ToBigs

Lecture 8

Self-Attention and Transformers

투빅스 18기
이화여자대학교 이선민

Lecture Plan

01 From RNN to attention-based NLP models

02 The Transformer model

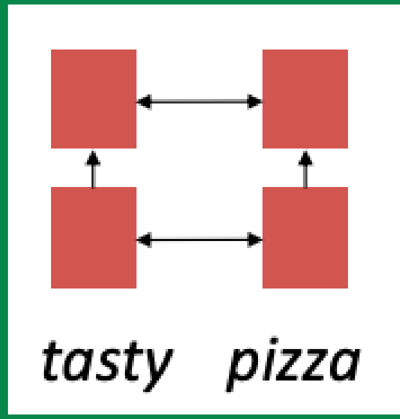
03 Great results with Transformers

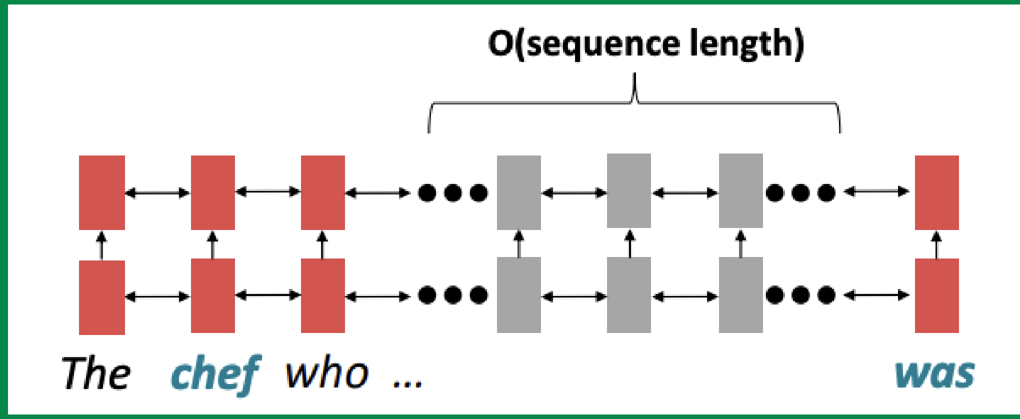
04 Drawbacks and variants of Transformers

ToBigs

Linear Interaction Distance

- Previous hidden state -> Future hidden state 방향으로 학습 진행
- 멀리 떨어진 단어 사이의 상호 작용이 어려움

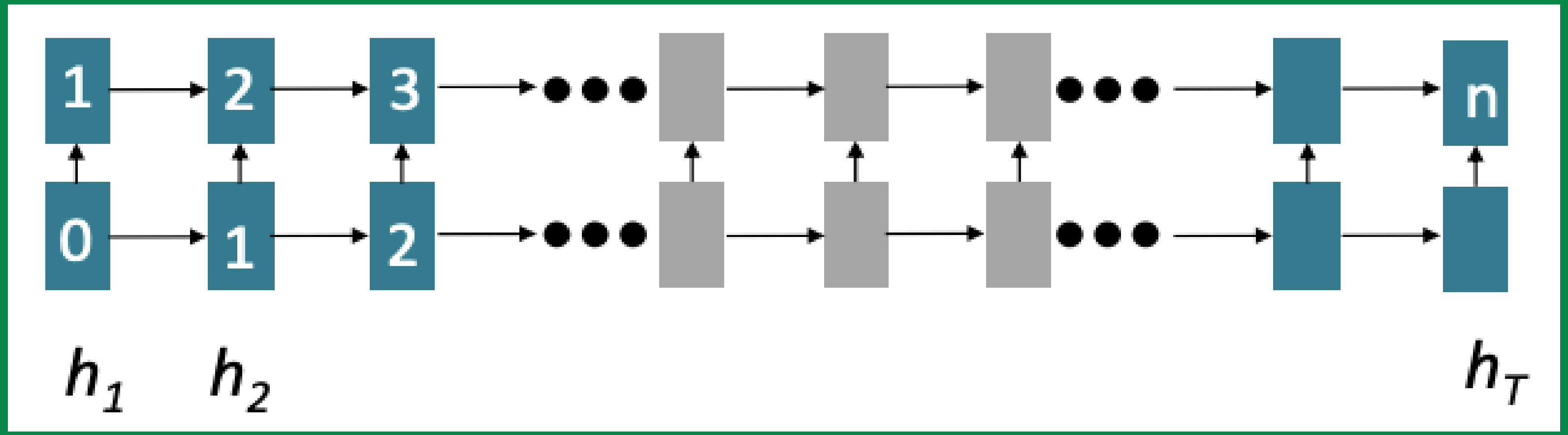




Vanishing Gradient problem !

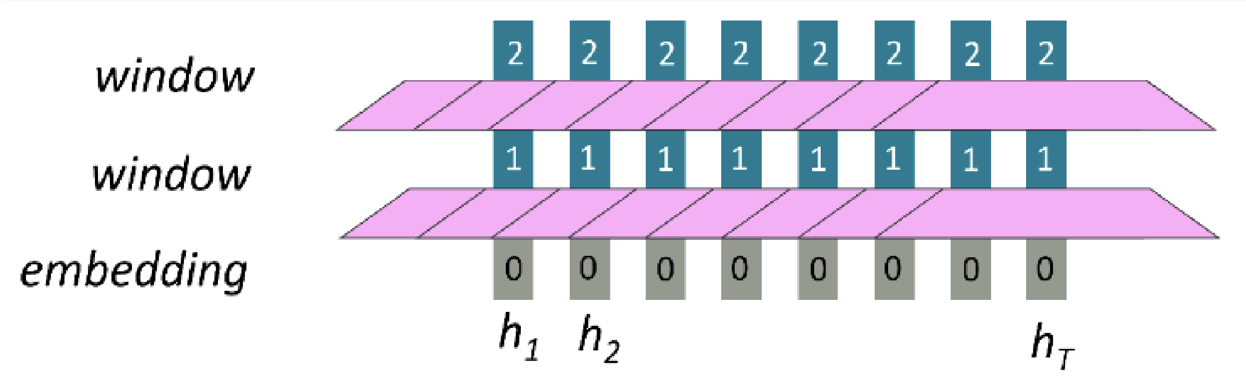
Lack of Parallelizability

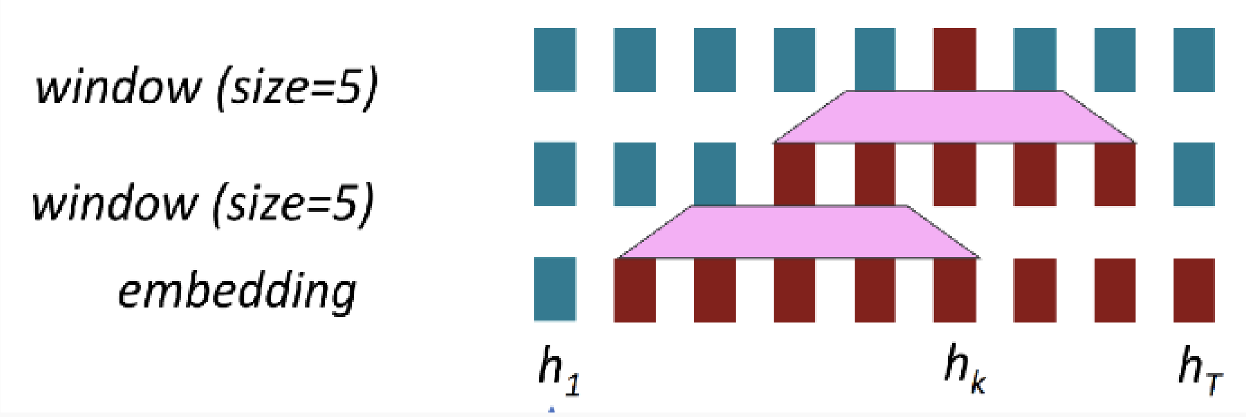
- Previous hidden state가 연산되어야 Next hidden state 연산 가능
- GPU를 사용한 병렬 연산 불가능

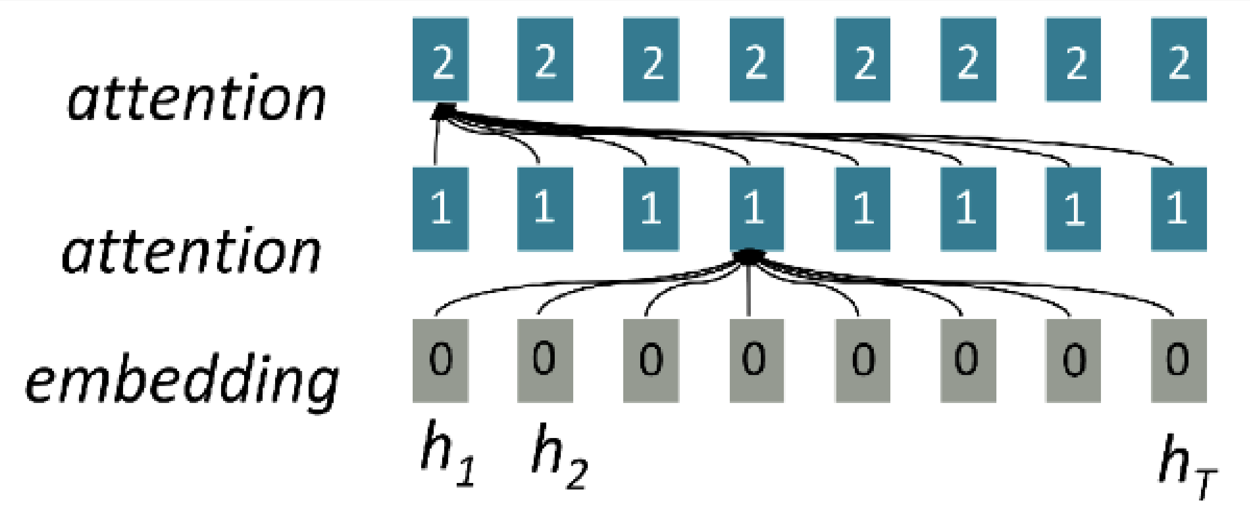


Other Building Blocks

RNN을 대신하여 사용할만한 다른 Building Block들을 살펴보자.







- Embedding layer : 각각 독립적으로 연산 가능
- Attention layer : Previous layer의 연산만 완료되면, Next layer의 모든 sequence state에 대해 계산 가능

Long Distance Dependency !

2 problems Solved !

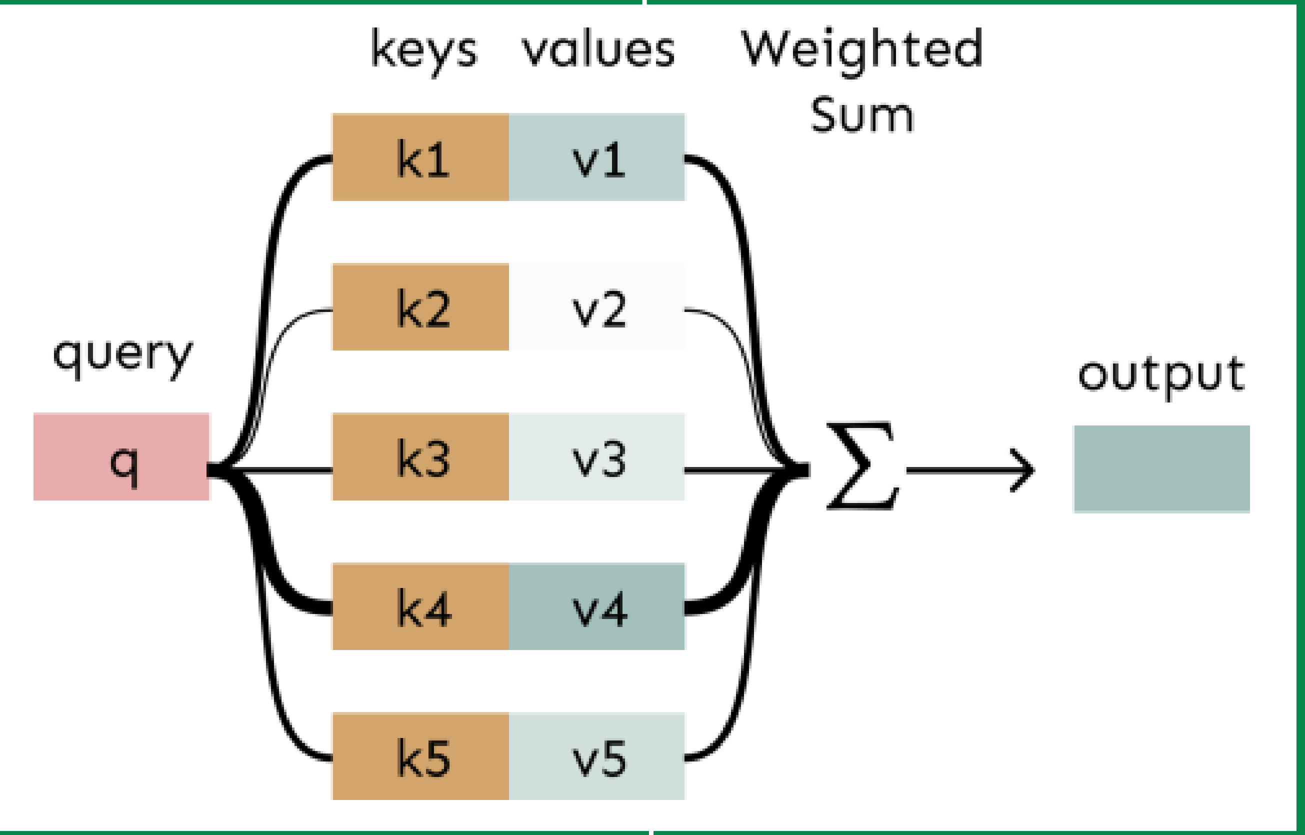
Word Window

Self-Attention

ToBigs

Self-Attention

- **Self-Attention**
- The Transformer
- Great results with Transformers
- Drawbacks and variants of Transformers



Query | **Key** | **Value**

Query	현재 보고있는 단어의 representation (다른 단어를 평가하는 기준)
Key	Query와 관련된 단어를 찾을 때, label처럼 활용되는 vector
Value	Query와 Key를 통해 탐색하여 실제로 사용할 값

LSTM layer처럼 쌓기만하면 해결될까?

참고 :
https://github.com/yookyungkho/DSBA_CS224N_2021/blob/main/slides/dsba_cs224n2021_lec09_gunhono.pdf

Problem of Self-Attention

3가지 문제 : 순서에 대한 정보 없음, 선형 결합만 존재함, Future sequence data 활용 가능함

Sequence Order

병렬 처리로 동시에 처리하므로, 순서에 대한 정보 없음

위치 벡터 사용 !

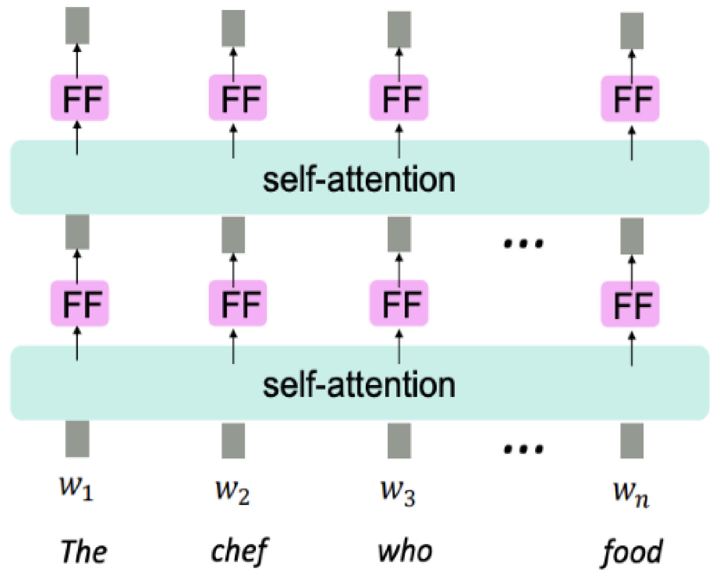
ex. Sinusodial

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

Nonlinearities

단순한 Weighted avarages의 선형 결합

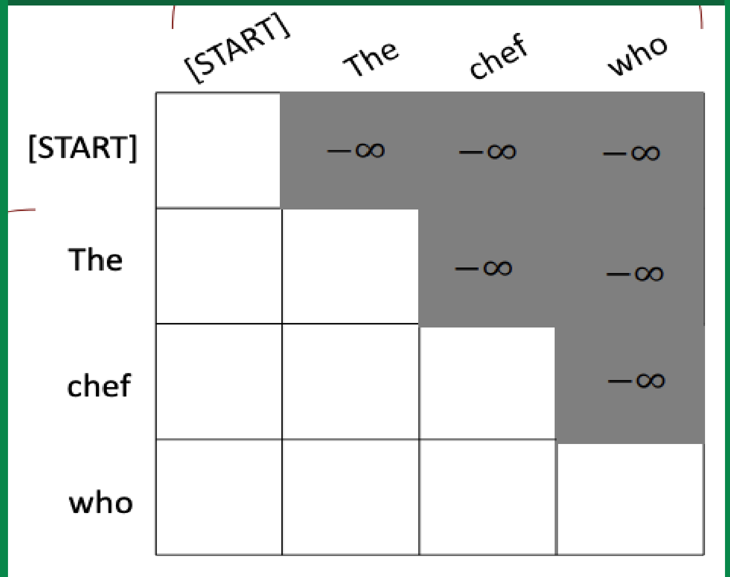
FF Network !



Future Sequence

Decoder에서 Future Sequence data를 볼 수 있음

Masking !



ToBigs

Self-Attention

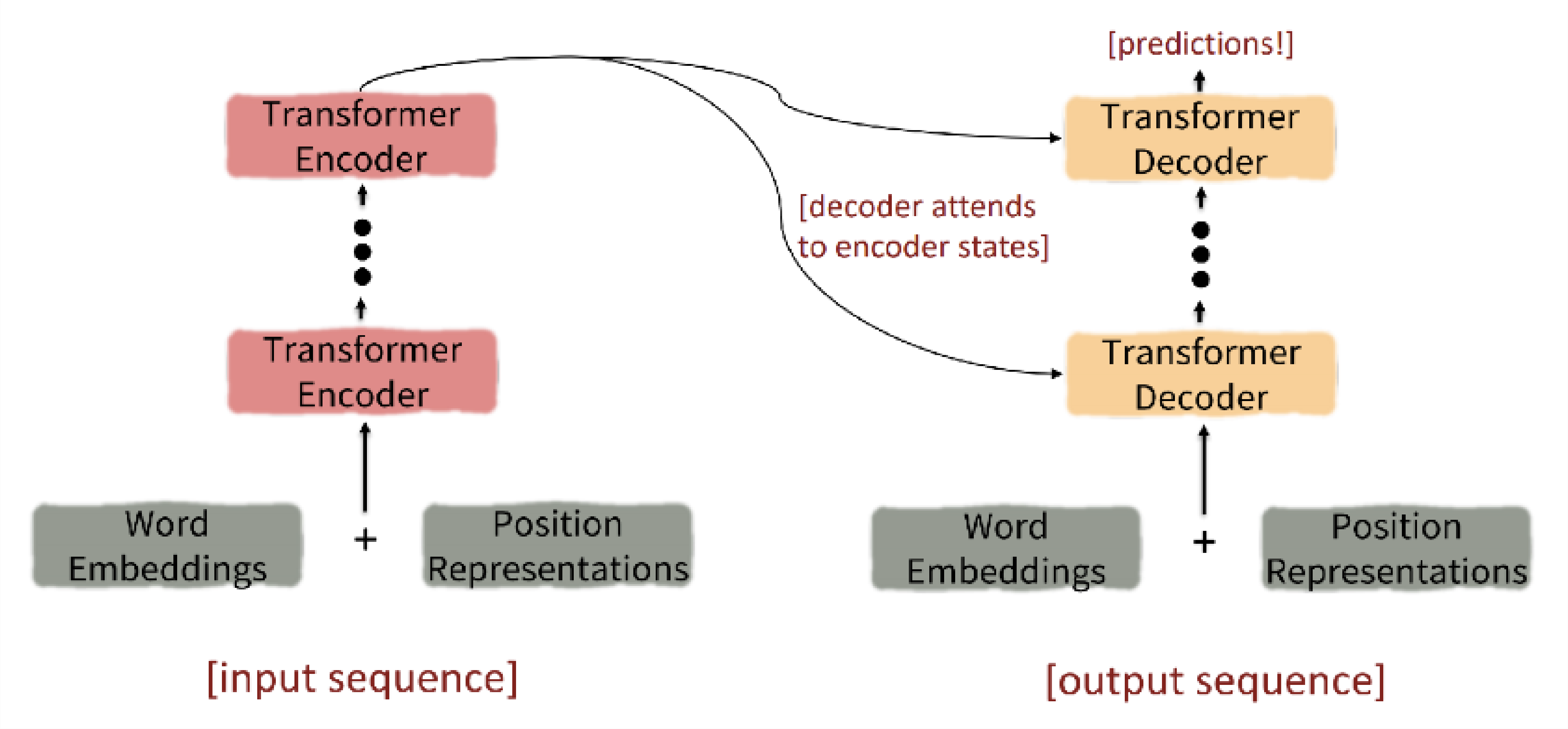
The Transformer

Great results with Transformers

Drawbacks and variants of Transformers

Transformer Model

Transformer Encoder & Decoder

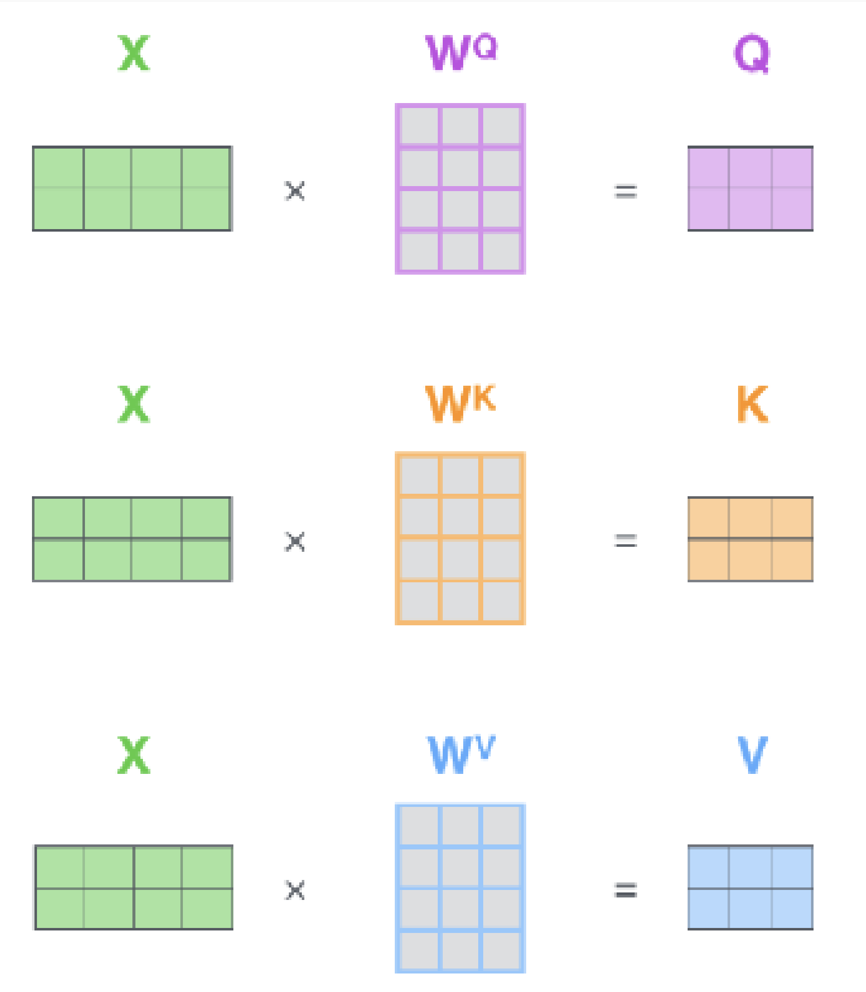


ToBigs

Transformer Query, Key, Value

Self-Attention과 달리 학습하는 대상의 Query, Key, Value를 행렬로 나타낸다.

- Self-Attention
- **The Transformer**
- Great results with Transformers
- Drawbacks and variants of Transformers



행렬 Q, K, V
= 우리가 찾아야하는 미지수

- Input vector가 d dimension을 가질 때, 그를 결합한 행렬을 X라 하자. $X = [x_1; \dots; x_T] \in \mathbb{R}^{T\times d}$
- Query, Key, Value를 계산하기 위해 행렬 X와 행렬 K, Q, V를 dot product한다. $XK \in \mathbb{R}^{T\times d}, XQ \in \mathbb{R}^{T\times d}, XV \in \mathbb{R}^{T\times d}$
- Attention score 계산을 위해 다음의 행렬 연산을 수행한다. $XQ(XK)^\top$

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

- Input vector가 d dimension을 가질 때, 그를 결합한 행렬을 X라 하자. $X = [x_1; \dots; x_T] \in \mathbb{R}^{T\times d}$
- Query, Key, Value를 계산하기 위해 행렬 X와 행렬 K, Q, V를 dot product한다. $XK \in \mathbb{R}^{T\times d}, XQ \in \mathbb{R}^{T\times d}, XV \in \mathbb{R}^{T\times d}$
- Attention score 계산을 위해 다음의 행렬 연산을 수행한다. $XQ(XK)^\top$

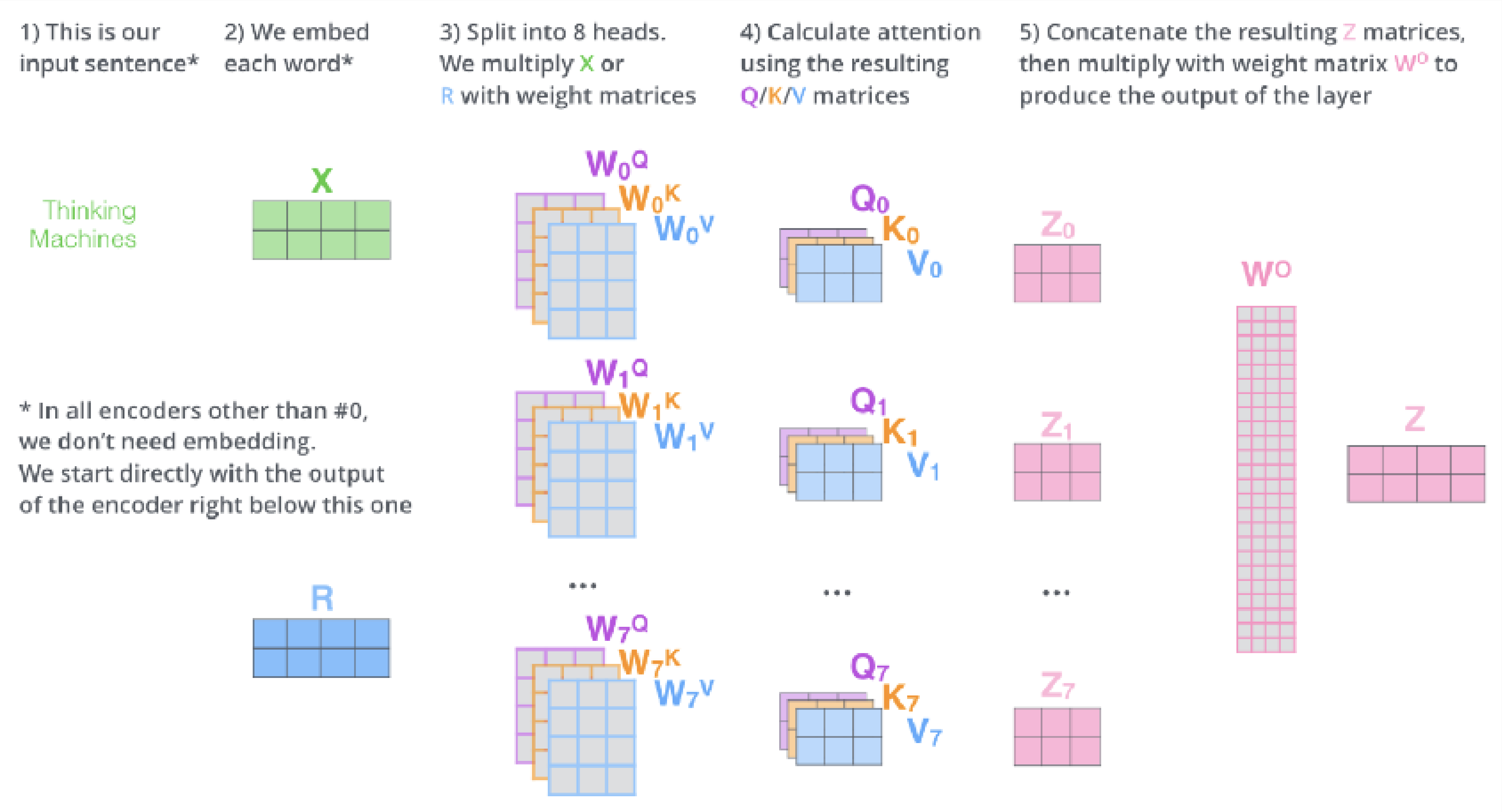
$$XQ \quad K^\top X^\top = XQK^\top X^\top \in \mathbb{R}^{T\times T}$$

- Attention score 결과를 softmax를 통해 가중치를 얻고 그를 가중합하여 outpu을 얻는다.

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

Transformer Multi-head Attention

Look at Different things, and construct value vectors differently !



Transformer Multi-head Attention

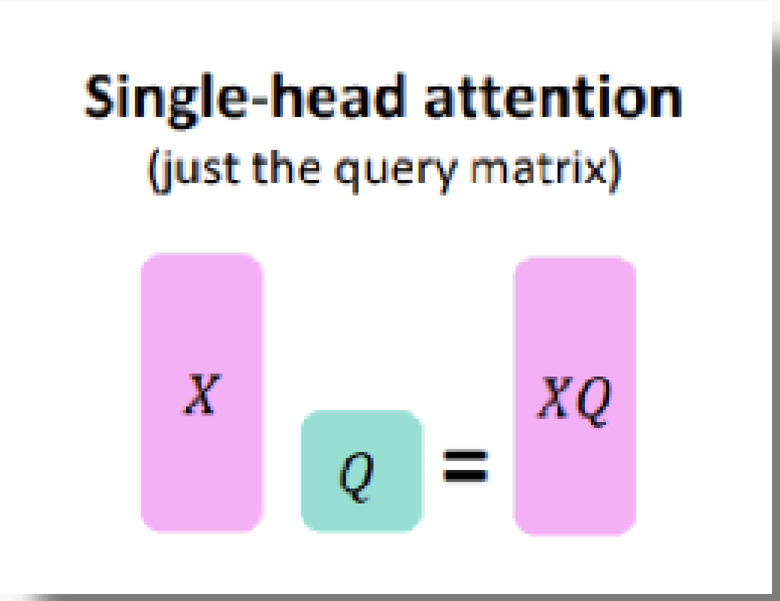
Single-head Attention과 계산 과정이 동일하다.

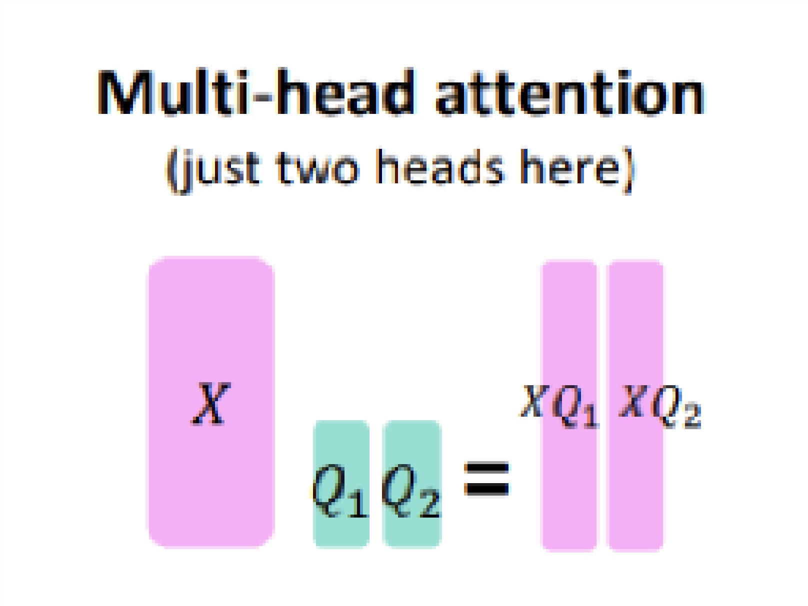
$$Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$$

h 는 head의 개수, $\ell = 1, 2, \cdots, h$

$$\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell, \text{ where } \text{output}_\ell \in \mathbb{R}^{d/h}$$

$$\text{output} = Y[\text{output}_1; \ldots; \text{output}_h], \text{ where } Y \in \mathbb{R}^{d \times d}$$





독립적으로 동시에 수행되므로, **Self-Attention과 계산량이 동일**하다.

ToBigs

Transformer Multi-head Attention

W^o matrix를 곱해주는 이유는 무엇일까 ?

Question ?

Multi-head Attention을 할 때, 각 헤드들의 output을 concat 한 후 W^o matrix를 곱해줍니다.
각 layer의 input output 차원을 동일하게 유지하기 위해 W^o matrix를 곱해준다고 이해를 했습니다.
본 논문의 구현상으로는 512차원을 8개의 헤드로 나눠 각각의 헤드가 64차원을 가지고, concat 하면 d_model의 차원과 같아지는데 W^o matrix를 곱해주는 이유는 무엇인가요?

질문을 정리하자면, **W^o matrix의 역할**은 무엇인가요?

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, \ldots, \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Correct Answer ?

if $T = 10,\ d = 512,\ h = 8$

input : $X\ (R^{T \times d})$

$R^{10 \times 512}$

self - attention : $R^{T \times d}$

$R^{10 \times 512}$

multi-head attention : $R^{T \times \frac{d}{h}}$ $\xrightarrow{\text{concat}}$ $R^{(T \times \frac{d}{h}) \times h}$

$R^{10 \times \frac{512}{8}}$ $\quad$ $R^{(10 \times \frac{512}{8}) \times 8} = R^{10 \times 512}$

W^o : $R^{(h \times \frac{d}{h}) \times d}$

목표 차원수 = 64

$R^{(8 \times \frac{512}{8}) \times 64} = R^{512 \times 64}$

최종 Output : multi-head attention output $\times$ W^o $= R^{10 \times 64}$

$R^{10 \times 512}$ $\quad$ $R^{512 \times 64}$

W^o matrix는 목표 차원수로 맞춰주는 역할을 한다.

Transformer Modeling Improvements

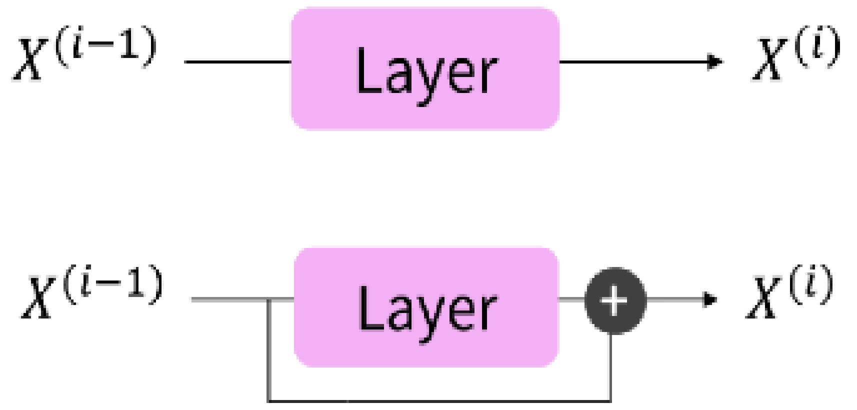
Residual Connection

자기 자신을 더해준다.

- 미분 결과값이 너무 작아 gradient propagation이 원할하지 않은 경우, gradient를 보정함
- 기울기 Smoothing 효과 -> local minimum에 빠지지 않도록 함

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

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



Inductive Bias !

이전 state보다 얼마나 달라졌는지 학습한다.

Transformer Modeling Improvements

Layer Nomalization

Gradient를 nomalize 한다.

- 한 layer에서 하나의 input sample x에 대해 모든 feature에 대한 평균과 분산을 구해 nomalization 함

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

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

Transformer Modeling Improvements

Scaling the Dot product

Dot product의 결과가 너무 커지지 않도록 유지한다.

- 차원 수가 늘어날수록, dot product의 결과가 커짐 = softmax 결과가 치중됨
- 모든 sequence에서 gradient propagating이 잘 되도록 유지해줌
- Attention score을 좀 더 다양한 vector에 분배함

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

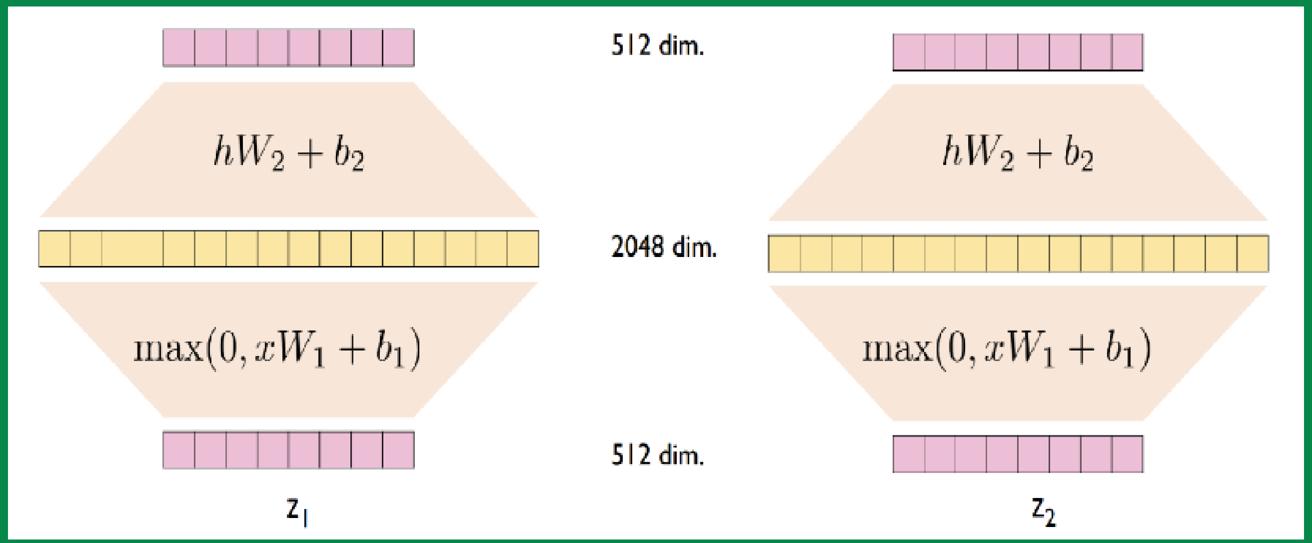
KEY (K)

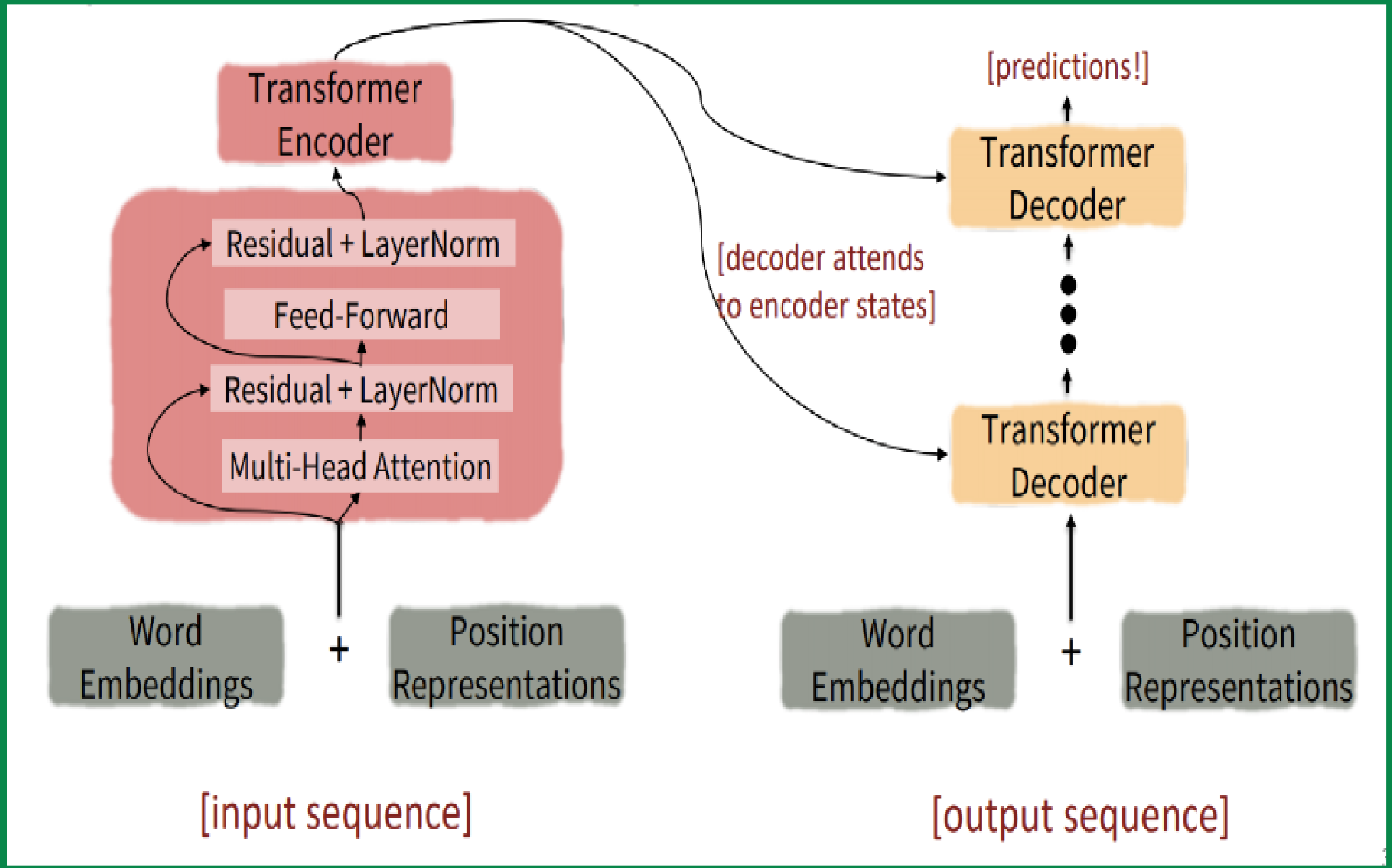
$softmax(\frac{QK^T}{\sqrt{d_k}})$	I	kicked	the	ball
QUERY (Q) I	0.66 0.91	0.16 0.06	0.07 0.01	0.11 0.02
kicked	0.24 0.18	0.47 0.67	0.08 0.03	0.21 0.12
the	0.19 0.13	0.11 0.04	0.37 0.48	0.33 0.35
ball	0.16 0.11	0.20 0.14	0.28 0.28	0.36 0.47

ToBigs

- Self-Attention
- **The Transformer**
- Great results with Transformers
- Drawbacks and variants of Transformers

Encoder

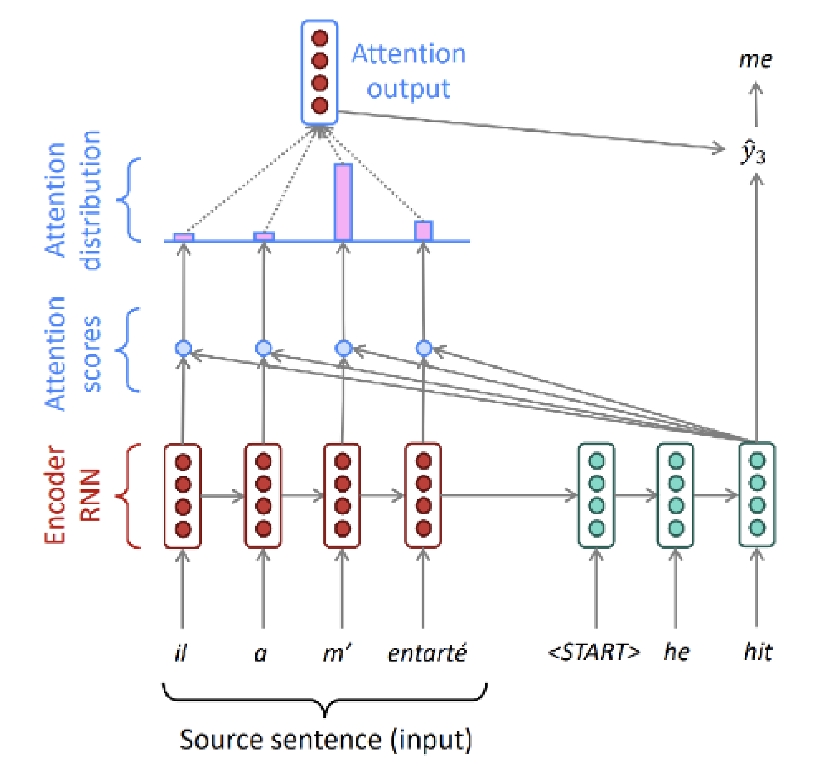




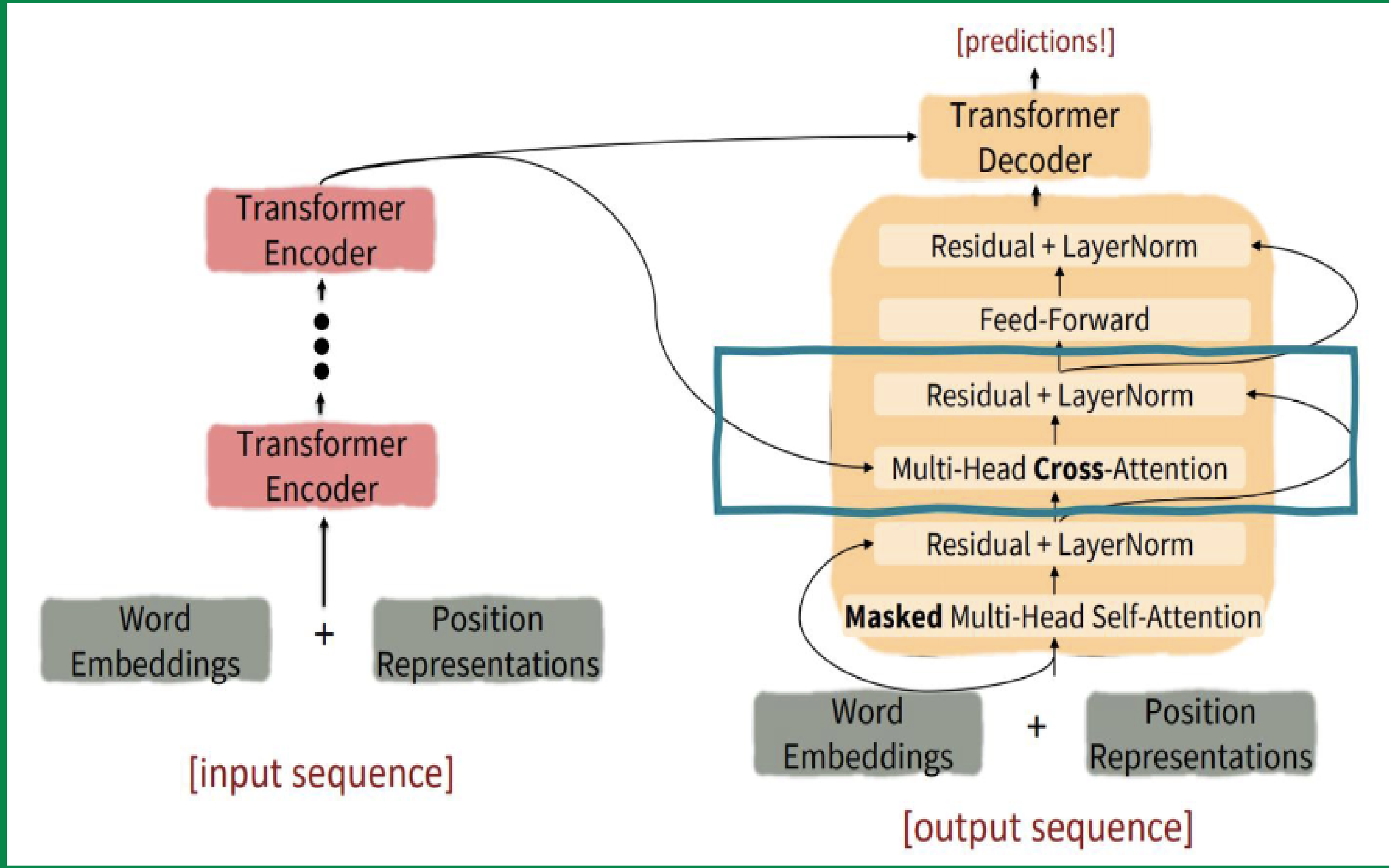
Input	Word Embedding + Position Representation
Multi head Attention	새로운 representation으로 표현됨
Residual + Layer Nomalization	Transformer의 성능을 높여줌
Feed Forward Network	Sequence마다 독립적으로 적용됨
Residual + Layer Nomalization	Transformer의 성능을 높여줌
Transformer Encoder	여러 개의 Encoder를 거친 후, Decoder로 들어감

ToBigs

Decoder



- Self-Attention
- **The Transformer**
- Great results with Transformers
- Drawbacks and variants of Transformers



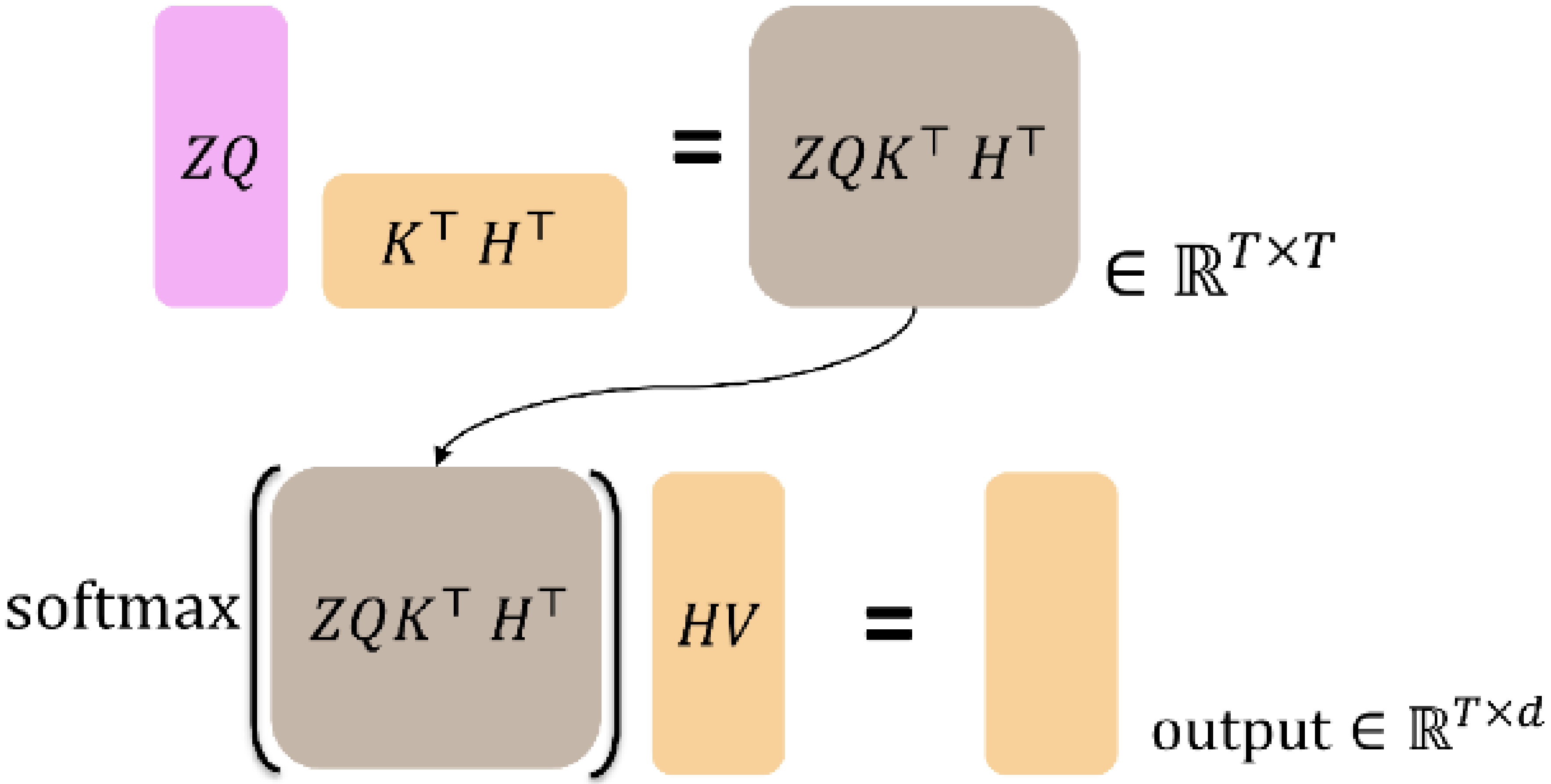
Input	Word Embedding + Position Representation
Multi head Attention	Masked Multi head Attention
Residual + Layer Nomalization	Transformer의 성능을 높여줌
Multi head Attention	Multi head Cross Attention
Residual + Layer Nomalization	Transformer의 성능을 높여줌
Transformer Encoder	여러 개의 Encoder를 거친 후, Decoder로 들어감

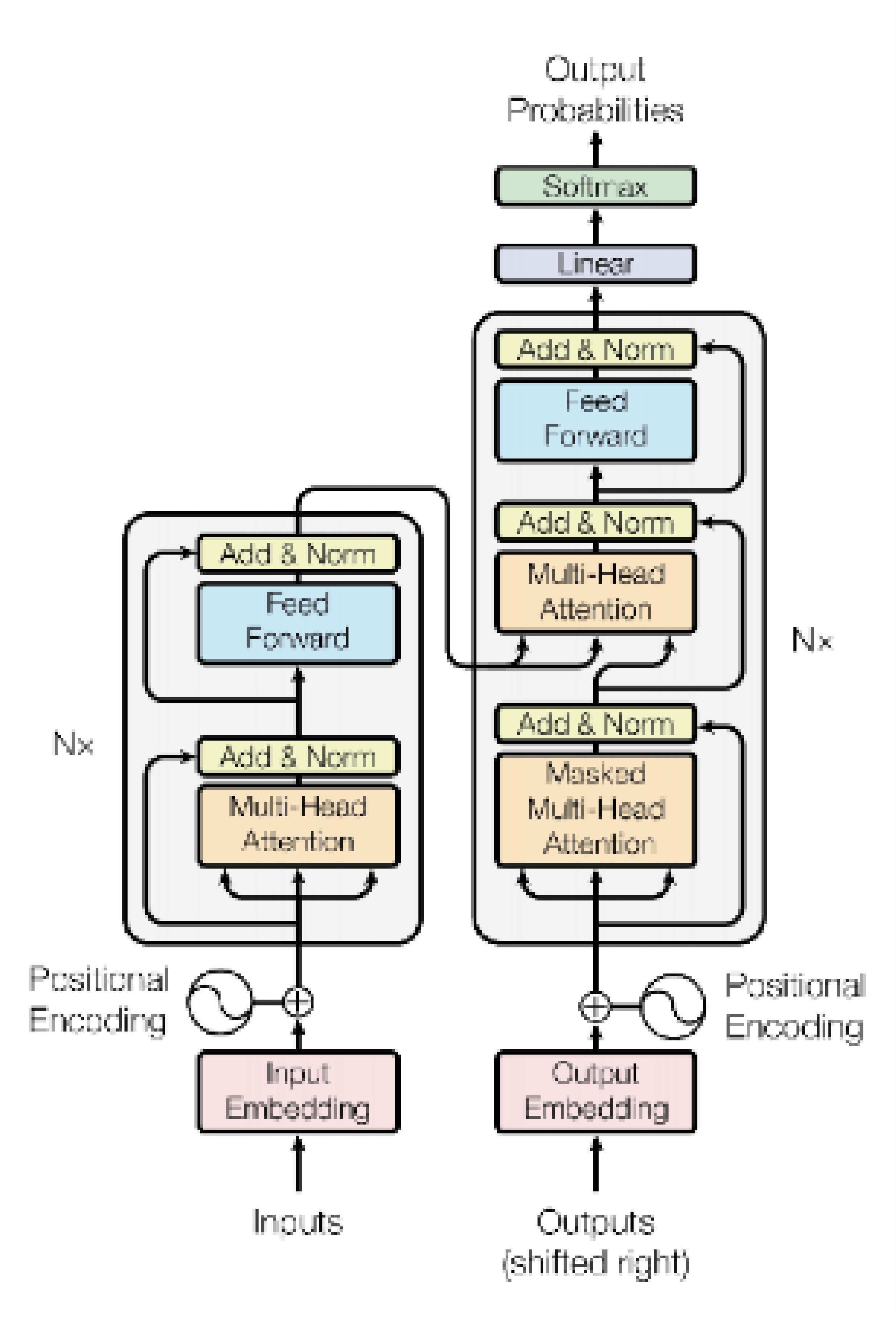
Encoder & Decoder

- h: encoder의 output vector
  z: decoder의 input vector

Let H $= [h_1; \dots ; h_T] \in \mathbb{R}^{T\times d}$
Let Z $= [z_1; \dots ; z_T] \in \mathbb{R}^{T\times d}$

- Decoder에서 현재 처리하는 단어의 Query를 가져오고, Encoder의 Key로 탐색한 결과를 Encoder의 Value로 가중 합한다.
- Attention score 계산을 위해 다음의 행렬 연산을 수행한다. $ZQ(HK)^{\top}$
- Attention score 결과를 softmax를 통해 가중치를 얻고 그를 가중합하여 output을 얻는다.





Transformer Summary

seq2seq

Self-Attention을 이용하여 문장 내 다른 단어들로부터 힌트를 받아 현재 단어 Encoding

Positional Encoding

Positional representation을 추가하여 새로운 vector 생성

Encoder-Decoder

- 동일한 개수 사용
- Multi head Attention 사용
- FF network, Residual connection, Layer nomalization 사용

Transformer Ability

다양한 분야에서 좋은 성능을 보이고 있다.

Machine translation

기존 SOTA 모델보다
훨씬 높은 성능을 보임

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$

Nonlinearities

기준보다 훨씬 낮은
Perplexity
= 높은 성능

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

Future Sequence

현재 1위
Microsoft
alexander v-team

Transformer 사용 !

Rank	Name	Model
1	Microsoft Alexander v-team	Turing ULR v5

GLUE Benchmark
https://gluebenchmark.com/leaderboard

Perplexity

언어 모델을 평가하기
위한 평가 지표

Drawbacks of Transformer

몇 가지 한계점이 존재한다.

Quadratic Computation

Sequence length가
증가함에 따라
계산량이 2차식으로 증가

- 짧은 문장 = not that big a deal !
- 긴 문장 (Document) = T^2이 기하급수적으로 증가하여 computing에 부담됨

$$O(T^2 d)$$

Position Representation

절대적 위치만 나타내는
Sinosoidual 사용

- Relative linear position attention : 상대적인 위치
- Dependency syntax-based position : 구조적인 정보 고려
- Rotary Position Embedding : 매 layer 마다 위치 정보 고려

Rotary Position Embedding

https://arxiv.org/abs/2104.09864

Variants of Transformer

변형을 통해 한계점을 해결할 수 있다.

Linformer

Sequence length의 차원을
낮춰 계산량을 줄임

- Projection을 통해 Key, Value의 Sequence length dimension을 낮춤
- Inference time을 보면 높음

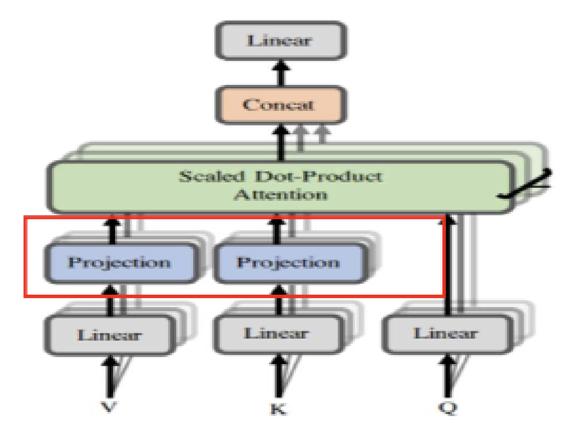
BigBird

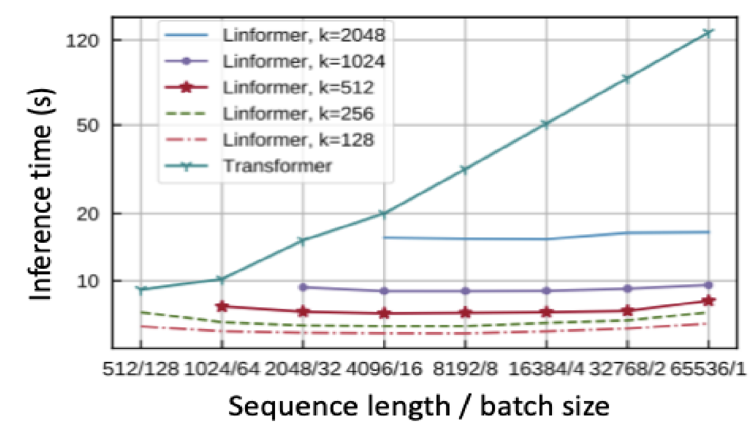
최적의 조합만 계산함

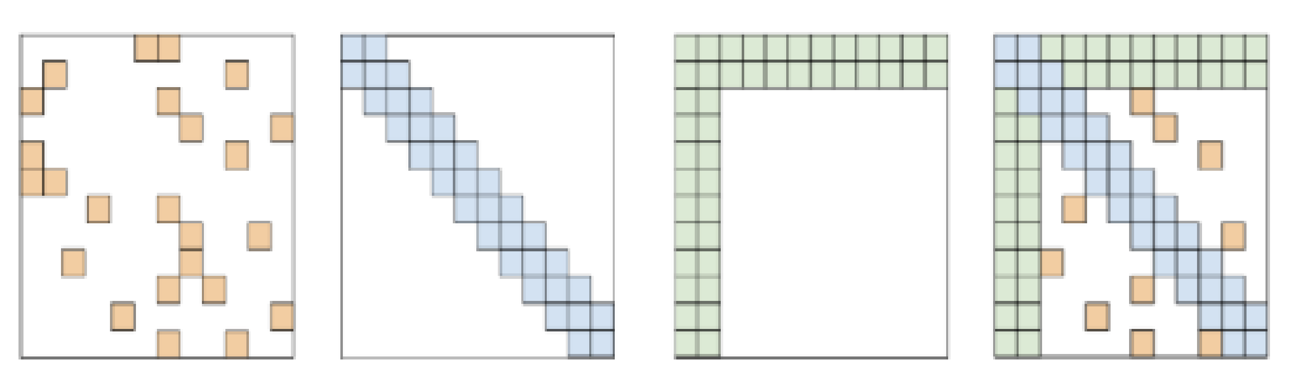
- 모든 pair의 Attention을 계산하지 않음
- 최적의 조합만 계산하여, 계산량 줄임

Inference time

딥러닝 모델이 커지면서,
빠른 Inference time이
매우 중요해짐







(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

ToBigs

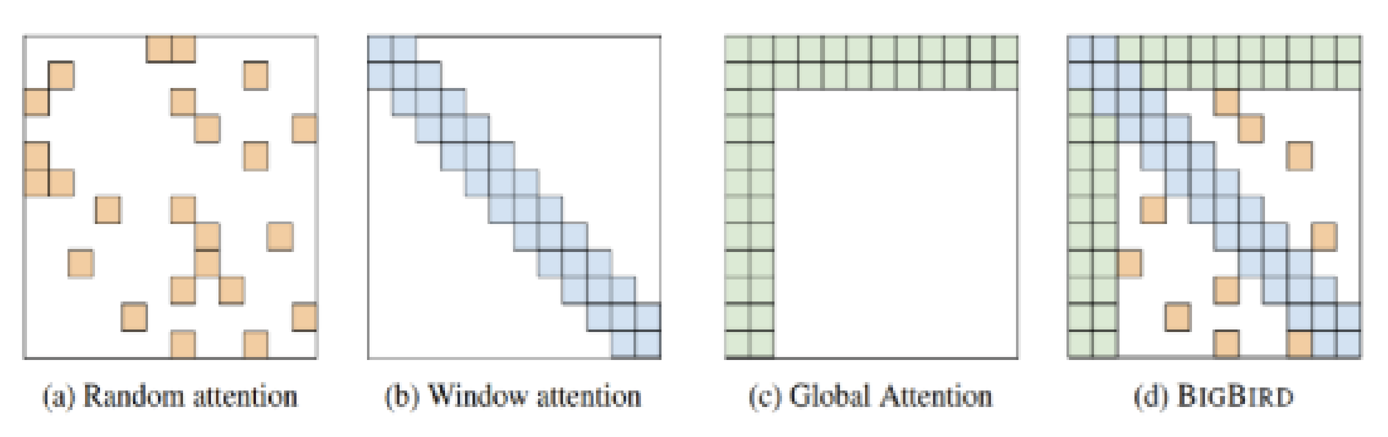
- Self-Attention
- The Transformer
- Great results with Transformers
- **Drawbacks and variants of Transformers**

Variants of Transformer

변형을 통해 한계점을 해결할 수 있다.

Question ?

어텐션에 대한 그림으로 4종류가 있는데,
이것들이 의미하는 바가 무엇인지
궁금합니다.



BigBird

- **(a) Random attention** : Query와 r개의 random keys 간의 attention
- **(b) Window attention** : Query 양옆 w개의 keys와의 attention
- **(c) Global attention** : Query와 g개의 global token들과의 attention
- **(d) BigBird** : 위 3개를 모두 합친 방식 (가장 성능이 좋게 나옴)

ToBigs

감사합니다.