"Attention is All You need"

[Transformer]

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

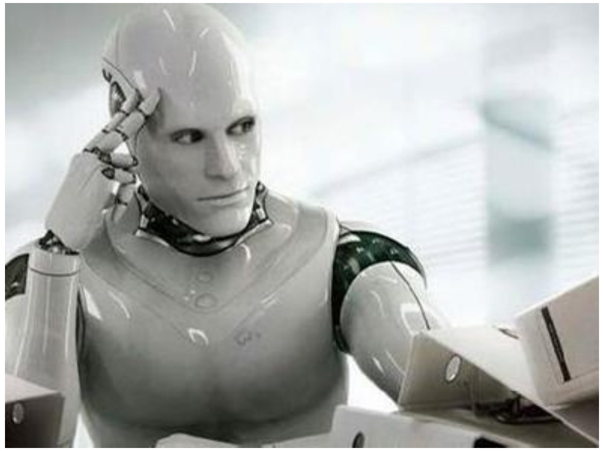
Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Presenter : Soyoung yoo

How to neural machine translation?

기계번역

I love you →  → 난 널 사랑해

How to neural machine translation?

Input		Prediction
I	⇨	난
love	⇨	사랑해
You	⇨	널

How to neural machine translation?

Input		Prediction
I	⇨	난
love	⇨	사랑해
You	⇨	널
I love you	⇨	난 사랑해 널

How to neural machine translation?

Input		Prediction
I	⇨	난
love	⇨	사랑해
You	⇨	널
I love you	⇨	난 사랑해 널
정답		난 널 사랑해

How to neural machine translation?

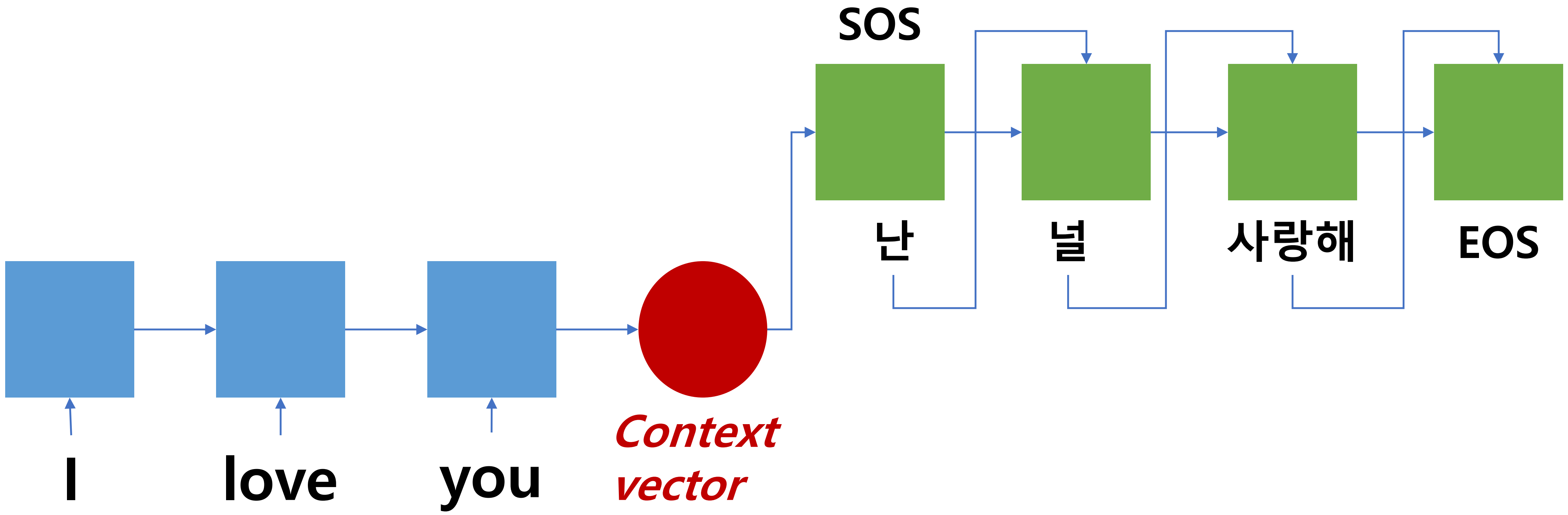
Input		Prediction
I	⇨	난
love	⇨	사랑해
You	⇨	널
I love you	⇨	난 사랑해 널
정답		난 널 사랑해
영어는 SVO		한국어는 SOV

How to **n**eural **m**achine **t**ranslation?

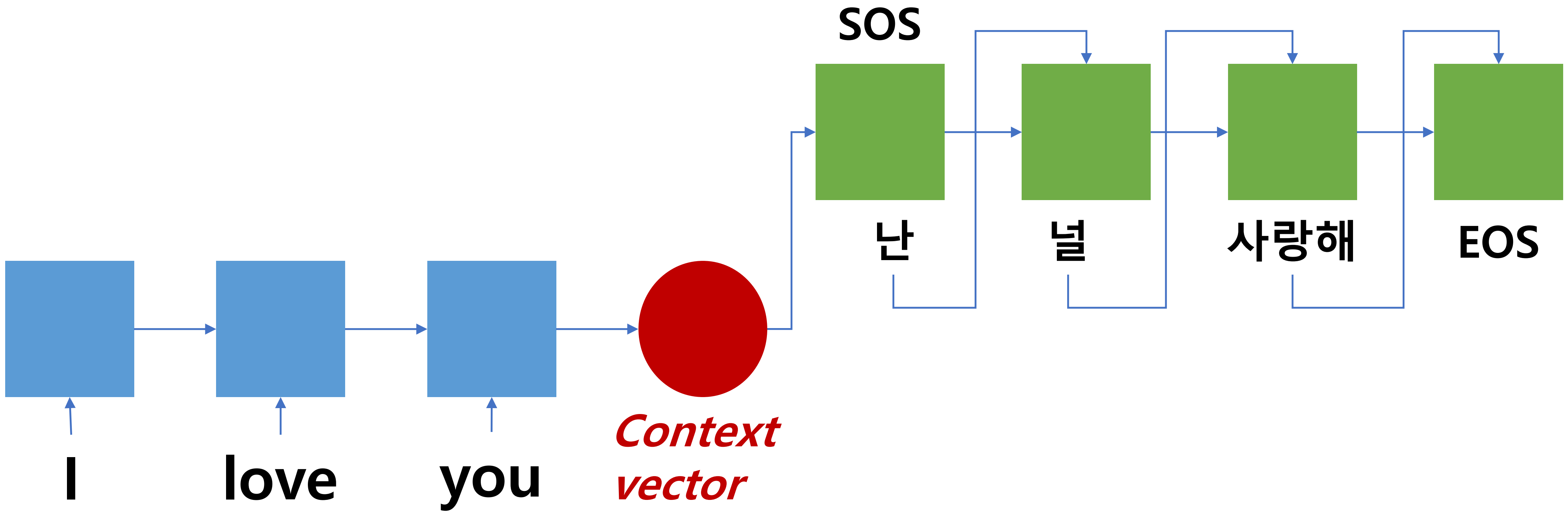
How are you? 잘 지내?

3개 단어 2개 단어

How to neural machine translation?

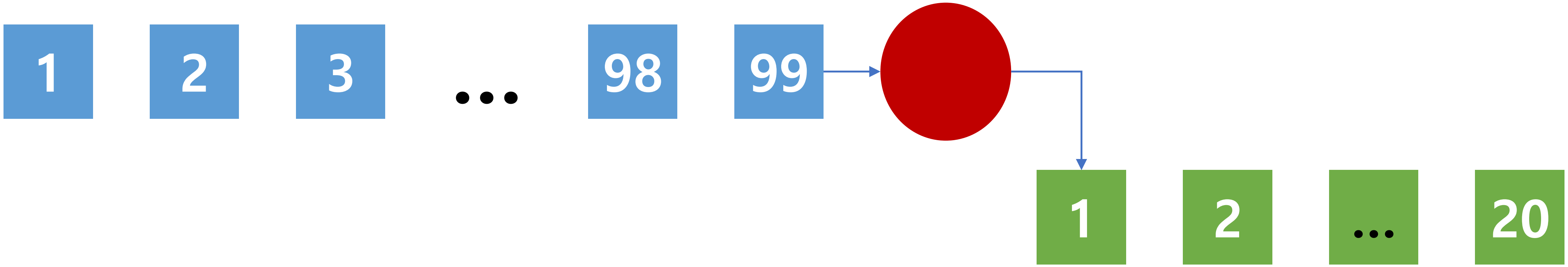


How to neural machine translation?



We call it Encoder Decoder Architecture or *seq2seq model*

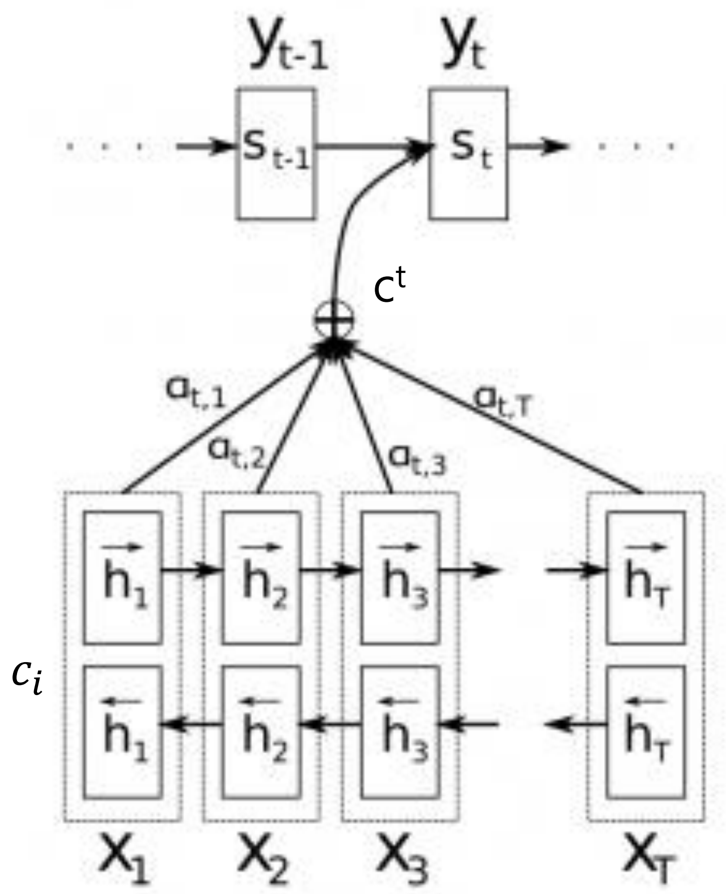
Seq2seq Model Limitation



문장의 단어가 많은 경우 Context Vector의 크기가 충분하지 않으면
모든 정보를 표현하기 어렵기 때문에 번역이 잘 안되는 문제가 발생

How can we improve this?

Seq2seq Model with Attention mechanism

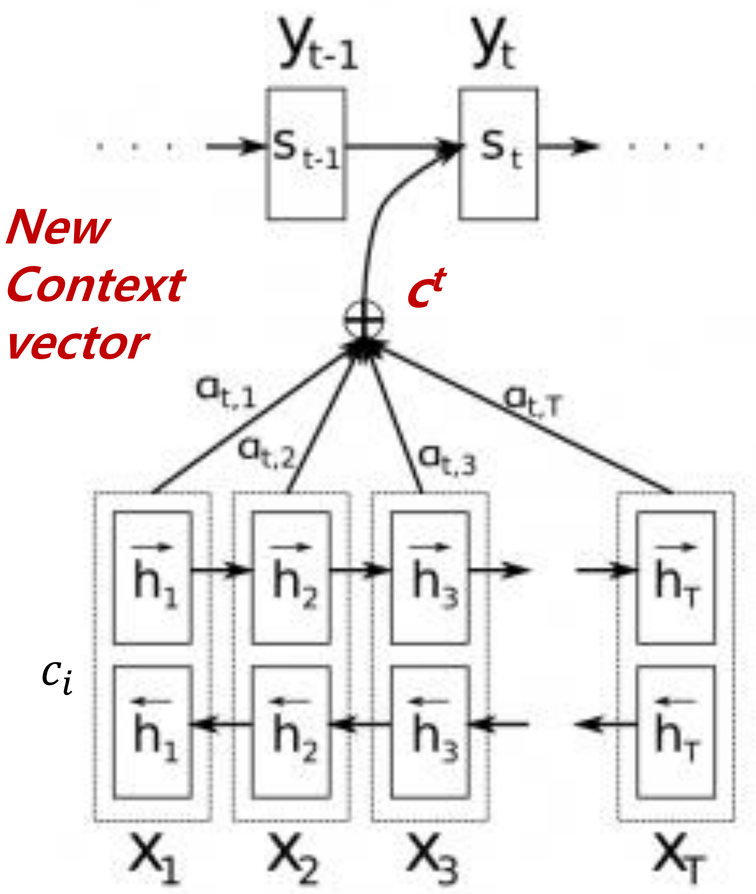


Encoder의 전체 h를 고려함.

한 state 마다
Attention이 적용되어서
y에 관련된 x의 h로 context vector로 만들어
Decoder로 전달됨.

How can we improve this?

Seq2seq Model with Attention mechanism

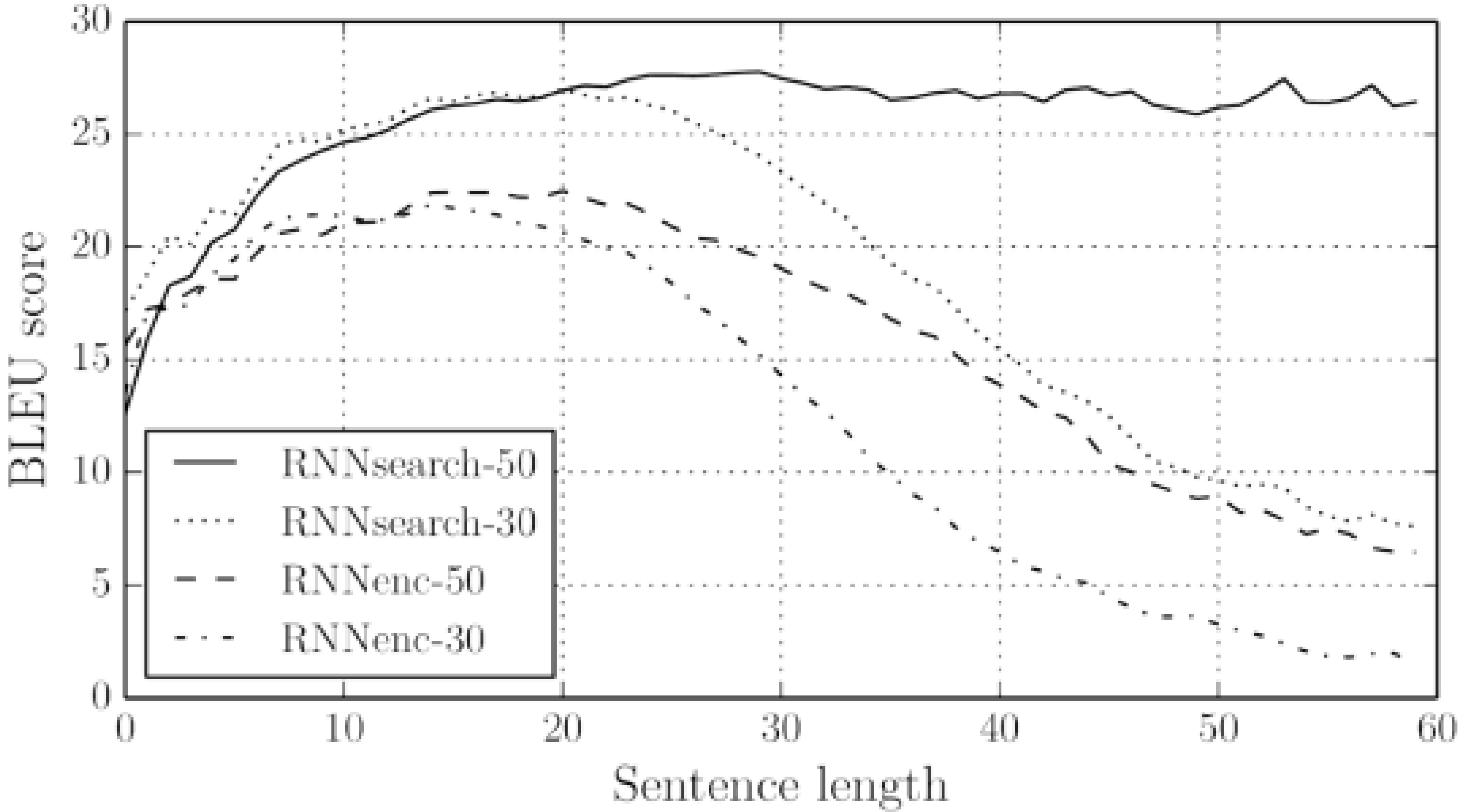


Encoder의 전체 h를 고려함.

한 state 마다
Attention이 적용되어서
y에 관련된 x의 h로 context vector로 만들어
Decoder로 전달됨.

- $e_i^t = f(c_i, s_{t-1})$
- $a_i^t = softmax(e_i^t)$ # 0~1사이의 확률 값으로 변환
  # Attention Weight [Energy]
- $C^t = \sum_{i=1}^{T} a_i^t c_i$

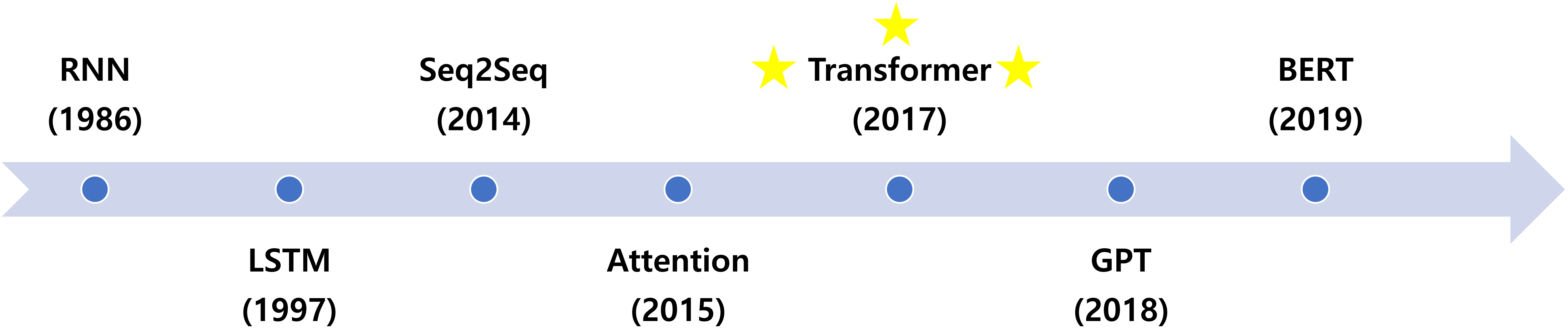
Seq2Seq with Attention Result



Neural Machine Translation Milestones

- **2021년 기준 최신 고성능 모델들은 Transformer 아키텍처를 기반으로 사용되고 있음**

GPT : Transformer의 Decoder 아키텍처 사용
BERT : Transformer의 Encoder 아키텍처 사용



Transformer

- **3줄 요약**

  ◆ **Evolution of encoder & decoder architecture**

  ◆ **You even don’t need neither RNN nor CNN**

  ◆ **Faster train, but better performance**

Transformer Architecture

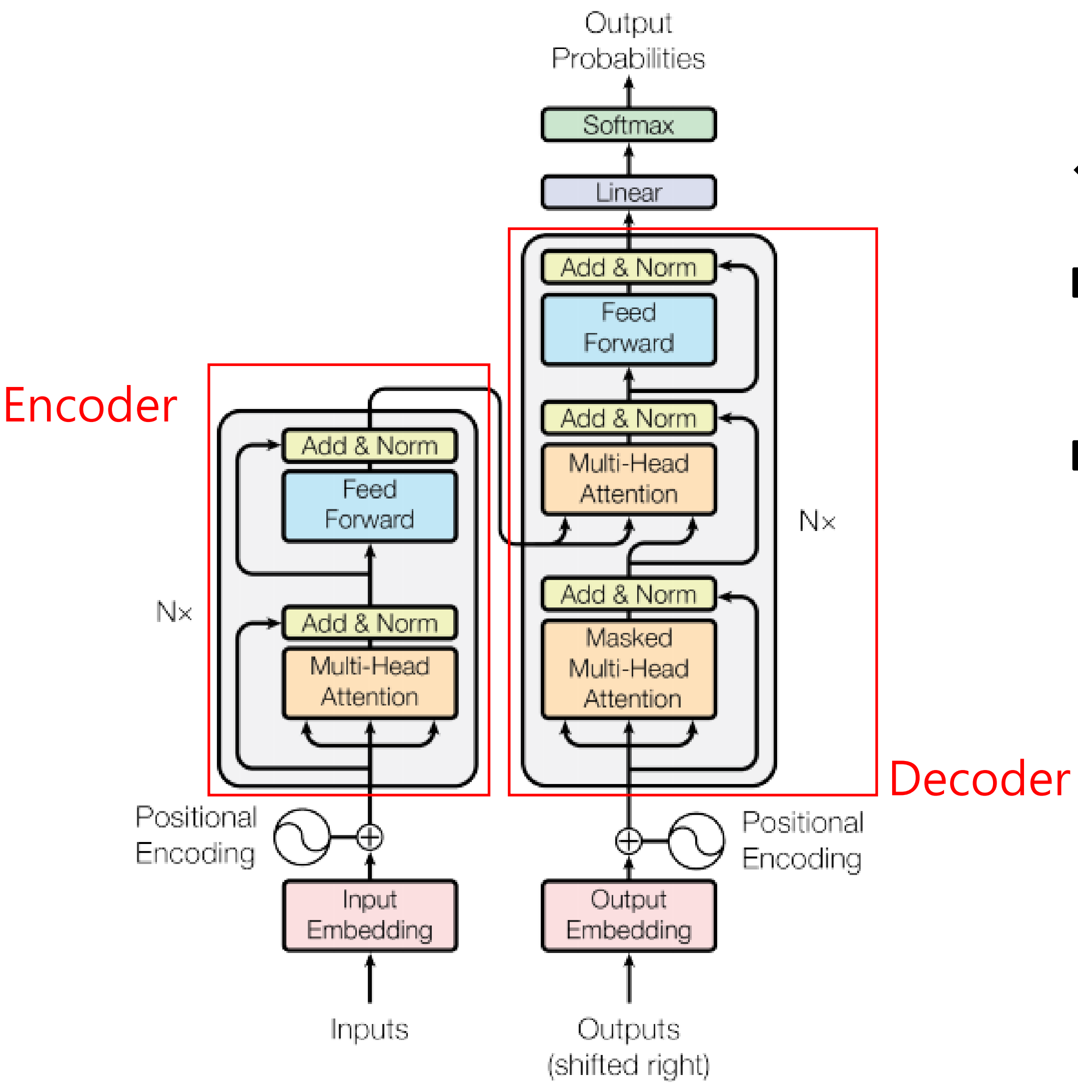


Figure 1: The Transformer - model architecture.

◆ 주요 모듈

Encoder :

Multi Head Attention

FFNN

Decoder :

Masked Multi Head Attention

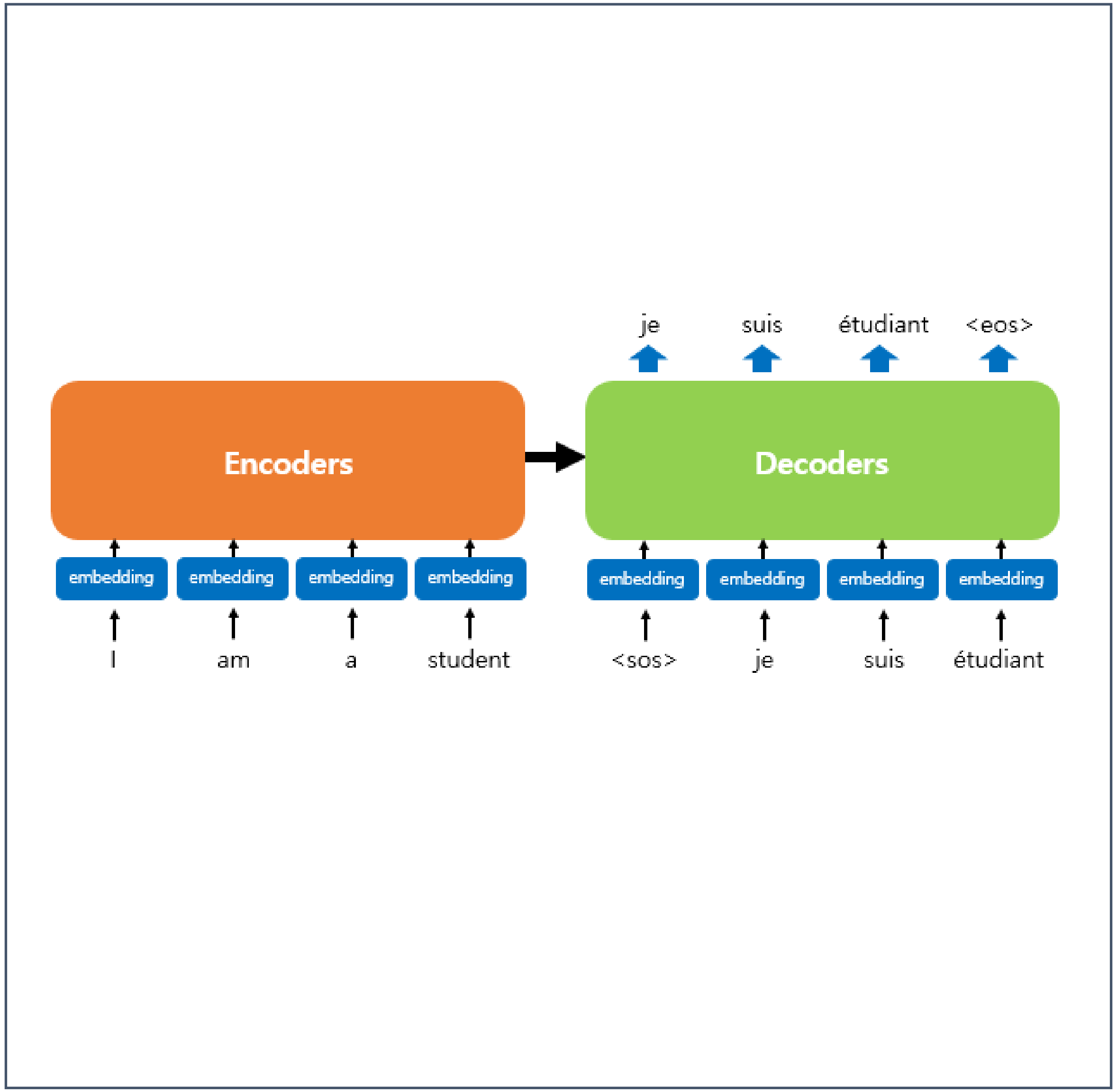
Multi Head Attention

FFNN

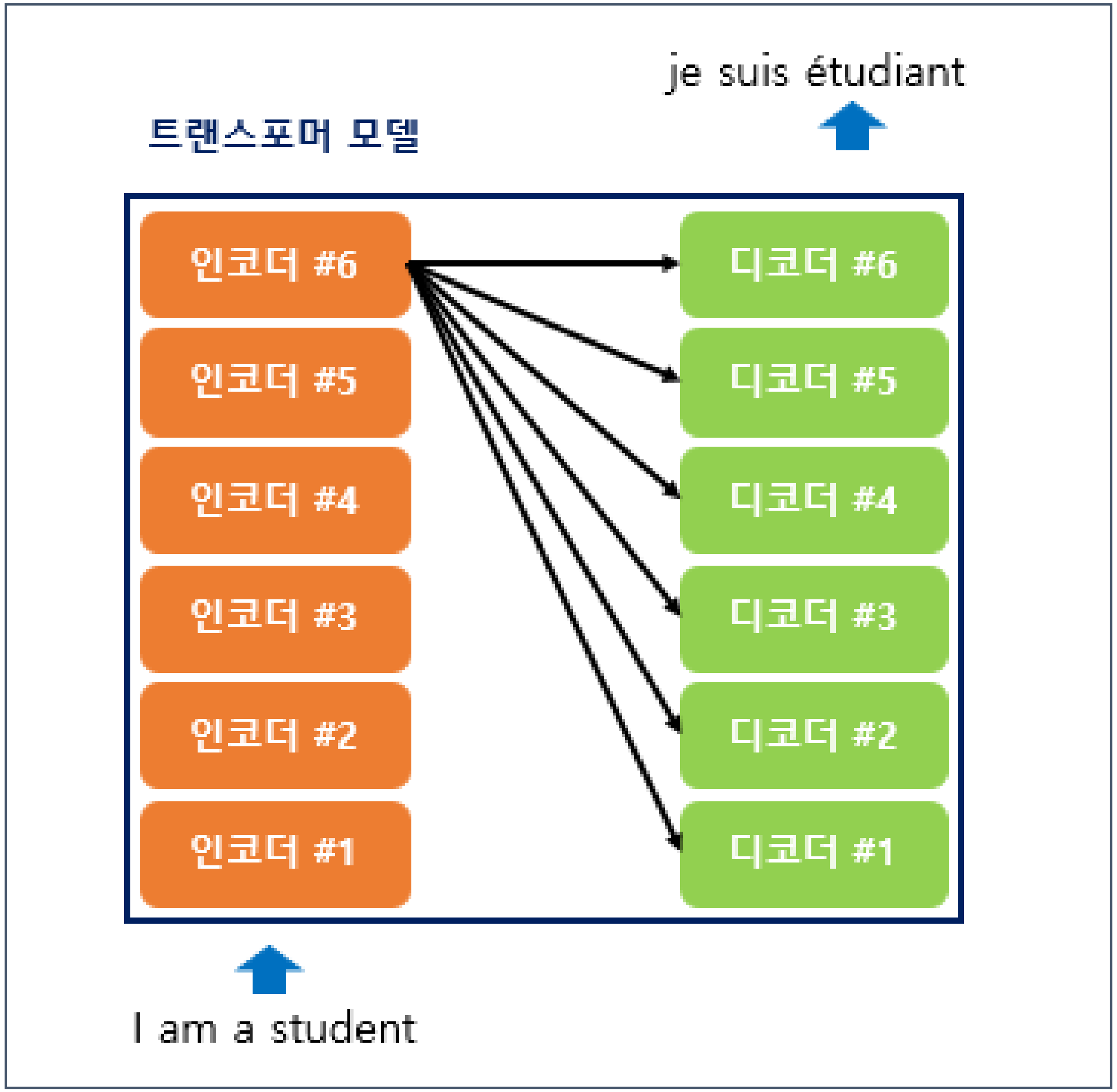
+) Residual layers,

+) N is 6

Simple Architecture



인코더와 디코더가 1개씩 존재하는 트랜스포머의 구조



인코더와 디코더가 6개씩 존재하는 트랜스포머의 구조

Positional Encoding

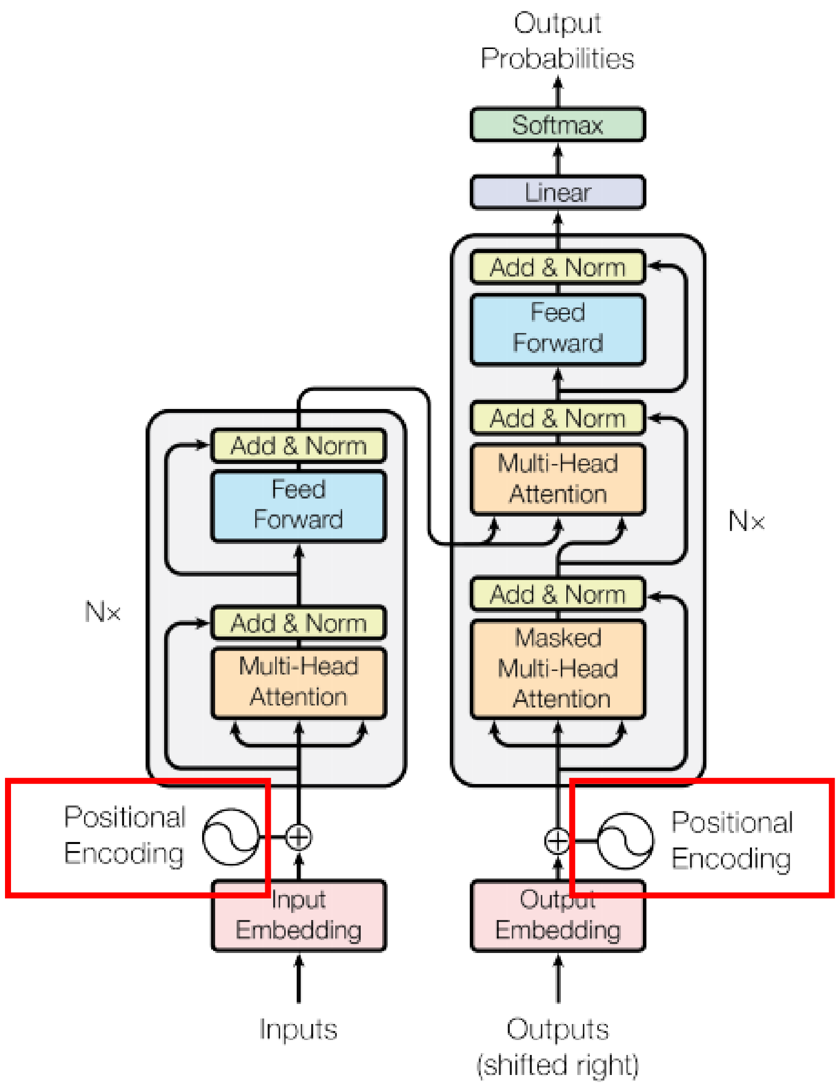
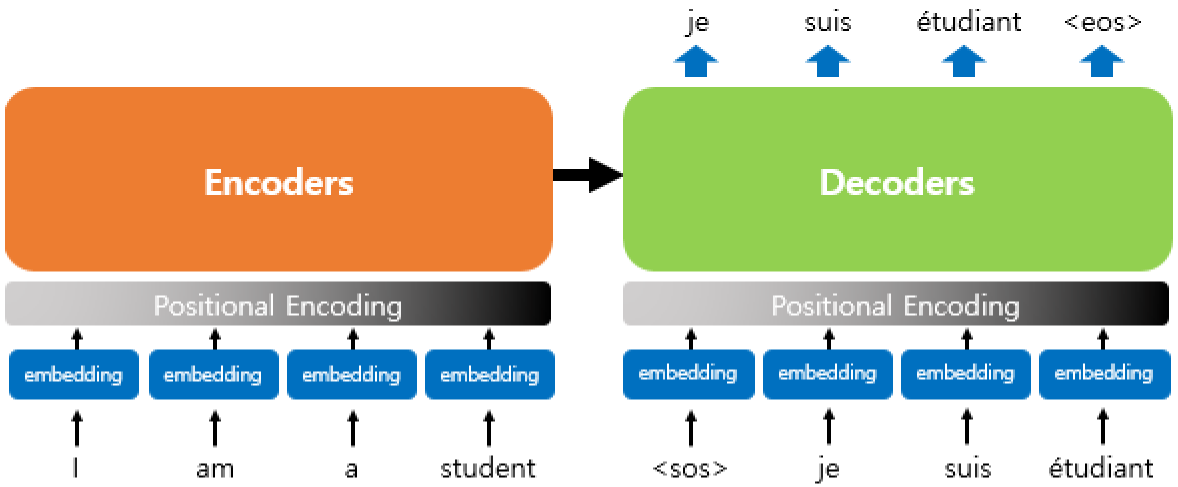
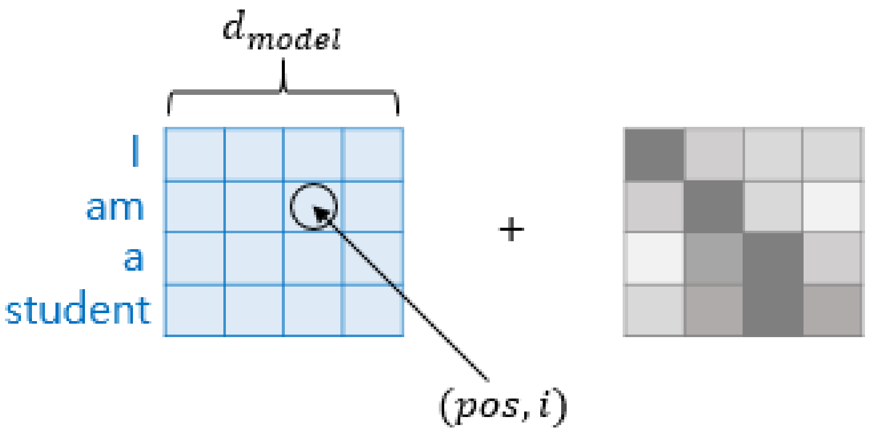


Figure 1: The Transformer - model architecture.

Positional Encoding

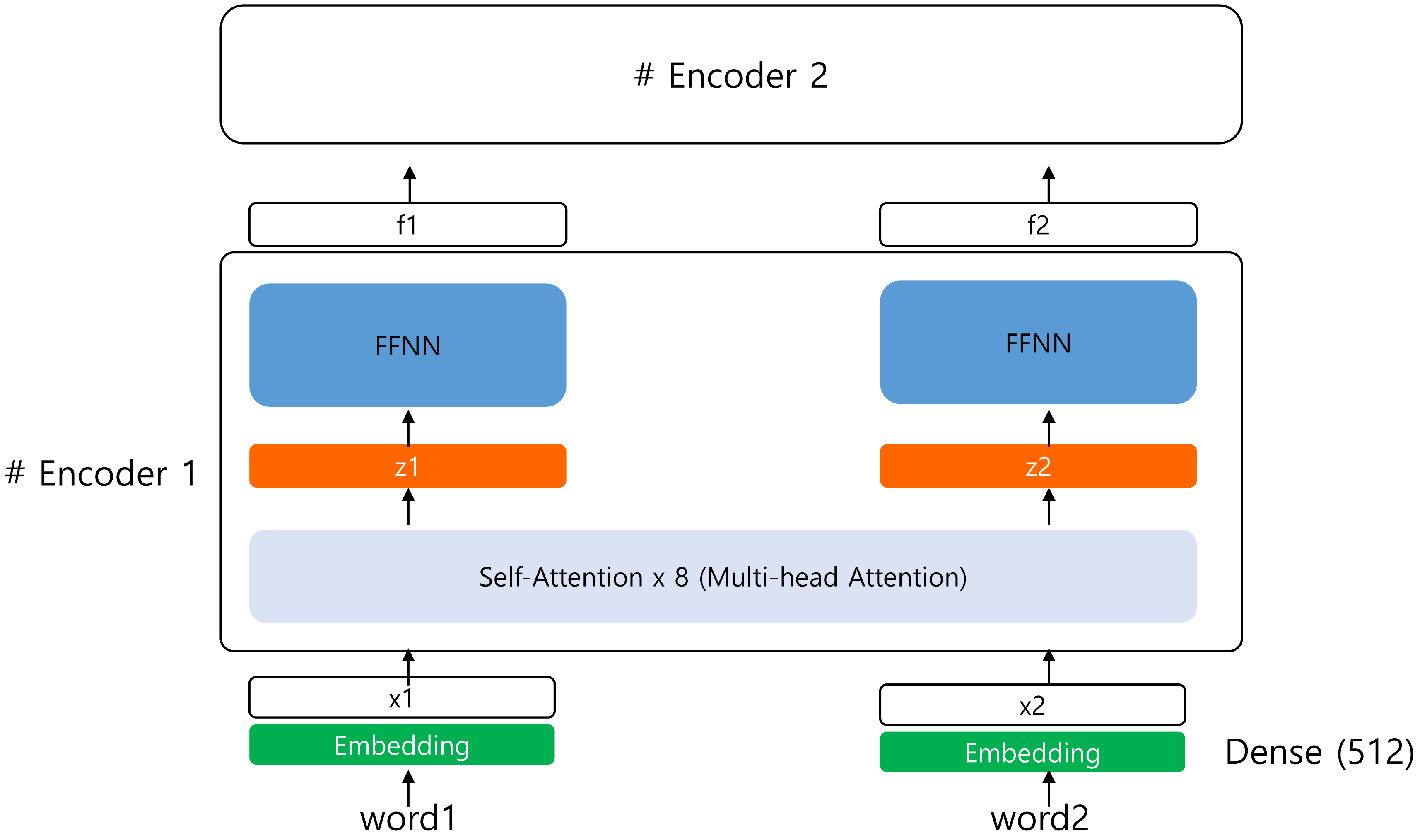




$$PE_{(pos,\,2i)} = sin(pos/10000^{2i/d_{model}})$$

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

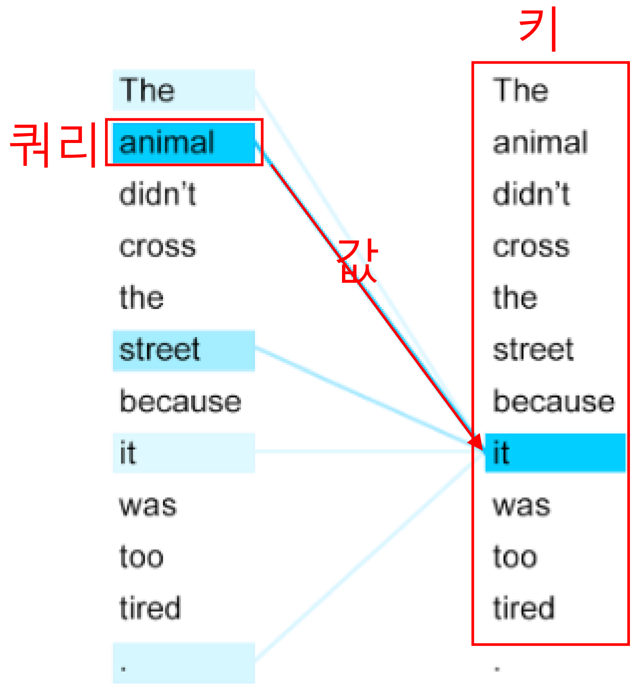
Encoder


Encoder 2
f1
f2
FFNN
FFNN
Encoder 1
z1
z2
Self-Attention x 8 (Multi-head Attention)
x1
x2
Embedding
Embedding
Dense (512)
word1
word2

Encoder #self-Attention

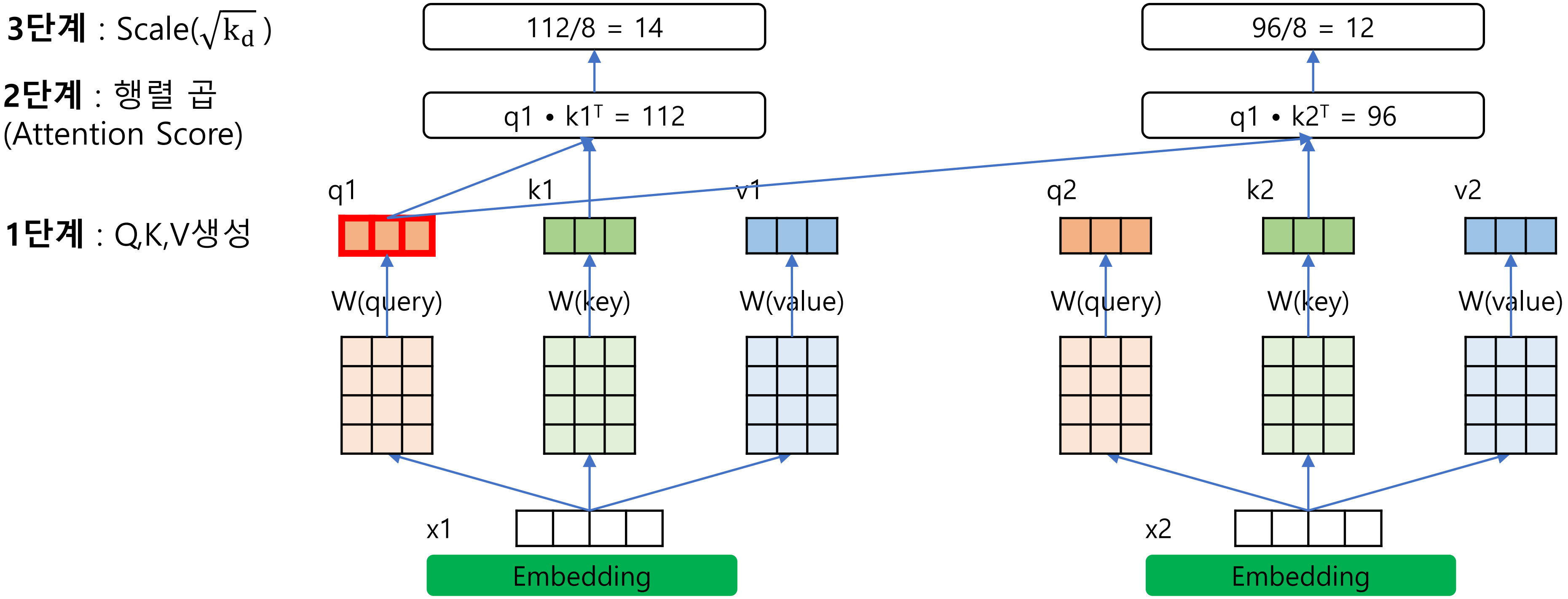
Attention

- 주어진 '쿼리(Query)'에 대해서 모든 '키(Key)'와의 유사도를 각각 구함.
- 그리고 구해낸 이 유사도를 가중치로 해서 키와 Mapping되어있는 각각의 '값(Value)'에 반영함
- 그리고 유사도가 반영된 '값(Value)'을 모두 Weighted Sum



Encoder #self-Attention

- **Transformer : Self Attention**



Encoder #self-Attention

- **Transformer : Self Attention**

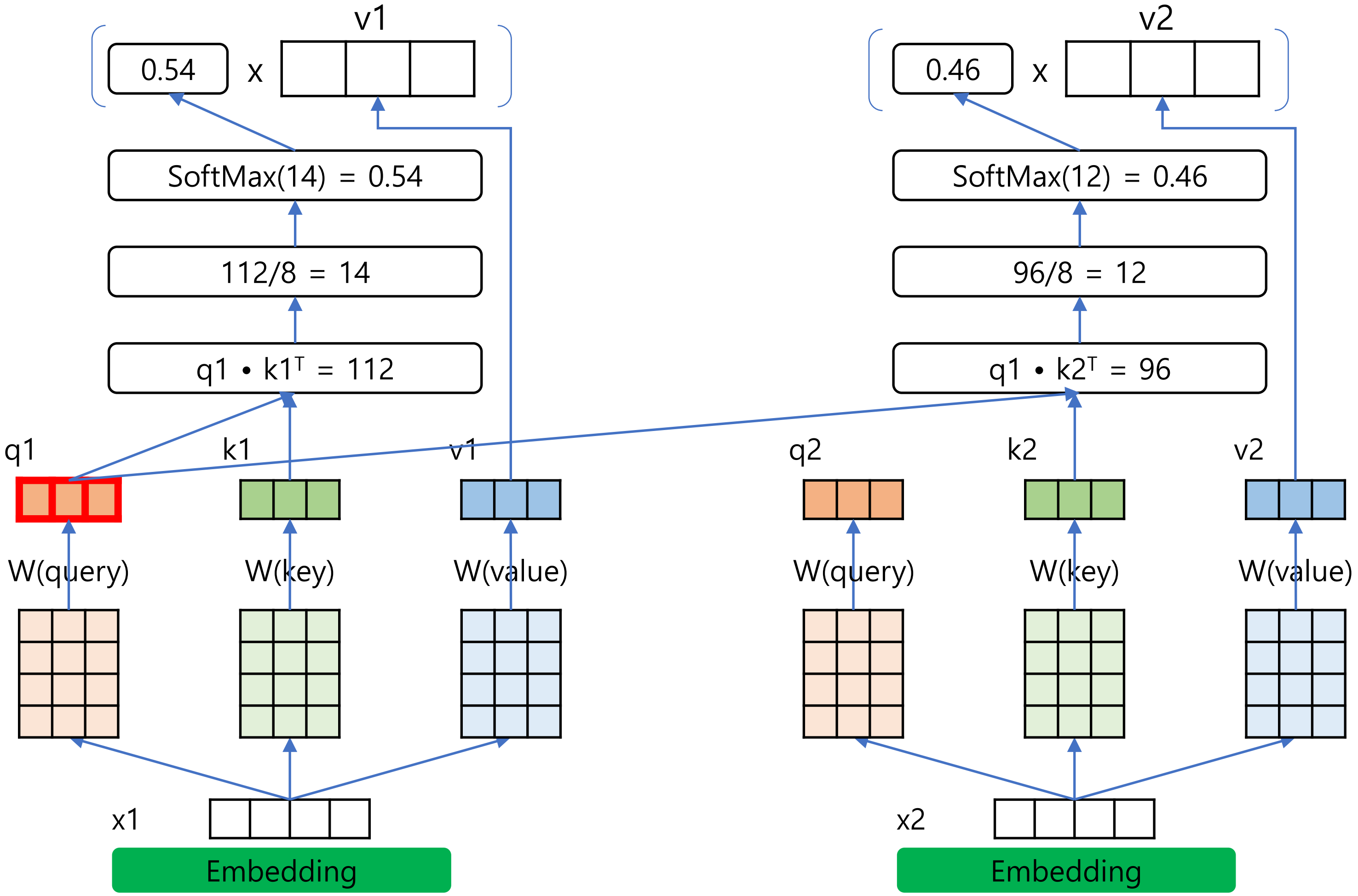
5단계 : 행렬 곱

4단계 : SoftMax

3단계 : Scale($\sqrt{k_d}$)

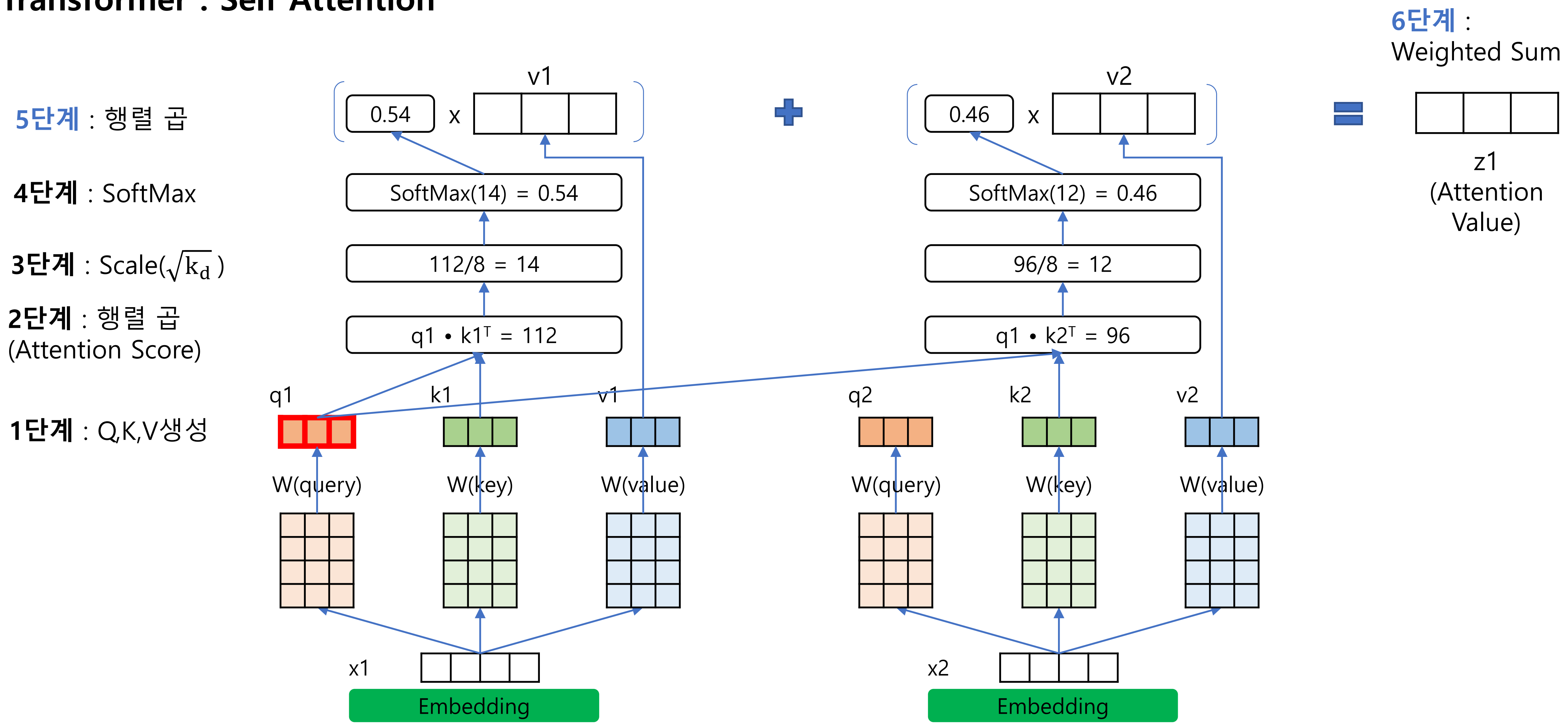
2단계 : 행렬 곱
(Attention Score)

1단계 : Q,K,V생성



Encoder #self-Attention

- **Transformer : Self Attention**

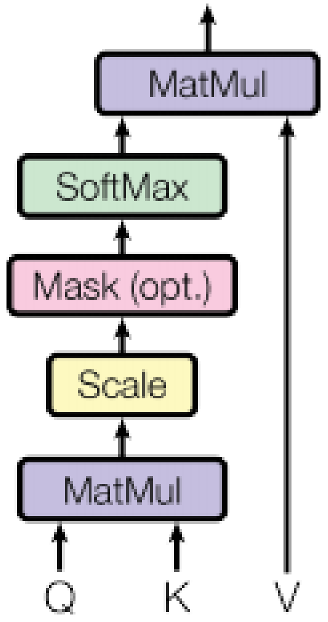


Encoder #self-Attention

- **Transformer : Self Attention**

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

Scaled Dot-Product Attention



Encoder #self-Attention

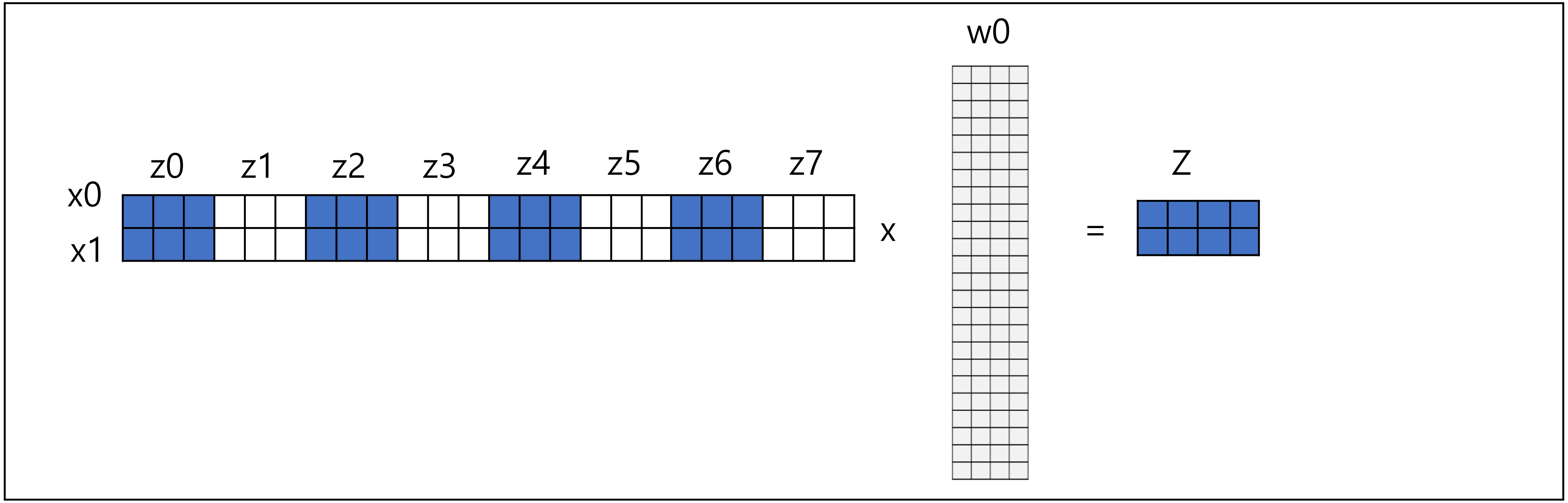
- **Self Attention X 8 = Multi-head Attention**

◆ Self Attention을 여러 번 처리한 것으로 보면 됨.

◆ Convolution의 필터를 여러 개 두는 것처럼 Attention을 여러 번 적용하면 다양한 표현을 학습하는 효과를 얻음.

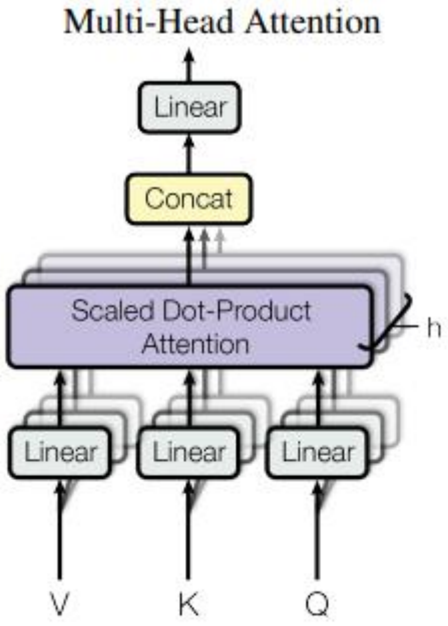
◆ 8개의 Q, K, V를 가지고 결과값 Z 8개 출력

◆ 이때 Z는 Concatenate 시켜주고 Weight 행렬을 곱해서 하나의 Z로 만들어 버림



Encoder #self-Attention

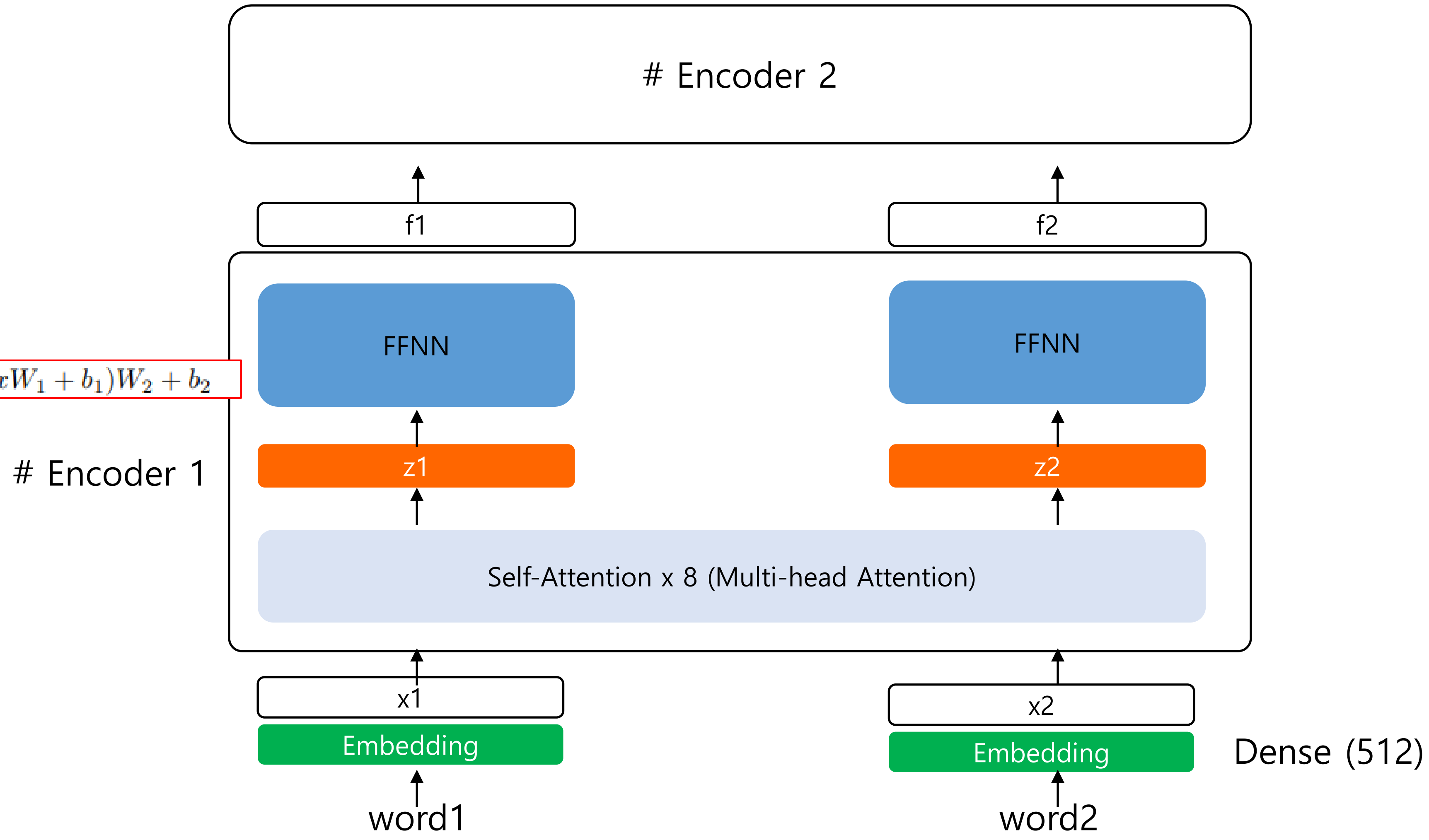
- **Self Attention X 8 = Multi-head Attention**



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

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Encoder #FFNN

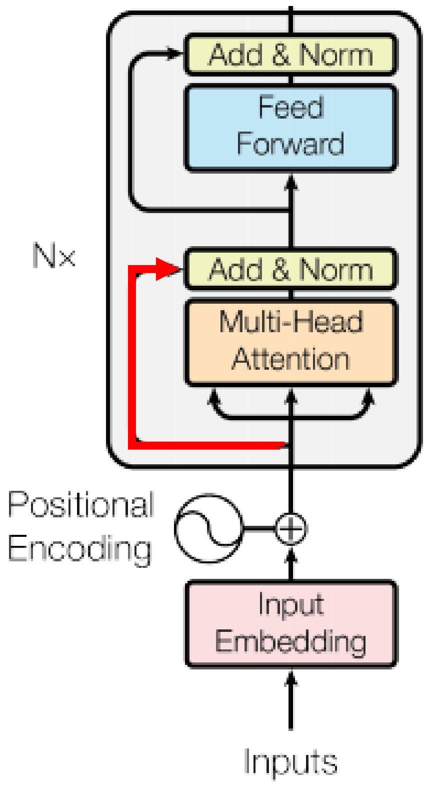


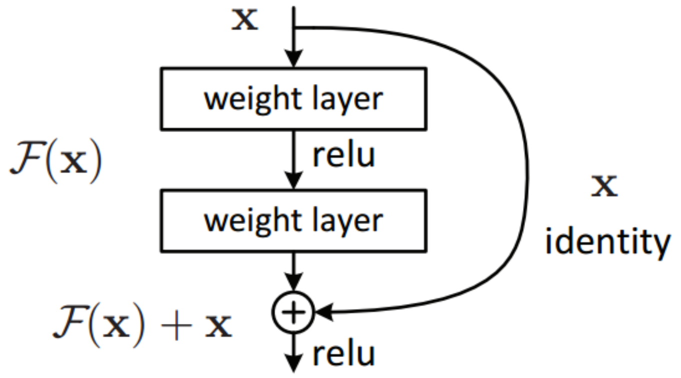
$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Encoder #Residual layer

◆ 정보손실을 막을 수 있게 Residual connection을 하여 이전 정보를 더해줘 모델 학습을 도움

◆ Residual connection을 거친 결과는 이어서 층 정규화 과정을 거치게 됨





Decoder #Masked multi head attention

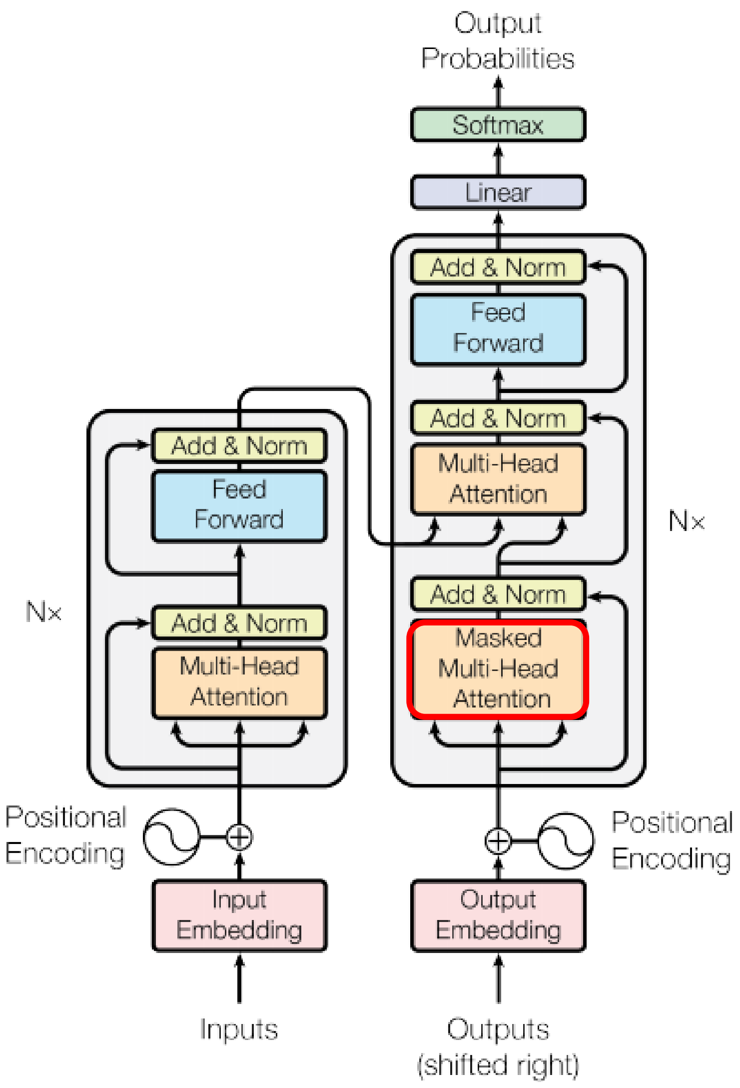
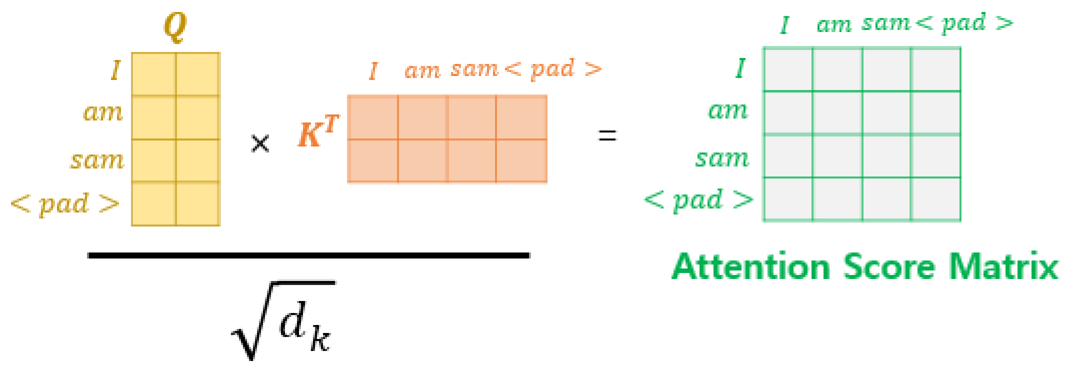
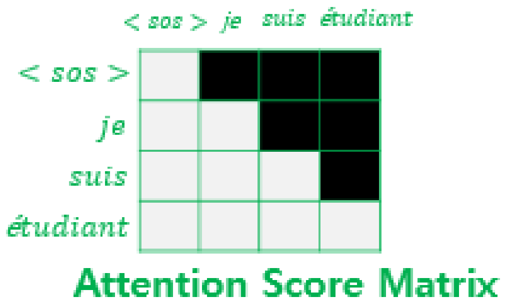


Figure 1: The Transformer - model architecture.





Decoder #인코더-디코더 attention

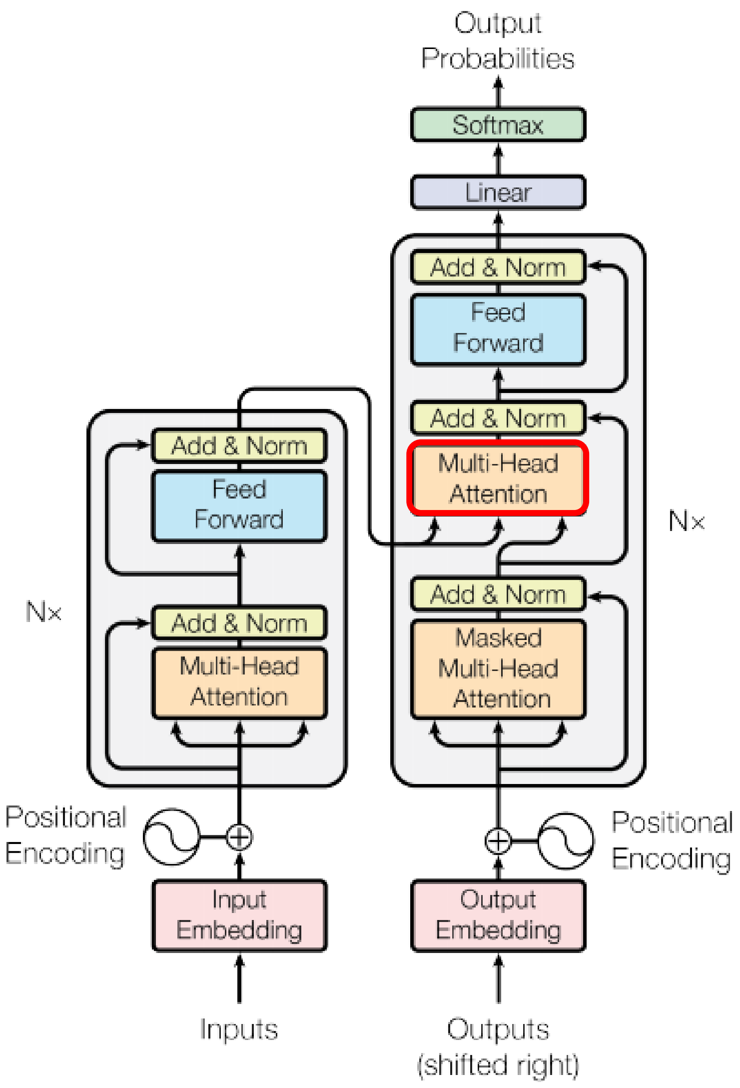
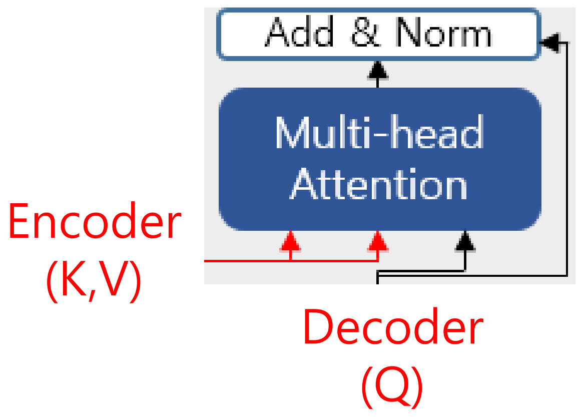
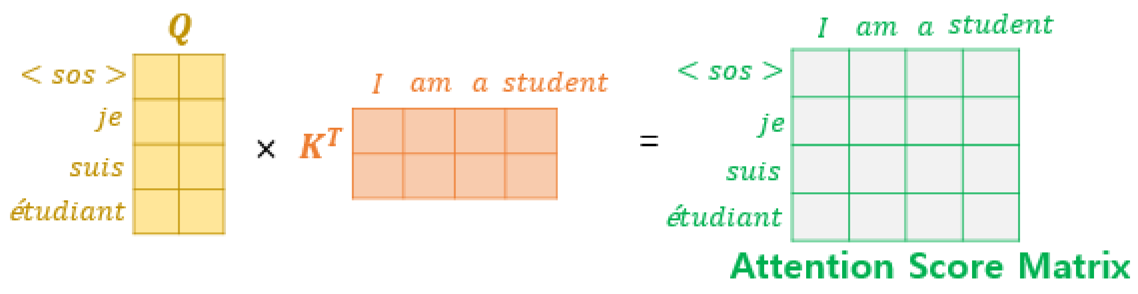


Figure 1: The Transformer - model architecture.





Label Smoothing

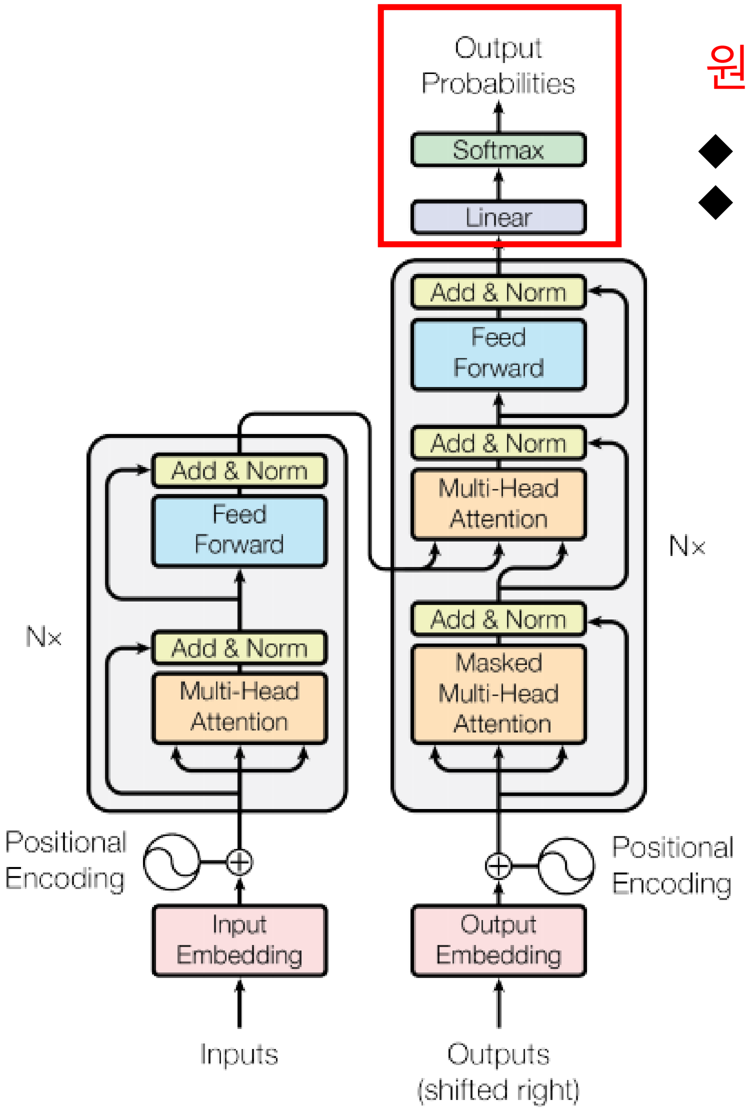


Figure 1: The Transformer - model architecture.

원 핫 대신 Label Smoothing

- ◆ 정답은 1에 가까운 값 오답은 0에 가까운 값으로 출력해서
- ◆ 유사한 단어에 대해서도 잘 학습되게 함.

Reference

- [Original paper] https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
- http://jalammar.github.io/illustrated-transformer/
- https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- 허민석/트랜스포머/ https://youtu.be/mxGCEWOxfe8
- 딥 러닝을 이용한 자연어 처리 입문 /https://wikidocs.net/31379