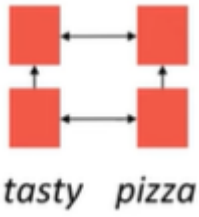
Lecture 9 - Self Attention and Transformers

▼ From recurrence (RNN) to attention-based NLP models

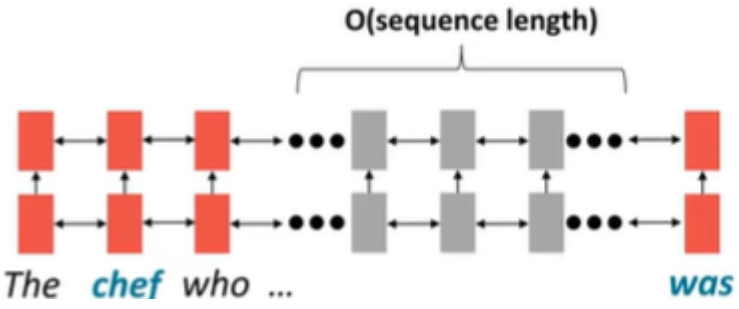
문제 1 Linear interaction distance

⌛ 멀리 떨어져 있는 단어들간의 상호작용이 어렵다.

> RNN → left to right으로 왼쪽부터 오른쪽으로 순차적으로 연산



- 위 그림처럼 tasty와 pizza는 단어끼리 가깝게 있기 때문에 상호작용이 용이 (워드 임베딩에도 잘 적용된다.)

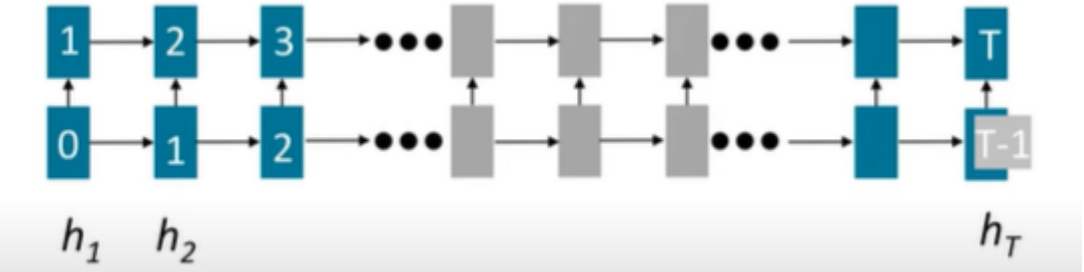


- 하지만 위 그림에서 chef와 was는 관계대명사 절에 의해 거리가 매우 멀리 떨어져 있음
- RNN은 멀리 떨어져 있는 단어 쌍이 interaction하기 위해서는 O(sequence length)만큼 소요되고, Gradient 문제로 멀리 떨어진 dependency는 학습하기가 어려움

문제 2 Lack of parallelizability - 병렬 연산 불가

> Future hidden state를 처리하려면 반드시 그 직전의 hidden state의 계산이 완료되어야 한다.

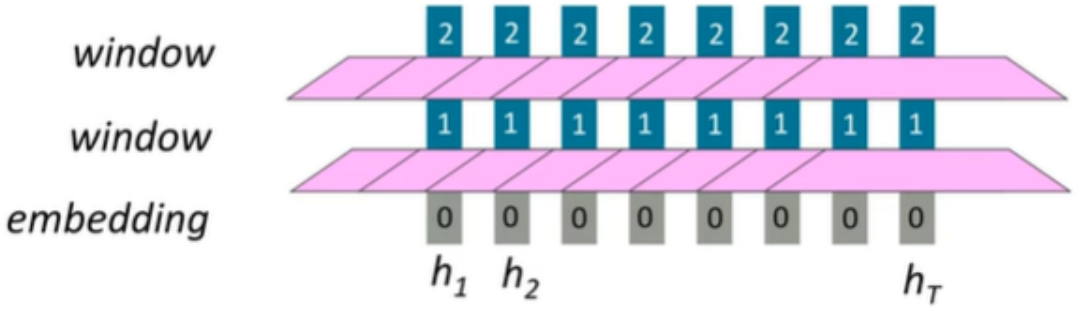
⌛ 순서대로 처리되는 RNN의 연산 특성상 GPU를 활용해 시간 차원에 대한 병렬 불가능하다. → 모델이 복잡해질수록 불리한 부분



- 마지막 hidden state에는 O(sequence length)만큼 연산이 이뤄져야 처리 가능하다.

How about word windows?

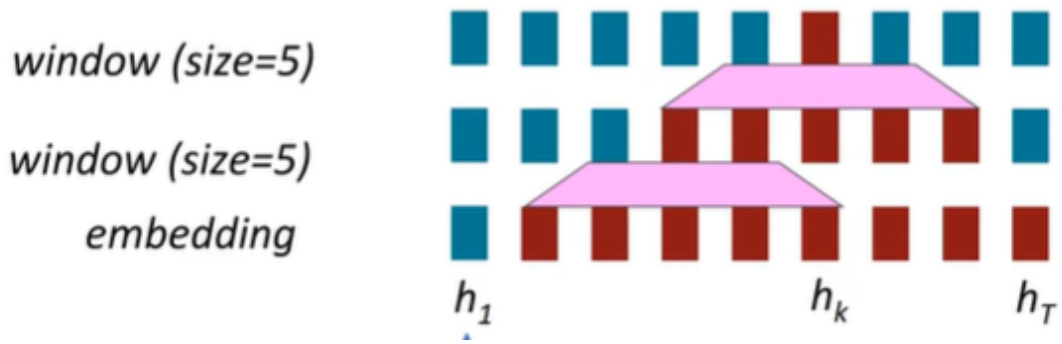
> word window model : local context를 aggregate



인코더 형태

- 위 그림에서 볼 수 있듯 sequence length가 증가해도, 병렬처리가 불가능한 연산 증가하지 않는다.
  - (필수 연산 수는 독립적으로 각각 1씩만 증가하게된다.)

> 구조적으로 Long-distance dependency 장기 의존성 문제 발생



- Local contexts를 통합하는 word window의 특성상 멀리 떨어져 있는 dependency 반영 → window layer 깊게 쌓아야 한다.
- (size = 5) 인코터 출력 h_k에 대해서 h_1은 반영되지 않는다.

How about Attention?

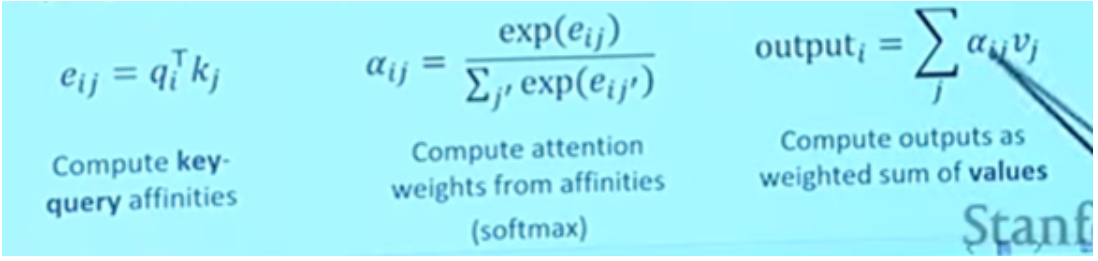
> Attention operates on queries, keys and values.

▼ Q, K, V

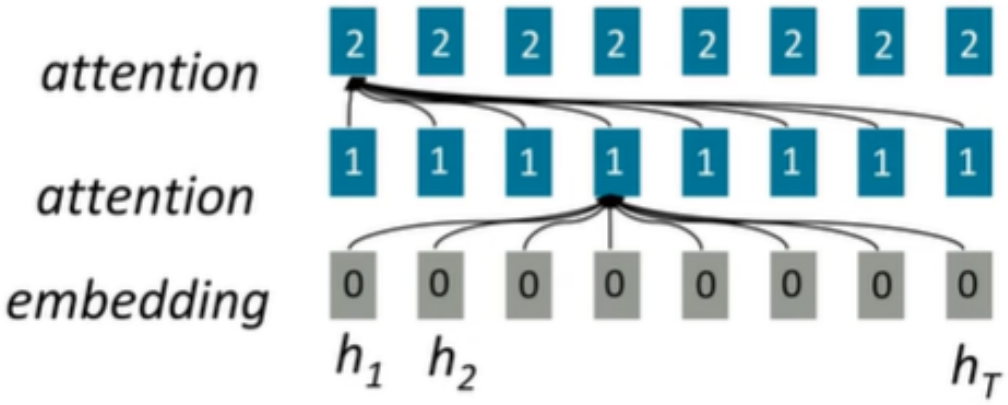
Query : 현재 보고 있는 단어의 representation으로 다른 단어를 평가하는 기준

Key : 쿼리와 관련 있는 단어를 찾기 위해서 label처럼 활용되는 벡터

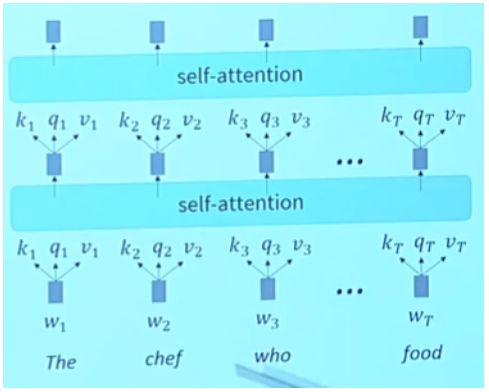
Value : Query와 Key를 통해 탐색하여 실제로 사용하는 값



attention score와 weight, output(가중합)에 대한 계산 과정



- attention 구조에 맞게 모든 state 화살표로 연결되어 있다.
- attention layer에서 이전 layer 연산이 완료된다 모든 seqeunce state가 연산 가능
- attention layer에서는 가로 sequence에 대해 병렬연산 가능
- sequence length 증가시, 독립적으로 연산의 수 증가해서 병렬처리 가능
- 모든 단어 연결되어 어떤 단어 사이에도 O(1) 연산만으로 상호작용 가능



Self-Attention만을 쌓아 NLP 모델 만들기에는 아직 문제점 존재

1. **Doesn't have an inherent notion of order(문제 : 순서에 대한 내재적 개념 없음) → Seqeunce order(솔루션)**

> Sequence 순서에 대해서 표현해야 한다. (위치 정보를 포함하는 새로운 Query, Key, Value 값 재정의

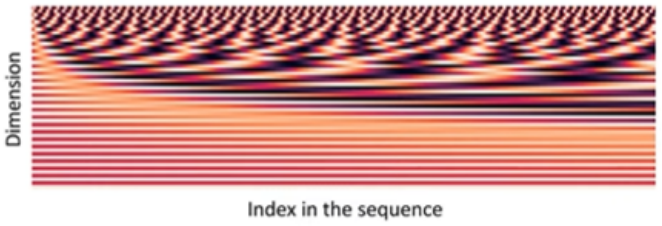
⌛ 포지션 벡터 사용!

Position Vectors $p_i \in \mathbb{R}^d$, for $i \in \{1,2,\ldots,T\}$

$$v_i = \tilde{v}_i + p_i$$
$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$

- 위치 벡터를 통해 인덱스에 따라서 순차적으로 계산된다.

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



식(왼쪽) 각기 다른 주기를 가진 사인 함수들을 연결(concatenate)하여 위치를 나타내는 방법, 그래프(오른쪽) 차원 축과 sequence index에 대한 축으로 그래프 표현

⌛ 핵심 → 다른 index일 때 다른 값을 가지는 벡터이다.

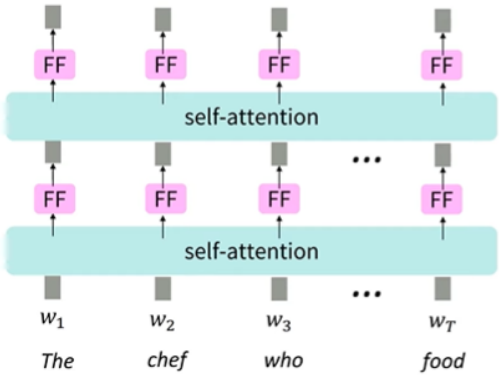
장점 : 절대적 위치 비교가 아닌 상대적인 비교가 중요하기 때문에 함수주기 < sequence 길이 일 때에도 정보를 잃지 않는다.

단점 : 학습이 불가능하다. (학습 가능한 매개변수 x) → 학습가능한 파라미터 p_i 설정!!!

---

2. **No nonlinearities for deep learning! It's all just weigthed averages(문제점) → Adding nonliearities in self-attention(솔루션)**

> self-attention → 단순한 가중합 형태의 선형 결합, 따라서 비선형 함수 필요

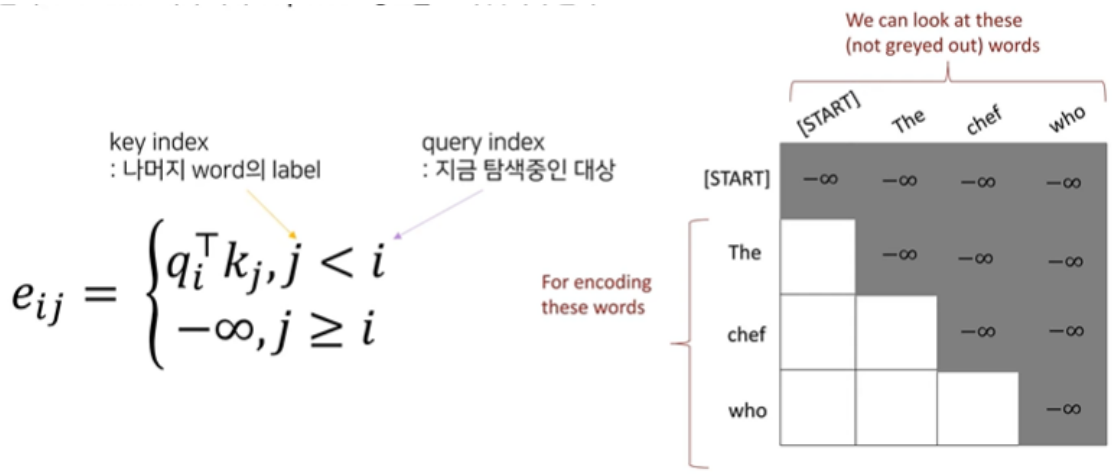


$$m_i = MLP(\text{output}_i) = W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$

- 각각의 attention output 벡터에 feedforward network 추가 (m_i), 비선형 함수 ReLU 사용 !!!

3. **Need to ensure we don't "look at the future" when predicting a sequence(문제점) → Masking the future in self-attention(솔루션)**

> Decoder에서 language modeling 수행 시, 미래 sequence 정보를 볼 수 없어야 한다.
> 순차적 → 미래 sequence 연산에 활용 x
> self-attention → 병렬 연산 가능한 구조, 미래 단어를 연산에 활용할 수 있음 (말이 안 됨 !!)



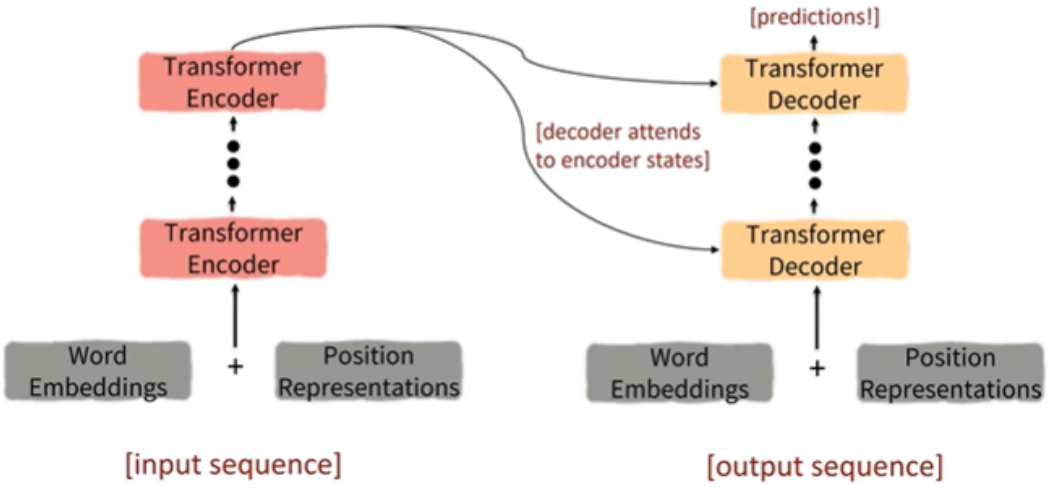
- key index < query index의 경우에만 값 계산
- 아닌 경우 - 무한대 적용
- masking한 행렬 형태 얻을 수 있다.

⌛ Self-Attention, NLP 빌딩블록으로써 역할을 할 수 있따. 그럼 Transformer 로 넘어가자!

▼ Introducing the Transformer model

The Transformer Encoder : Key-Query-Value Attetion

> Transformer의 Encoder-Decoder 구조

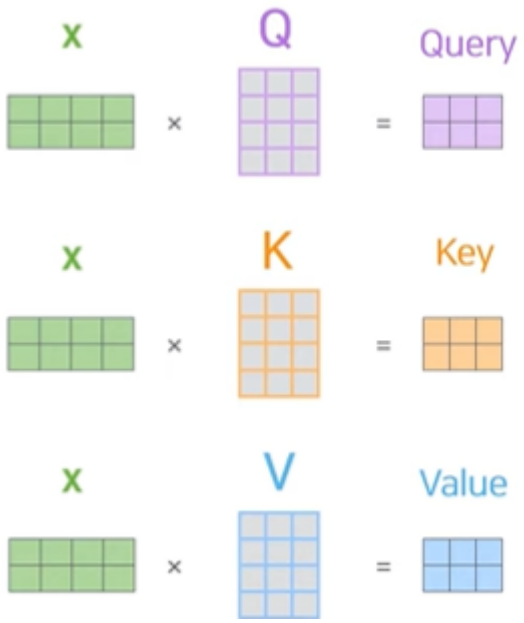


- 기존 Seq2Seq의 encoder decoder 구조 유지

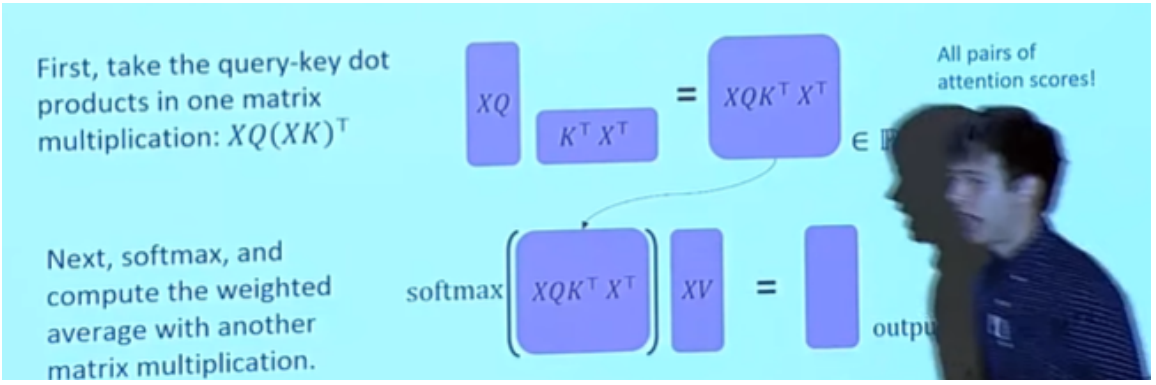
Let $x_1, \ldots, x_T$ be input vectors to the Transformer encoder; $x_i \in \mathbb{R}^d$

$k_i = Kx_i$, where $K \in \mathbb{R}^{d \times d}$ is the key matrix.
$q_i = Qx_i$, where $Q \in \mathbb{R}^{d \times d}$ is the query matrix.
$v_i = Vx_i$, where $V \in \mathbb{R}^{d \times d}$ is the value matrix.

- $x_i \rightarrow$ input vector, k_i, q_i, v_i 각각 K,Q,V에 input vector를 곱한 형태이다.



- 행렬 Q,K,V는 우리가 찾아야 하는 미지수이다.

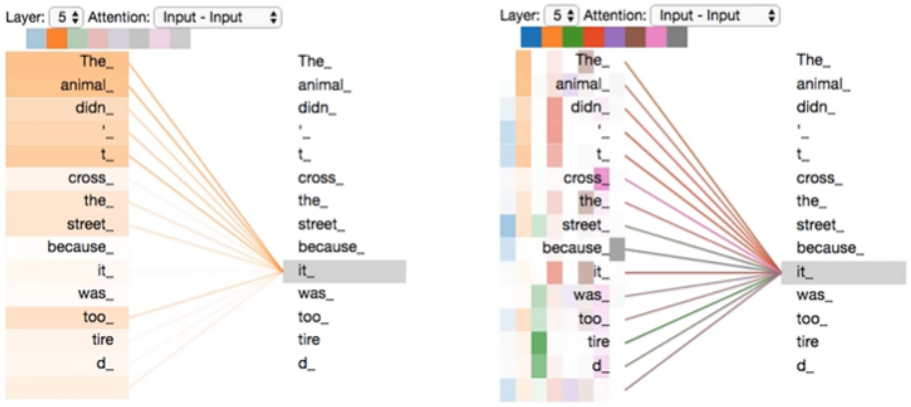


ATtention score를 계산하기 위해 query와 key에 대해서 dot product를 진행하고, 이 결과를 softmax를 이용해 가중치를 얻고 이를 가중합하여 output tensor 구할 수 있다.

The Transformer Encoder : Multi-headed attention

> 한번에 여러 부분에 집중하는 것을 가능하게 해주는 Multi-headed attention!

· 한번에 여러 부분에 집중하기 위해 여러 개의 attention head구성



single head(좌) multi head(우)

- 우측의 경우 각각 단어에서 다른 attention을 산출해 낸다. 이를 결합해 최종적인 self-attention representation 보여준다.

$Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$ h 는 head의 개수, $\ell = 1, 2, \cdots, h$

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

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

- Q,K,V의 행렬들이 $d \times \frac{d}{h}$ 차원을 가진다.
- 각 head의 결과 이어붙이면 input과 같은 $d \times d$ 차원 output으로 산출할 수 있다.
- 동일한 양의 연산을 효율적으로 할 수 있다. (아래 그림에서 같은 연산량, 같은 차원임을 확인할 수 있따)

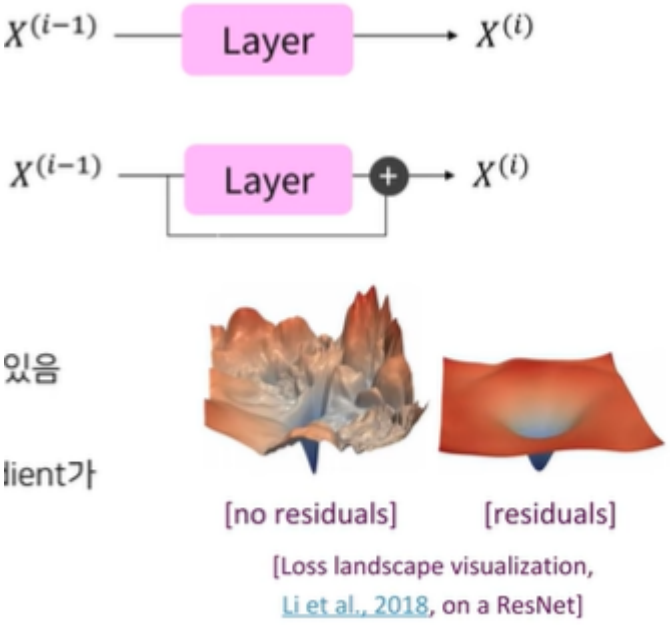
The Transformer Encoder : Residual connections (Training Tricks)

> ResNet에서도 나왔던 것 처럼 Residual connection → 자기 자신을 더해주는 것이다.

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

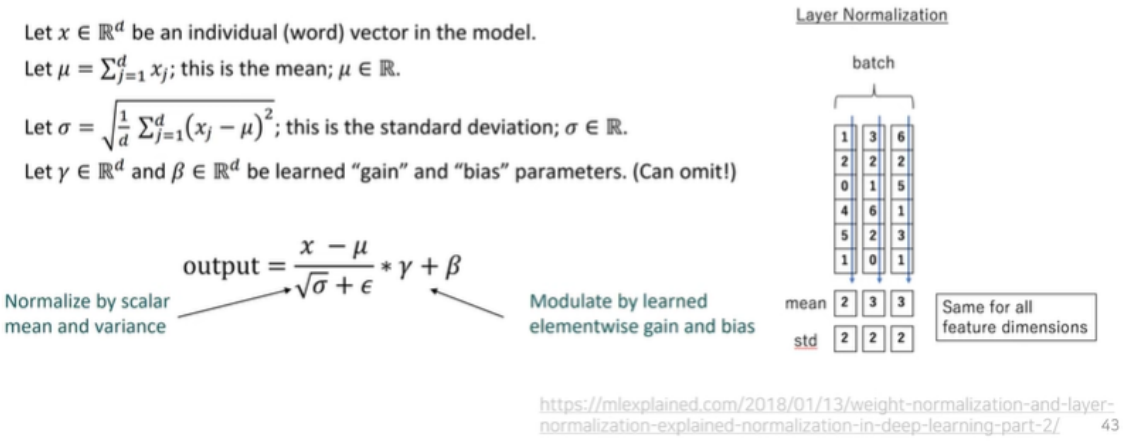
$$X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$$

- 위 그림처럼 Layer에 $X^{(i-1)}$를 더해주는 형태로, 이를 다시 해석해본다면, 이전 state보다 얼마나 변화했는지를 학습한다고도 볼 수 있다.
- gradient vanishing에 대해서도 New $f(x) = old\ f(X) + x$라고 할때, 미분할 경우 +1이 생기기 때문에 기울기가 매우 작더라도 1만큼 추가되는 효과가 있다.
- ++ 기울기 스무딩 효과 존재 → 지역해에 빠지는 문제점 보완 (아래 그림 참고)



The Transformer Encoder : Layer normalization (Training Tricks)

⏳ Layer 내에서 하나의 input sample x에 대해서 a모든 feature에 대한 평균과 분산을 구해서 normalization 수행하는 것

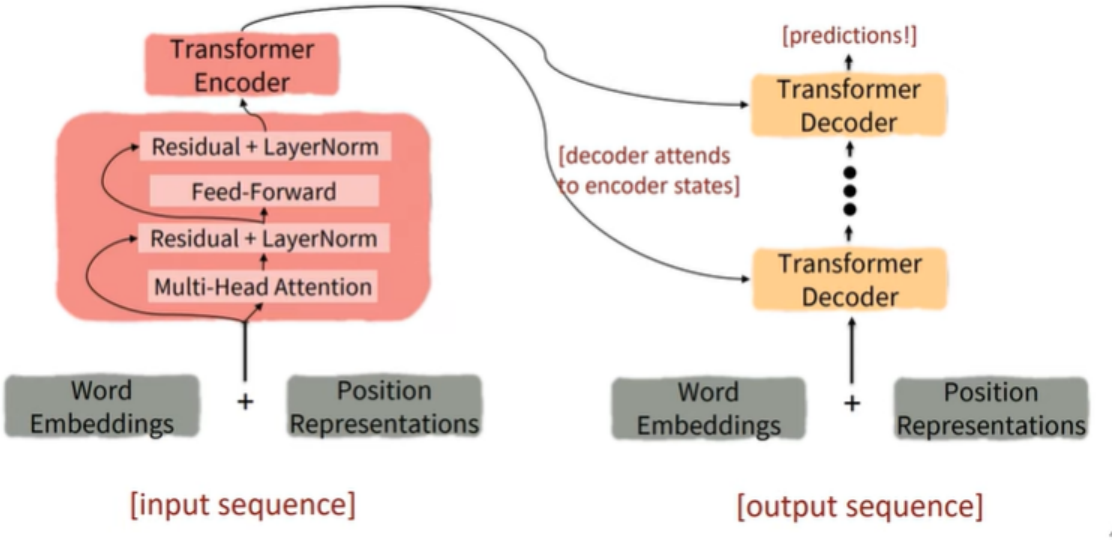


The Transformer Encoder : Scaled Dot Product

> 차원이 커질수록, vector 내적값도 커진다
> 모든 sequence간의 gradient 전파가 잘 유지하기 위해서는 dot product 결과가 너무 커지지 않게 해야 한다!

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

Encoder & Decoder Detail

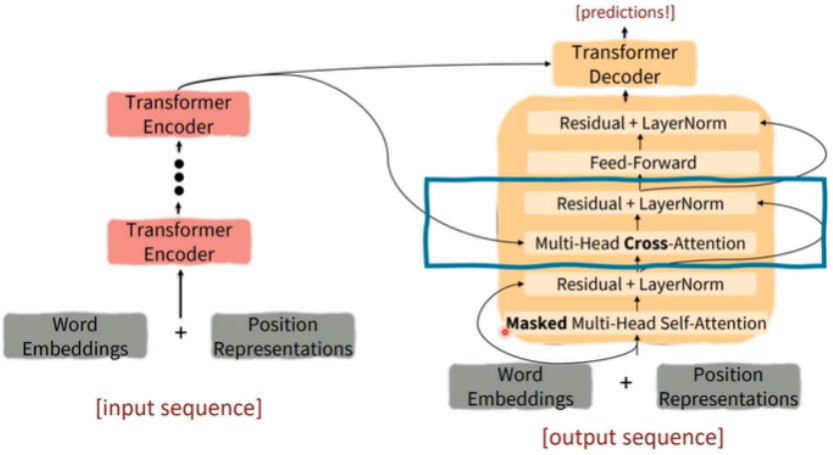


- Encoder part
  - word embeddings + position vector → Multi-Head Attention 수행
  - Residual Connection(자기 자신과 연결)과 Layer Normalization을 수행

- Feed Forward 수행

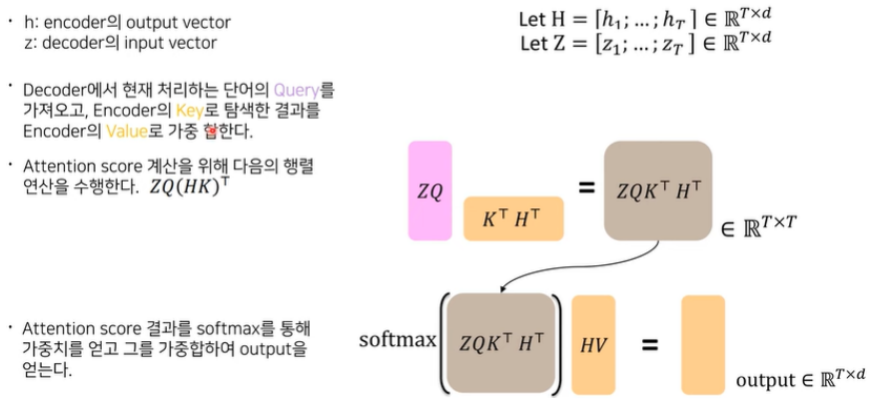
> *하나의 인코더 수행과정이고 마지막 인코더까지 모두 다 수행된다면, 디코더로 넘어간다.*

- Decoder part



- 초반부분은 Encoder와 동일하다. 하지만 미래의 sequence 결과에 대해서 반영하지 못하게끔 Masked Multi-Head Self-Attention을 수행한다.
- Residual Connection과 Layer Normalization 수행
- Multi-Head Cross-Attention 수행
  - Cross → 인코더와 디코더를 오가는 Attention 의미 (Cross-Attention = Encoder Decoder Attention)
  - Decoder에서 현재 처리하는 단어의 Query를 가져오고, Encoder의 Key로 탐색한 결과를 Encoder의 Value로 가중 합하는 것이다.

  ▼ 참고



- Feed Forward 수행

> *Transformer 총정리*

- *Seq2Seq 모델*
- *Self-Attention :*
  - *단어 처리시 문장 내의 다른 단어들로부터 힌트를 받아 현재 단어를 인코딩*
- *Positional Encoding*
  - *단어들을 set으로 취급하기 때문에 한 번에 행렬로 input한다 따라서 위치정보 x*
  - *추가적인 position representation을 더해 positional encoding을 수행*
- *Encoder-Decoder 구조*
  - *각 Encoder, Decoder 동일한 개수 사용*
  - *Multi-head self-attention*
  - *Position-wise feed-forward network*
  - *Residual connection*
  - *Layer Normalization*

▼ Great results with Transformers

Machine Translation

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**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

- 기존의 SOTA로 여겨졌던 모델들 보다 높은 성능 보여줌

Document generation

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

- 기준으로 여겨지는 seq2seq 모델보다 훨씬 낮은 perlexity를 보여주며 훨씬 더 좋은 성능 보여줌

Aggregate Benchmark

Rank	Name	Model	URL	Score
1	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8
2	HFL iFLYTEK	MacALBERT + DKM		90.7
3	Alibaba DAMO NLP	StructBERT + TAPT		90.6
4	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6
5	ERNIE Team - Baidu	ERNIE		90.4
6	T5 Team - Google	T5		90.3

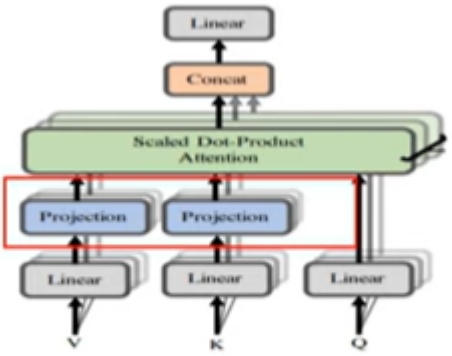
- 벤치마크 점수 transformer 기반 모델들이 순위권을 차지함. (2021년 기준)

▼ Drawbacks and variants of Transformers

단점 1 : sequence length가 증가함에 따라 계산량이 2차식으로 증가 $O(T^2d)$

- Improve

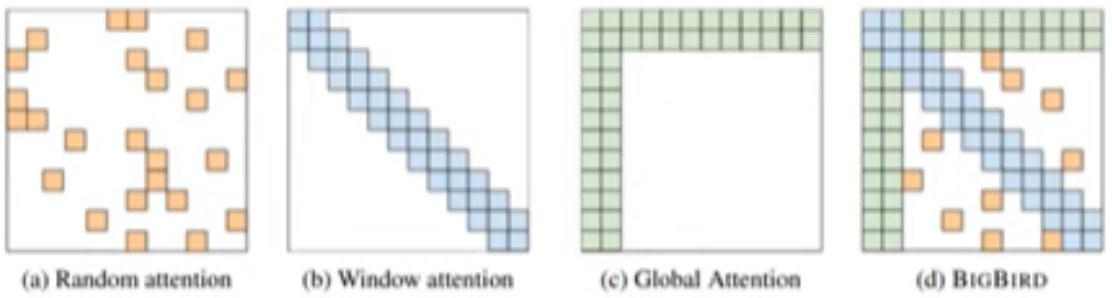
Linformer [Wang et al., 2020]



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

⏳ projection을 통해 value와 key의 sequence length dimension을 낮춘다!

BigBird [Zaheer et al., 2021]



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

단점 2 : Position representations에 대한 개선의 여지가 존재!