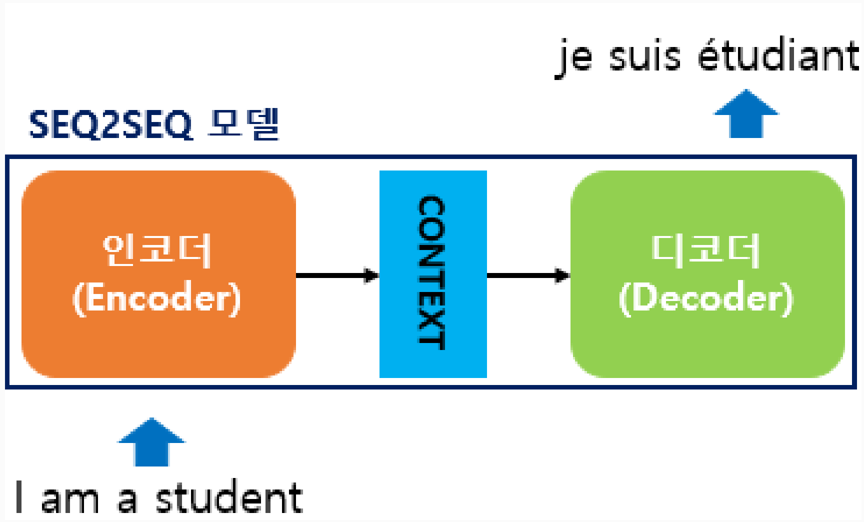
ATTENTION & TRANSFORMER

정인호

Sequence To Sequence

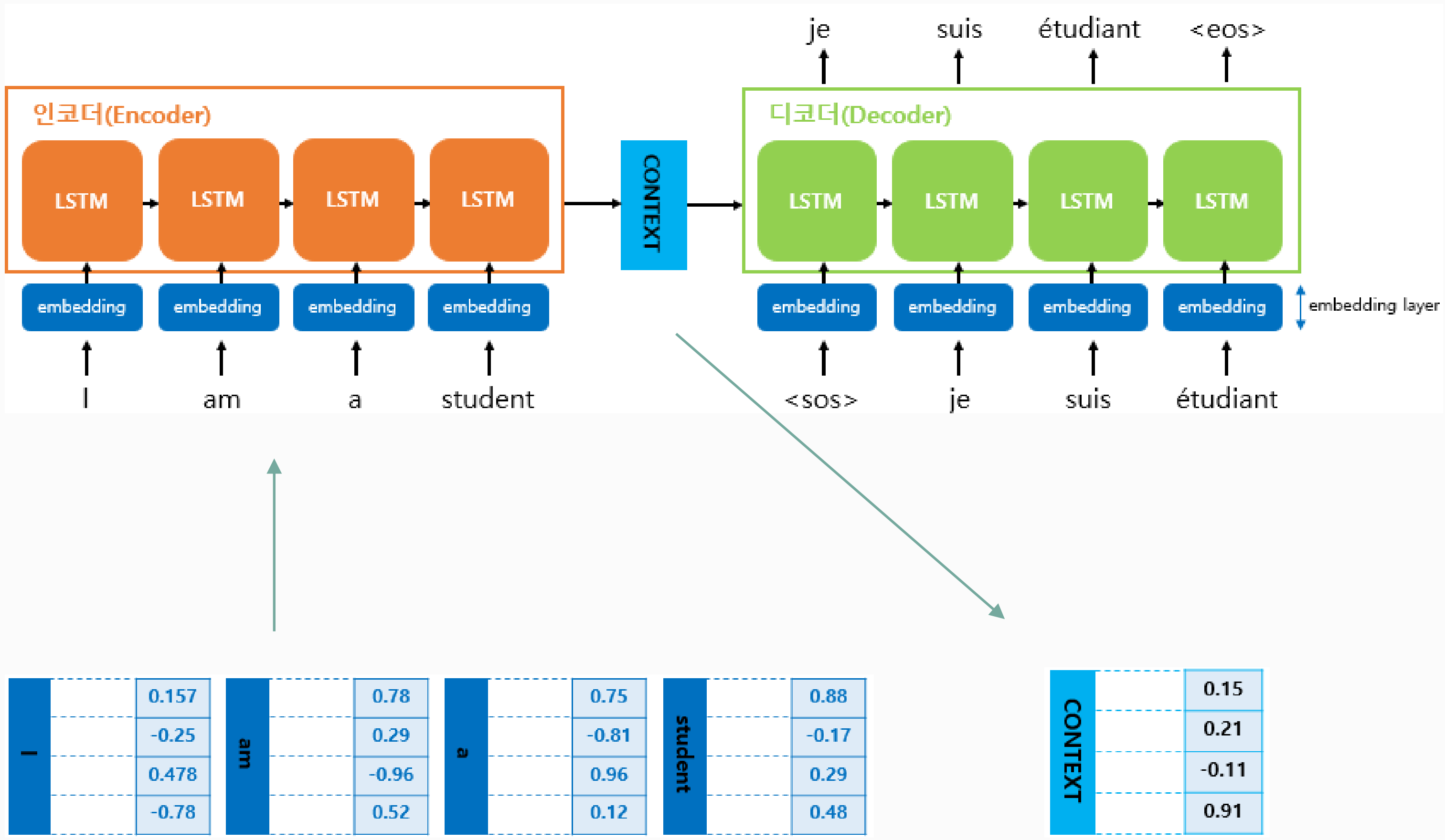
Sequence-to-Sequence : 입력된 시퀀스로부터 다른 도메인의 시퀀스를 출력하는 모델

Ex) 챗봇(Chatbot), 기계 번역(Machine Translation)



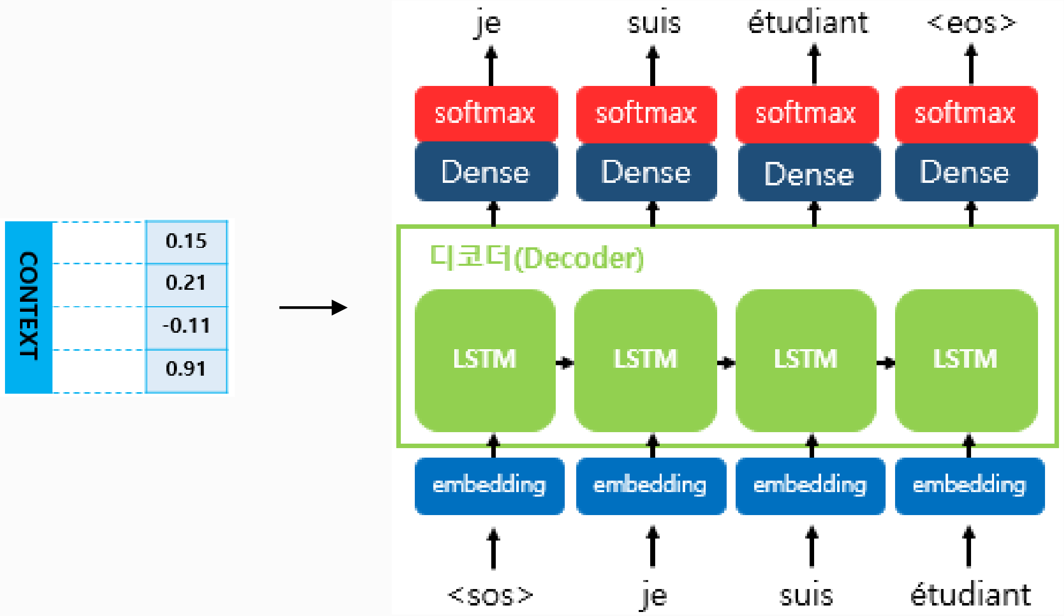
Sequence To Sequence

RNN을 활용한 Seq2Seq



Sequence To Sequence

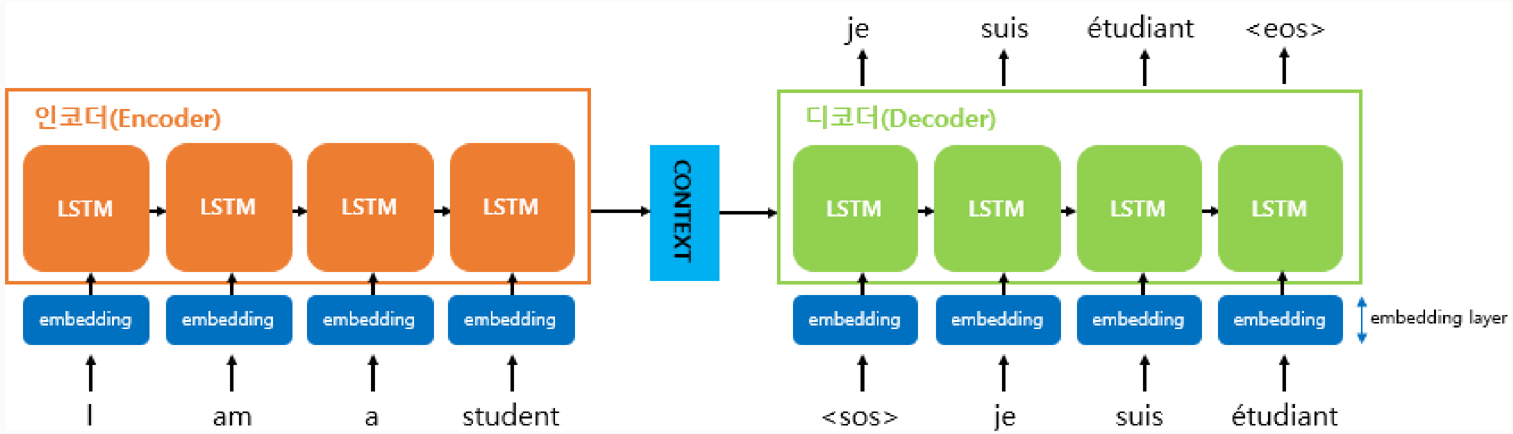
Detail of Decoder



Dense층의 weights는 공유

Sequence To Sequence

RNN을 활용한 Seq2Seq의 문제?



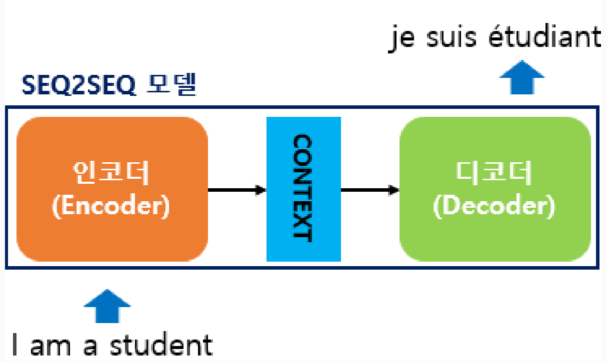
하나의 고정된 벡터(context vector)로 모든 정보를 압축하려 하니 정보 손실이 발생

(문장의 길이가 길수록 품질이 떨어짐)

Attention

Attention

Attention allows the model to focus on the relevant part of the input sequence as needed



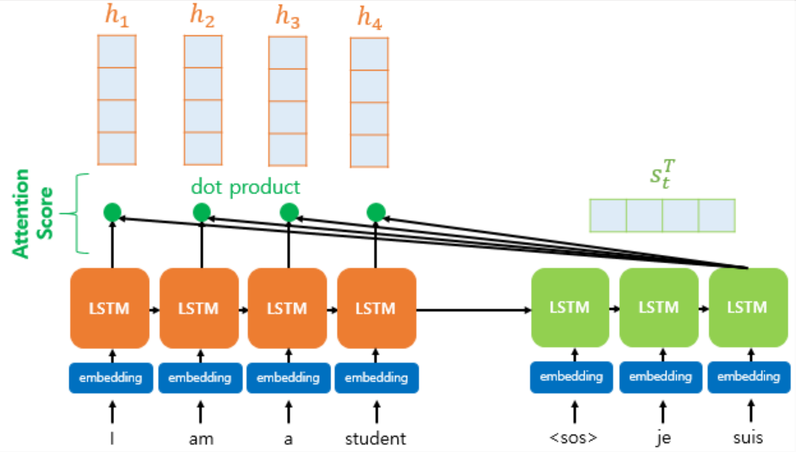
✓ Bahadanau attention (Bahdanau et al., 2015)

- Attention scores are <u>separated trained</u>, the current hidden state is a function of the context vector and the previous hidden state

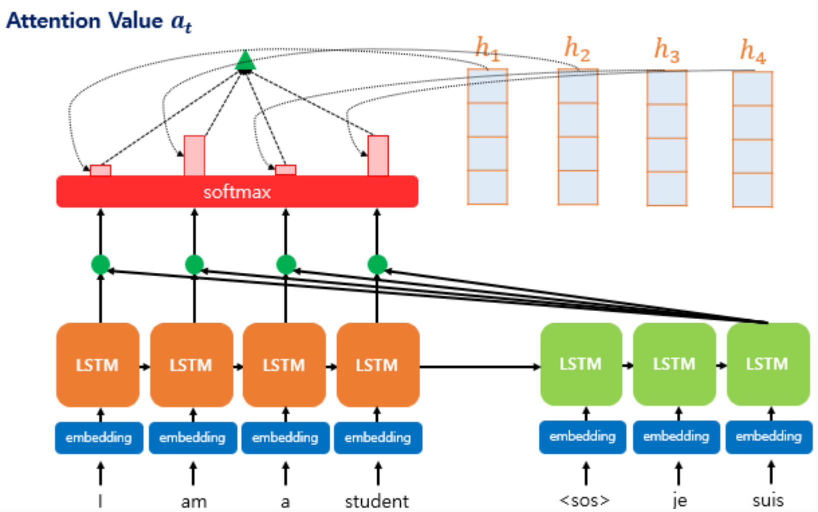
✓ Luong attention (Luong et al., 2015)

- Attention scores are <u>not trained</u>, the new current hidden state is the simple tanh of the weigthed concatenation of the context vector and the current hidden state of the decoder

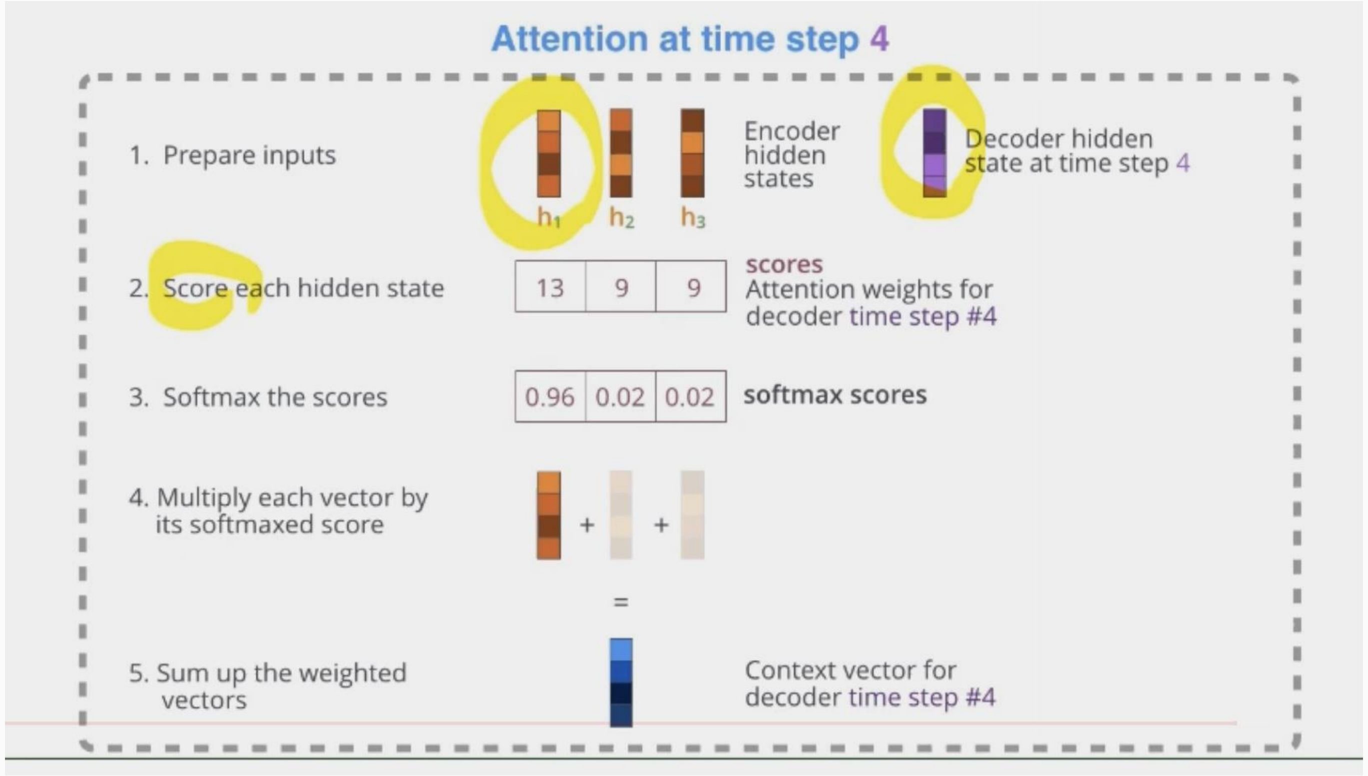
Attention



h_1
h_2
h_3
h_4
Attention Score
dot product
s_t^T
LSTM
embedding
I
am
a
student
<sos>
je
suis



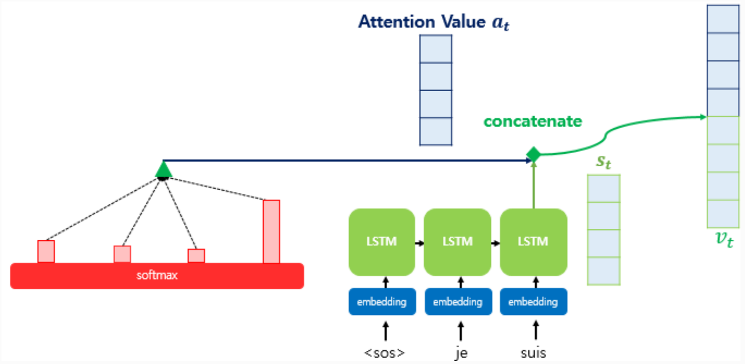
Attention Value a_t
h_1
h_2
h_3
h_4
softmax
LSTM
embedding
I
am
a
student
<sos>
je
suis



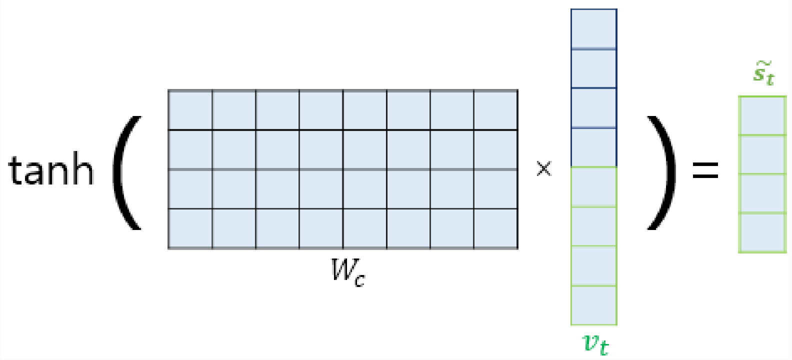
Attention at time step 4
1. Prepare inputs
Encoder hidden states
h_1
h_2
h_3
Decoder hidden state at time step 4
2. Score each hidden state
13
9
9
scores
Attention weights for decoder time step #4
3. Softmax the scores
0.96
0.02
0.02
softmax scores
4. Multiply each vector by its softmaxed score
+
+
=
5. Sum up the weighted vectors
Context vector for decoder time step #4

Attention

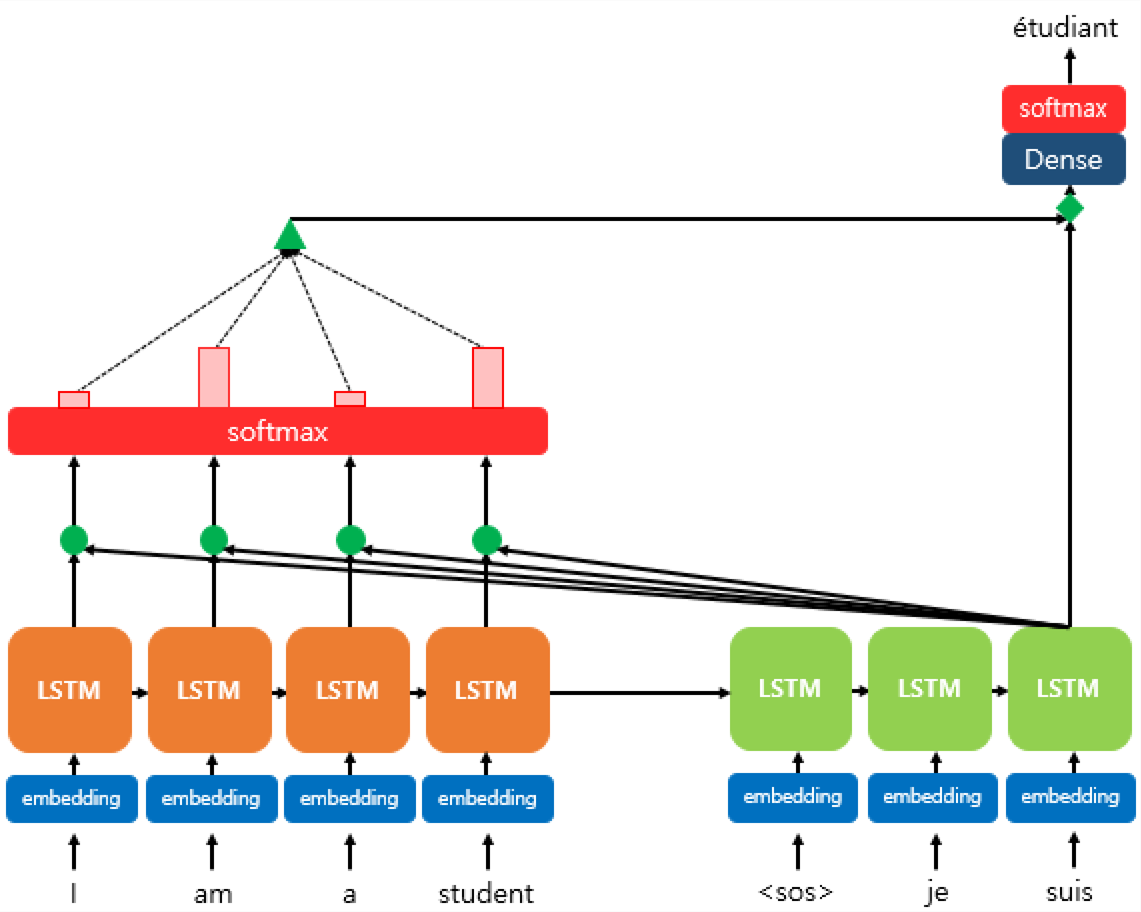
- Concatenate attention value and current hidden state of decoder.

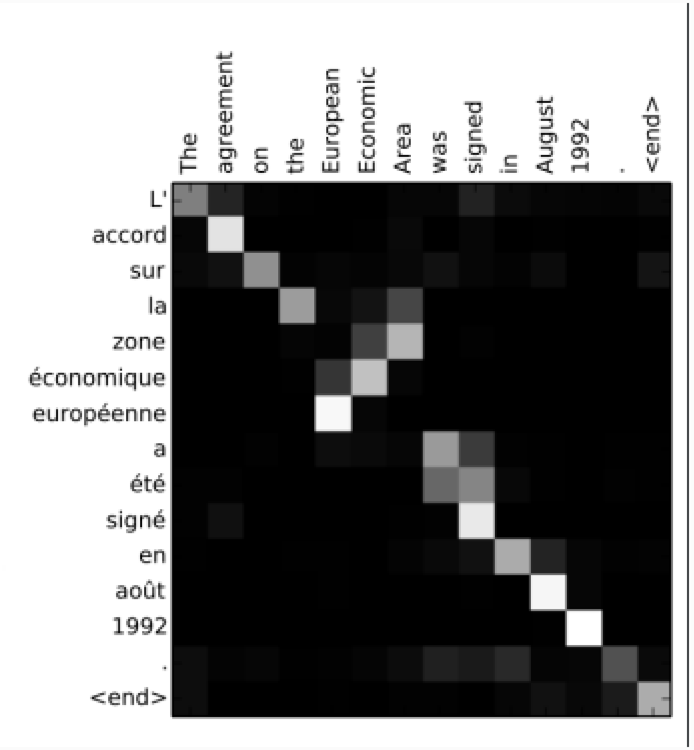


- Calculate new hidden state vector

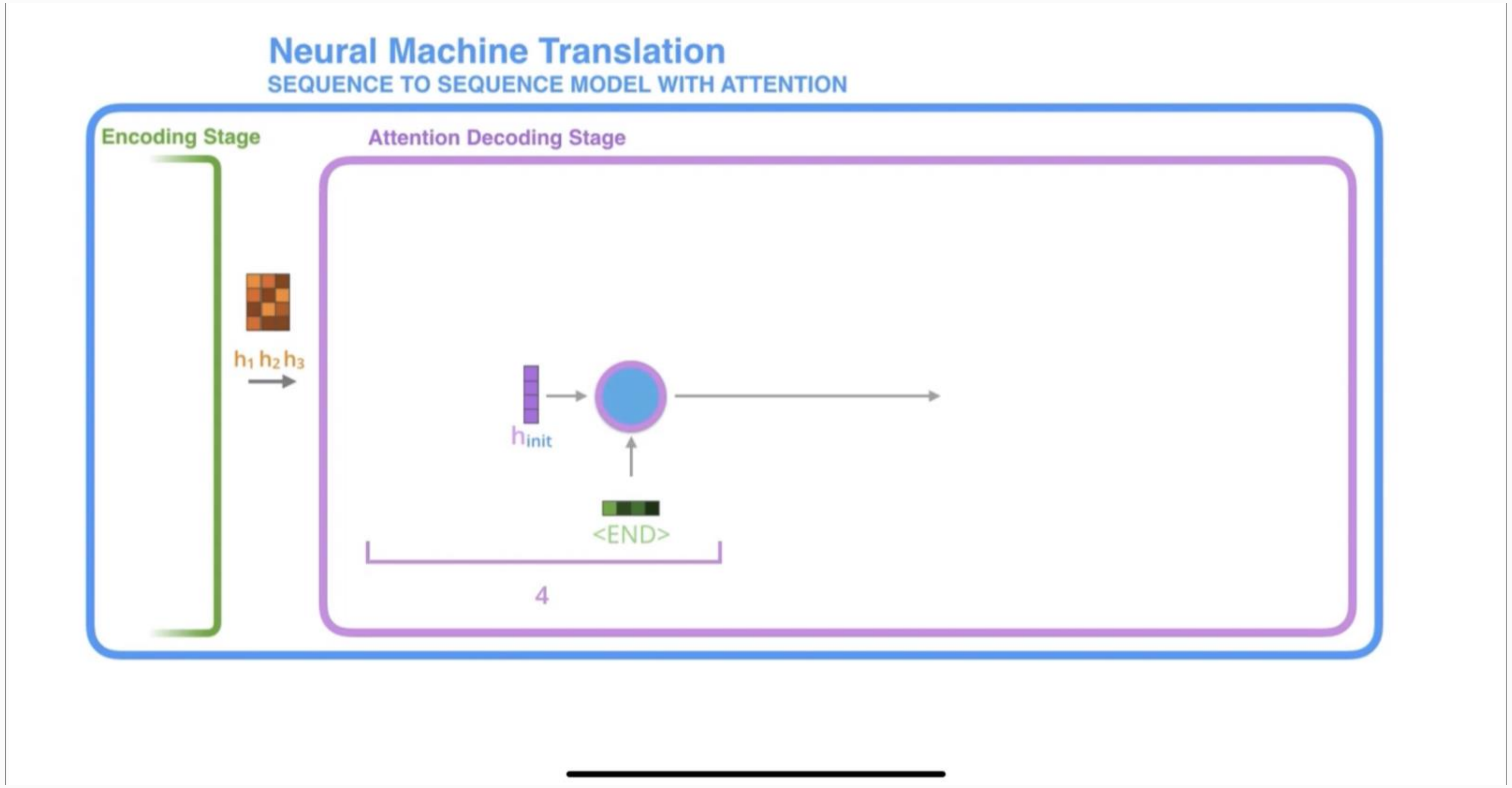


Working mechanism of attention process

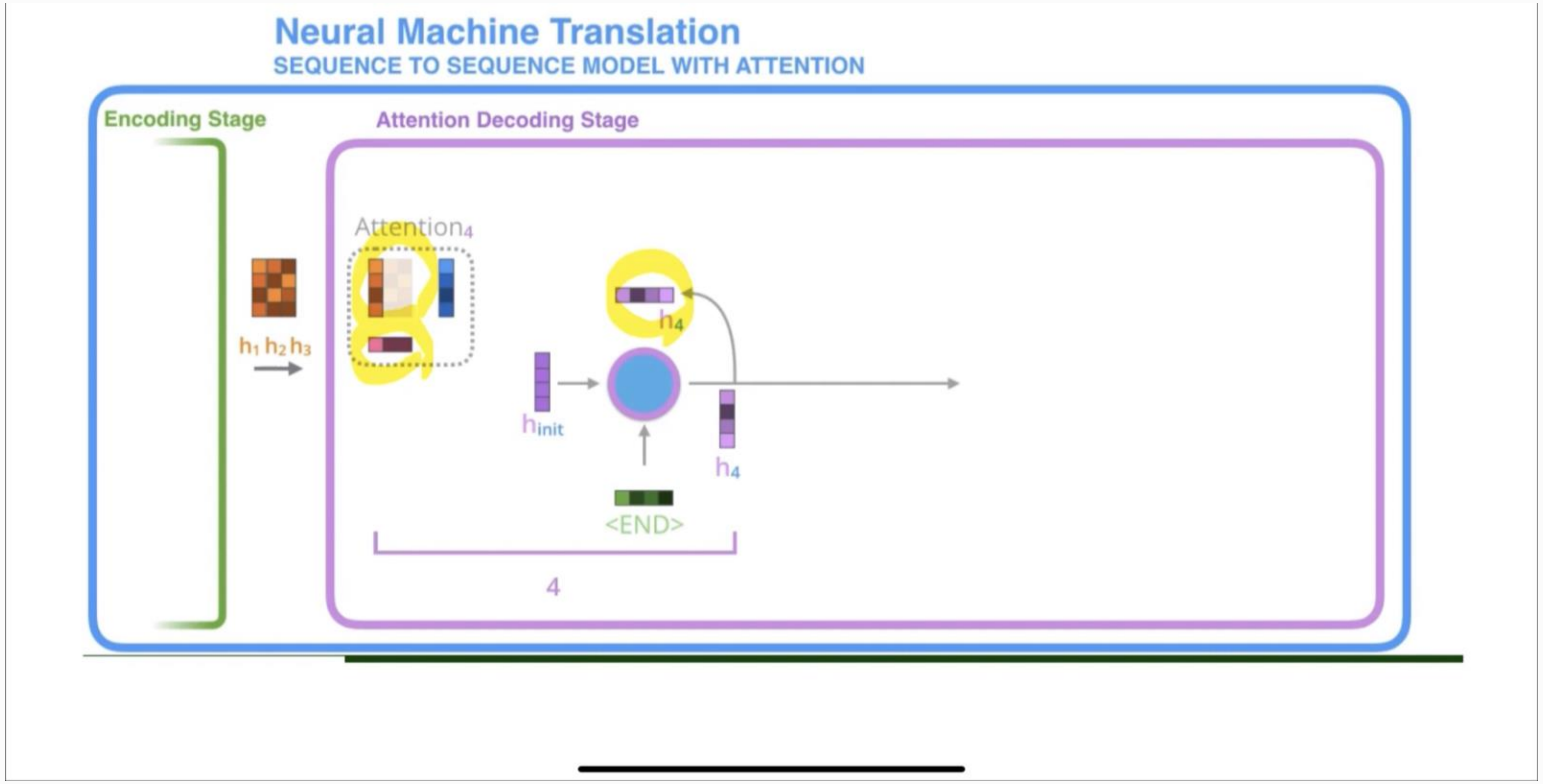




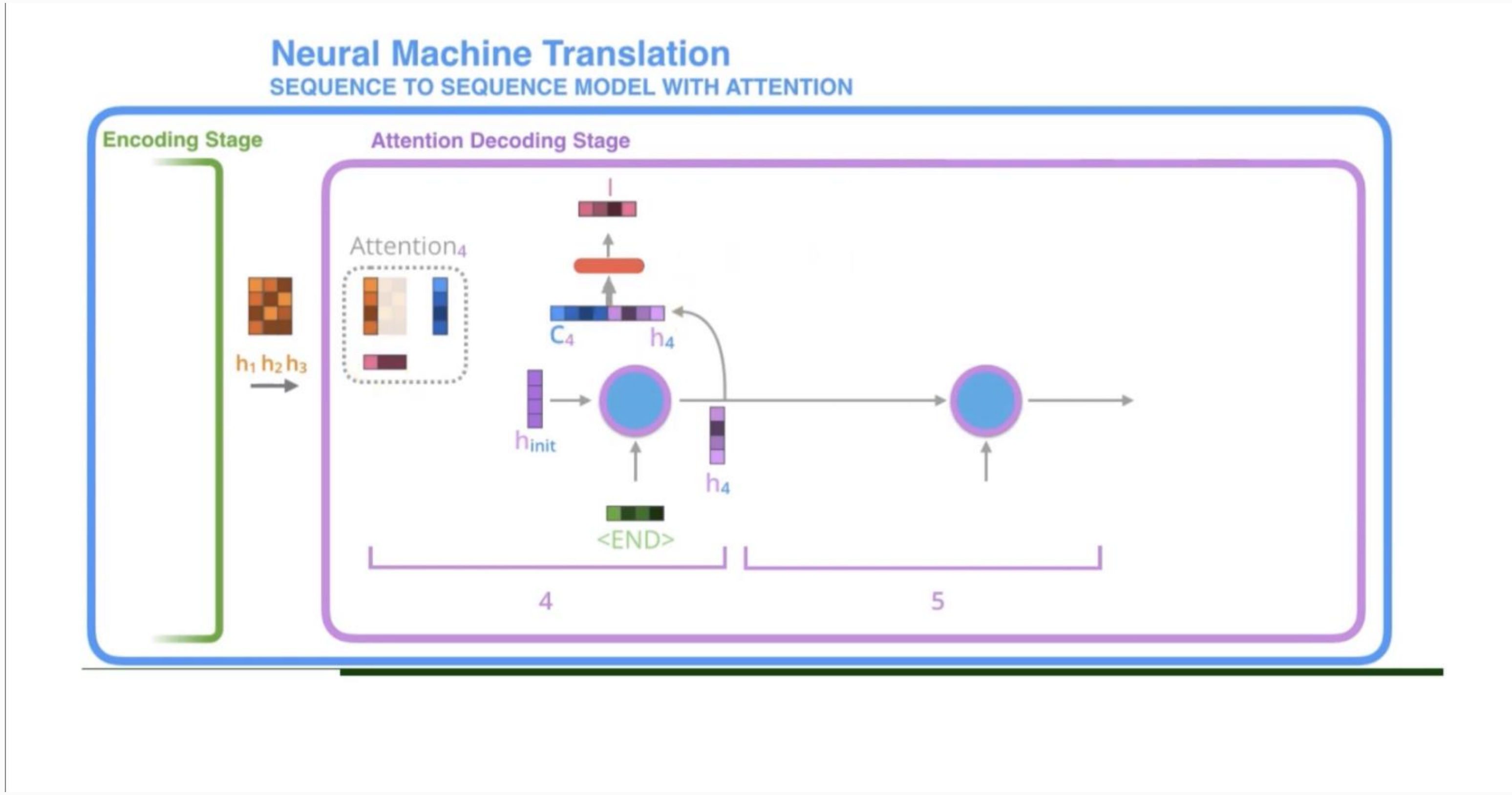
Working mechanism of attention process



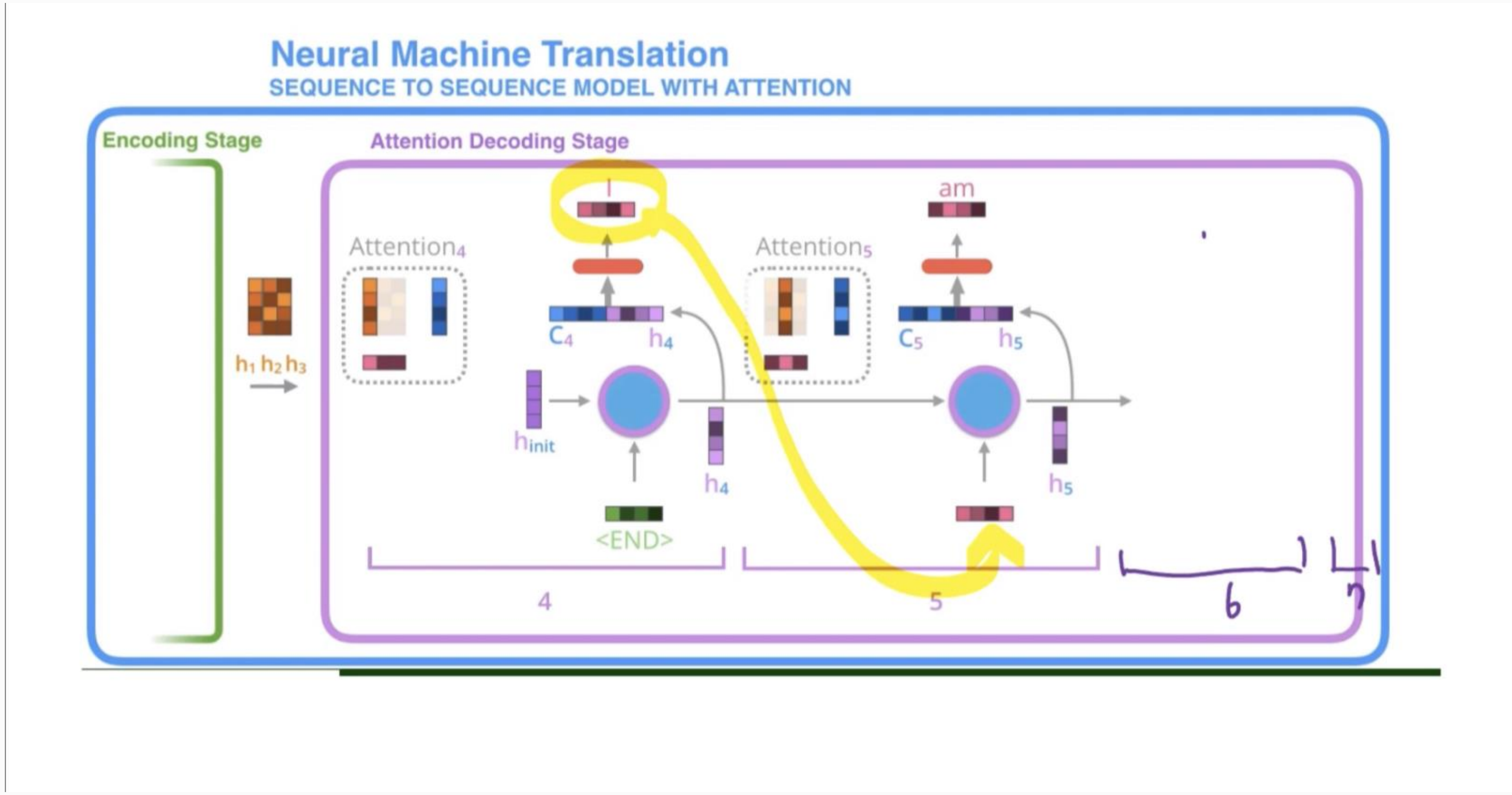
Working mechanism of attention process



Working mechanism of attention process



Working mechanism of attention process



Attention is all you need!

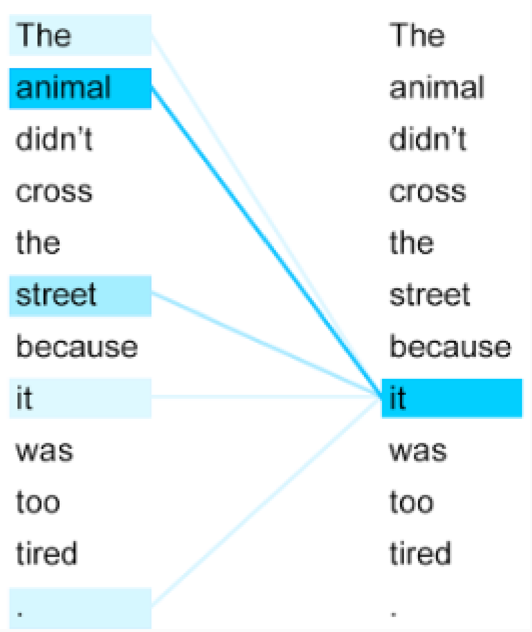
2017년 구글이 발표한 논문인 “Attention is all you need”에서 나온 모델

기존의 seq2seq 모델과 달리 RNN을 사용하지 않고, Attention만을 이용한 인코더-디코더 모델

→ Input data가 순차적으로 입력될 필요가 없기 때문에 병렬적으로 처리 가능

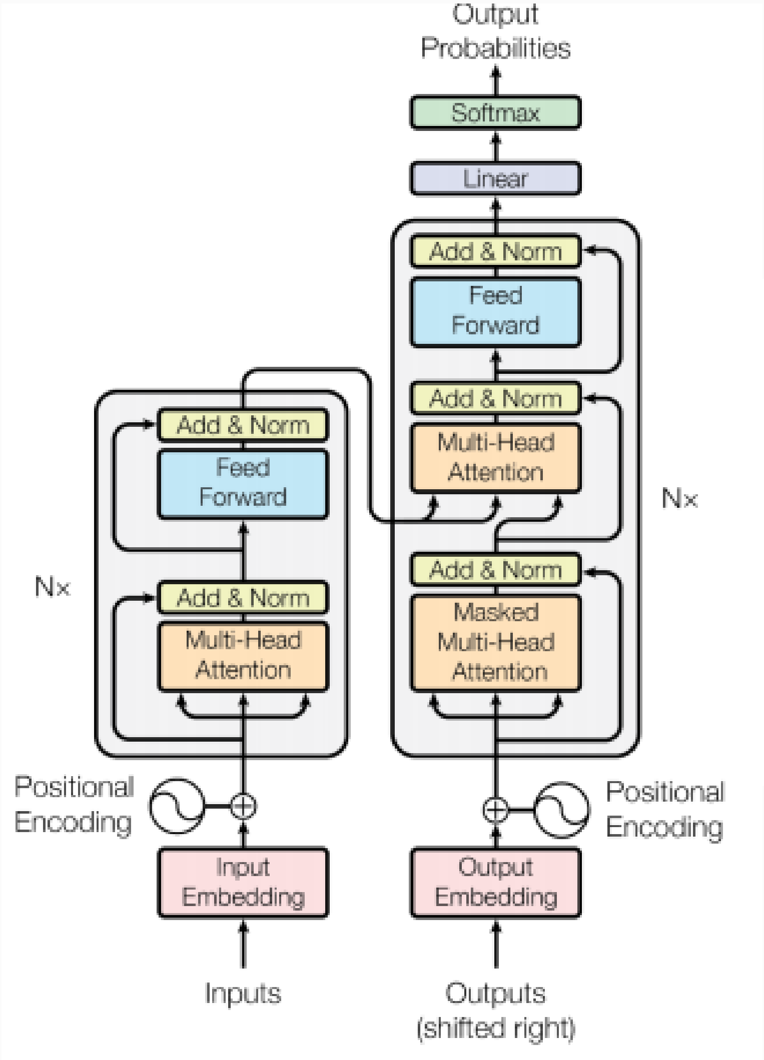
Q. RNN을 사용하지 않으면 어떻게 문맥을 파악?

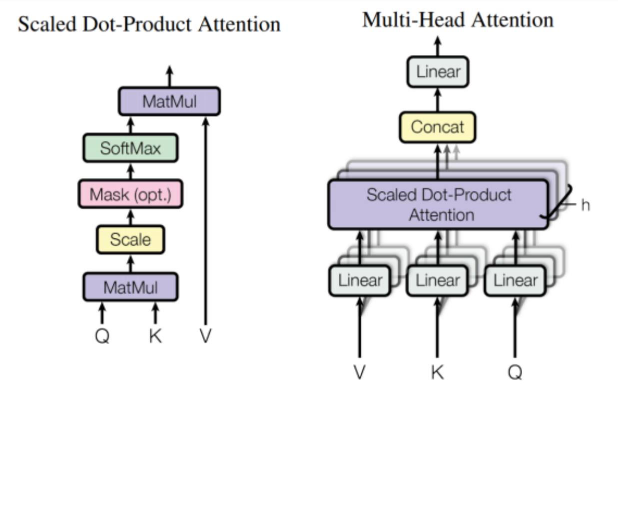
A. 자기 자신을 Attention시키는 self-attention 이용!

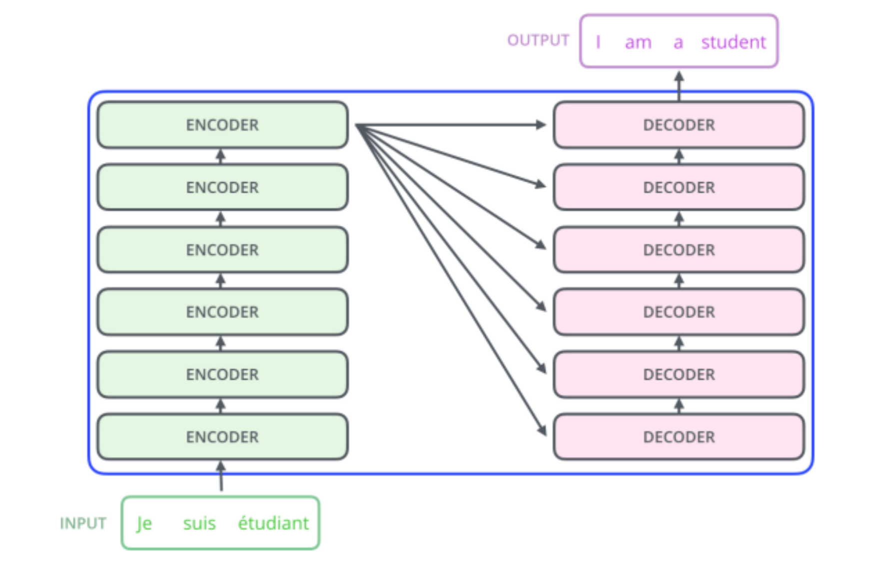


Transformer

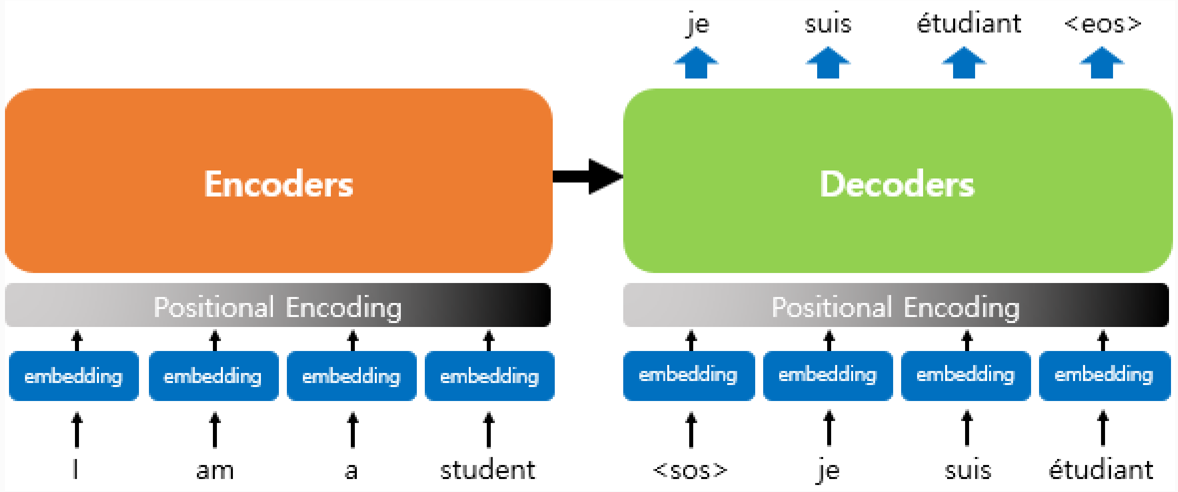
constructure

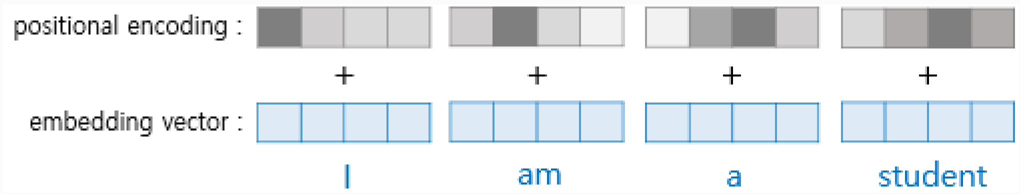






Positional Encoding



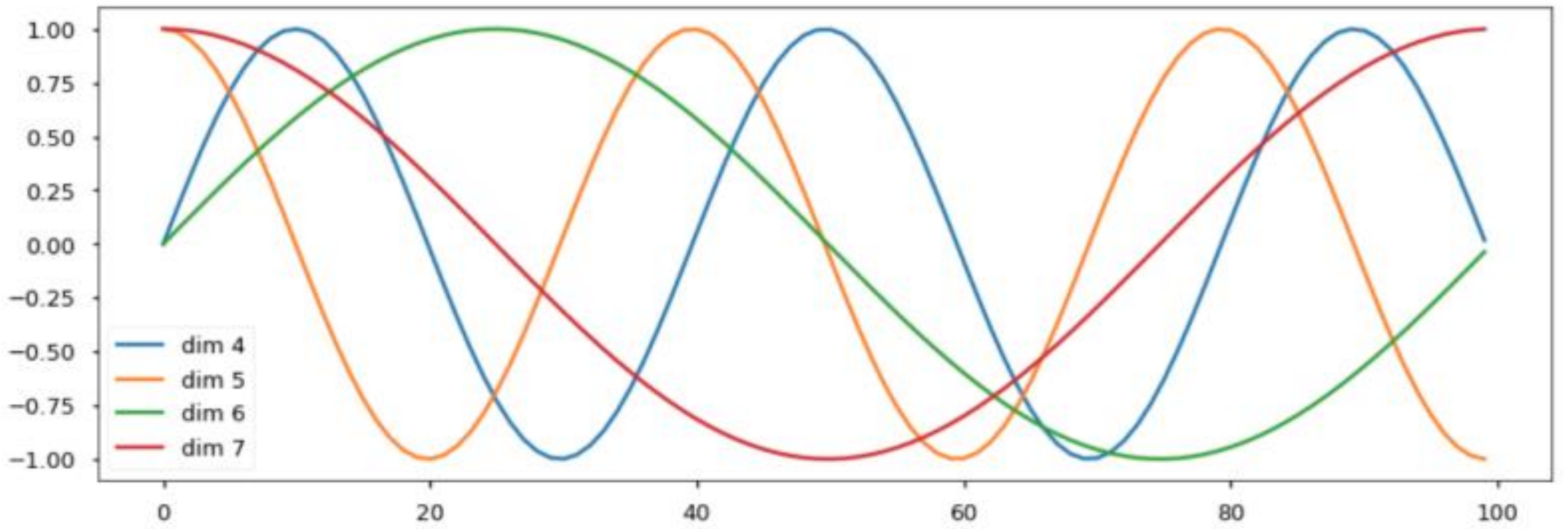


Positional Encoding

- Two properties that a good positional encoding scheme should have
  - ✓ The norm of encoding vector is the same for all positions
  - ✓ The further the two positions, the larger the distance
    - A Simple Example (n = 10, dim = 10)

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

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

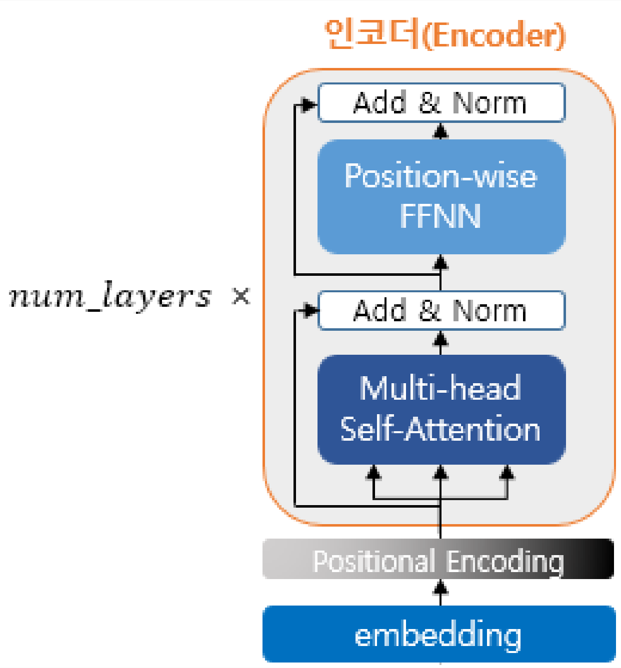


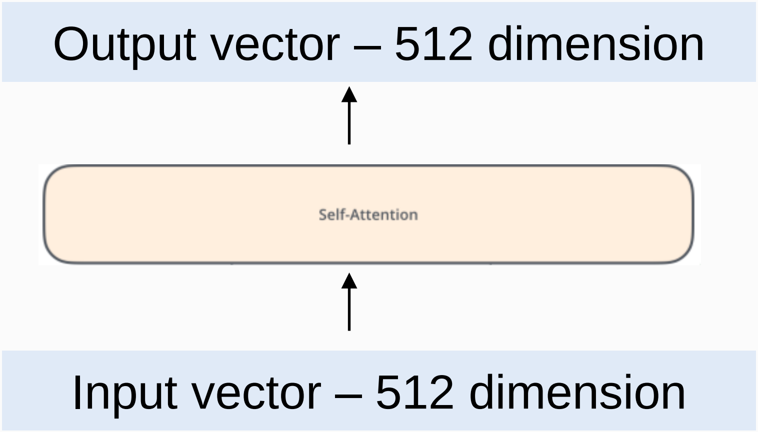
Positional Encoding

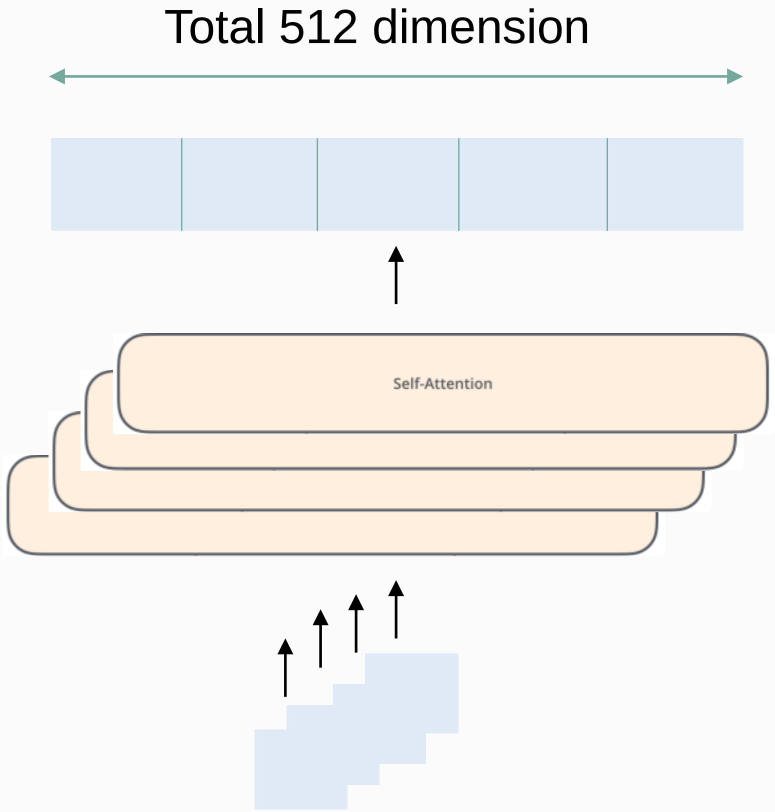
- A Simple Example (n = 10, dim = 10)
  - ✓ Distances between two positional encoding vectors

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
X1	0.000	1.275	2.167	2.823	3.361	3.508	3.392	3.440	3.417	3.266
X2	1.275	0.000	1.104	2.195	3.135	3.511	3.452	3.442	3.387	3.308
X3	2.167	1.104	0.000	1.296	2.468	3.067	3.256	3.464	3.498	3.371
X4	2.823	2.195	1.296	0.000	1.275	2.110	2.746	3.399	3.624	3.399
X5	3.361	3.135	2.468	1.275	0.000	1.057	2.176	3.242	3.659	3.434
X6	3.508	3.511	3.067	2.110	1.057	0.000	1.333	2.601	3.169	3.118
X7	3.392	3.452	3.256	2.746	2.176	1.333	0.000	1.338	2.063	2.429
X8	3.440	3.442	3.464	3.399	3.242	2.601	1.338	0.000	0.912	1.891
X9	3.417	3.387	3.498	3.624	3.659	3.169	2.063	0.912	0.000	1.277
X10	3.266	3.308	3.371	3.399	3.434	3.118	2.429	1.891	1.277	0.000

Multi-head Self Attention



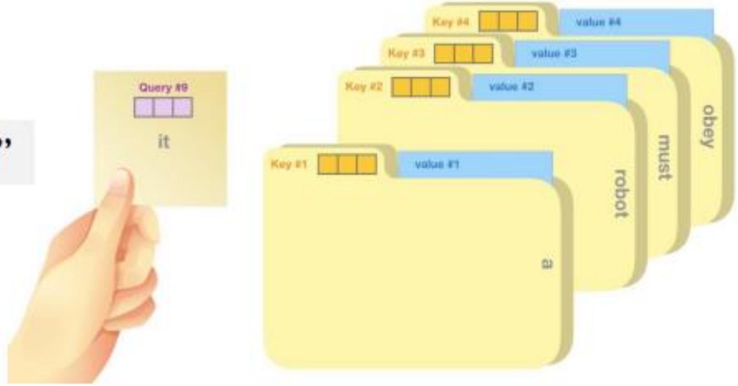




Self-Attention

- Self-Attention in Detail
  - ✓ Step 1: Create three vectors from each of the encoder's input vectors
    - 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.

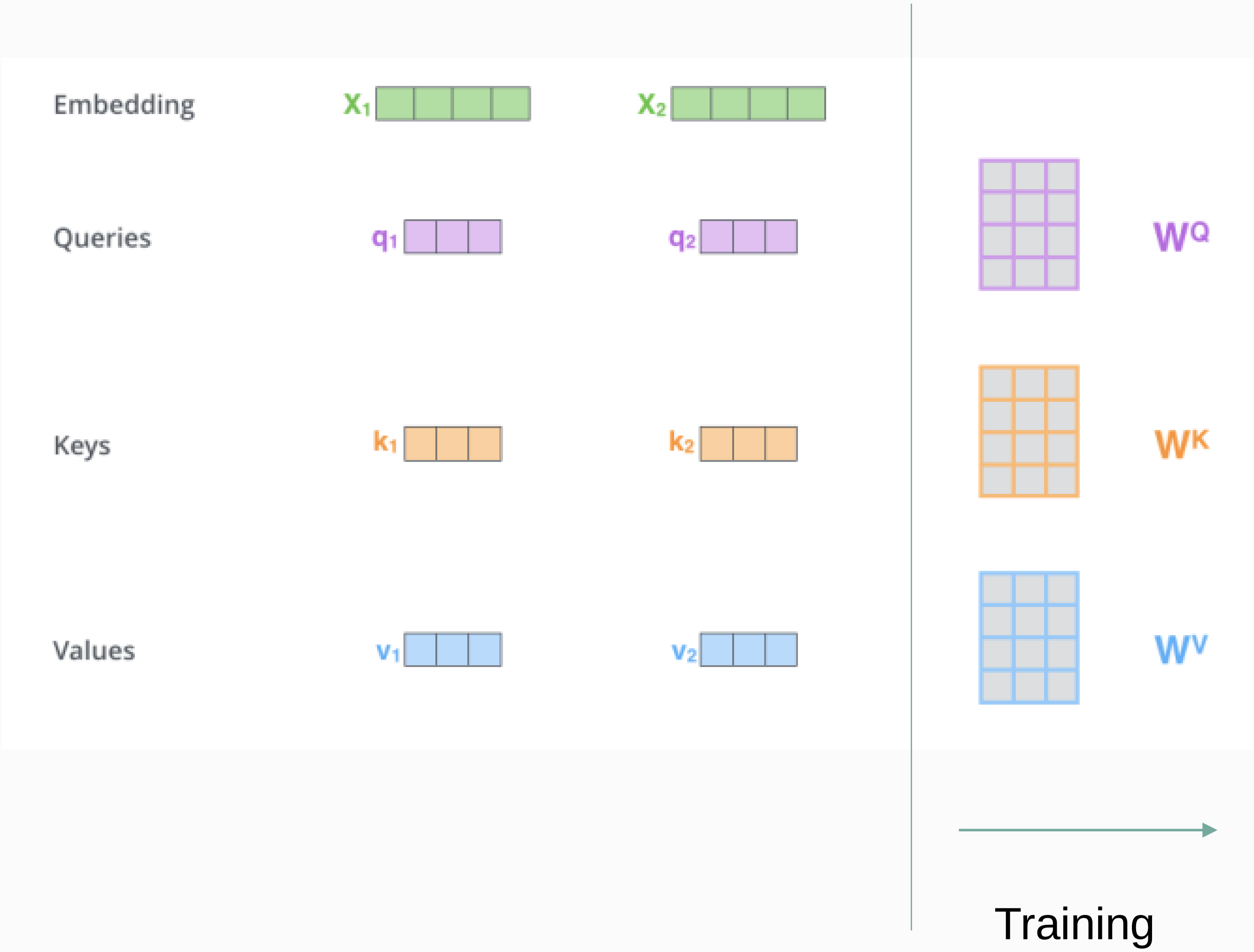
"A robot must obey the orders given it"



Note) These new vectors are smaller in dimension that the embedding vector

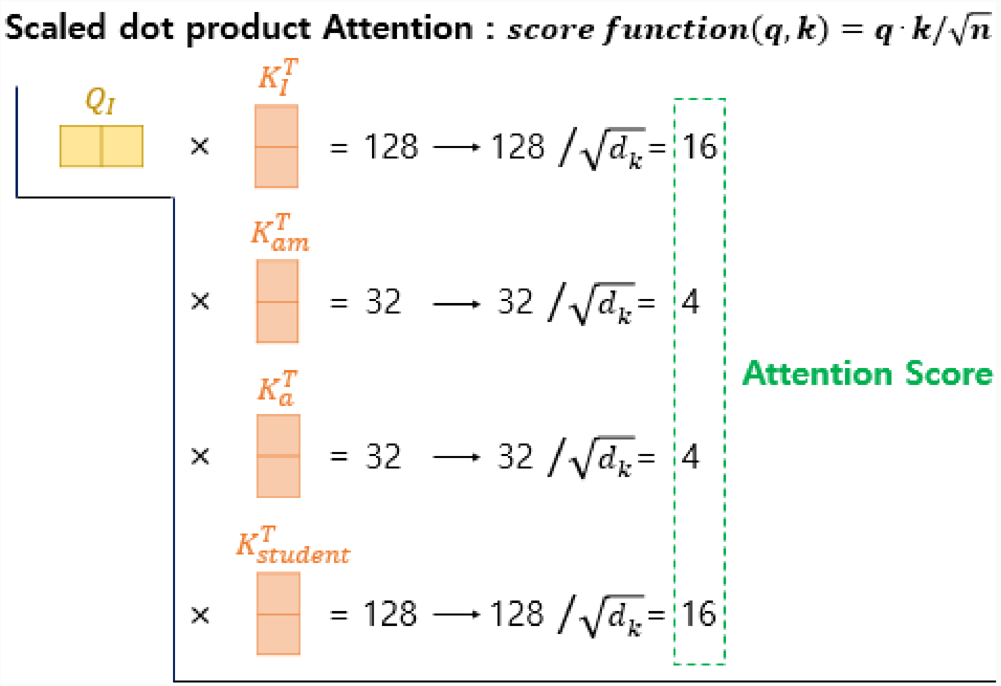
- Q, K, and V are 64-dim. while embedding and encoder input/output vectors are 512-dim.
- They do not have to be smaller but it is an architecture choice to make the computation of multi-headed attention (mostly) constant

Self-Attention



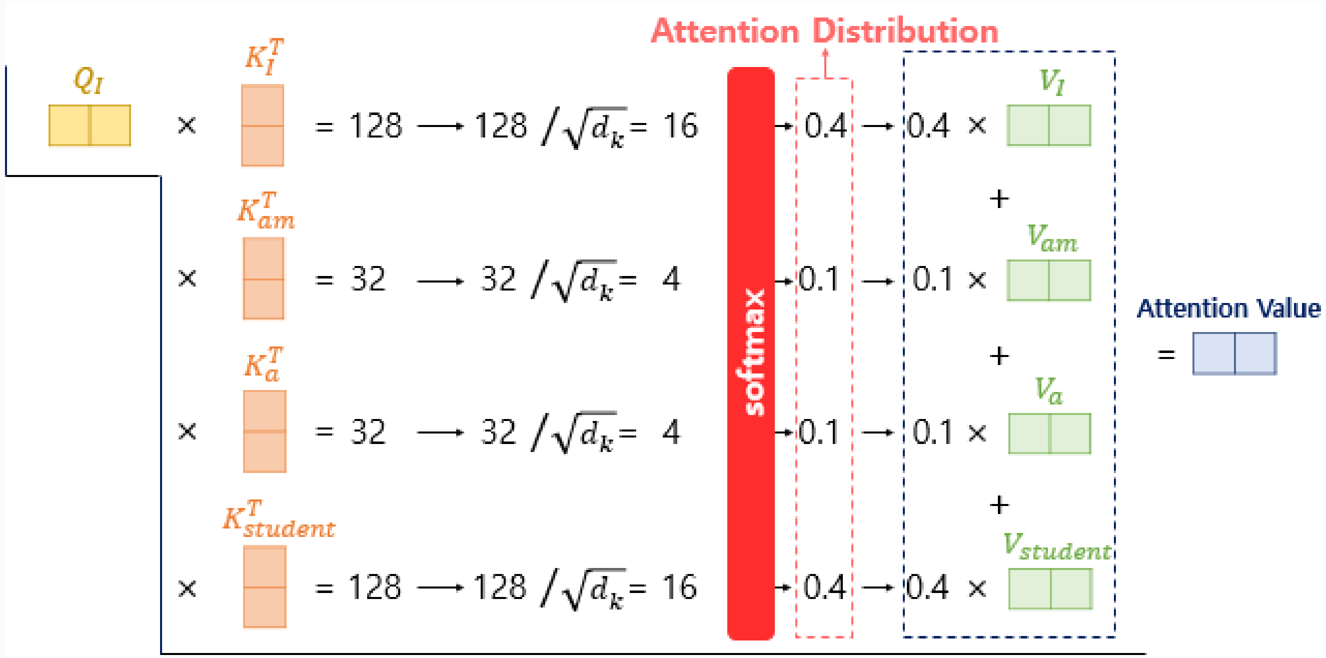
Self-Attention

Ex) I am a student



Self-Attention

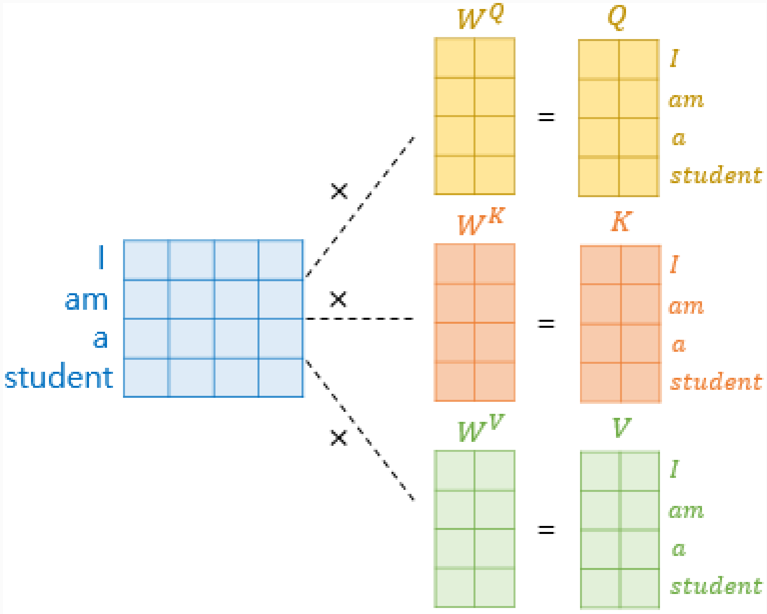
Ex) I am a student

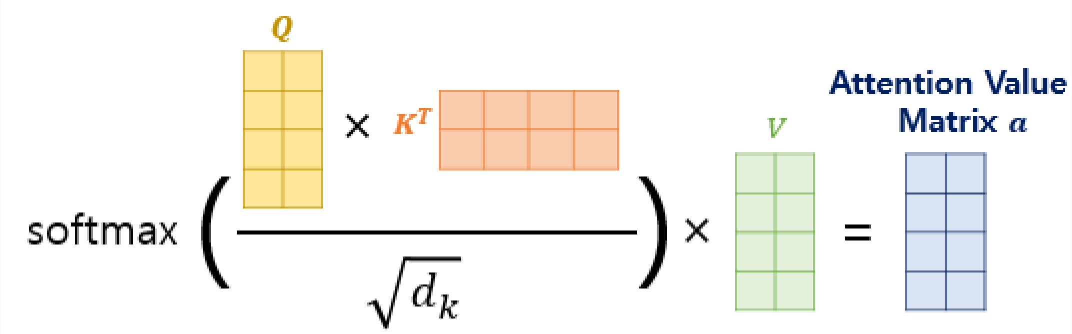


Self-Attention

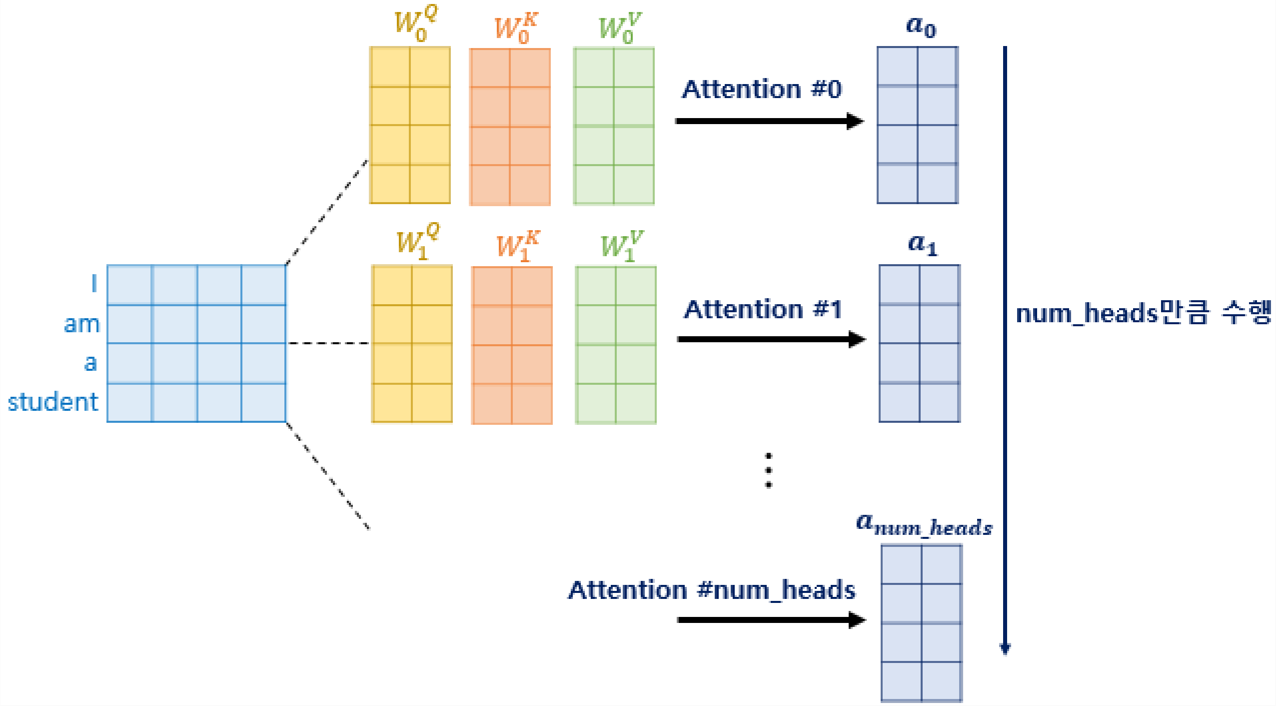
Ex) I am a student

행렬 연산으로 일괄 계산

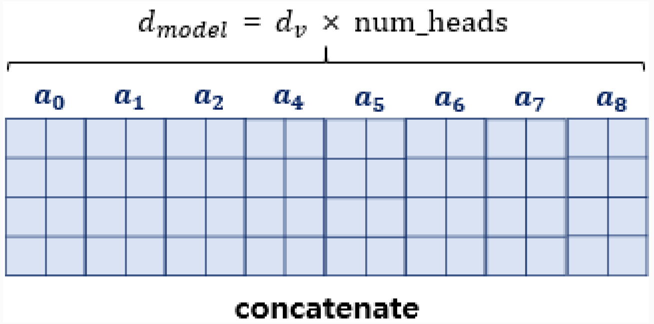




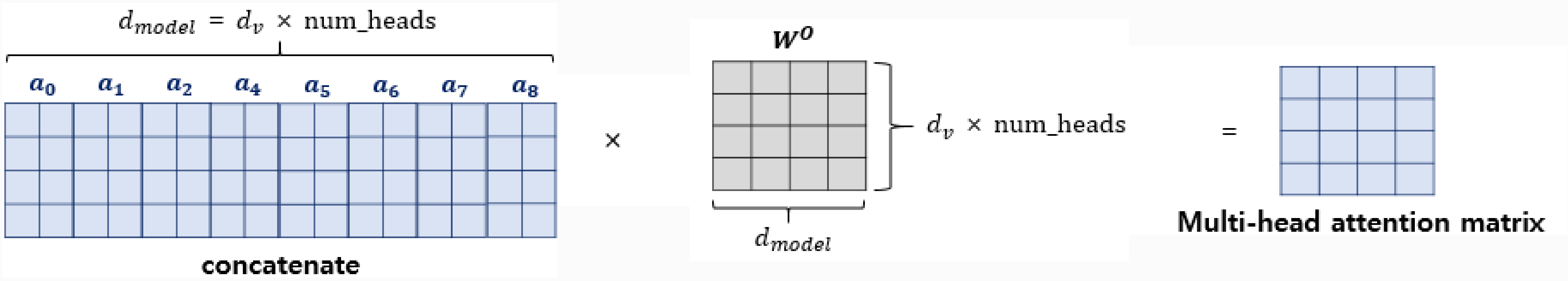
Multi-head Attention

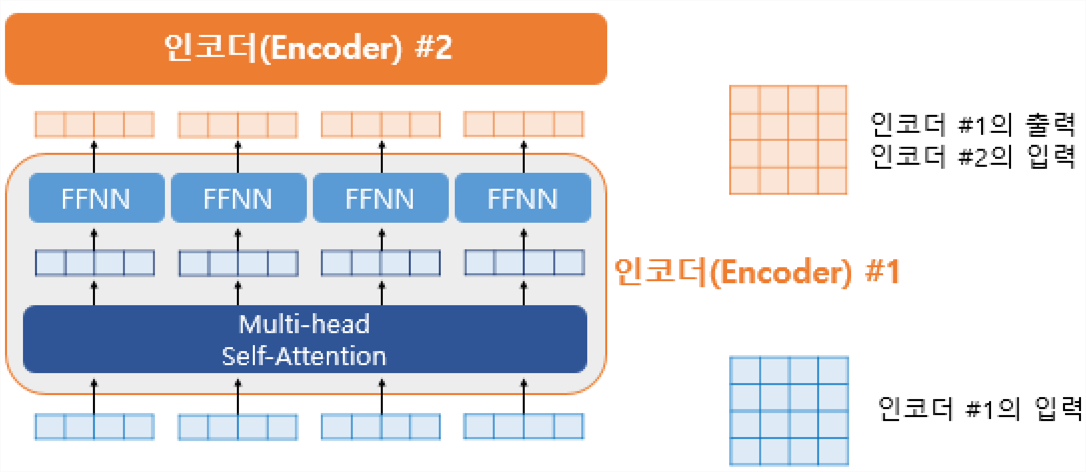


concatenate

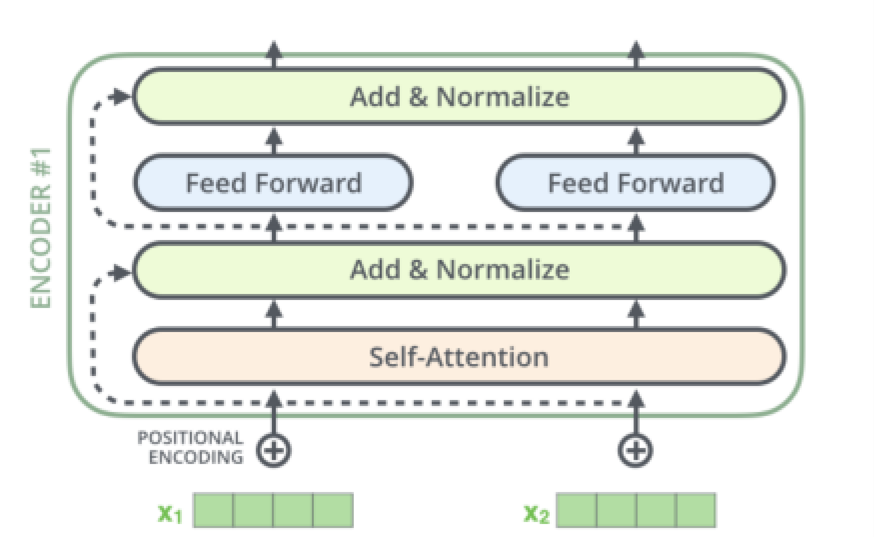


Multi-head Attention

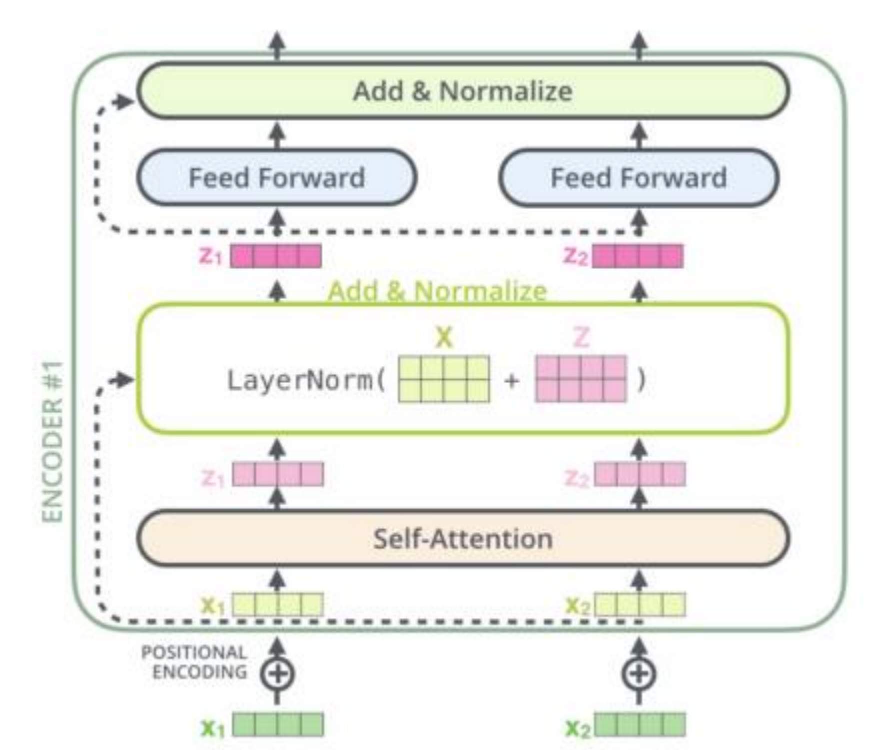




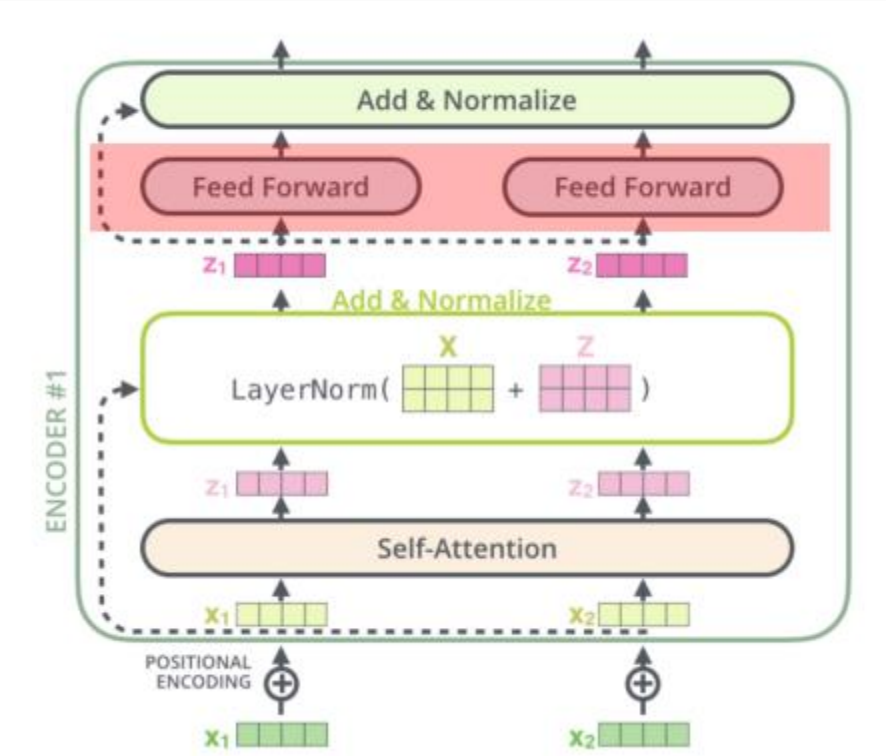
Residual connection & Layer nomalization



F(x) + x → F'(x) + 1

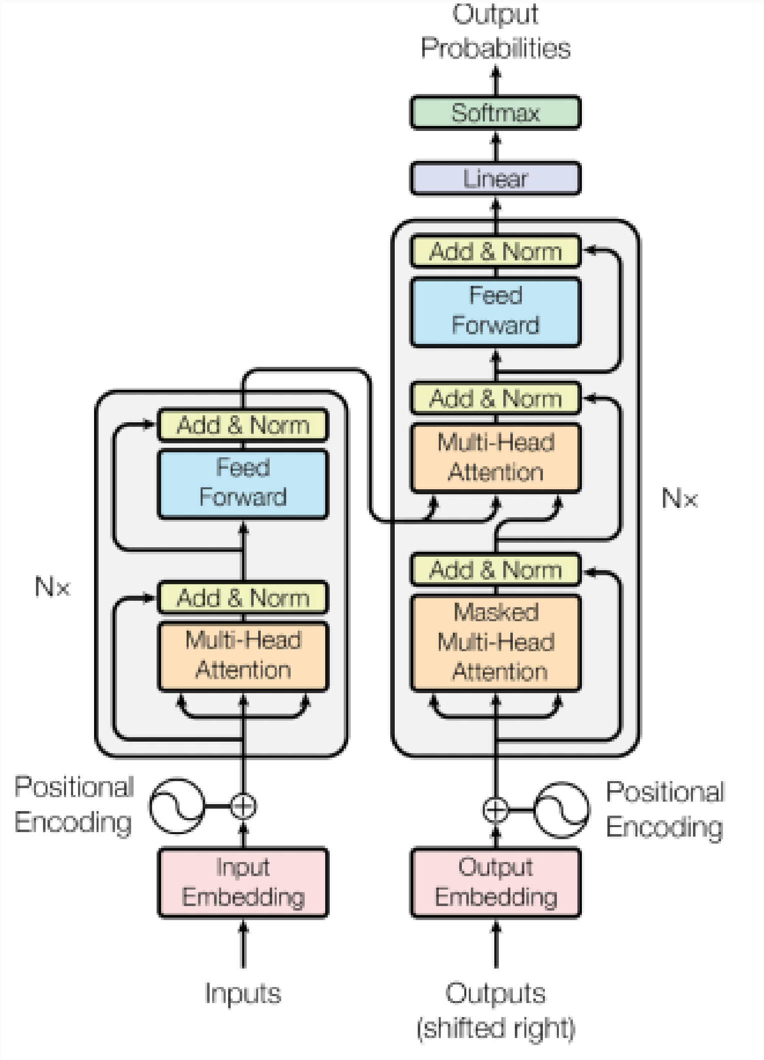


Feed Forward Neural Network



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

Decoder



Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Masked Multi-head Attention

- Masked Multi-head Attention

Queries

robot	must	obey	orders

X

Keys

robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders

=

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Apply Attention Mask →

Masked Scores (before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Masked Scores (before softmax)

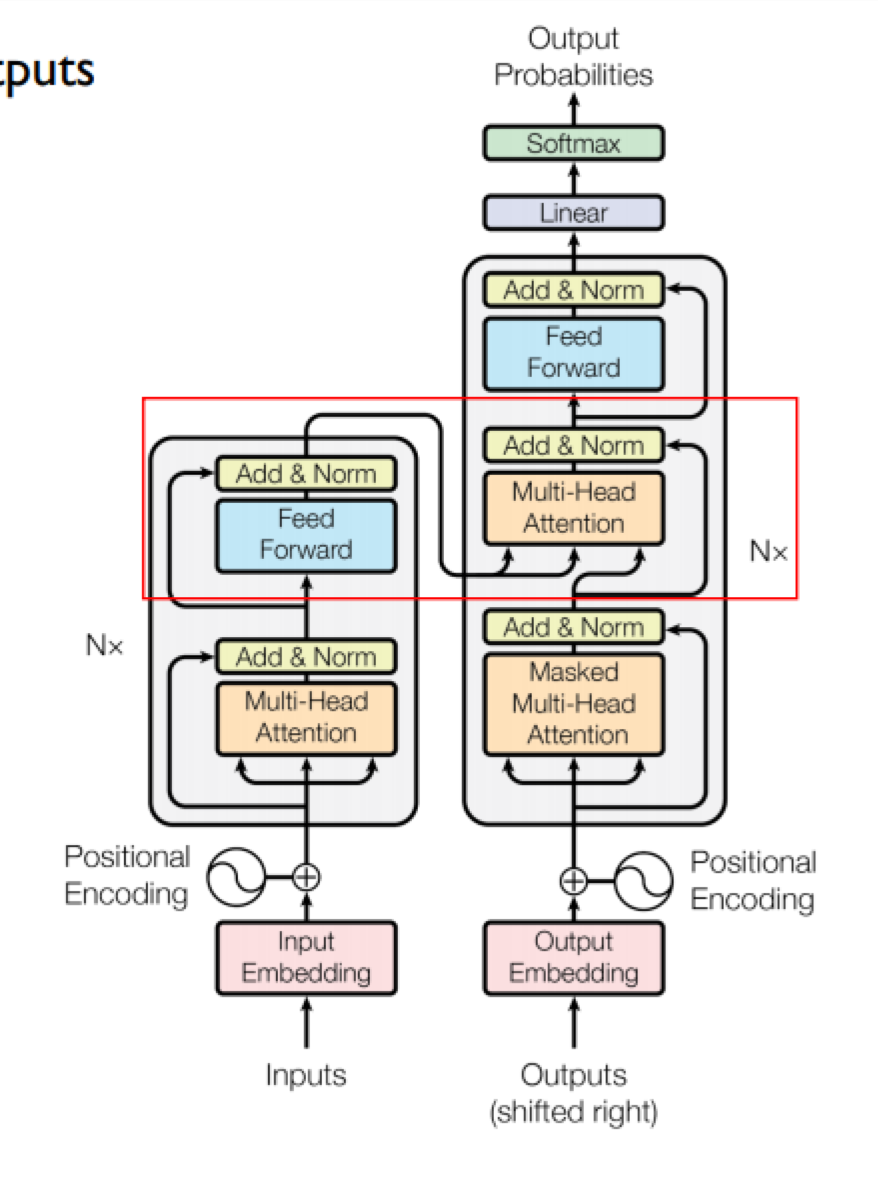
0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Softmax (along rows) →

Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

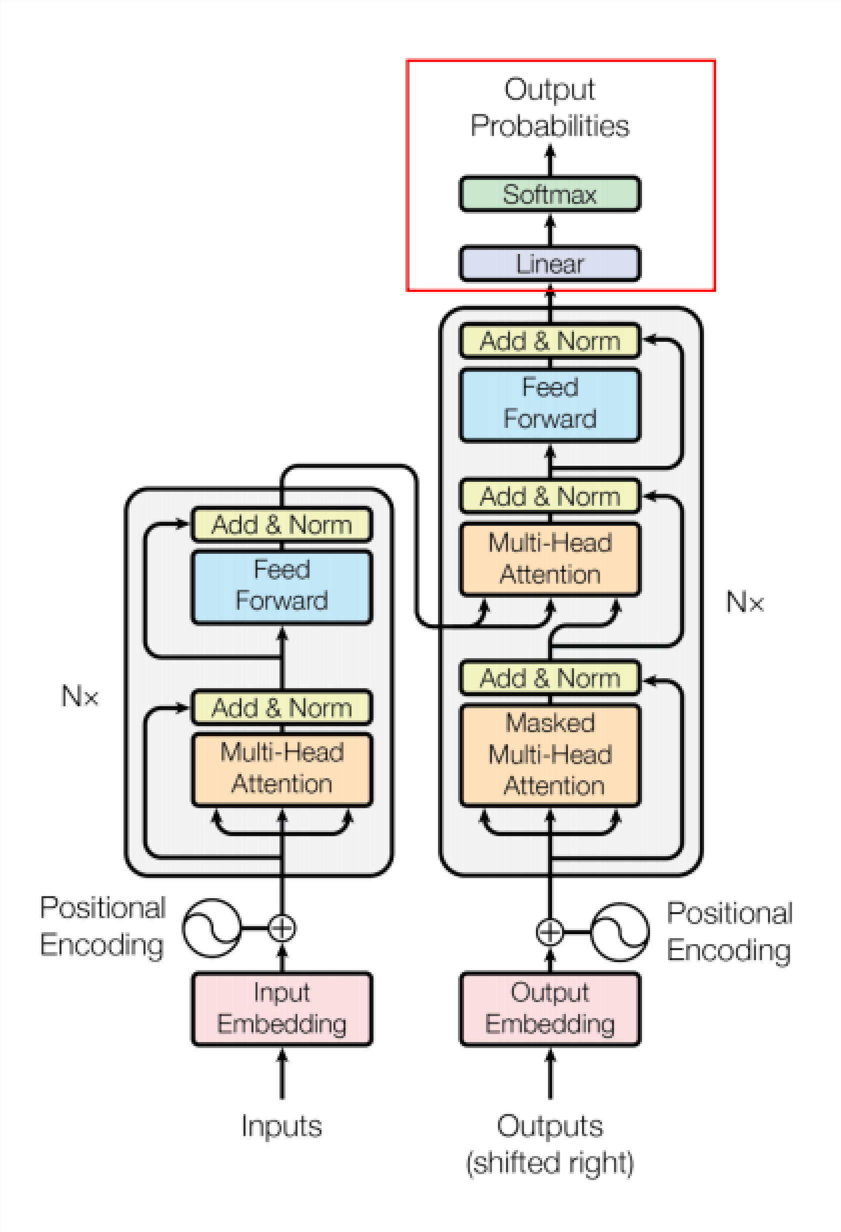
Incoder-Decoder Attention



영상자료 : (58분 34초)

https://www.youtube.com/watch?v=Yk1tV_cXMMU&list=PLetSlH8YjIfVzHuSXtG4jAC2zbEAErXWm&index=17

Linear and Softmax Layers



Q & A

참고자료
-Pilsung Kang School of Industrial Management Engineering Korea University
(https://www.youtube.com/watch?v=Yk1tV_cXMMU&list=PLetSlH8YjIfVzHuSXtG4jAC2zbEAErXWm&index=17)
-딥러닝을 이용한 자연어 처리 입문
(https://wikidocs.net/31379)