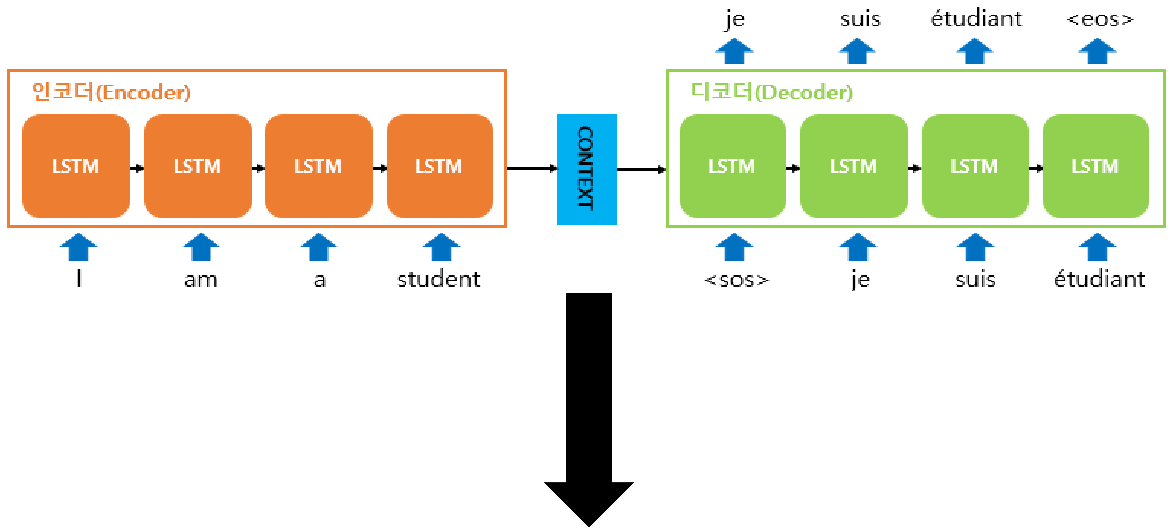
Transformer

- Encoder

23.08.29 유하영

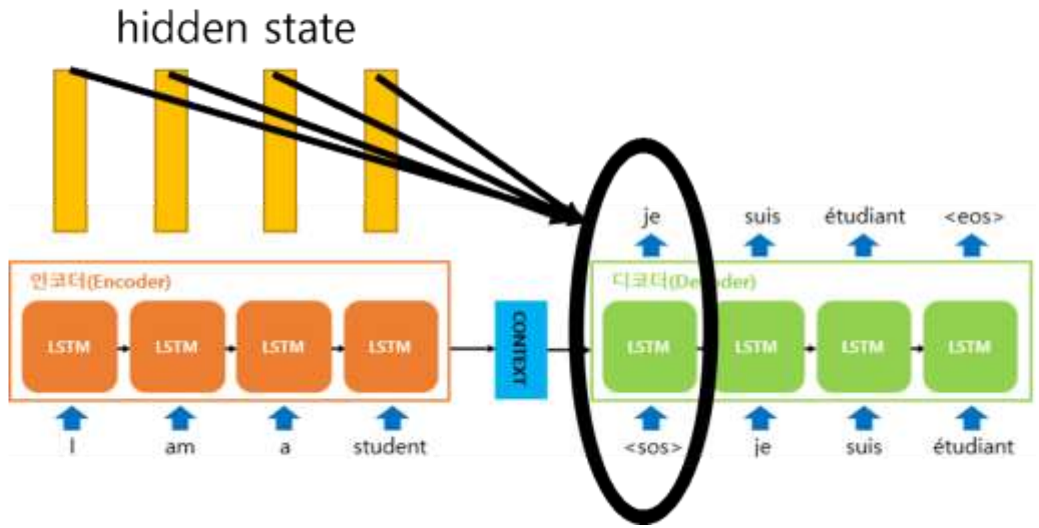
seq2seq



Encoder의 마지막 hidden state만
참조해 출력

-> 정보 손실

Attention

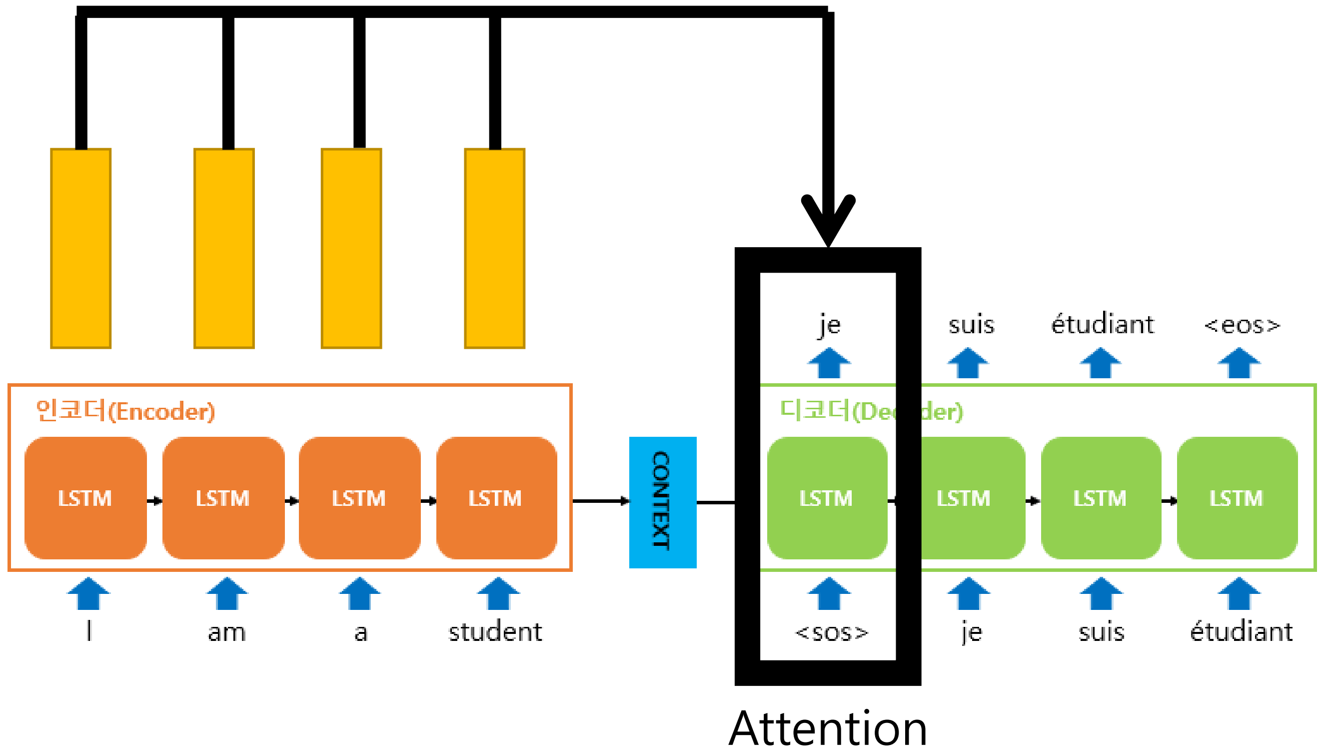


Decoder의 매 timestep마다
Encoder의 모든 hidden state 값들을
Attention 한다

Attention mechanism

1) Attention score

dot product

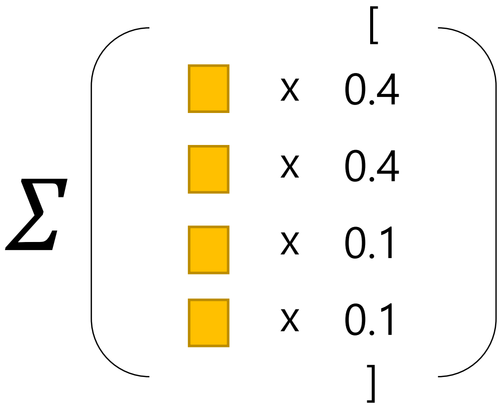


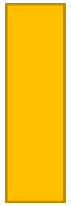
2) Attention distribution

softmax()

= [0.4 0.4 0.1 0.1]

3) Attention value

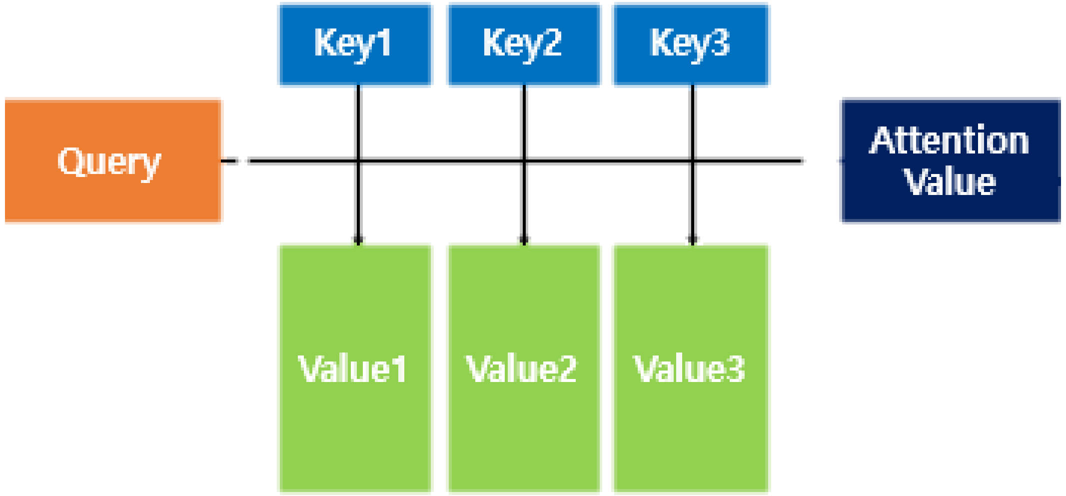




: word's
hidden state

Attention(Q,K,V) = Attention Value

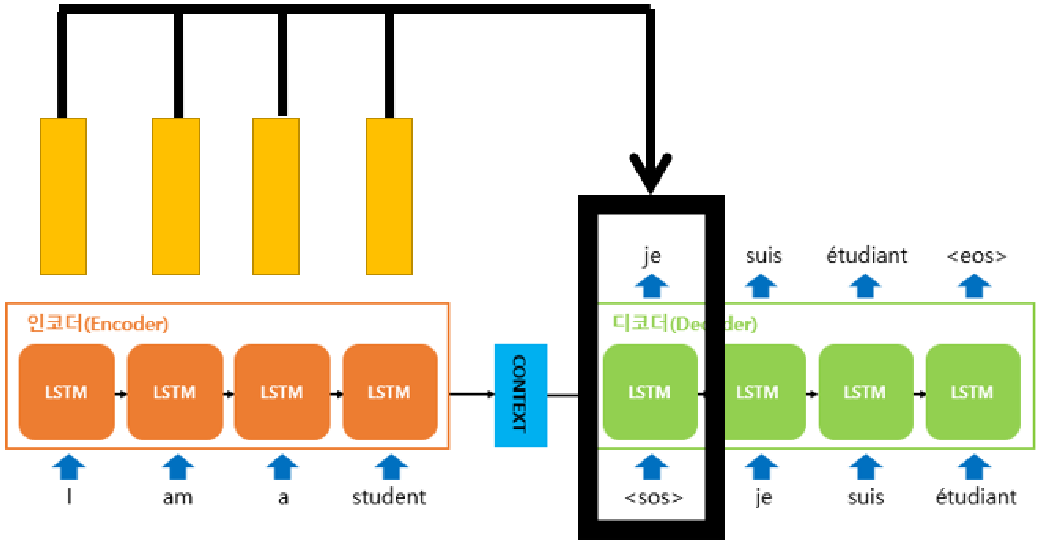
* 어텐션 메커니즘 구성요소를
간결하게 나타냄



value 집중할 위치의 정보
->가중치로 최종 값 계산

key 집중할 위치 결정 -> 가중치(연관성)를 계산

: **인코더**의 은닉상태



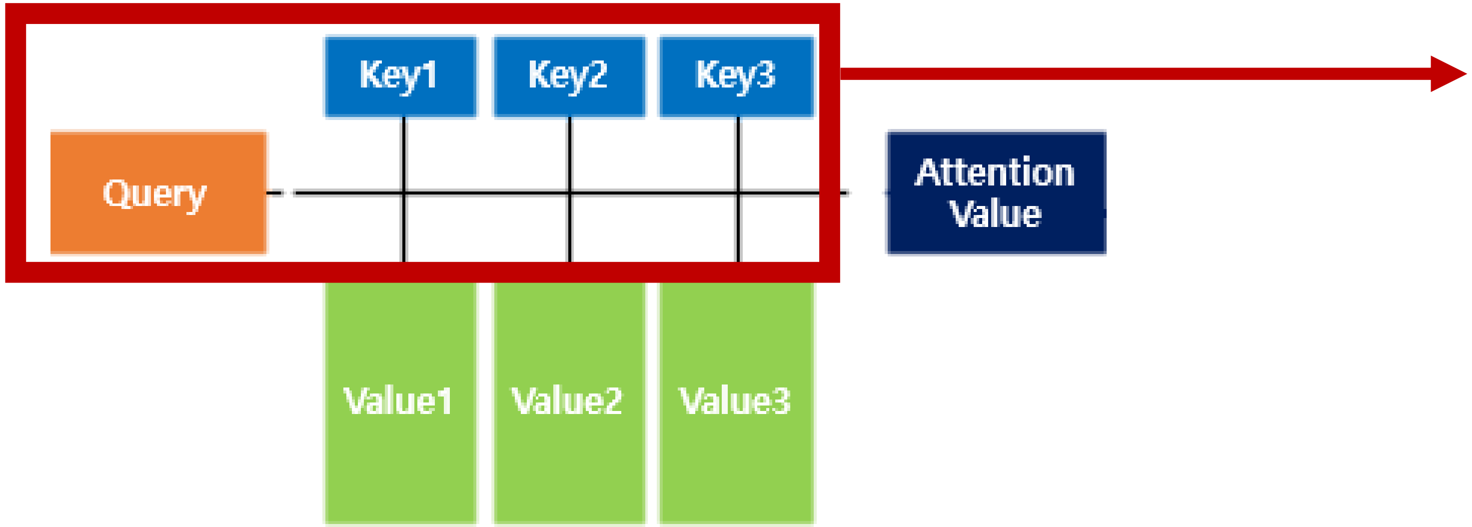
Attention

Query

: **디코더**의 은닉상태

Attention(Q,K,V) = Attention Value

* 어텐션 메커니즘 구성요소를
간결하게 나타냄



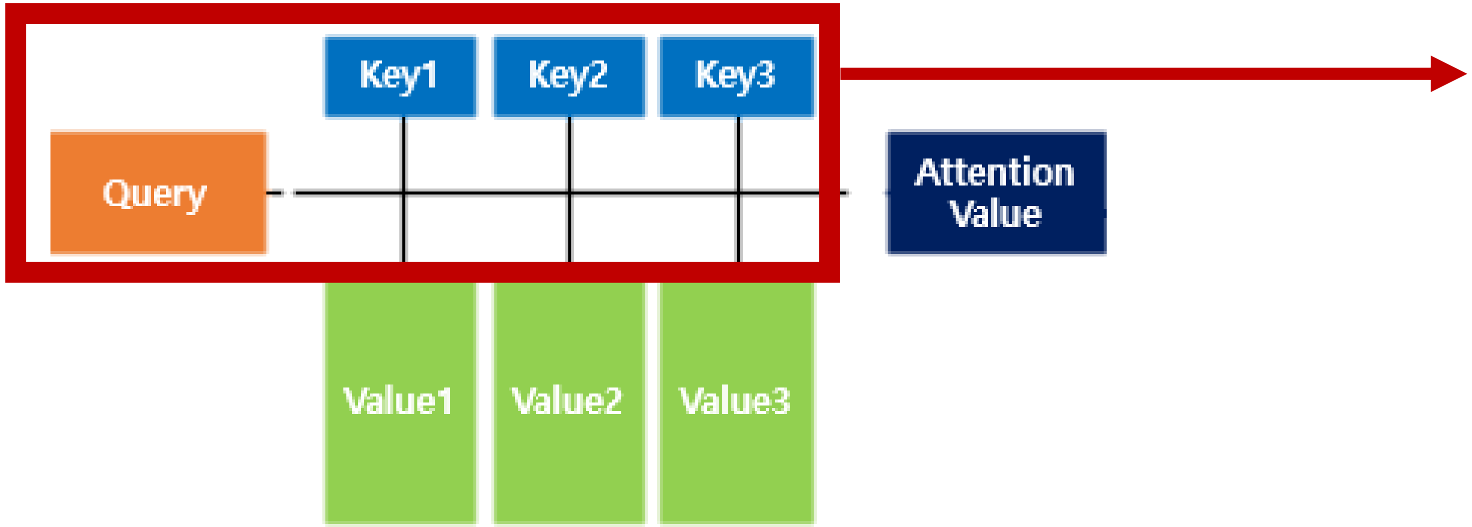
1) Attention score

Query와 Key 사이의

유사도 계산

Attention(Q,K,V) = Attention Value

* 어텐션 메커니즘 구성요소를
간결하게 나타냄

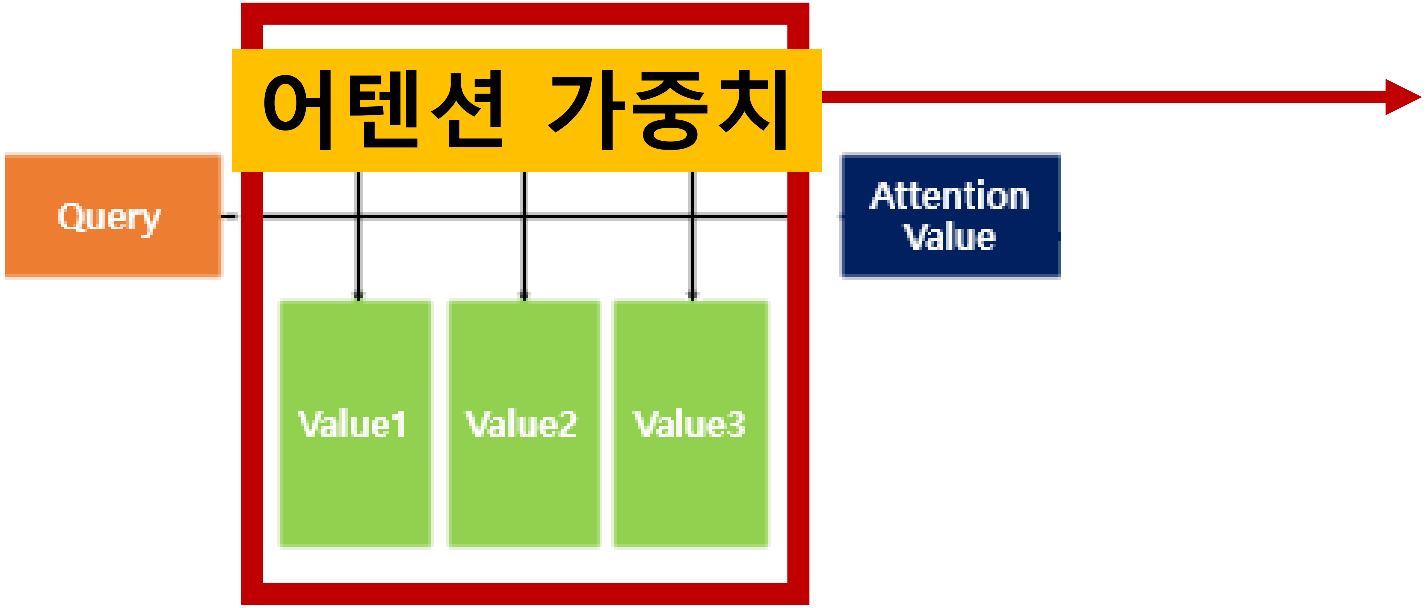


2) Attention distribution

softmax 함수를 적용하여
어텐션 가중치를 얻음

Attention(Q,K,V) = Attention Value

* 어텐션 메커니즘 구성요소를
간결하게 나타냄



3) Attention value

어텐션 가중치를
value에 가중합하여
Attention value를 얻는다

Attention

하나의 벡터로 압축하는 과정에서
여전히 **정보 손실**

RNN없이 **Attention만으로**
Encoder와 Decoder을 만들자

self - Attention

같은 문장 내에서
단어들 간의 관계를 고려하여
어텐션 계산

Transformer

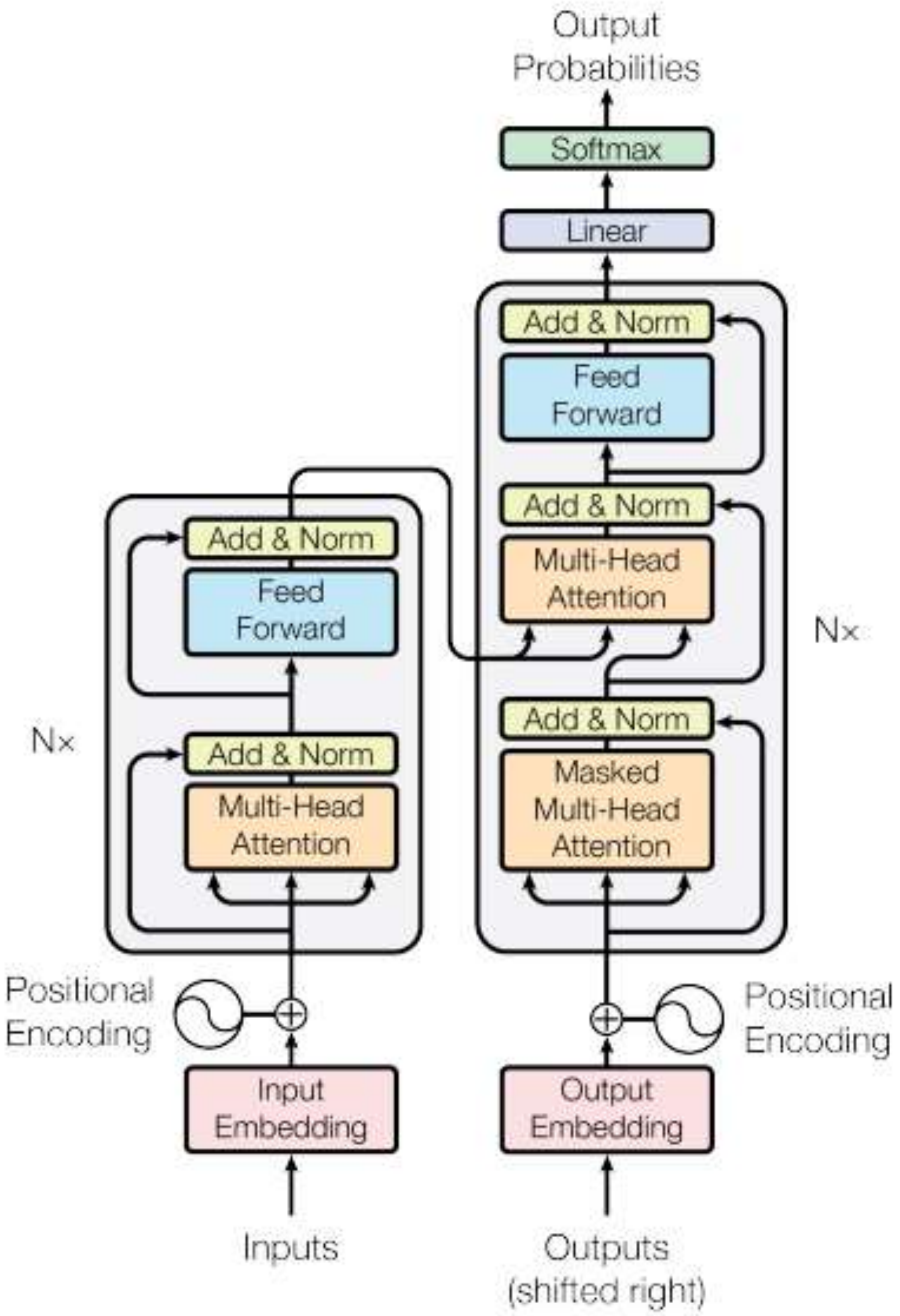
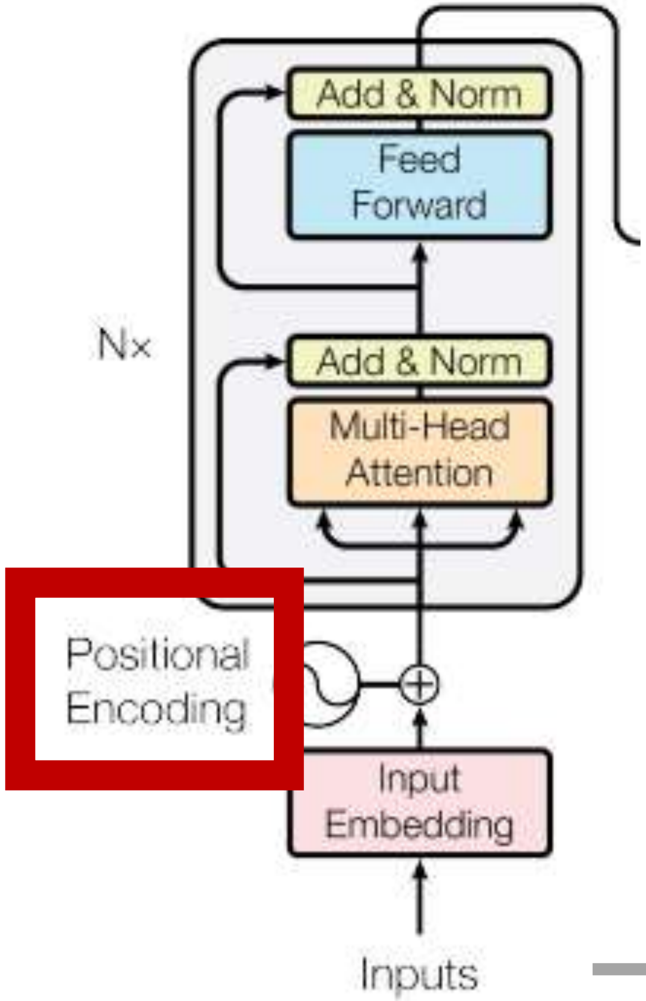
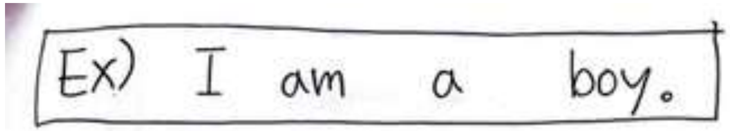


Figure 1: The Transformer - model architecture.

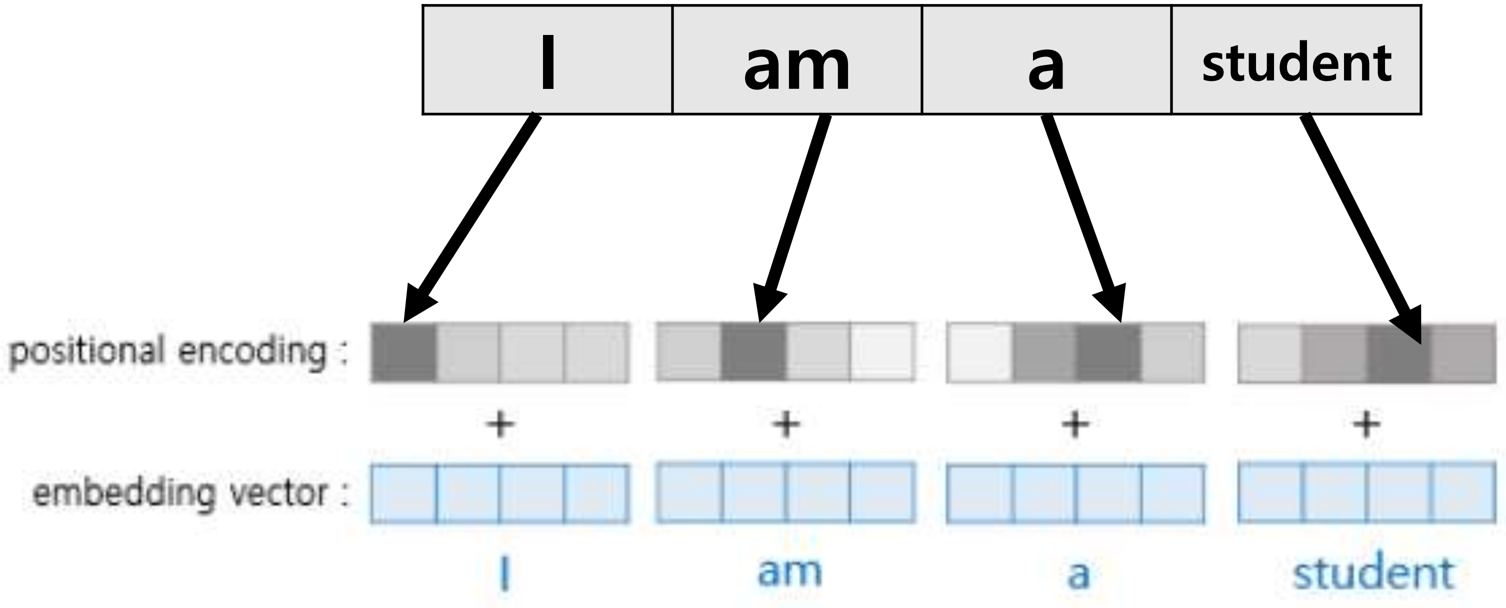
Transformer





Positional Encoding

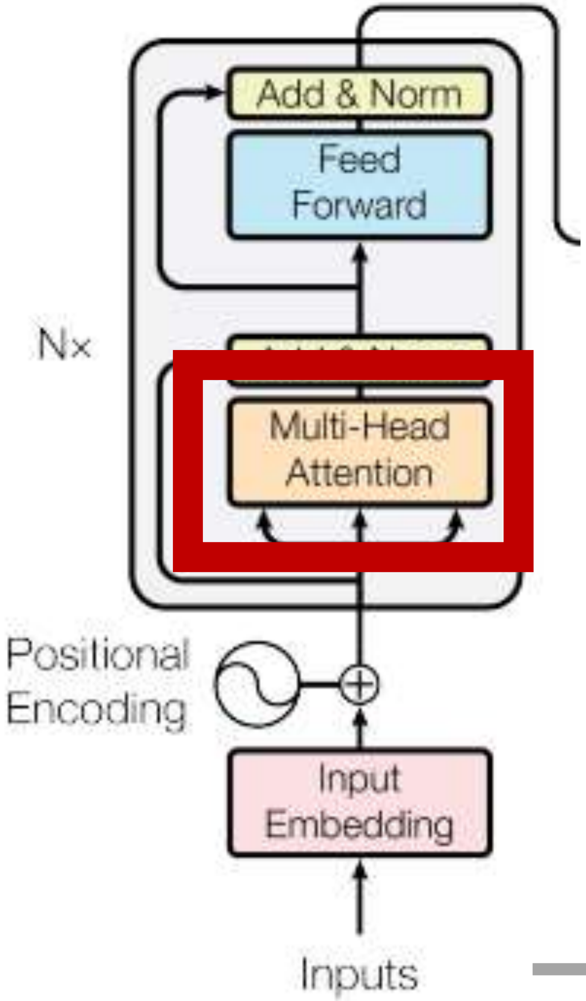
: 각 단어의 임베딩 벡터에 위치 정보를 더하는 것



같은 단어라도 **문장 내 위치에 따라**서
임베딩 벡터값이 달라짐

-> 순서 정보를 고려하기 위해 PE 반영

Transformer



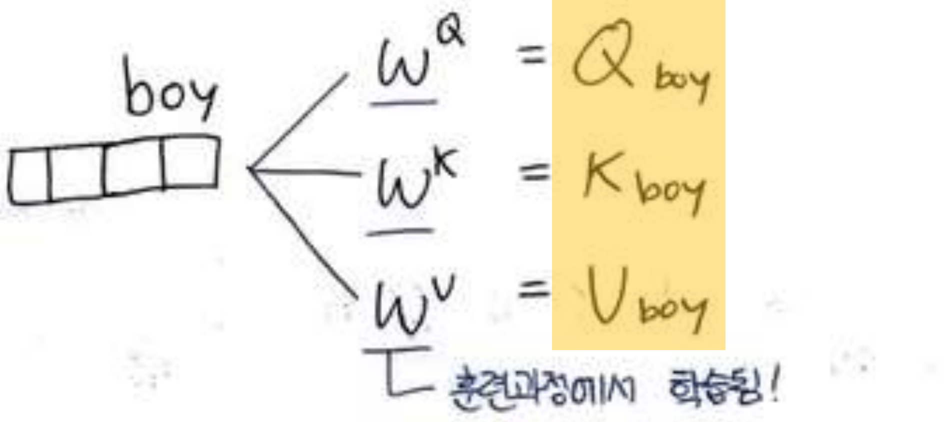
Ex) I am a boy.

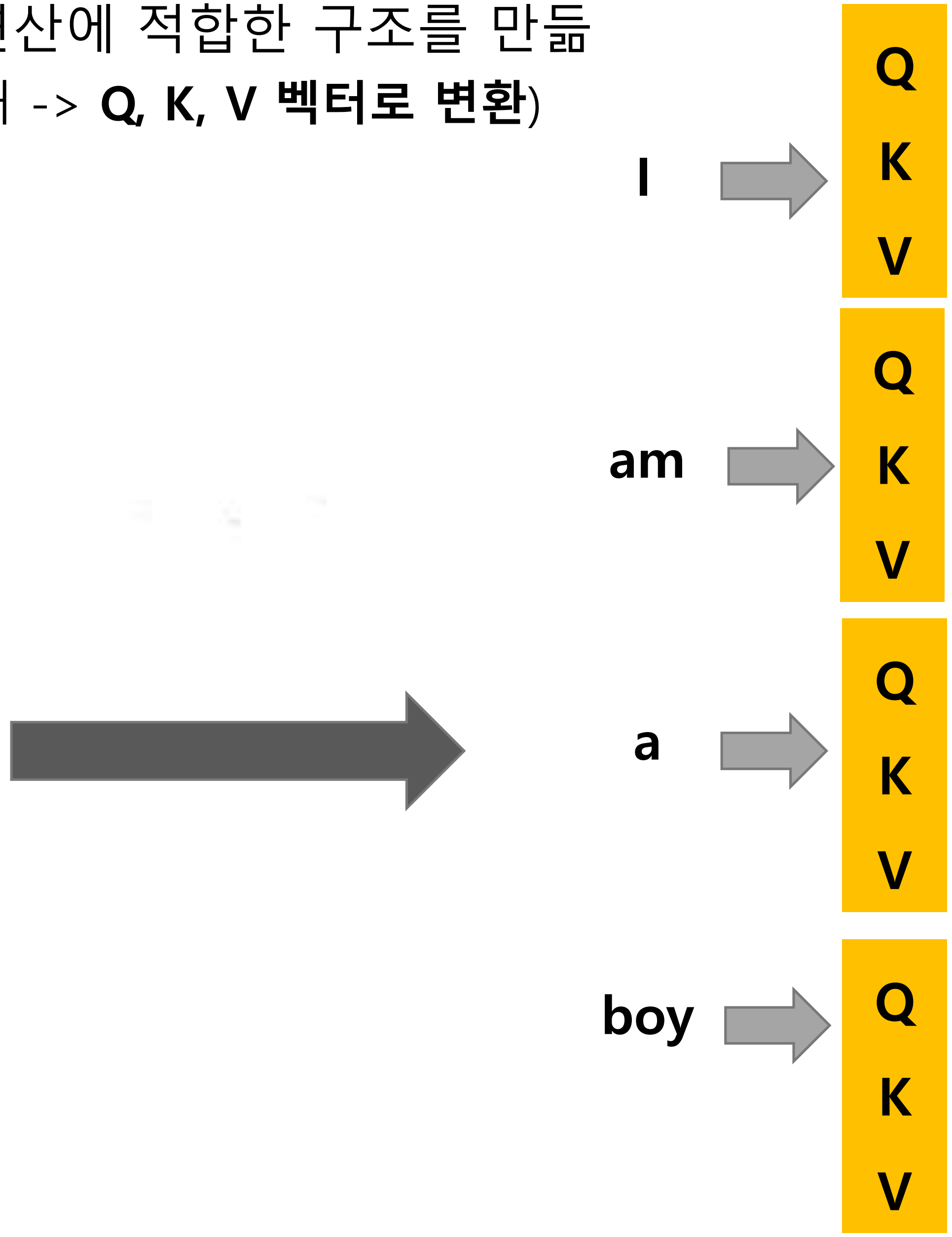
선형 연산

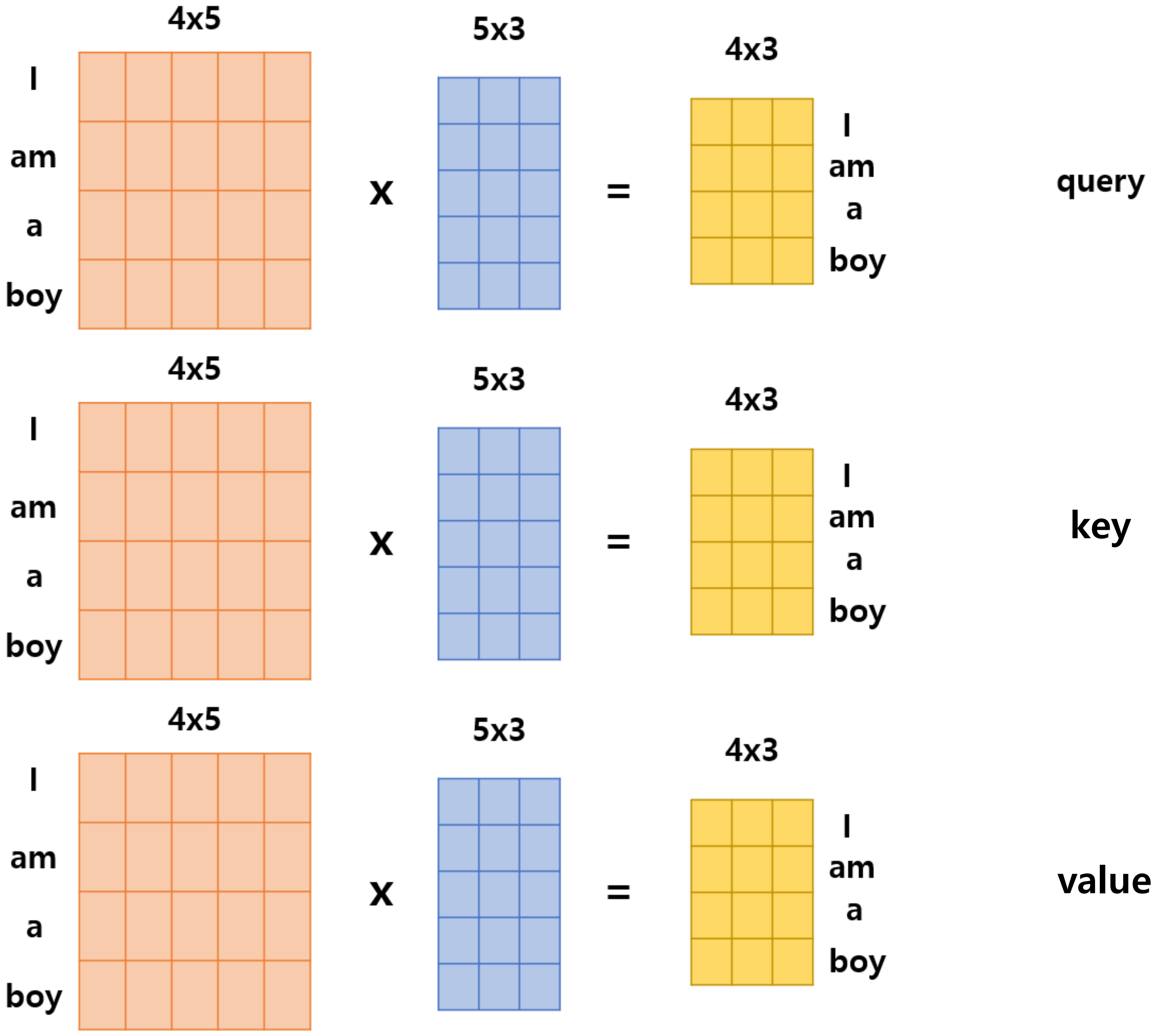
: 차원을 줄여서 병렬 연산에 적합한 구조를 만듦
(입력 문장 -> 임베딩 벡터 -> **Q, K, V 벡터로 변환**)

Ex) I am a boy.

1) 각 단어들로 부터 Q.K.V 벡터 얻기





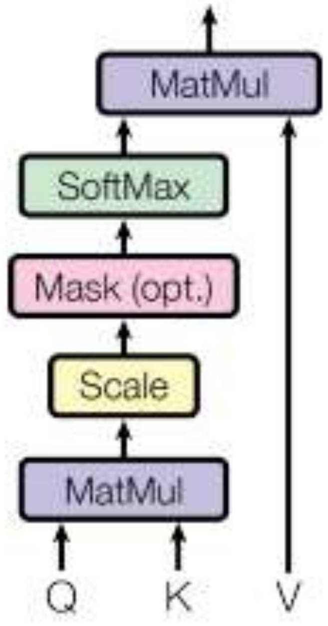


행렬 연산을 통한
일괄 계산 가능

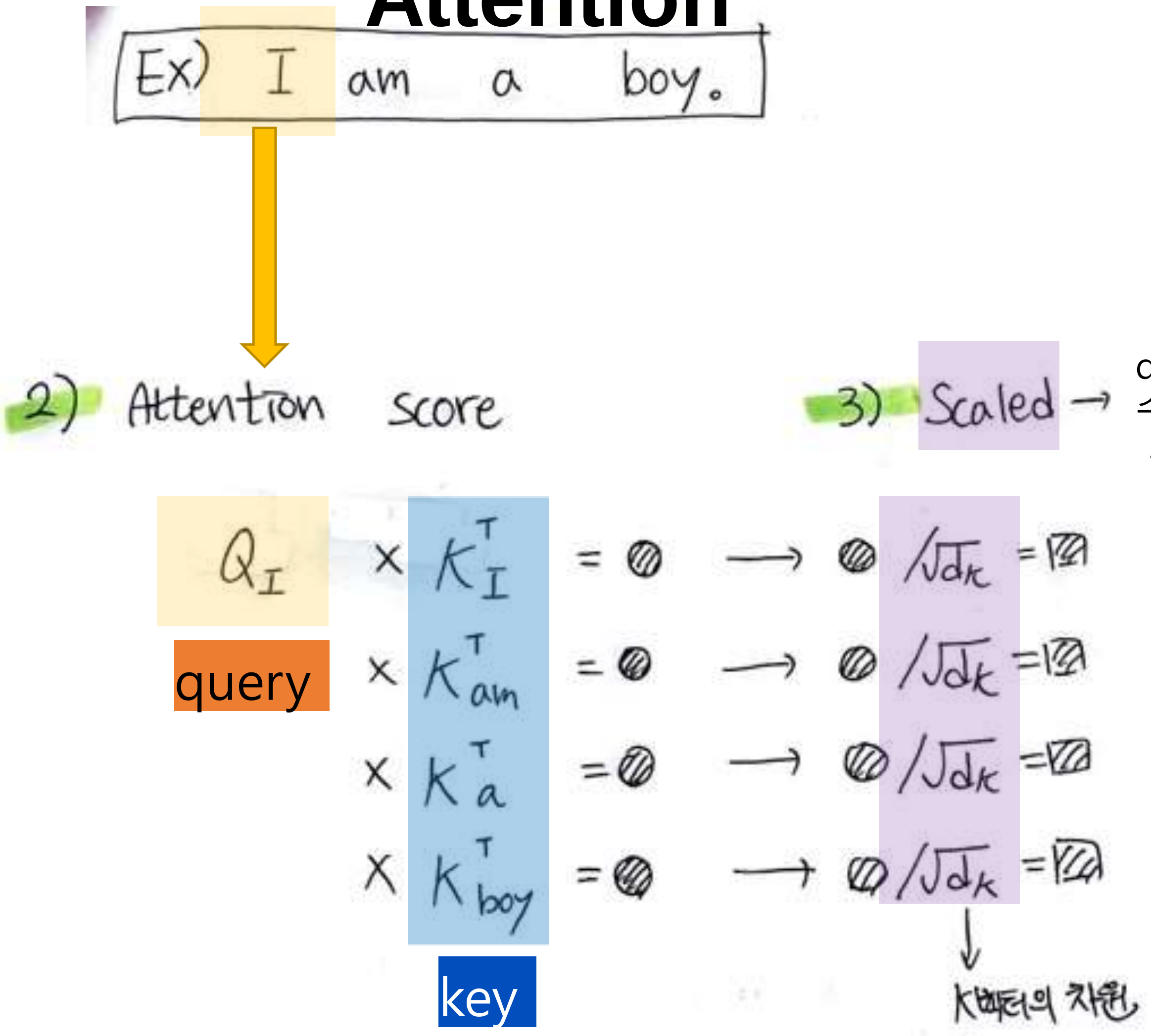
Scaled Dot-Product Attention

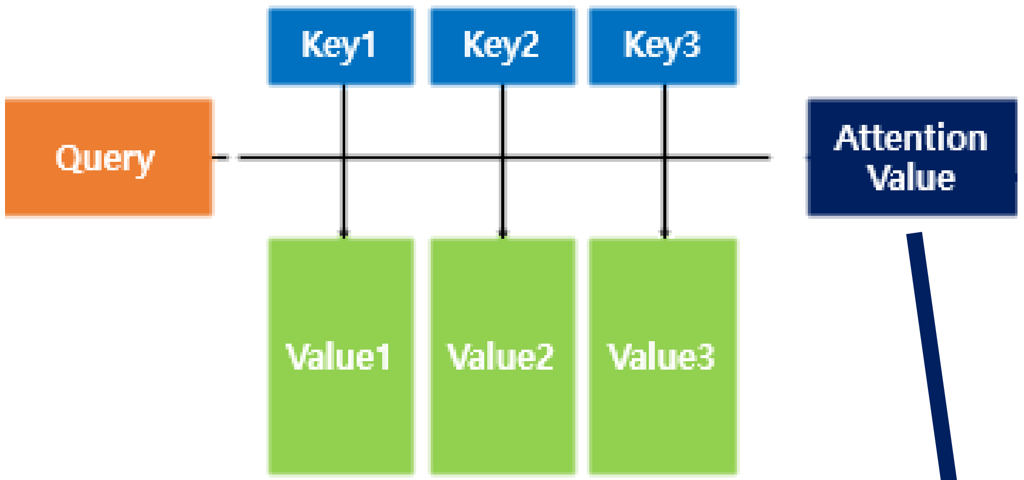
(self-attention)

Scaled Dot-Product Attention

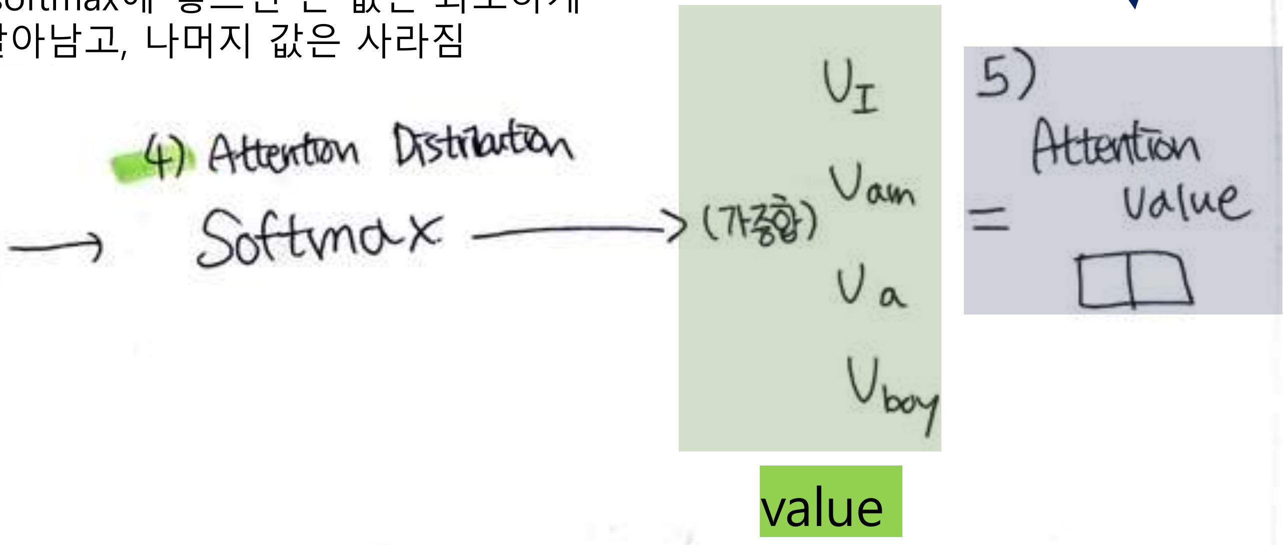


Scaled Dot-Product Attention

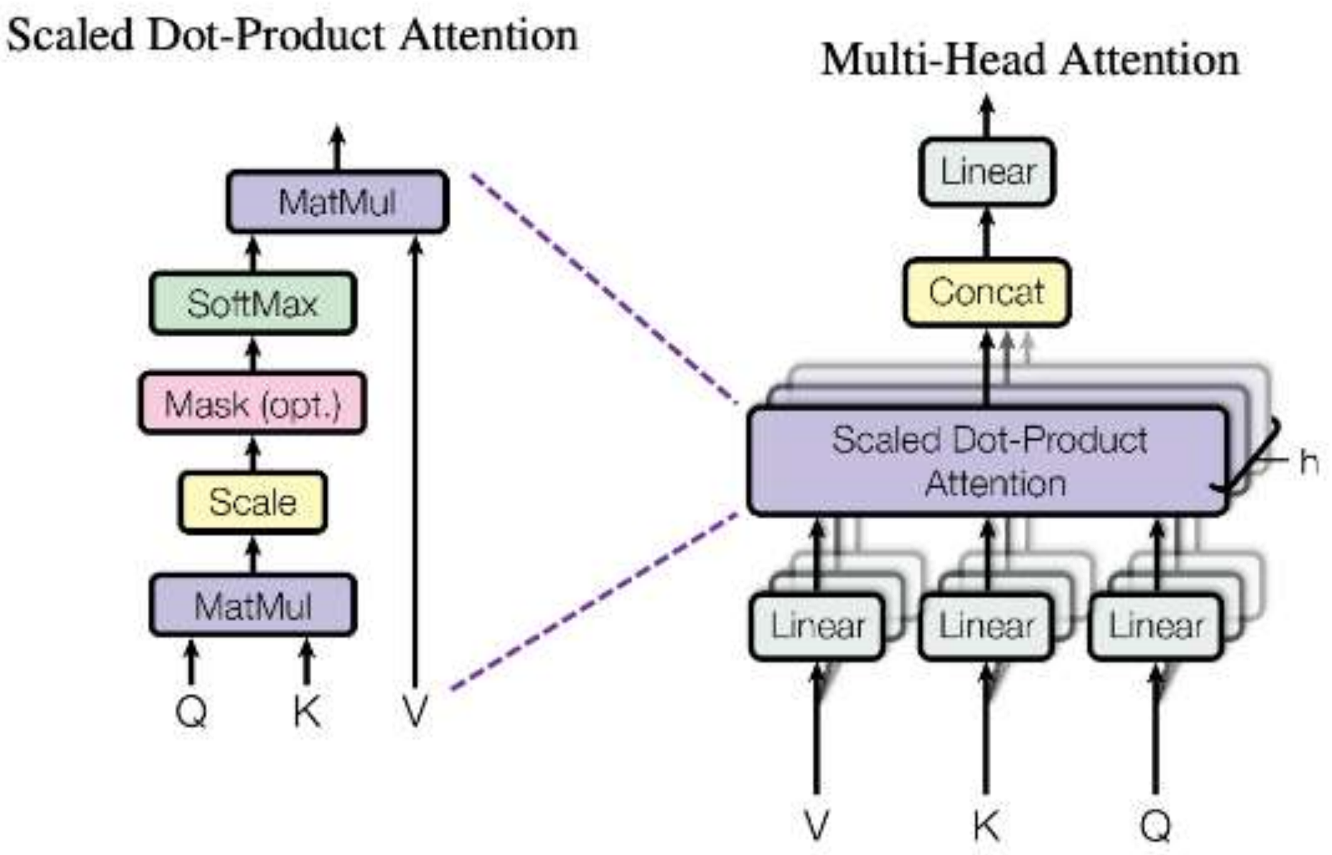




dot product 계산시 문장의 길이가 길어질 수록 결과값 커짐
-> softmax에 넣으면 큰 값은 과도하게 살아남고, 나머지 값은 사라짐



Multi-head Attention



Multi-head Attention

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$

: Self-Attention을 **병렬로 h번 학습**

Which do you like better, coffee or tea? - 문장 타입에 집중하는 어텐션

Which do you like better, coffee or tea? - 명사에 집중하는 어텐션

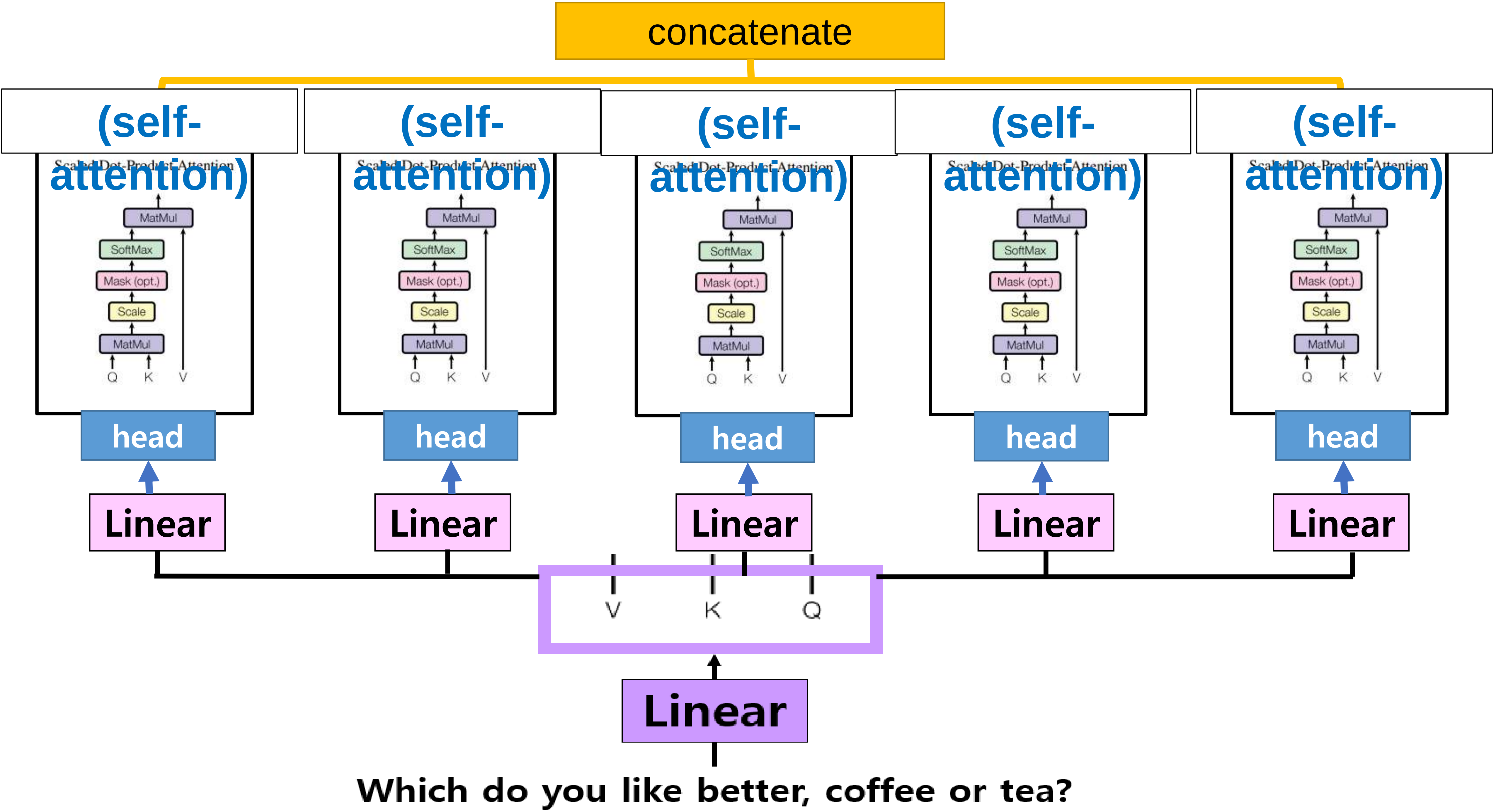
Which do you like better, coffee or tea? - 관계에 집중하는 어텐션

Which do you like better, coffee or tea? - 강조에 집중하는 어텐션

다양한 시각에서
자동으로
정보 수집 및 포착

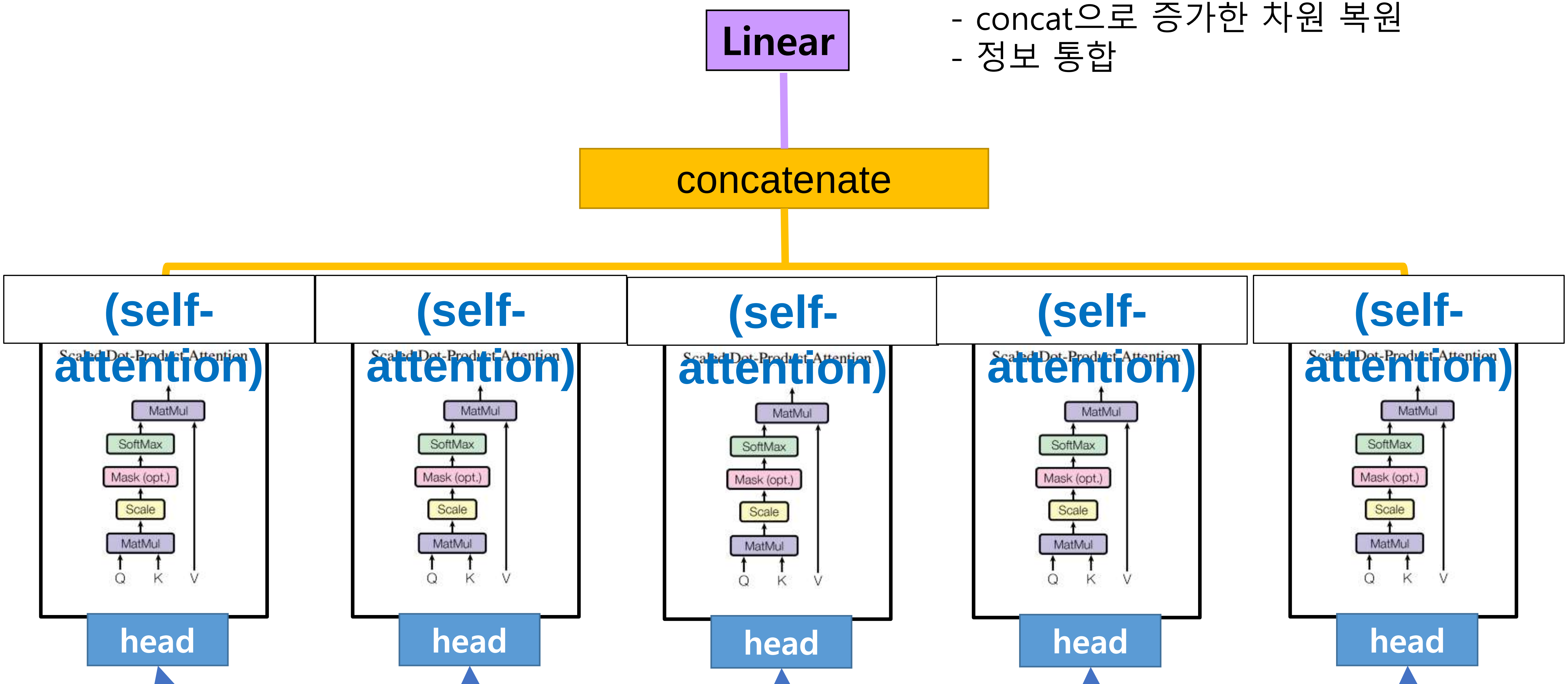
-> 표현력 풍부

복잡한 관계 파악 용이

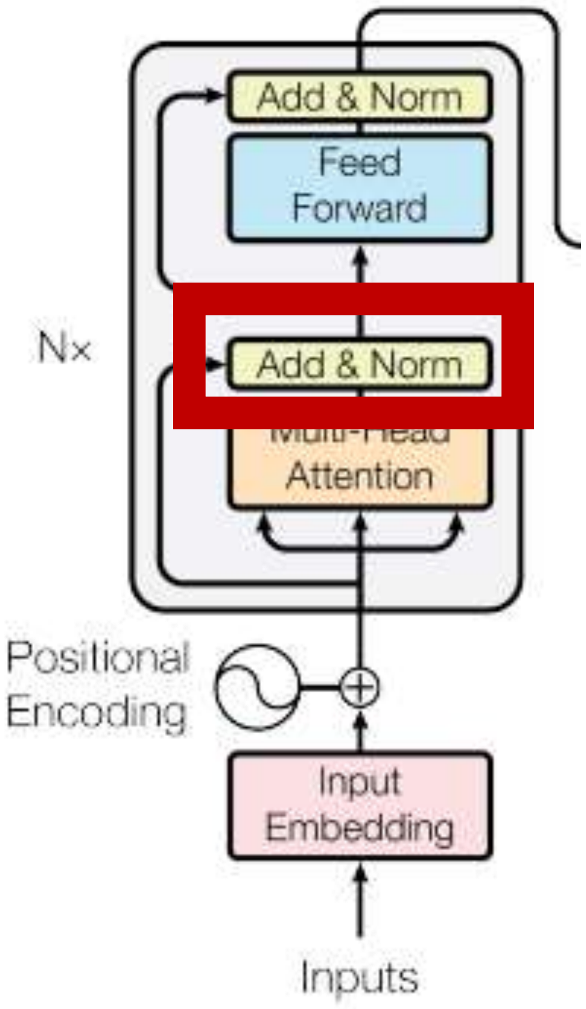


concatenate
(self-attention)
Scaled Dot-Product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
head
Linear
(self-attention)
Scaled Dot-Product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
head
Linear
(self-attention)
Scaled Dot-Product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
head
Linear
(self-attention)
Scaled Dot-Product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
head
Linear
(self-attention)
Scaled Dot-Product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
head
Linear
V
K
Q
Linear
Which do you like better, coffee or tea?

- concat으로 증가한 차원 복원
- 정보 통합



Transformer



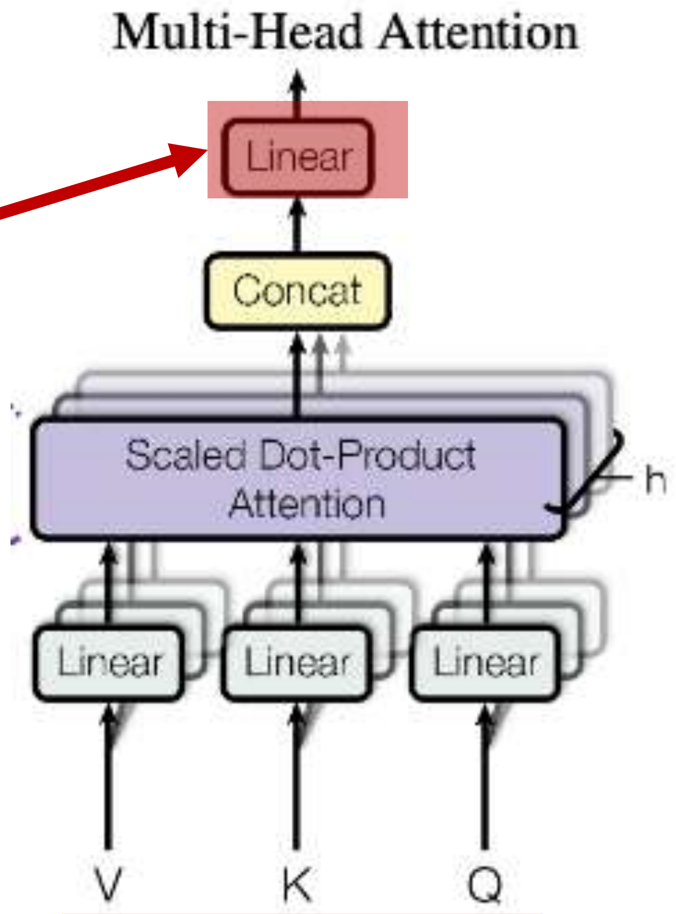
Add & Norm

Add

잔차 연결(Residual Connection)

$$H(x) = x + Multi - head\ Attention(x)$$

gradient 소실문제 완화, 안정성 증진



원본 데이터

Add & Norm

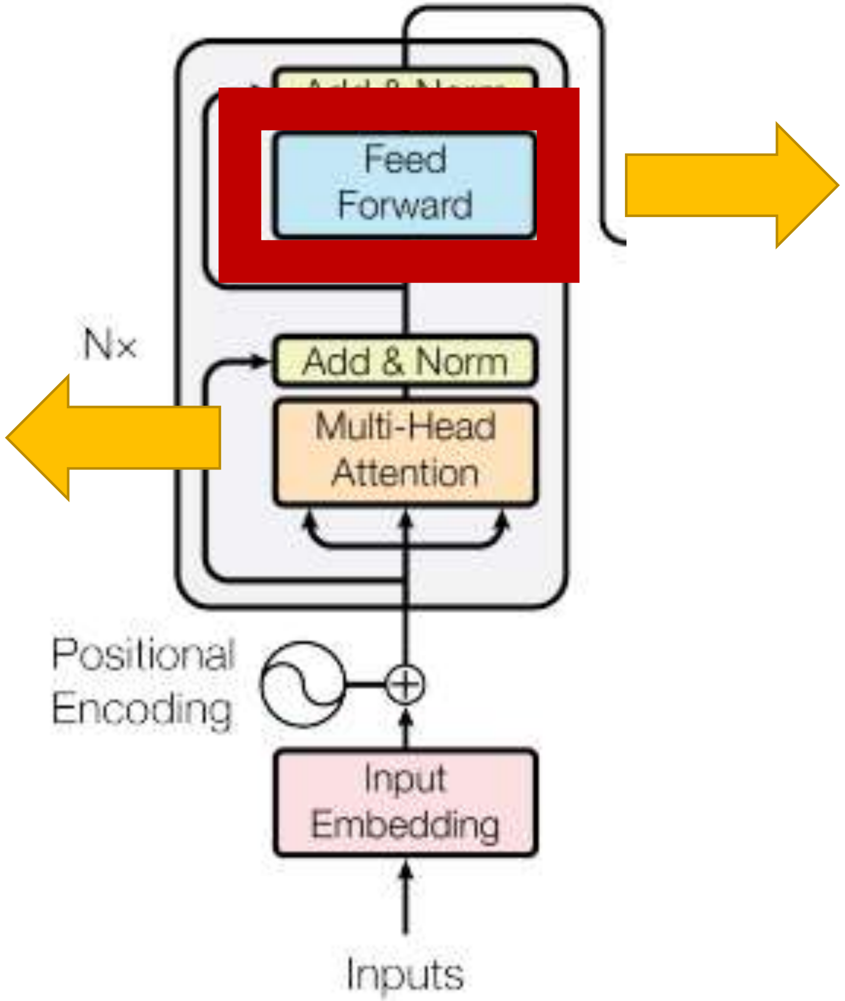
Norm

층 정규화(Layer Normalization)

: 통계적 분포 조절(안정화)

keras.
LayerNormalization()

Transformer

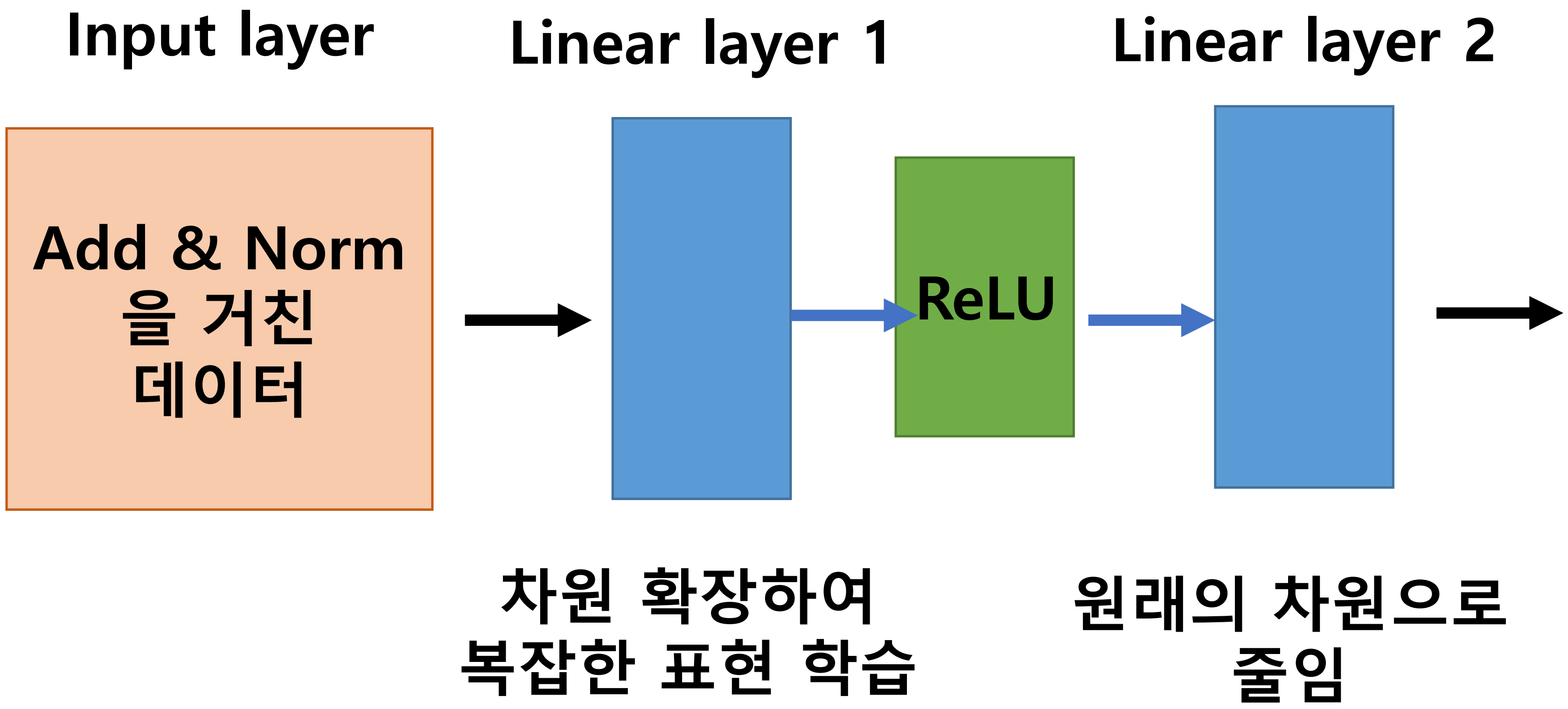


토큰의 표현을 발전시켜
더욱 정교한 표현을 얻음
(복잡한 특징이나
패턴 추가로 학습)

시퀀스 내,
토큰 사이의 관계 파악

Position-wise FFNN

: 전체 임베딩 시퀀스를 하나의 벡터로 처리하지 않고
각 임베딩을 독립적으로 처리
(병렬 처리)



Transformer