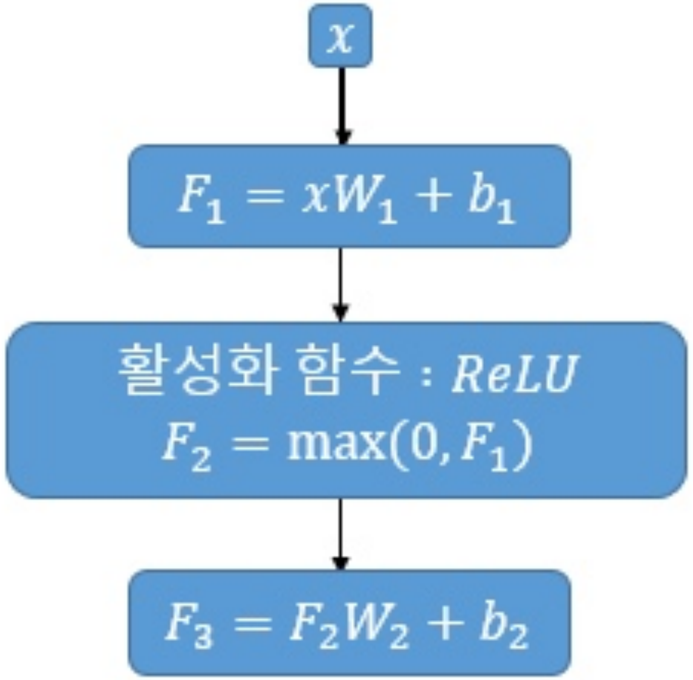
여기까지가 첫번째 서브층의 연산!
두번째 서브층 연산은 포지션 와이즈 피드 포워드 신경망.

# Position-wise Feed Forward Neural Network

포지션 와이즈 피드 포워드 신경망은 단순 피드 포워드 신경망이다.

은닉층에서는 활성화 함수로 ReLU 함수를 사용한다.

d_ff 는 피드 포워드 신경망의 은닉층의 크기로 논문에서는 2,048이 사용되었다.




$x = (\text{seq\_len}, d_{model})$

$W_1 = (d_{model}, d_{ff})$

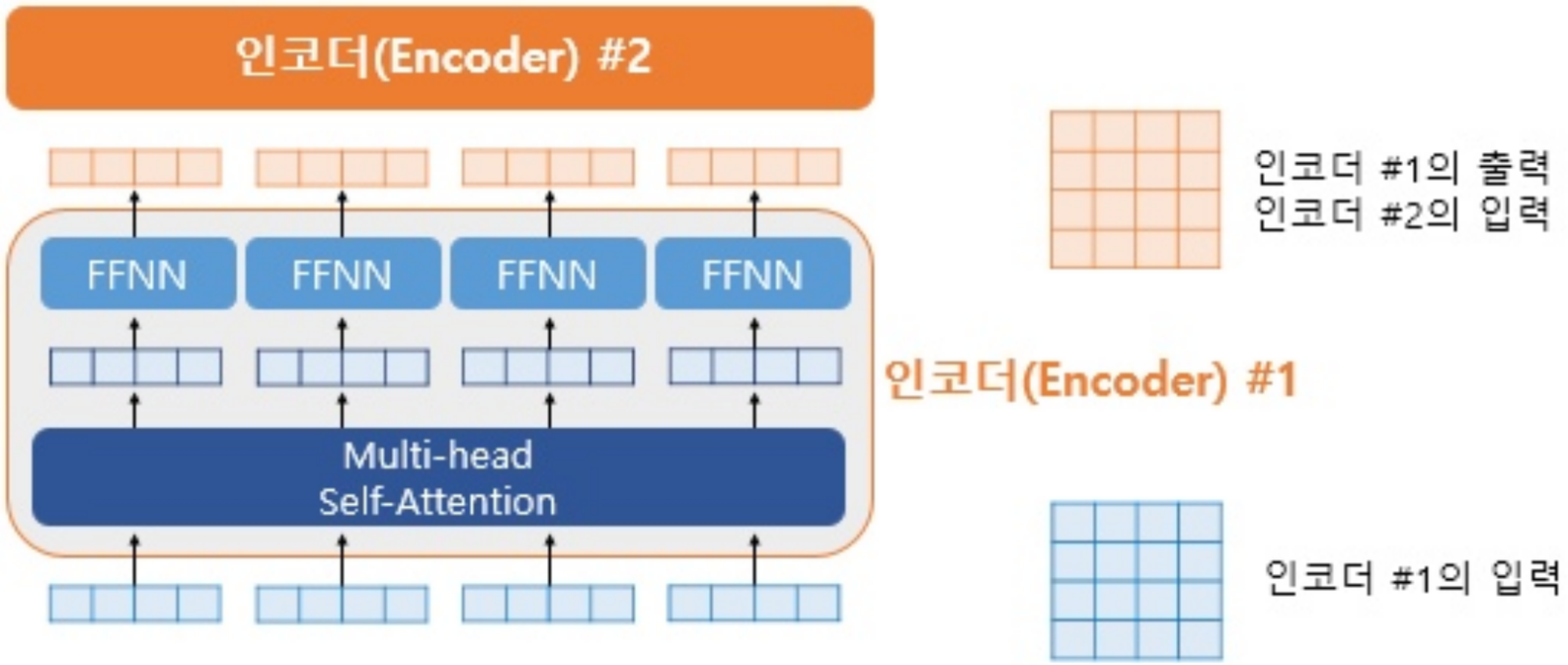
$W_2 = (d_{ff}, d_{model})$

# Position-wise Feed Forward Neural Network

포지션 와이즈 피드 포워드 신경망은 단순 피드 포워드 신경망이다.

은닉층에서는 활성화 함수로 ReLU 함수를 사용한다.

d_ff 는 피드 포워드 신경망의 은닉층의 크기로 논문에서는 2.048이 사용되었다.

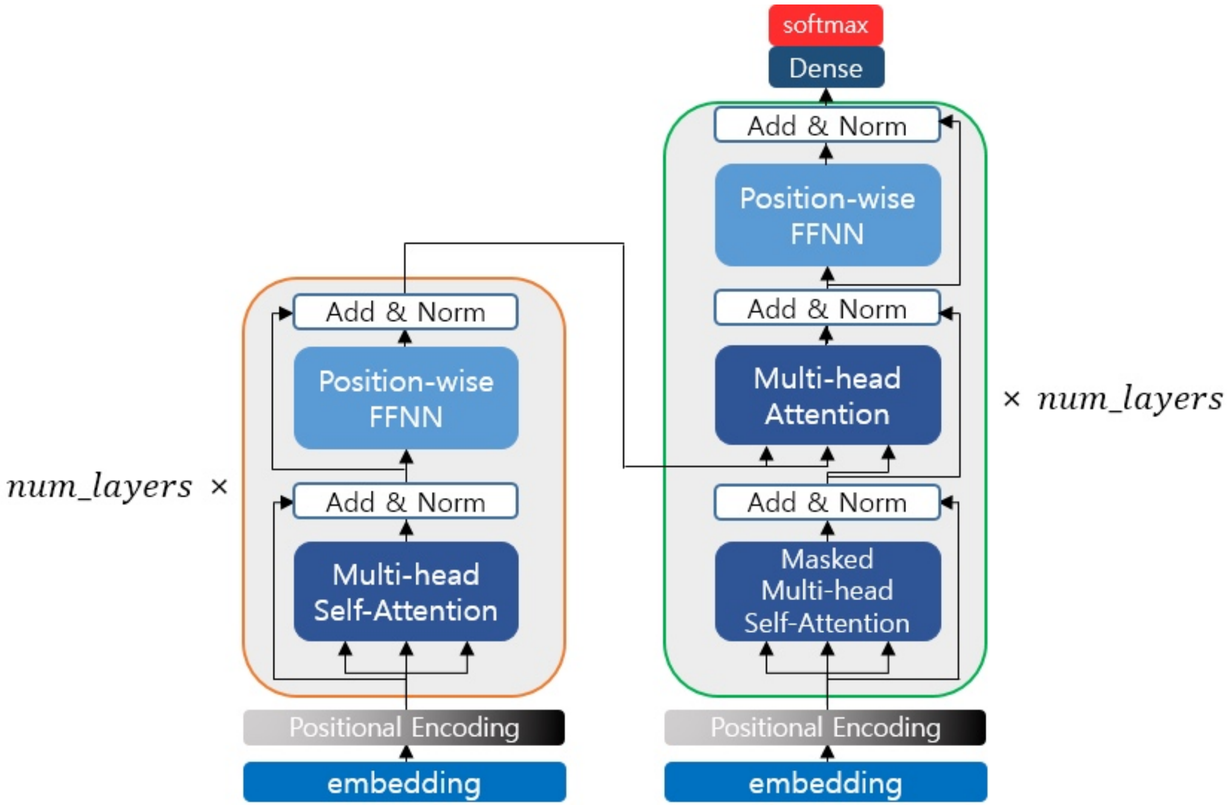


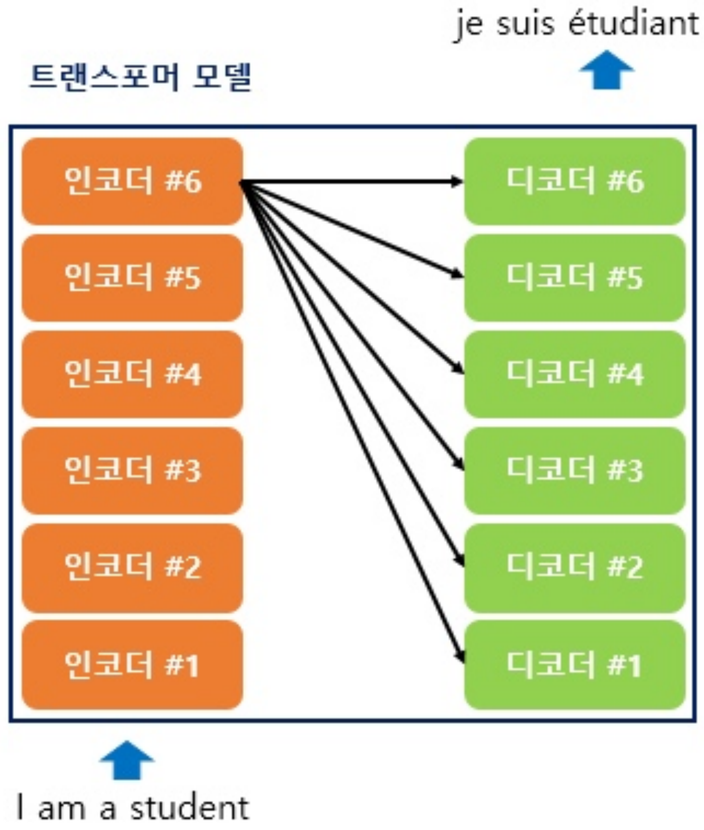

$$x = (\text{seq\_len}, d_{model})$$
$$W_1 = (d_{model}, d_{ff})$$
$$W_2 = (d_{ff}, d_{model})$$

입력의 크기는 두번째 서브층을 통과한 후에도 여전히 보존된다.

# From Encoder to Decoder

총 num_layers(논문에서는 6)회의 인코더 연산을 한 후, 마지막 인코더의 출력이 디코더로 전달된다.
디코더는 인코더의 출력을 전달받아서 다시 num_layers(논문에서는 6)회의 연산을 한다



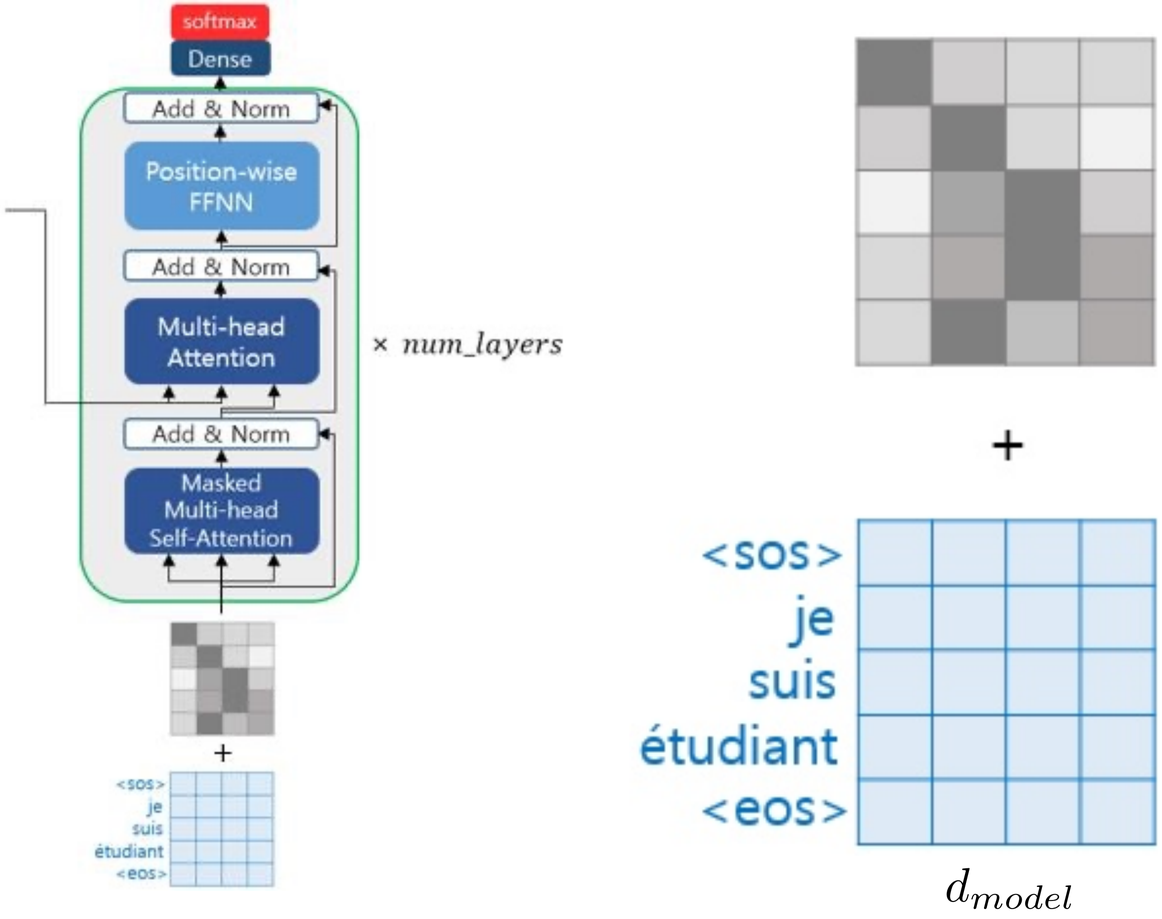
# Transformer Decoder

# Transformer Decoder : Positional Encoding

트랜스포머의 디코더 또한 포지셔널 인코딩을 사용한다.

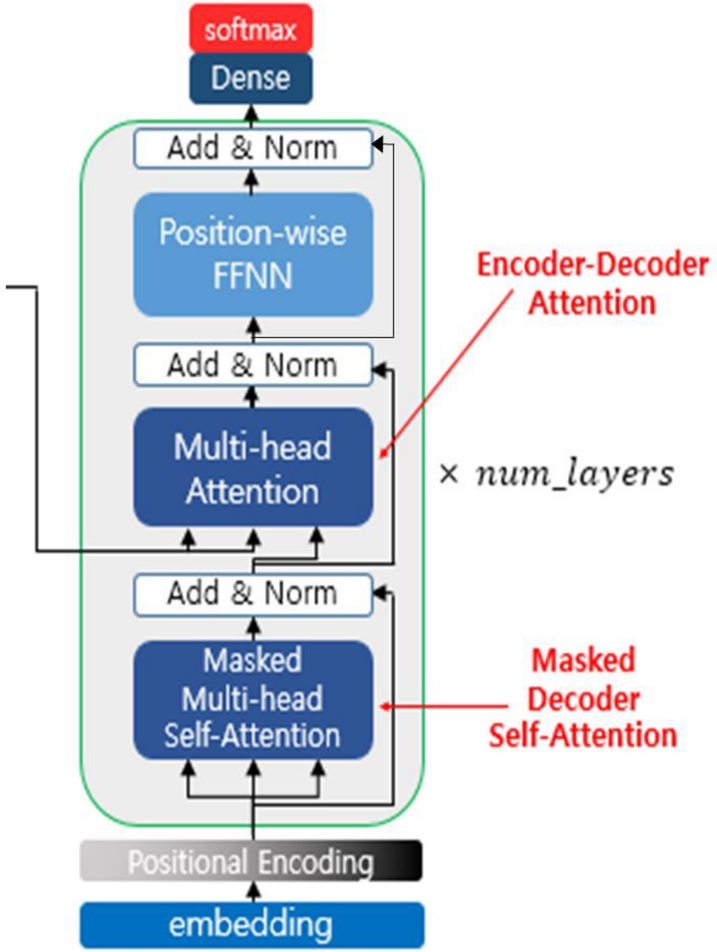
디코더의 입력인 문장 행렬에서는 (seq2seq와 마찬가지로) 시작 토큰과 종료 토큰이 있다.

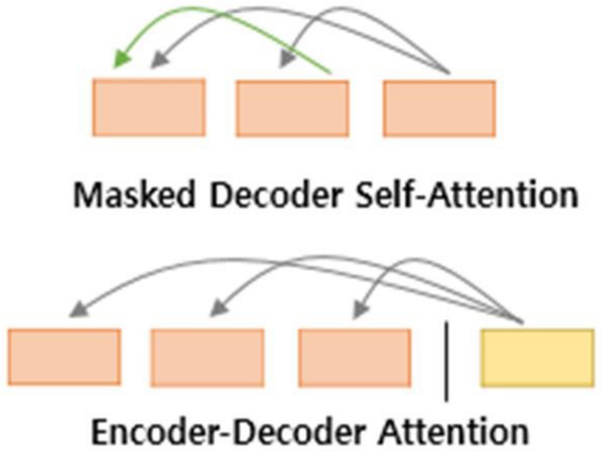
인코더의 마지막 층에서 온 출력



# Transformer Decoder : Positional Encoding

트랜스포머의 디코더에는 총 두 개의 어텐션이 존재.




Q, K, V 가 같은 곳에

Q, K, V 가 다른 곳에

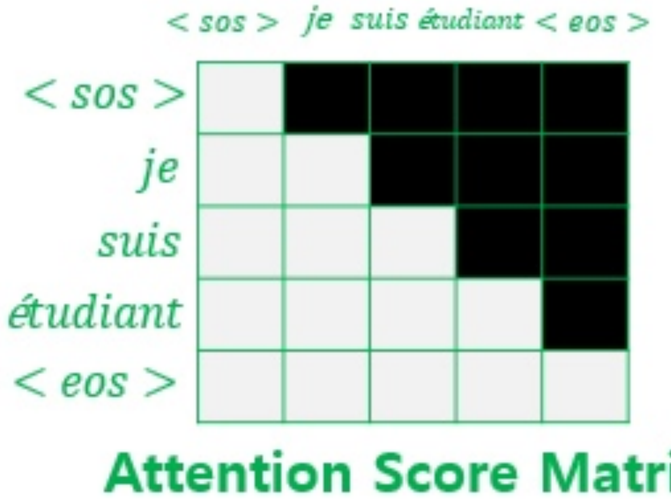
# Transformer Decoder : Self-Attention

트랜스포머 디코더의 Masked self-Attention은 근본적으로 Self-Attention과 동일하다.

# Transformer Decoder : Self-Attention

트랜스포머 디코더의 Masked self-Attention은 근본적으로 Self-Attention과 동일하다.

하지만 어텐션 스코어 매트릭스에 직각 삼각형의 마스킹을 해준다.

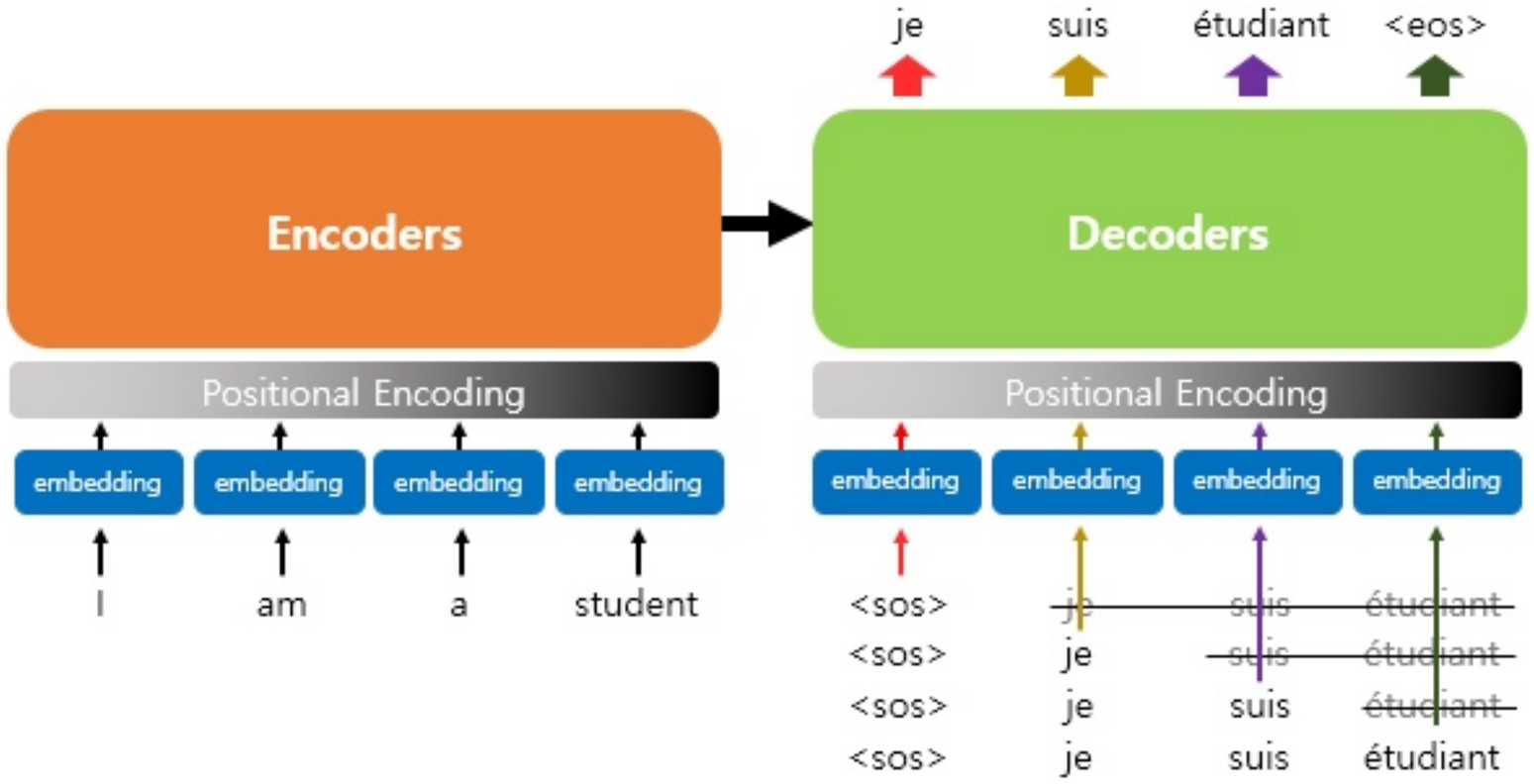



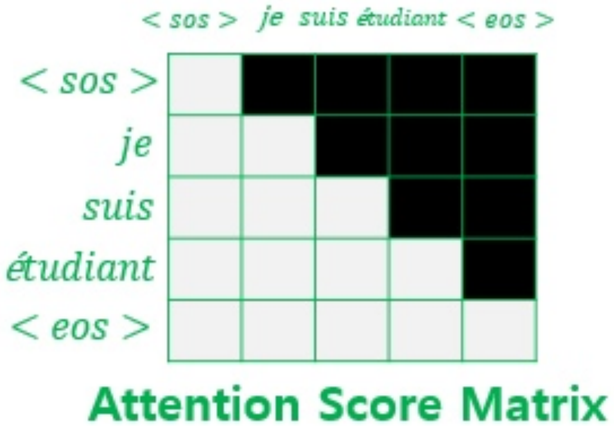
# Transformer Decoder : Self-Attention

트랜스포머 디코더는 훈련 과정에서 실제 예측할 문장 행렬을 입력으로 넣어준다.

훈련 과정에서 정답을 알려주는 이 과정을 교사 강요(Teacher Forcing)라 부른다.

그런데 셀프-어텐션을 할 때, 다음 단어를 이미 알고있다는 가정 하에 어텐션을 해서는 안된다.



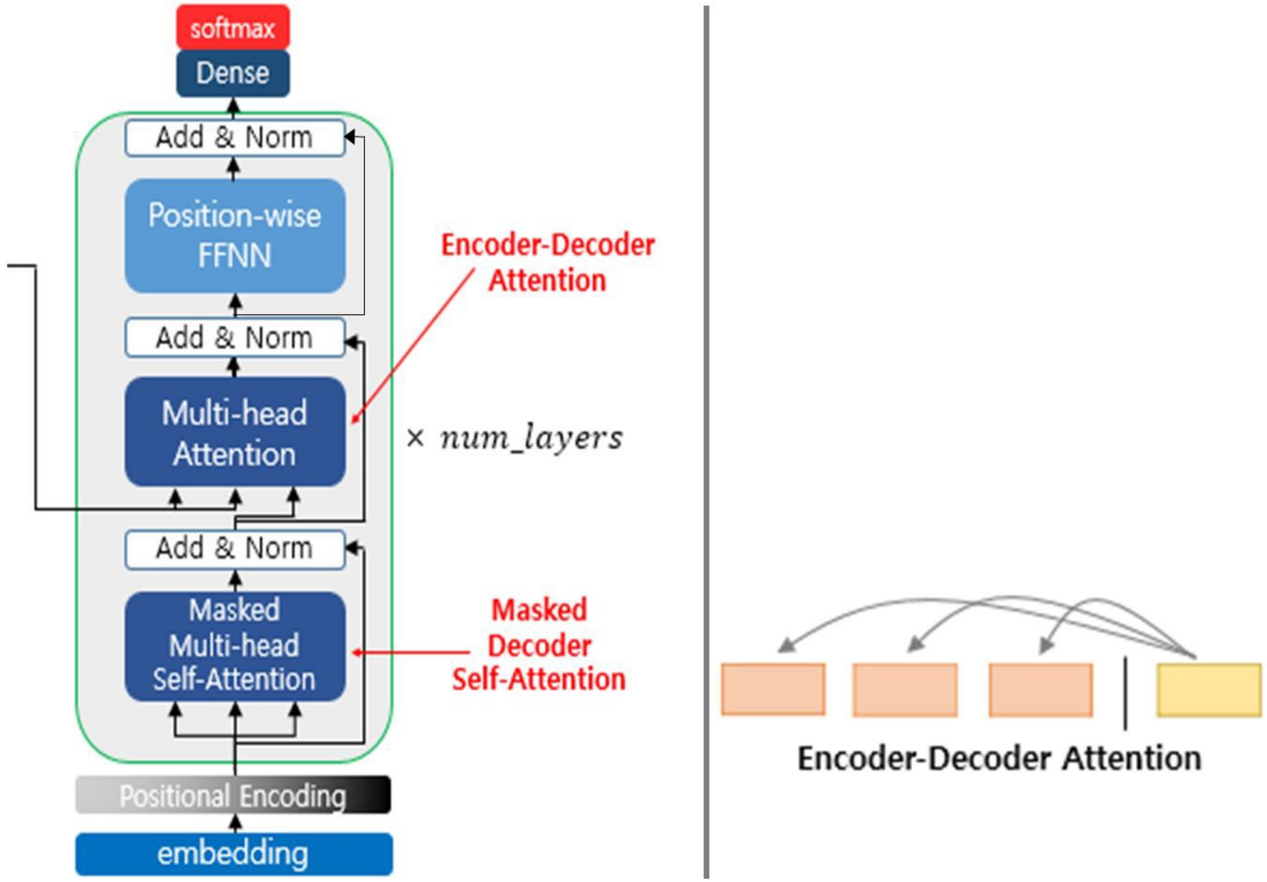




Attention Score Matrix

# Transformer Decoder : Encoder-Decoder Attention

트랜스포머 디코더에서 이루어지는 인코더-디코더 어텐션은 Q와 K, V는 다르다.

Q는 디코더의 벡터인 반면, K, V는 인코더의 벡터이다.







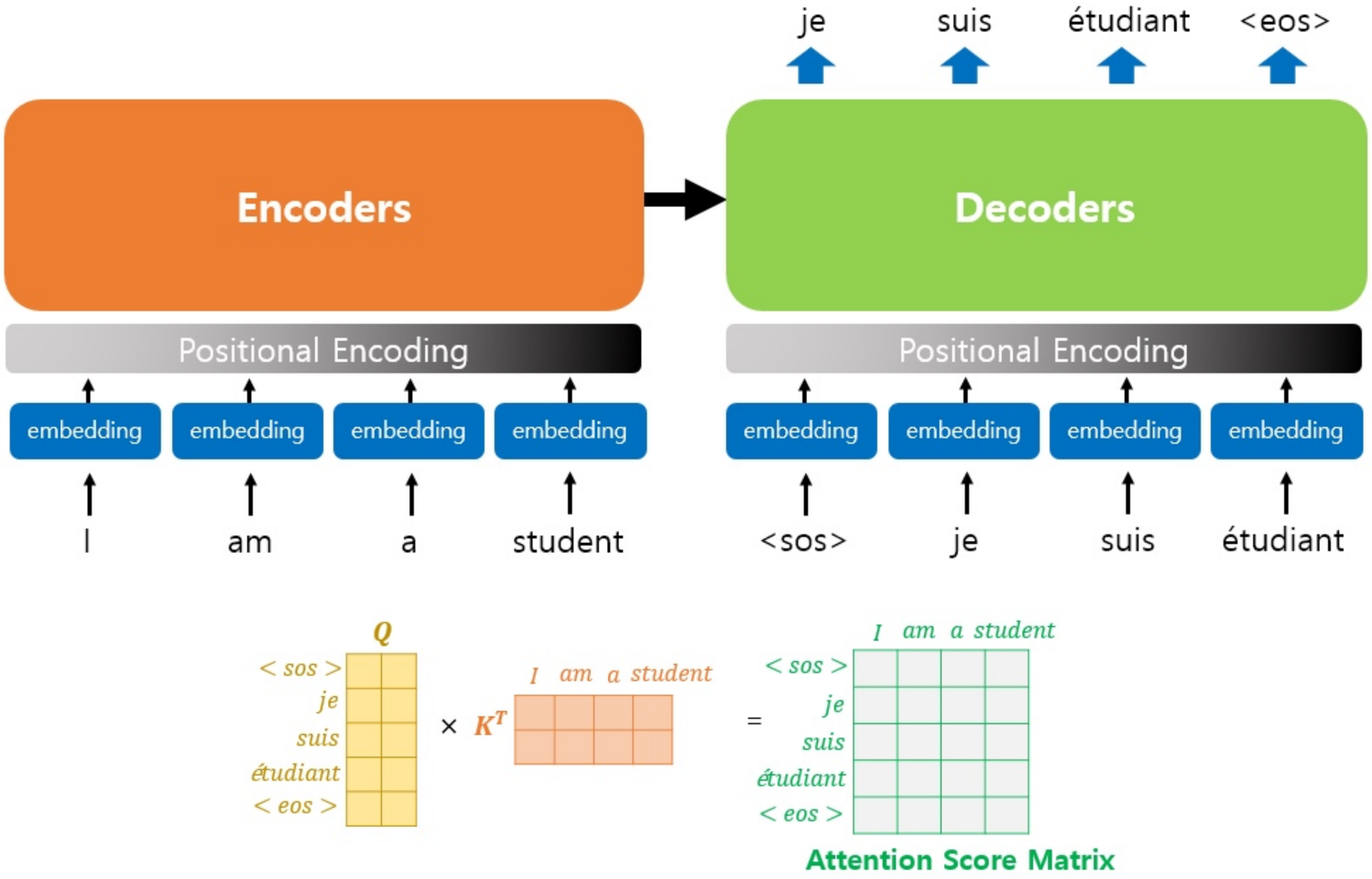
Query: Decoder 벡터

Key, Value : Encoder 벡터

# Transformer Decoder : Encoder-Decoder Attention

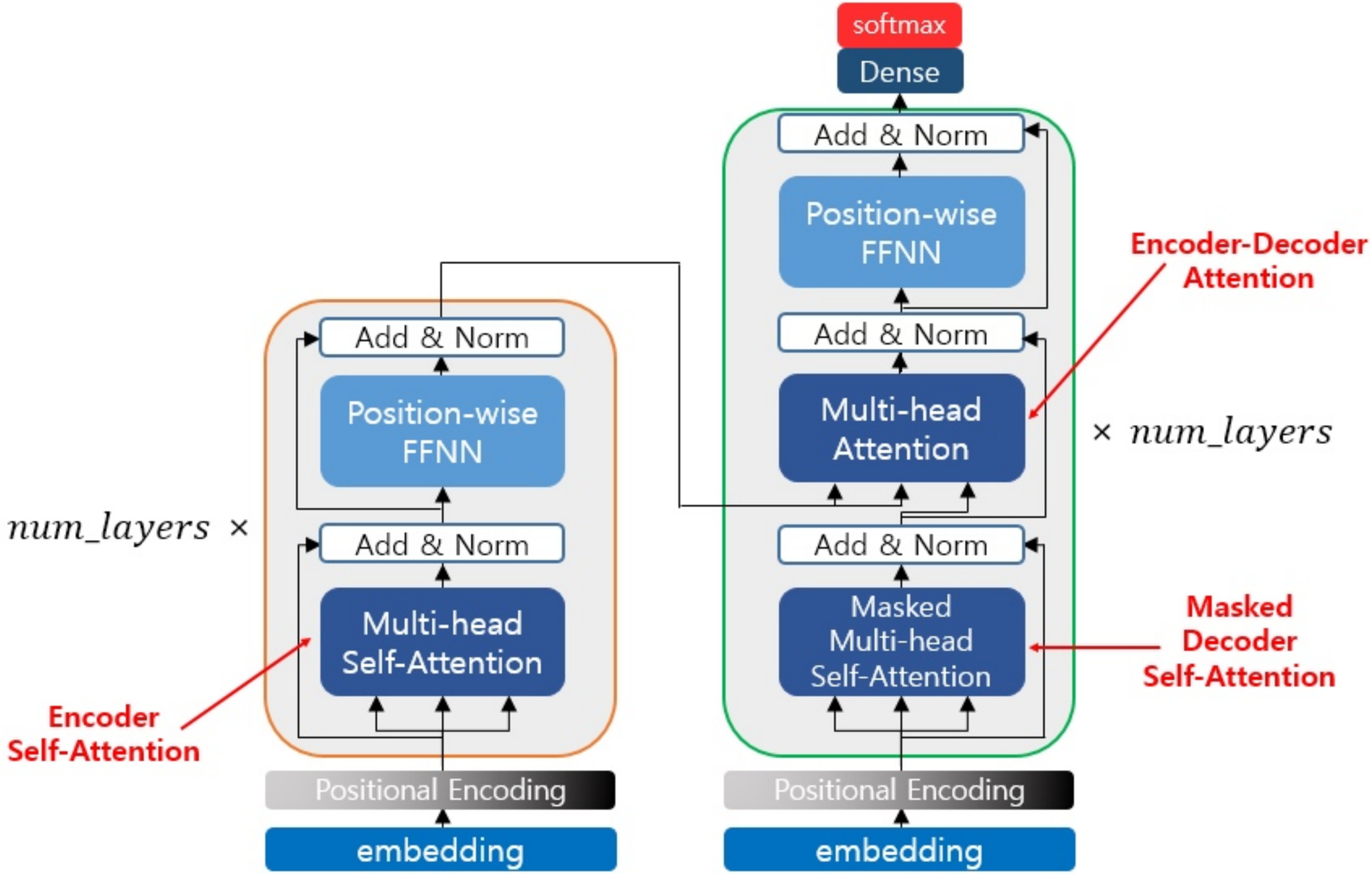
트랜스포머 디코더에서 이루어지는 인코더-디코더 어텐션은 Q와 K, V는 다르다.

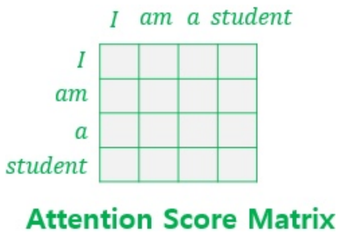
Q는 디코더의 벡터인 반면, K, V는 인코더의 벡터이다.



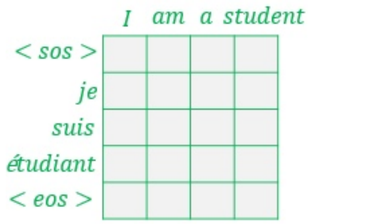
# Transformer Decoder : Attention Mechanism

트랜스포머는 총 세 종류의 어텐션이 존재.

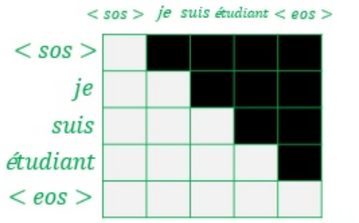



Attention Score Matrix



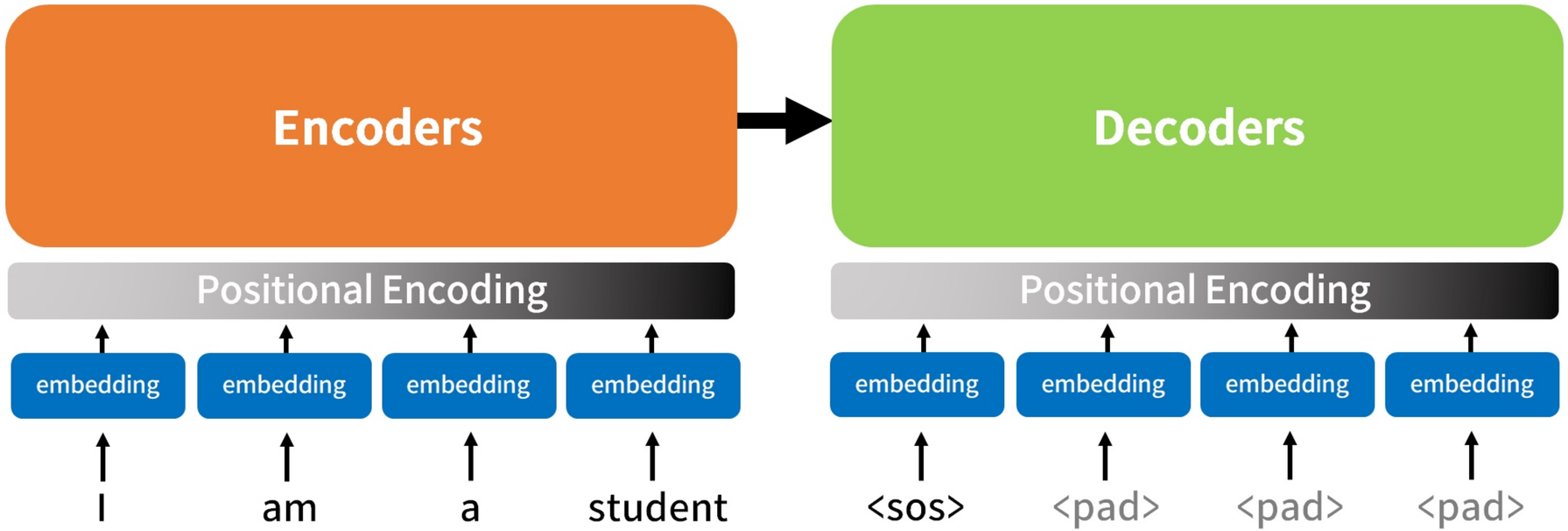

Attention Score Matrix




Attention Score Matrix

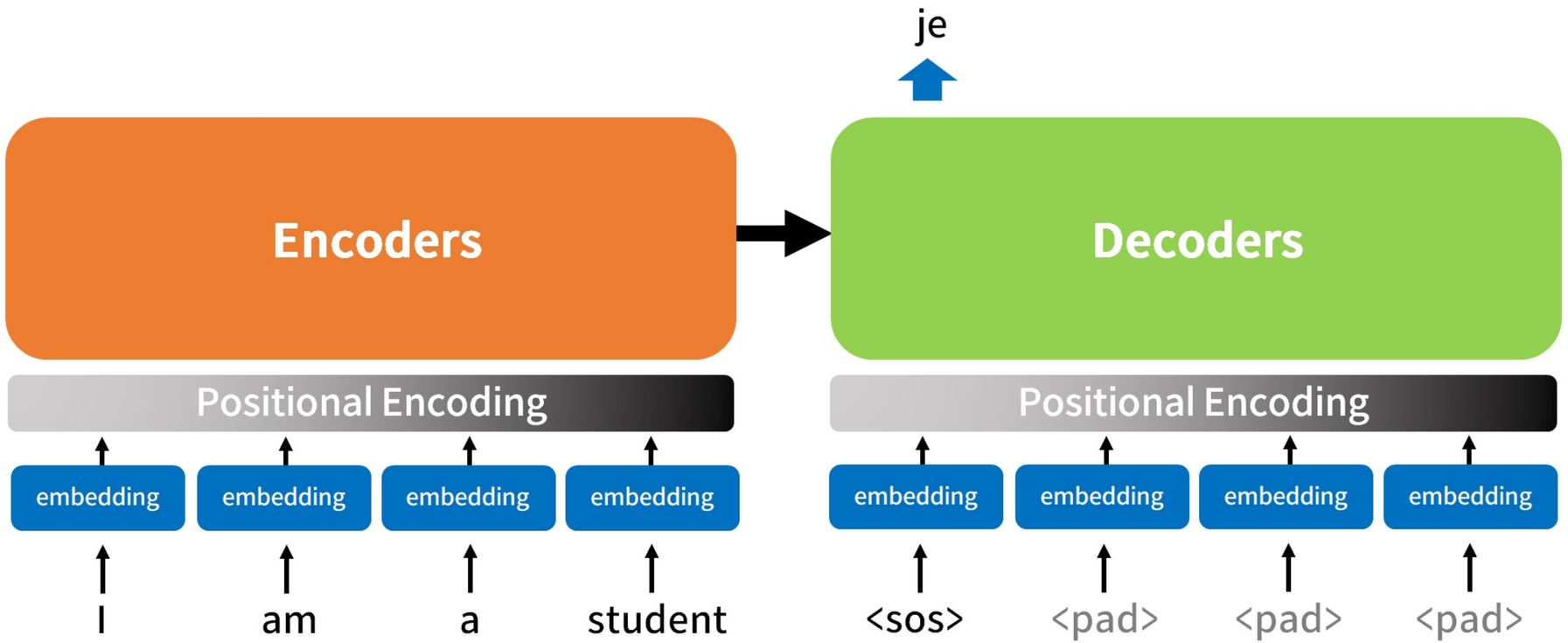
# Transformer: Inference

테스트 단계(실제 번역기나 챗봇이 구동될 때)에는 디코더가 단어를 1개씩 생성해내야 한다.



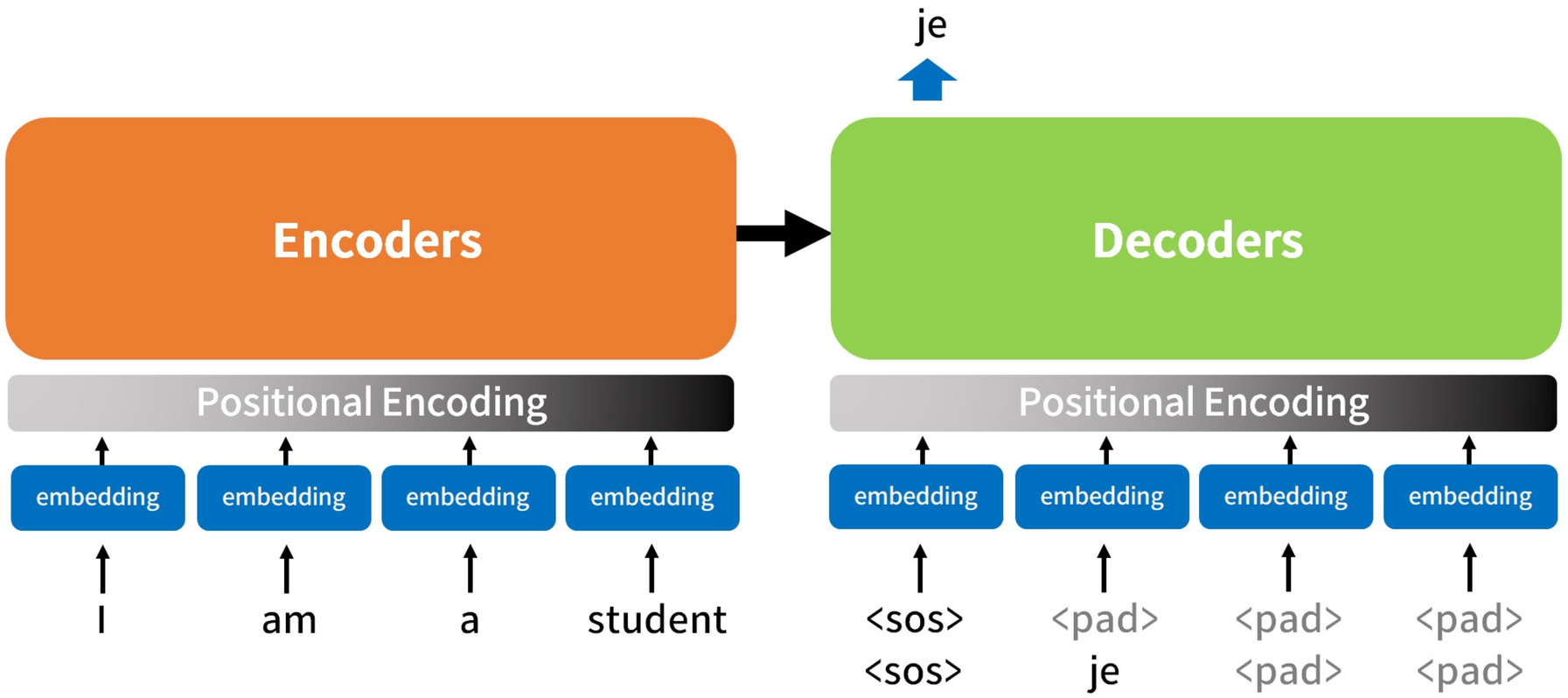
# Transformer: Inference

테스트 단계(실제 번역기나 챗봇이 구동될 때)에는 디코더가 단어를 1개씩 생성해내야 한다.



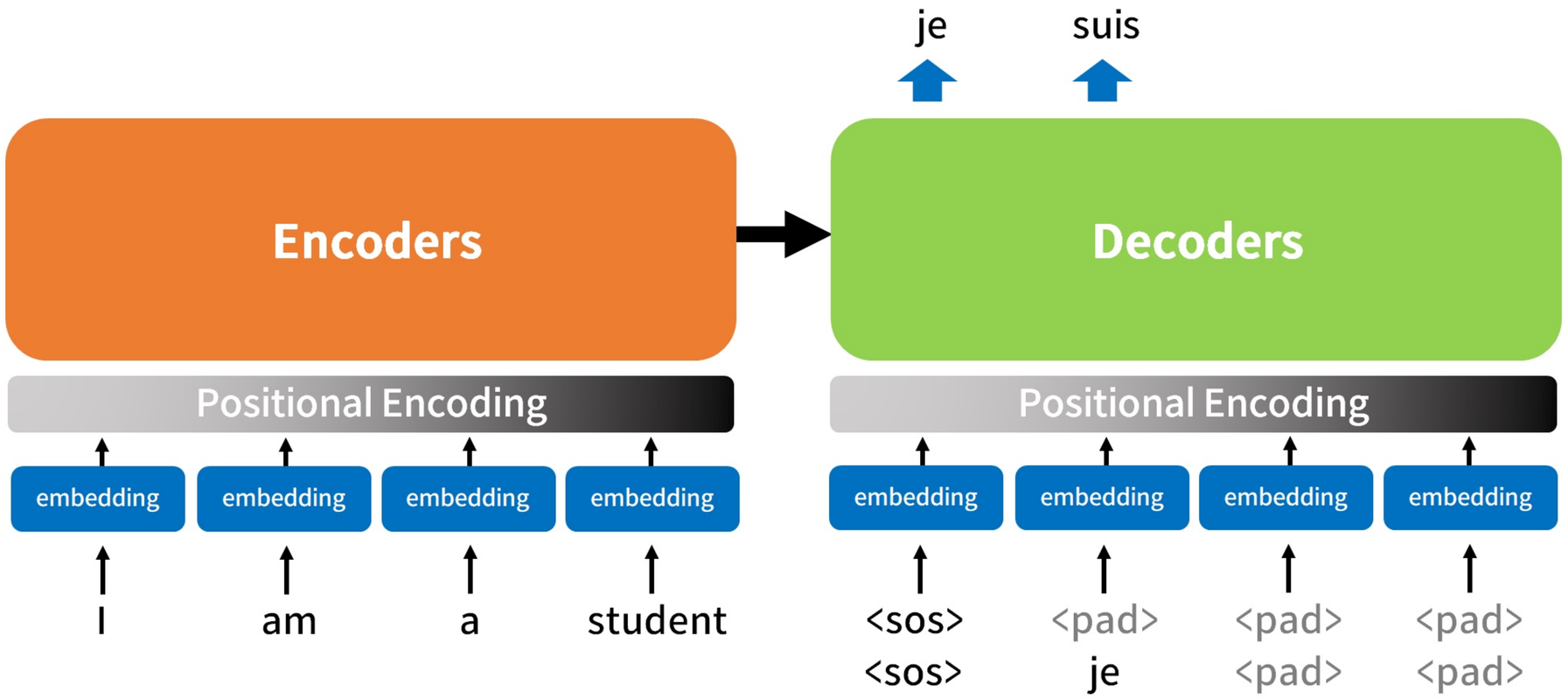
# Transformer: Inference

테스트 단계(실제 번역기나 챗봇이 구동될 때)에는 디코더가 단어를 1개씩 생성해내야 한다.



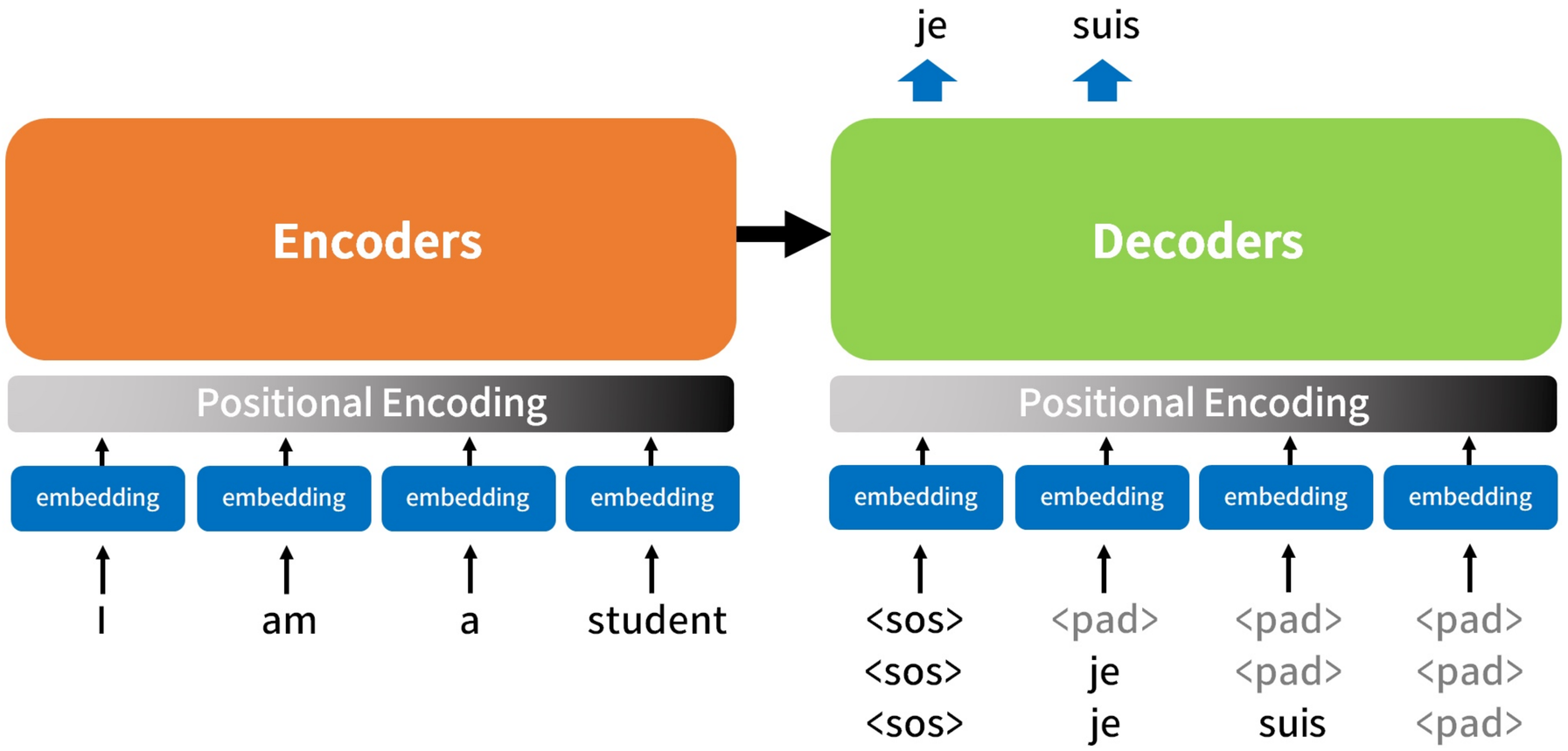
# Transformer: Inference

테스트 단계(실제 번역기나 챗봇이 구동될 때)에는 디코더가 단어를 1개씩 생성해내야 한다.



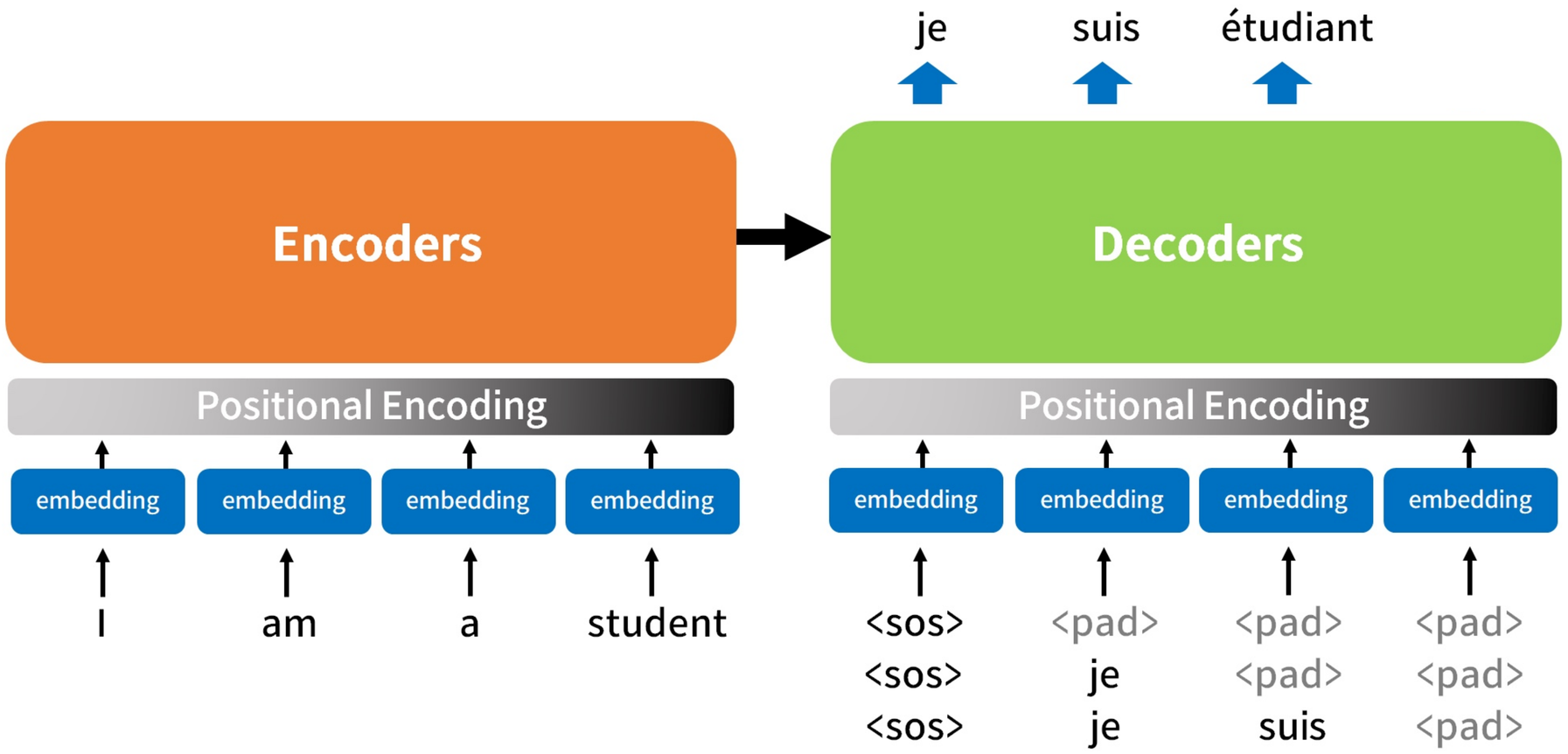
# Transformer: Inference

테스트 단계(실제 번역기나 챗봇이 구동될 때)에는 디코더가 단어를 1개씩 생성해내야 한다.



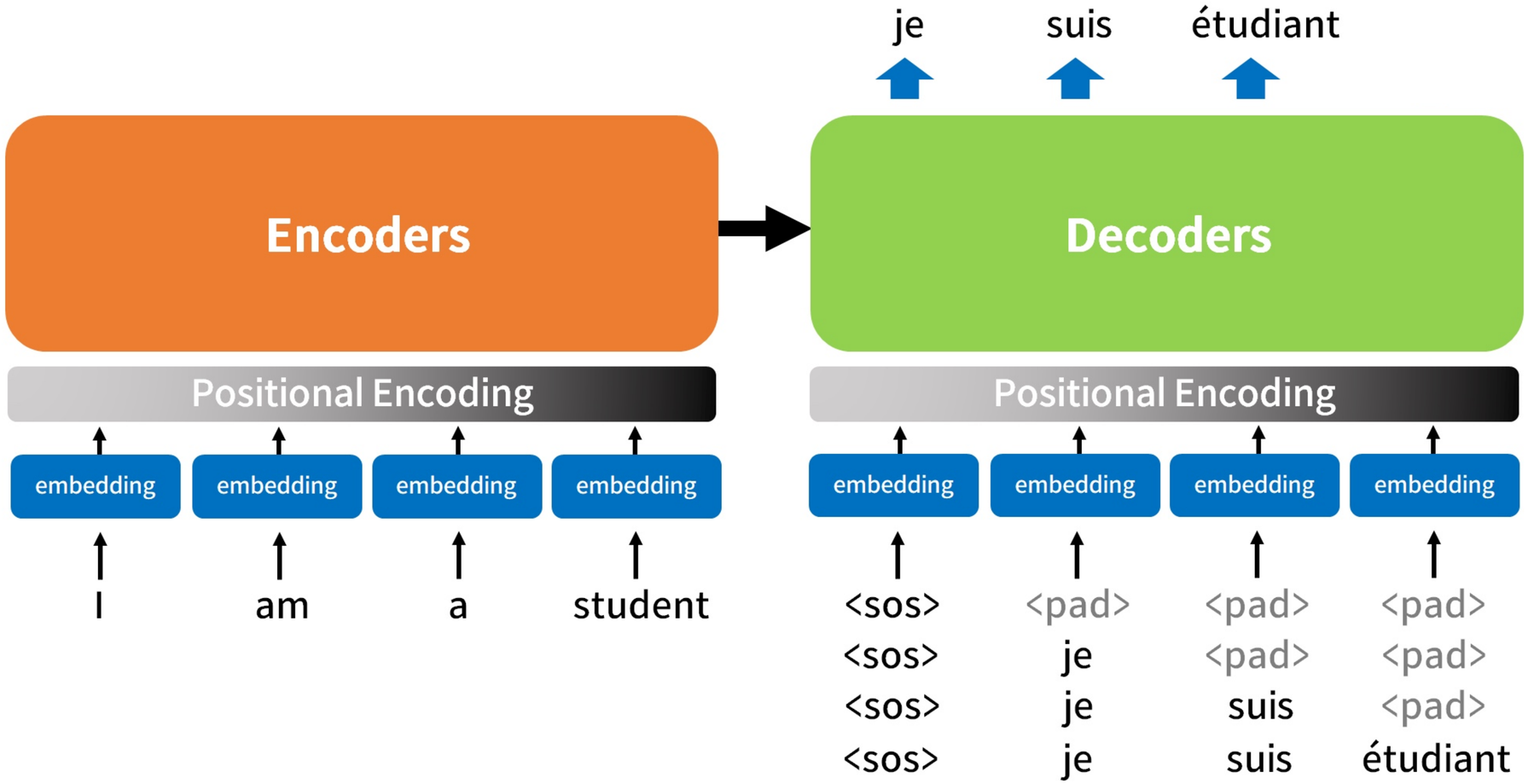
# Transformer: Inference

테스트 단계(실제 번역기나 챗봇이 구동될 때)에는 디코더가 단어를 1개씩 생성해내야 한다.



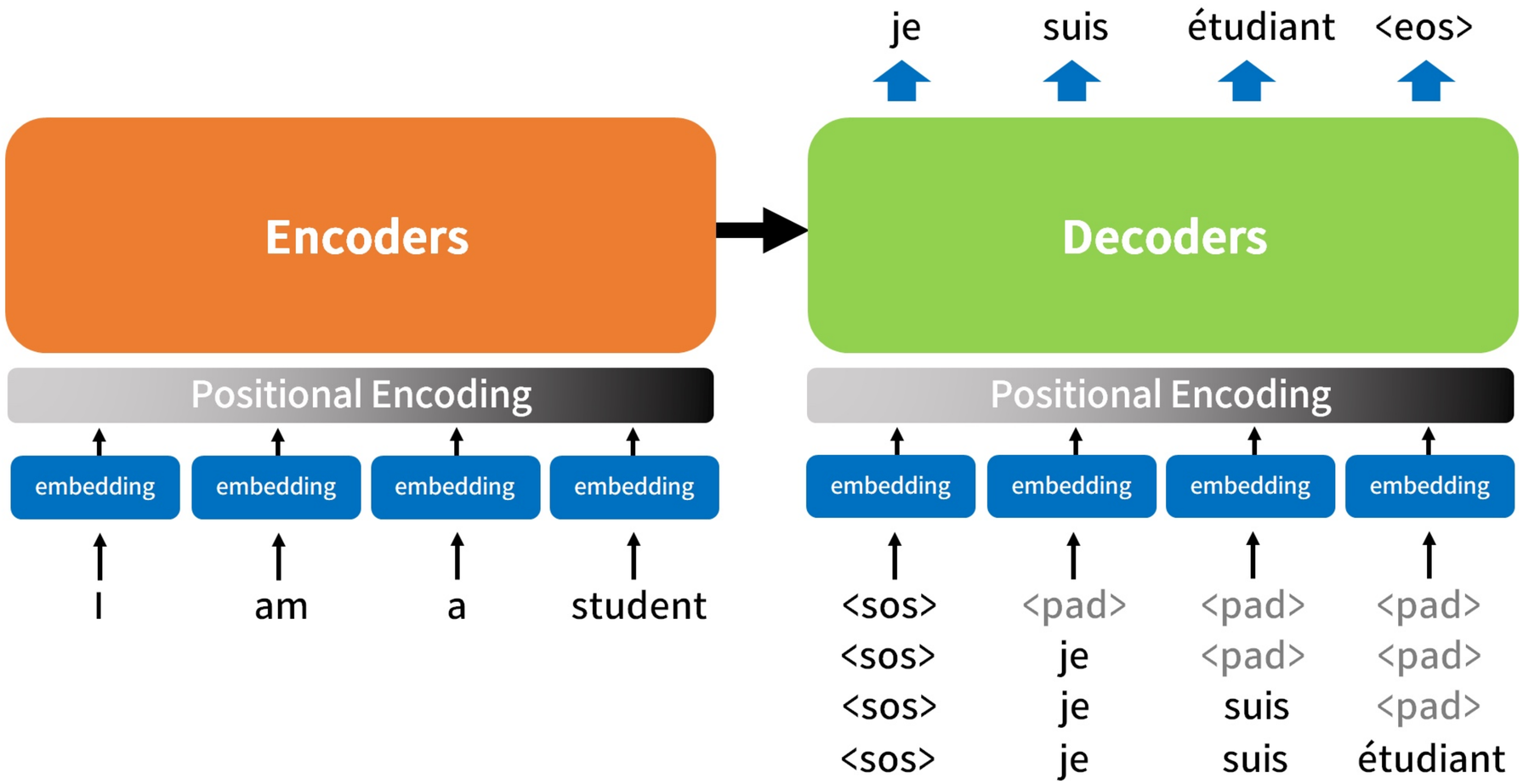
# Transformer: Inference

테스트 단계(실제 번역기나 챗봇이 구동될 때)에는 디코더가 단어를 1개씩 생성해내야 한다.



# Transformer: Inference

테스트 단계(실제 번역기나 챗봇이 구동될 때)에는 디코더가 단어를 1개씩 생성해내야 한다.



# Transformer 최종 정리

# Transformer 최종 정리