

## Self-Attention and Transformers

### Recurrent models 문제점

① Linear interaction distance issue 멀리 떨어진 단어간 상호작용 어려움

RNNs are unrolled "left-to-right",

encodes linear locality: nearby words often affect each other

문제는 RNNs take  $O(\text{seq.length})$  steps for distant word pairs to interact

$\Rightarrow$  hard to learn long-distance dependencies (gradient 문제)

② Lack of parallelizability issue 병렬연산 불가능

GPU는 한번에 여러 토큰끼리 계산 가능

But, future RNN hidden states can't be computed in full before  
past RNN hidden states have been computed 순차有

Inhibits training on very large datasets

Alternative:

Word window

- aggregate local contexts

- Sequence length 증가해도 병렬처리 불가한 연산 증가 X

- 하지만 long distance dependency issue 해결 불가능

Attention

- Set of values로부터 정보에 접근하고 통합하기 위해

각 단어의 representation을 query로 다른

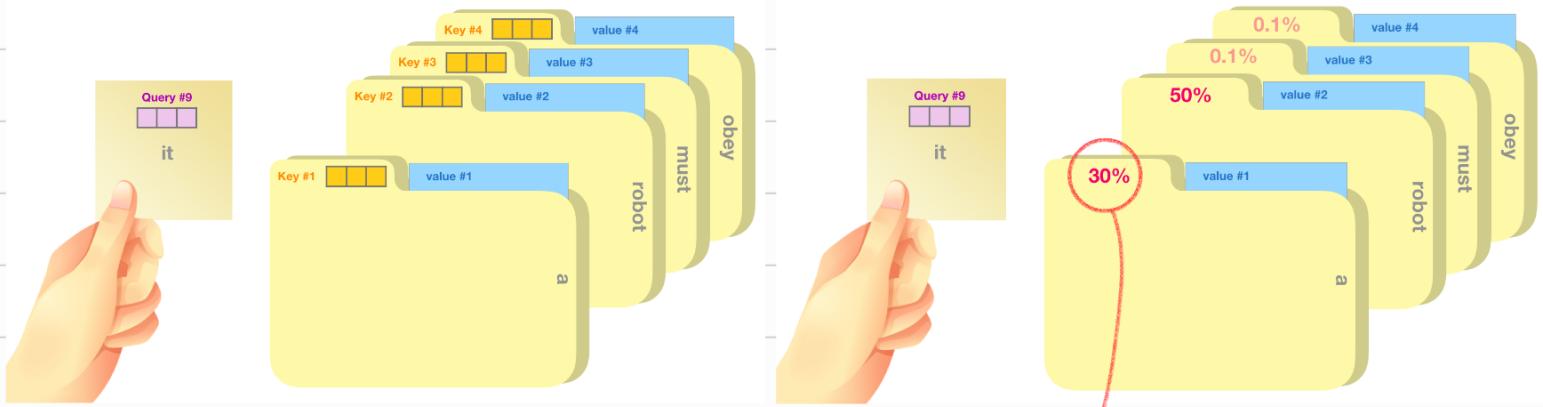
- Number of unparallelizable operations does not increase sequence length

- Maximum interaction distance:  $O(1)$  since all words interact at

- RNN의 문제를 해결 every layer

한 문장 안에서 attention 계산  $\Rightarrow$  Self-attention 병렬연산 O, Long distance O

# Self-Attention



Query : 현재 보고 있는 단어의 representation  
다른 단어를 평가하는 기준

Key : query 와 관련 있는 단어를 찾을 때 label/체계 활용되는 vector  
query 와 key 를 통해 softmax를 거쳐 attention score 를 구한 것

Value: query 와 key 를 통해 탐색해 실제로 사용할 값

$$V_i = K_i = Q_i = x_i$$

Attention score  $e_{ij} = q_i^T k_j$

Attention weight  $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_j \exp(e_{ij})}$

Attention output  $\text{output}_i = \sum_j \alpha_{ij} v_j$  가중치

Attention 만으로 NLP 모델 만들 수 없는 이유

- 1) 순서 정보 없음
- 2) 선형 결합만 있음
- 3) 미래 sequence data 활용

## 1) Sequence order

위치 정보를 포함하는 query key value 만들어야

Position vectors  $P_i \in \mathbb{R}^d$ , for  $i \in \{1, 2, \dots, T\}$

$$\begin{pmatrix} q_i = \tilde{q}_i + p_i \\ k_i = \tilde{k}_i + p_i \\ v_i = \tilde{v}_i + p_i \end{pmatrix} \text{ 를 first layer에 넣으면 } \begin{matrix} \text{위치} \\ \text{정보} \end{matrix} \text{ 를 포함한 채로 연산} \\ (\text{tildes})$$

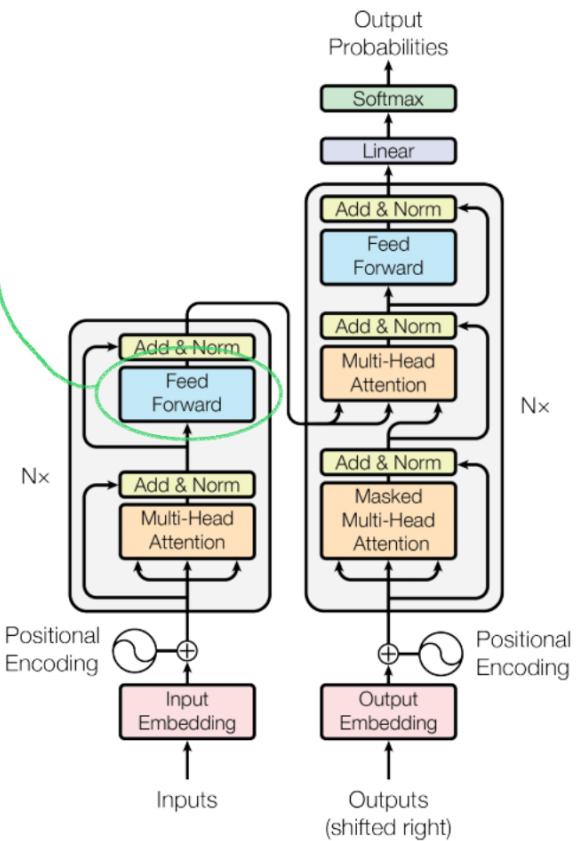
\* Transformer에서는 sin, cos 조합한 Sinusoidal position vector 사용

## 2) Nonlinearities in self-attention

Feed Forward network로 해결 가능

$$m_i = \text{MLP}(\text{output}_i) \\ = W_2 * \text{ReLU}(W_1 * \text{output}_i + b_1) + b_2$$

같은 layer의 FF는 같은 parameter 공유



## 3) Decoder에서 미래 seq 정보 접근하기

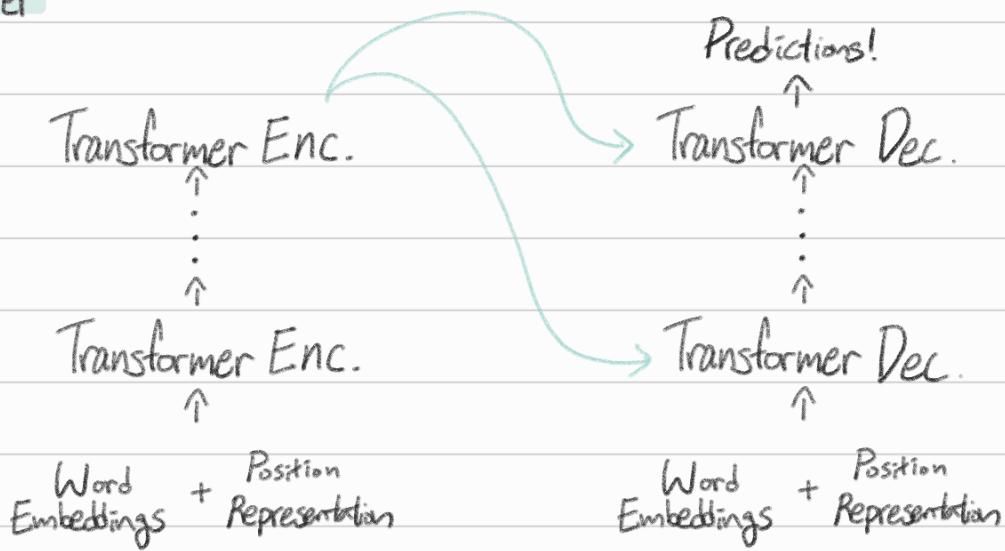
방법 연산이 가능하다는 것은

미래 단어 예측에 미래 seq 정보 사용 가능

query index가 key index 보다 클 때만 계산

$$e_{ij} = \begin{cases} q_i^T k_j, j < i \\ -\infty, j \geq i \end{cases} \quad \text{(<Masking>)}$$

# Transformer



$x_1, x_2, \dots, x_T \rightarrow$  Transformer Enc.의 input vectors ;  $x_i \in \mathbb{R}^d$

$K \in \mathbb{R}^{d \times d}$  가 key matrix일 때  $k_i = Kx_i$

$Q \in \mathbb{R}^{d \times 1}$  가 query matrix일 때  $q_i = Qx_i$

$V \in \mathbb{R}^{d \times d}$  가 value matrix일 때  $v_i = Vx_i$

1. Input vector가  $d$  차원일 때, 결합한 행렬이  $X = [x_1; x_2; \dots; x_T] \in \mathbb{R}^{T \times d}$

2.  $X$  와  $d \times d$  matrix  $K, Q, V$  를 dot product 한다

$XK, XQ, XV \in \mathbb{R}^{T \times d}$  라는 key, query, value 산출

3. Attention score 계산하기 위해 query 와 key 곱

$$XQ(XK)^T = XQK^T X^T \in \mathbb{R}^{T \times T}$$

4. Attention score 결과를 softmax 통해归一화하고 결합하여 output 도출

$$\text{Softmax}(XQK^T X^T)XV = \text{output}$$

Multi-headed attention : 한 번에 여러 부분에 집중

1. Input sentence given

2. Embed each word

3. Split into 8 heads, multiply  $X$  or  $R$  with weights matrix

#### 4. Calculate attention

#### 5. Concat output matrices $Z$ ,

Multiply with weight matrix  $W^D$  to produce the output of the layer

⇒ Input vector 와 같은 크기의 Output vector

Residual Connection : 자기 자신을 더해

$$X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$$

i-1 state와 얼마나 다른지 학습

New  $f(x) = \text{Old } f(x) + x$  라고 표현할 때,

미분 결과  $f'(x) + 1$  이되어 gradient가 아주 작아도 1만큼 이어줄

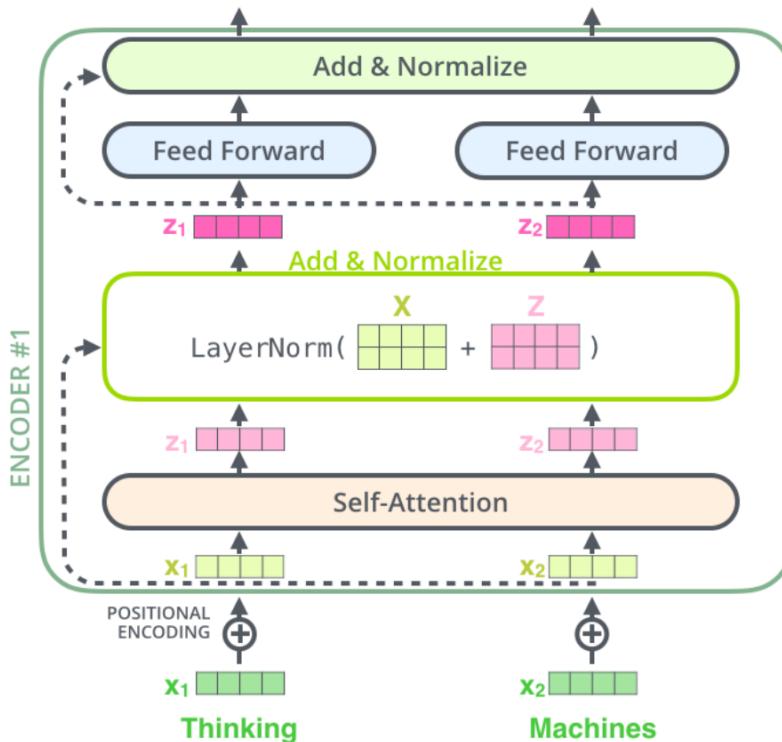
기울기를 smoothing ⇒ global optima로 수렴

Layer normalization : 한 layer에서 하나의 input sample  $x$ 에 대해

모든 feature에 대한 평균과 분산으로 normalization 해주는 것

gradient를 normalize해서 흐리게

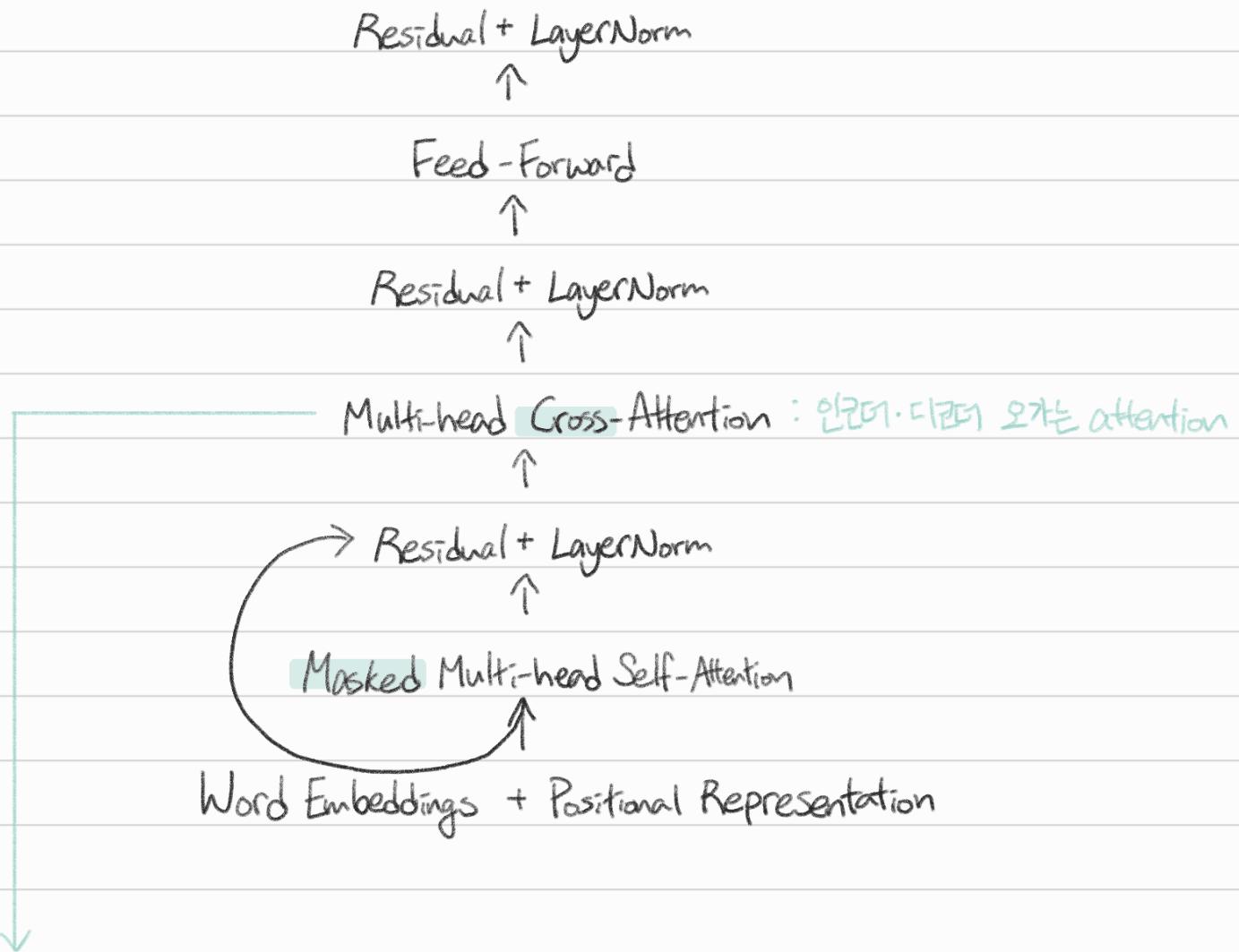
Scaling the dot product : dimension이 커지면 vector의 dot product도 커지는 경향이  
너무 커서 다른 값 무시하는 일 없도록 scaling



## Feed Forward

- Sequence마다 독립적으로 적용
- 하나의 layer 안에서는 동일한 parameter 공유

## Decoder



$h$ : encoder의 output vector

$z$ : decoder의 input vector

Decoder에서 현재 처리하는 **Query**인  $ZQ$  가져오고,

Encoder의 **Key**인  $HK$ 를 전치해  $(HK)^T$ 를 곱해  $ZQK^T H$  att.score 계산

$ZQK^T H$ 를 Encoder의 **Value**인  $HV$ 로 가중함 :  $ZQK^T H V$

(output)