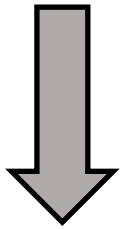
Transformer

Embedding

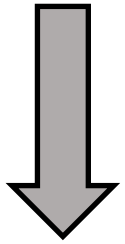
"Apple"



컴퓨터는 못 알아듣는데?

숫자 할당 (ex. 3)

➔ Vector 화: 원-핫 인코딩 (ex. [0, 0, 0, 1, … , 0])



단어들의 관계성을 표현할 수 있었으면 좋겠는데?

차원이 너무 커지는데?

밀집 벡터 (ex. [0.5, 0.1, 0.1, … , 0.1])

Predictive Embedding – Word2Vec

신경망 구조/모델을 이용하여 특정 문맥에서 어떤 단어가 나올지 예측하면서 단어를 벡터로(임베딩)을 학습하는 방법

Word2Vec

신경망 알고리즘, 주어진 텍스트에서 텍스트의 각 단어마다 하나씩 일련의 벡터를 출력

유사한 의미의 단어들은 비슷한 벡터로 표현 (코사인 유사도, 거리가 가깝다.)
단어의 동의어를 찾을 수 있음
학습 방식: CBOW(Continuous Bag of Words), Skip-Gram

CBOW

주변에 있는 단어들을 입력으로 중간에 있는 단어들을 예측(출력)하는 방법

예문: The fat cat sat on the mat

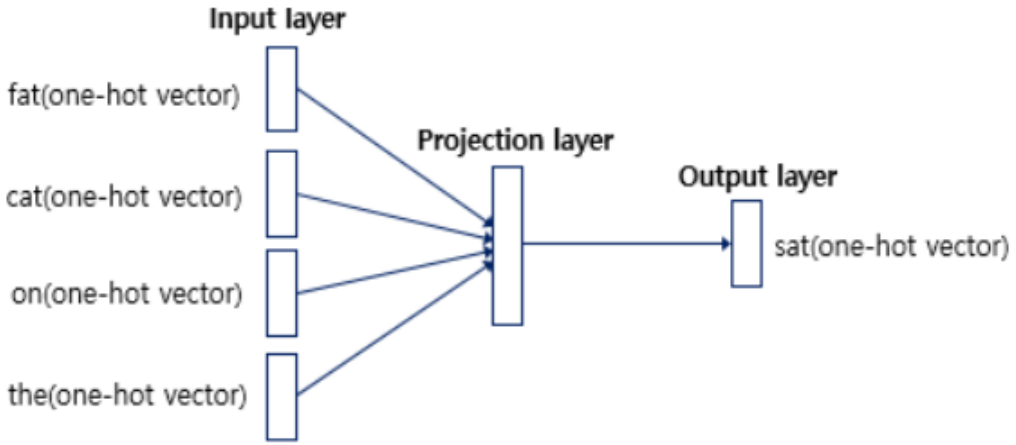
중심 단어	주변 단어
The fat cat sat on the mat	[1, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0]
The fat cat sat on the mat	[0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0]
The fat cat sat on the mat	[0, 0, 1, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0]
The fat cat sat on the mat	[0, 0, 0, 1, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]
The fat cat sat on the mat	[0, 0, 0, 0, 1, 0]	[0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]
The fat cat sat on the mat	[0, 0, 0, 0, 0, 1]	[0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]
The fat cat sat on the mat	[0, 0, 0, 0, 0, 0]	[0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]

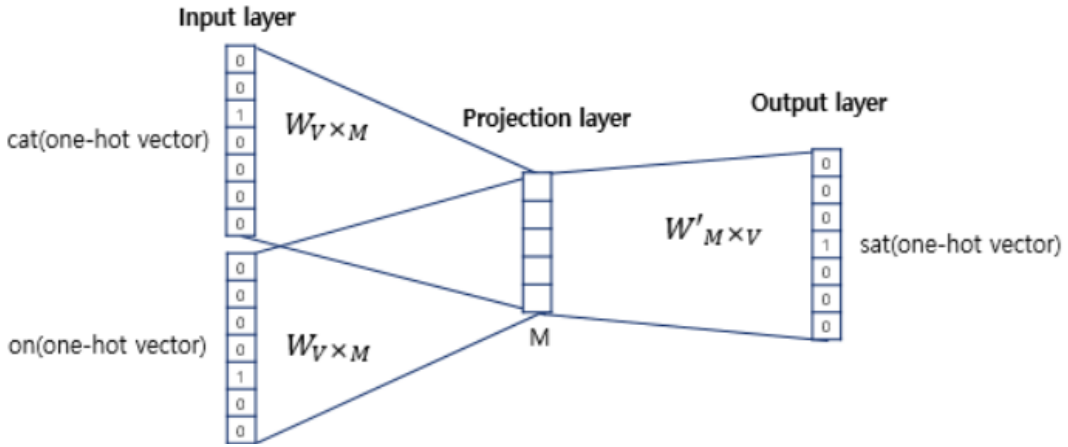
CBOW

주변에 있는 단어들을 입력으로 중간에 있는 단어들을 예측(출력)하는 방법

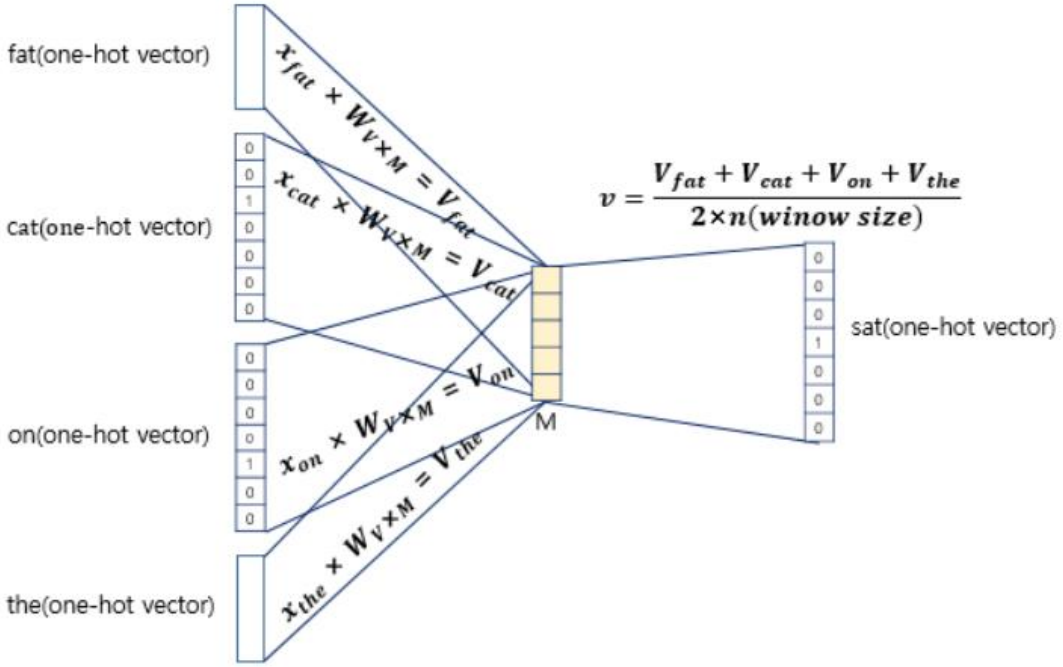
예문: The fat cat sat on the mat

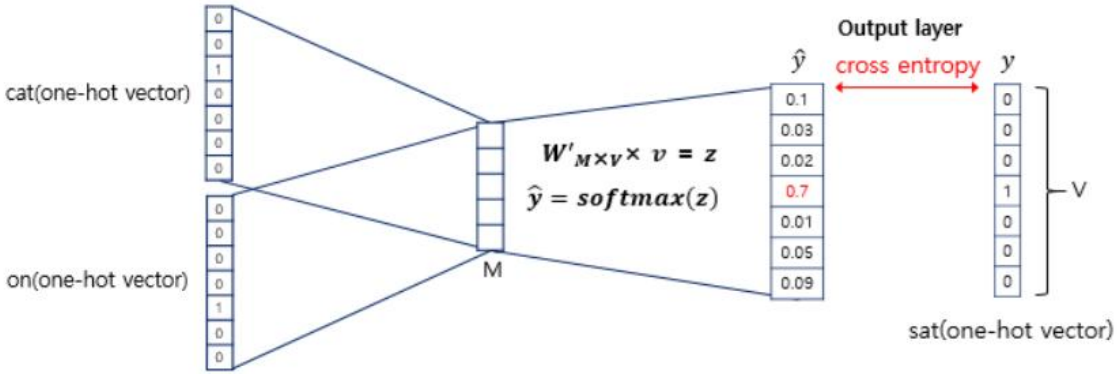
n=2 n=2





CBOW

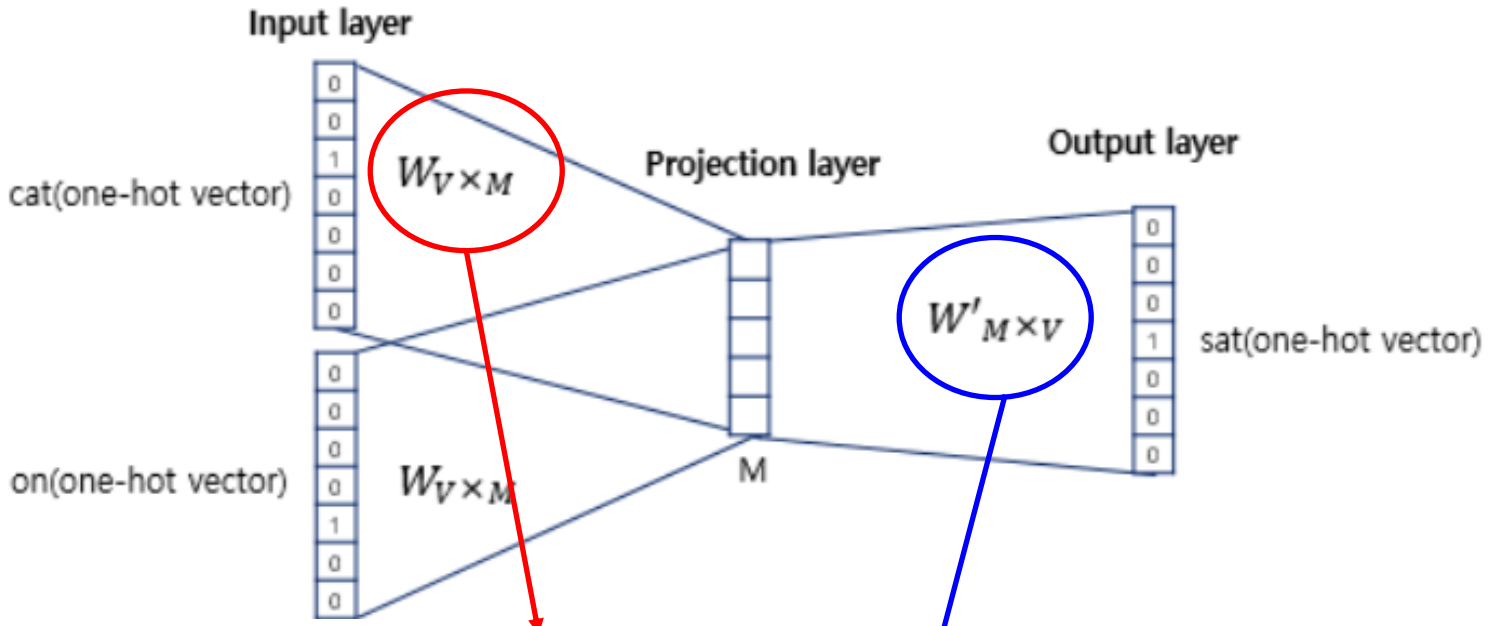




Loss Func.: cross-entropy ($cost(\hat{y}, y) = -\sum_{j=1}^{V} y_j \log(\hat{y}_j)$)

학습 대상: $W_{V\times M}, W'_{M\times V}$

CBOW



주변 단어를 벡터로 변환: $W_{V\times M}$
➔ 이 단어가 어떻게 맥락으로 주어져야 잘 예측하나?

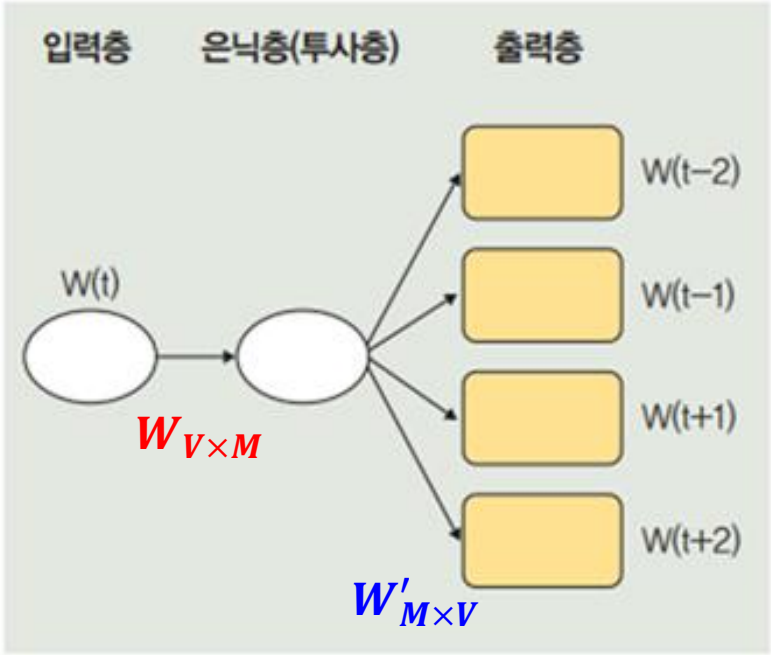
$W'_{M\times V}$: 중앙 단어를 예측
➔ 이 단어가 예측 대상일 때 어떤 맥락과 어울리나?

Transpose 아님 ➔ 각각 따로 학습

Embedding Vector: $W_{V\times M}$ or mixture of ($W_{V\times M}$ and $W'_{M\times V}$)

Skip-Gram

중간에 있는 단어를 입력으로 주변에 있는 단어들을 예측(출력)하는 방법



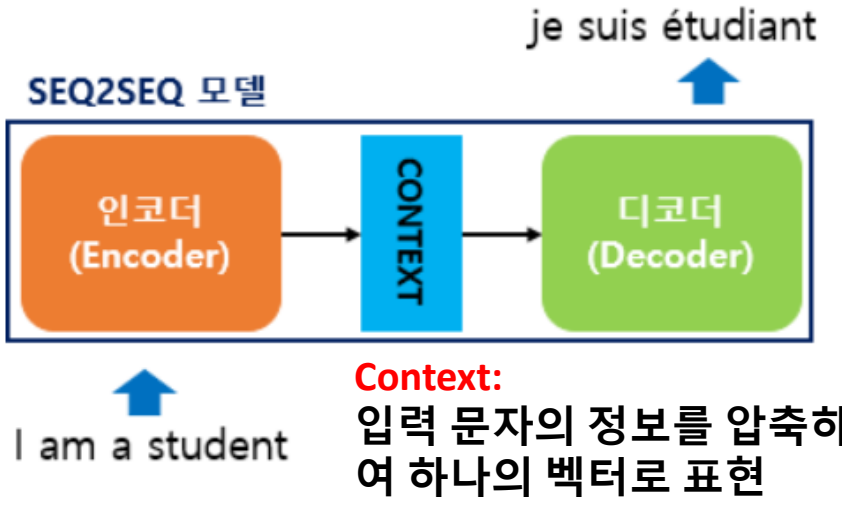
Loss Func.: cross-entropy ($cost(\hat{y}, y) = -\sum_{j=1}^{V} y_j \log(\hat{y}_j)$)

학습 대상: $W_{V\times M}, W'_{M\times V}$

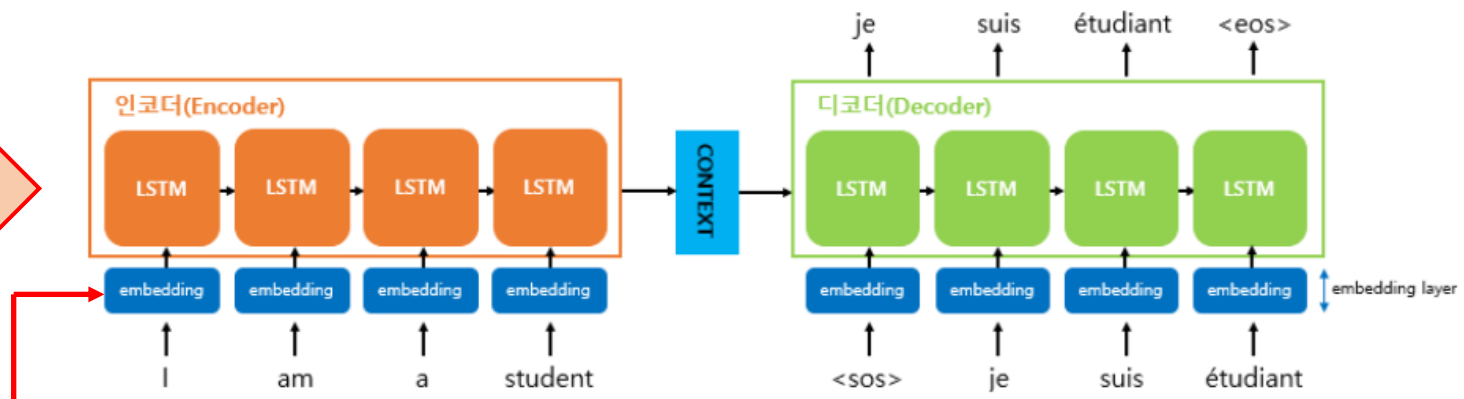
Seq2Seq

Seq2Seq: 인코더와 디코더로 구성

Encoder: 입력 문장의 모든 단어들을 순차적으로 입력 받은 뒤 마지막 state(vector)를 디코더로 전송
Decoder: Encoder로 부터 vector(context)를 받아서 번역된 단어를 하나씩 출력



Context:
입력 문자의 정보를 압축하여 하나의 벡터로 표현



Word Embedding
단어 → Vector
컴퓨터가 이해하기 쉽게 + 데이터의 의미 및 관계성 보존

"Sequence to Sequence Learning with Neural Networks,"
[1409.3215] Sequence to Sequence Learning with Neural Networks

Attention

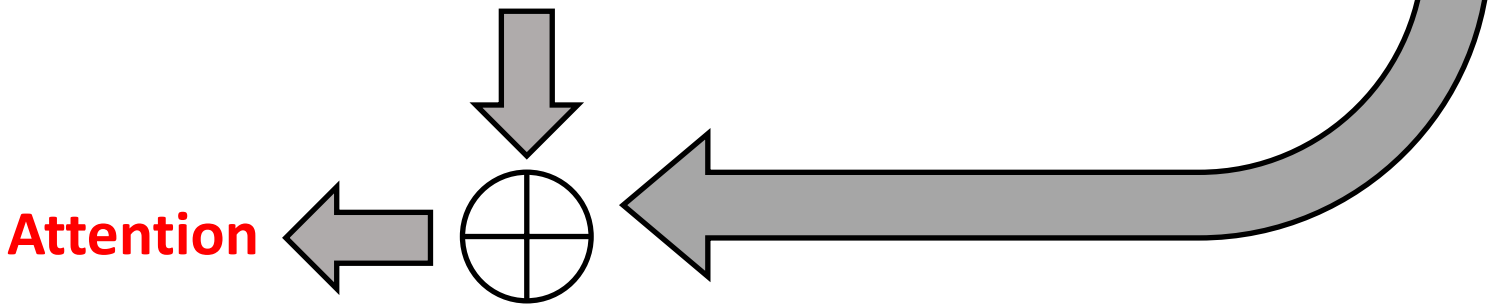
Seq2Seq: 문장의 정보를 하나의 context로 정리하는 데 성공했음!

문장이 길어지면 고정된 크기의 context에 다 담을 수 없는데?

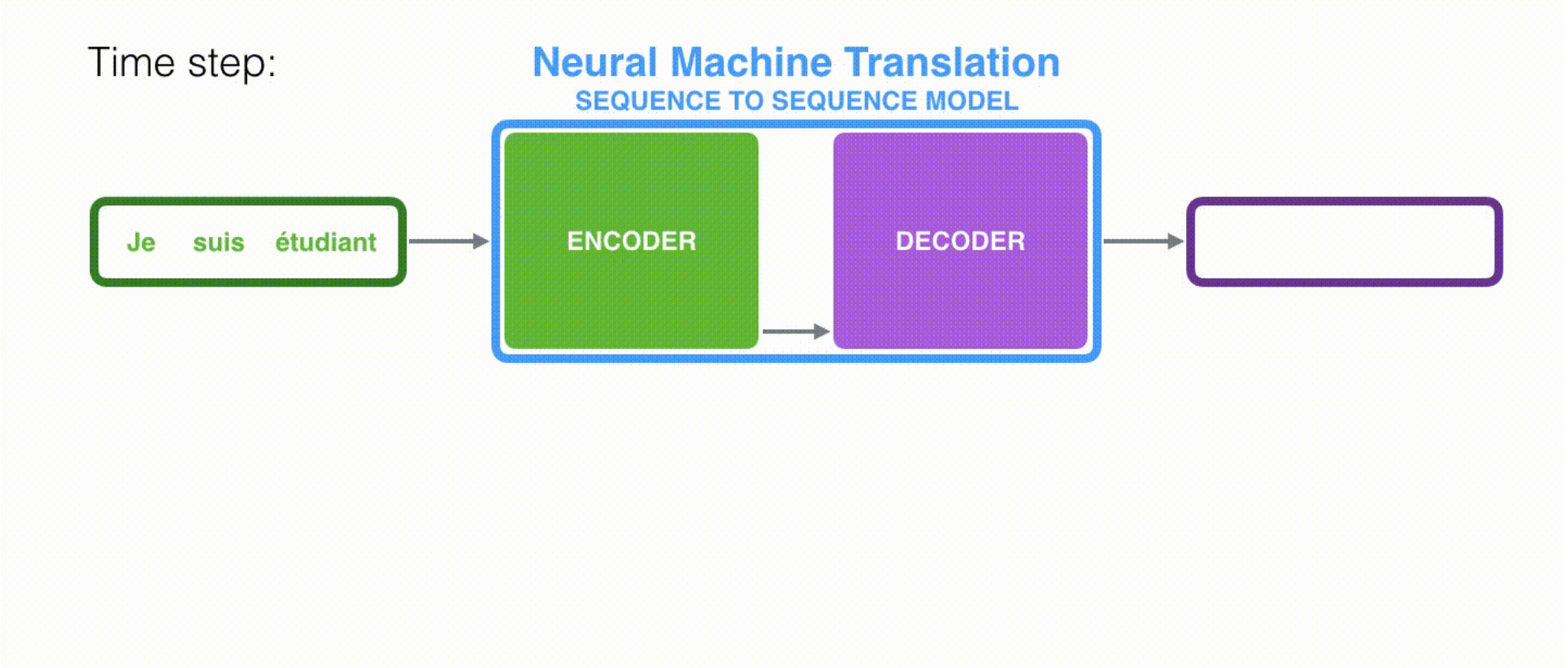
문장이 길어지면 담을 수 있는 context도 늘리자!

너무 정보가 많은데?

문장 전체에서 현재 시점에 중요한 단어에 집중 해 보자!!



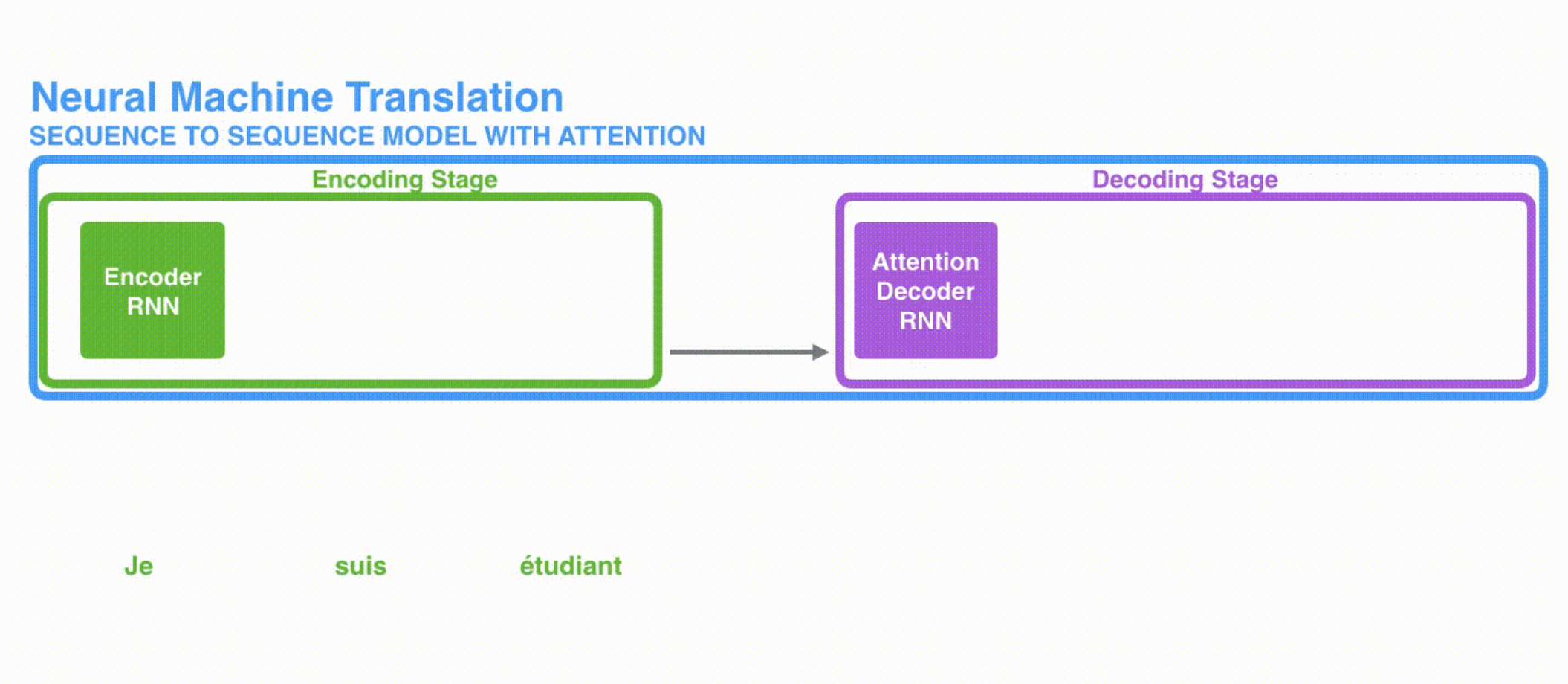
Attention: Seq2Seq



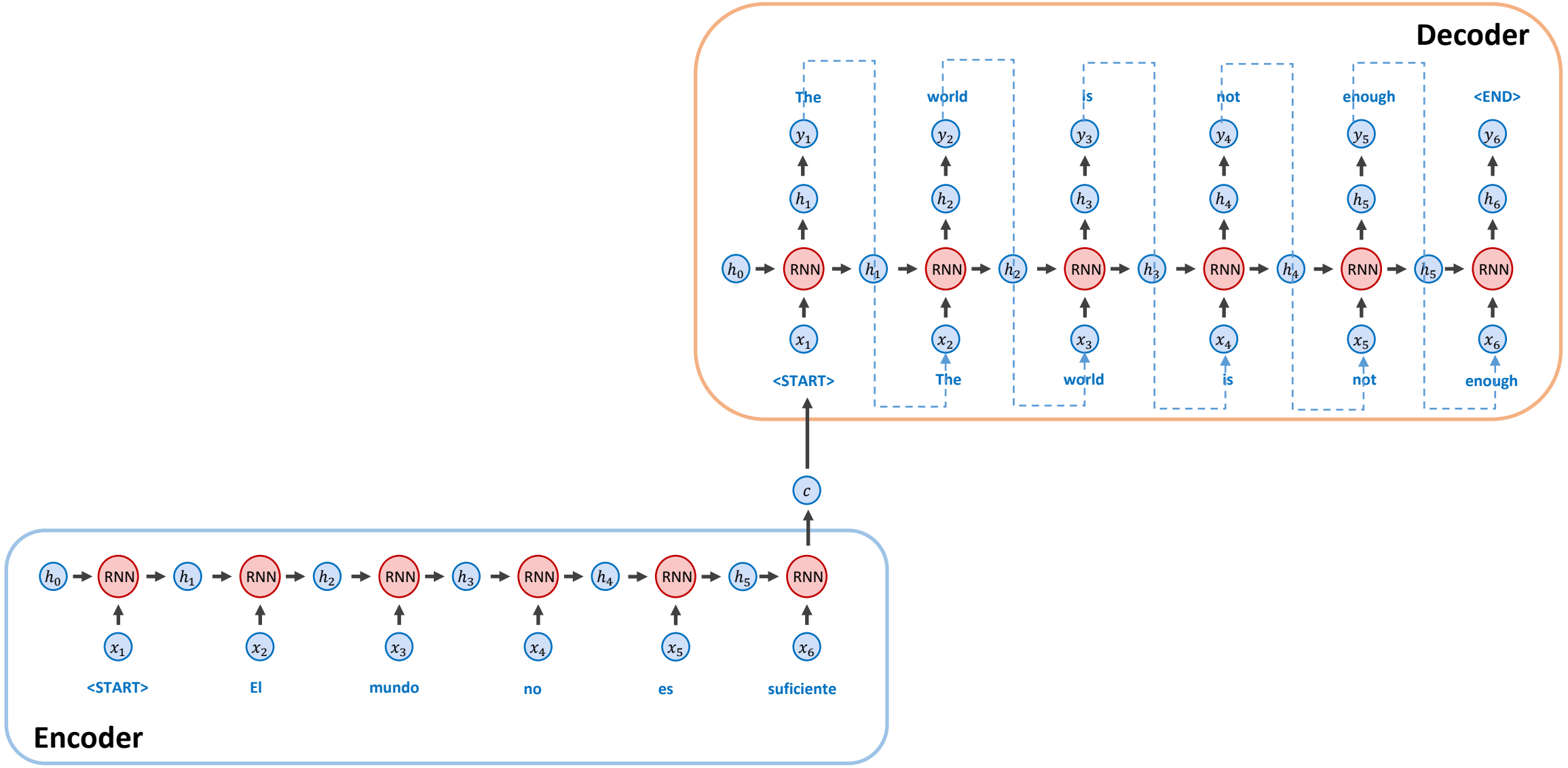
"Neural Machine Translation by Jointly Learning to Align and Translate,"

[1409.0473] Neural Machine Translation by Jointly Learning to Align and Translate

Attention: Overview

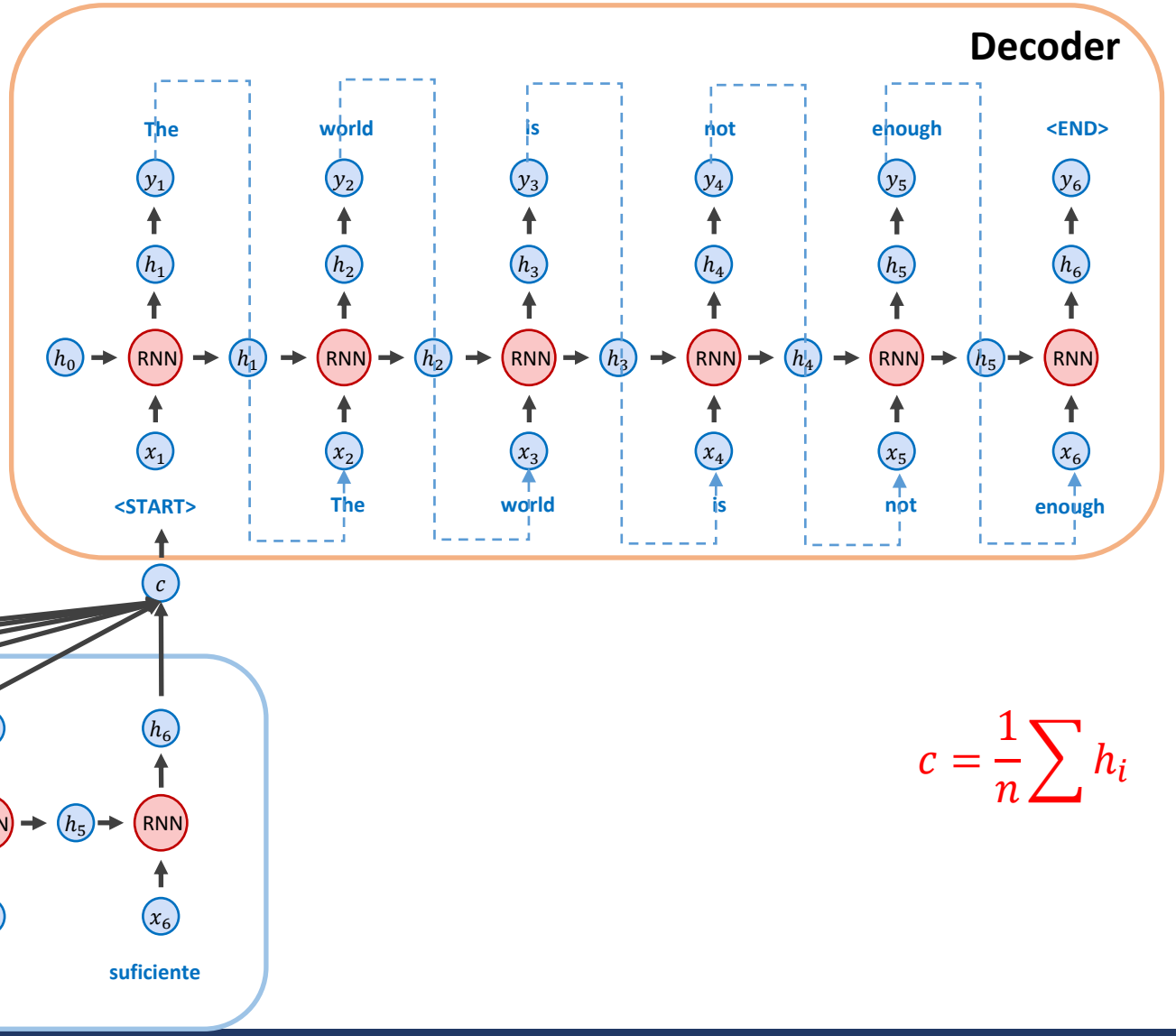


Attention


Decoder
The
world
is
not
enough
<END>
RNN
<START>
The
world
is
not
enough
c
Encoder
<START>
El
mundo
no
es
suficiente

Attention

아마, 전체 hidden 정보를 디코더에 추가적으로 알려주면 더 좋지 않을까?

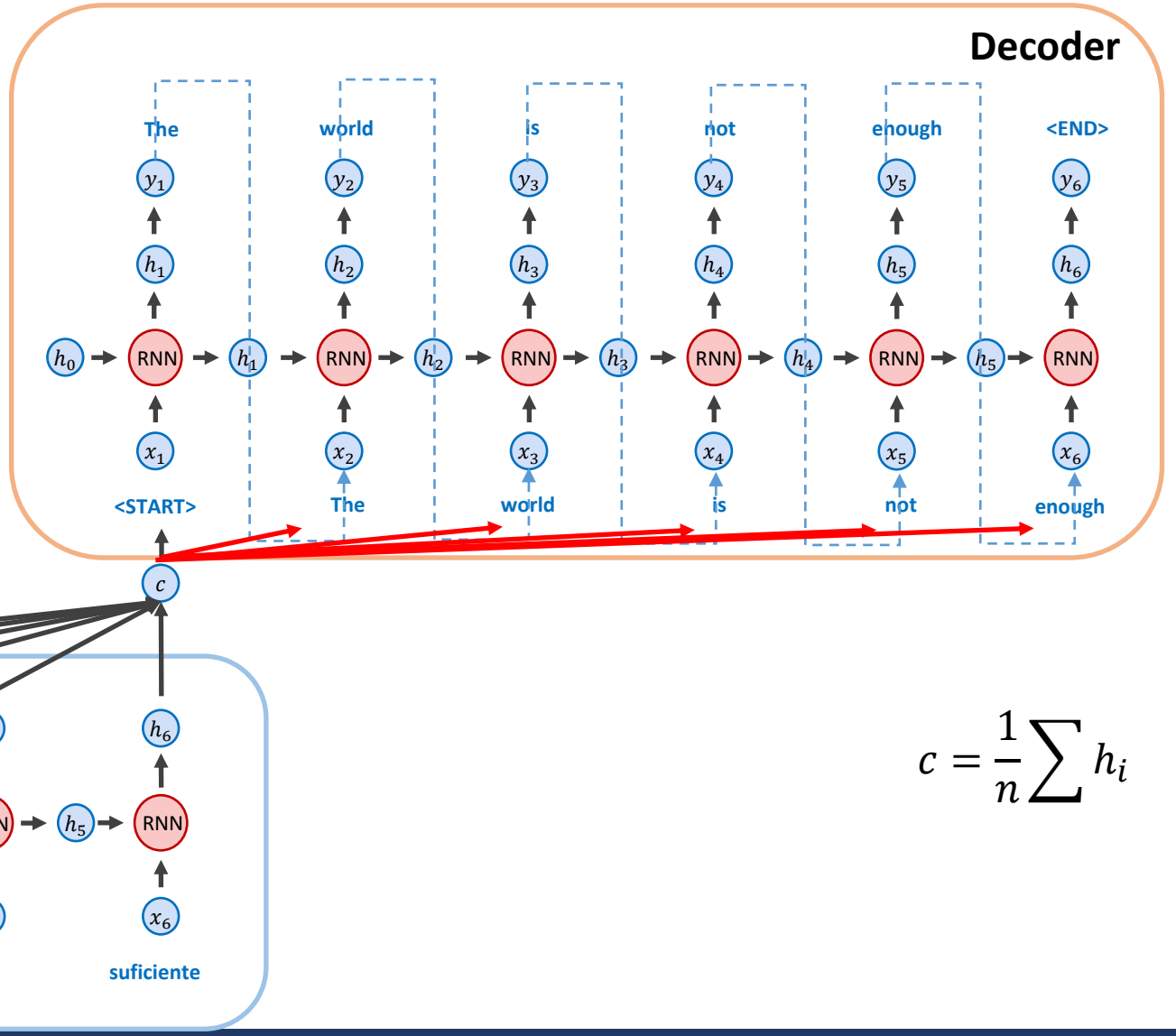


$$c = \frac{1}{n}\sum h_i$$

Attention

아마, 전체 hidden 정보를 디코더에 추가적으로 알려주면 더 좋지 않을까?

➔ 모든 디코더 단계에 알려주면 더 좋겠는데?



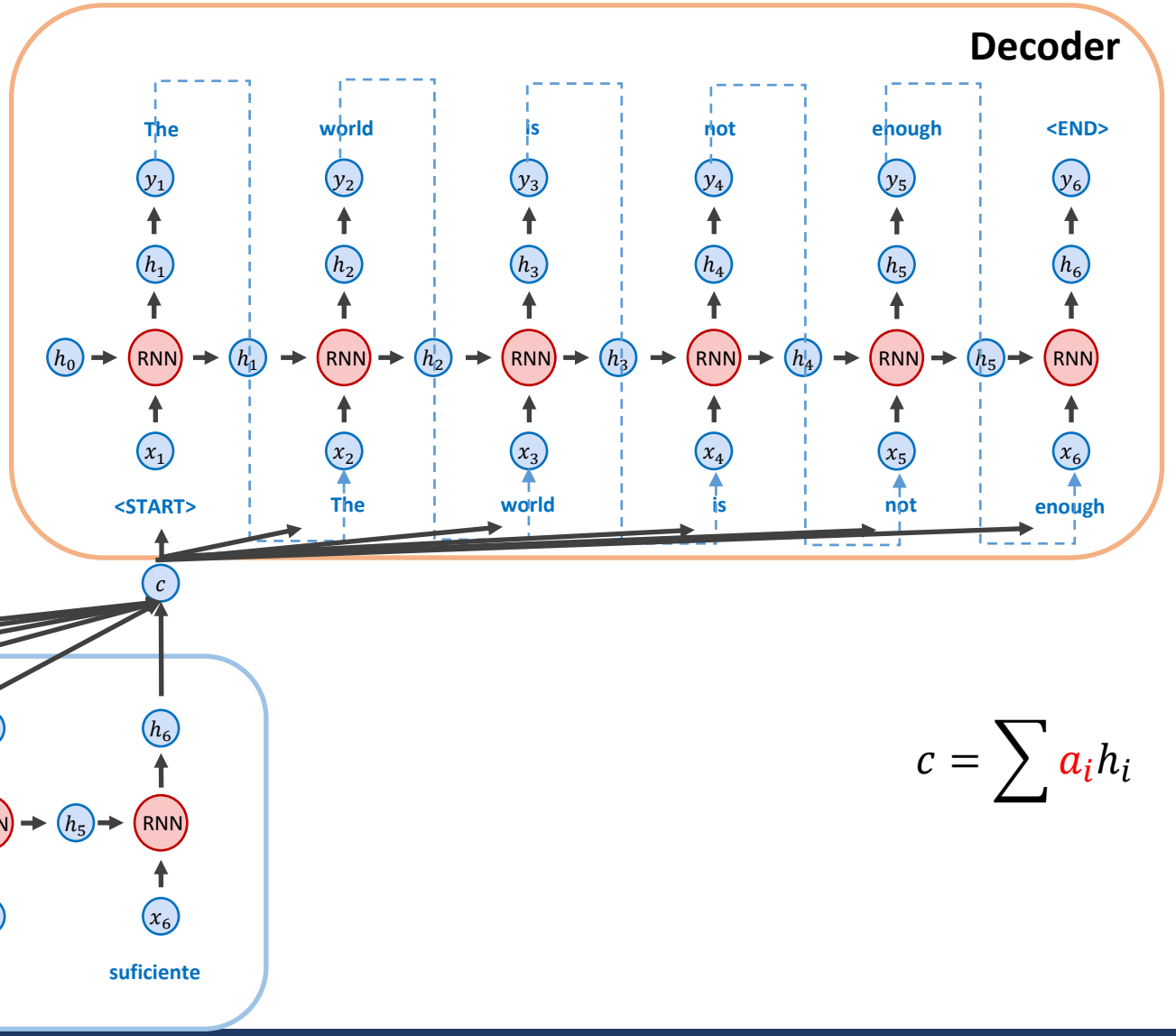
$$c = \frac{1}{n}\sum h_i$$

Attention

아마, 전체 hidden 정보를 디코더에 추가적으로 알려주면 더 좋지 않을까?

➔ 모든 디코더 단계에 알려주면 더 좋겠는데?

➔ 평균보다는 잘 구해진 weight로 합하면 더 좋을 것 같은데?



$$c = \sum a_i h_i$$

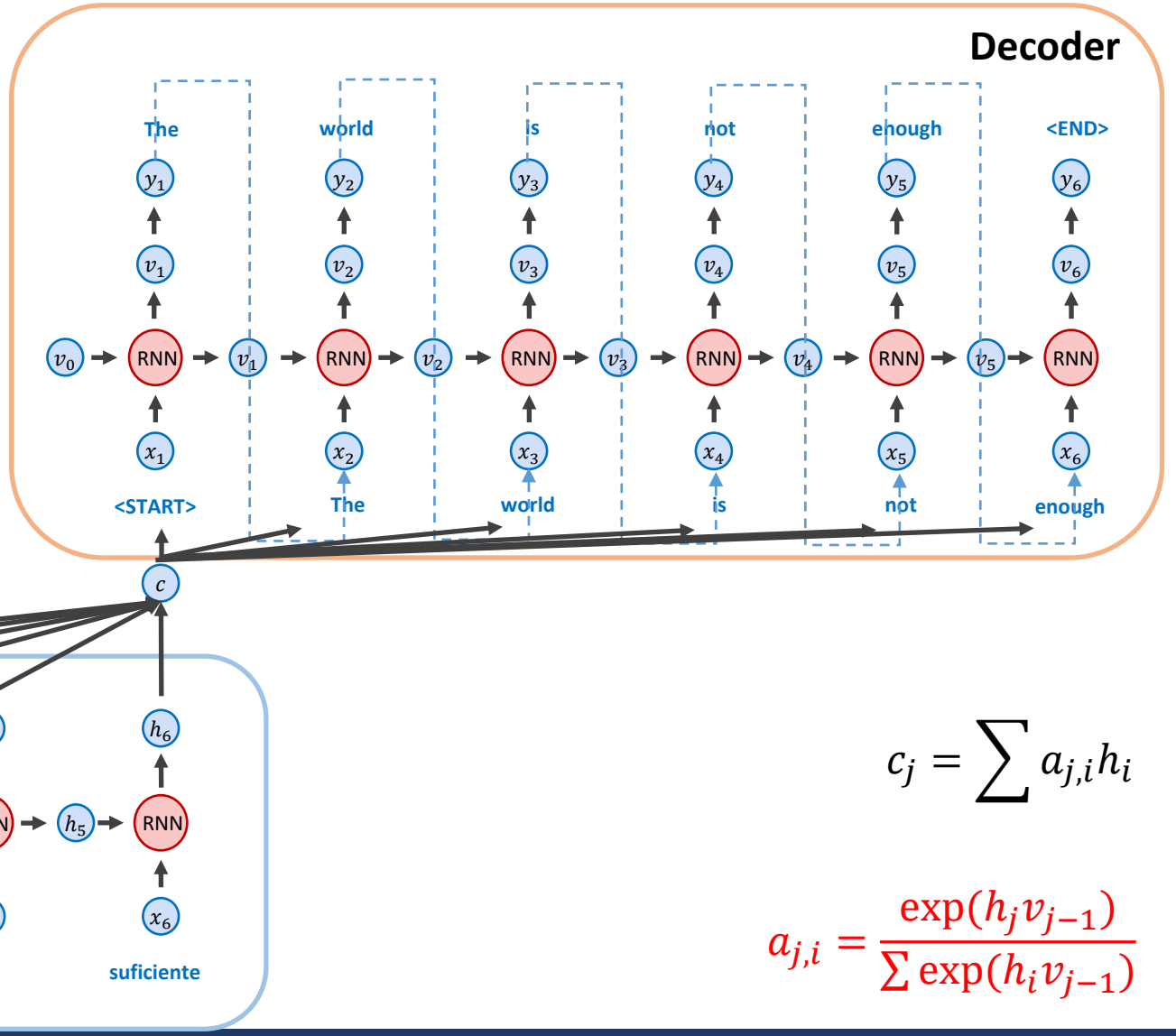
Attention

아마, 전체 hidden 정보를 디코더에 추가적으로 알려주면 더 좋지 않을까?

➔ 모든 디코더 단계에 알려주면 더 좋겠는데?

➔ 평균보다는 잘 구해진 weight로 합하면 더 좋을 것 같은데?

➔ 각 시간별로 weight 가 바뀌면 더 좋겠는데?



$$c_j = \sum a_{j,i} h_i$$

$$a_{j,i} = \frac{\exp(h_j v_{j-1})}{\sum \exp(h_i v_{j-1})}$$

Attention

Attention at time step 4

Attention

Encoder

Transformer

RNN + Attention

순차적으로 입력하니 느리다.

병렬화

어떻게?

Self-Attention

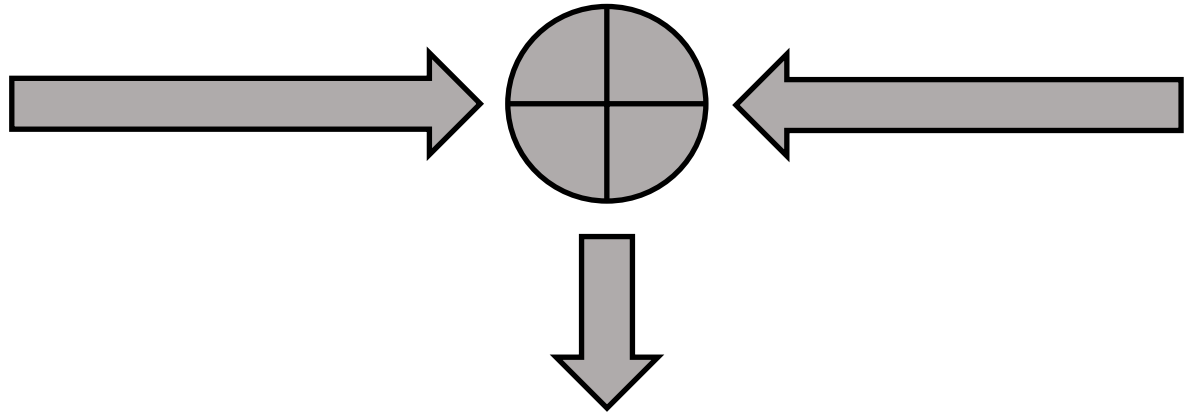
Word2Vec

문맥에 따라 바뀌면 좋겠다.

주변 정보도 보자!

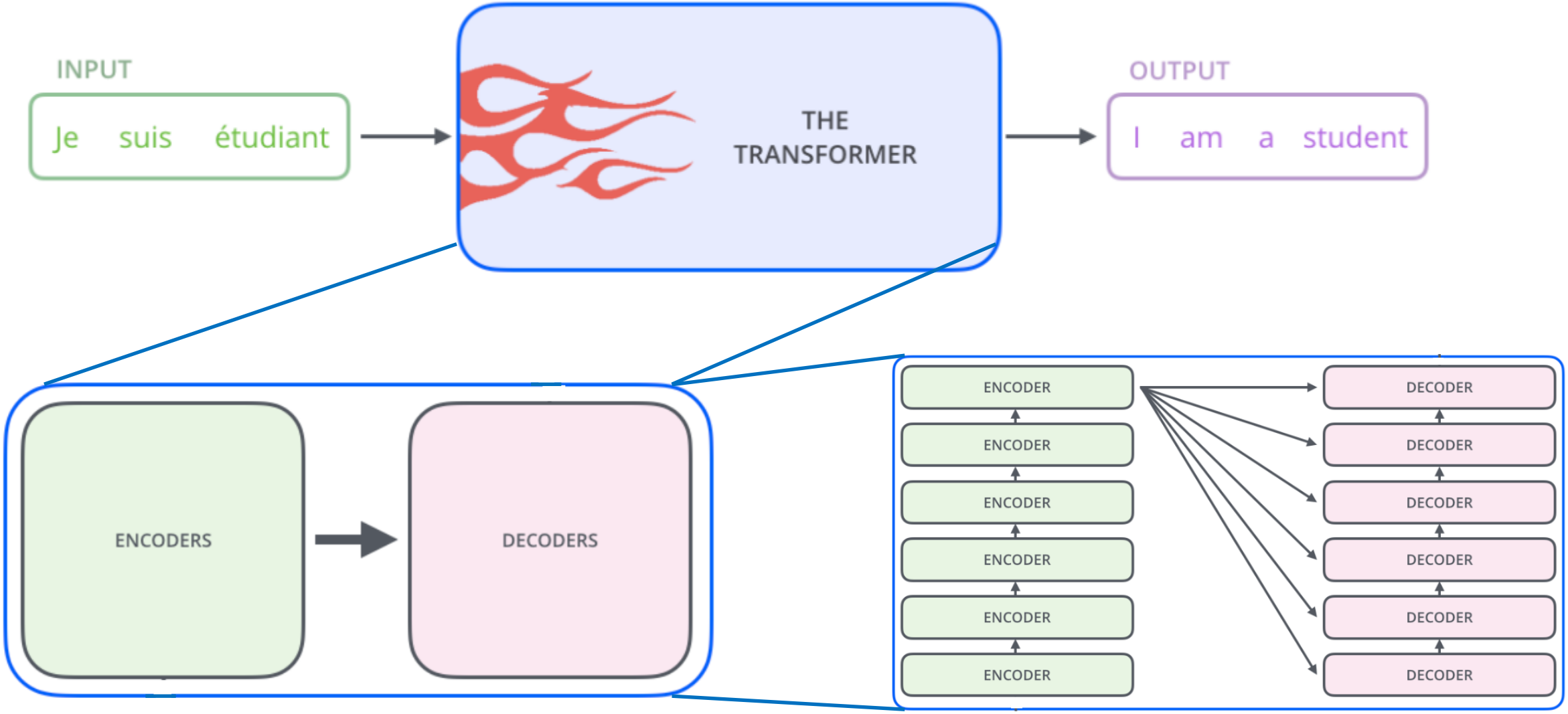
순서는?

Positional Embedding

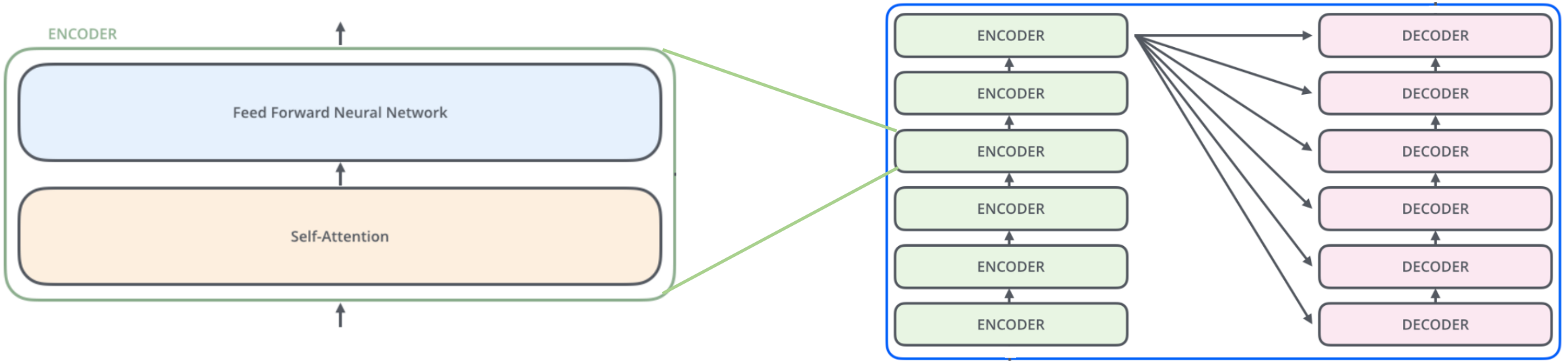


Transformer

Transformer



Encoder

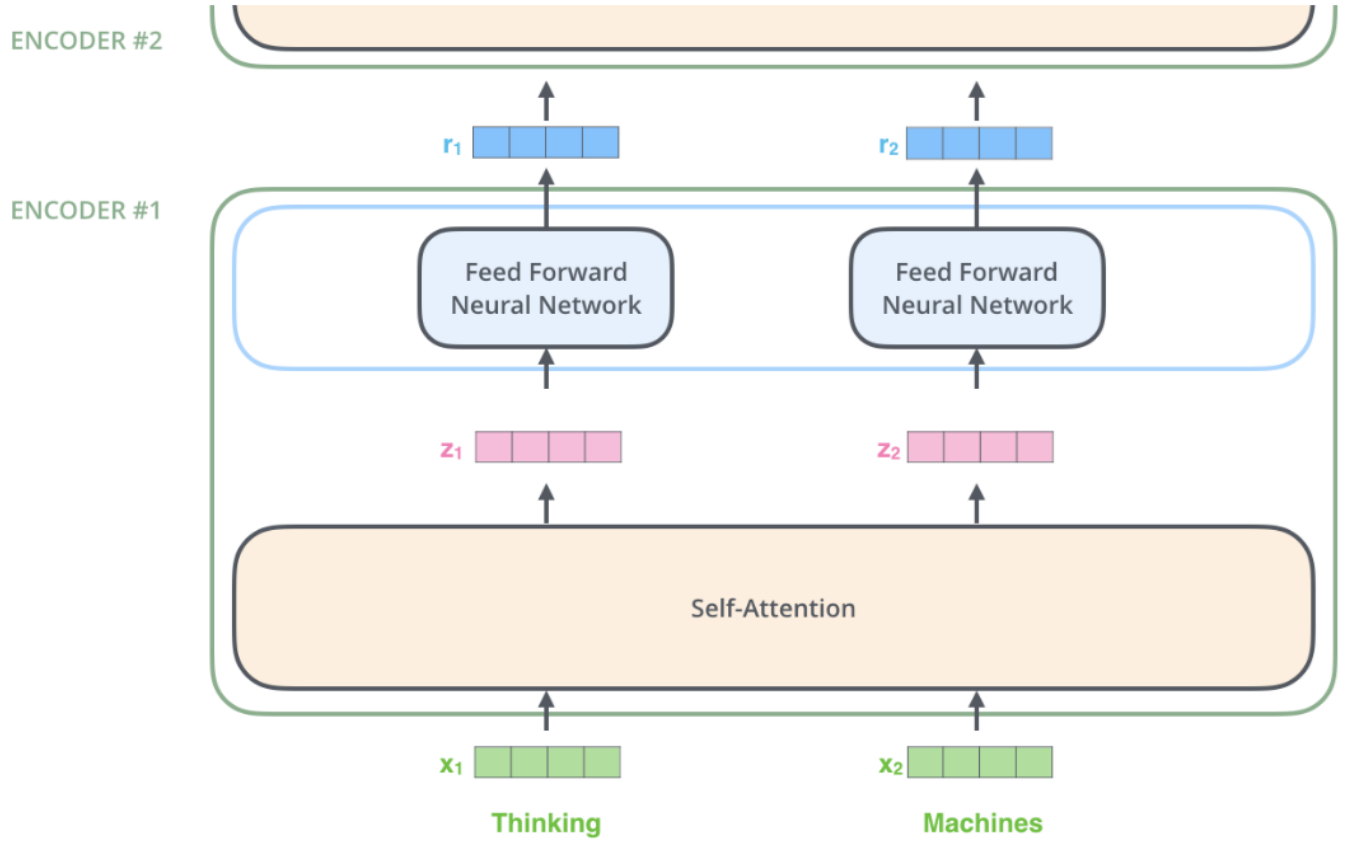


동일한 구조의 encoder 반복

두 개의 sublayer로 구성: Self-Attention Layer + Feed Forward Neural Network Layer

각 encoder 는 weight 공유 X

Encoder

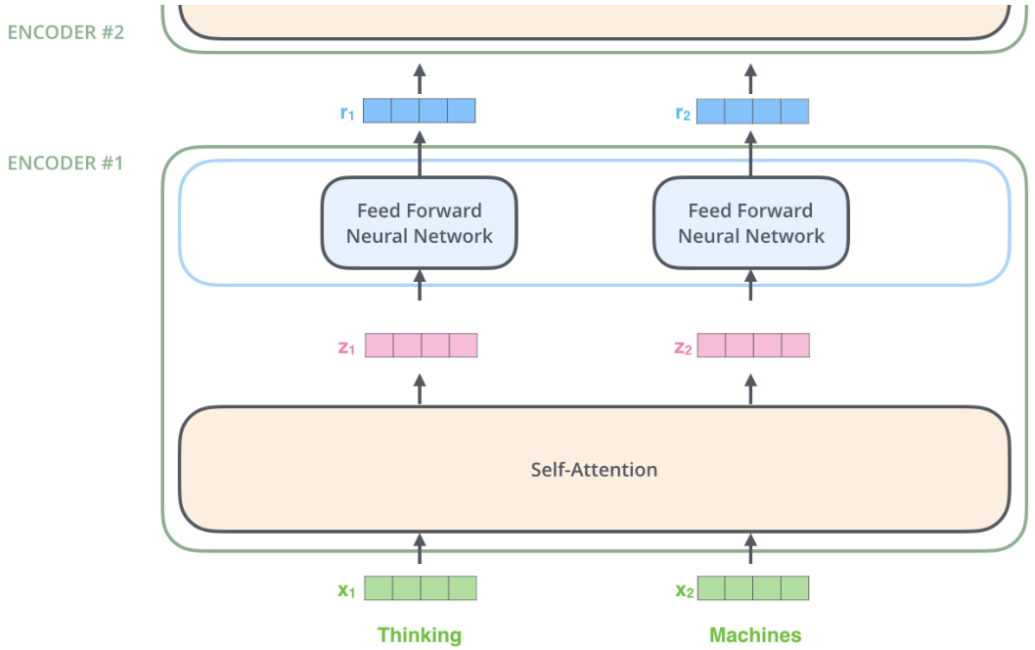


첫 번째 인코더 입력: 임베딩 벡터

(N + 1) 번째 인코더 입력: N 번째 인코더에 의해 만들어진 결과 vector

마지막 인코더 출력: 각 디코더로 전달

Encoder



Attention layer

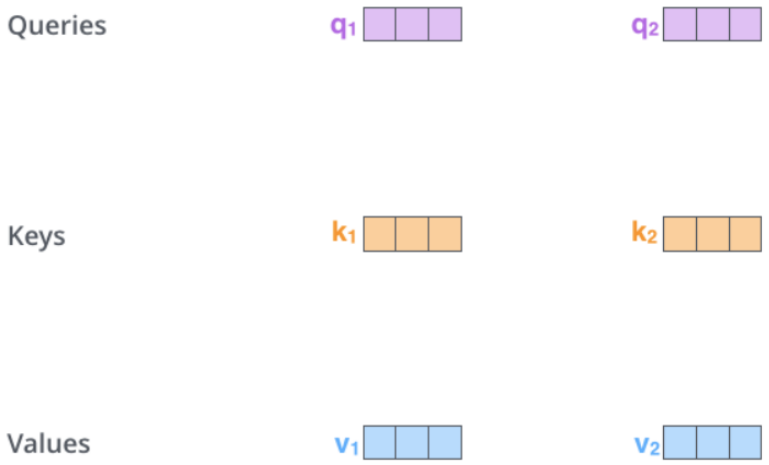
. 입력 벡터를 모두 활용 ➔ 문맥 이해 (장거리 의존성 파악)

Feed Forward Neural Network

. 자기에게 주어진 벡터만 처리 ➔ 비선형 변환 (복잡한 특징 추출가능) 및 특징 추출

. 다음 레이어로 전달할 정보를 전달하기 위해 정제

Encoder – Multi-head Self-Attention



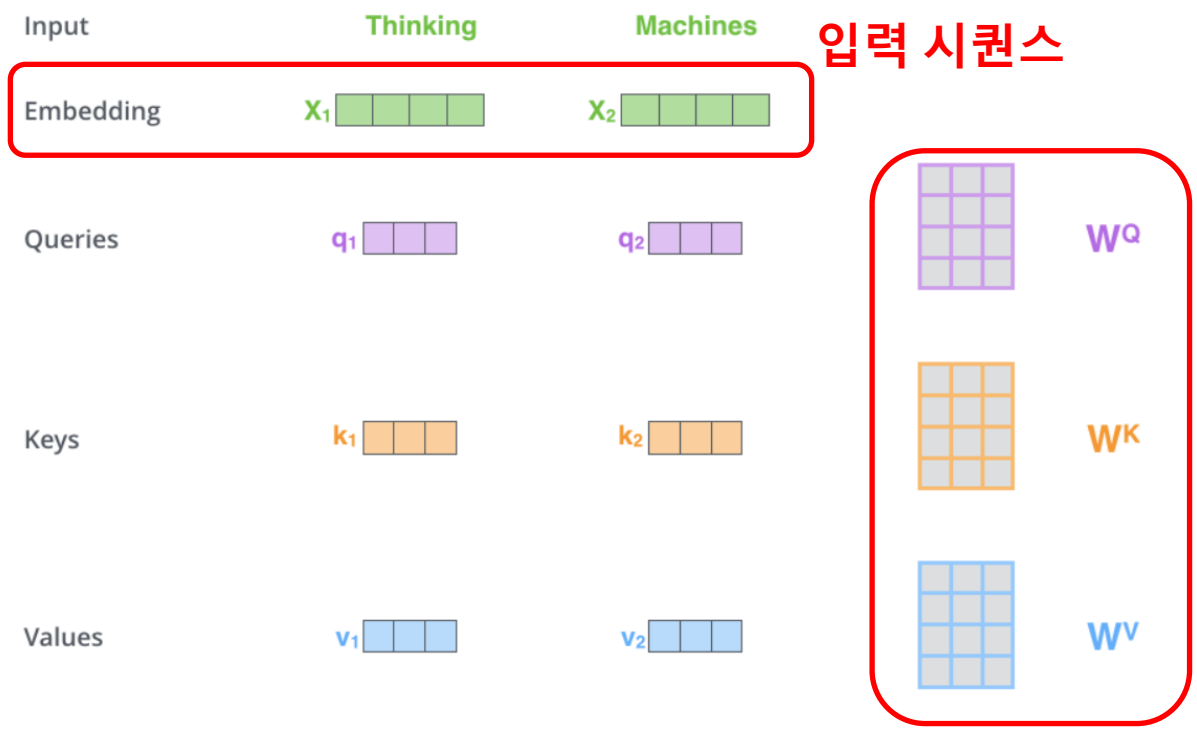
Query: 원하는 정보를 찾기 위한 검색어

Key: 가지고 있는 정보의 요약 (제목)

Value: Key(제목, 검색어)에 해당하는 실제 정보, 데이터

➔ 새로운 정보를 생성할 때,
검색어(Query)와 제목(Key)의 유사성을 계산하고,
이를 활용하여 기존 정보(Value)들을 잘 취합하자!!

Encoder – Multi-head Self-Attention

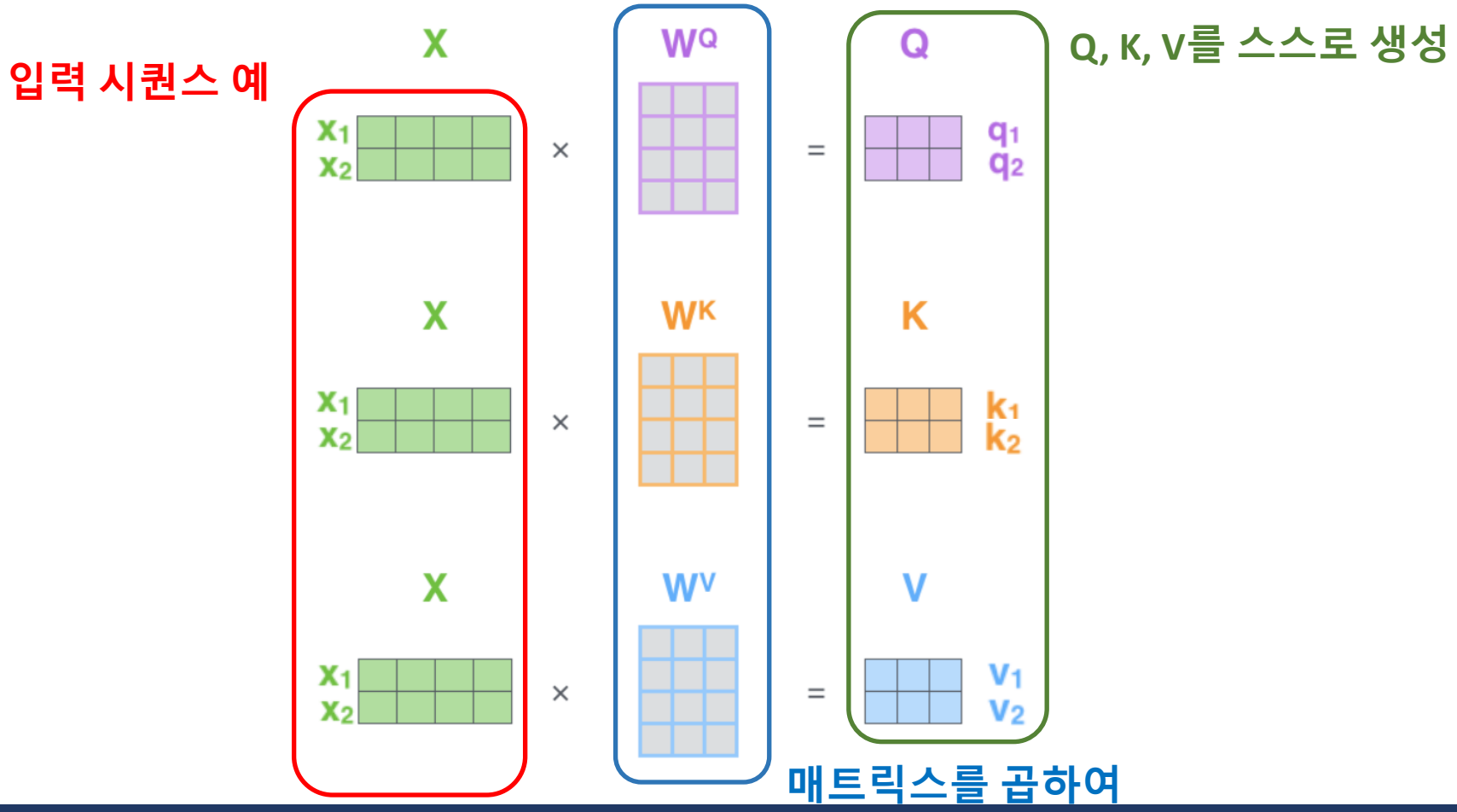


입력된 정보를 기반으로 스스로를 관계를 파악하고, 이를 활용하여 자기 자신의 표현을 개선한다.

➔ 입력 시퀀스를 기반으로 Query, Key, Value를 스스로 생성한다.

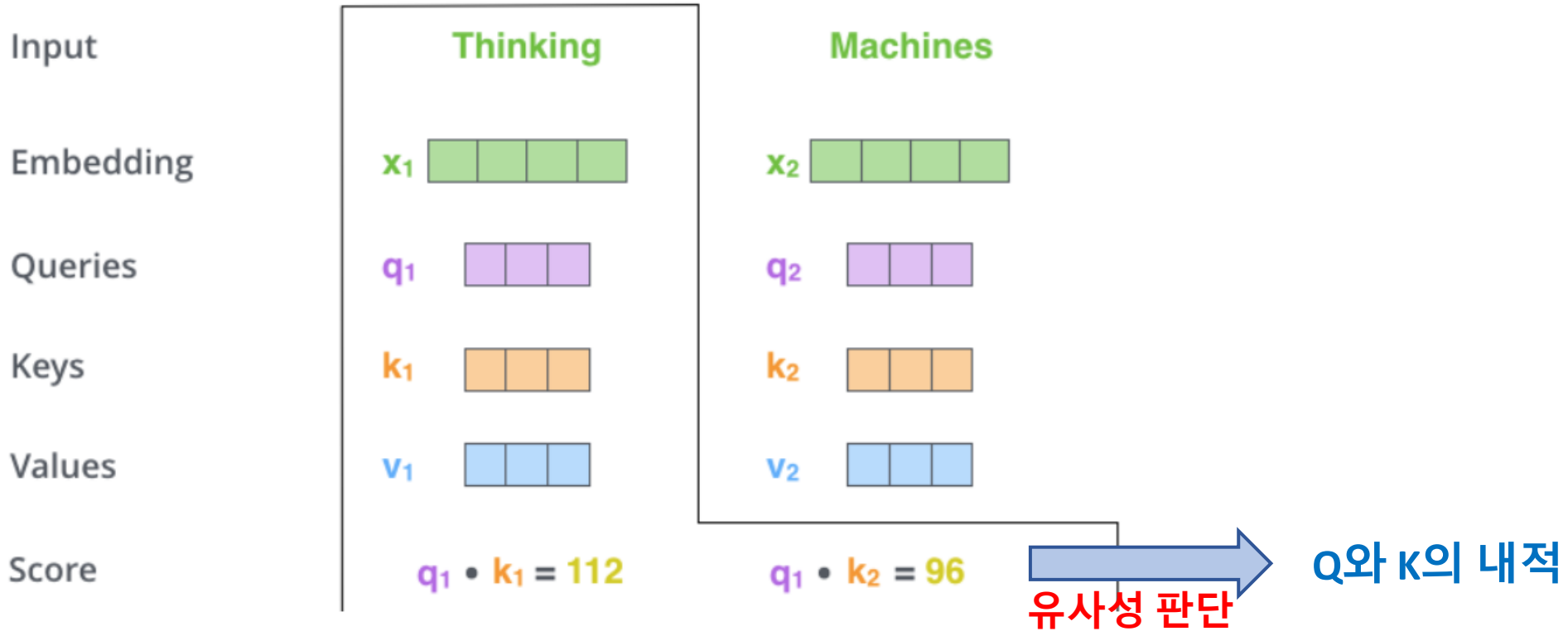
Encoder – Multi-head Self-Attention

입력 시퀀스를 기반으로 Query, Key, Value를 스스로 생성한다.



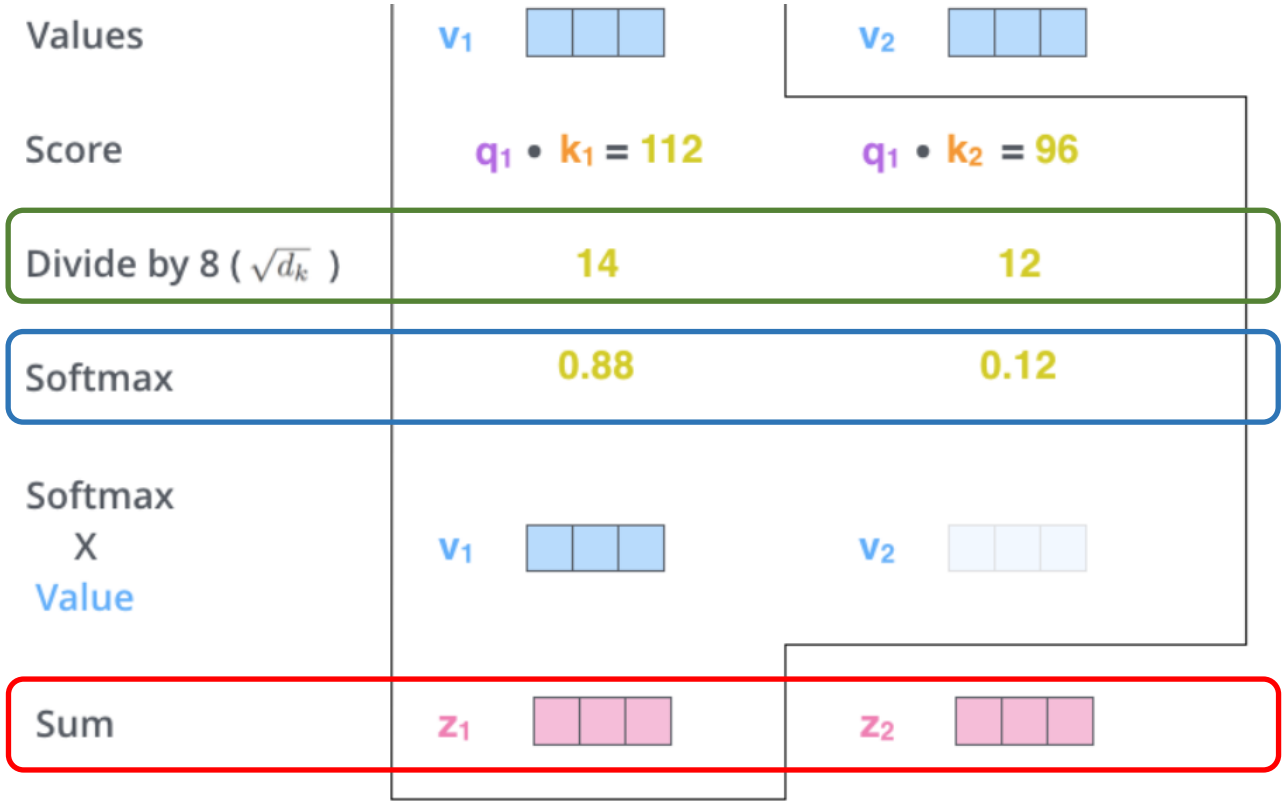
Encoder – Multi-head Self-Attention

새로운 정보를 생성할 때,
검색어(Query)와 제목(Key)의 유사성을 계산하고,
이를 활용하여 기존 정보(Value)들을 잘 취합하자!!



Encoder – Multi-head Self-Attention

새로운 정보를 생성할 때,
검색어(Query)와 제목(Key)의 유사성을 계산하고,
이를 활용하여 기존 정보(Value)들을 잘 취합하자!!



계산된 유사도

유사도 정규화

유사도를 확률화

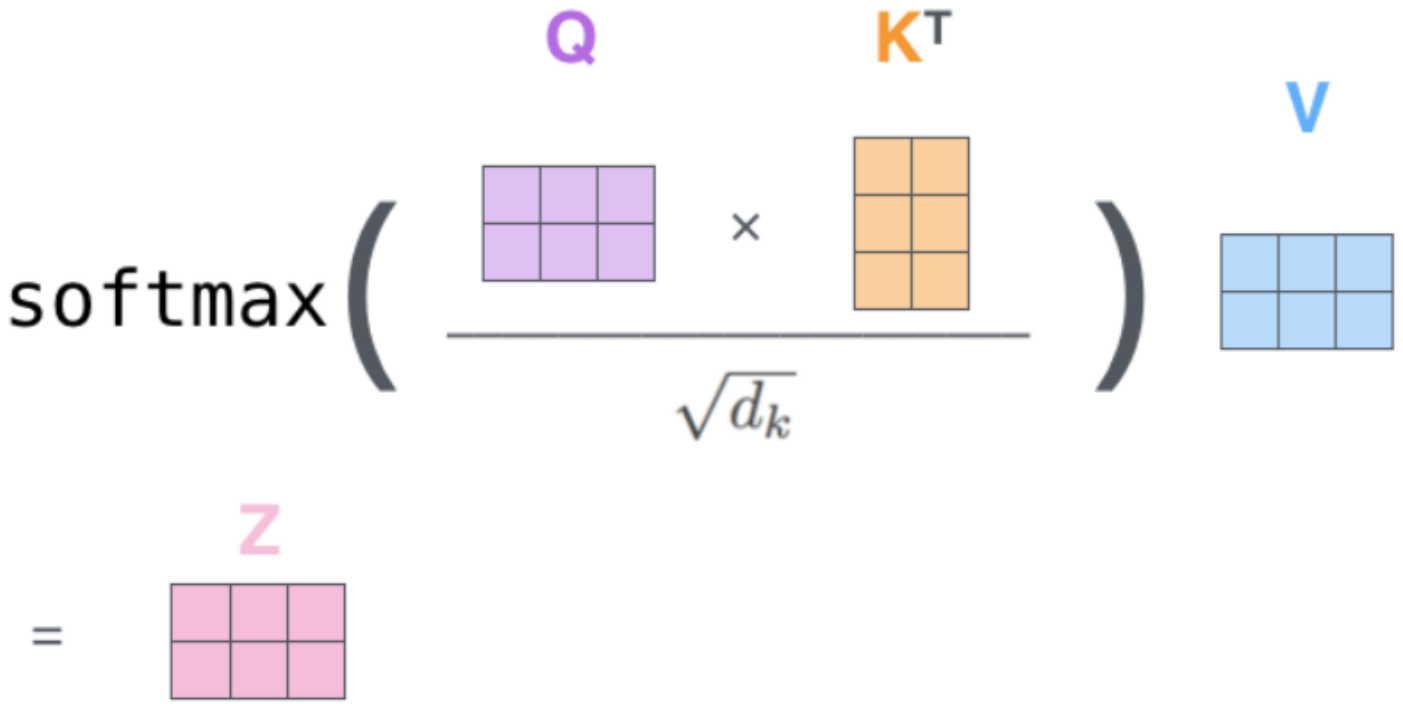
기존의 value를 weighted sum

d_k : key vector의 차원

본 예제 그림에서는 3이지만,
Transformer 에서는 64
➔ $\sqrt{d_k}$ = 8

Encoder – Multi-head Self-Attention

행렬 형태로 표현한 self-attention 계산



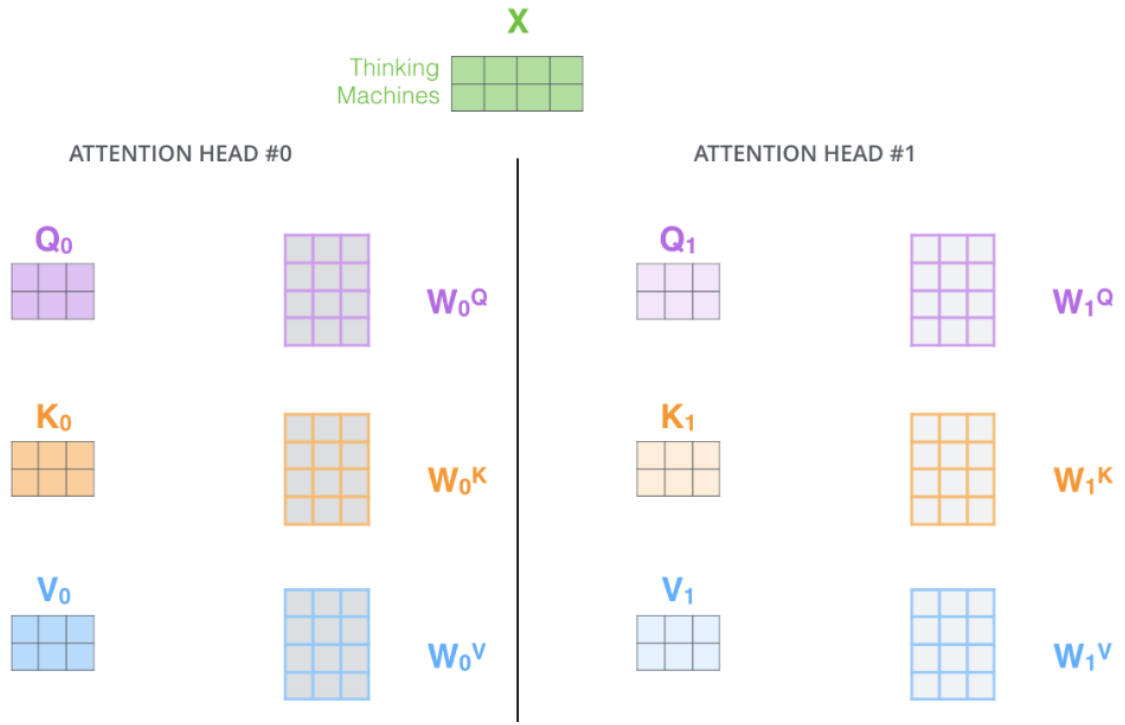
Encoder – Multi-head Self-Attention

야마타노오로치:
일본 신화에 나오는 꼬리와 머리가
여덟 개 달렸다고 하는 거대한 용

Head: 한 가지 시야·관점 ➔ 한 가지 측면으로 분석

Multi-head: 문맥을 여러 관점에서 분석 ➔ 강력해 짐

Encoder – Multi-head Self-Attention

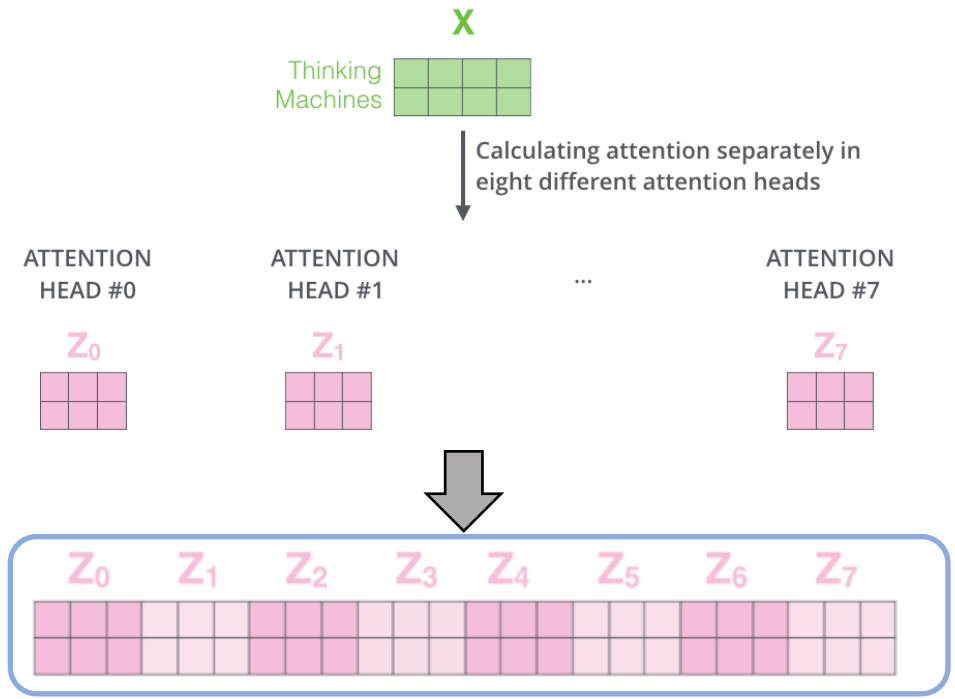


하나의 attention head는 각 token 별로 하나의 vector (representation) 생성

Multi-head attention: 여러 개의 vector 표현 ➔ 여러 개의 representation 생성 가능

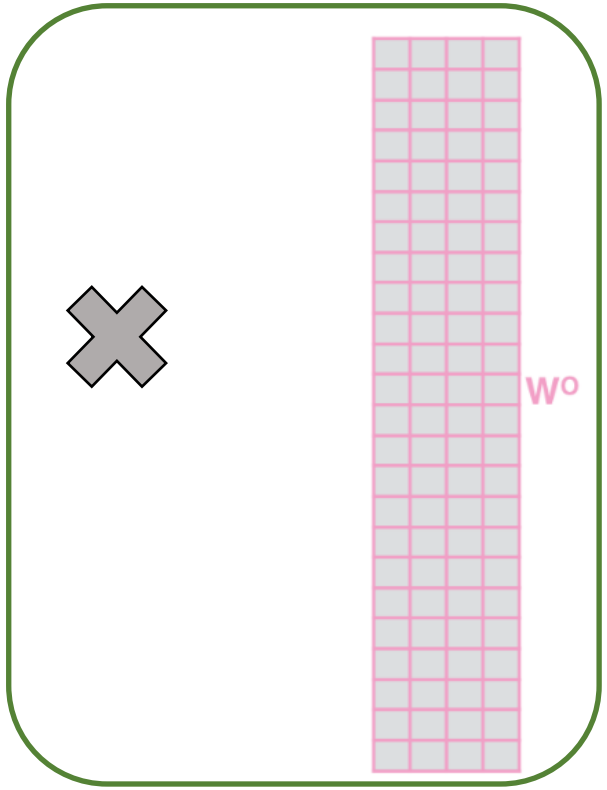
논문에서는 8개의 attention head를 가짐 / 서로 다른 표현에 맞는 Weight 각각 생성

Encoder – Multi-head Self-Attention

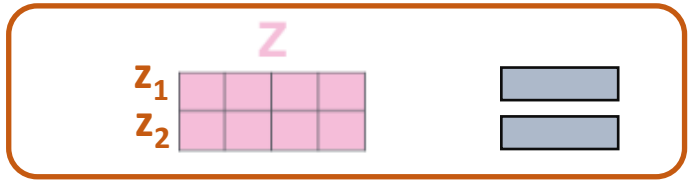


1. 생성된 attention head들을 concatenate

2. 학습한 weight 와 matrix 곱하기



3. FFNN으로 전달할 vector들 (matrix) 생성



Encoder – Multi-head Self-Attention

1) This is our input sentence*

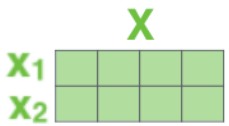
2) We embed each word*

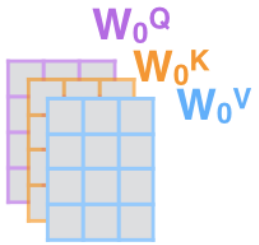
3) Split into 8 heads. We multiply X or R with weight matrices

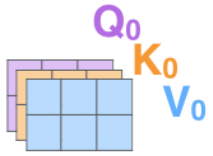
4) Calculate attention using the resulting Q/K/V matrices

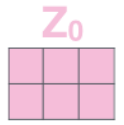
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

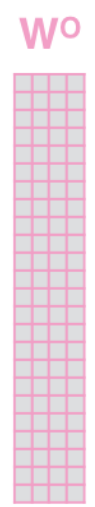
Thinking Machines

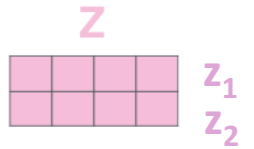




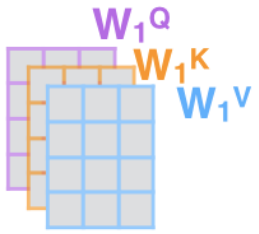


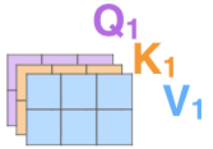


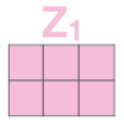


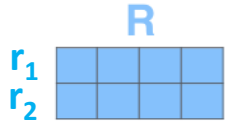


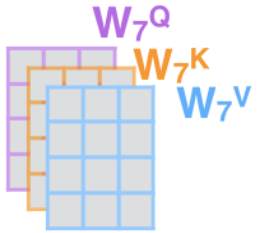
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

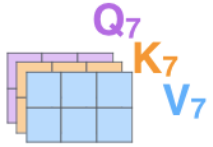


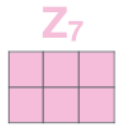




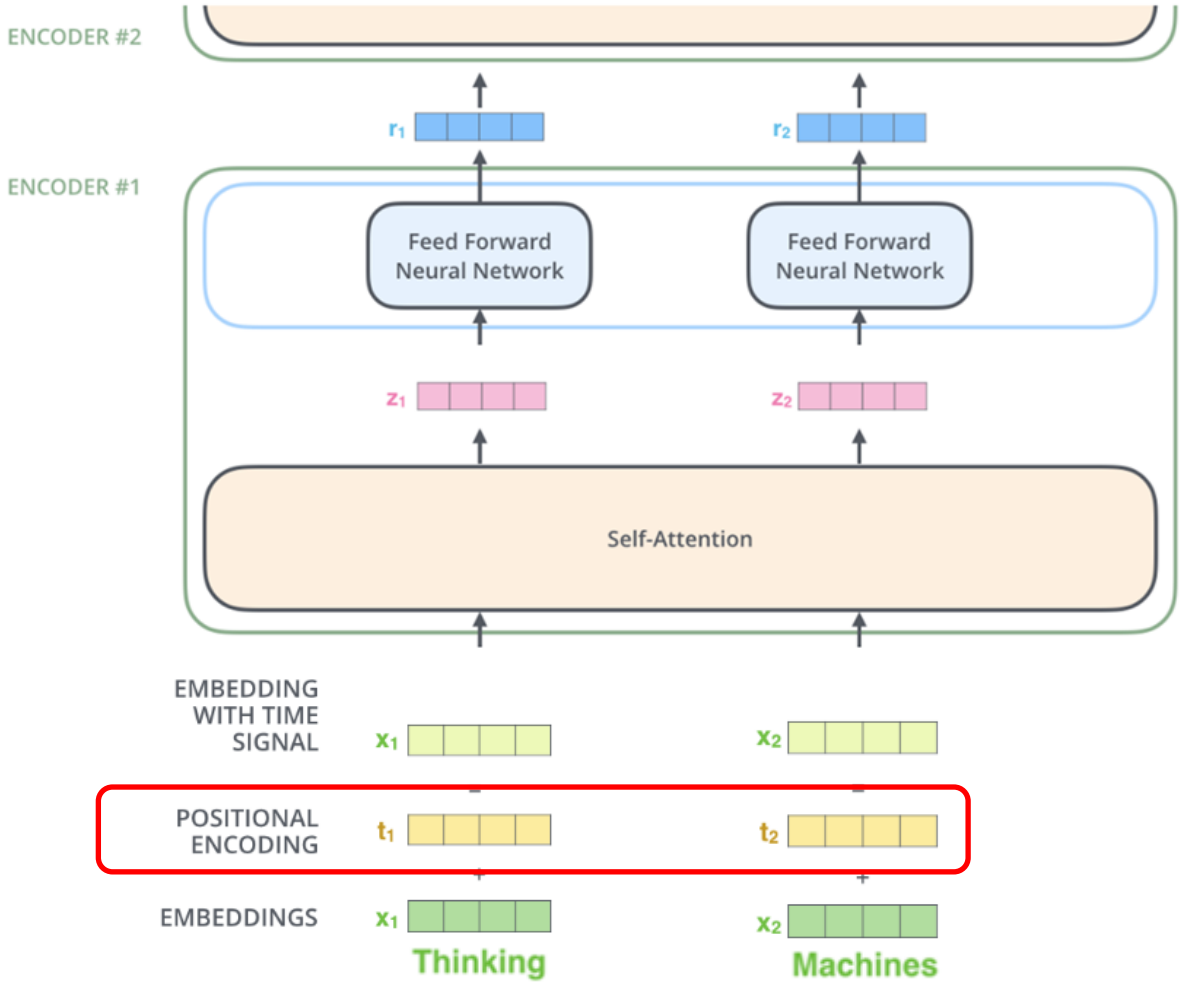








Encoder – Positional Embedding



Self-Attention

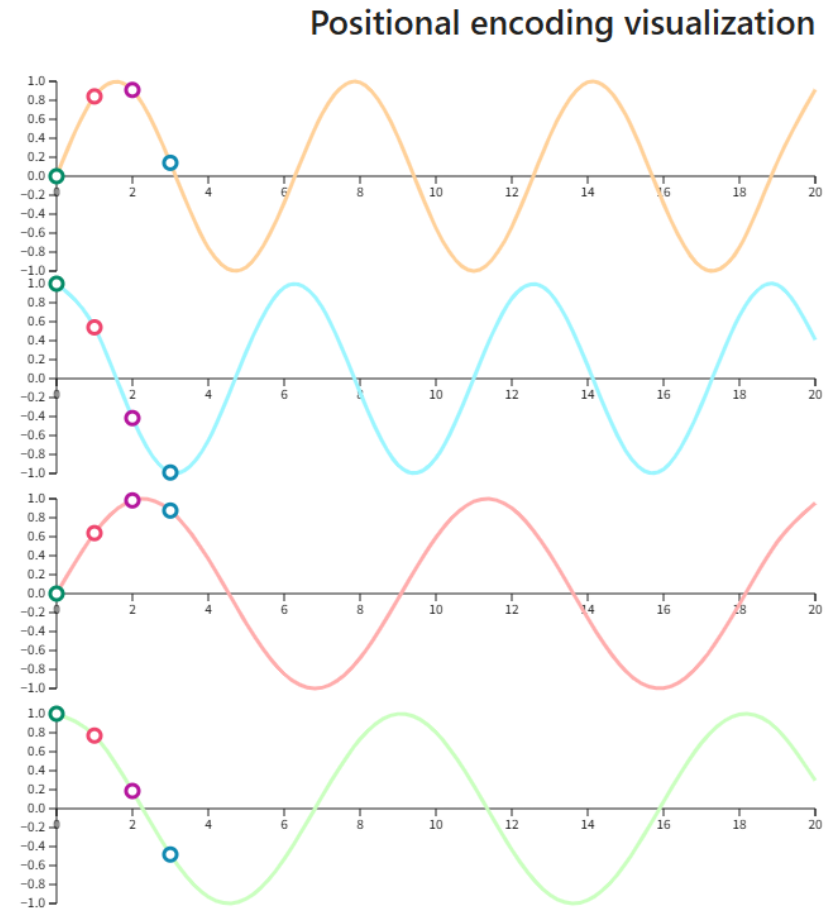
Self-Attention 은 병렬화가 목적
➔ 순서가 상관 없어짐

순서 정보도 문맥 이해에 중요한 요소

입력시 순서 정보도 추가 해 주자!

Positional Embedding 을 통해 순서 정보 추가

Encoder – Positional Embedding



몇 번째 token 이냐? (pos)

p0	p1	p2	p3	
0.000	0.841	0.909	0.141	i=0
1.000	0.540	-0.416	-0.990	i=1
0.000	0.638	0.983	0.875	i=2
1.000	0.770	0.186	-0.484	i=3

해당 token의 몇 번째 값이냐? (2i, 2i+1)

Positional Encoding

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

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

Settings: d = 50

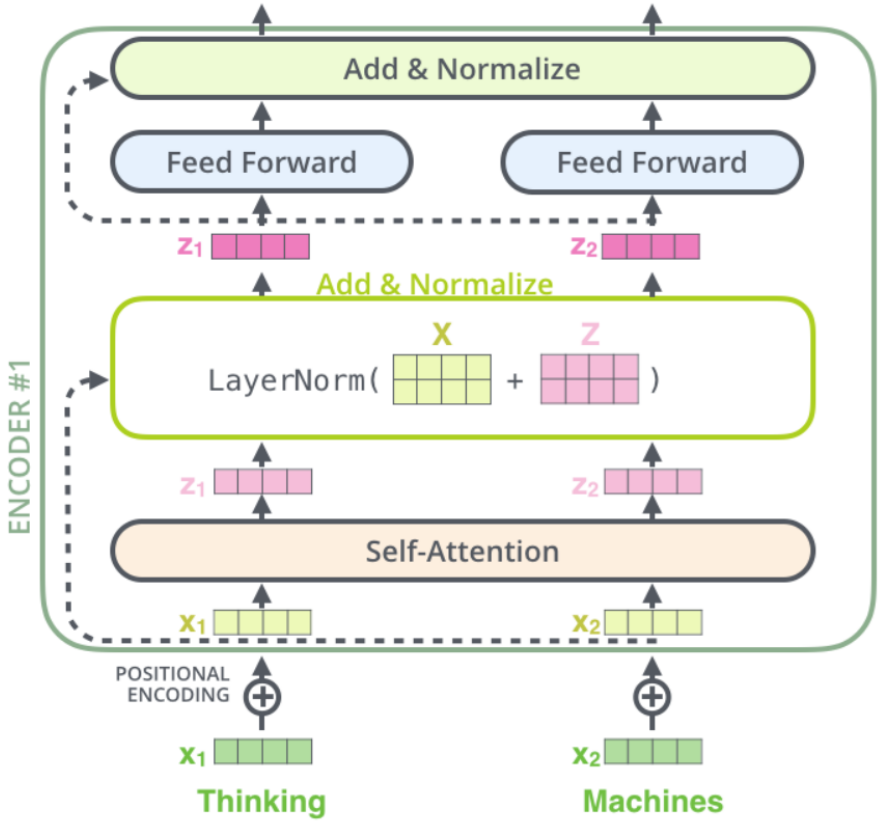
The value of each positional encoding depends on the *position* (*pos*) and *dimension* (*d*). We calculate result for every *index* (*i*) to get the whole vector.

2 번째 토큰의 2 번째 원소 값:

$$\boldsymbol{PE_{(2,2\times1)}} = \sin\left(\frac{2}{10000^{\frac{2\times1}{50}}}\right) = \mathbf{0.983}$$

논문에서는 d_{model} = 512

Encoder – Residual Connection

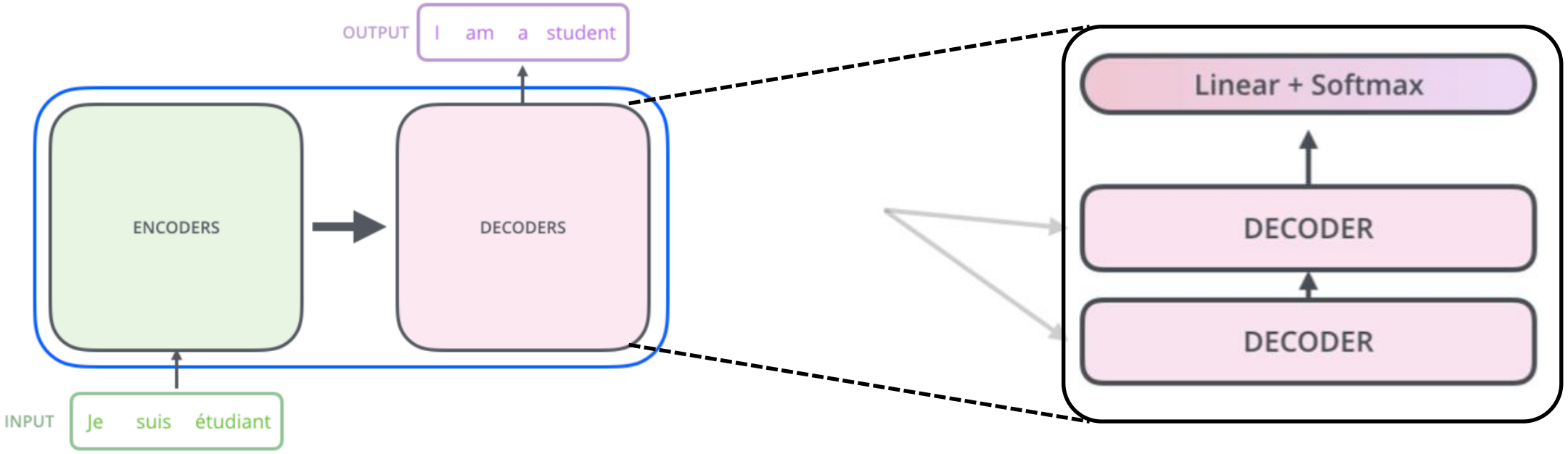


각 encoder 내의 sub-layer: residual connection으로 연결 + layer-normalization

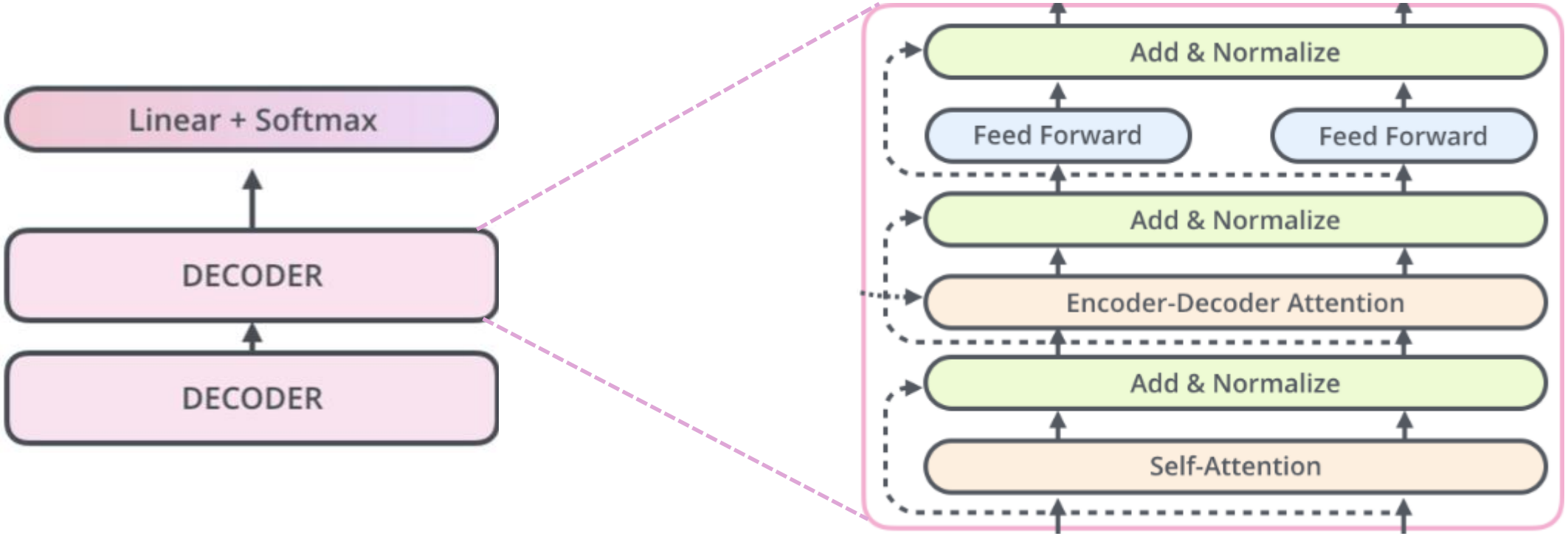
Residual connection: 심층 신경망의 학습 안정성 보장

Layer-normalization: 각 Token별로 임베딩 값들을 정규화 (평균: 0, 분산: 1)

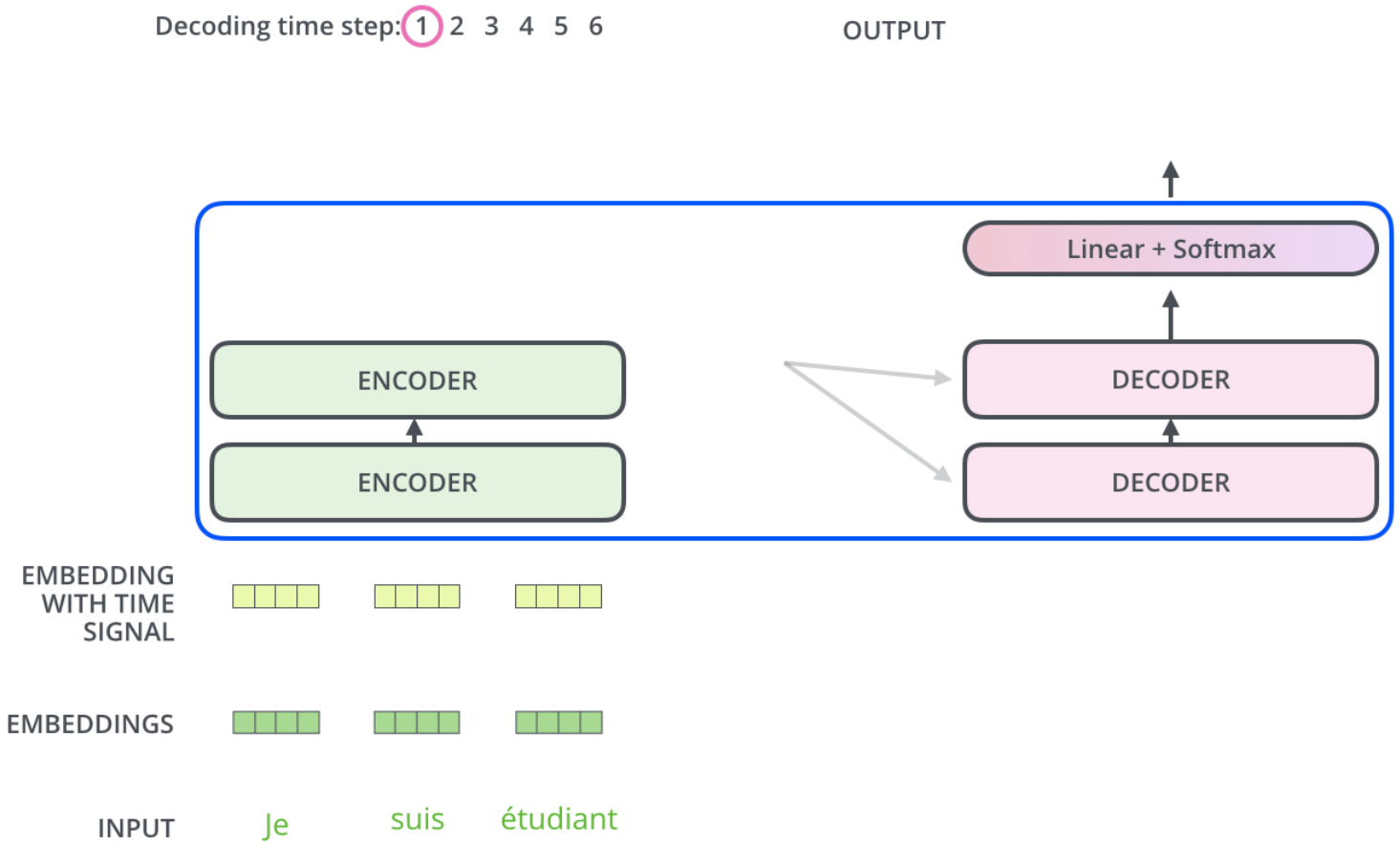
Decoder


OUTPUT
I am a student
ENCODERS
DECODERS
INPUT
Je suis étudiant
Linear + Softmax
DECODER
DECODER

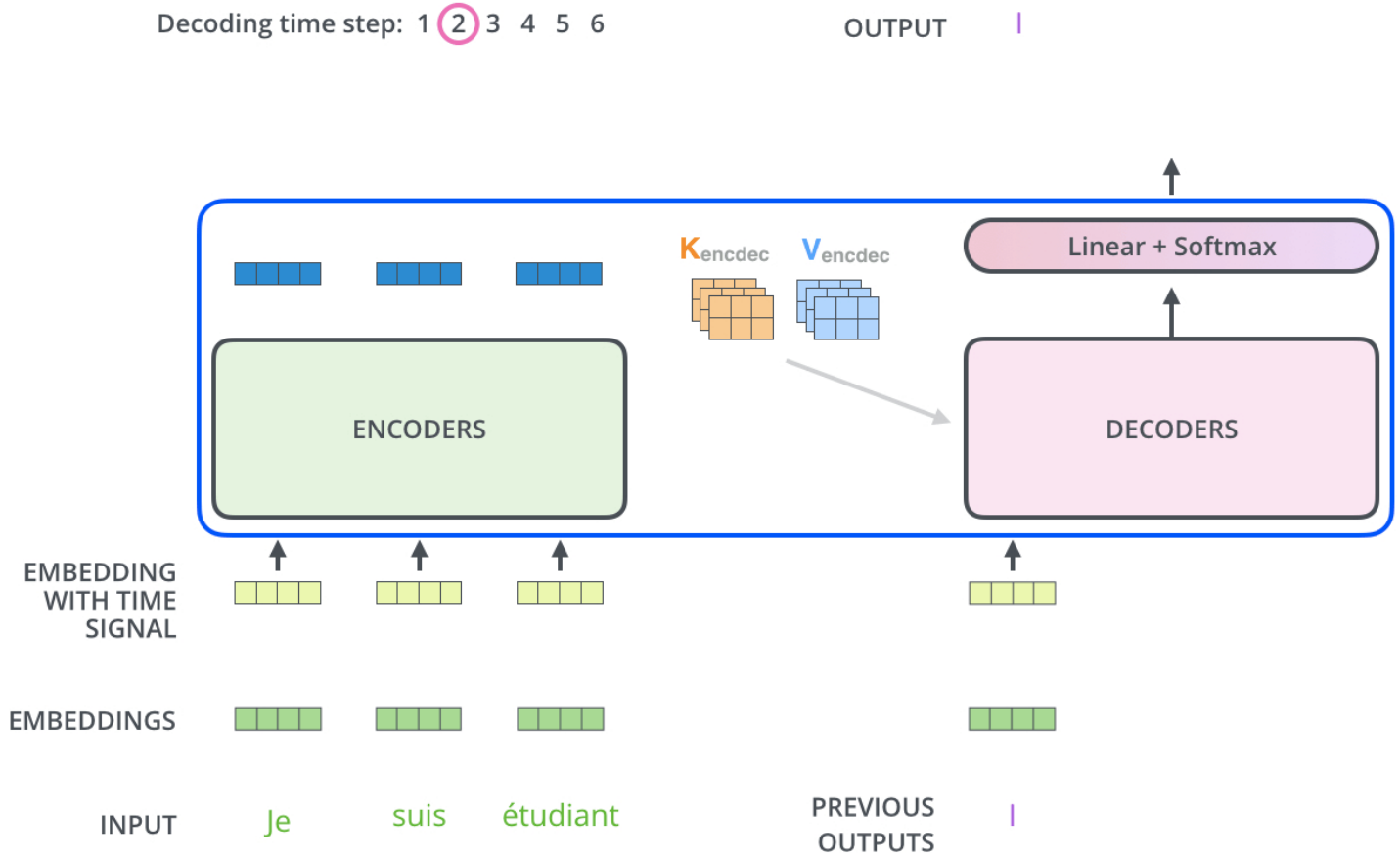
Decoder


Linear + Softmax
DECODER
DECODER
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention

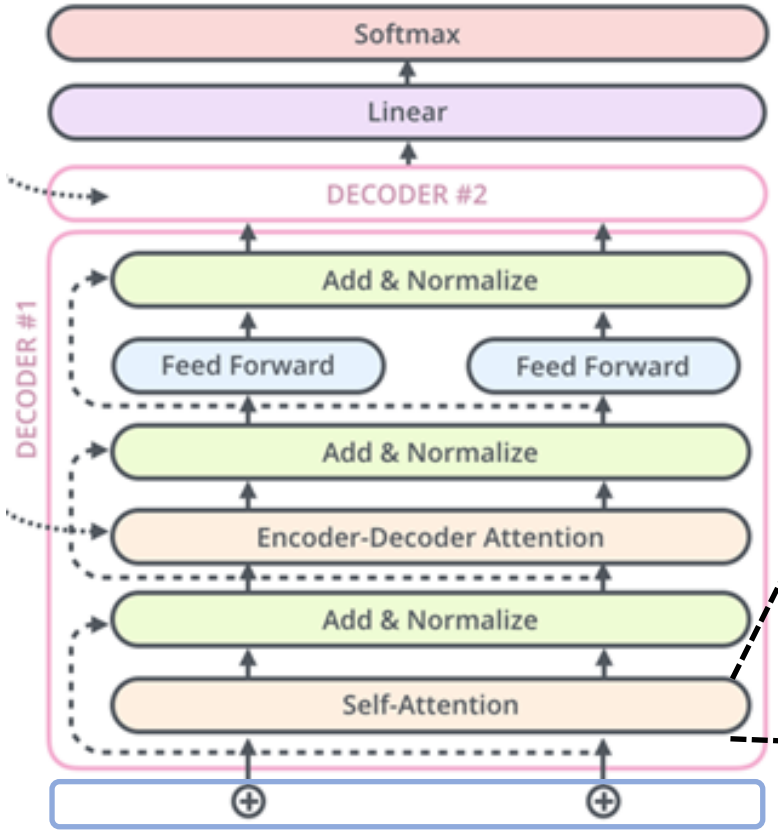
Decoder


Decoding time step: 1 2 3 4 5 6
OUTPUT
Linear + Softmax
ENCODER
ENCODER
DECODER
DECODER
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant

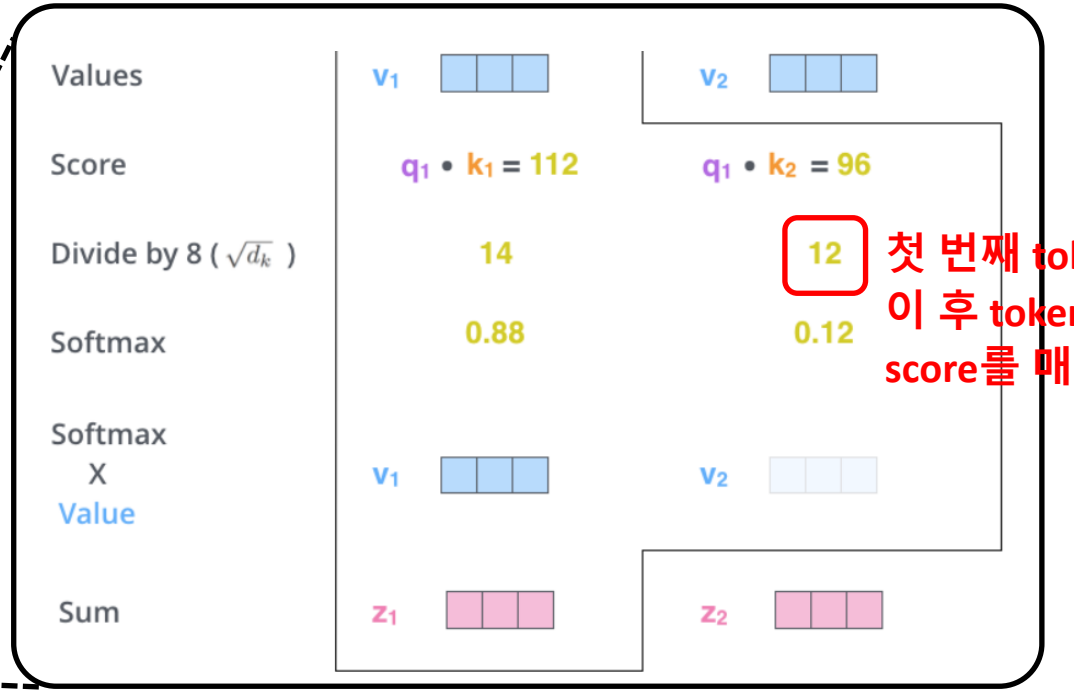
Decoder


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Linear + Softmax
K encdec
V encdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Decoder – Masked Decoder Multi-head Self-Attention



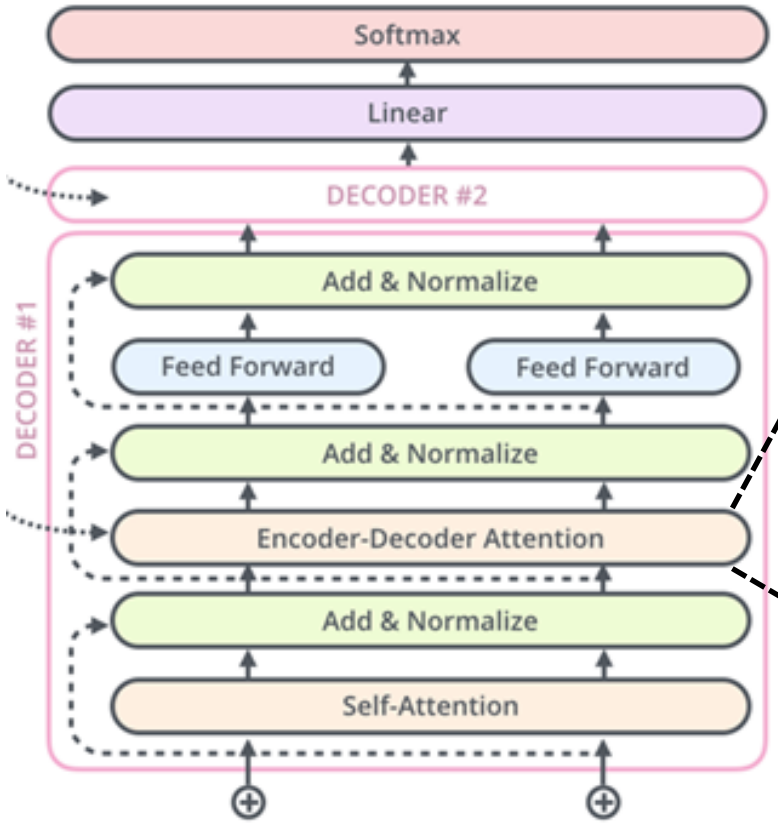
Positional Embedding: 위치 정보 추가

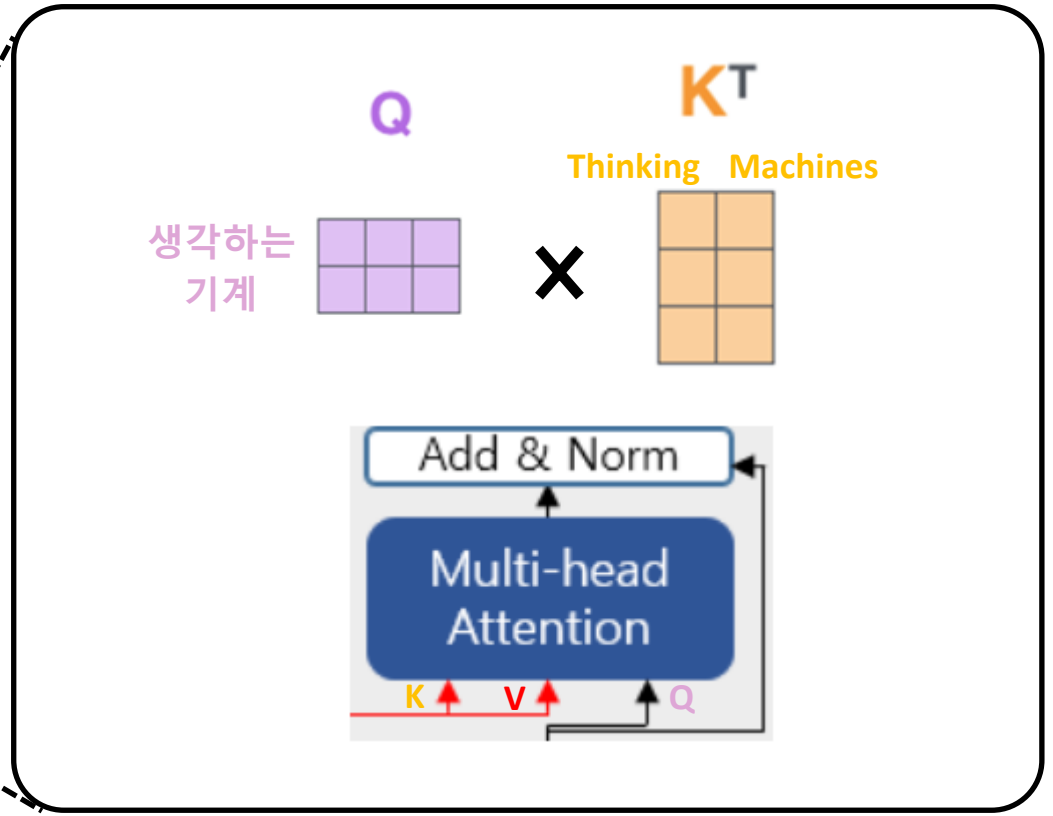


첫 번째 token 처리시,
이 후 token (두 번째, 세 번째)
score를 매우 작은 음수로 처리

디코더는 한 token씩 해석 해 나가는 과정
➔ 현재 처리하는 token 이후의 정보는 무의미 함
➔ Look-ahead mask 도입

Decoder – Encoder-Decoder Multi-head Attention





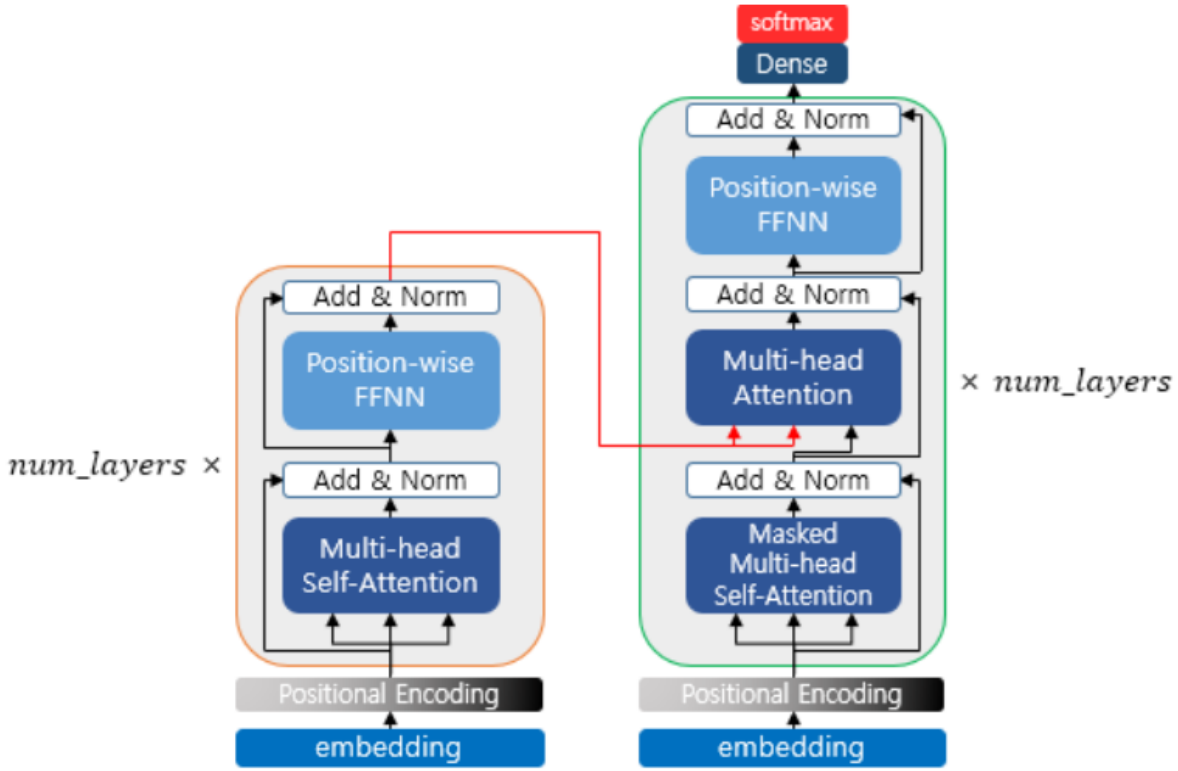
디코더: 문장 전체의 내용을 기반으로 하나씩 해석해 가는 과정

➔ Key, Value: 문장 전체에 대한 검색어 및 내용

➔ Query: 현재까지 해석한 내용 정보

➔ Key, Value: From Encoder / Query: From Decoder

Decoder – Encoder-Decoder Multi-head Attention

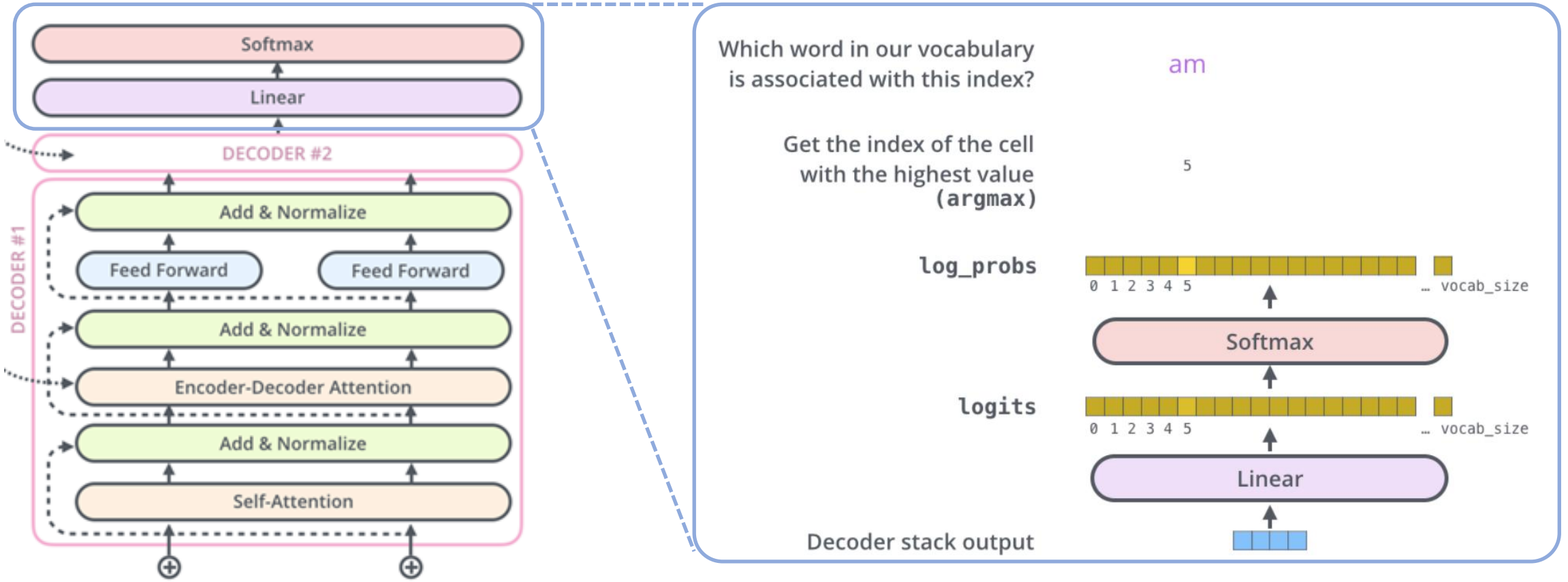


인코더의 셀프 어텐션 : Query = **Key** = Value

디코더의 마스크드 셀프 어텐션 : Query = **Key** = Value

디코더의 인코더-디코더 어텐션 : Query : 디코더 벡터 / **Key** = Value : 인코더 벡터

Decoder – Linear & Softmax



Decoder의 출력

➔ Linear layer: 토큰 벡터를 단어장에 projection 및 각 단어가 될 점수 (차원확장)

➔ Softmax layer: 점수를 각 단어가 될 확률화

Transformer – 실습

The Annotated Transformer