CS224N_Lecture 9&10

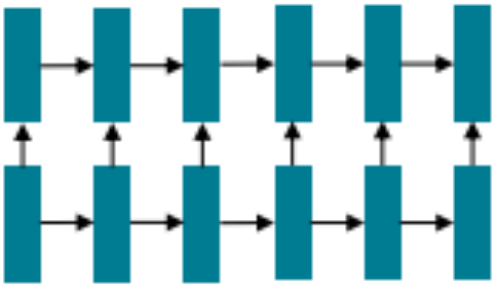
Lecture 9

Self-Attention and Transformers

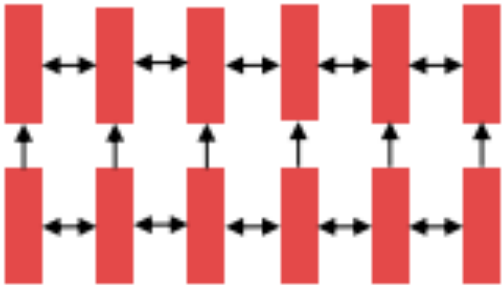
1. Self-Attention

Review RNN

- NLP task에서 RNN의 사용방법에 대해 알아봤다

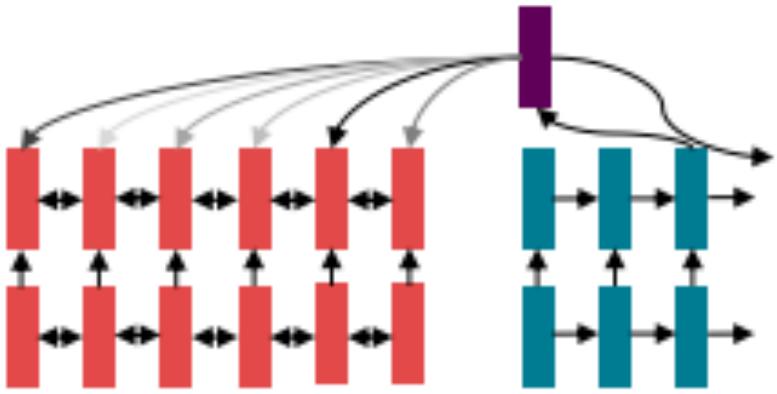


one-direction



bi-direction

- Use Attention - allow flexible access to memory

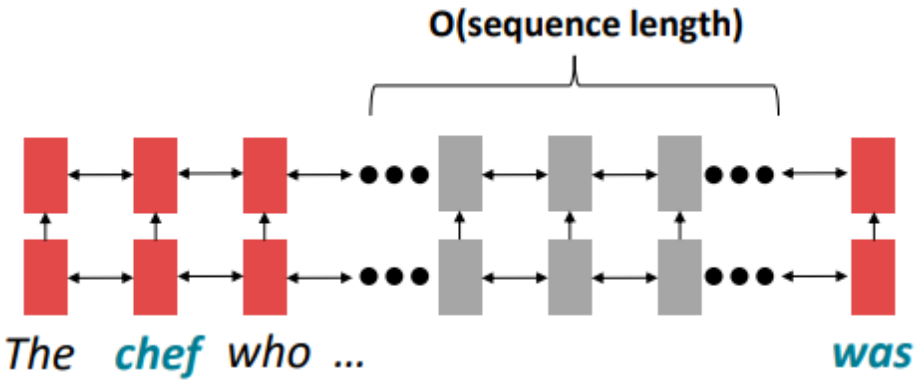


attention method

- 순차적으로 진행하는 RNN!

Problems with RNN

- Linear Interaction distance
  - Encoding **linear locality** → 단어가 순서대로 들어가니깐 가까이 있는 것들끼리 영향 ↑ (one of thing about distributional hypothesis)
  - **Problem** : 멀리 떨어져 있는 것들끼리의 interaction을 알기 위해선 떨어진 길이 *O(sequence length)* 만큼의 step이 필요 ⇒ Hard to learn long-distance dependencies



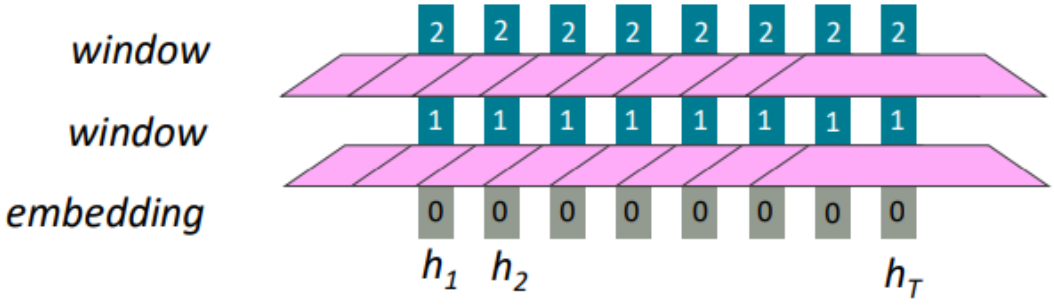
- Lack of parallelizability
  - 들어오는 순서대로 학습이 되기 때문에 병렬적으로 수행 불가능

    예시) I kicked the ball.

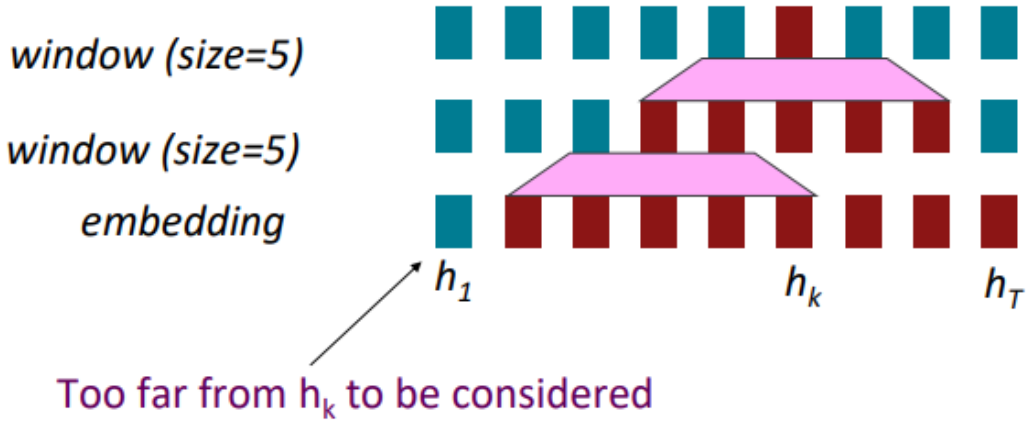
    - the 이전에 kicked가 계산되어 hidden state로 kicked에 대한 값이 넘어가야지만 the 에 대한 계산 가능

- Alternatives

1. word windows



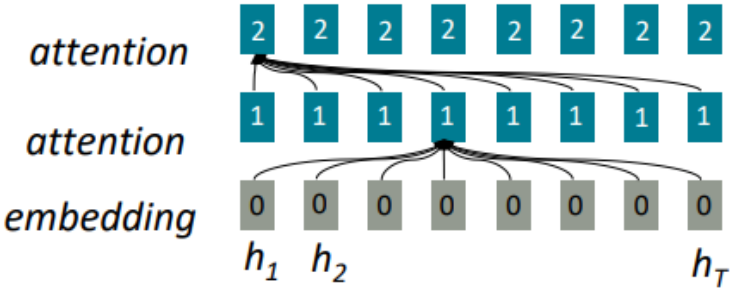
    - 모든 단어를 독립적으로 embed 한다.
    - word window classifier layer를 stacking.
    - get O(1) dependence in time



    - long-distance dependency ⇒ stacking wider word window layers , 멀리 떨어진 단어의 interaction 고려
    - **Maximum interaction distance = sequence length / window size**
    - 하지만 h_1은 h_k를 encoding하는데 고려되지 않는 문제점 (여전히) 발생

2. attention

    - Attention solves two problems of RNN.



- word's representation을 쿼리 취급한다.
- 문장 한번에 들어가 연산하므로 문장이 길이가 연산에 상관 X

self-attention

- Query, Key, Values : d dimension을 가지는 vector, drawn from same source(sentence)

$$v_i = k_i = q_i = x_i$$

x_i : one vec per word

  - 각 vector들의 역할
    - query : 주어진 벡터들 중에 어떤 벡터를 선별적으로 가져올지에 대한 기준이 되는 vector
    - key : query 벡터와 내적을 통해 유사도를 구하는 재료 vector
    - value : 내적으로 구해진 유사도, 가중치를 결합하여 encoding을 만들 재료 vector

- operation

$$e_{ij} = q_i^\top k_j$$

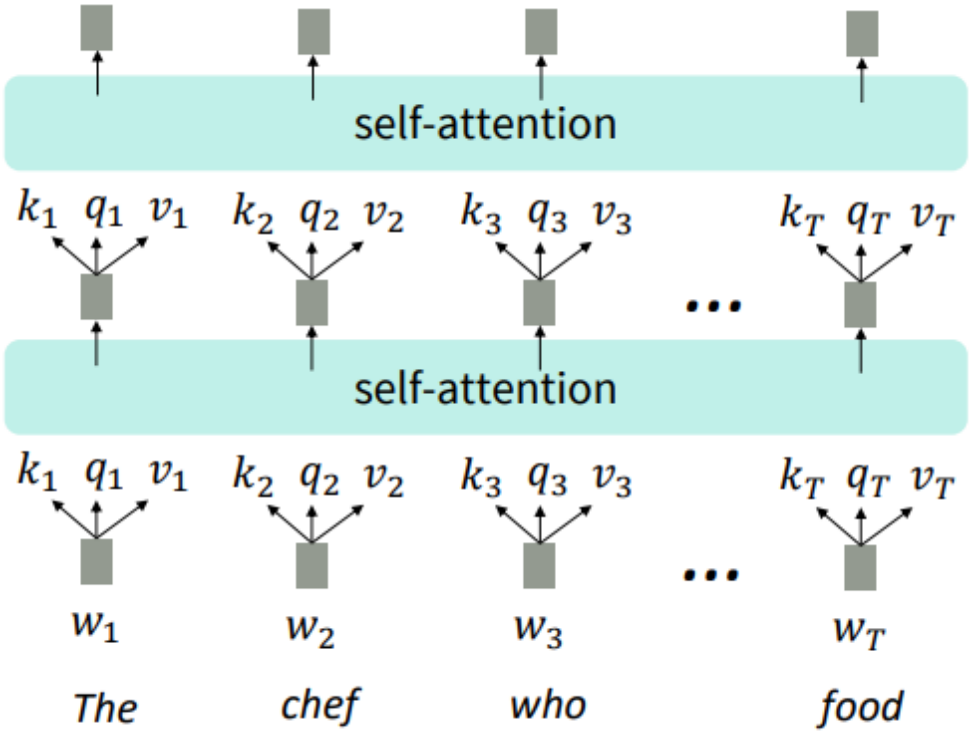
Compute **key-query** affinities

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

Compute attention weights from affinities (softmax)

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

Compute outputs as weighted sum of **values**



- 한계점
  - No inherent order
    - solution : representing sequence order
    - p_i : position vectors

$p_i \in \mathbb{R}^d$, for $i \in \{1,2,\ldots,T\}$ are position vectors

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

add position vector and embedding vector

- Do this at first layer

Construct p_i, How to positional embedding?

- 방법 1 : 데이터에 0~1 사이의 label을 붙인다. 0이 첫번째, 1이 마지막 단어

  예시) I (0) / love (0.5) / you (1)

  → Input의 총 크기를 알 수 없다.

  → delta (=단어의 label 간의 차이) 값이 일정한 의미를 갖고 있지 않음
- 방법 2: 각 time-step마다 선형적으로 숫자를 할당

  예시) I (1) / love (2) / you (3)

  → 총 크기에 따라 가변적, delta 일정해짐

  → 숫자가 매우 커진다.

  → 훈련 시 학습할 때 보다 큰 값이 들어오면 모델의 일반화가 어려워짐 (특정 범위를 갖고 있지 않다)

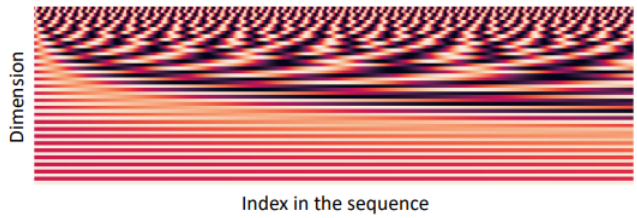
- 필요한 기준
  1. 각 time-step마다 하나의 유일한 encoding 값을 출력 (문장에서 단어의 위치)
  2. 서로 다른 길이의 문장에 있어서 두 time-step간 거리는 일정
  3. 모델에 대한 일반화가 가능 → 더 긴 길이의 문장에 적용 가능, 특정 범위 내에 있어야 함
  4. 하나의 key 값 처럼 결정

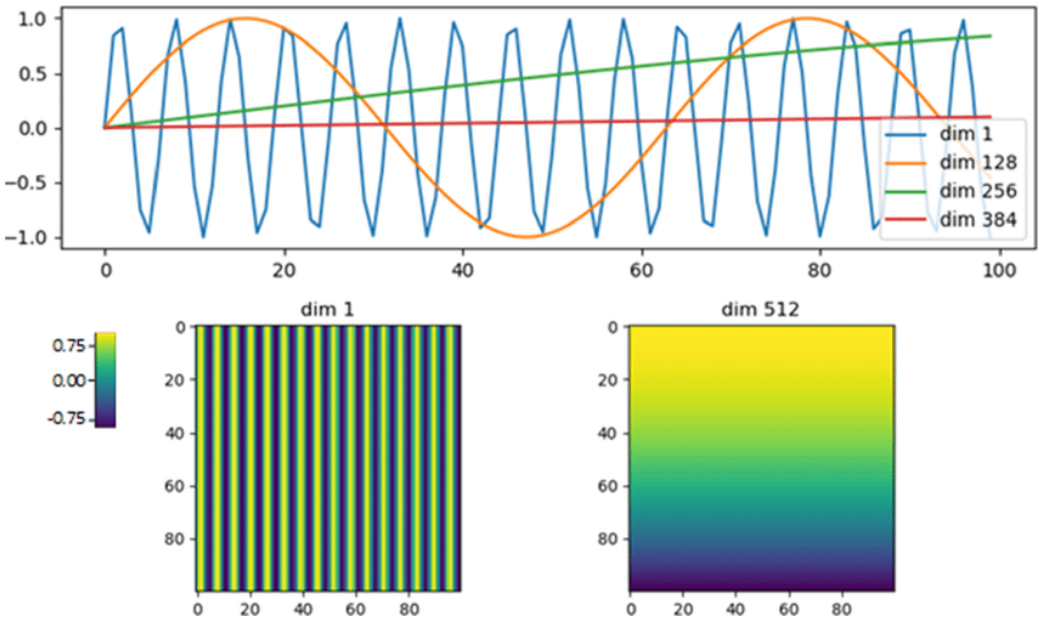
- Sinusoidal position representation
  - 위 기준을 모두 충족시킨 embedding 방법
  - d-dimensional vector 로 문장 내 특정 위치 표현
  - 모델 자체 내에서 사용 되지 않음 → 단어의 순서 정보를 표현하기 위해 input을 늘렸다.

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

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





- 모델의 별다른 노력없이 상대적 position을 갖게 됨
- not learnable parameter

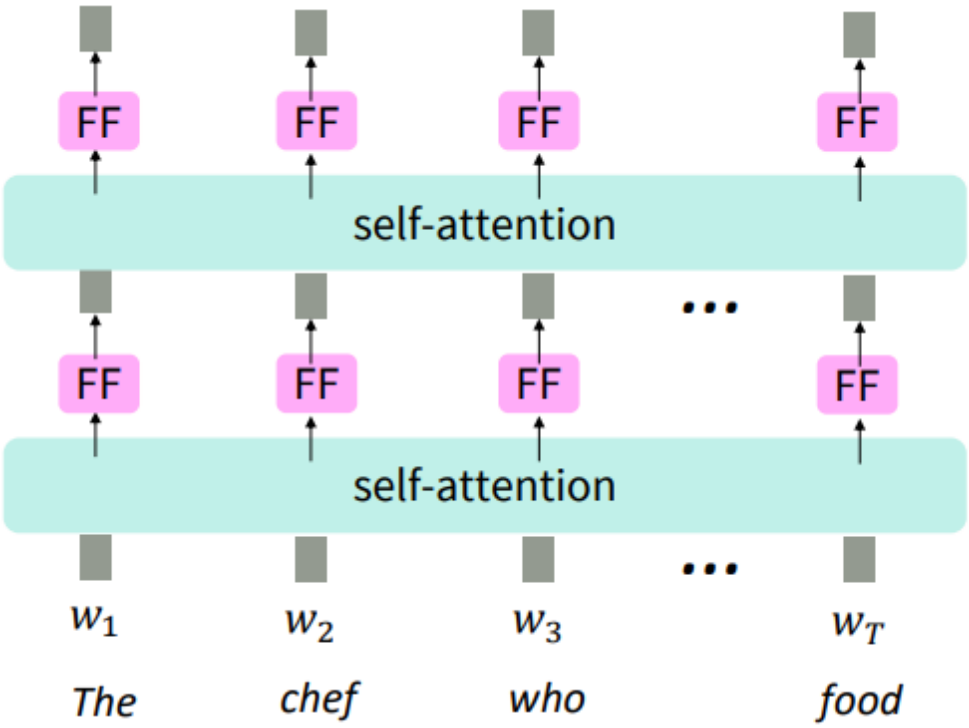
cf)

▼ Learned absolute postition representations

- 데이터에 맞는 각각의 위치 정보를 구할 수 있다.
- 범위 밖의 indices 를 추론하기 힘들다. (범위(1, ... , T)를 정해놓고 학습을 시켰기 때문에)

- 그 외의 position representation에 대한 연구 : Relative linear position attention , Dependency syntax-based postion

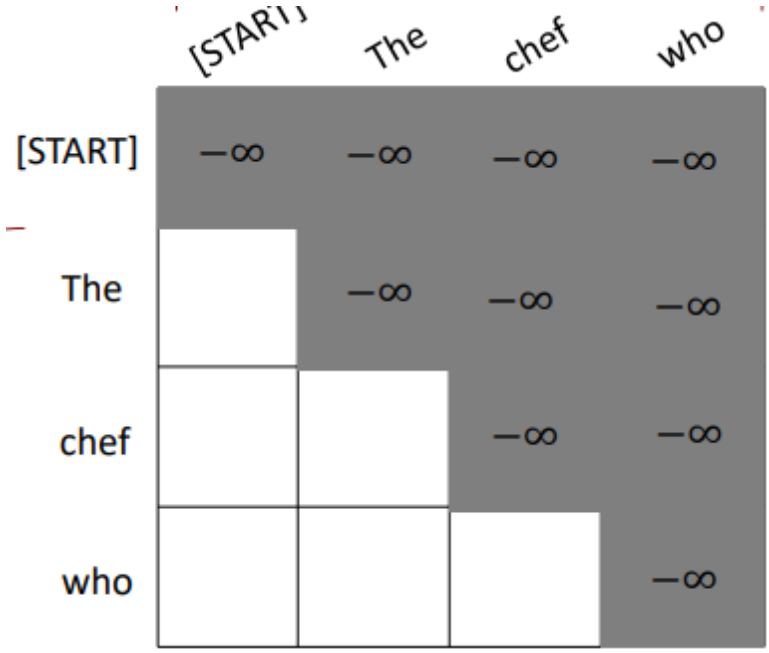
- No nonlinearity
  - non-linearity + abstract features ⇒ great deep learning
  - solution : add **feed-forward network**



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

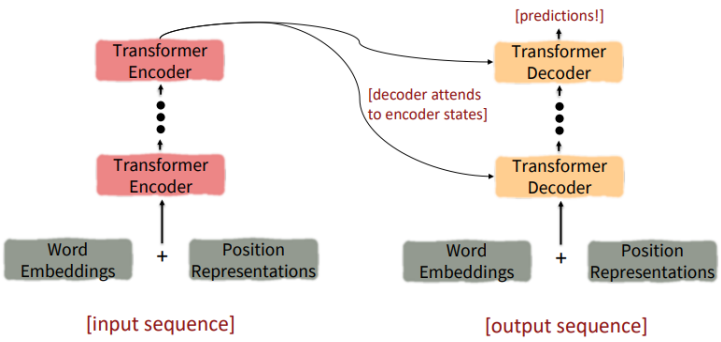
Apply FF layer

- Need to ensure “We don’t look at the future.”
  - To use self-attention in decoders, need to ensure we can’t see the future.
  - solution : masking the future ⇒ setting attention scores to negative infinity



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

2. Transformers



Encoder

- Key-Query-Value Attention

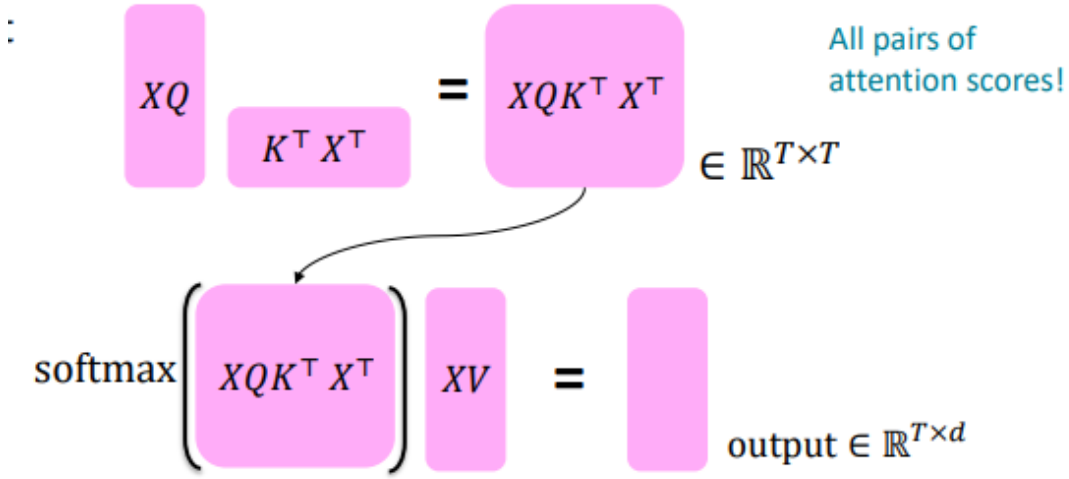
  $x_1, ..., x_T$: input vectors to encoder (d - dim)

  - keys, quries and values

    $k_i = Kx_i$, $q_i = Qx_i$, $v_i = Vx_i$ (K, Q, V are different through linear transformation)

    ⇒ allow different aspects

- Matrix From : $ouput = softmax(XQ(KV)^T)XV$



- Multi-headed attention
  - Why multi-head attention 필요?

    → 동일한 sequence가 주어졌을 때, 서로 다른 기준으로 여러 측면에서의 정보를 뽑아낼 필요

    예시) I went to the school. / I studied hard. / I took the rest.

    ⇒ 어떤 문장에서는 I 에 대한 행동을 중심으로 하는 정보

    ⇒ 어떤 문장에서는 I에 대한 장소의 정보

    → 예시처럼 서로 다른 정보를 병렬적으로 뽑고 마지막에 합치는 형태로 구성

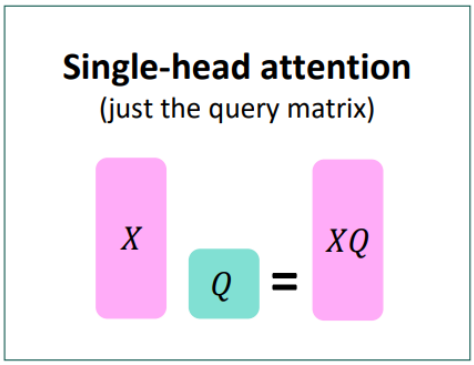
$$Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d\times\frac{d}{h}}$$

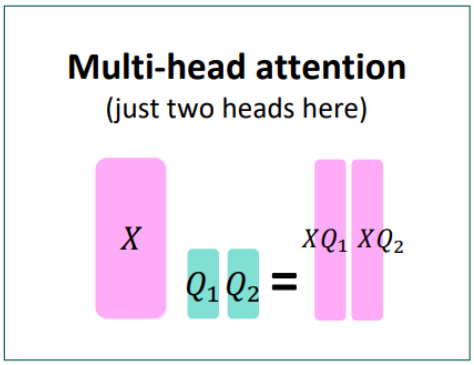
h is the number of attention heads.

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

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

Y 는 출력 차원을 맞춰주는 역할

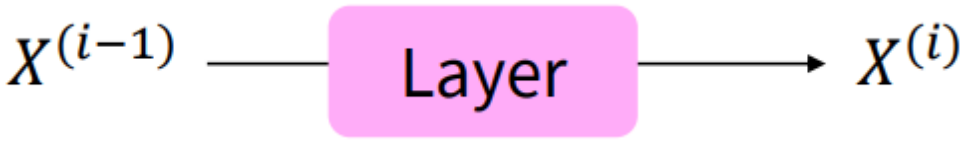




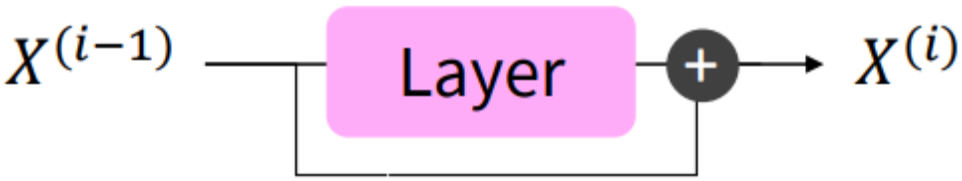
same amount of computation as single-head self attention and having different distributions for each of the different heads

- Training Tricks
  - Residual connections

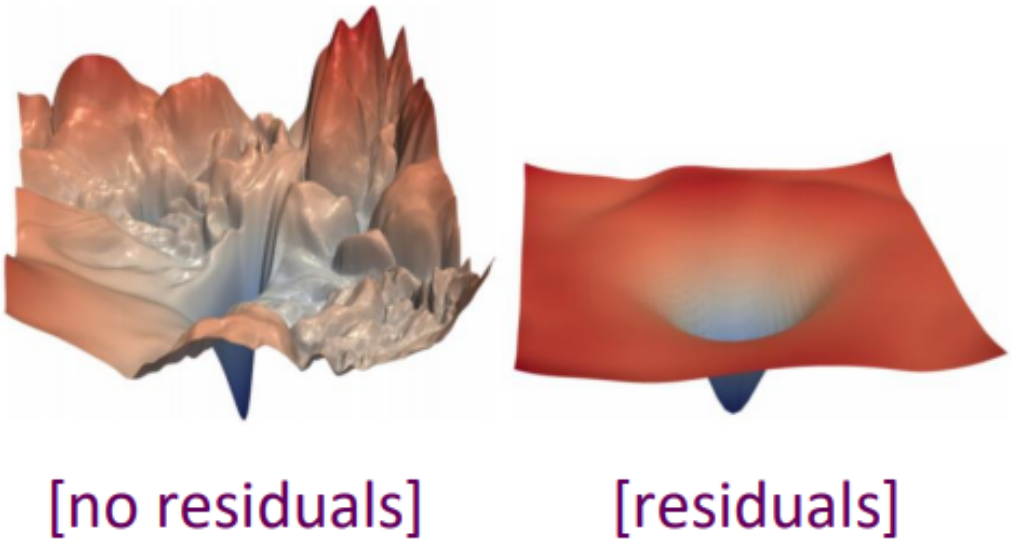
    Instead of



    Using this method

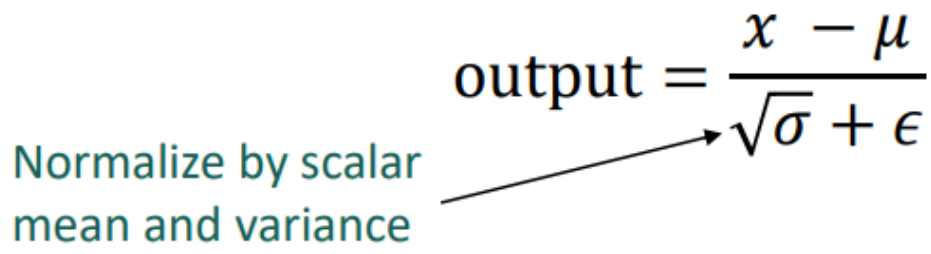


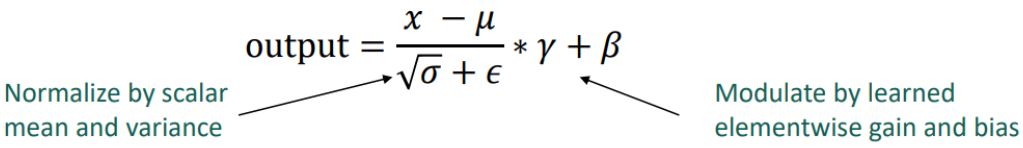
    - 이런 connection을 통해 gradient vanishing 문제에 빠지는걸 방지한다.



no residuals : local minimum 에 빠지기 쉽다. / Residuals : much smoother → better training

- Layer normalization
  - To help models train faster





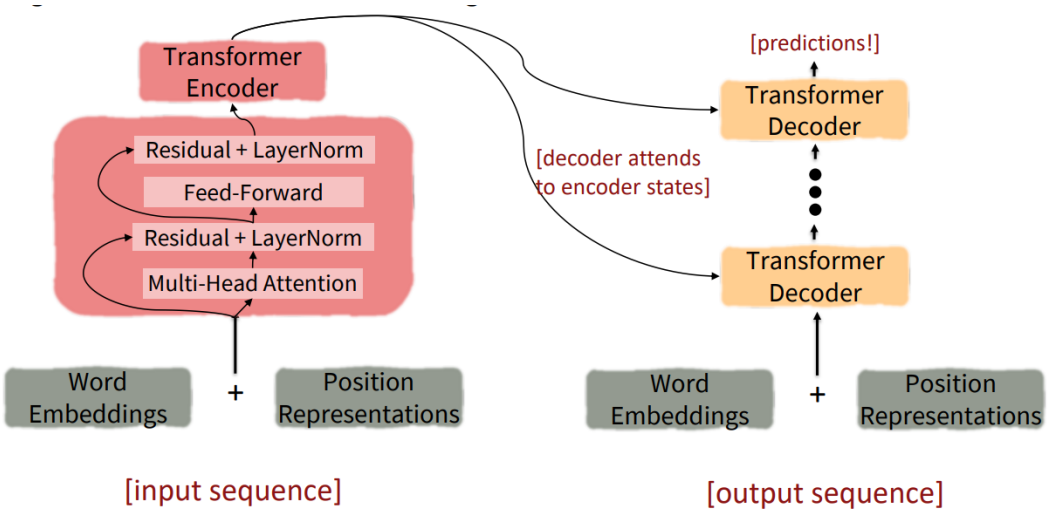
어떤 논문에서는 gain & bias 가 별 도움이 안된다고 가끔씩 사용한다고 함

  - layer norm 을 안하면 학습이 잘 안되는 것 만큼 transformer에서 중요한 것이 됨!

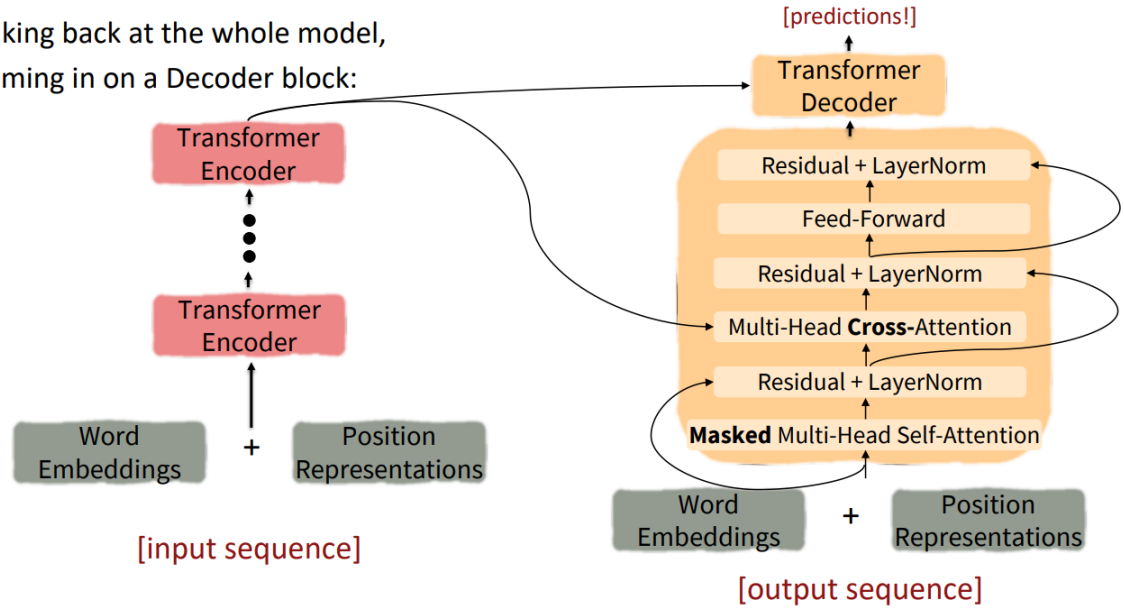
- Scaled Dot Product
  - Query, Key 내적값의 분산이 커짐 → 유사도(가중치)의 분포가 커짐 → 특정값에 몰림 → gradient vanishing ⇒ 스케일링 해주면 해결!

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

▼ Transformer encoder-decoder block



king back at the whole model,
ming in on a Decoder block:



Decoder

- Cross-Attention
  - $h_1, ..., h_T$: output vectors from **encoder**

- $z_1, ..., z_T$: input vectors from **decoder**

- keys and values are drawn from the **encoder**

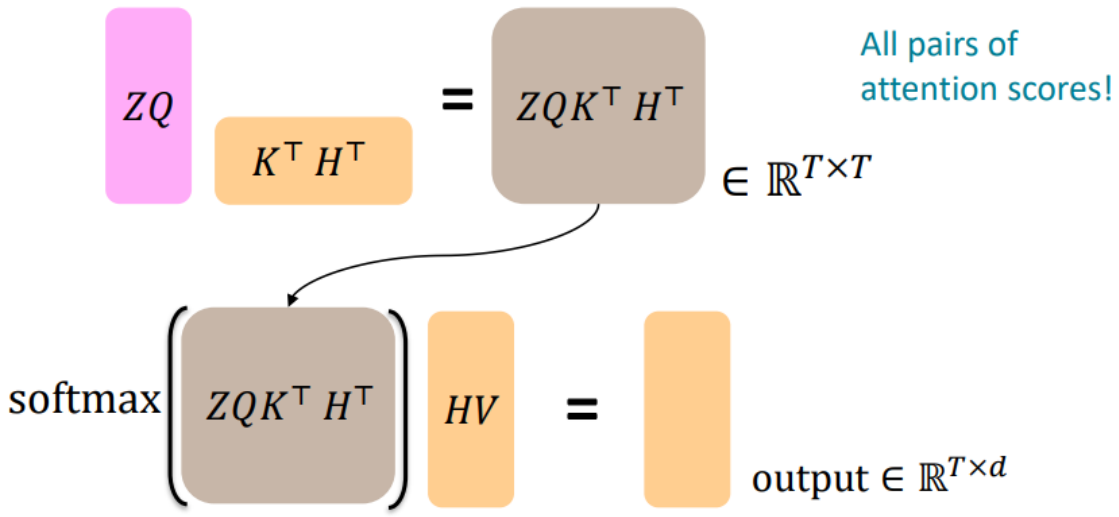
$$k_i = Kh_i, v_i = Vh_i.$$

- queries are drawn from the **decoder**

$$q_i = Qz_i.$$

$\mathrm{H} = [h_1; ...; h_T] \in \mathbb{R}^{T \times d}$ be the concatenation of encoder vectors.
$\mathrm{Z} = [z_1; ...; z_T] \in \mathbb{R}^{T \times d}$ be the concatenation of decoder vectors.

$$output = softmax(ZQ(HK)^T)HV$$



Great Results

- 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**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$

- Document generation (summary)

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

Drawbacks

1. Quadratic compute in self-attention
   a. 모든 쌍의 interaction을 구한다는 의미는 sequence length가 길어질 수록 연산량이 급격히 많아짐 (recurrent model은 grow linearly)
   b. self-attention is 병렬적으로 계산하지만 $O(T^2 d)$ 만큼 계산량

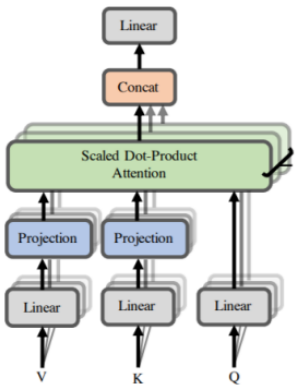
$$XQ \quad K^\top X^\top = XQK^\top X^\top \in \mathbb{R}^{T \times T}$$

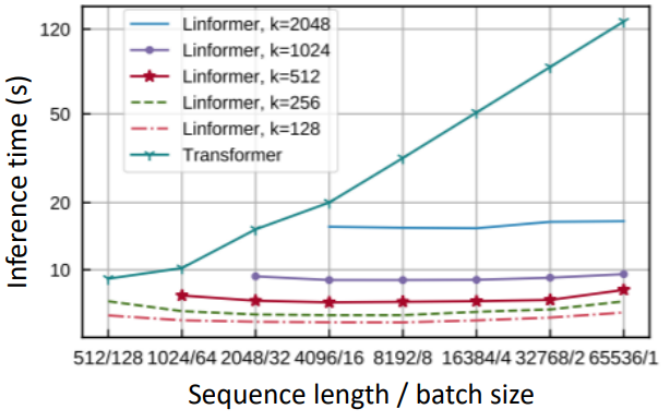
T : the sequence length , d : dimensionality

→ In practice , we set T = 512.

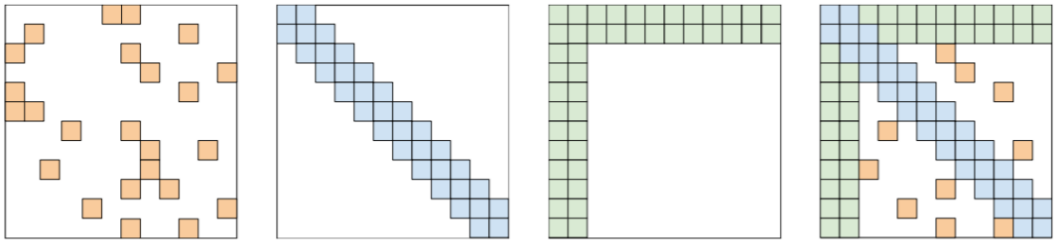
solution

1. Linformer : key 와 value의 차원을 낮게 projection





2. Bigbird : all- pairs interaction을 다른 attention interaction의 집단으로 대체



⇒ 그래도 normal transformer 가 가장 많이 쓰임!

2. Position representations

   solution

   1. Relative linear position attention
   2. Dependency syntax-based position

Lecture 10

- 다음주에 꼭 해올게요 8n8