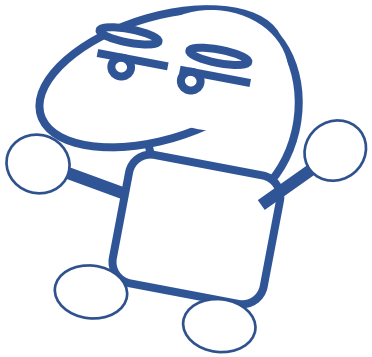
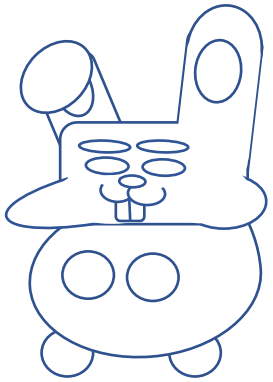
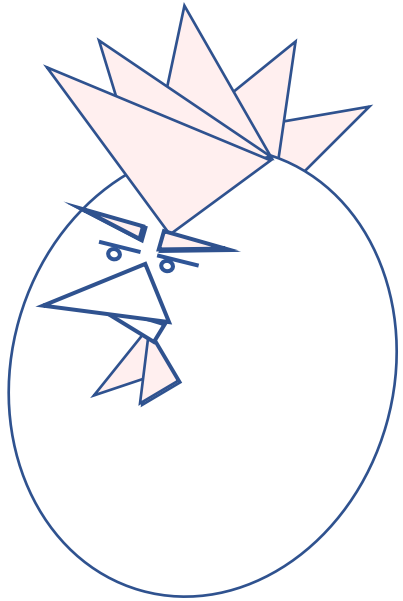
Attention is all you need

2024-02-07

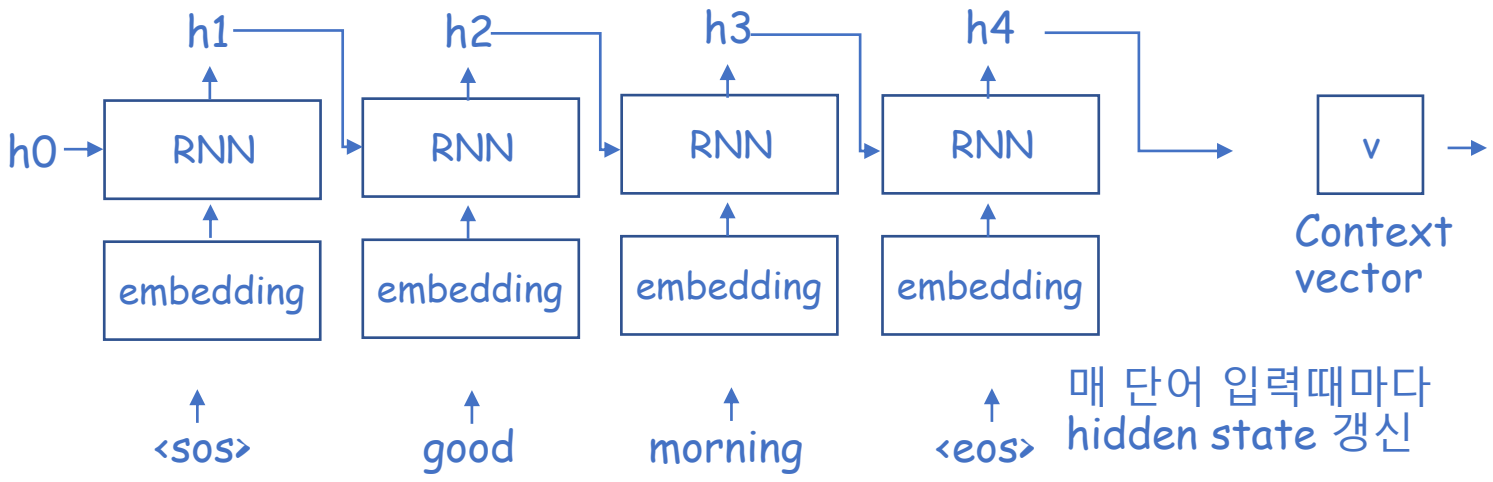
Wonjun-Jeong

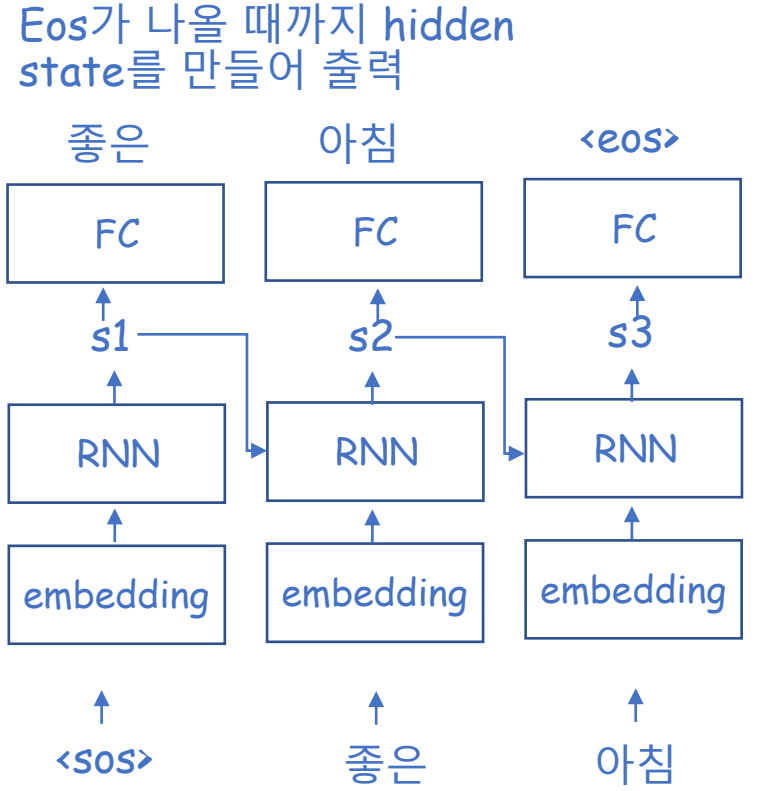






Motivation



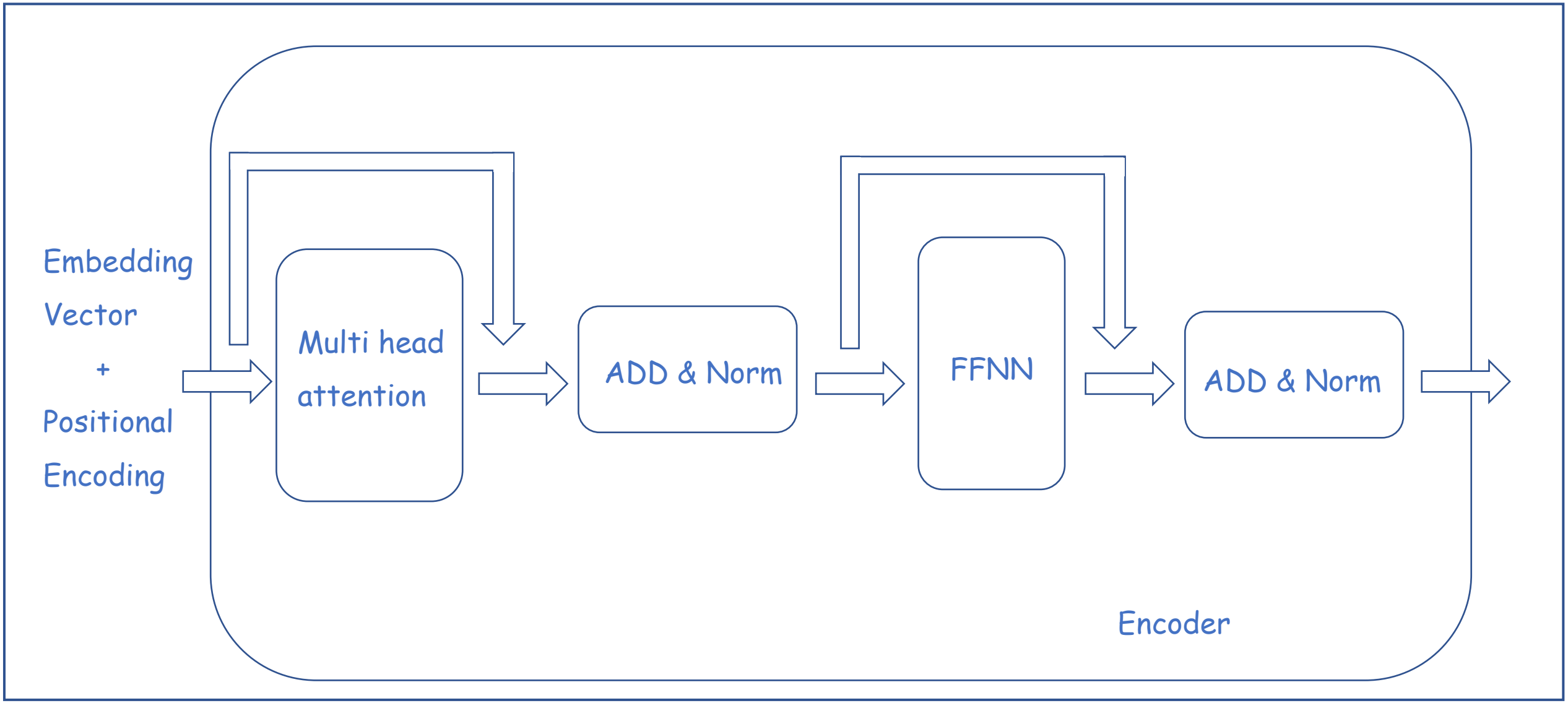


소스 문장의 길이에 상관 없이
항상 같은 크기로 context vector 생성

-> 병목 현상의 원인

Attention만을 이용하는
Transformer가 개발

Encoder


Embedding
Vector
+
Positional
Encoding
Multi head
attention
ADD & Norm
FFNN
ADD & Norm
Encoder

Positional Encoding

Transformer에서는 LSTM, RNN을 사용X

->Positional Encoding을 통해 위치 정보를 입력

같은 단어라도 입력 순서에 따라 다른 임베딩

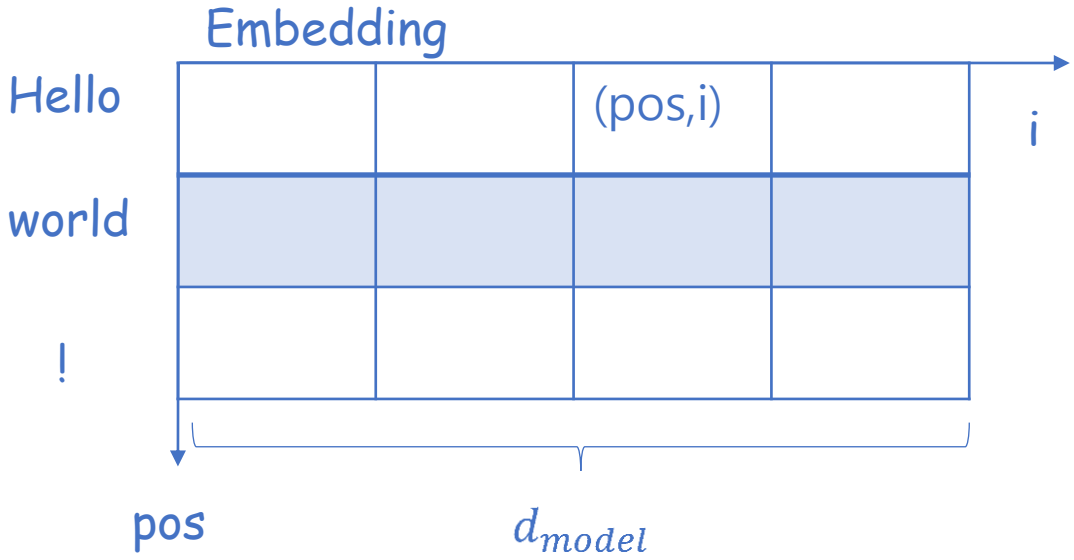
Positional Encoding은 주기함수를 활용해 정의

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Pos -> 단어의 위치

i -> 임베딩 값의 위치



+

Positional Encoding

Hello

world

!

PE(0,0)	PE(0,1)	PE(0,2)	PE(0,3)
PE(1,0)	PE(1,1)	PE(1,2)	PE(1,3)
PE(2,0)	PE(2,1)	PE(2,2)	PE(2,3)

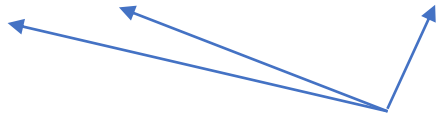
Attention

Attention -> 문맥에 따라 집중할 단어를 결정하는 방식

쿼리Q = 물어보는 질문(분석 대상이 되는 단어에 대한 가중치 벡터)
키 K = 질문의 대상(모든 단어, 각 단어와 쿼리 사이 연관성 확인)
값 V =키에 매칭된 실제 값(key의 의미를 나타내는 가중치 벡터)

Self attention = Query, key, value가 서로 같은 attention
-> 입력 문장에서 각 단어가 어떤 단어와 연관성 계산

ex) 나는 귤을 먹었는데 그것은 너무 셨다.



나는 귤을 먹었는데 **그것**은 너무 셨다

$$Attention(Q,K,V)$$

$$= softmax\left(\frac{QK^T}{d_k^{0.5}}\right)V$$

$$head_i$$

$$= Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$Multihead(Q,K,V)$$

$$= Concat(head_i, \dots, head_h)W^o$$

Attention

Attention -> 문맥에 따라 집중할 단어를 결정하는 방식

쿼리Q = 필요한 것

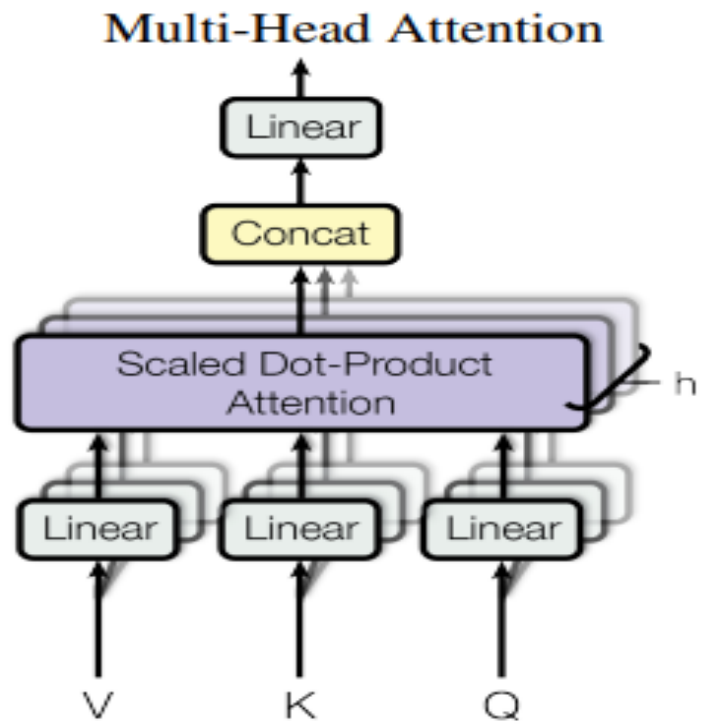
->분석 대상이 되는 단어에 대한 가중치 벡터

키 K = 질문의 대상 (유사도 계산)

->모든 단어, 각 단어와 쿼리 사이 연관성 확인

값 V =키에 매칭된 실제 값

->key의 의미를 나타내는 가중치 벡터



Self attention = Query, key, value가 서로 같은 attention

-> 입력 문장에서 각 단어가 어떤 단어와 연관성 계산

ex)나는 귤을 먹었는데 그것은 너무 셨다.

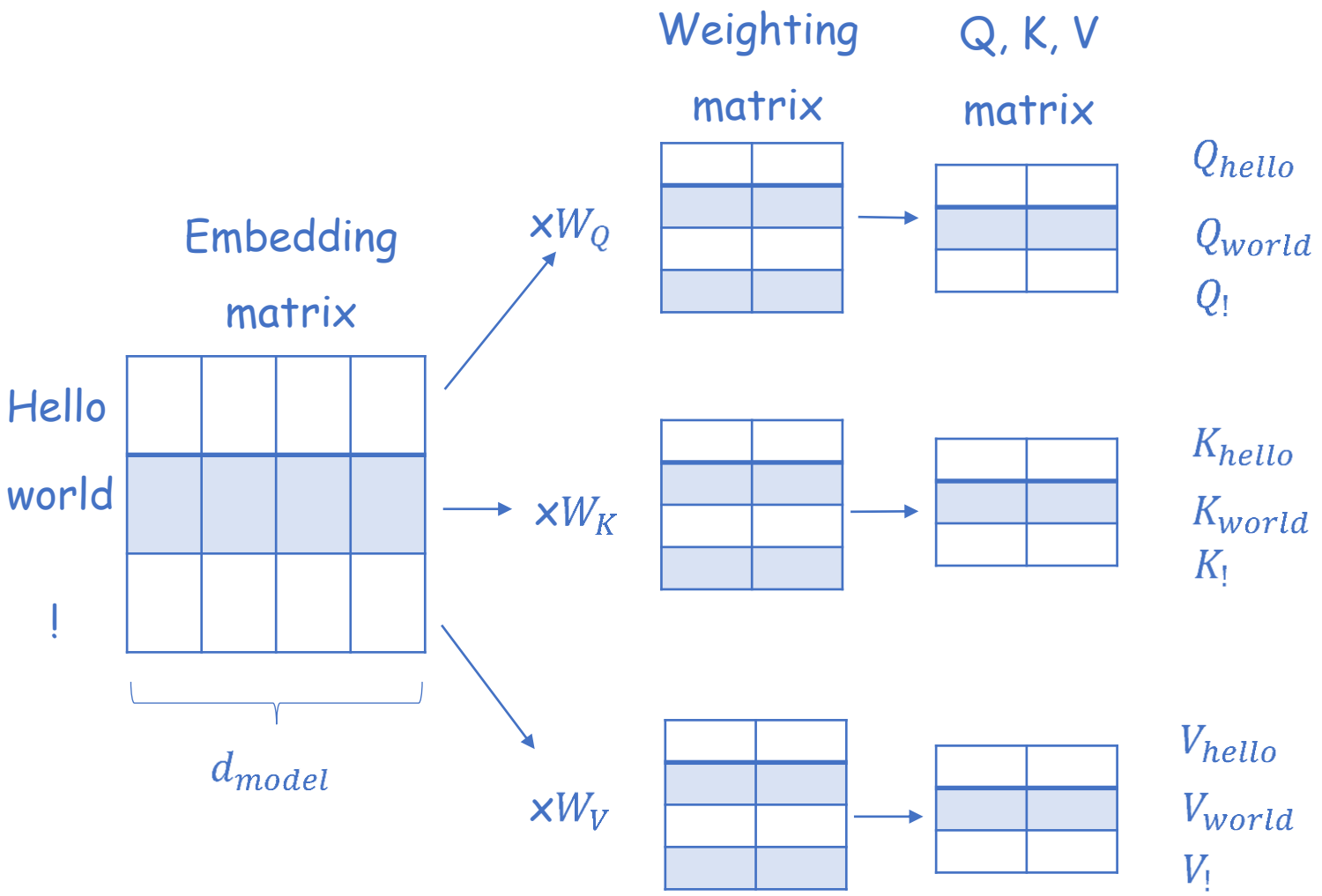
나는 귤을 먹었는데 **그것**은 너무 셨다

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{d_k^{0.5}}\right)V$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$Multihead(Q,K,V) = Concat(head_i, \dots, head_h)W^o$$

Attention



Embedding 차원 = d_{model}

가중치 행렬 = d_{model}X $\frac{d_{model}}{\#\ of\ heads}$

Q&K&V의 차원

=$d_q = d_v = d_k = \frac{d_{model}}{\#\ of\ heads}$

Q, K, V에 대한 W행렬은
임의의 값을 가진 행렬
-> training을 통해 최적화

Self Attention

$$softmax(x_i) = \frac{e^{x_i}}{\sum e^{x_j}}$$

Q matrix

37.7	...	4.2
4.498	...	68.4351
5.81	...	0.381

Q_{hello}
Q_{world}
$Q_!$

×

K matrix

K_{hello}	K_{world}	$K_!$
2.21	73.61	9.03
...	...	...
65.01	2.25	74.3

→

Hello
world
!

Hello world !

100	60	70
80	110	60
80	70	90

Attention score

→

0.97	0.007	0.023
0.022	0.97	0.018
0.21	0.06	0.73

QK^T은
얼마나 유사한지
의미

$$softmax\left(\frac{QK^T}{d_k^{0.5}}\right)$$

-> 확률분포 계산

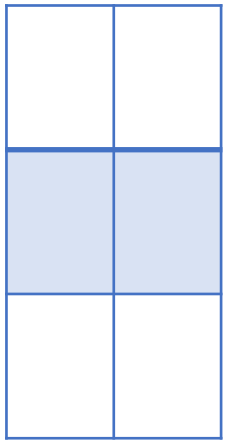
Self Attention

$softmax\left(\frac{QK^T}{d_k^{0.5}}\right)$

0.90	0.0.05	0.05
0.05	0.80	0.15
0.05	0.10	0.85

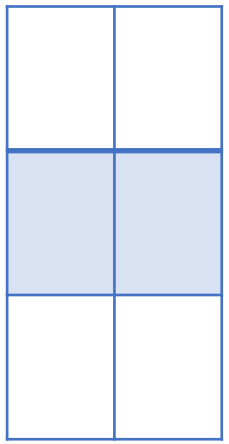
×

V matrix



Attention value

$= softmax\left(\frac{QK^T}{d_k^{0.5}}\right)V$

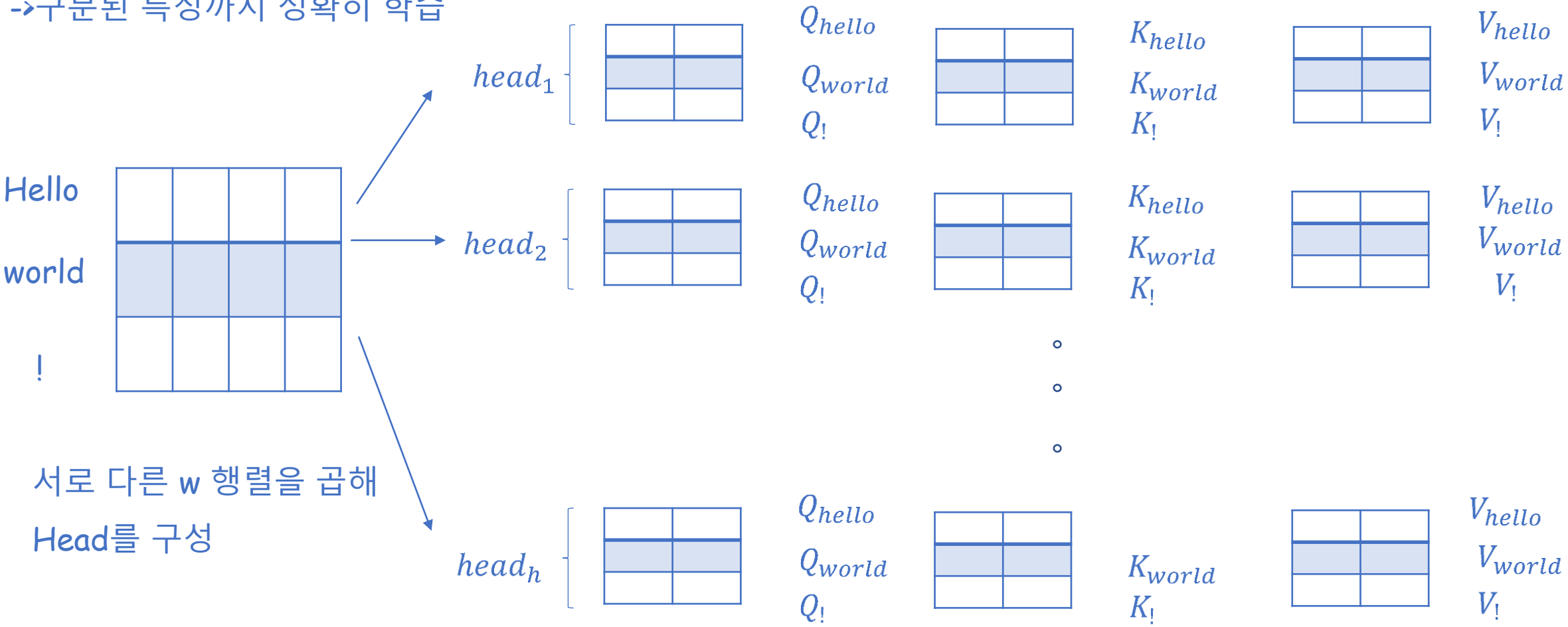


Multi-Head Attention

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

h개의 서로 다른 attention 컨셉 (명사에 집중, 동사에 집중, etc)

->구분된 특징까지 정확히 학습

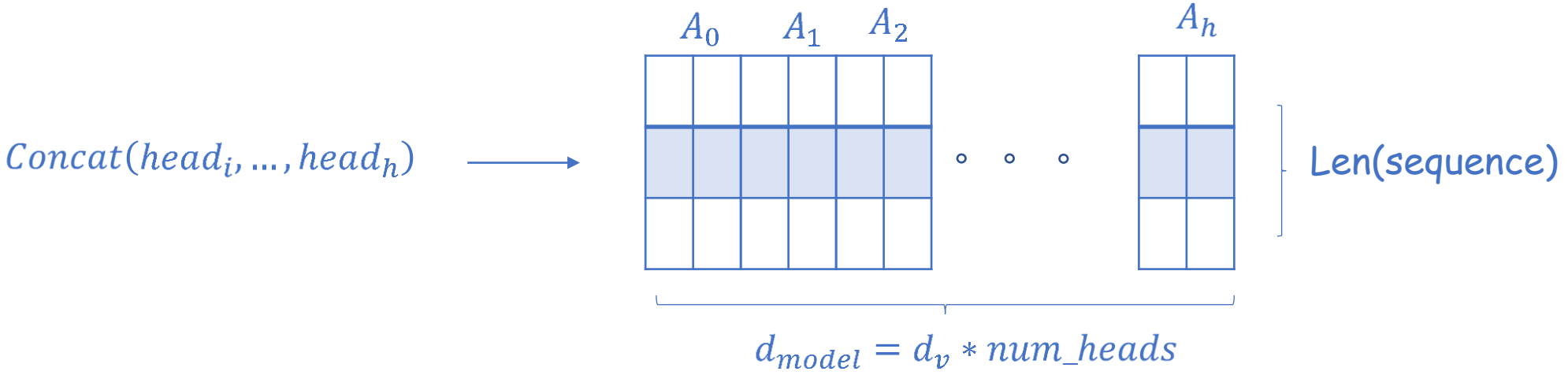


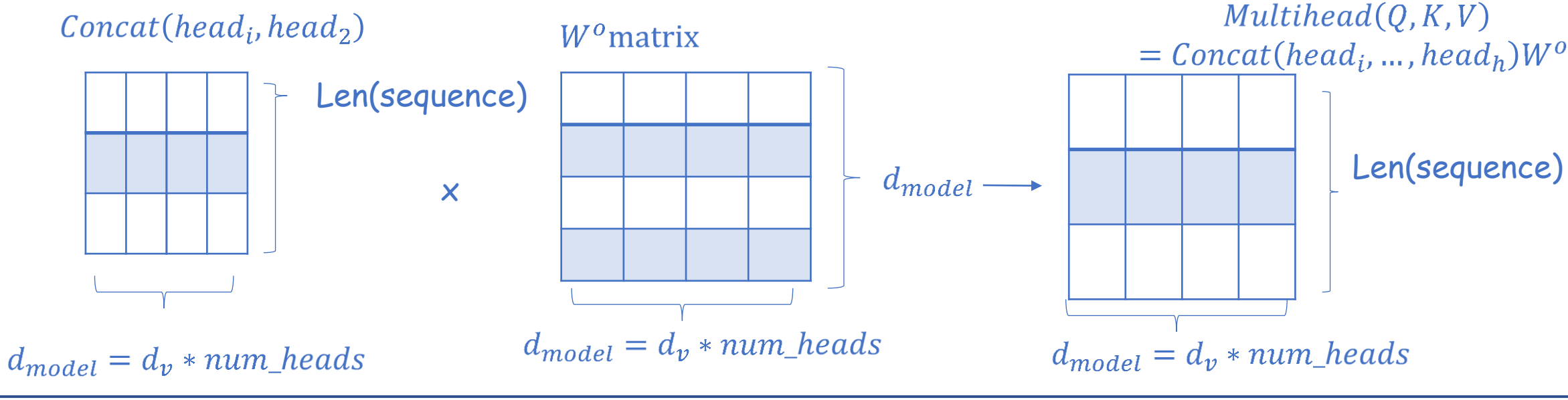
서로 다른 w 행렬을 곱해

Head를 구성

Multi-Head Attention

$$Multihead(Q,K,V) = Concat(head_i, \dots, head_h)W^o$$





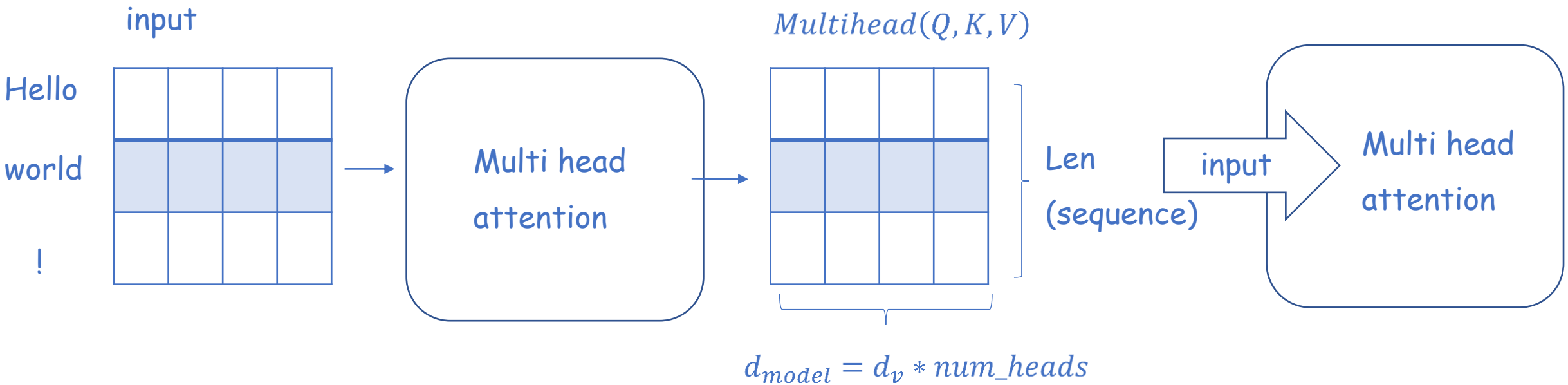
Multi-Head Attention

attention is all you need 기준

encode(docoder)의 개수= 6

Multi head attention의 출력의 크기 = 인코더의 입력의 크기

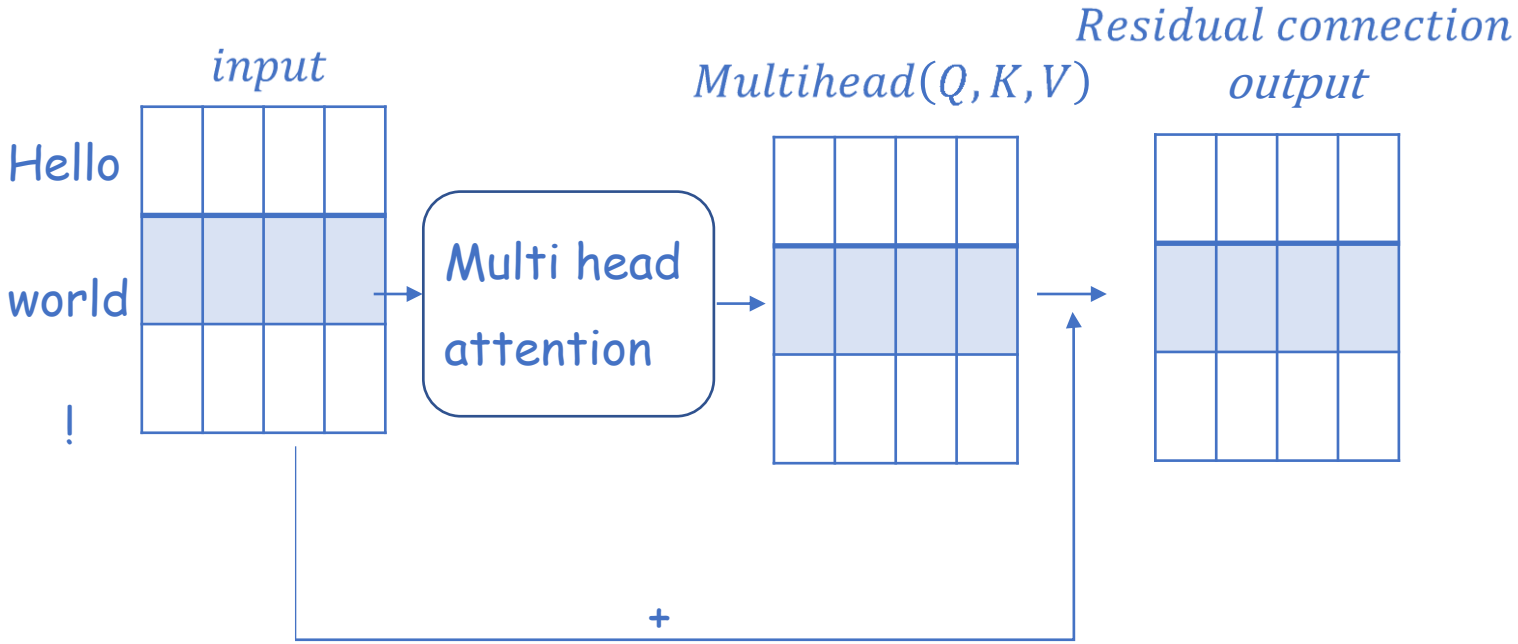
-> transformer 는 동일한 구조의 encoder(decoder)를 쌓아서 설계



Residual connection

멀티 헤드어텐션의 경우

F(x) = x + multi head Attention(x)



잔차 연결(Residual connection)

-> input+ output

F(x) = x + f(x)

X= 레이어의 입력

f(x) = 레이어의 출력

-> 결과적으로 입력의 정보를 유지한채

잔여 정보인 새로운 특성을 학습

각 data의 경로들이 독립적이라 가정

-> 각 경로들을 하나의

small network로 해석

FFNN

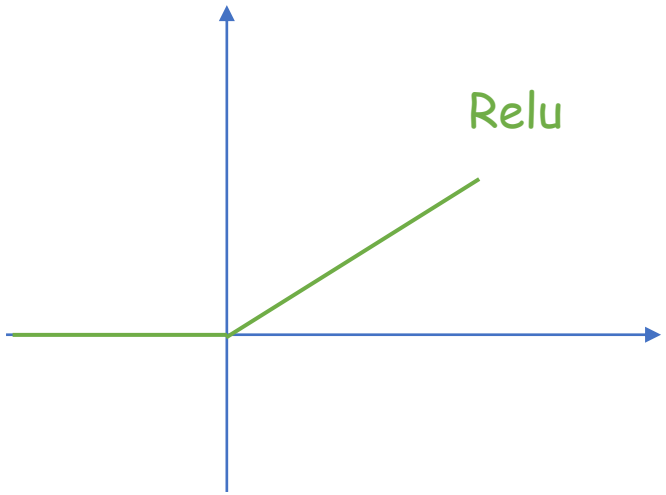
$$FFNN(x) = Max(0, xW_1 + b1)W_2 + b_2$$

x → $F_1 = xW_1 + b_1$ → activation $F_2 = \max(0, F_1)$ → $F_3 = F_2W_2 + b_2$

활성화 함수 = Relu

-> F= $Max(0, xW_1 + b1)$

->비선형성을 추가



x = Multi head의 출력

->크기가 len(seq) X d_{model} 인 행렬

d_{ff}= transformer내부 피드 포워드

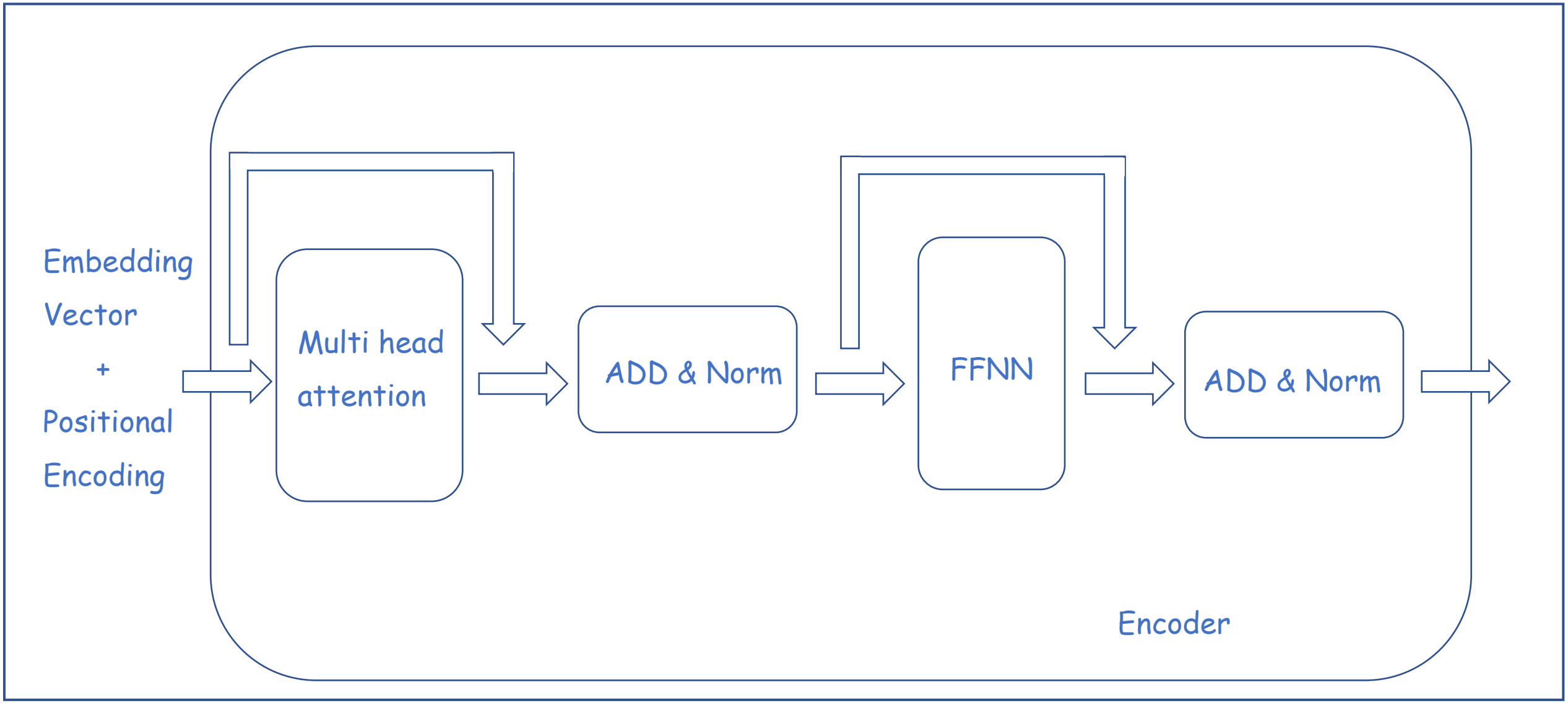
신경망의 은닉층의 크기 = 2048

가중치 W1의 크기 = d_{model}X d_{ff}

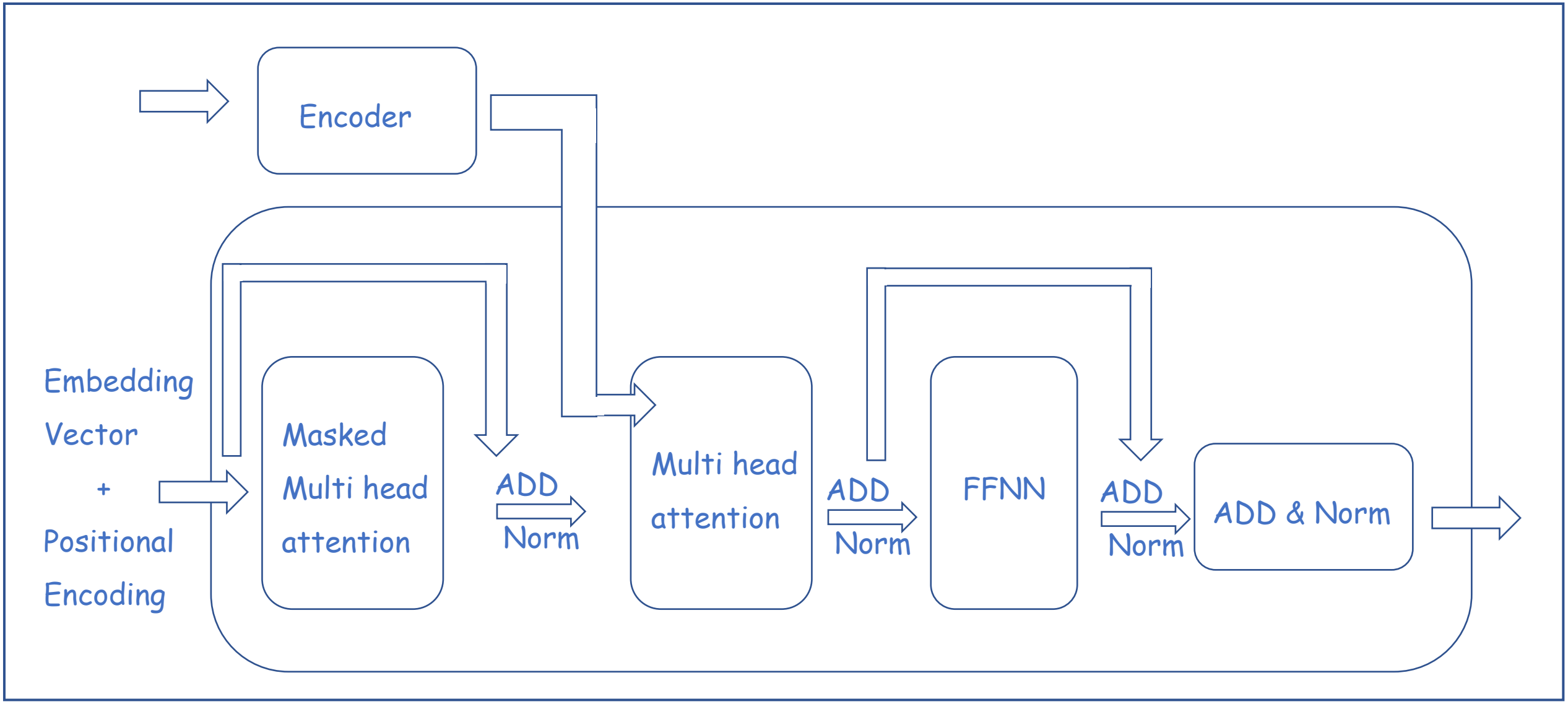
가중치 W2의 크기 = d_{ff}X d_{model}

FFNN을 거쳐도 행렬의 크기는 보존

Encoder


Embedding
Vector
+
Positional
Encoding
Multi head
attention
ADD & Norm
FFNN
ADD & Norm
Encoder

Decoder


Encoder
Embedding
Vector
+
Positional
Encoding
Masked
Multi head
attention
ADD
Norm
Multi head
attention
ADD
Norm
FFNN
ADD
Norm
ADD & Norm

Decoder

학습 과정에서 Decoder는 번역 문장 행렬을 입력으로 사용 -> multi head self attention

그 후 Decoder의 Q와 Encoder의 K,V를 이용해서 attention -> Decoder and Encoder attention

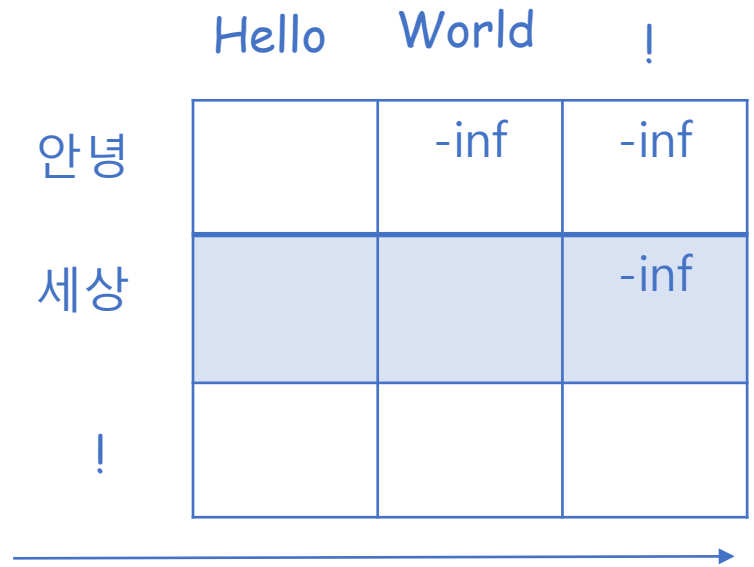
ex) 나는 귤을 먹었는데 그것은 너무 셨다.

I ate tangerines and **it** was too sour

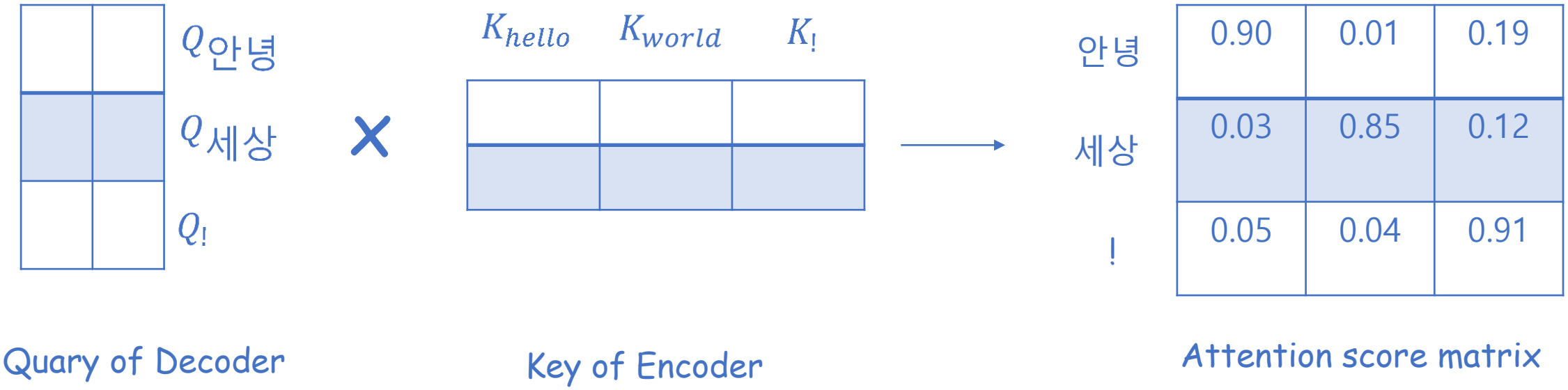
전체 문장을 입력으로 받기에 너무 미래의 단어들도 참고할 수 있음

->현재 시점의 예측에서 현재 보다 미래의 단어들을 참고 하지 못하도록 방지

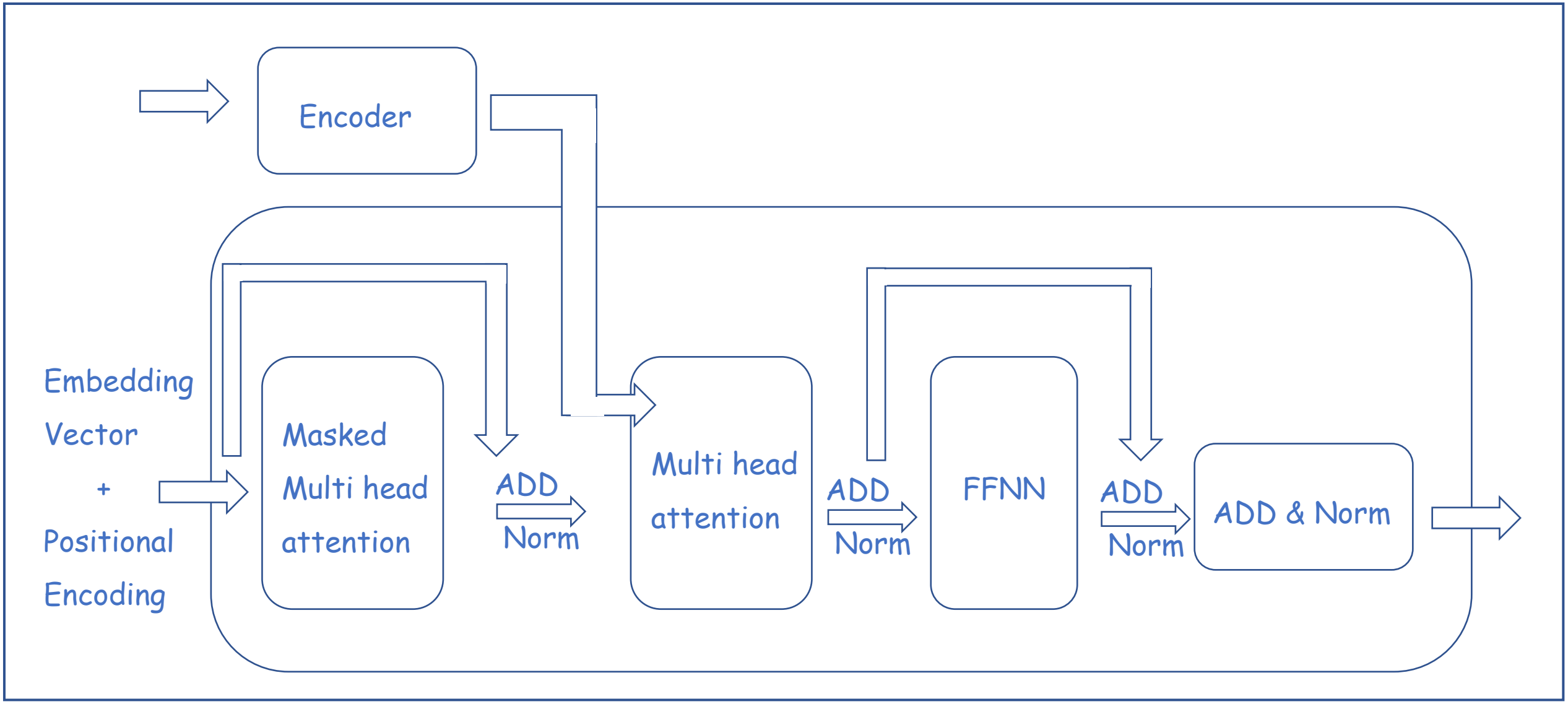
= look-ahead mask



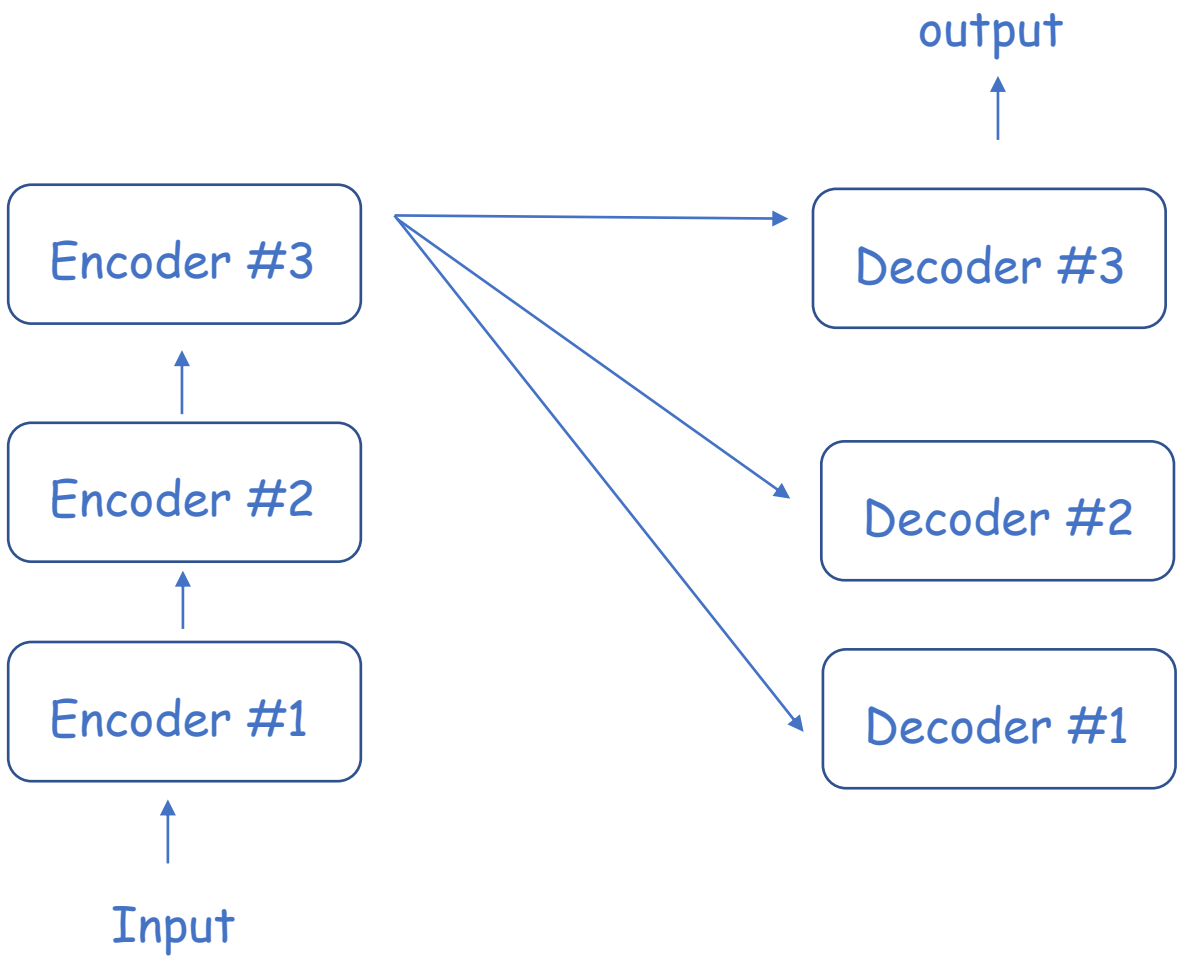
Decoder


$Q_{안녕}$
$Q_{세상}$
$Q_{!}$
X
K_{hello} K_{world} $K_{!}$
안녕
세상
!
hello world !
0.90 0.01 0.19
0.03 0.85 0.12
0.05 0.04 0.91
Quary of Decoder
Key of Encoder
Attention score matrix

Decoder


Encoder
Embedding
Vector
+
Positional
Encoding
Masked
Multi head
attention
ADD
Norm
Multi head
attention
ADD
Norm
FFNN
ADD
Norm
ADD & Norm

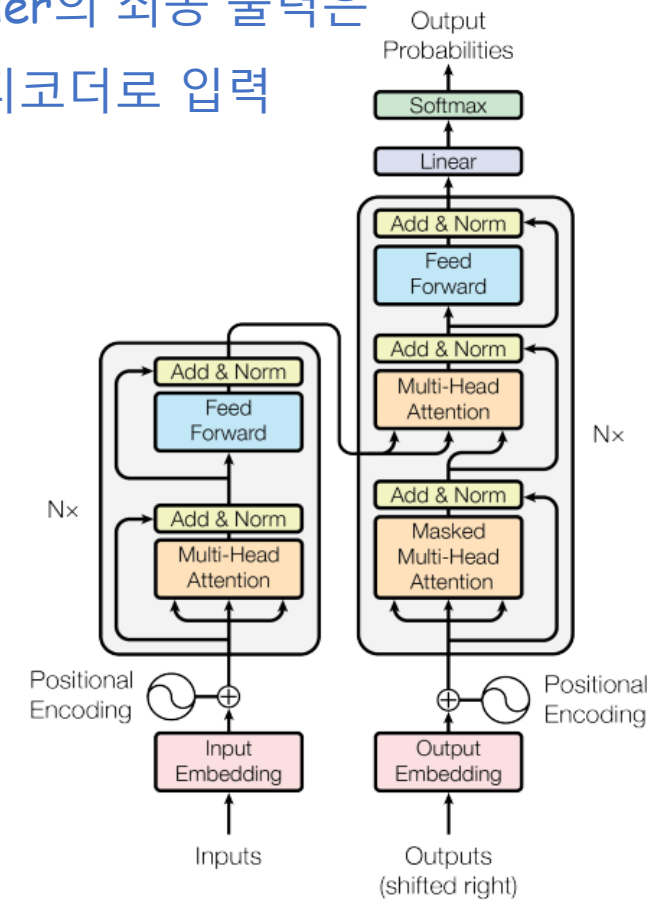
Transformer



Encoder를 쌓아서 모델을 구성

->Encoder의 최종 출력은

모든 디코더로 입력



(i 번째 encoder의 Q,K,V는

이전 encoder의 출력으로 부터 파생)