

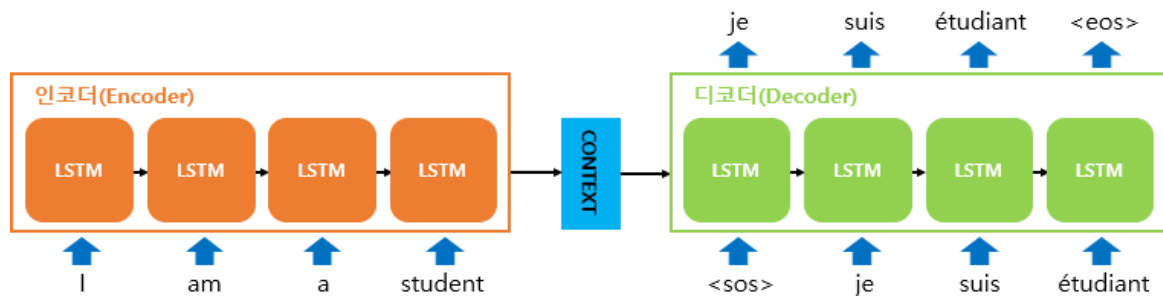
# Attention is All you need

Vaswani, Ashish, et al. "Attention is all you need."  
*Advances in neural information processing systems* 30 (2017).

## 단일 Attention mechanism을 사용한 Transformer 모델을 제안

- 1) 품질이 우수, 병렬성이 뛰어남, 훈련 소요시간 줄어듦
- 2) English-to-German, English-to-French 번역 작업에서 향상된 성능을 보임.

## Recurrent neural network의 병렬처리 문제



Attention mechanism 제시

long sequence -> 정보손실↑

## Model architecture

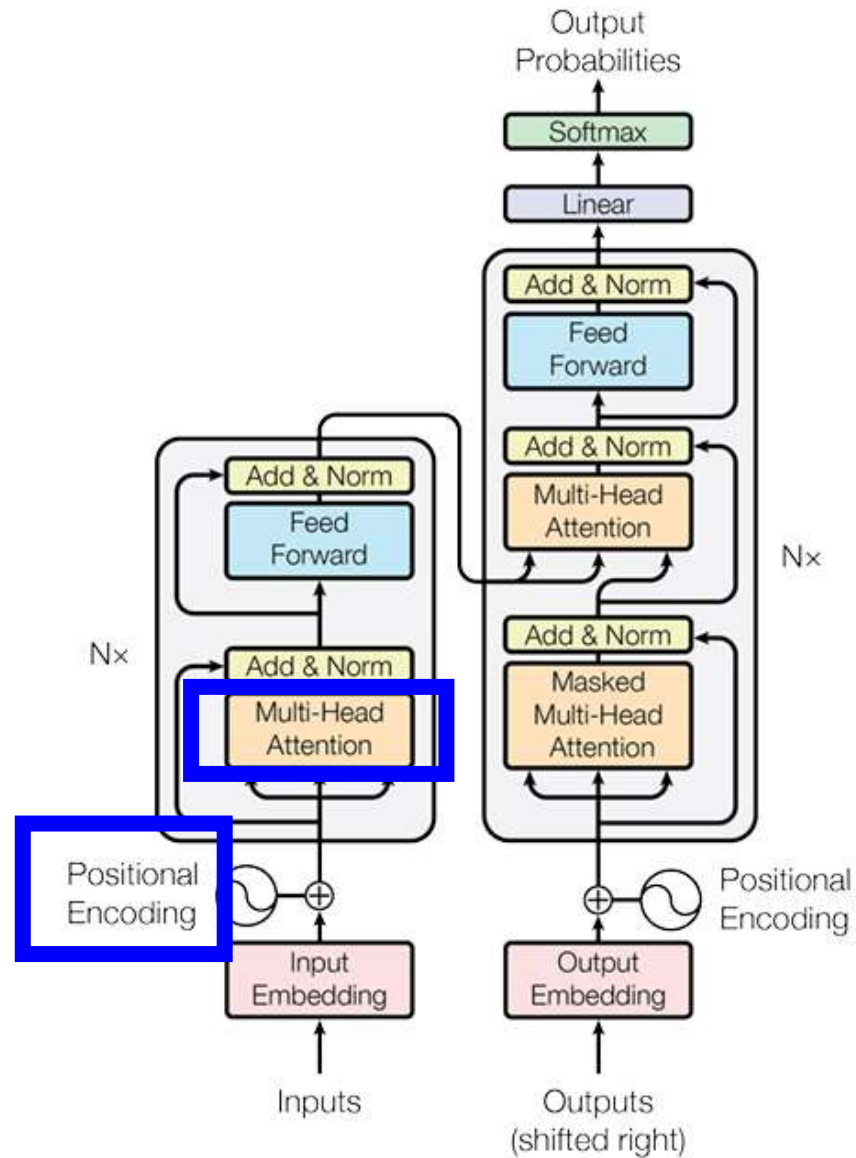
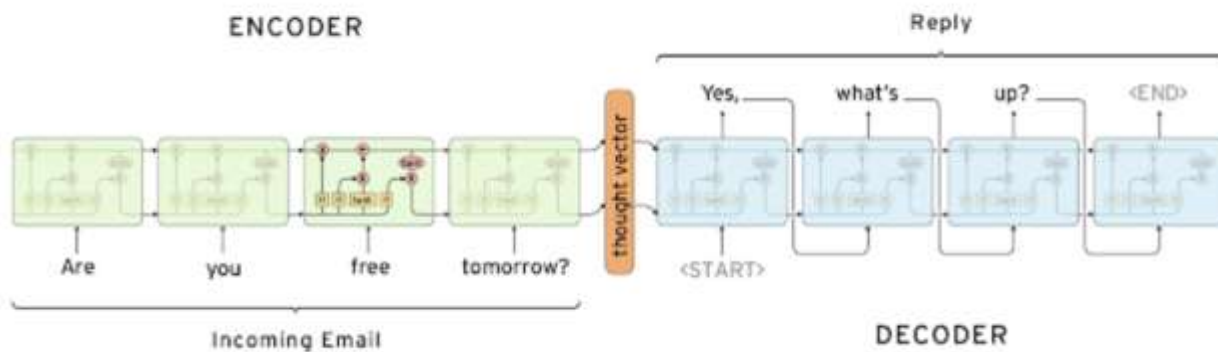


Figure 1: The Transformer - model architecture.

# 1. Positional Encoding

## seq2seq

순차적 처리

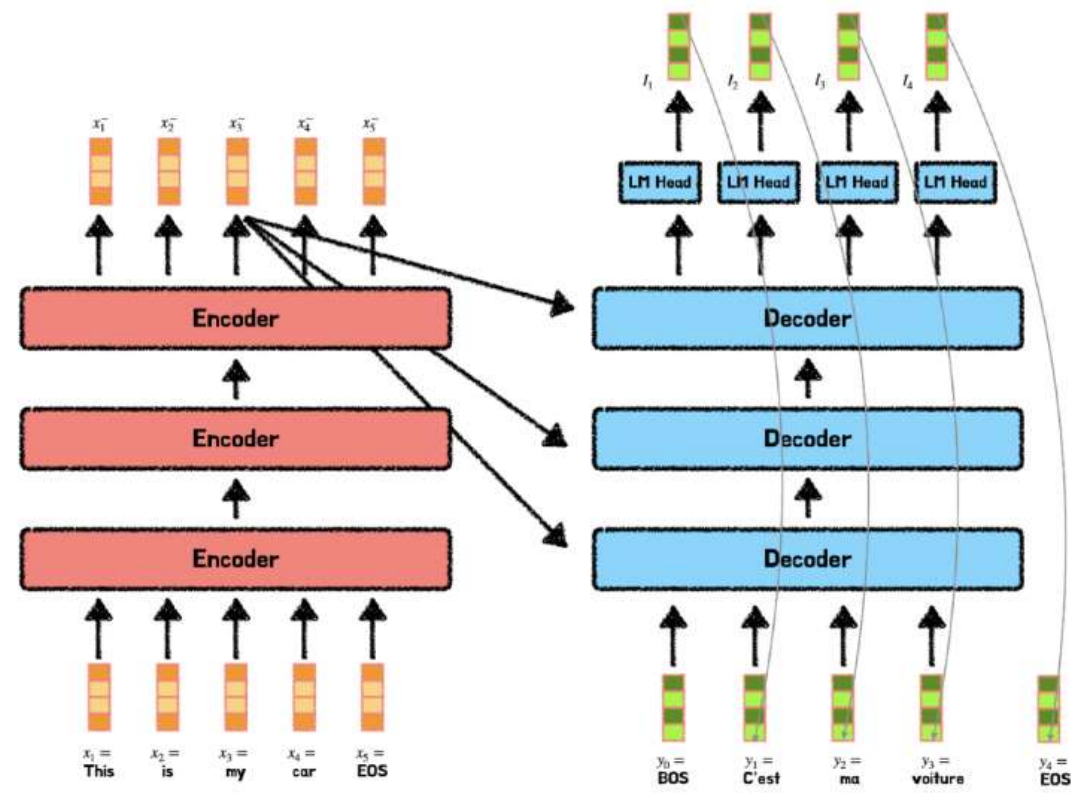


Attention만을 사용하여  
인코더와 디코더를 연결시켜보자.

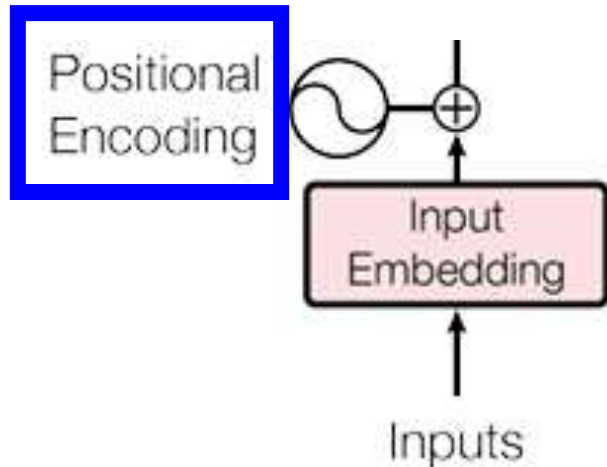
- layer 수를 줄여 연산량을 줄이고
- gradient vanishing 문제를 해결하자

## Transformer

병렬 처리



# Positional Encoding



"모델이 상대적 위치에 따라 쉽게 학습할 수 있도록 다음과 같은 함수를 제시한다."

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

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

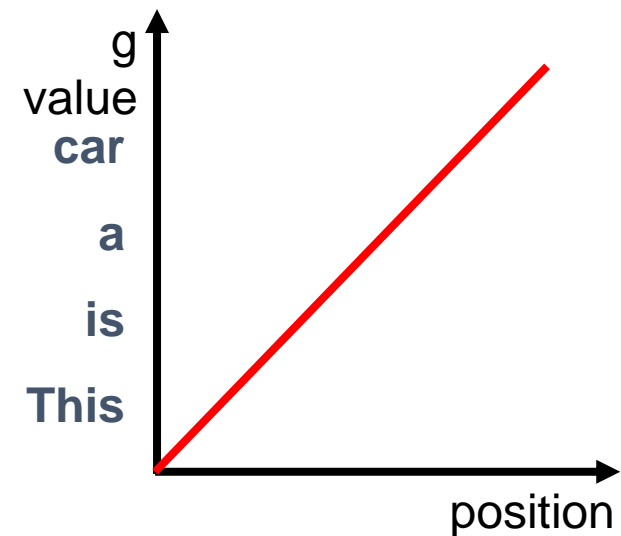
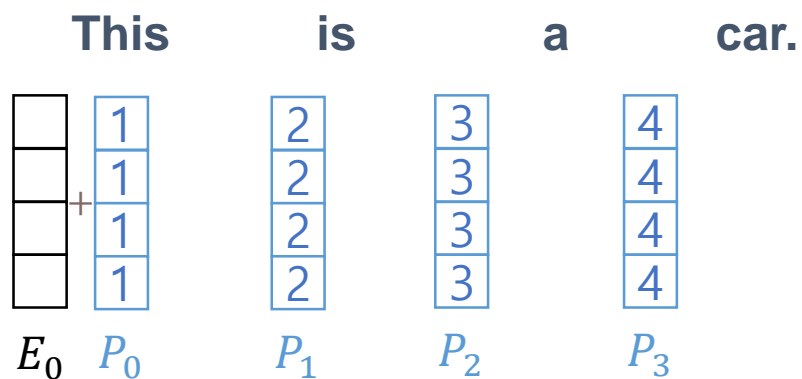
$pos$  : 단어의 위치  
 $i$  : 차원 인덱스  
 $d$  : 인코딩 벡터의 차원

## Positional Encoding

규칙 1. 모든 위치 값은 시퀀스 길이나 input 시퀀스와 관계없이 동일한 식별자를 가져야 한다.  
(시퀀스 내용이나 길이가 변경되더라도, 위치 임베딩은 동일하게 유지)

규칙 2. 모든 위치 벡터 값은 너무 크면 안된다.

제안 1) 시퀀스 크기에 비례해서 일정하게 커지는 정수값 부과

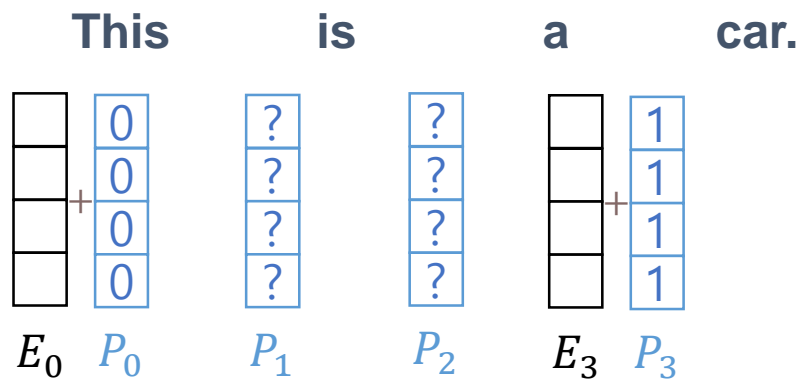


규칙 2 위반 -> 단어의 의미 훼손

## Positional Encoding

- 규칙 1. 모든 위치 값은 시퀀스 길이나 input 시퀀스와 관계없이 동일한 식별자를 가져야 한다.  
(시퀀스 내용이나 길이가 변경되더라도, 위치 임베딩은 동일하게 유지)
- 규칙 2. 모든 위치 벡터 값은 너무 크면 안된다.

제안 2) 0~1 사이의 값으로 정규화 (첫 번째 토큰:0, 마지막 토큰:1, 나머지:1/단어 수)



규칙 1 위배



# Positional Encoding

- 규칙 1. 모든 위치 값은 시퀀스 길이나 input 시퀀스와 관계없이 동일한 식별자를 가져야 한다.  
(시퀀스 내용이나 길이가 변경되더라도, 위치 임베딩은 동일하게 유지)
- 규칙 2. 모든 위치 벡터 값은 너무 크면 안된다.

제안 3) 상대적 위치에 따른 위치 인코딩 정보

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

**sin / cos ?**

-1~1사이의 값을 갖는 주기함수

(cf. sigmoid 사용?

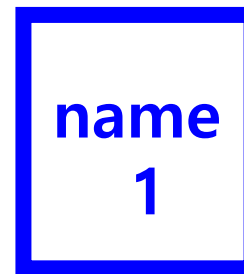
긴 문장 시퀀스가 주어졌을 시,  
위치 벡터 값들의 차이가 없음)

\*혼용해서 사용하는 것이 보다 풍부한 학습

## Positional Encoding

$pos$  : 단어의 위치  
 $i$  : 차원 인덱스  
 $d$  : 인코딩 벡터의 차원

문장 : My **name** is Hayeong  
 $pos$  : 0 1 2 3



$pos = 1$

$i = 0, d = 4$

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

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

$P_1$

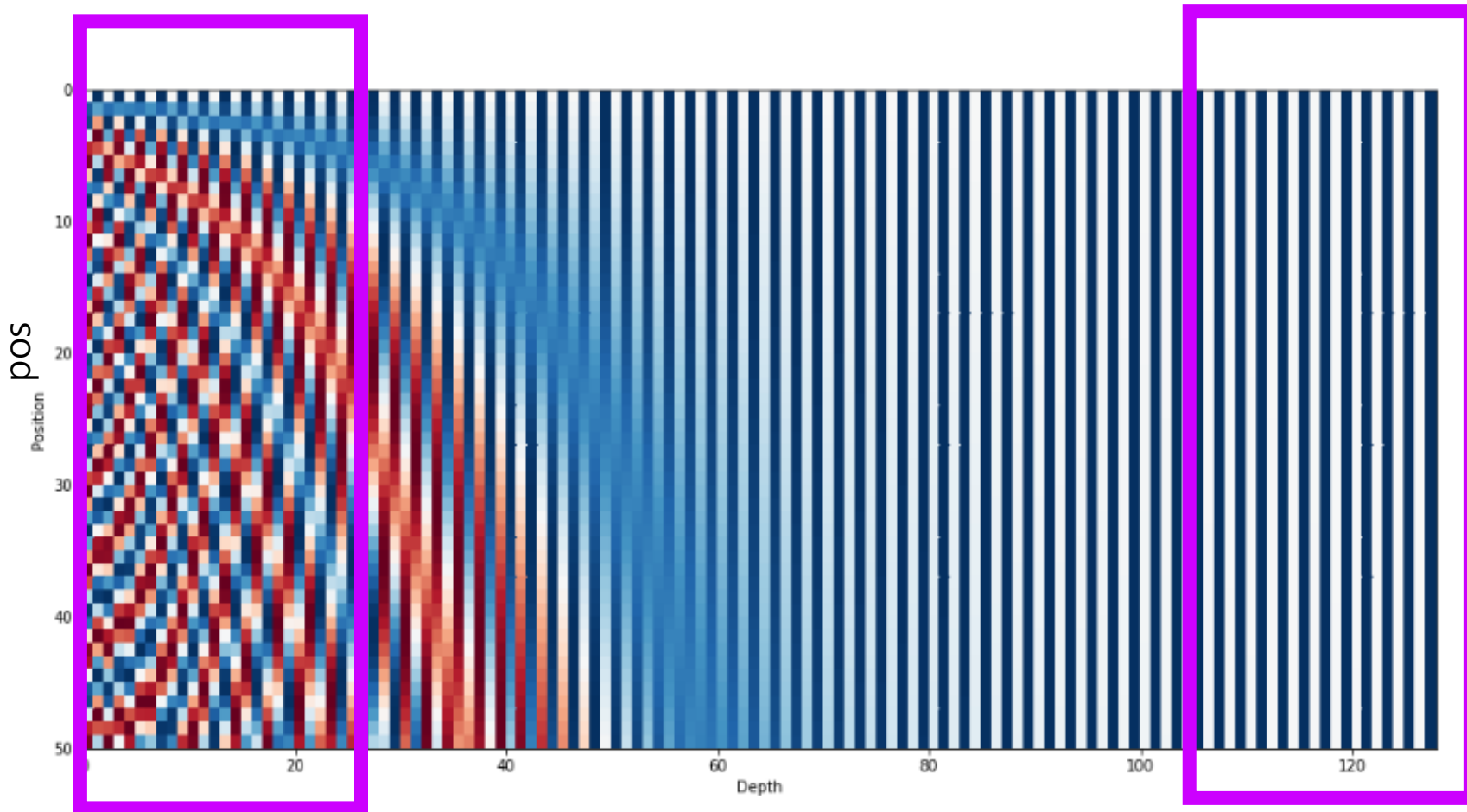
0.8
0.5
0.01
1.0

$i = 0$	$PE_{(1,0)} = \sin(1/10000^{0/4})$	$\sin(1) \approx 0.8$
	$PE_{(1,1)} = \cos(1/10000^{0/4})$	$\cos(1) \approx 0.5$
$i = 1$	$PE_{(1,2)} = \sin(1/10000^{2/4})$	$\sin(1/100) \approx 0.01$
	$PE_{(1,3)} = \cos(1/10000^{2/4})$	$\cos(1/100) \approx 1.0$

# 1. Positional Encoding

128차원 임베딩 벡터에 대하여 위치 인코딩을 통한  
< 문장의 시각적 표현 >

**[초기 차원]**  
낮은 주파수  
긴 주기  
  
전체 문맥 등  
넓은 범위의  
의미적 정보 파악



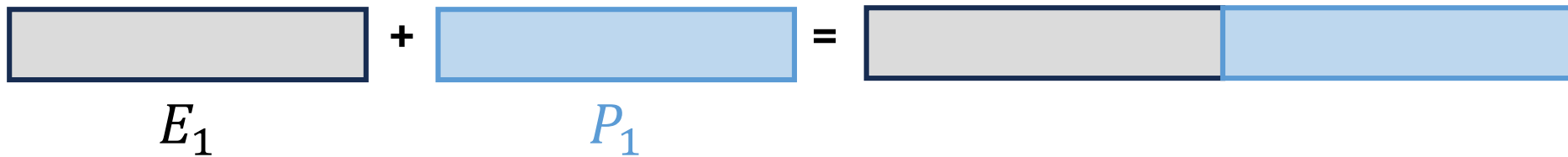
**[후기 차원]**  
높은 주파수  
짧은 주기  
  
문법적 특징 등  
세밀한 정보 파악

벡터 차원(i)

## Summation vs Concat

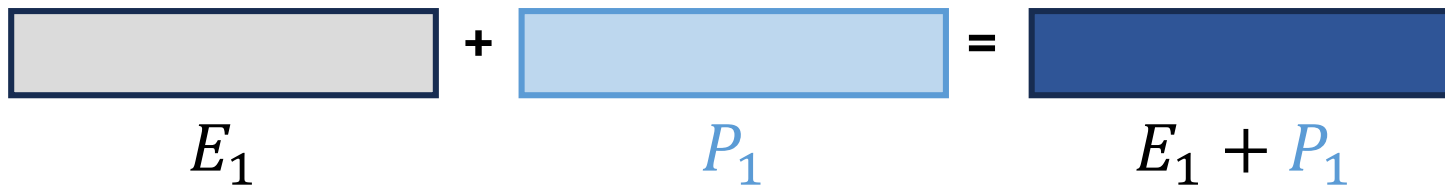
My **name** is Hayeong.

### Concat

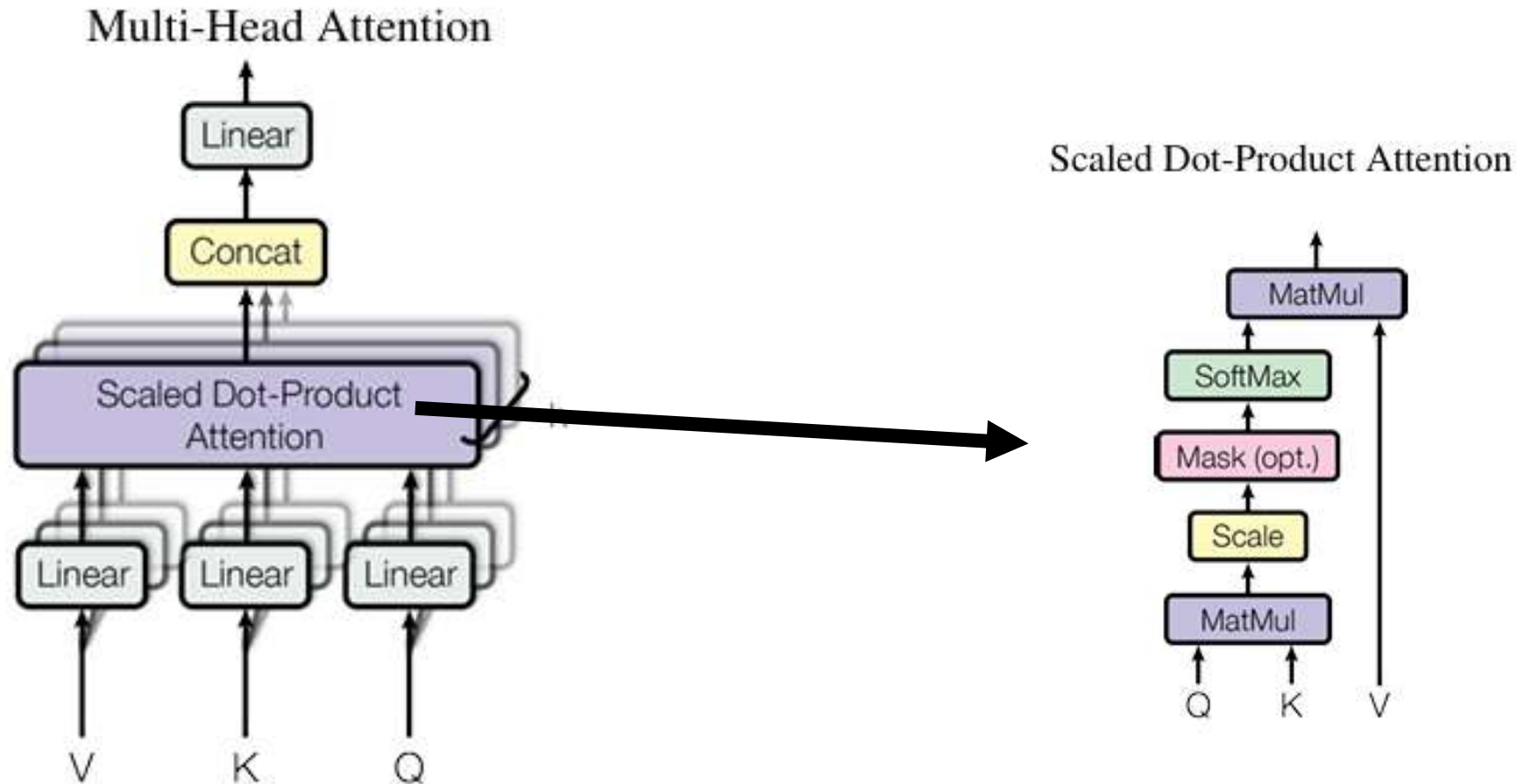


비용문제

### Summation

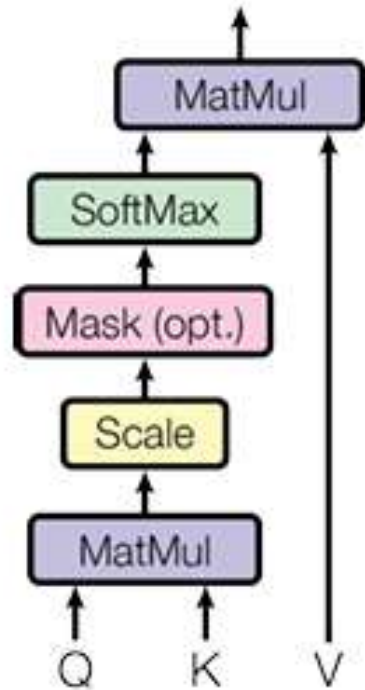


# Multi-Head Attention



# Scaled Dot-Product Attention

Scaled Dot-Product Attention



## Input sequence

→ 임베딩 행렬로 변환 + PE

→  $\square \times W^Q, \square \times W^K, \square \times W^V,$

→ **Q, K, V**

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

Q : 모든 입력 벡터

K : 모든 입력 벡터

V : 모든 입력 벡터

Q-K(유사도 파악)

V(가중치 합계 계산/QK의 유사한만큼 V값)

### Multi-Head Attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where  $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

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

Self-Attention을 병렬로  $h$ 번 학습  
다양한 관점에서 정보 수집 위함

Add

## - Residual Connection

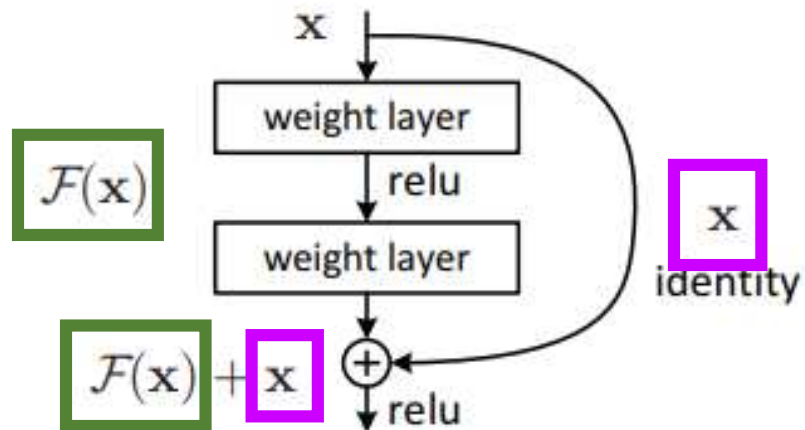


Figure 2. Residual learning: a building block.

$$F(x) + x$$

“shortcut connections” : skipping one or more layers

- 신경망의 깊이가 깊어질 때 발생할 수 있는 성능 저하 문제 완화

크레디트 스치 모델 아치

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	21.43	5.71

Error rates on ImageNet validation.



Add

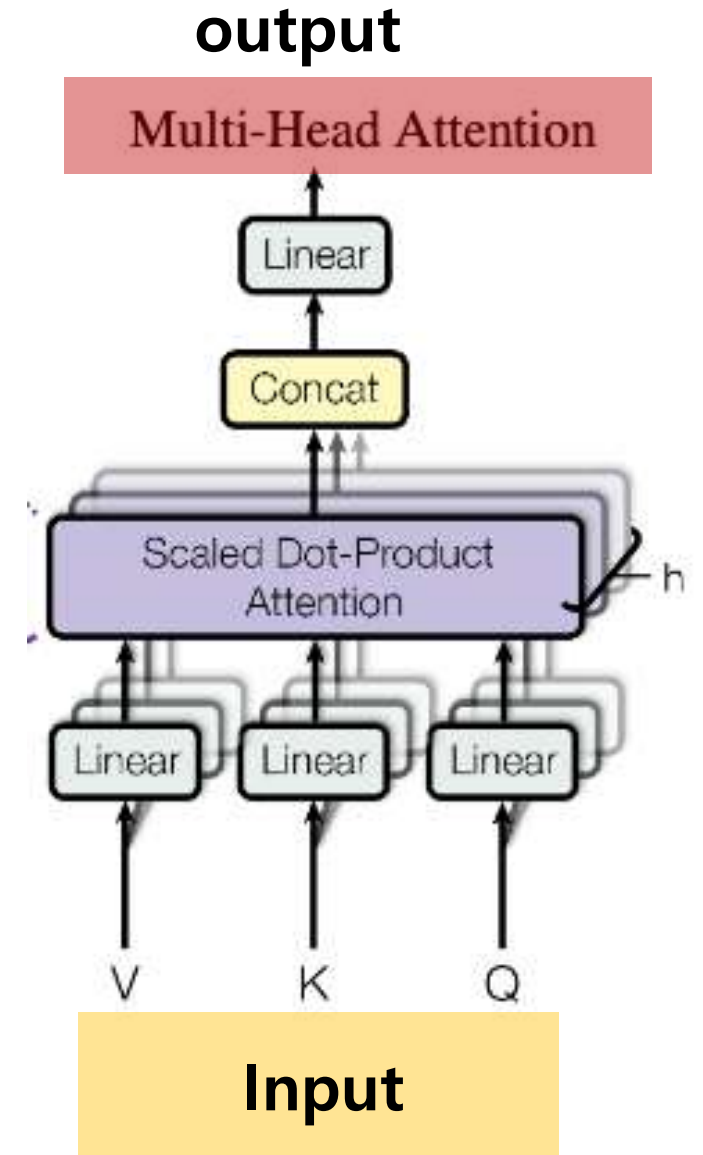
## - Residual Connection

$$\text{output} = \text{Input} + \text{Sublayer}(\text{Input})$$

(ex. self-attention, FFNN)



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



## Norm

### - Layer Normalization

순차적 데이터 처리나 각 시점(time step)에서  
독립적으로 데이터를 처리해야 하는 RNN과 같은 구조에서 사용되는 정규화 방법

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

$$LN(\mathbf{z}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{(\mathbf{z} - \boldsymbol{\mu})}{\sigma} \odot \boldsymbol{\alpha} + \boldsymbol{\beta},$$
$$\boldsymbol{\mu} = \frac{1}{D} \sum_{i=1}^D z_i, \quad \sigma = \sqrt{\frac{1}{D} \sum_{i=1}^D (z_i - \boldsymbol{\mu})^2},$$

$\boldsymbol{\alpha}$  : 각 요소의 스케일 조정  
 $\boldsymbol{\beta}$  : 시프트 조정

### Position-wise FFNN

position-wise fully connected feed-forward network

: 각 Head가 만들어낸 Attention을 치우치지 않게 균등하게 섞는 역할

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

Diagram illustrating the dimensions of the FFN operation:

- $(N, d_{model})$  is the input dimension (indicated by a downward arrow from  $x$ ).
- $(d_{model}, d_{ff})$  is the output dimension (indicated by an upward arrow from the result).
- $(d_{ff}, d_{model})$  is the intermediate dimension (indicated by a downward arrow from  $W_2$ ).

$$H = \text{ReLU}(W_1 x + b_1)$$

입력 벡터를 더 높은 차원의 표현 공간으로 매핑

$$z = W_2 H + b_2$$

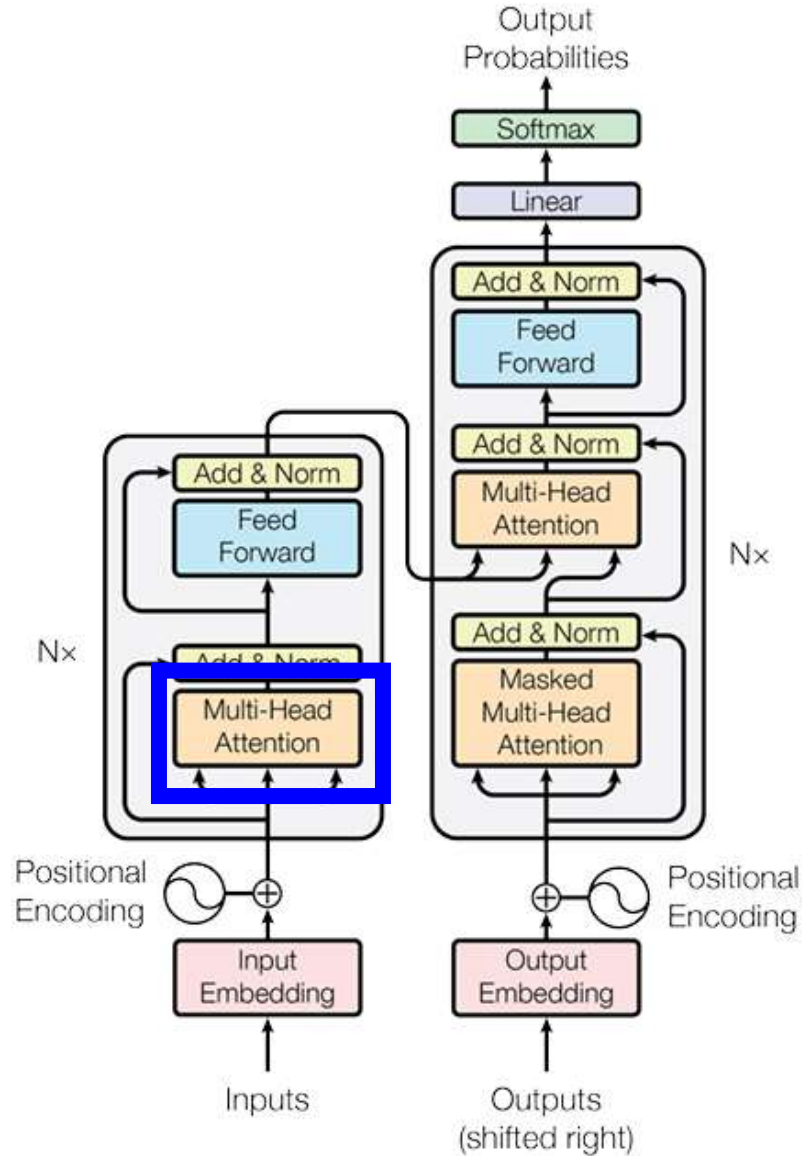
확장된 벡터  $H$ 를 다시 원래 차원  $d_{model}$ 로 매핑

$N$  : 입력 데이터의 개수

$d_{model}$  : 모델의 차원 수 (512)

$d_{ff}$  : 피드포워드 네트워크의 내부 레이어 차원 수..일반적으로 4배(2048)

## 5. Applications of Attention in our Model

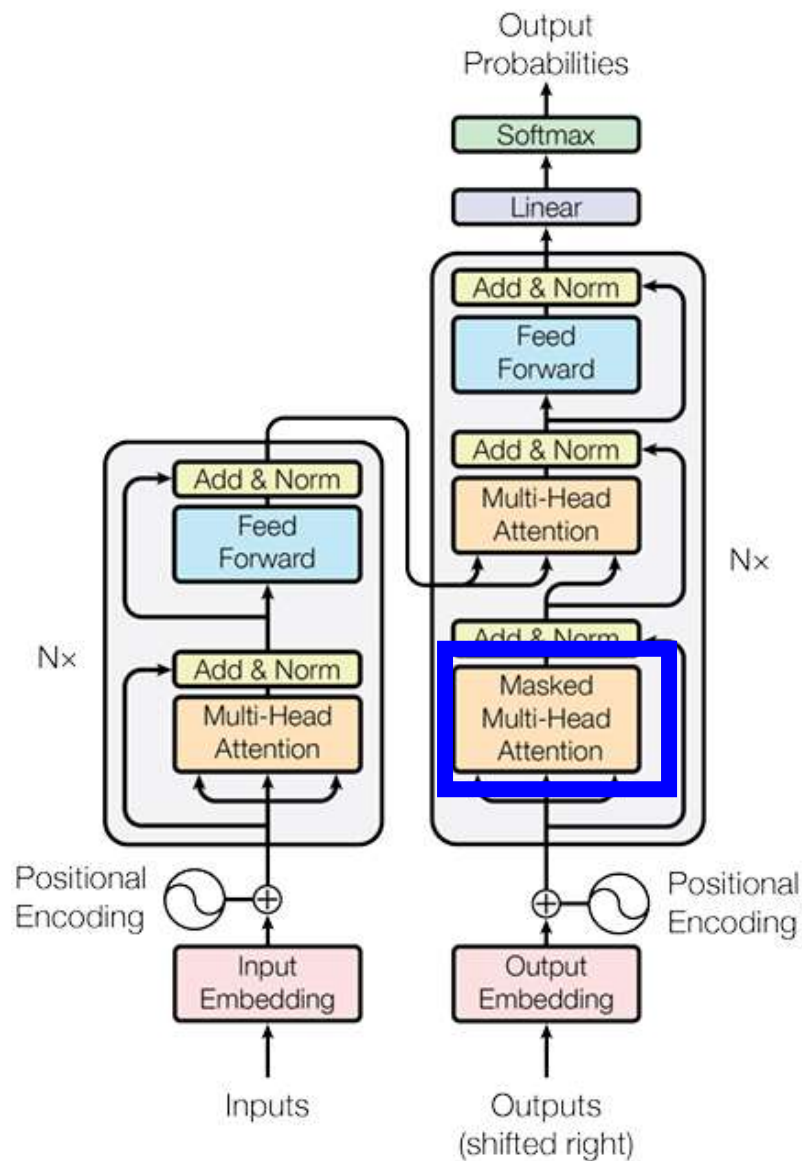


### 1. self-attention in Encoder

Q, K, V 모두 Encoder로부터 가져온다.  
(단어들 간의 연관성 학습)

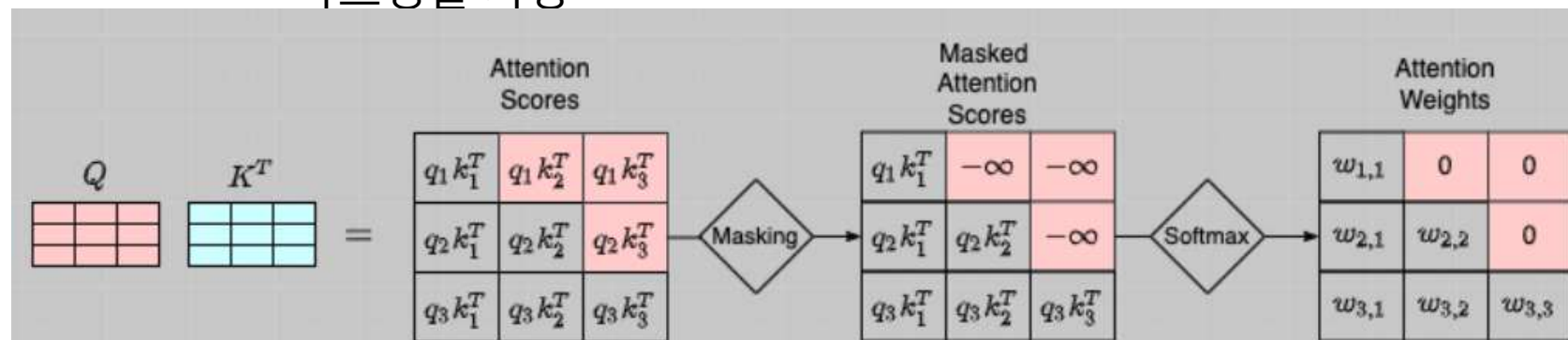
Figure 1: The Transformer - model architecture.

## 5. Applications of Attention in our Model



### 2. self-attention in Decoder (Masked)

미래의 위치에 있는 토큰 정보를 미리 "보지 않도록" 하기 위해  
마스킹을 사용



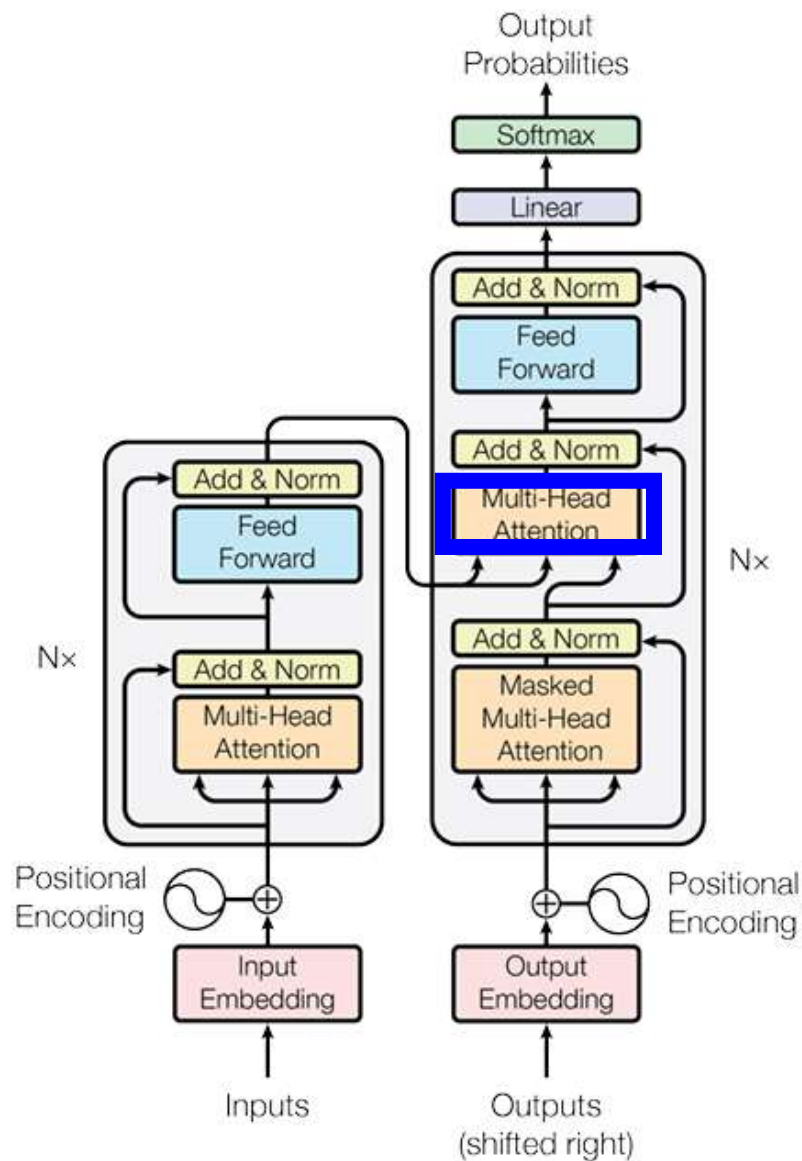
$i < j$  는

주어진 입력 값이 볼 수 없는 미래 시점의 입력 값과의 유사도

- ➔ 행렬의 대각선 윗부분을  $-\infty$ 로 변경
- ➔ softmax를 취하여 Attention Weight를 0으로 만듦 (미래 시점 값 고려 X)

Figure 1: The Transformer - model architecture.

## 5. Applications of Attention in our Model



### 3. Encoder-Decoder attention

Query -> 이전 디코더 레이어에서 가져옴  
key, Value -> 인코더의 출력에서 가져옴

seq2seq에서는 context vector를 통해 전체 입력 시퀀스에 대한 정보 통합했음

->

Transformer은 인코더의 K,V 정보를 활용하여 다음 토큰 예측!

Figure 1: The Transformer - model architecture.



			어텐션 헤드수    어텐션 K 차원    어텐션 V 차원																				
			$N$	$d_{\text{model}}$	$d_{\text{ff}}$	$h$	$d_k$	$d_v$	$P_{\text{drop}}$	$\epsilon_{ls}$	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$									
base			6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65									
Model	BLEU		(A)			1 4 16 32	512 128 32 16	512 128 32 16				5.29	24.9										
	EN-DE	EN-FR										5.00	25.5										
4.91												25.8											
23.75												5.01	25.4										
ByteNet [18]	23.75		(B)									5.16	25.1	58									
Deep-Att + PosUnk [39]	39.2											5.01	25.4	60									
GNMT + RL [38]	24.6	39.92																					
ConvS2S [9]	25.16	40.46																					
MoE [32]	26.03	40.56																					
Deep-Att + PosUnk Ensemble [39]	40.4		(C)	2 4 8	256 1024		32 128	32 128				6.11	23.7	36									
GNMT + RL Ensemble [38]	26.30	41.16										5.19	25.3	50									
ConvS2S Ensemble [9]	26.36	<b>41.29</b>										4.88	25.5	80									
Transformer (base model)	27.3	38.1										5.75	24.5	28									
Transformer (big)	<b>28.4</b>	<b>41.8</b>										4.66	26.0	168									
			(D)		1024 4096				0.0 0.2			5.12	25.4	53									
												4.75	26.2	90									
			(E)									5.77	24.6										
												4.95	25.5										
												4.67	25.3										
												5.47	25.7										
			big	6	1024	4096	16		0.3		300K	4.92	25.7	213									
												<b>4.33</b>	<b>26.4</b>										

학습된 임베딩으로  
대체