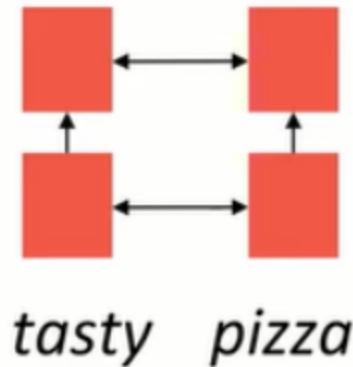


Self Attention & Transformer

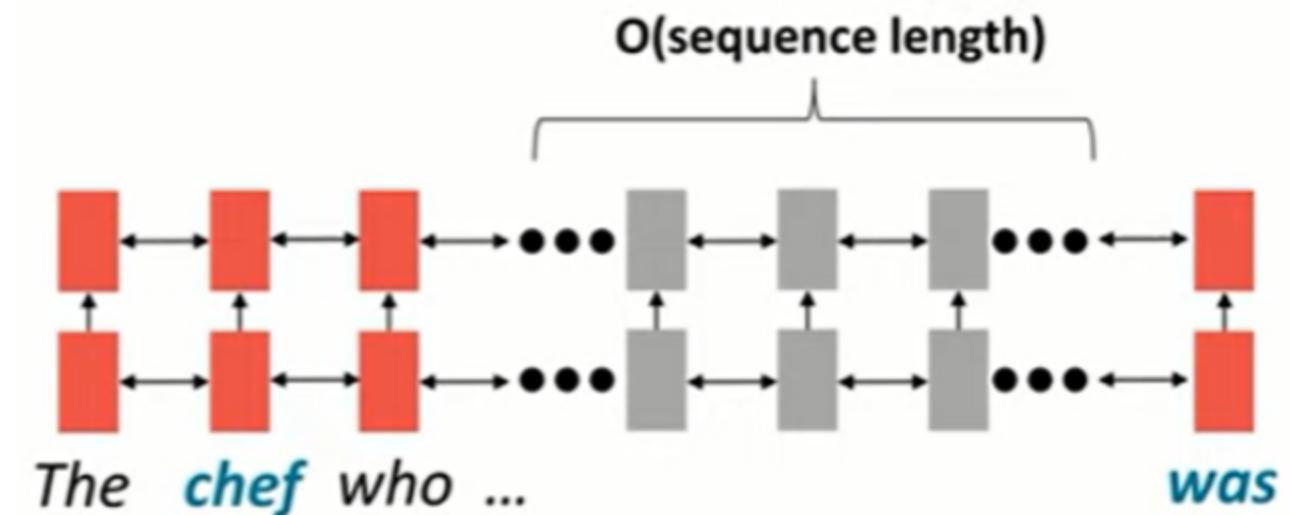
RNN to Attention-based NLP models

■ RNN의 문제점: 1. Linear Interaction distance (Long distance dependency)

RNN은 왼쪽에서 오른쪽으로 순차적으로 연산이 진행
→ 가까운 단어 사이의 상호작용은 원활히 반영됨



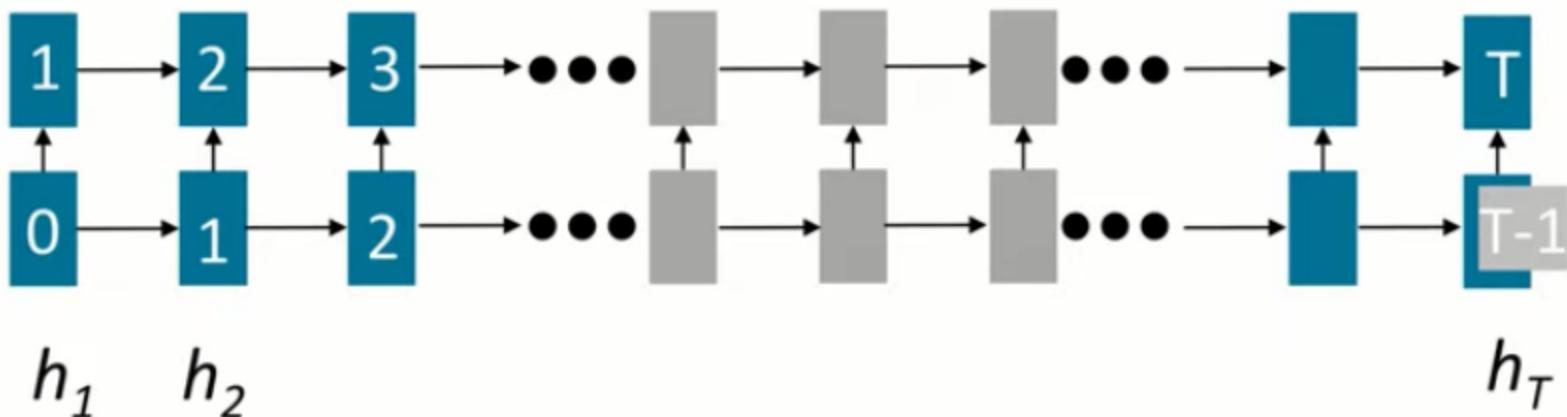
오른쪽처럼 주어 *chef*와 동사 *was*가 멀리 떨어져 있는 경우,
RNN은 멀리 떨어진 단어 쌍이 영향을 주고받기 위해서는
O(sequence length) steps 만큼 연산이 이루어져야 하고
그 과정에서 Gradient 문제로 dependency 반영이 어렵다.



RNN to Attention-based NLP models

■ RNN의 문제점: 2. Lack of parallelizability (병렬 연산의 불가능)

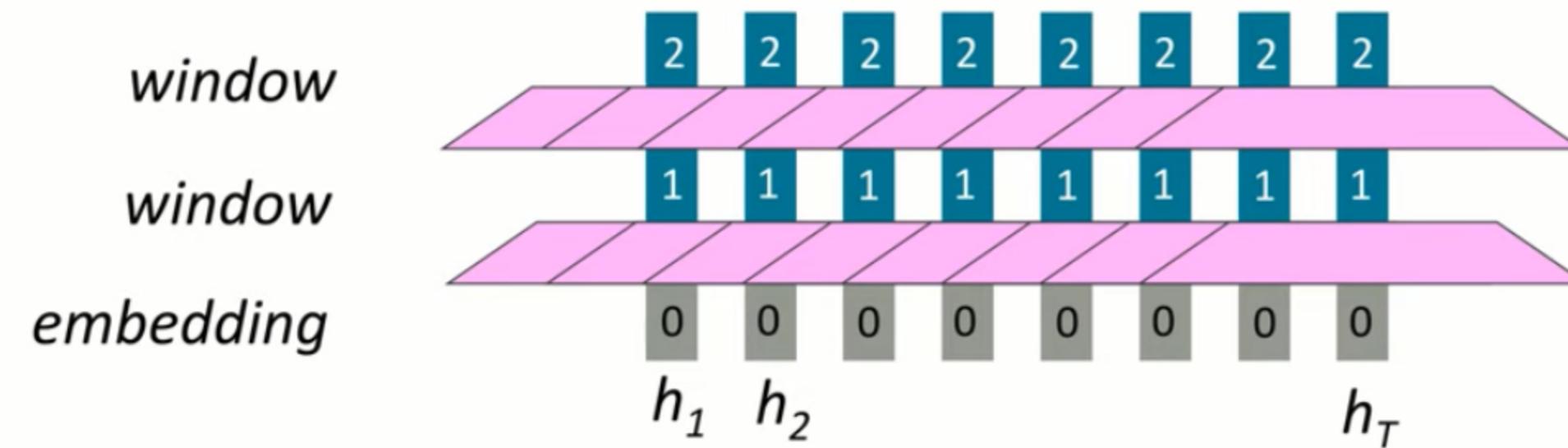
RNN이 순차적으로 연산이 이루어져야 하기 때문에 발생하는 문제점



이전 hidden state가 연산되어야 다음 hidden state가 연산될 수 있다.
→ GPU를 활용한 병렬 연산이 불가능하다.

RNN to Attention-based NLP models

■ word windows



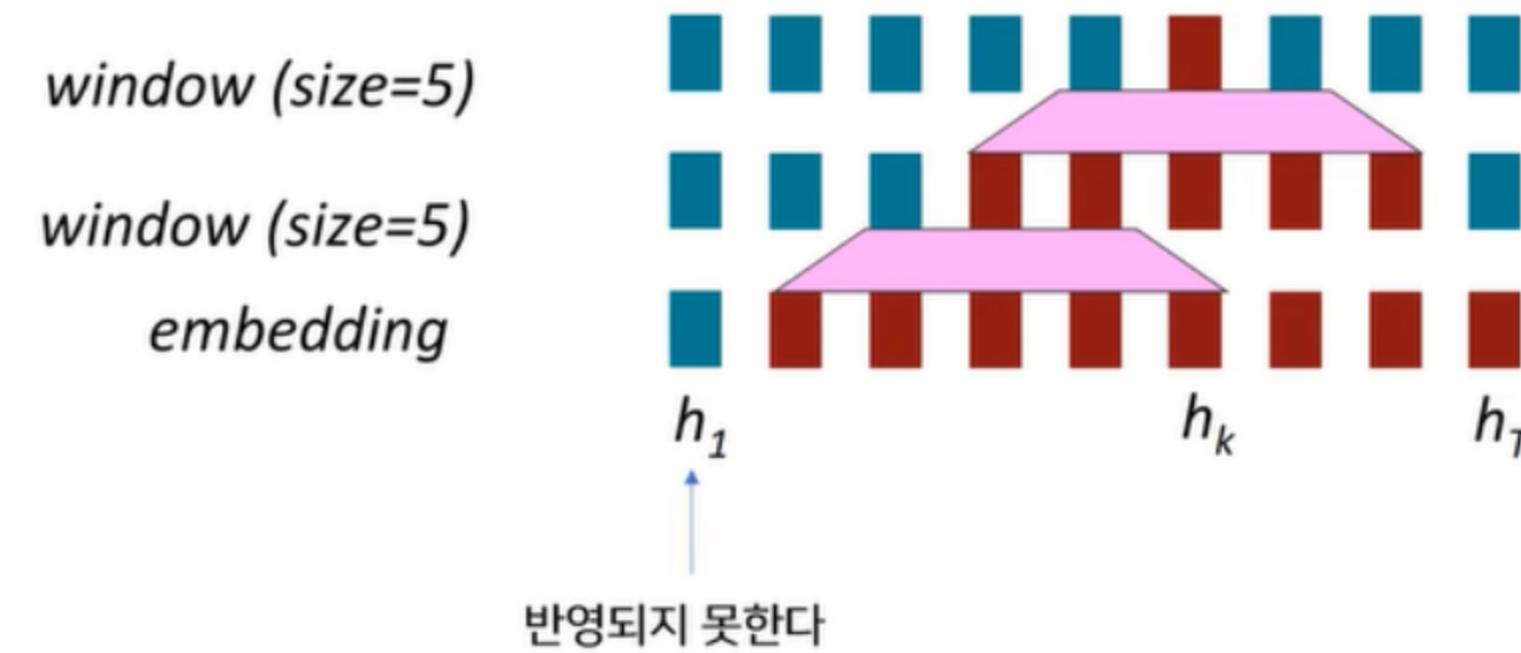
word window의 경우 sequence length가 증가해도 병렬처리가 불가한 연산이 증가하지 않는다.

Embedding layer에서 각각의 word가 독립적으로 embedding 할 수 있고
그 위 window layer에서는 아래 layer에서 연산이 이루어지기만 하면 되기 때문에 필수 연산의 수는
각각 독립적으로 1씩만 증가한다.

RNN to Attention-based NLP models

■ word windows

그러나 Long-distance dependency 문제는 해결하지 못한다.



word window는 붉은색으로 표시한 window size만큼의 local context만을 통합한다.

제일 위에 있는 layer가 인코더의 output을 의미한다고 할 때 h_1 이 반영되지 못하고 있다.

즉, 멀리 있는 단어 사이의 상호작용이 어렵다.

멀리 떨어진 단어의 dependency를 반영하기 위해서는 더 깊게 window를 쌓아야 하는 한계점이 존재한다.

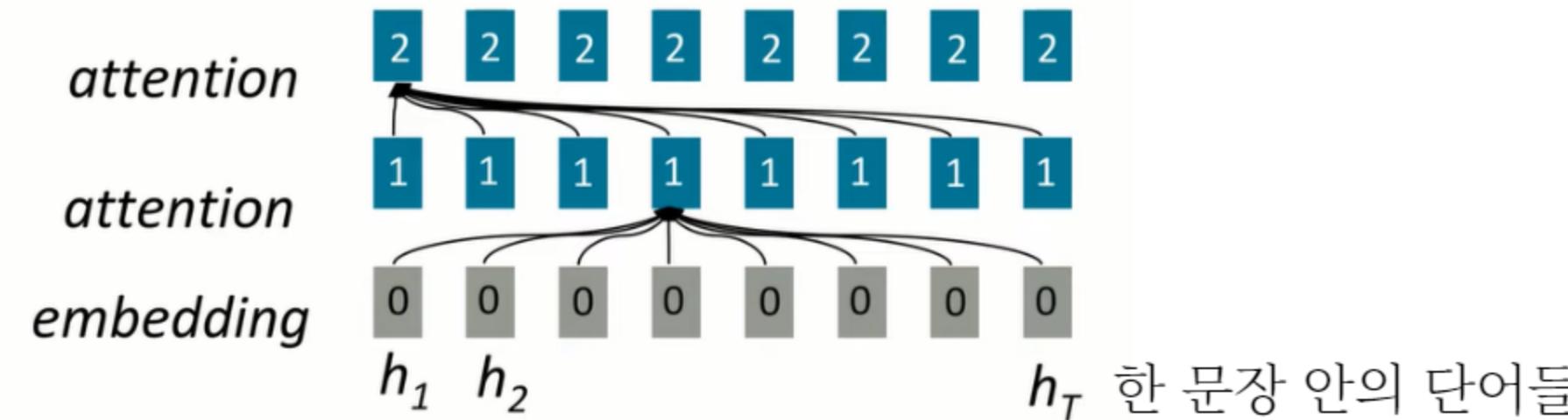
RNN to Attention-based NLP models

■ Attention

building block으로 attention을 사용한다면?

(attention은 디코더에서 인코더로 주목해야하는 부분을 파악하는 방식)

아래 그림은 Attention을 한 문장 안에서 고려한 경우



embedding layer : 각각 독립하게 연산이 가능

attention layer: 바로 이전 layer만 완료되면 다음 layer에 모든 sequence state가 연산이 가능

layer 사이의 depth로는 병렬연산이 불가능하지만 가로 sequence에 대해서는 병렬 연산이 가능하다.

더 많은 layer가 추가되어도 연산의 수가 독립적으로 증가하기 때문에 병렬 연산을 유지

모든 state가 attention으로 연결되어 있기 때문에 멀리 떨어진 단어도 O(1) 연산으로 상호작용이 가능

RNN to Attention-based NLP models

■ Self Attention

Self Attention: encoder deocder 사이가 아닌 한 문장 안에서 Attention을 계산하는 방법

Self Attention은 병렬 연산이 가능하고 Long distance dependency 문제가 발생하지 않는다
→ 새로운 building block으로 적절

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

병렬연산 가능 Long distance dependency
문제 해결

RNN to Attention-based NLP models

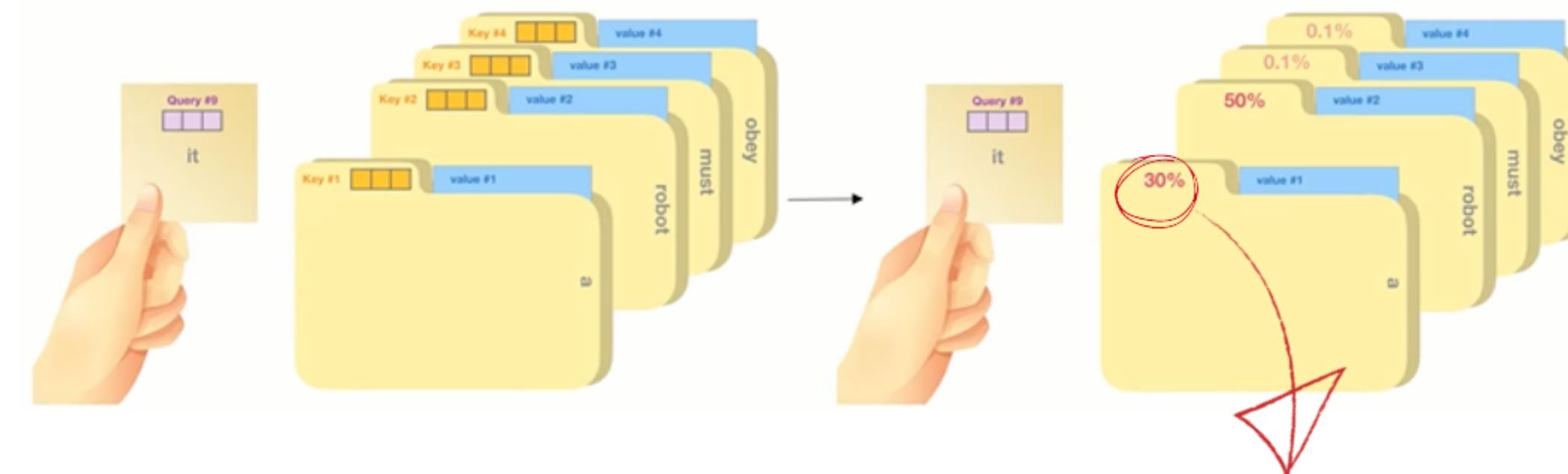
■ Self Attention

Query, Key, Value : RNN의 hidden state 값을 용도에 따라 구분한 것이라고 볼 수 있다.

Query: 현재 보고 있는 단어의 representation, 다른 단어를 평가하는 기준

Key: 현재 내가 들고 있는 query와 관련 있는 단어를 찾을 때 label처럼 활용되는 벡터

Value: query와 key를 통해 탐색하여 실제로 사용할 값



it이라는 단어를 들고 집중해야 할 부분을 key를 통해 탐색

query와 key를 곱해서 softmax를 취해 attention score를 구한 것

RNN to Attention-based NLP models

■ Self Attention

self attention의 query, key, value는 용도에 따라 구분한 것인데
구분을 생각하지 않고 모두 hidden state라고 생각한다면 attention과 동일한 식

Self-Attention

$$v_i = k_i = q_i = x_i$$

query, key, value를 input vector x 와 동일하게 정의

Attention score : $e_{ij} = q_i^\top k_j$

self attention은 한 문장 안에서 이루어지기 때문에
 s , h 구분 없이 용도에 따라 벡터를 구분

Attention weight : $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$

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

Attention

$$\mathbf{e}^t = [\mathbf{s}_t^T \mathbf{h}_1, \dots, \mathbf{s}_t^T \mathbf{h}_N] \in \mathbb{R}^N$$

s : 디코더 hidden state h : 인코더 hidden state

$$\alpha^t = \text{softmax}(\mathbf{e}^t) \in \mathbb{R}^N$$

$$\mathbf{a}_t = \sum_{i=1}^N \alpha_i^t \mathbf{h}_i \in \mathbb{R}^h$$

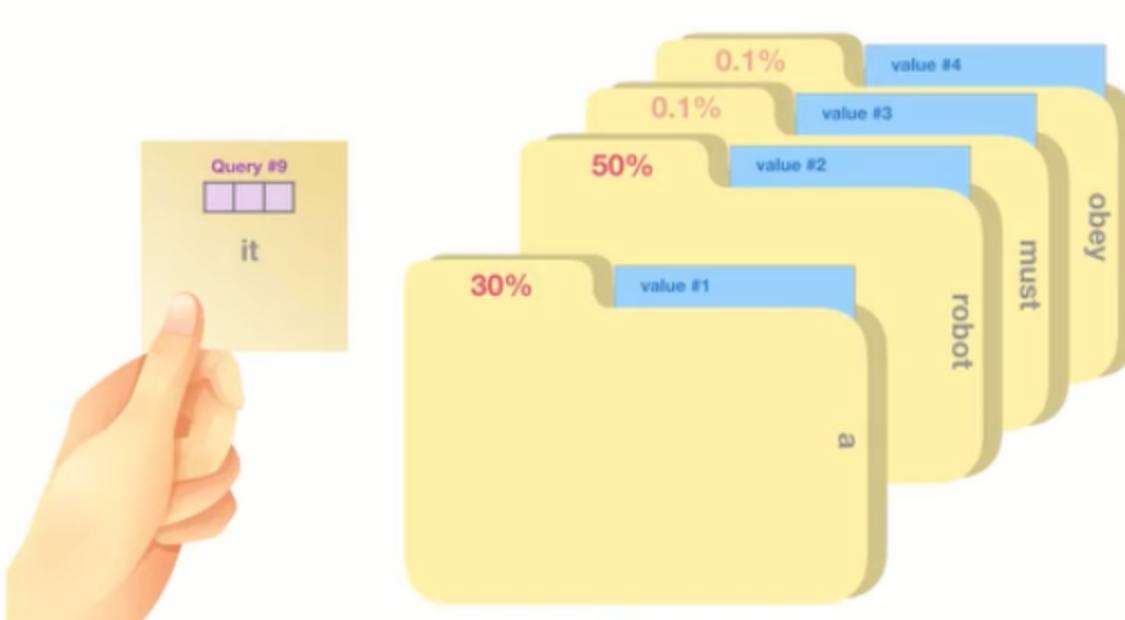
RNN to Attention-based NLP models

■ Self Attention

문장: A robot must obey the orders given it

1. it을 query로 탐색한다면 의미상 관련이 깊은 a와 robot에 높은 attention score가 산출
2. attention score를 이용해 모든 word의 value를 가중 합하면 output으로서 it에 대한 새로운 representation을 얻음

→ self attention: 단어 임베딩 벡터에 attention을 반영하여 새롭게 재표현하는 함수



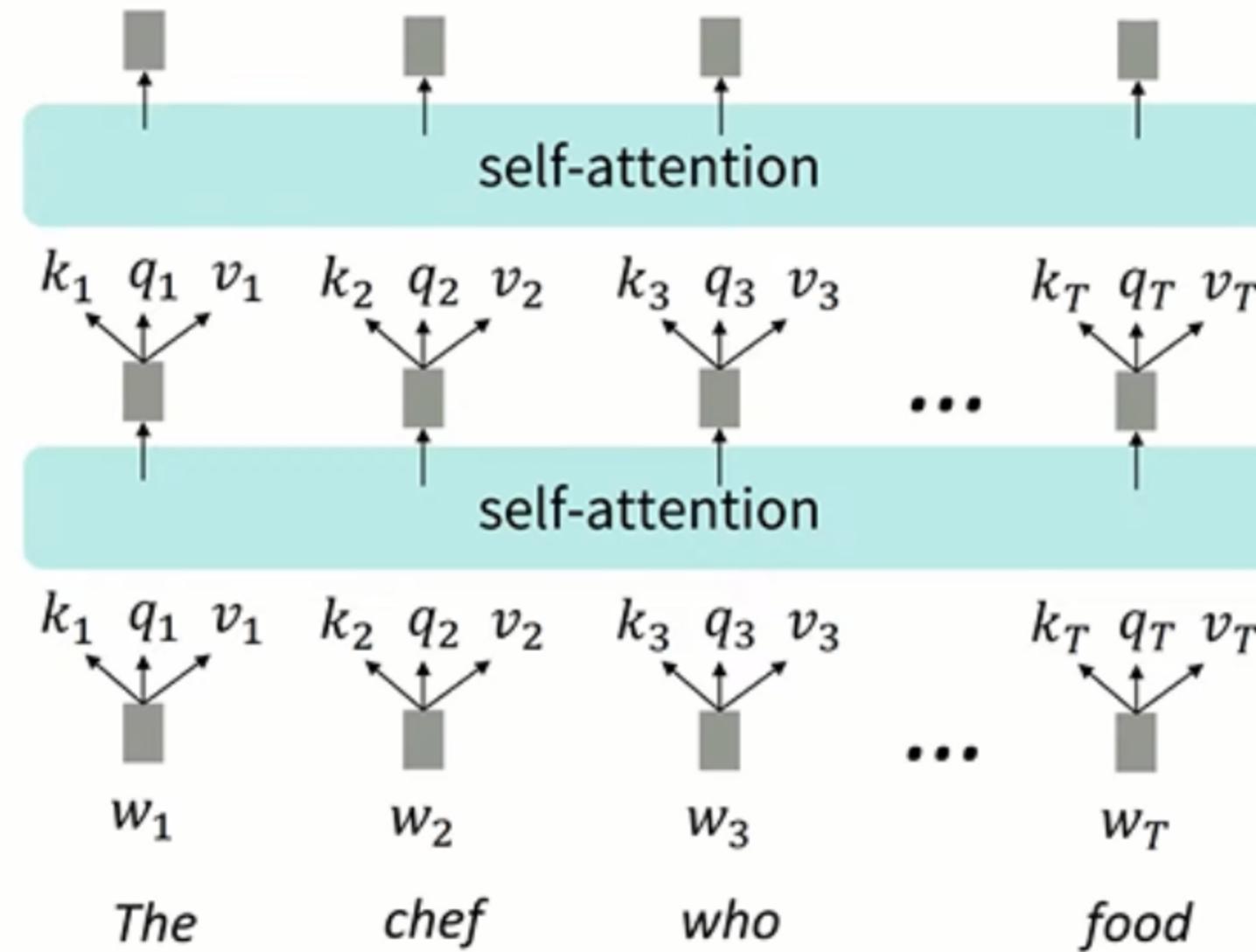
Word	Value vector	Score	Value X Score
<S>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
Sum:			

RNN to Attention-based NLP models

- 단순히 Self Attention을 쌓기만 하면 충분한 NLP 모델이 될 수 없다.

문제 1. Sequence order

self attention은 병렬 처리로 동시에 처리가 가능하기 때문에 순서에 대한 정보가 없다.



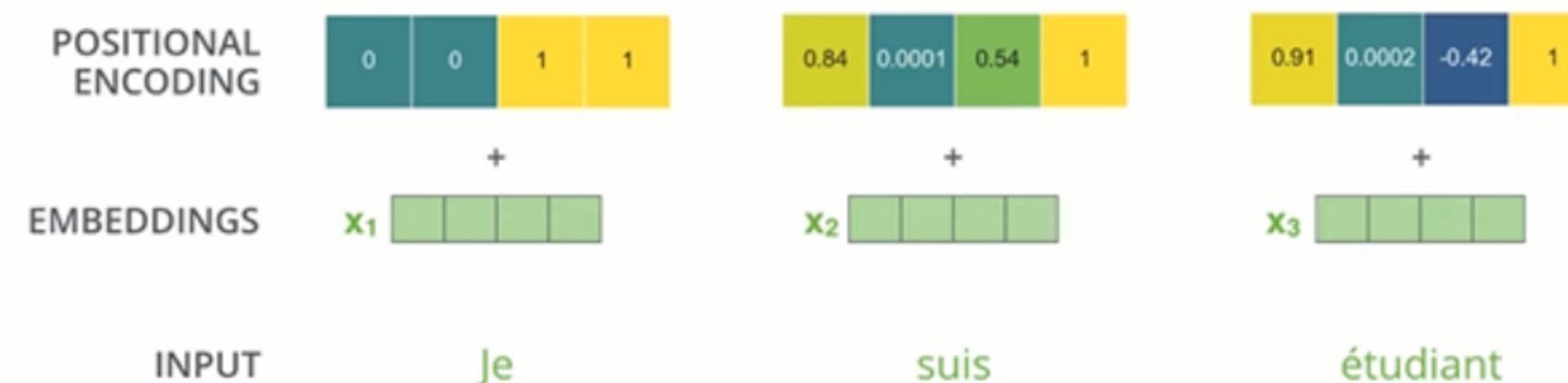
RNN to Attention-based NLP models

■ 문제 1. Sequence order

문제를 해결하기 위해서 위치 정보를 포함하는 query, key, value를 만들어야 함

위치 정보를 표현하는 position vector를 정의하고 기존의 query, key, value에 더해준다.

$$\begin{aligned} v_i &= \tilde{v}_i + p_i \\ \text{Position Vectors } p_i &\in \mathbb{R}^d, \text{ for } i \in \{1, 2, \dots, T\} \\ q_i &= \tilde{q}_i + p_i \\ k_i &= \tilde{k}_i + p_i \end{aligned}$$

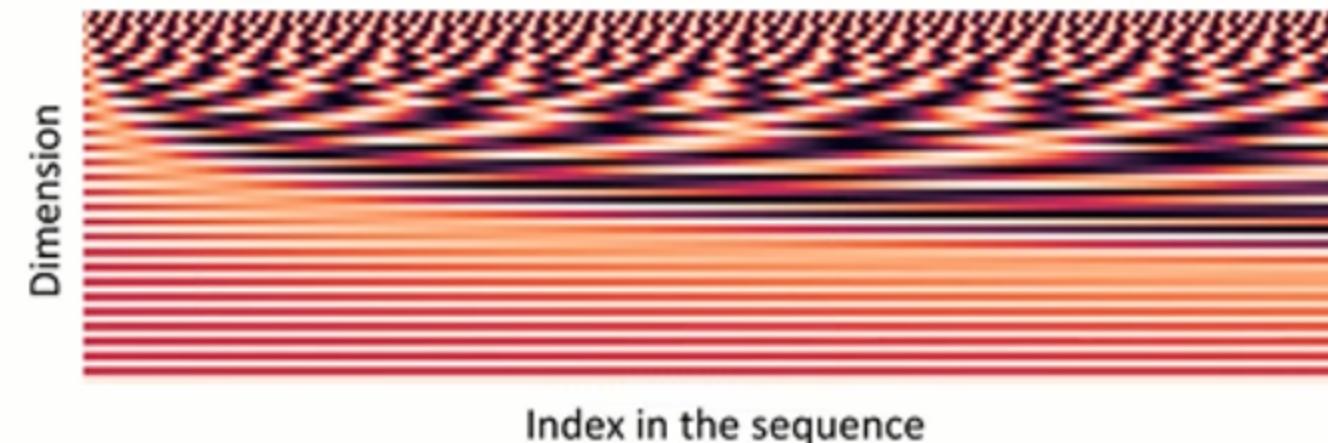


RNN to Attention-based NLP models

■ 문제 1. Sequence order

논문에서는 \sin , \cos 를 이용한 Sinusoidal를 position vector로 사용

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



다른 index를 갖고 있다면 다른 값을 갖는 벡터

→ 절대적인 위치마다 다른 값을 가지므로 query, key, value에 더해졌을 때도
위치 정보를 embedding 할 수 있다.

장점: 상대적인 비교가 중요하므로 함수 주기보다 sequence가 길어도 정보를 잃지 않는다.

단점: 식을 보면 학습할 수 있는 파라미터가 없다. 데이터에 맞게 다른 위치 정보를 포함할 수 없다.

RNN to Attention-based NLP models

■ 문제 1. Sequence order_ 학습 가능한 position vector

- Relative linear position attention [Shaw et al., 2018] : 상대적 위치

Model	Position Information	EN-DE BLEU	EN-FR BLEU
Transformer (base)	Absolute Position Representations	26.5	38.2
Transformer (base)	Relative Position Representations	26.8	38.7
Transformer (big)	Absolute Position Representations	27.9	41.2
Transformer (big)	Relative Position Representations	29.2	41.5

- Dependency syntax-based position [Wang et al., 2019] : 구조 고려

Model Architecture	Zh⇒En					En⇒De WMT14
	MT03	MT04	MT05	MT06	Avg	
Hao et al. (2019c)	-	-	-	-	-	28.98
Transformer-Big	45.30	46.49	45.21	44.87	45.47	28.58
+ Structural PE	45.62	47.12 [†]	45.84	45.64 [†]	46.06	28.88
+ Relative Sequential PE	45.45	47.01	45.65	45.87 [†]	46.00	28.90
+ Structural PE	45.85 [†]	47.37 [†]	46.20 [†]	46.18 [†]	46.40	29.19 [†]

상대적인 위치를 적용한 방식이 높은 성능을 보임

장점: 현재 데이터에 맞게 학습이 가능

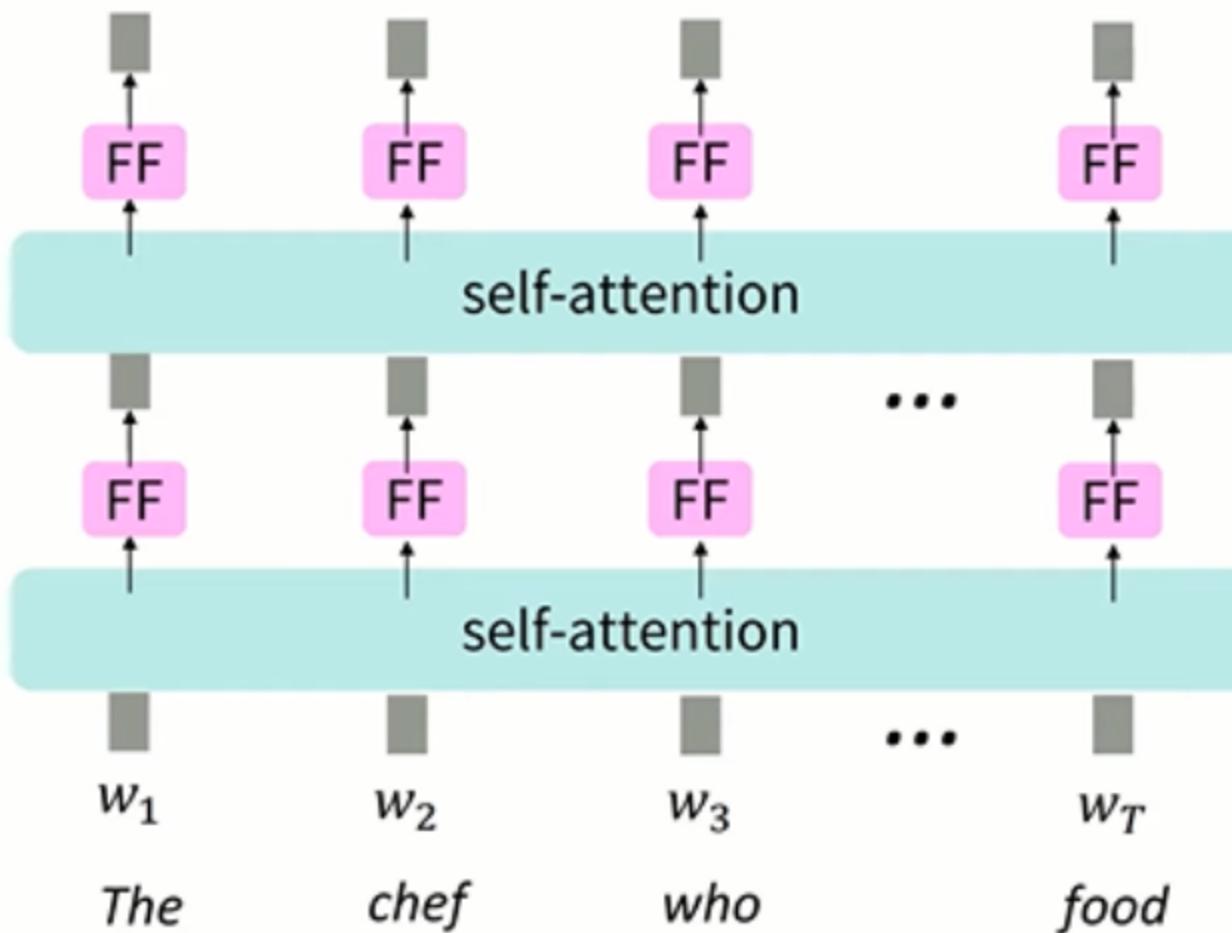
단점: 정해놓은 length 이상의 문장은 처리가 불가능

RNN to Attention-based NLP models

■ 문제 2. nonlinearities in self-attention

Self attention이 단순한 가중합 형태의 선형결합이기 때문에 NLP 모델로 작동하기 위해서는 비선형 함수가 필요하다.

→ 각각의 attention output 벡터에 feed forward network를 추가하는 것으로 해결



Feed Forward Network:

$$\begin{aligned} m_i &= \text{MLP}(\text{output}_i) \\ &= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2 \end{aligned}$$

ReLU를 비선형 함수로 활용

같은 layer의 FF는 같은 파라미터를 공유

RNN to Attention-based NLP models

■ 문제 3. decoder에서 미래 sequence 정보를 볼 수 있음

decoder에서 Language Modeling을 수행할 때 미래 sequence 정보를 볼 수 있다는 문제를 해결해야한다.
RNN의 경우 순서대로 연산이 이루어지기 때문에 별다른 처리가 없어도 미래 state를 연산에 활용하지 않는다.

그러나 self attention의 경우 병렬 연산이 가능한 구조이기 때문에 미래 단어를 예측하는
Language Modeling에서도 미래 단어를 연산에 활용할 수 있다.

$$e_{ij} = \begin{cases} q_i^\top k_j, & j < i \\ -\infty, & j \geq i \end{cases}$$

key index
: 나머지 word의 label

query index
: 지금 탐색중인 대상

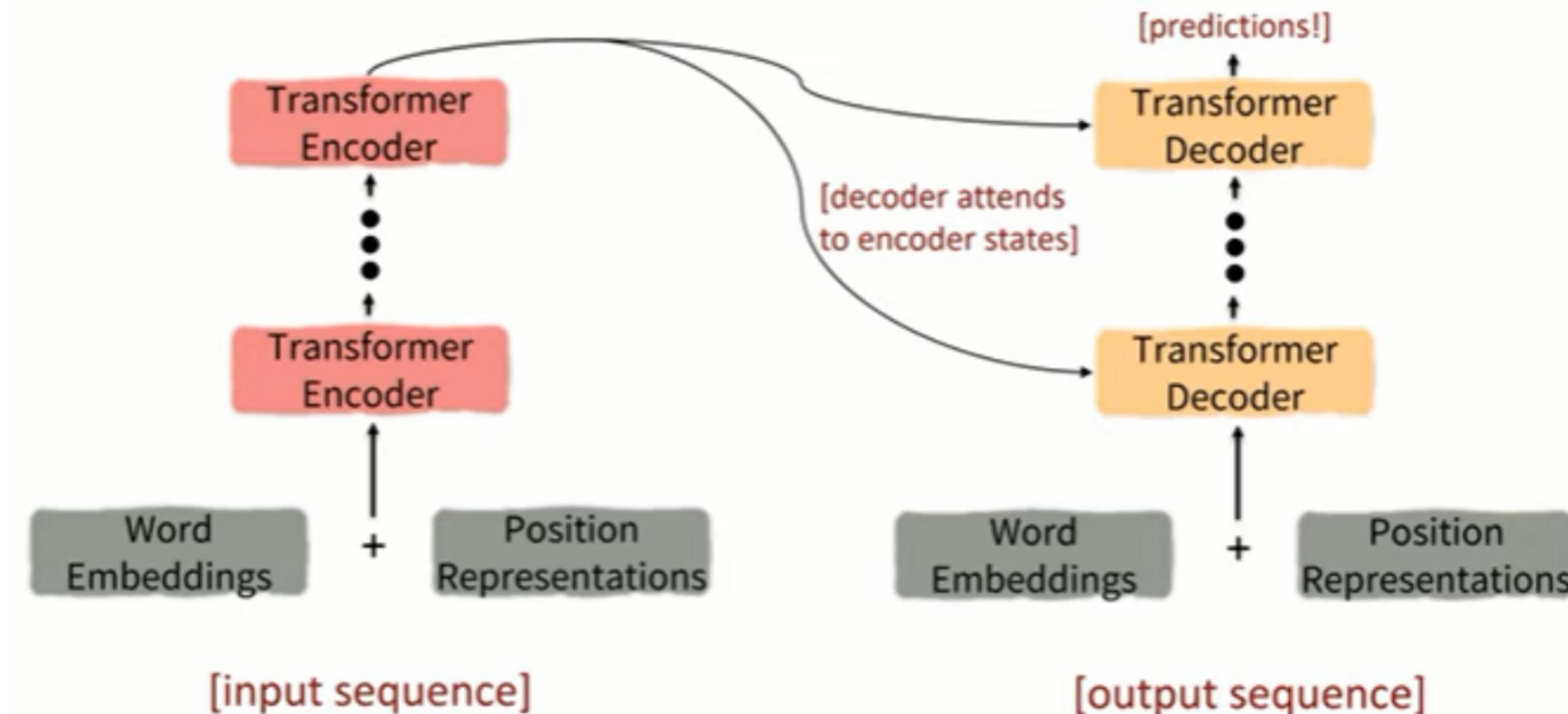
query idx가 key idx보다 클 때만 계산하고,
그렇지 않으면 $-\infty$ 를 대입 → 마스킹

query idx가 현재 탐색중인 단어이기 때문에
query idx보다 작은 key idx는 앞선 단어를 의미한다.

Transformer model structure

■ Transformer Encoder and Decoder

기존 Seq2Seq 모델의 Encoder-Decoder 구조는 그대로 유지하면서 building block이 Transformer로 변환



Transformer model

■ Query, Key, Value for Transformer

- 기본적인 self attention에서는 Query, Key, Value가 input 벡터 x 와 같은 값
- transformer에서는 학습하는 대상 Q, K, V를 input embedding x 와 곱하여 Query, Key, Value를 구한다.

$$x \times Q = \text{Query}$$

A diagram illustrating the computation of the Query. It shows a green input vector x (a 3x3 grid) multiplied by a purple weight matrix Q (a 3x3 grid) to produce the Query result (a 3x3 grid). The result is labeled "Query".

$$x \times K = \text{Key}$$

A diagram illustrating the computation of the Key. It shows a green input vector x (a 3x3 grid) multiplied by a orange weight matrix K (a 3x3 grid) to produce the Key result (a 3x3 grid). The result is labeled "Key".

$$x \times V = \text{Value}$$

A diagram illustrating the computation of the Value. It shows a green input vector x (a 3x3 grid) multiplied by a blue weight matrix V (a 3x3 grid) to produce the Value result (a 3x3 grid). The result is labeled "Value".

Transformer model

■ Query, Key, Value for Transformer

- X : input vector가 d dimension을 가질 때 이를 결합한 행렬
- Query, Key, Value를 계산하기 위해 행렬 X 와 행렬 Q , K , V 를 dot product
- 다음 행렬 연산을 수행

$$XQ \quad K^\top X^\top = XQK^\top X^\top \in \mathbb{R}^{T \times T}$$

- softmax를 취해 Attention score를 얻고 그를 가중합하여 output 산출

$$\text{softmax} \left(\begin{matrix} XQK^\top X^\top \end{matrix} \right) XV = \text{output} \in \mathbb{R}^{T \times d}$$

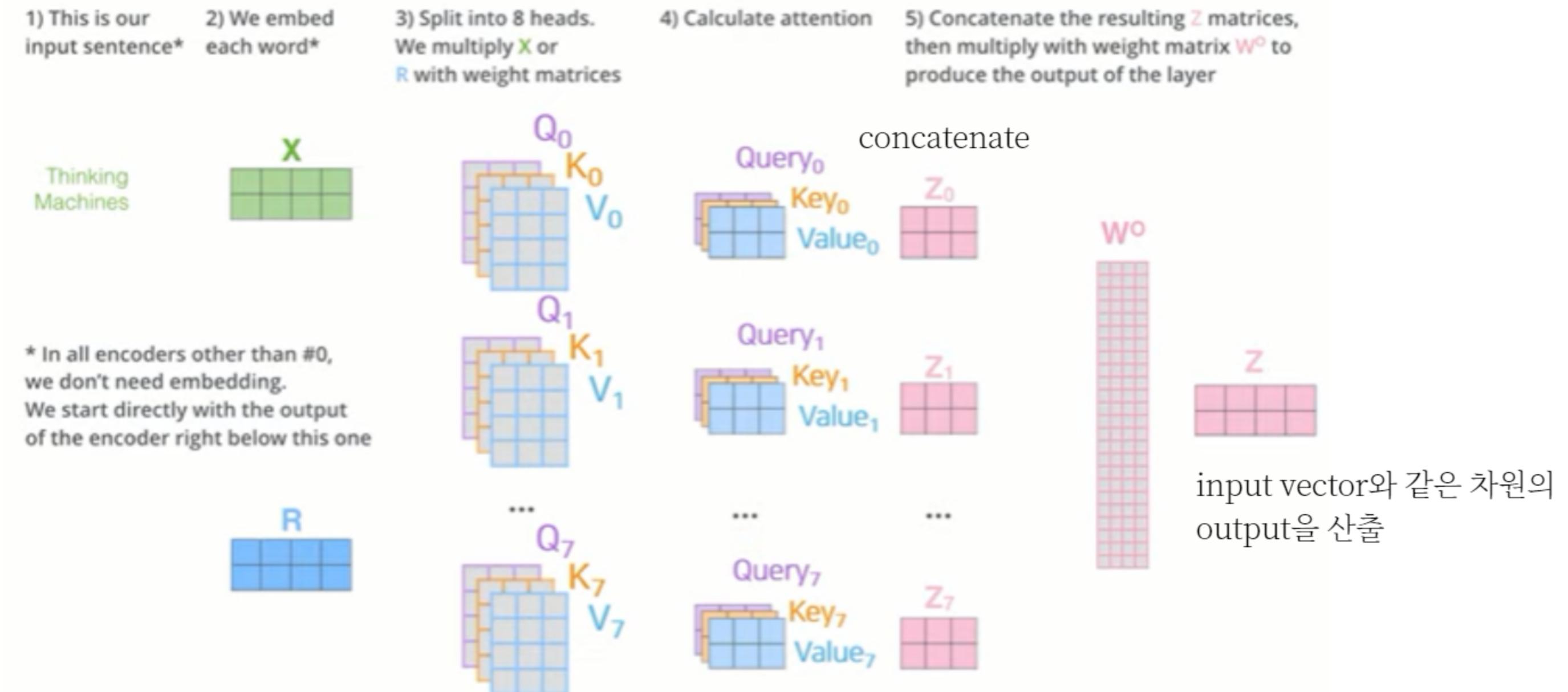
$$X = [x_1; \dots; x_T] \in \mathbb{R}^{T \times d}$$

$$XK \in \mathbb{R}^{T \times d}, XQ \in \mathbb{R}^{T \times d}, XV \in \mathbb{R}^{T \times d}$$

Transformer model

■ Multi-headed attention

- 한번에 여러 부분에 집중할 수 있도록 함
- self attention의 연산을 여러개의 head에서 수행하는 것으로 다양한 특징을 학습하도록 함



Transformer model

■ Multi-headed attention

- 각각의 head는 다른 부분에 집중하고, 다른 value vector를 산출한다.
- Multi-head attention을 수행하여도 결국 input과 같은 크기의 output을 산출한다.
- Multi-head attention을 수행하여도 총 계산 량은 동일하다.

→ 같은 양의 연산으로도 동시에 여러부분에 집중이 가능해진다.

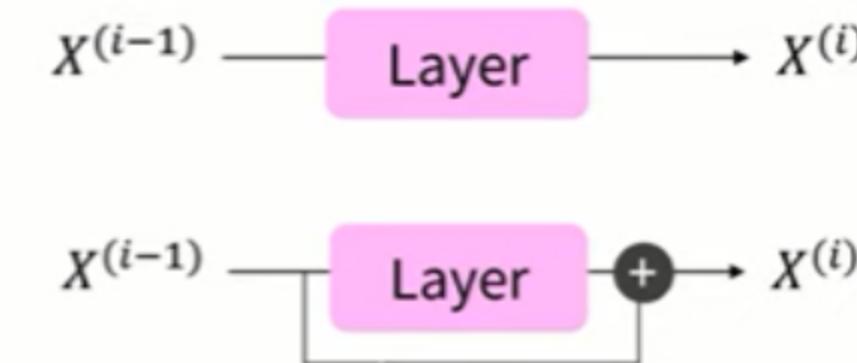


Transformer model

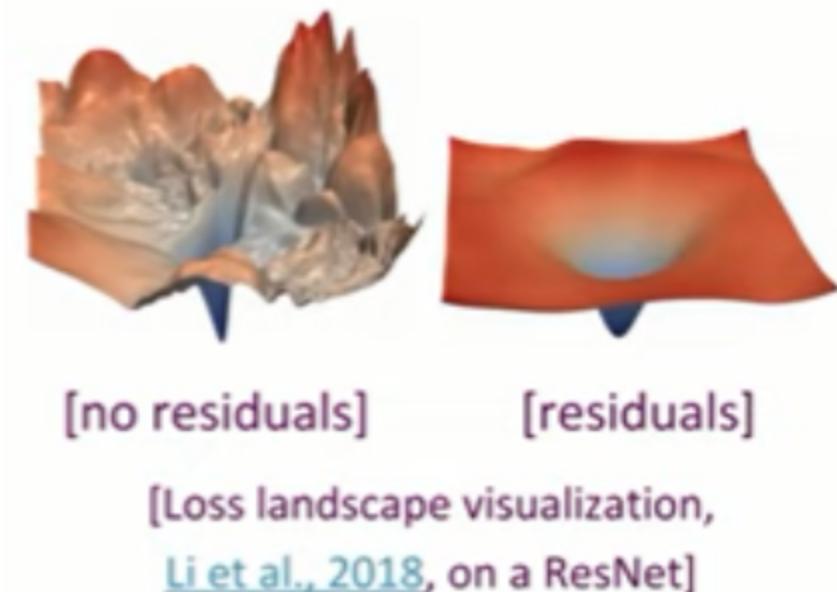
■ Residual connection

- ResNet에서 등장한 기법 : 자기 자신을 더해준다.

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



- 이전 (i-1) state와 얼마만큼 달라졌는지를 학습한다고 해석해볼 수도 있다.
- New $f(x) = \text{old } f(x) + x$ 라고 표현할 때 미분 결과 $f'(x) + 1$ 로 gradient가 아주 작아도 보존해 주는 역할로도 해석할 수 있다.
- 기울기를 smoothing 해주는 효과가 있어서 local minimum에 빠지지 않게 할 수 있다.



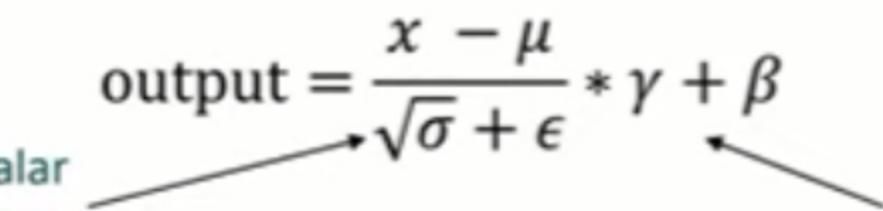
Transformer model

■ Layer Normalization

- 한 layer에서 하나의 input sample x 에 대해 모든 feature에 대한 평균과 분산을 구해 normalization 하는 것 gradient를 normalize 해주기 때문에 학습이 안정되고 속도가 빨라지는 효과가 있다.

$$\text{output} = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

Normalize by scalar mean and variance Modulate by learned elementwise gain and bias



Transformer model

■ Scaling the dot product

- dimension이 커지면 벡터의 dot product 결과도 커지는 경향이 있다.

$$\text{output}_\ell = \text{softmax}(XQ_\ell K_\ell^\top X^\top) * XV_\ell \longrightarrow \text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$

Transformer에서 Attention Score를 계산하는 과정에서 softmax 값이 몰리는 것은 특정 단어에만 강하게 연결이 되고 나머지와는 연결이 끊어지는 효과라고 볼 수 있다.

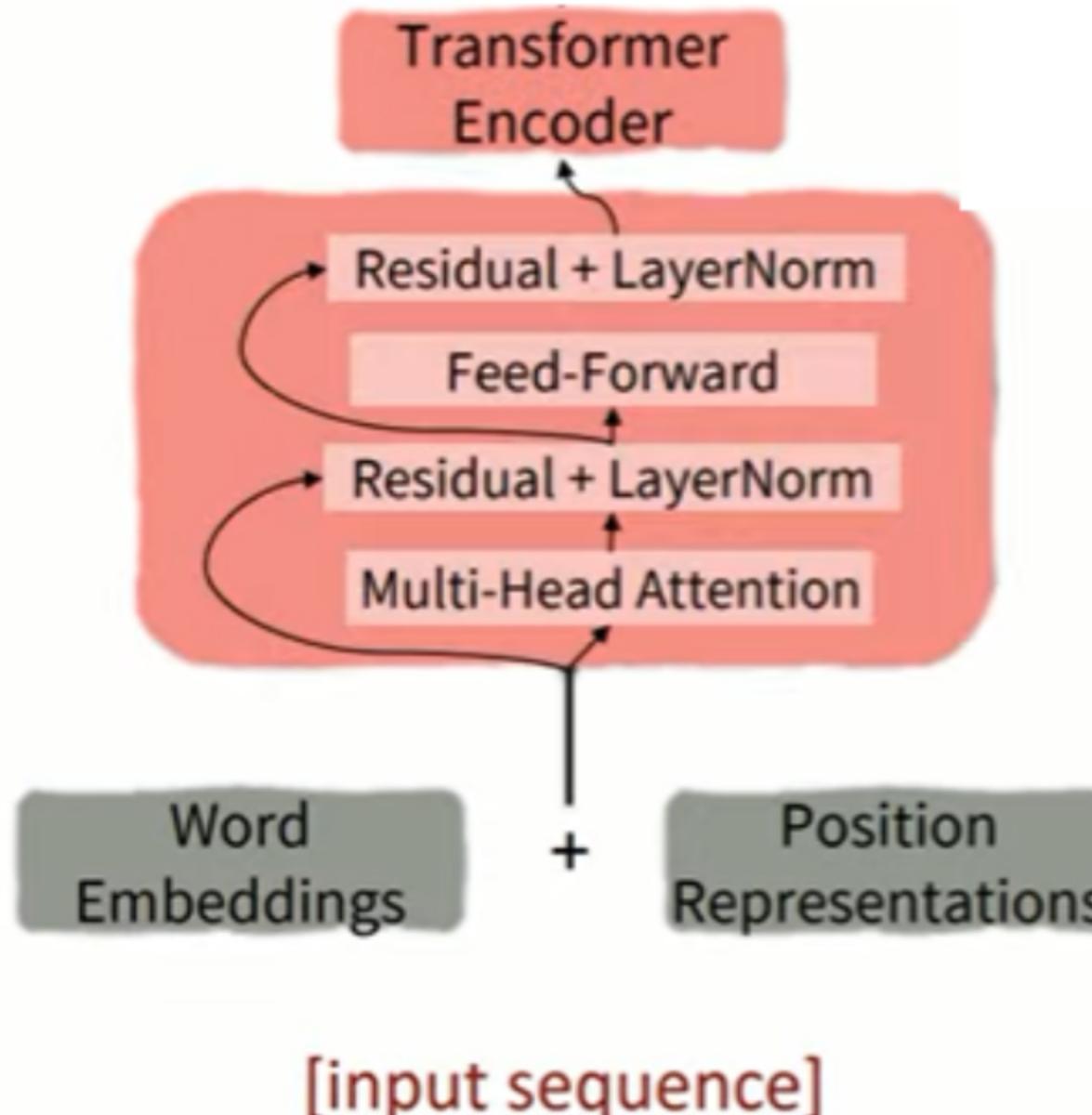
그러나 모든 시퀀스와의 gradient 전파가 잘 유지되기 위해서는 dot product의 결과가 너무 커지지 않도록 할 필요성이 있다.

→ 따라서 scaling을 진행한다.

→ Attention score을 좀 더 다양한 벡터들에게 분배되는 효과

Transformer model

■ Encoder Block

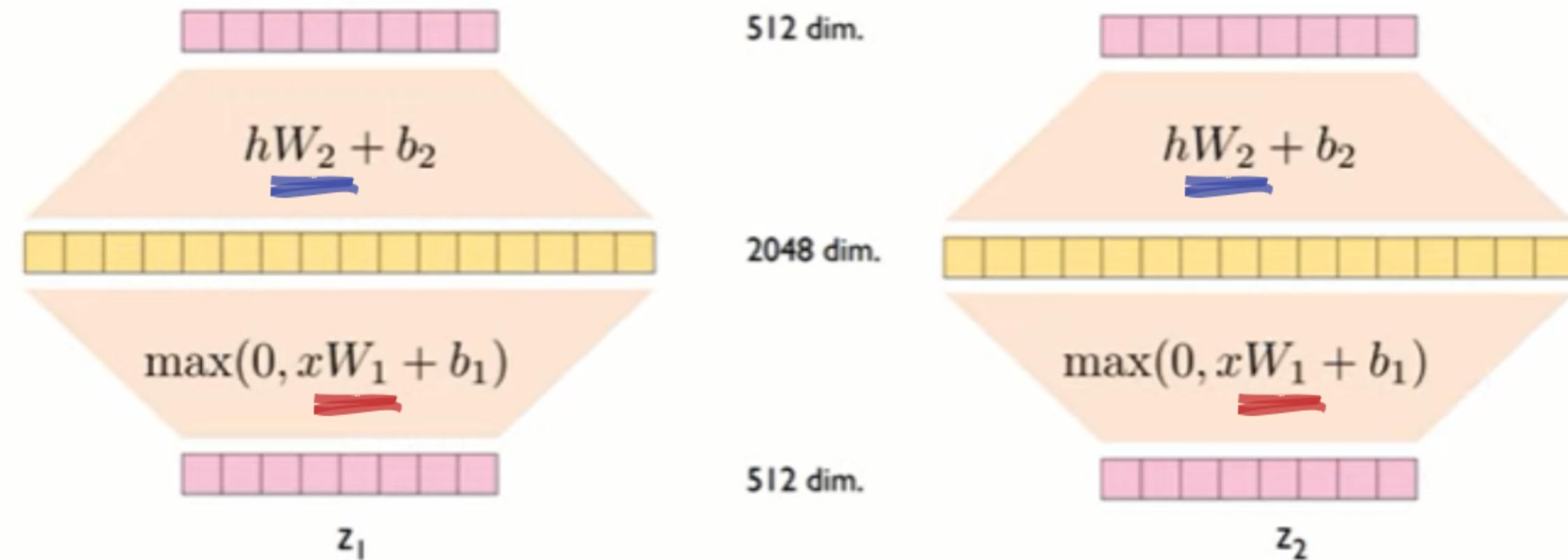


1. input word embedding이 들어오면 Position 벡터를 더해주어 위치 정보를 포함한 새로운 벡터가 되어 Multi-Head Attention으로 입력
2. Self Attention 과정을 거쳐 새로운 representation으로 변경
3. 자기 자신을 더해주는 Residual Connection을 수행한 뒤, Layer Normalization을 수행한 뒤 Feed Forward network으로 입력

Transformer model

■ Position-wise Feed Forward Networks

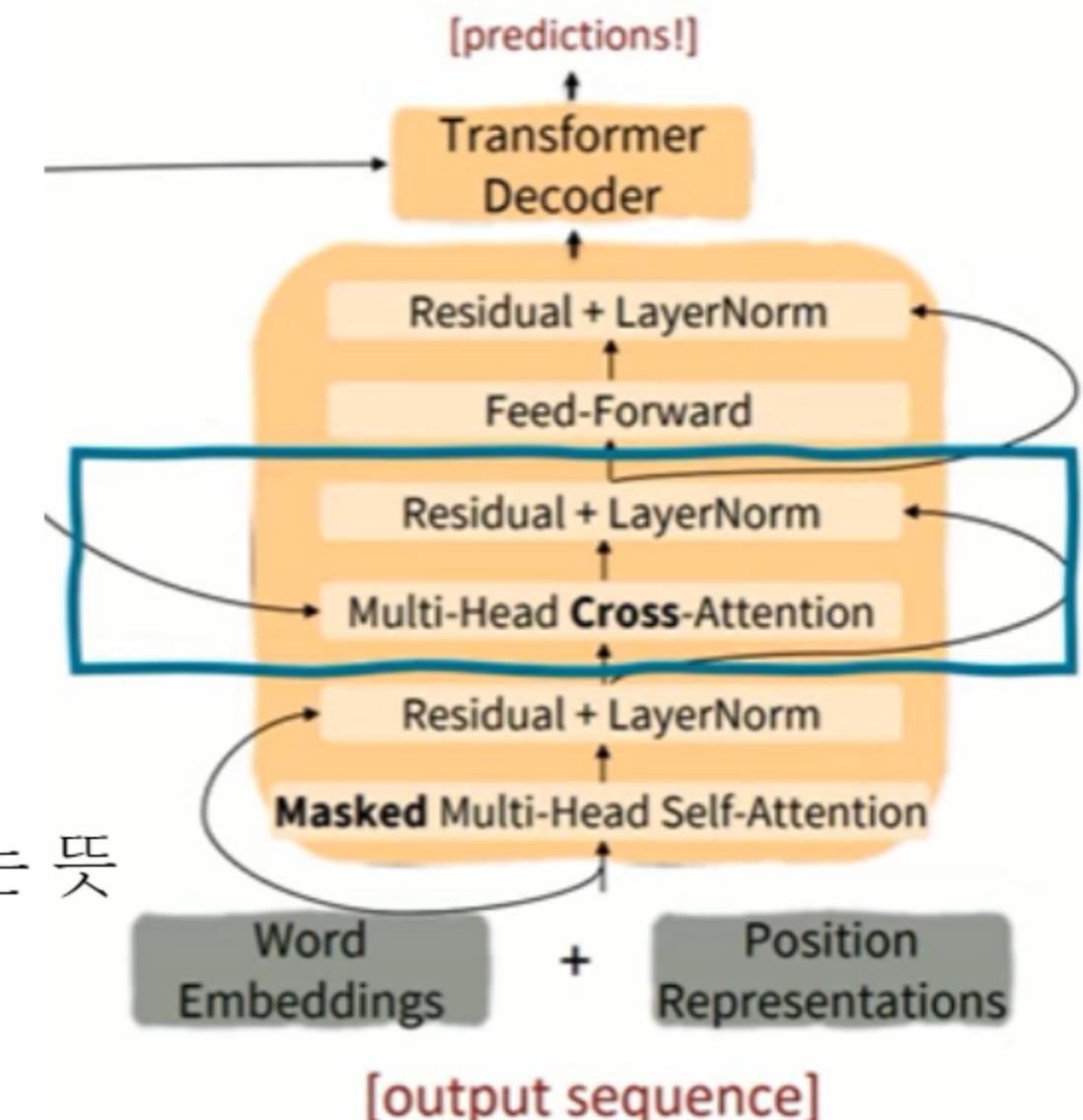
- Fully connected feed-forward network
- Sequence마다 독립적으로 적용된다.
- 같은 layer 안에서는 동일한 weight parameter를 공유한다.



Transformer model

■ Decoder Block

1. input word embedding이 들어오면 Position 벡터를 더해주어 위치 정보를 포함한 새로운 벡터가 되어 Multi-Head Attention으로 입력
2. Decoder에서 language Modeling을 수행할 때 미래 sequence 정보를 보면 안되기 때문에 Masked Multi-head attention을 수행
3. 자기 자신을 더해주는 Residual Connection을 수행한 뒤, Layer Normalization을 수행한 뒤 Multi-head cross Attention으로 입력
4. Multi-head cross Attention은 인코더와 디코더 사이를 오가는 attention이라는 뜻 cross attention은 Seq2Seq의 기본 Attention과 유사하다.
→ 디코더에서의 쿼리를 이용해 인코더를 탐색한다.

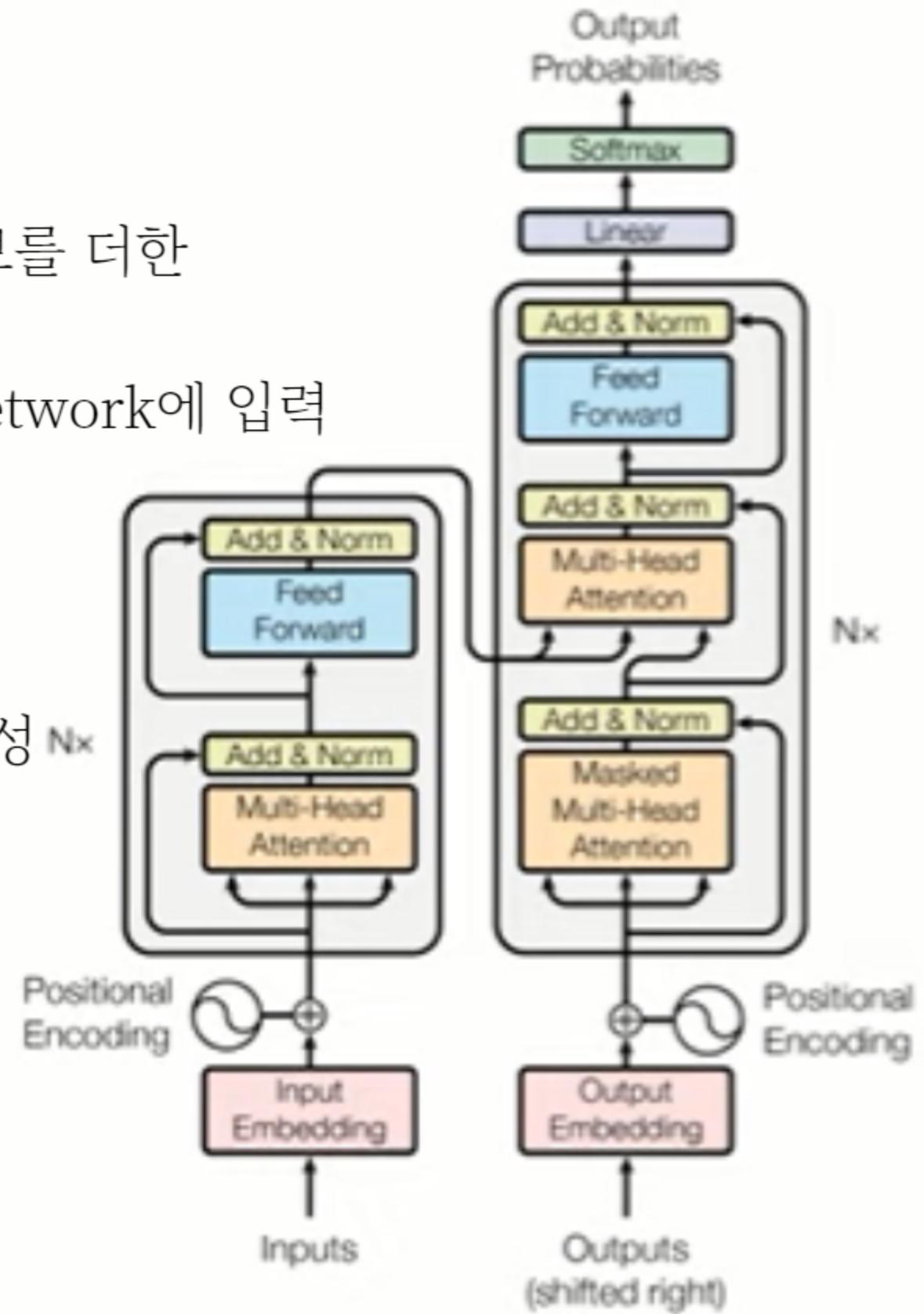


Transformer model

■ Transformer

- 인코더로 input vector가 들어오게 되면 embedding을 거쳐 position 정보를 더한 벡터를 Multi-Head Attention에 입력
- Residual Connection과 Layer Normalization을 거쳐 Feed Forward Network에 입력
- 이 결과를 다시 Residual Connection과 Layer Normalization을 거친다.
→ 최종 output은 다음 인코더나 마지막 인코더라면 디코더로 넘긴다.

- 디코더도 마찬가지로 input이 들어오면 위치 정보를 포함한 새로운 벡터 생성 Nx
- 미래 시퀀스를 볼 수 없도록 Masked Multi-Head Attention에 입력
- Residual Connection과 Layer Normalization을 거침
- Cross Multi-Head Attention에 입력(앞서 완료된 인코더와 통신)
- Residual Connection과 Layer Normalization을 거침
- Feed Forward Network를 거침
- Residual Connection과 Layer Normalization을 거침
- 다음 디코더로 넘기거나 마지막 디코더라면 최종 output을 산출



Drawbacks of Transformer

■ Transformer의 약점

1. Quadratic compute in self-attention

문장의 길이에 따라 계산량이 2차식으로 증가

$$O(T^2d)$$

2. Position representation에 개선의 여지가 존재

논문에서 제시한 Sinusoidal는 절대적인 위치를 표현하는 것이었고

이를 대체해서 상대적인 위치로 학습가능한 다른 방법들이 제시되고 상대적 위치를 현재 더 많이 사용하고 있다.

∴ Position Representation이 더 발전할 여지가 있다.

Drawbacks of Transformer

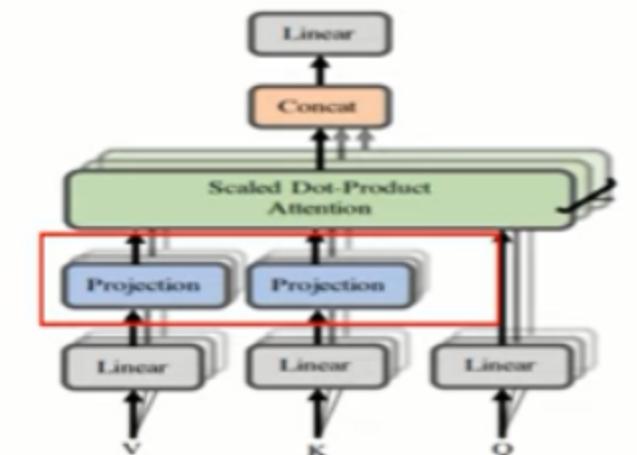
■ Improving on Quadratic compute in self attention Cost

문장의 길이에 따라 계산량이 2차식으로 증가하는 문제도 다양한 해결책이 연구되고 있다.

1. Linformer [Wang et al., 2020]

Sequence length의 차원을 낮춰 계산량을 줄이는 방법

key idea: projection을 통해 value와 key의 sequence length dimension을 낮춘다.



2. BigBird [Zaheer et al., 2021]

모든 pair 사이의 attention을 계산하지 않고 window, Global, Random을 적절히 조합한 만큼만 계산한다.

