

4. Attention & Transformer

Minsung Kim

Artificial Intelligence in Korea University(AIKU)

Department of Computer Science and Engineering, Korea University

Contents

- Machine Translation
- Attention
 - Attention
 - Self-Attention
- Transformer

Machine Translation

Machine Translation

- Machine Translation(MT)

- source language로 표현되어 있는 문장 x 를 target language로 표현된 문장 y 로 번역하는 task

$x:$ *L'homme est né libre, et partout il est dans les fers*



$y:$ *Man is born free, but everywhere he is in chains*

출처: CS224n lecture 7

Statistical Machine Translation (1990s – 2010s)

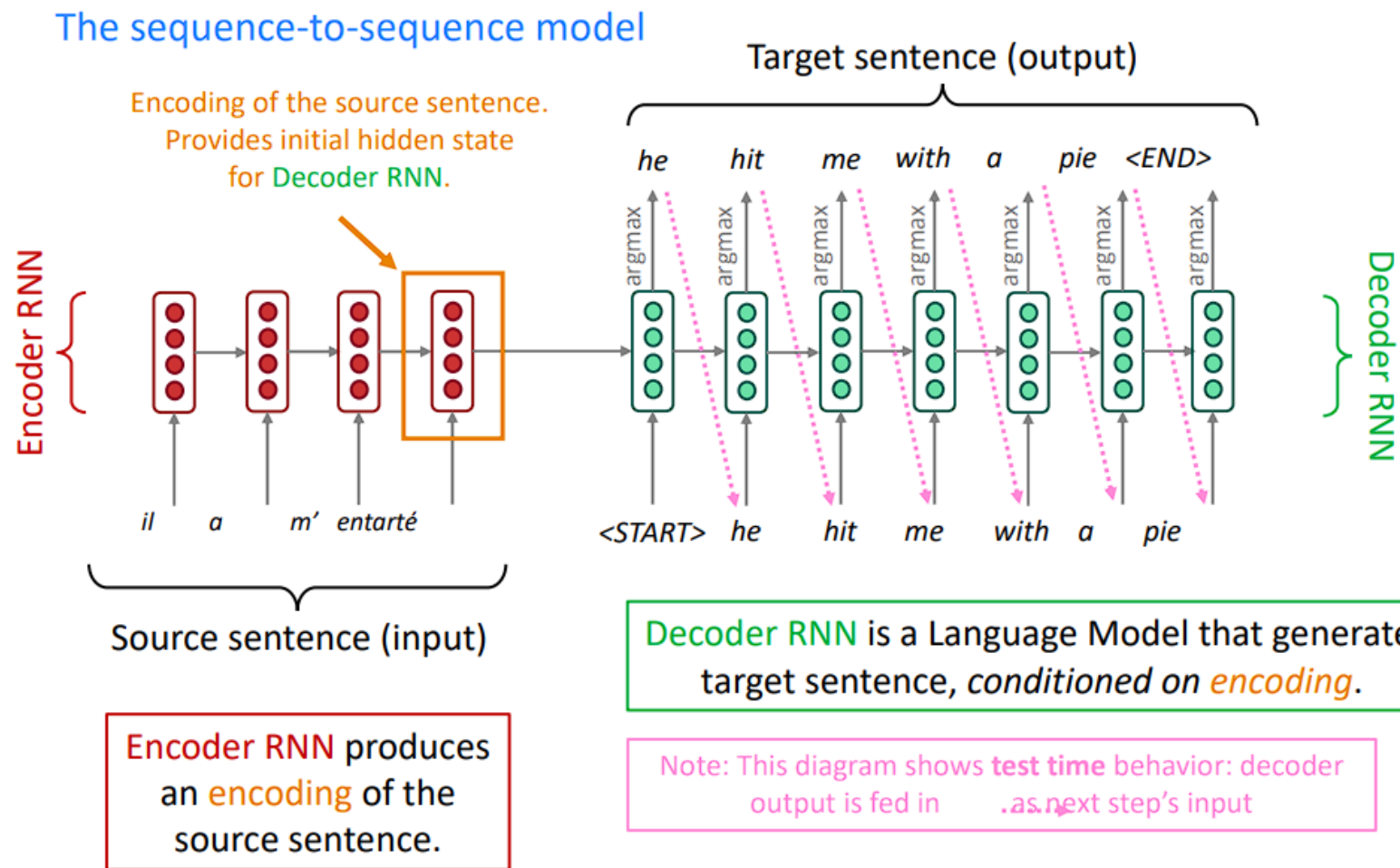
- 예전에는 MT를 수행하기 위해 data를 기반으로 한 probabilistic model로 접근
- 주어진 French sentence x 에 대해서 가장 좋은 English sentence y 를 찾자!
 - $\operatorname{argmax}_y P(y|x) = \operatorname{argmax}_y P(x|y)P(y)$ (using Bayes Rule)
- $P(x|y)$: Translation Model
 - 단어나 구절이 어떻게 번역되어야 하는가
 - Parallel data를 통해 학습
- $P(y)$: Language Model
 - Target language에 대한 지식, 자연스러운 문장이 어떤 구조인가
 - Monolingual data를 통해 학습

Neural Machine Translation

- Neural Machine Translation(NMT)
 - MT task를 end-to-end neural network를 통해 해결해보자.
 - Neural network architecture는 **sequence-to-sequence**(seq2seq) 모델이라고 부르며 2개의 RNNs으로 이루어져 있다.

Neural Machine Translation

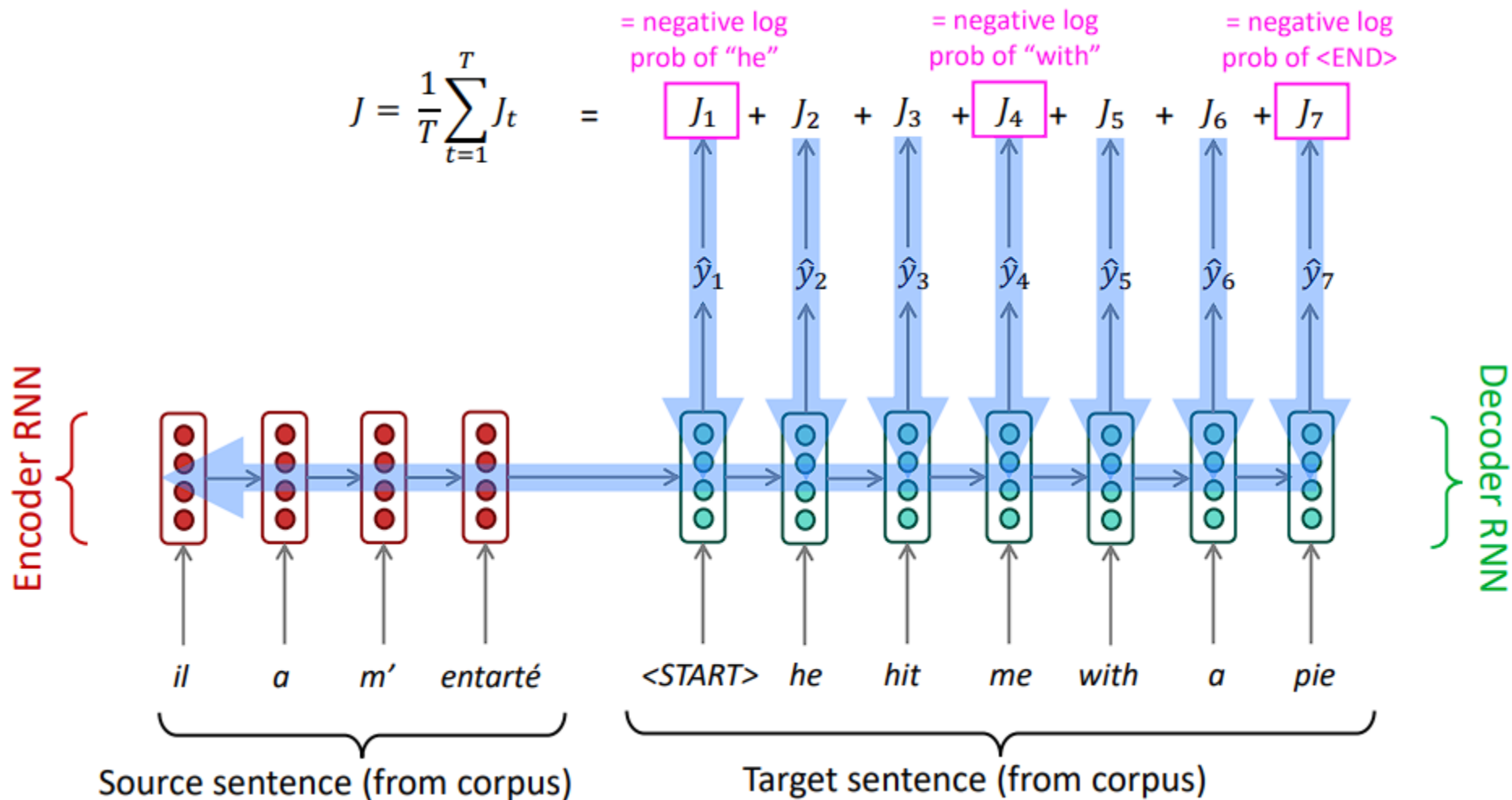
- Seq2Seq Model 구조



Neural Machine Translation

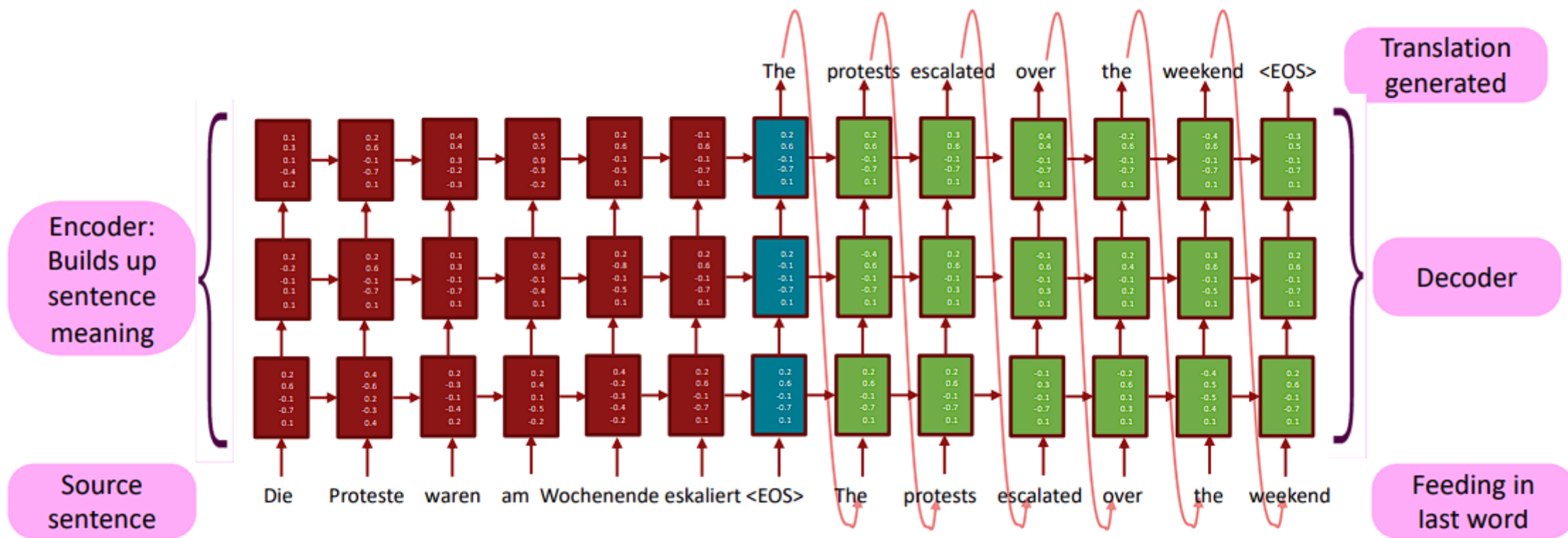
- Seq2Seq model은 Conditional Language Model이다.
 - Decoder가 **source sentence**와 **지금까지 예상한 target sentence**를 바탕으로 다음 단어를 예측하기 때문이다.
 - $P(y|x) = P(y_1|x)P(y_2|y_1, x)P(y_3|y_1, y_2, x) \dots P(y_T|y_1, \dots, y_{T-1}, x)$
- Seq2Seq model은 MT외에도 여러가지 task를 수행할 수 있다.
 - Summarization
 - Dialogue
 - Parsing
 - Code generation

Training a NMT system



출처: CS224n lecture 7

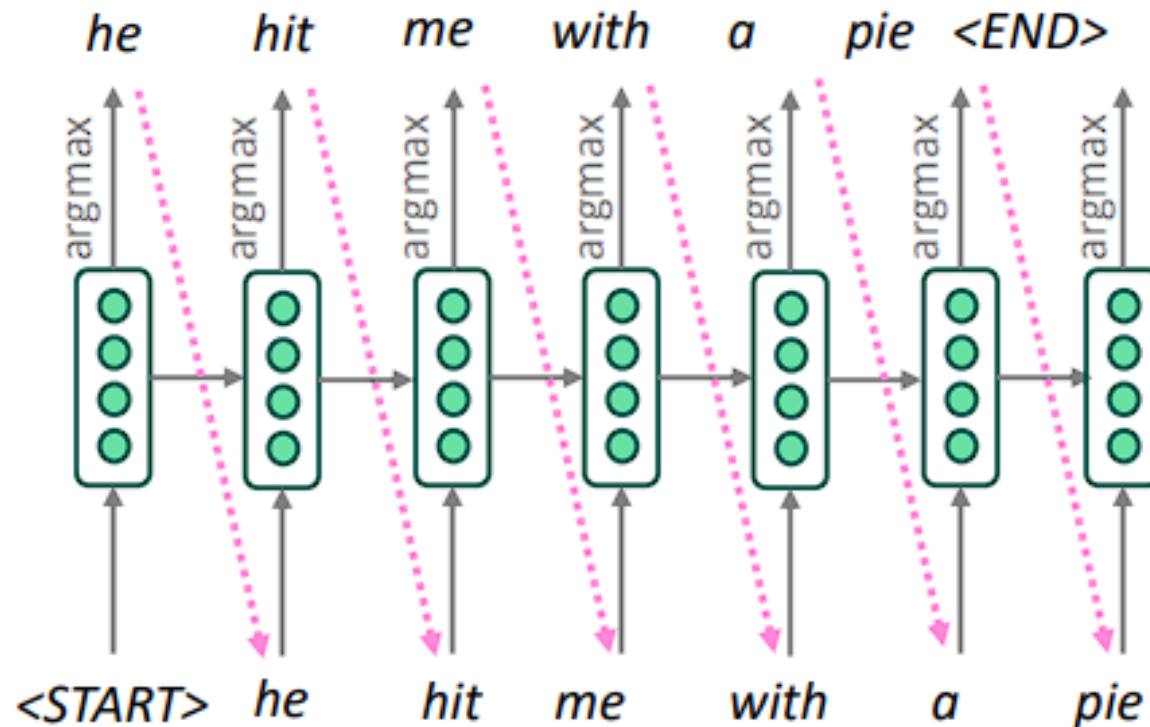
Multi-layer RNNs



출처: CS224n lecture 7

Greedy decoding

- 매 step마다 가장 확률이 높은 단어를 선택
 - 문제점은? 'he' 다음에 올 단어는?



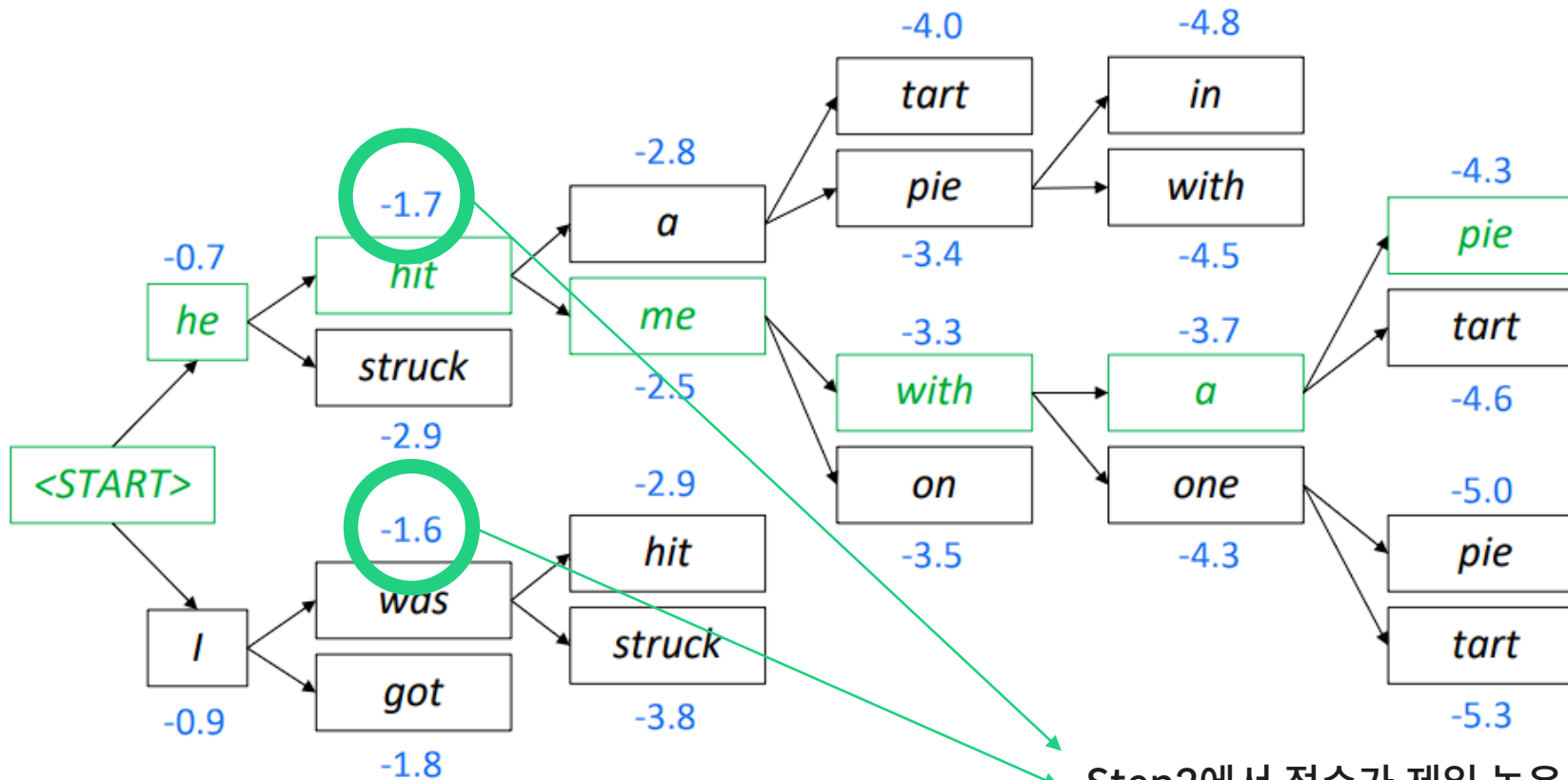
출처: CS224n lecture 7

Exhaustive search decoding

- 그렇다면 모든 가능한 y 를 계산해보자
 - 단어의 개수가 V 개, $\text{step}(\text{sequence의 길이})$ 가 t 라면 총 경우의 수는 V^t 개
- V^t 개의 경우를 모두 계산하는 건 시간과 비용 측면에서 좋지 않다.

Beam search decoding

- 각 step마다 확률이 가장 높은 k 개만 남겨둔다.
 - Optimal solution을 찾는다는 보장은 없지만 exhaustive search보다 훨씬 효율적



출처: CS224n lecture 7

Step2에서 점수가 제일 높은 2개에 대해서만
step3를 진행

Neural Machine Translation

- Advantages

- SMT보다 더 좋은 성능
- 하나의 end-to-end 모델
- 인간의 노력이나 보수 작업이 필요하지 않음

- Disadvantage

- Black box 모델
 - 디버깅이 힘들다
 - 모델이 어떠한 규칙으로 번역을 하는지 알 수 없다
 - Bias의 문제

Evaluation of Machine Translation

- BLEU (Bilingual Evaluation Understudy)
 - 모델이 번역한 결과와 사람이 직접 번역한 결과가 얼마나 비슷한지 n-gram을 기준으로 비교
 - 두 문장 사이에 겹치는 n-gram이 몇 개인지
 - 같은 의미를 가진 **좋은 번역**이더라도 BLEU 점수는 낮을 수 있다
 - Example) 예쁘다 vs 아름답다
 - R1: The cat is on the mat
 - R2: There is a cat on the mat
 - C1: The cat and the dog
 - C2: The The The The The.
 - C3: There is a cat on the mat.
 - C4: Mat the cat is on a there.
 - <https://medium.com/nlplanet/two-minutes-nlp-learn-the-bleu-metric-by-examples-df015ca73a86>

So, is Machine Translation solved?

- 관용구를 잘 해석하지 못한다.

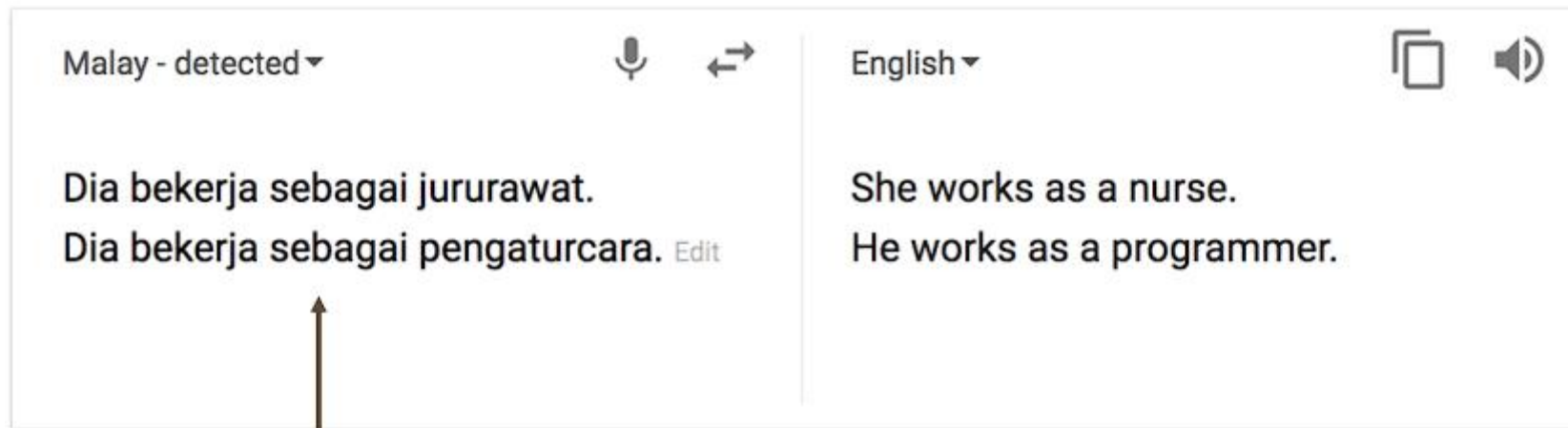


?

출처: CS224n lecture 7

So, is Machine Translation solved?

- 편향된 지식이 그대로 학습된다.

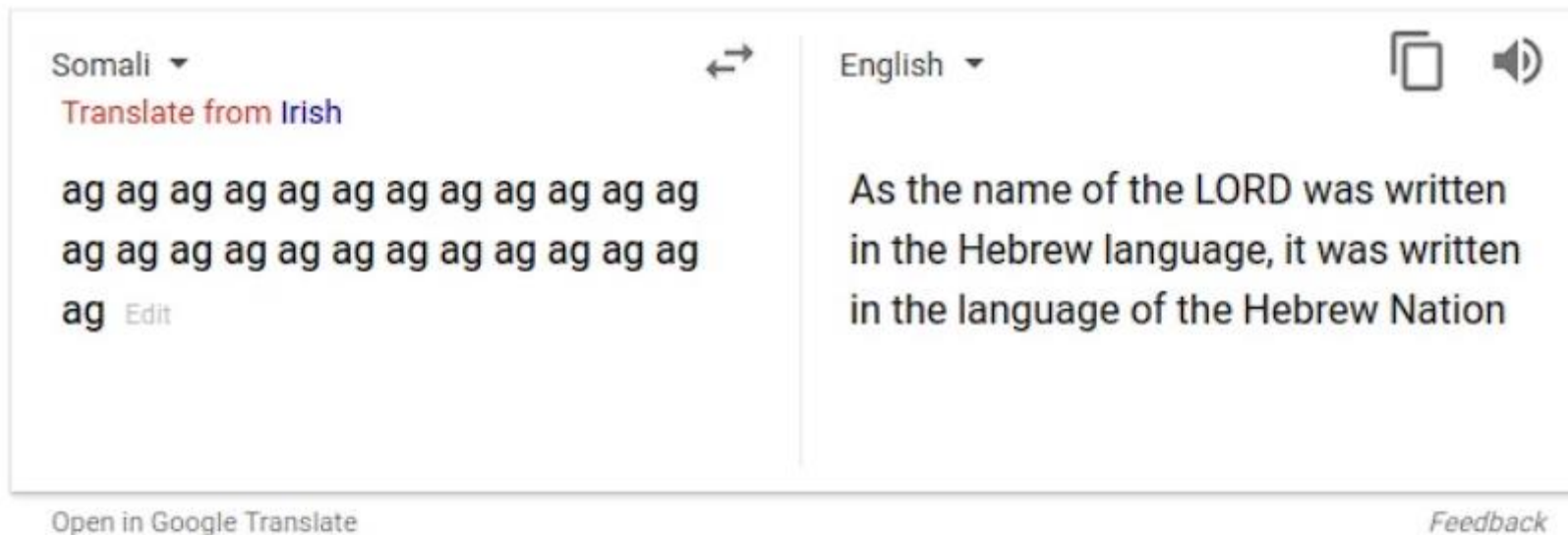


Didn't specify gender

출처: CS224n lecture 7

So, is Machine Translation solved?

- **Black box 모델**이기 때문에 이상한 번역을 해도 이유를 알 수 없다.

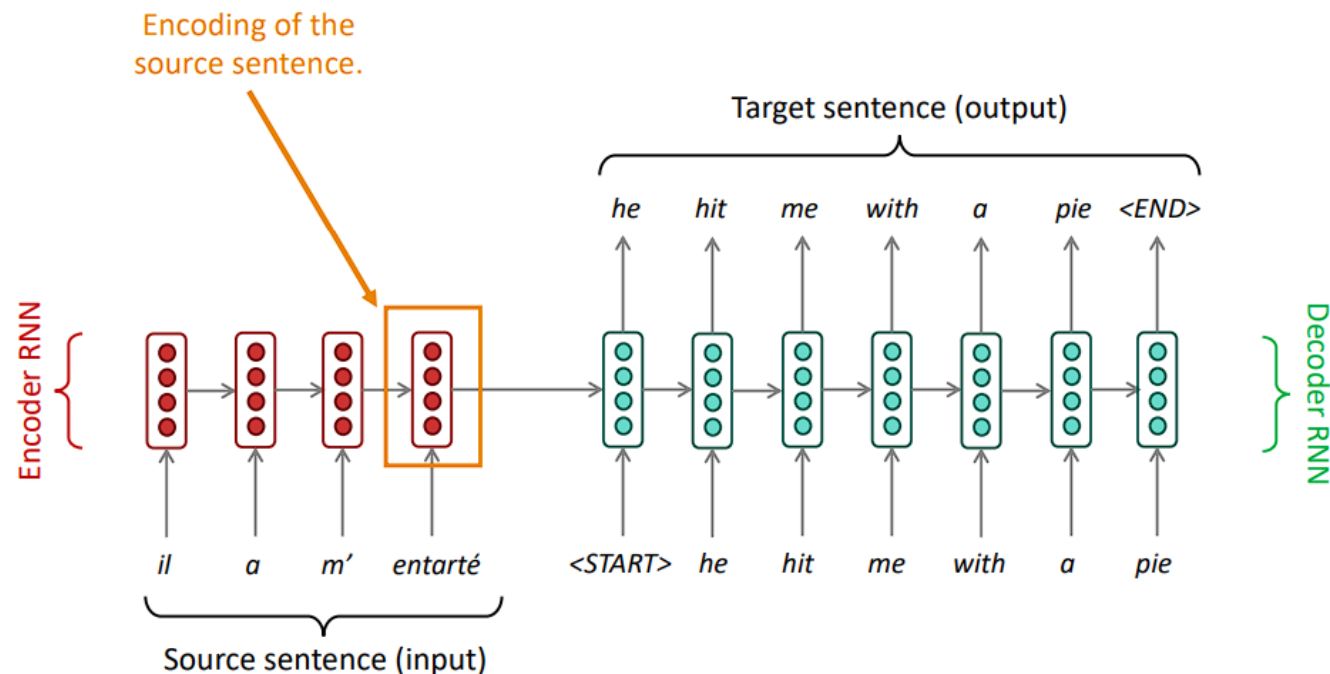


출처: CS224n lecture 7

Attention

Bottleneck problem

- Encoder의 **마지막 hidden state**를 통해 source sentence를 표현한다
 - 하나의 hidden state로 source sentence의 모든 정보를 담을 수 있을까?
 - 문장이 길어질수록 더 힘들 것이다.



출처: CS224n lecture 7

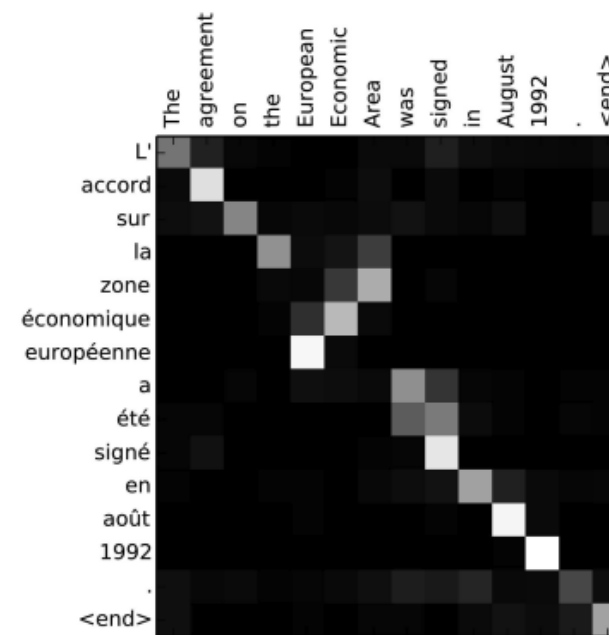
How to Solve Bottleneck problem?

- Attention

- 사전적 의미는 '주목'
- Decoder가 단순히 hidden state만을 받아서 decoding을 하지 않고 매 step마다 Encoder의 모든 hidden state를 살펴보고 어느 부분에 주목해서 Decoding을 할 것인가

- Decoder의 hidden state: Query vector
- Encoder의 hidden state: Key vector & Value vector

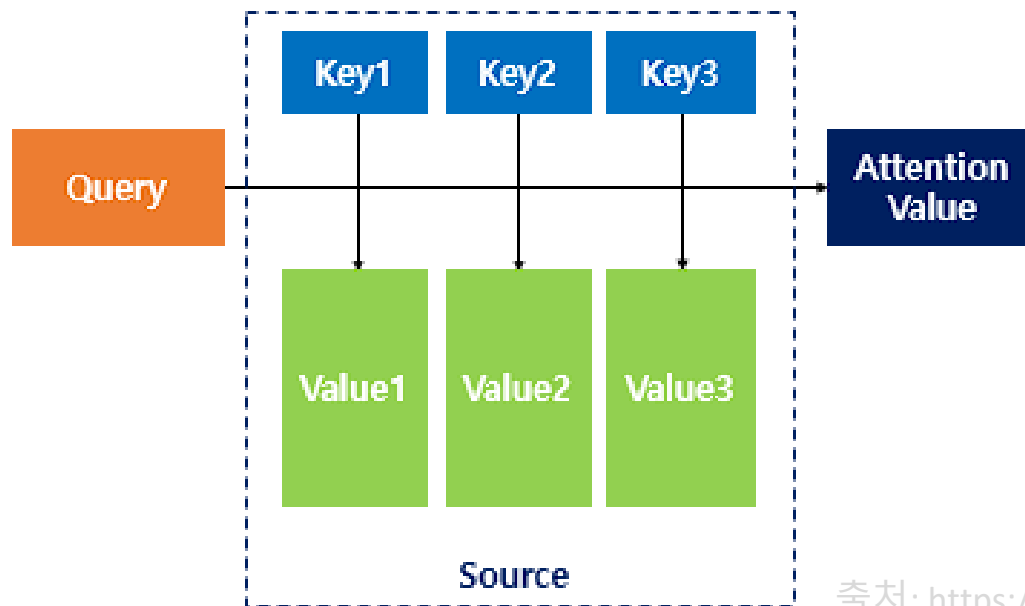
Visualize attention weights $a_{t,i}$



How to Solve Bottleneck problem?

- Attention

- Attention Mechanism에는 Query/Key/Value라는 개념이 등장한다.
- $\text{Attention}(Q, K, V) = \text{Attention Value}$
- Query는 step t 에서 Decoder의 hidden state
- Key와 Value는 모든 Encoder의 hidden states

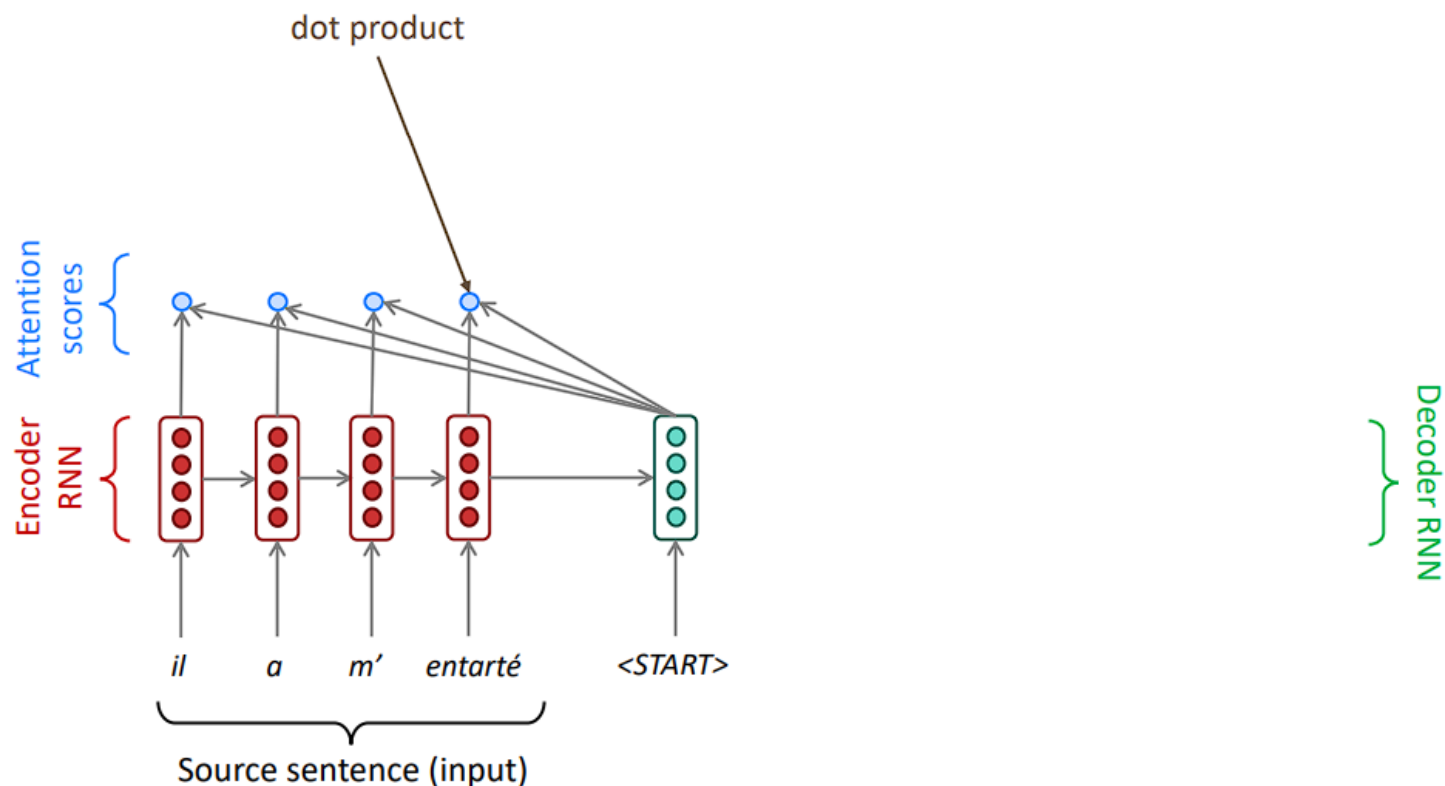


출처: <https://wikidocs.net/22893>

How to Solve Bottleneck problem?

- Attention

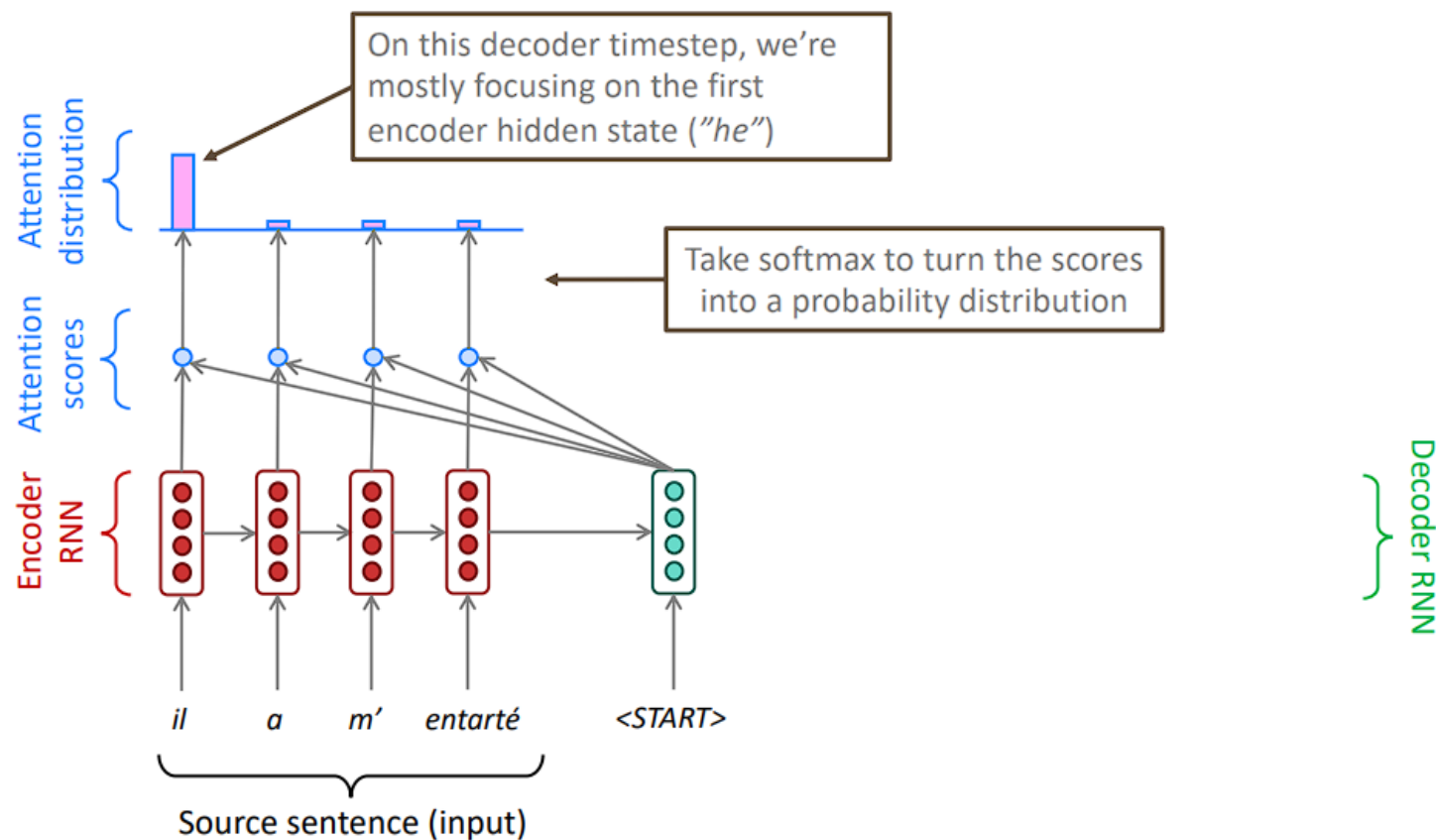
- 현재 step의 hidden state와 Encoder의 모든 hidden state와 Dot product 연산
- Dot product의 값은 각 Encoder hidden state에 대한 Attention score (유사한 정도 측정)



How to Solve Bottleneck problem?

- Attention

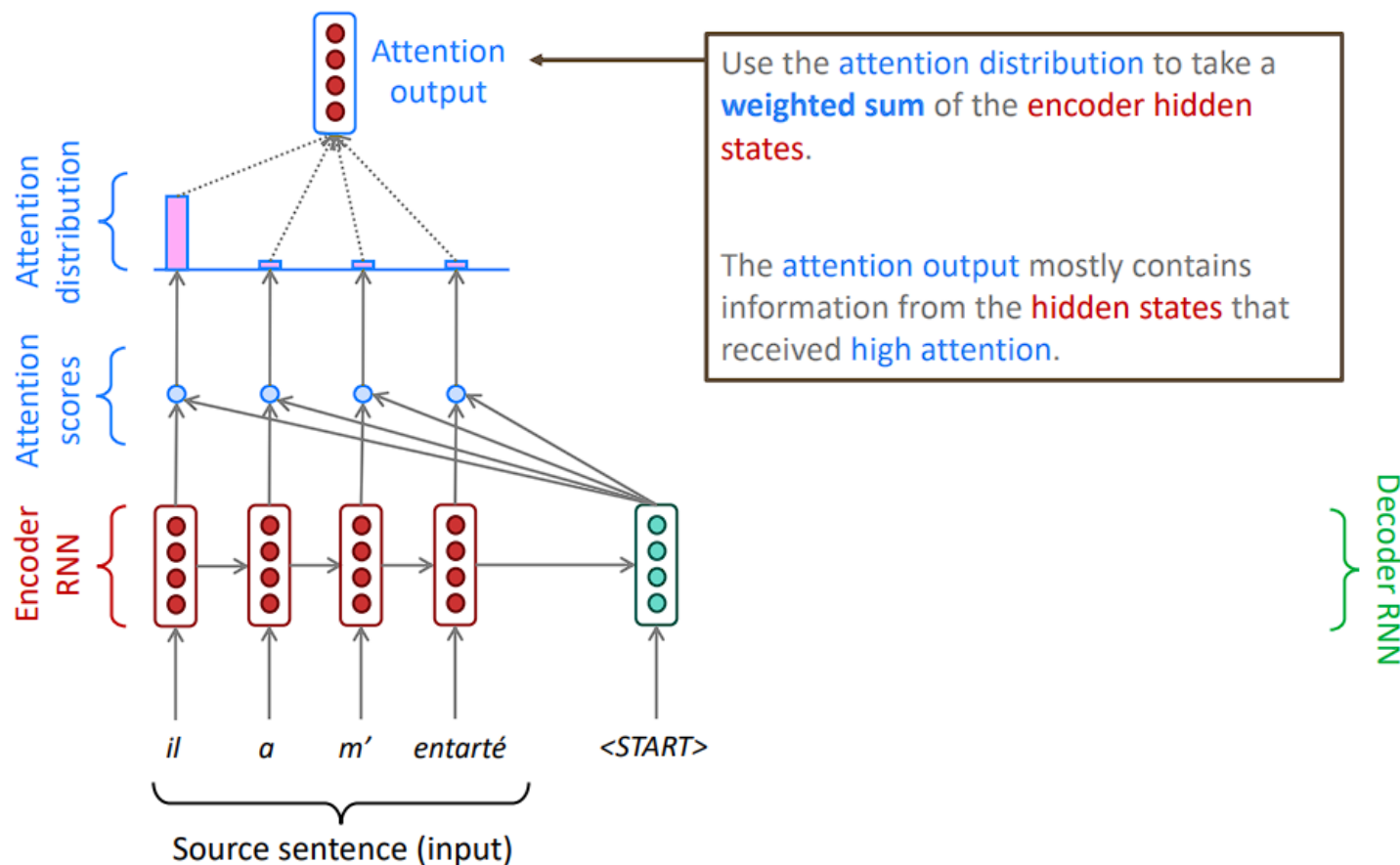
- Attention scores를 softmax 함수에 통과시켜 얻은 확률 값이 Attention distribution



How to Solve Bottleneck problem?

- Attention

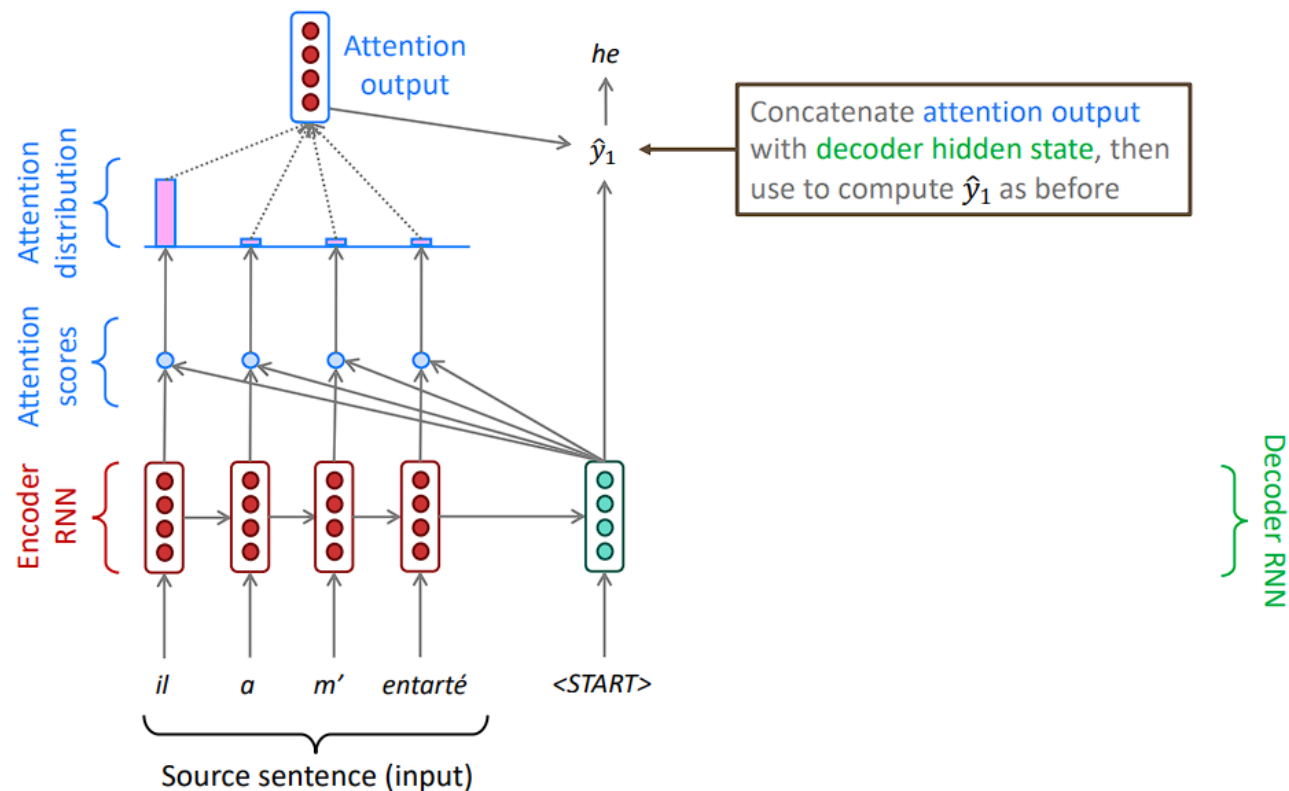
- Attention distribution에 기반한 Encoder hidden state의 Weighted sum이 Attention output



How to Solve Bottleneck problem?

- Attention

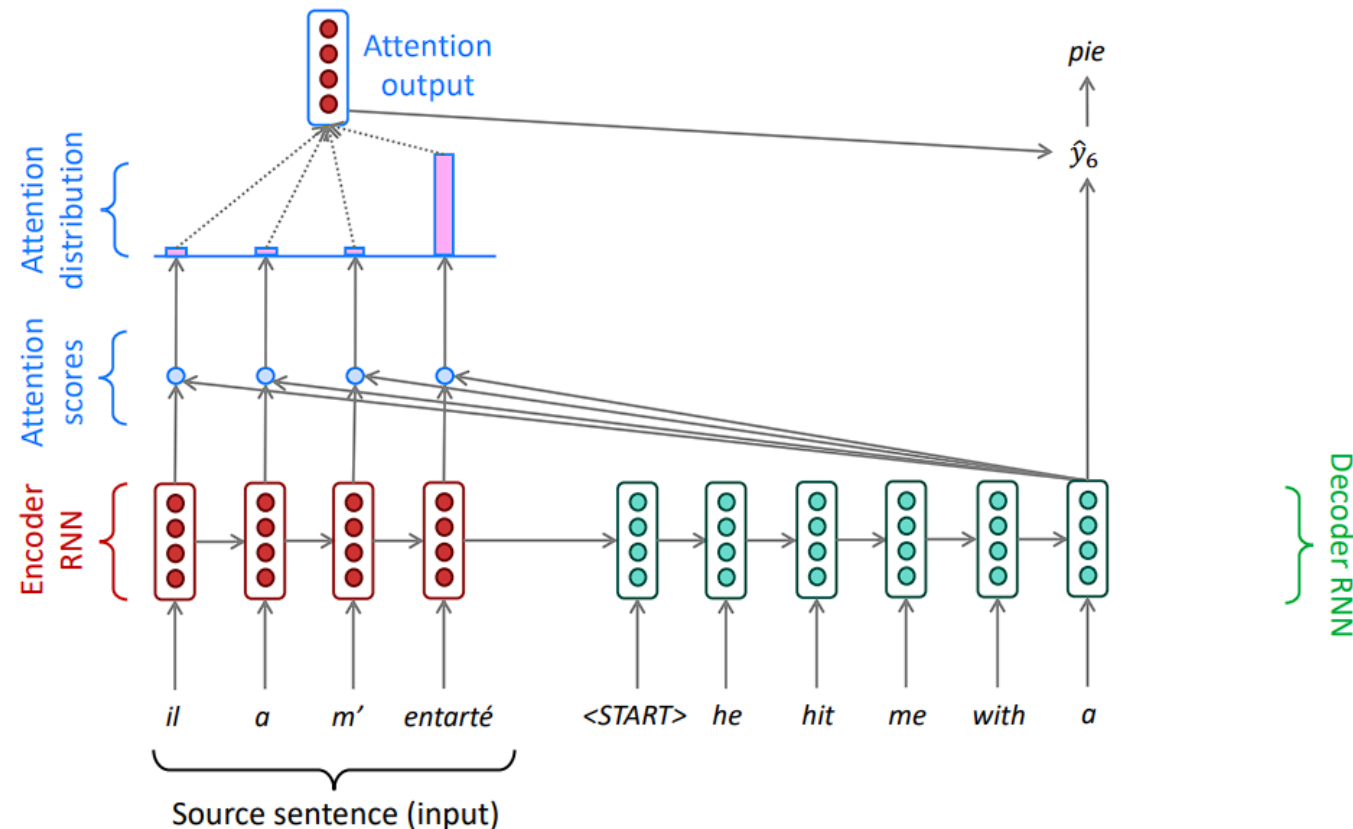
- Attention output과 Decoder의 hidden state를 Concatenation한 벡터를 다시 softmax 함수에 통과시켜 Output words를 예측



How to Solve Bottleneck problem?

- Attention

- 지금까지의 과정을 Decoder의 매 step마다 반복



Attention in equations

- Encoder hidden states: $h_1, \dots, h_N \in \mathbb{R}^h$
- Step t에서의
 - Decoder hidden state: $s_t \in \mathbb{R}^h$
 - Attention scores: $e^t = [s_t^T h_1, \dots, s_t^T h_N] \in \mathbb{R}^N$
 - Attention distribution: $\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$
 - Attention output: $a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$
 - Concatenate attention output with decoder hidden state: $[a_t; s_t] \in \mathbb{R}^{2h}$

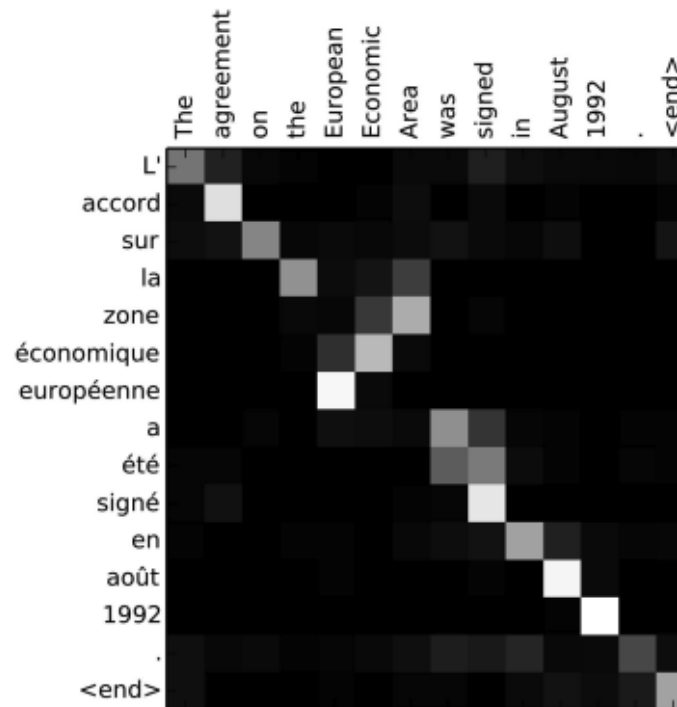
Attention in great

- Attention으로 인해 NMT의 성능이 더 좋아졌다.
- Attention의 작동 원리는 사람이 번역을 하는 방법과 유사하다.
 - Source sentence를 모두 외워서 번역하는 것이 아니라 매 step마다 source sentence를 다시 확인하면서 번역하기 때문이다.
- Attention을 통해 Bottleneck / Vanishing Gradient problem을 해결했다.
 - Decoder의 각 step에서 Encoder의 hidden state로 이어지는 shortcut을 제공하기 때문이다.
- Attention은 Interpretability를 제공한다.
 - 왜 그 부분을 주목하는지는 아직 알 수 없지만 최소한 Decoder의 각 step이 어느 부분을 주목해서 번역하는지, attention distribution을 통해 확인할 수 있다.

Attention in great

- Attention은 **Interpretability**를 제공해준다.
 - 왜 그 부분을 주목하는지는 아직 알 수 없지만 최소한 Decoder의 각 step이 어느 부분을 주목해서 번역하는지, attention distribution을 통해 확인할 수 있다.

Visualize attention weights $a_{t,i}$



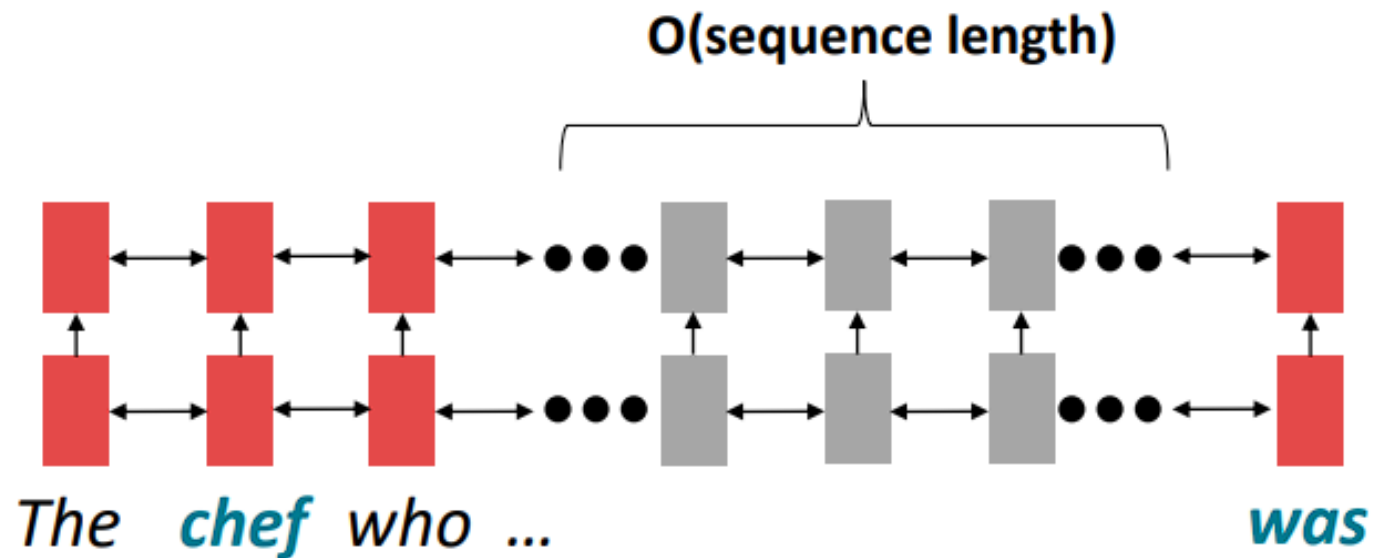
Attention is not only for MT

- MT뿐만이 아니라 전반적인 NLP task와 architecture에 적용할 수 있다.
- Value Vector (Encoder hidden state)와 Query Vector(Decoder hidden state)가 존재할 때, Query vector를 바탕으로 Value Vector의 어느 부분에 주목할 것인지...
- Value vector의 selective summary

Self-Attention

Issues with recurrent models

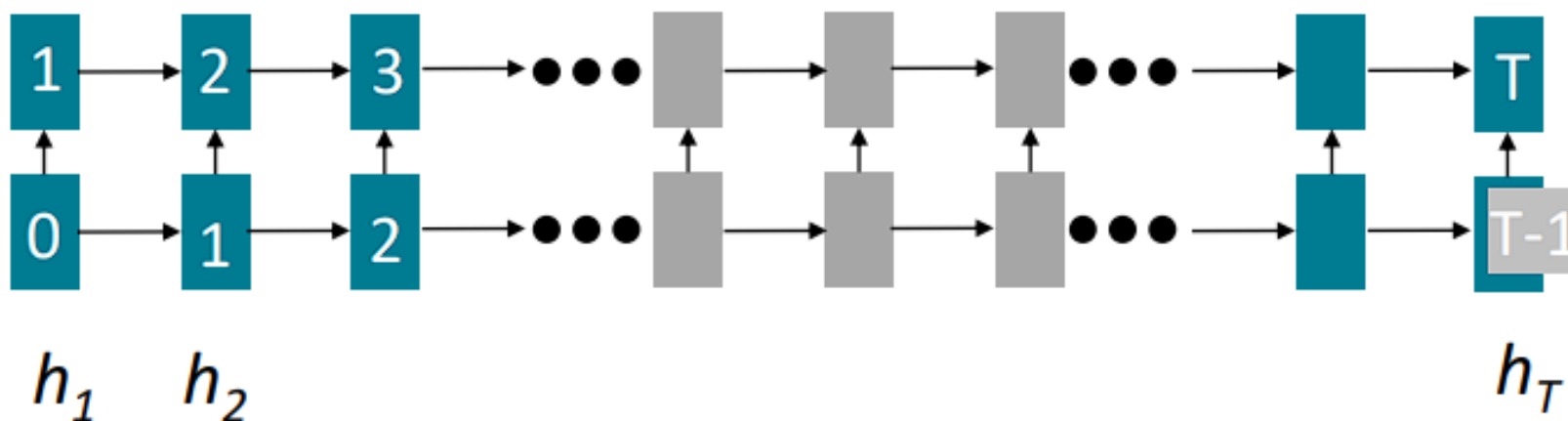
- Linear interaction distance
 - RNN은 왼쪽에서 오른쪽으로 차례차례 연결되는 구조
 - 떨어진 거리만큼 가야 서로 영향을 줄 수 있다
 - 멀리 떨어진 단어 사이의 의존성을 모델이 잘 반영하지 못한다. (Vanishing Gradient)



Issues with recurrent models

- Lack of parallelizability

- h_t 가 h_{t-1} 에 의존하는 구조이기 때문에, 병렬 처리를 할 수 없다.
- 데이터셋이 커질수록 학습 시간이 매우 오래 걸린다

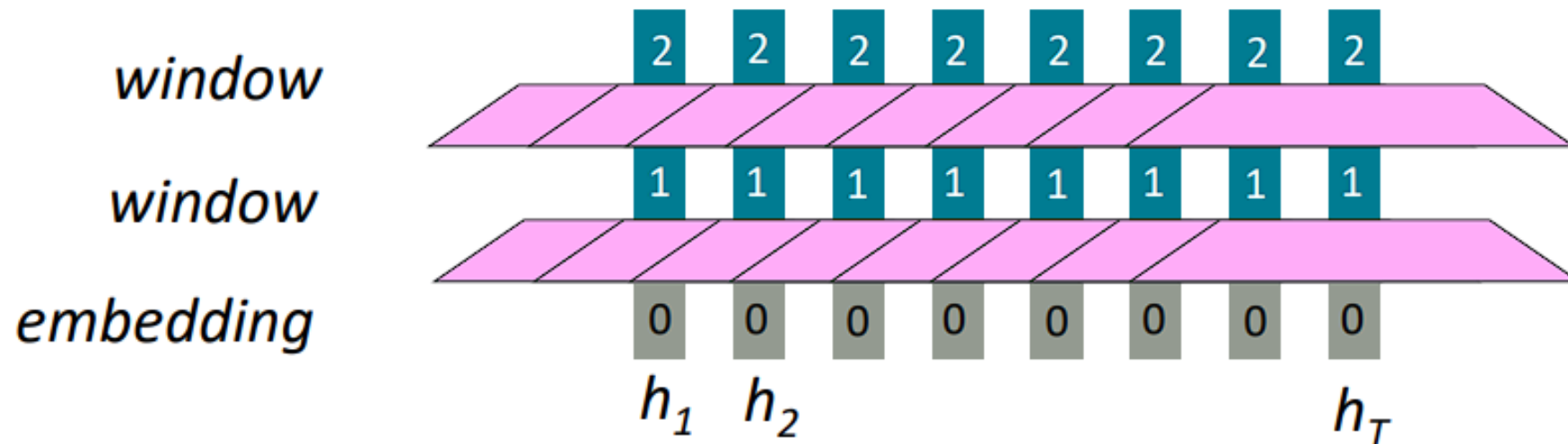


출처: CS224n lecture 7

Solution 1: word windows

- Word Window

- Local context를 파악할 수 있다.
- 서로 다른 window간의 의존성이 없기 때문에 병렬 처리가 가능하다.

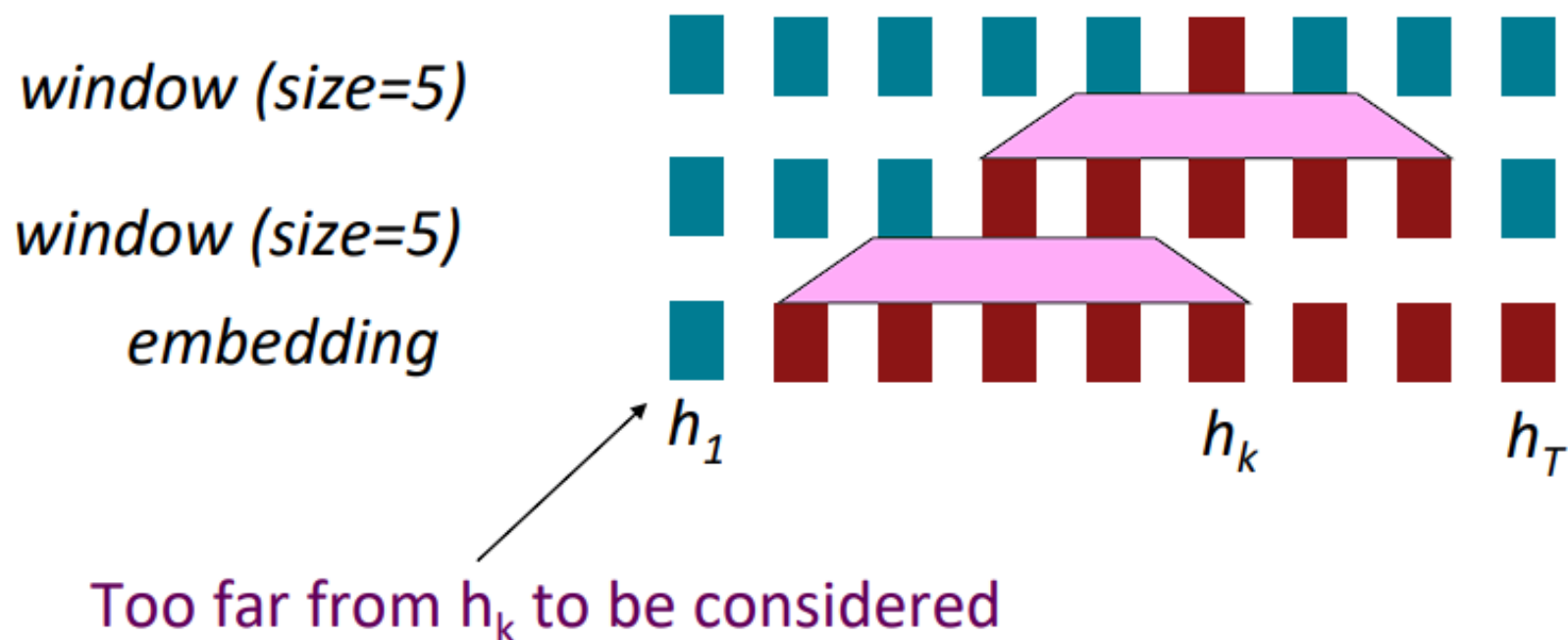


출처: CS224n lecture 7

Solution 1: word windows

- Word Window

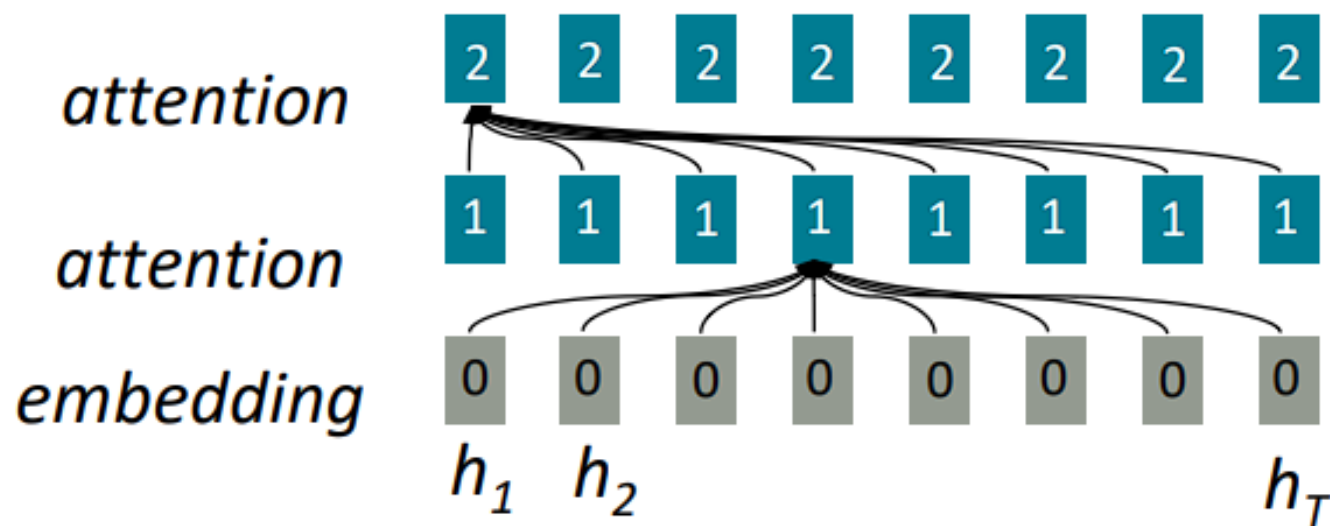
- 멀리 있는 단어들 사이의 관계는 word window를 여러 층으로 쌓으면 파악할 수 있다.
 - 마치 Receptive field (Recap)
- 하지만 여전히 **모든 단어 사이의 관계**를 확인할 수는 없다.



Solution 2: attention

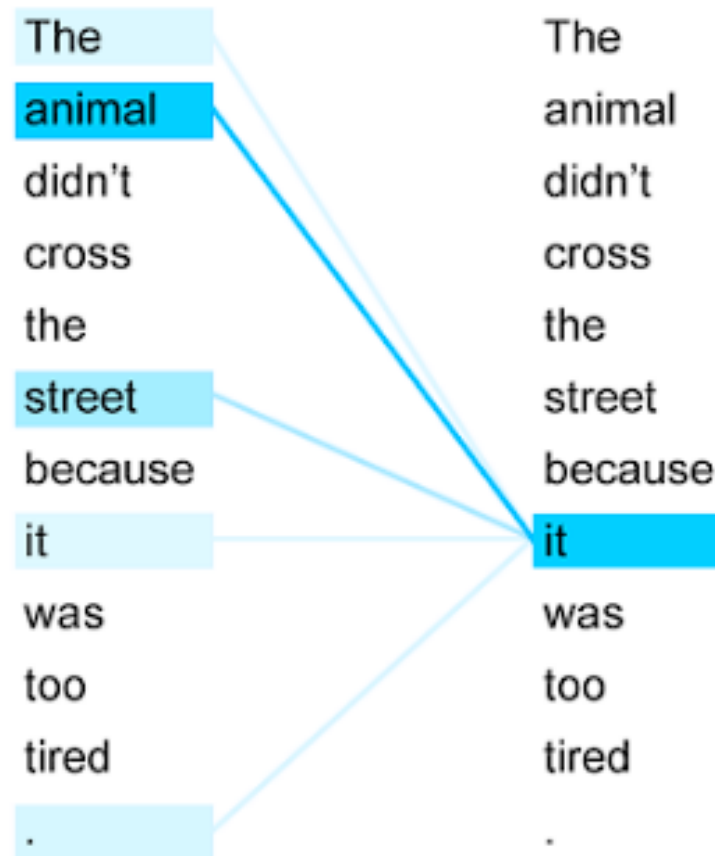
- Attention

- Self-attention: 한 문장 안에서 Attention을 자기 자신에 대해 구하는 것!
- 단어에 대한 Embedding vector만 존재하면 서로가 서로의 Embedding vector를 확인할 수 있으므로 병렬화 가능하다.
- 한 단어가 문장 내의 모든 단어의 Embedding vector를 확인하므로 멀리 떨어진 단어의 관계를 반영할 수 있다.



Solution 2: attention

- 같은 문장 안에서도 다른 부분을 봐야만 이해할 수 있는 단어들이 있다.



출처: <https://wikidocs.net/22893>

Self-Attention

- Queries: q_1, q_2, \dots, q_T . Each query is $q_i \in \mathbb{R}^d$
- Keys: k_1, k_2, \dots, k_T . Each query is $k_i \in \mathbb{R}^d$
- Values: v_1, v_2, \dots, v_T . Each query is $v_i \in \mathbb{R}^d$
- Self-attention에서는 queries, keys, values가 모두 동일하다.

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

Compute **key-query** affinities

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

Compute attention weights from affinities (softmax)

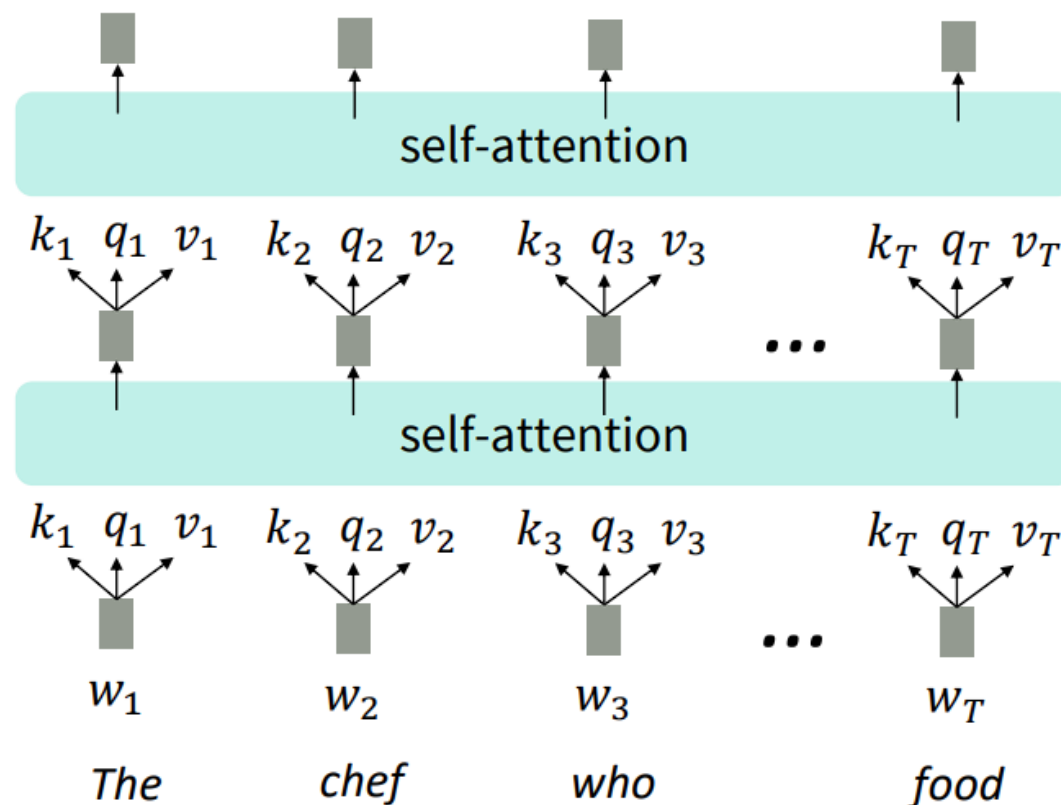
$$\text{output}_i = \sum_j \alpha_{ij} v_j$$

Compute outputs as weighted sum of **values**

출처: CS224n lecture 7

Self-Attention as an NLP building block

- RNN/LSTM에서 layer를 여러 층 쌓듯이, self-attention block을 여러 개 쌓아서 하나의 NLP building block을 형성한다.



출처: CS224n lecture 7

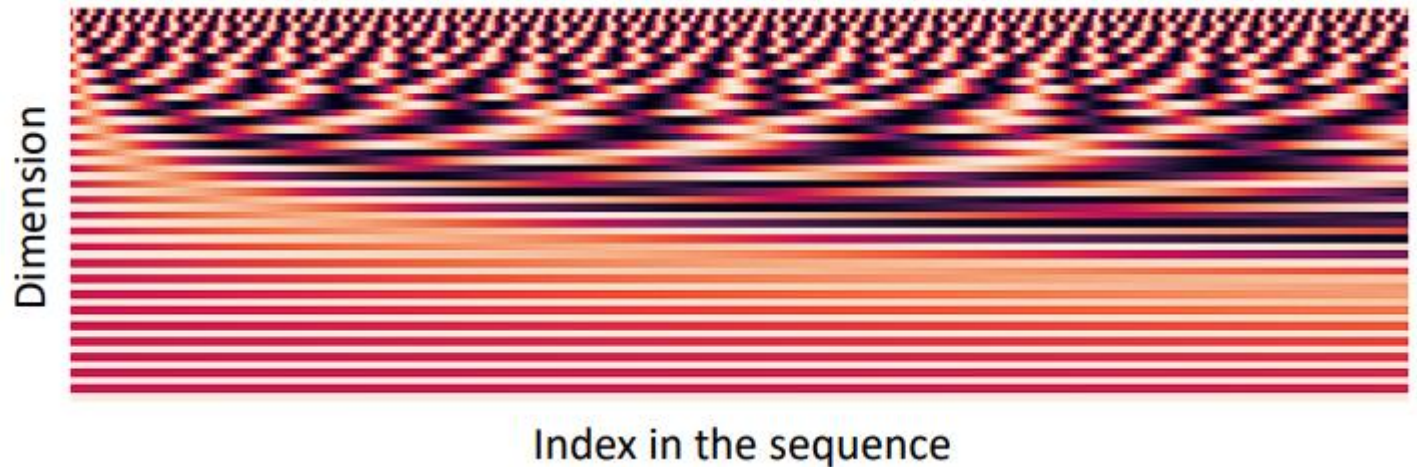
Problems of Self-Attention

- **순서에 대한 정보가 반영되지 않는다.**
 - RNN에서는 이전 step의 hidden state가 넘어오기 때문에 해당 정보가 더 이전의 내용임을 스스로 학습할 수 있다.
 - Self-attention에서는 모든 단어가 각각 동시에 모든 단어에 접근하므로 순서를 알 수가 없다.
- 각 단어의 Embedding vector에 **순서 정보를 담고 있는 vector**를 더해주자
 - $v_i = \tilde{v}_i + p_i$
 - $k_i = \tilde{k}_i + p_i$
 - $q_i = \tilde{q}_i + p_i$

Problems of Self-Attention

- Sinusoidal position representations

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



출처: CS224n lecture 7

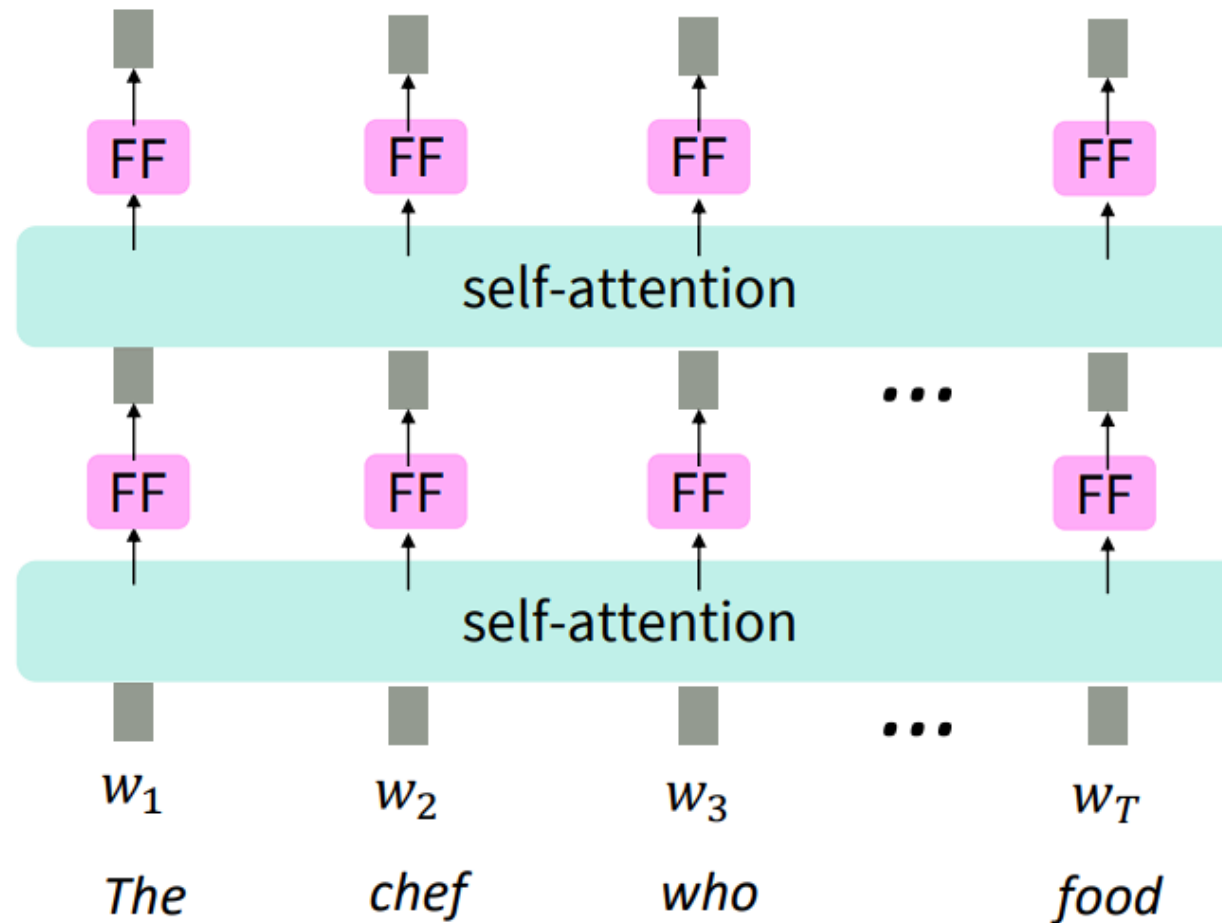
- More about Positional Encoding...
 - <https://www.blossominkyung.com/deeplearning/transfomer-positional-encoding>

Problems of Self-Attention

- **Nonlinearity**가 존재하지 않는다.
 - 딥러닝이 가능했던 이유는 선형적으로 task를 처리하는 것에서 벗어나 Nonlinearity를 추가했기 때문이다.
 - Self-attention은 단순히 **value vector들의 weighted sum**이다.
- 각 layer의 output vector에 **feed-forward network**를 추가한다.
 - $m_i = MLP(output_i) = W_2 * ReLU(W_1 \times output_i + b_1) + b_2$

Problems of Self-Attention

- Feed-forward network

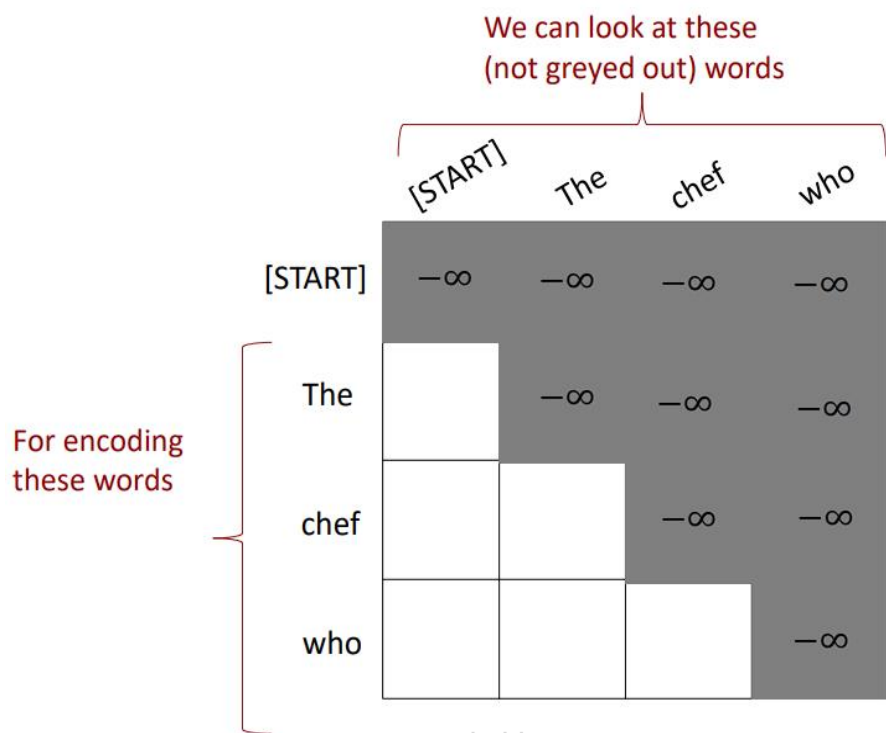


Problems of Self-Attention

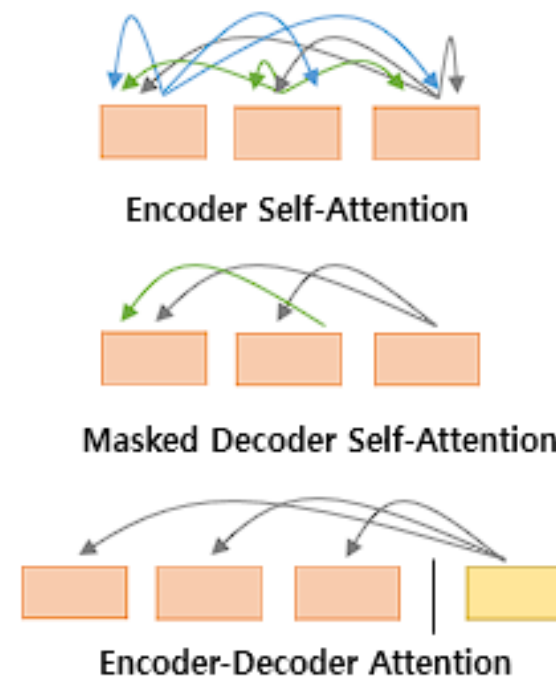
- Decoder는 미래를 보고 Decoding을 하면 안 된다.
 - 미래에 나올 단어가 무엇인지 안다면 Decoding을 하는 의미가 없다.
- 각 step에서 해당 step이후에 대한 정보를 $-\infty$ 로 masking한다.
 - $$e_{ij} = \begin{cases} q_i^T k_j, j < i \\ -\infty, j \geq i \end{cases}$$
 - $-\infty$ 의 값을 가지면 softmax 함수를 통과할 때 $e^{-\infty}$ 이 되면서 0으로 처리된다.

Problems of Self-Attention

- Decoder는 미래를 보고 Decoding을 하면 안 된다.
 - 미래에 나올 단어가 무엇인지 안다면 Decoding을 하는 의미가 없다.



출처: CS224n lecture 7



출처: <https://wikidocs.net/22893>

Transformer

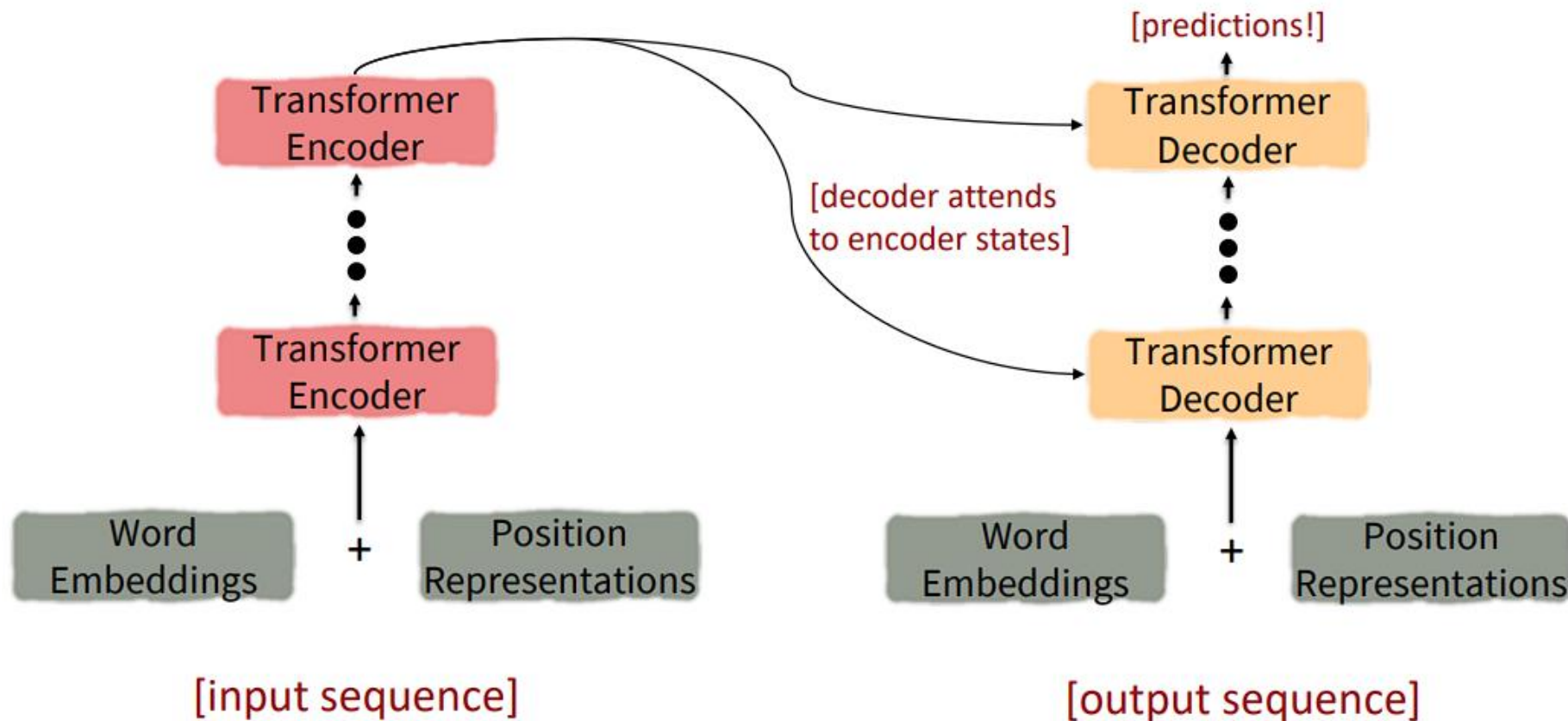
Attention Is All You Need



출처: <https://blog.promedius.ai/transformer/>

The Transformer Encoder-Decoder

- Transformer의 Encoder와 Decoder의 기본 구조

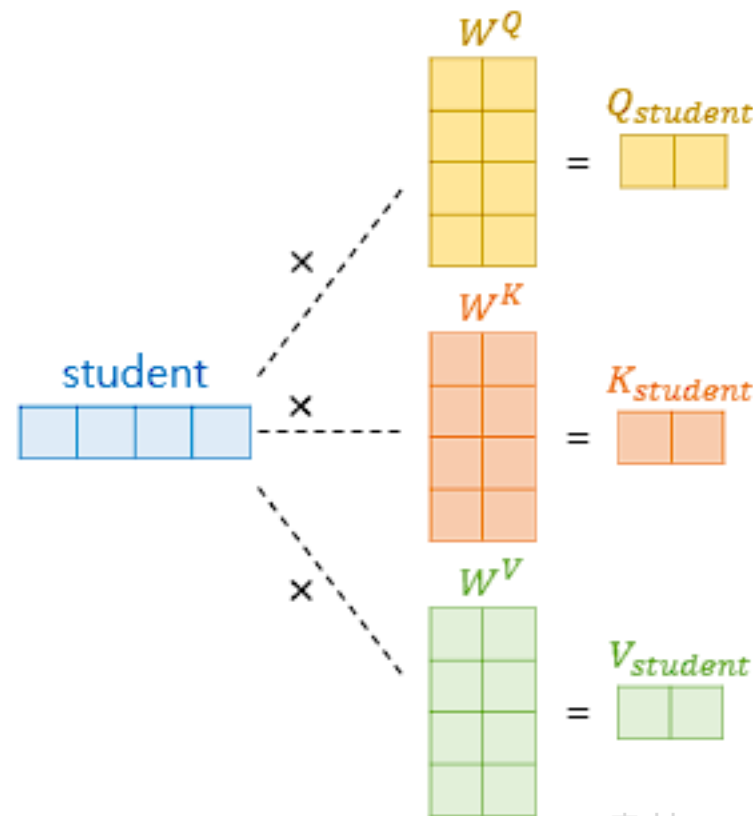


The Transformer Encoder: Key-Query-Value Attention

- Transformer Encoder의 input은 $x_1, \dots, x_T; x_i \in \mathbb{R}^d$
 - Embedding Vector + Positional Encoding
- Self-Attention에서 Key, Query, Value는 전부 같은 값을 사용한다고 했는데, 어떤 목적으로 사용하는지에 따라서 input vector의 어떤 부분이 사용되고 강조되는지가 달라진다.
 - $k_i = Kx_i; K \in \mathbb{R}^{d \times d}$: key matrix
 - $q_i = Qx_i; Q \in \mathbb{R}^{d \times d}$: query matrix
 - $v_i = Vx_i; V \in \mathbb{R}^{d \times d}$: value matrix
 - 새로운 matrix를 도입해서 input vector의 서로 다른 부분에 집중하도록 하는 것

The Transformer Encoder: Key-Query-Value Attention

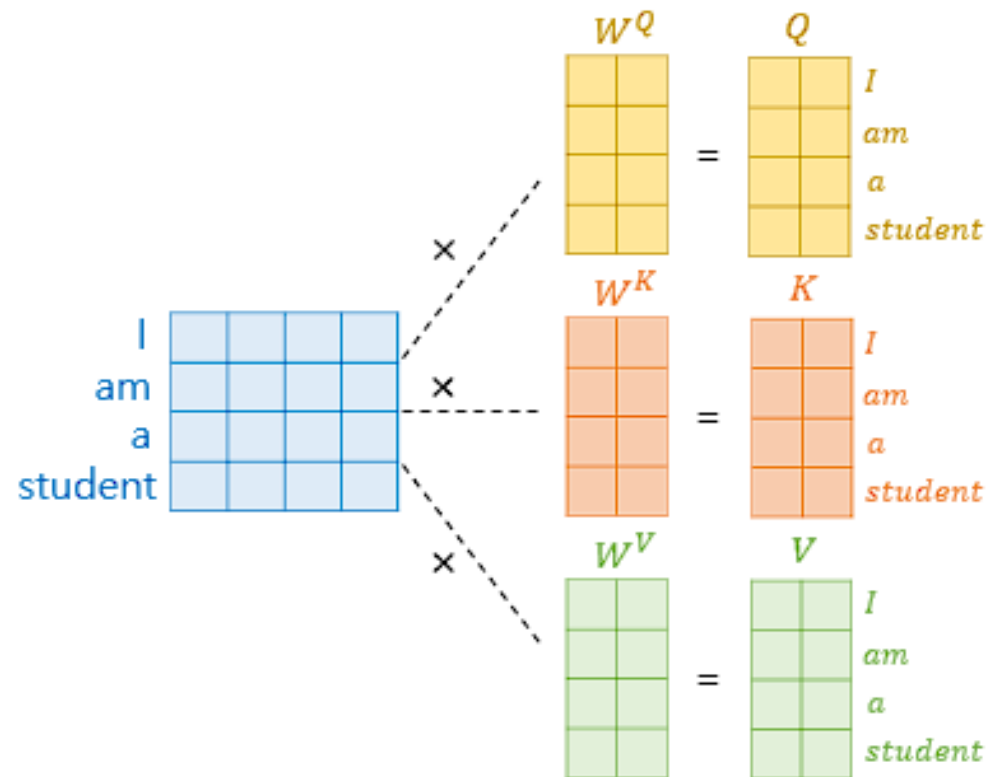
- Self-Attention에서 Key, Query, Value는 전부 같은 값을 사용한다고 했는데, **어떤 목적으로 사용하는지**에 따라서 input vector의 어떤 부분이 사용되고 강조되는지가 달라진다.



출처: <https://wikidocs.net/22893>

The Transformer Encoder: Key-Query-Value Attention

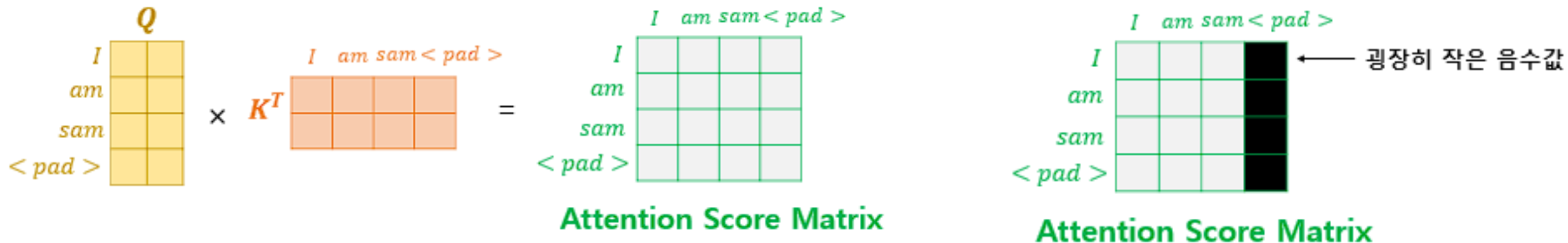
- 각 단어에 대해서 Attention을 구하기 위해 벡터 연산을 하는 것이 아니라 행렬 연산을 사용하면 일괄 계산이 가능하다.



출처: <https://wikidocs.net/22893>

The Transformer Encoder: Key-Query-Value Attention

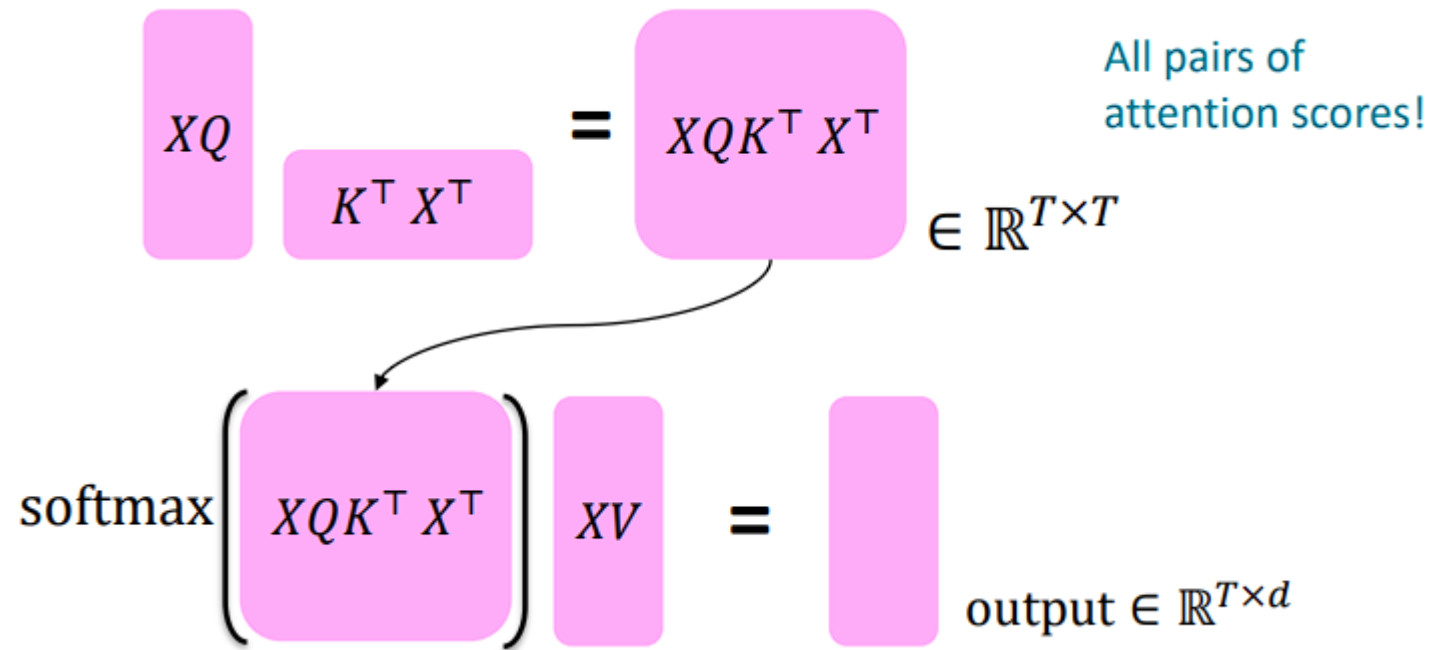
- Masking: input 문장의 길이가 짧을 경우 나머지 부분을 <PAD> 토큰으로 채워준다.



출처: <https://wikidocs.net/22893>

The Transformer Encoder: Key-Query-Value Attention

- Query와 Key를 dot product
 - $XQ(XK)^T \in \mathbb{R}^{T \times T}$
- Softmax에 통과시킨 뒤에 Value와 matrix multiplication
 - $\text{softmax}(XQ(XK)^T) \times XV$



출처: CS224n lecture 7

The Transformer Encoder: Multi-headed attention

- 기존에는 attention score가 가장 높은 부분에만 주목했다.
- 입력 Sequence에 여러 부분에 집중하도록 해보자.
 - Attention head라는 것을 여러 개 만들어서, attention이 입력 Sequence의 여러 부분에 집중하도록

Which do you like better, coffee or tea?

- 문장 타입에 집중하는 어텐션

Which do you like better, coffee or tea?

- 명사에 집중하는 어텐션

Which do you like better, coffee or tea?

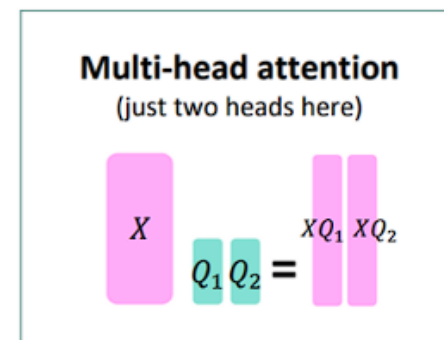
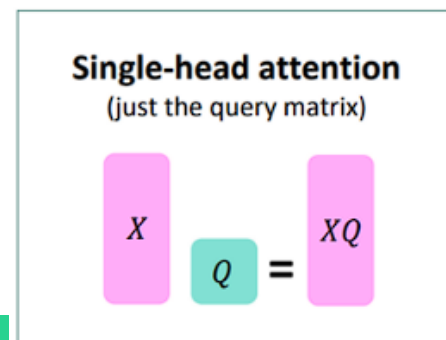
- 관계에 집중하는 어텐션

Which do you like better, coffee or tea?

- 강조에 집중하는 어텐션

The Transformer Encoder: Multi-headed attention

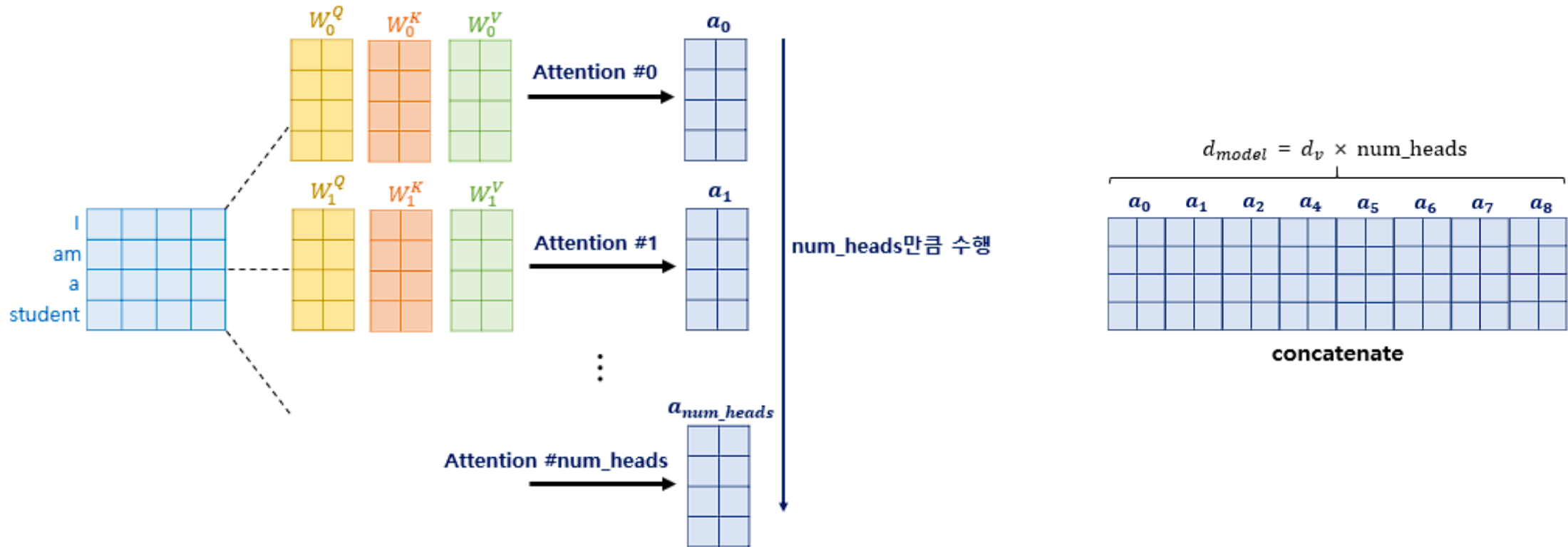
- 하나의 문장에 대해서 h 개의 관점에서 보고 싶다면 K, Q, V matrix를 h 개로 나눈다.
 - $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, h 는 attention heads의 개수가 되고 ℓ 은 $1 \sim h$ 사이의 값을 가진다.
- 각 attention head에 대해서 attention 계산을 각각 진행한다.
 - $output_\ell = softmax(XQ_\ell K_\ell^T X^T) * XV_\ell$; $output_\ell \in \mathbb{R}^{\frac{d}{h}}$
- 최종 output은 각 attention head의 output을 합친다.
 - $output = Y[output_1; \dots; output_h]$; $Y \in \mathbb{R}^{d \times d}$
- 연산량은 기존 Attention(Single-head attention)과 동일하다.



출처: CS224n lecture 7

The Transformer Encoder: Multi-headed attention

- 입력 Sequence에 여러 부분에 집중하도록 해보자.
 - Attention head라는 것을 여러 개 만들어서, attention이 입력 Sequence의 여러 부분에 집중하도록

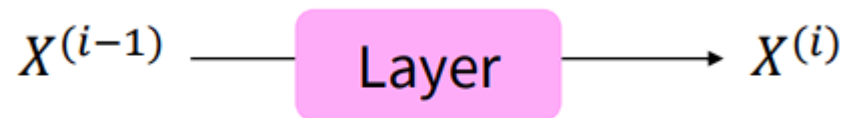


출처: <https://wikidocs.net/22893>

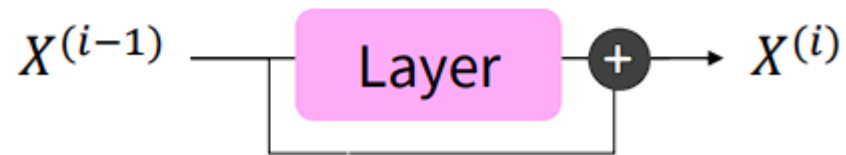
The Transformer Encoder: Residual connections

- Transformer에서도 ResNet에서 소개한 Residual connections를 이용한다.
 - Residual connections은 Loss 함수를 더 smooth한 형태로 만들어 쉽게 학습이 가능하다.

- Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)



- We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn “the residual” from the previous layer)



출처: CS224n lecture 7

The Transformer Encoder: Layer Normalization

- Layer Normalization도 모델에 상관없이 자주 사용되는 technique이다.
 - 자세한 내용은 다음주에 배울 것!
 - Normalization을 통해 불필요한 분산 정보를 제거해 학습을 안정화하는 방법

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

Normalize by scalar mean and variance

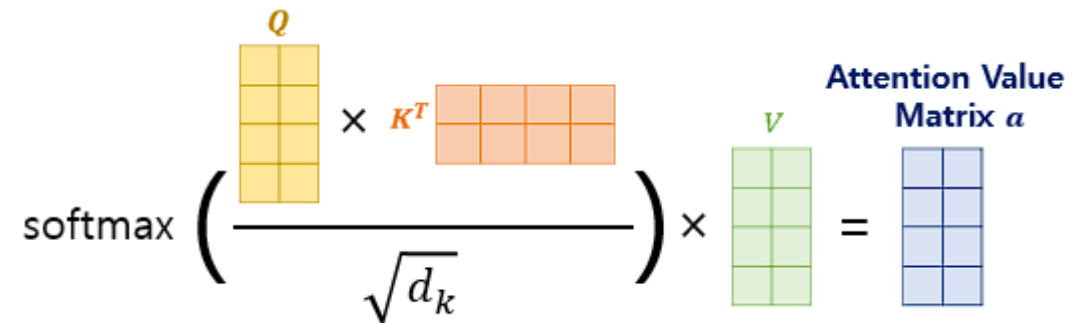
Modulate by learned elementwise gain and bias

출처: CS224n lecture 7

The Transformer Encoder: Scaled Dot Product

- Scaled Dot Product는 attention을 구하는 과정에서 attention score가 너무 커지는 것을 방지하기 위해 attention score를 $\sqrt{d/h}$ 로 나눠주는 것이다.
 - Attention score가 커지는 것이 왜 문제가 될까?
 - 값이 하나만 지나치게 크면 softmax 함수를 통과하면 확률이 1에 가깝다. 나머지는 gradient update가 굉장히 적게 일어난다.

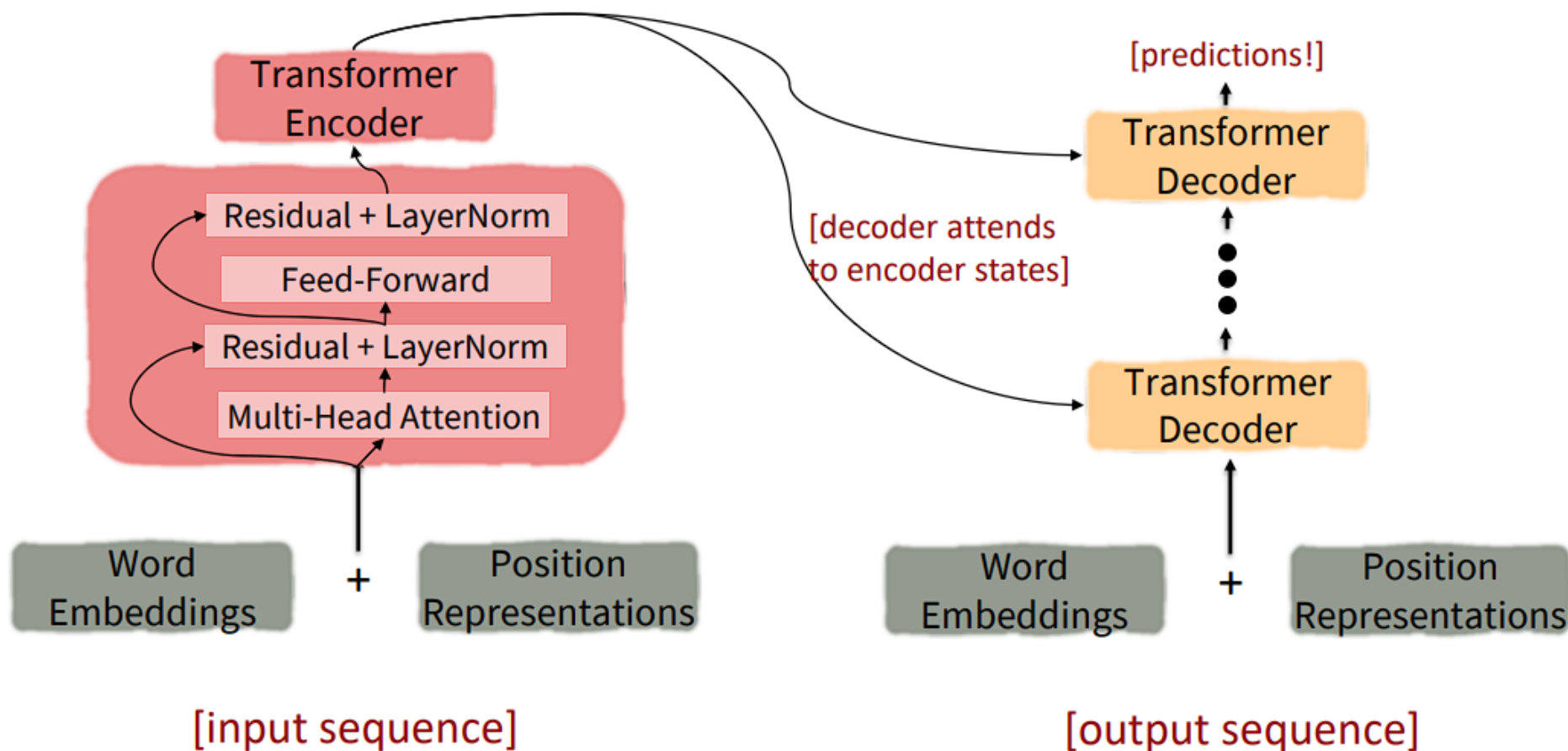
$$\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^T X^T}{\sqrt{d/h}}\right) * XV_\ell$$



출처: <https://wikidocs.net/22893>

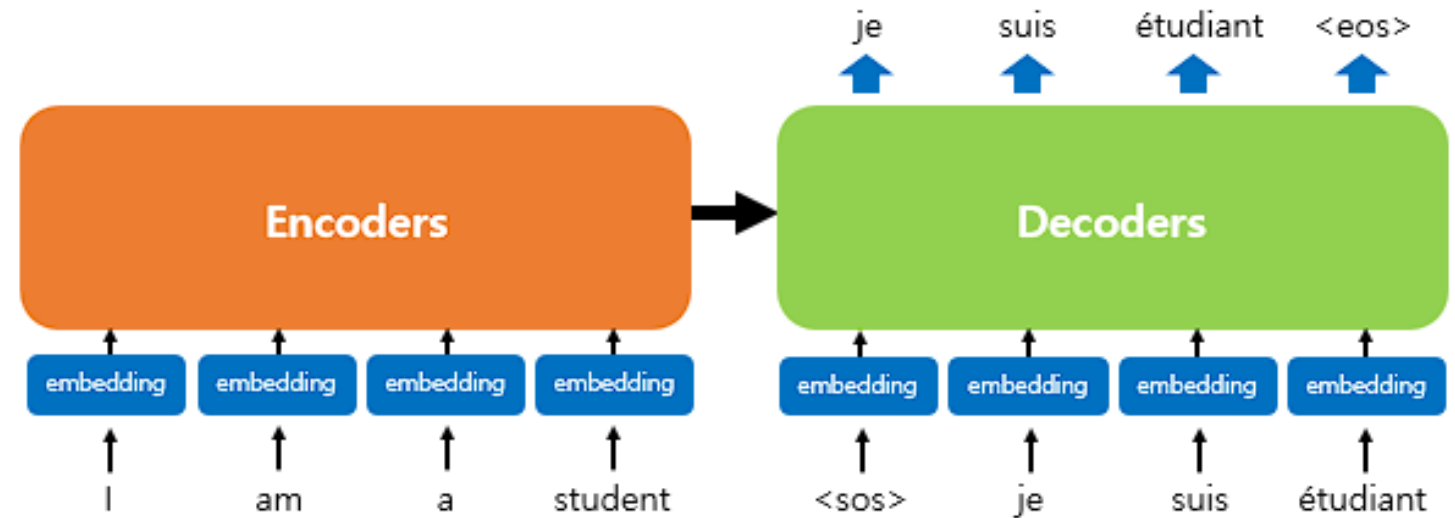
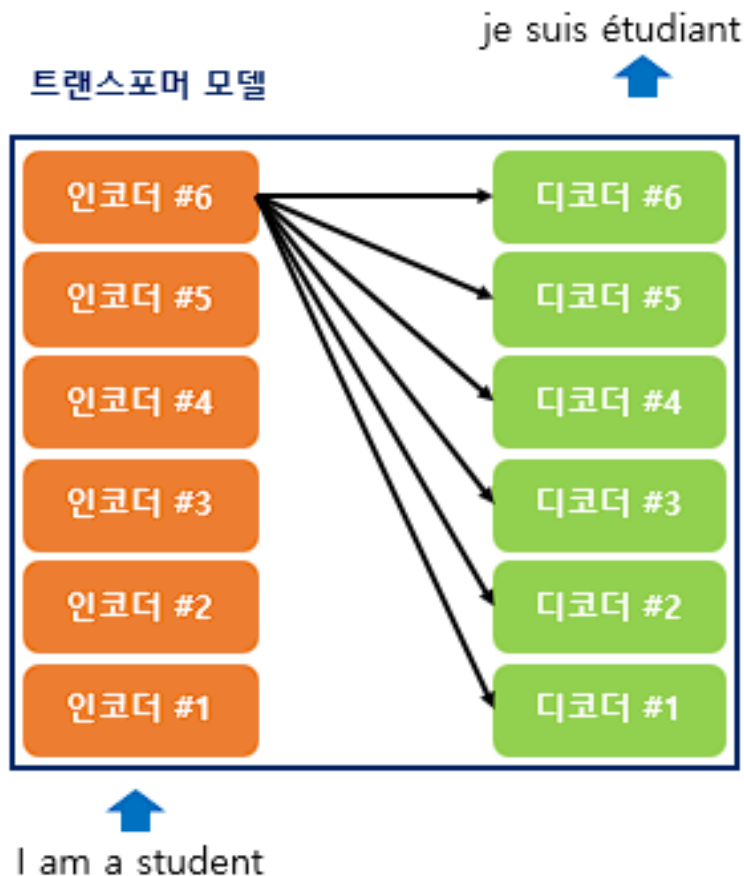
The Transformer Encoder-Decoder

- 지금까지 배운 개념을 적용하면 Transformer의 Encoder의 구조는 다음과 같다.



The Transformer Encoder-Decoder

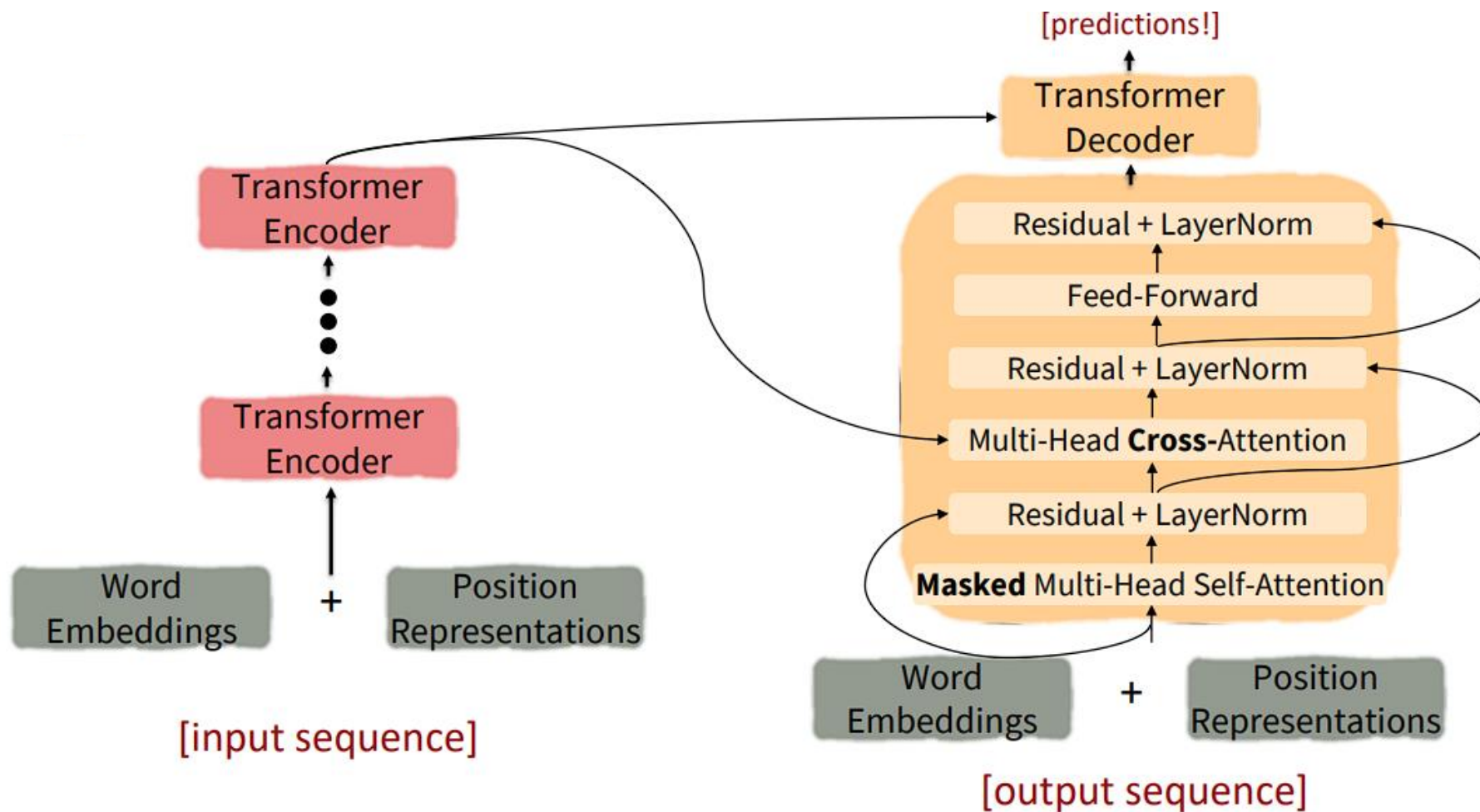
- 지금까지 배운 개념을 적용하면 Transformer의 Encoder의 구조는 다음과 같다.



출처: <https://wikidocs.net/22893>

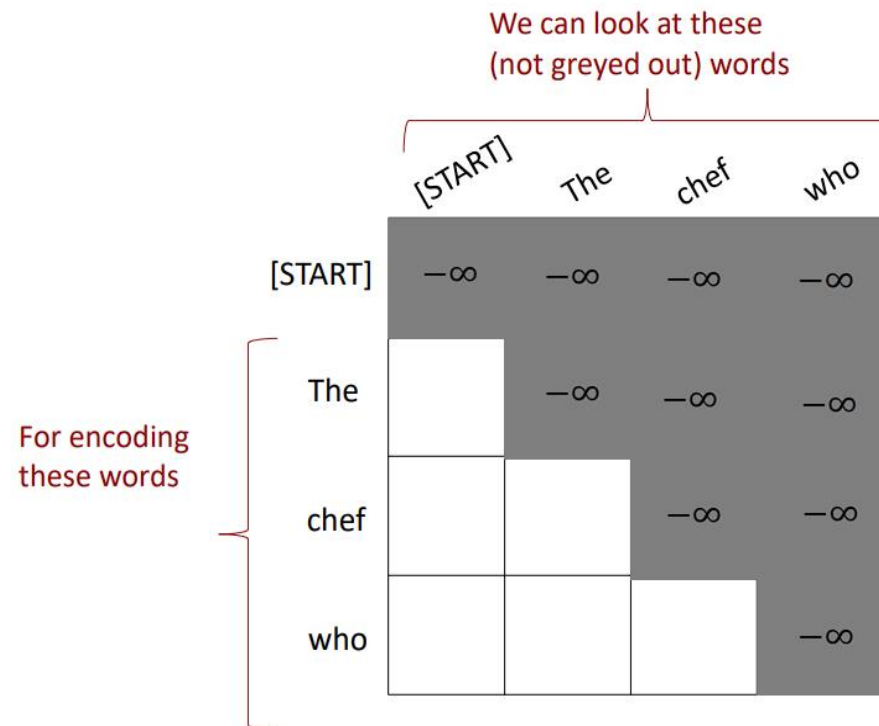
The Transformer Encoder-Decoder

- 다음은 Decoder의 구조이다. 아직 안 배운 개념들을 마저 배워보자.



The Transformer Decoder: Masked Self-Attention

- Decoder는 미래를 보고 Decoding을 하면 안 된다.
 - 미래에 나올 단어가 무엇인지 안다면 Decoding을 하는 의미가 없다.

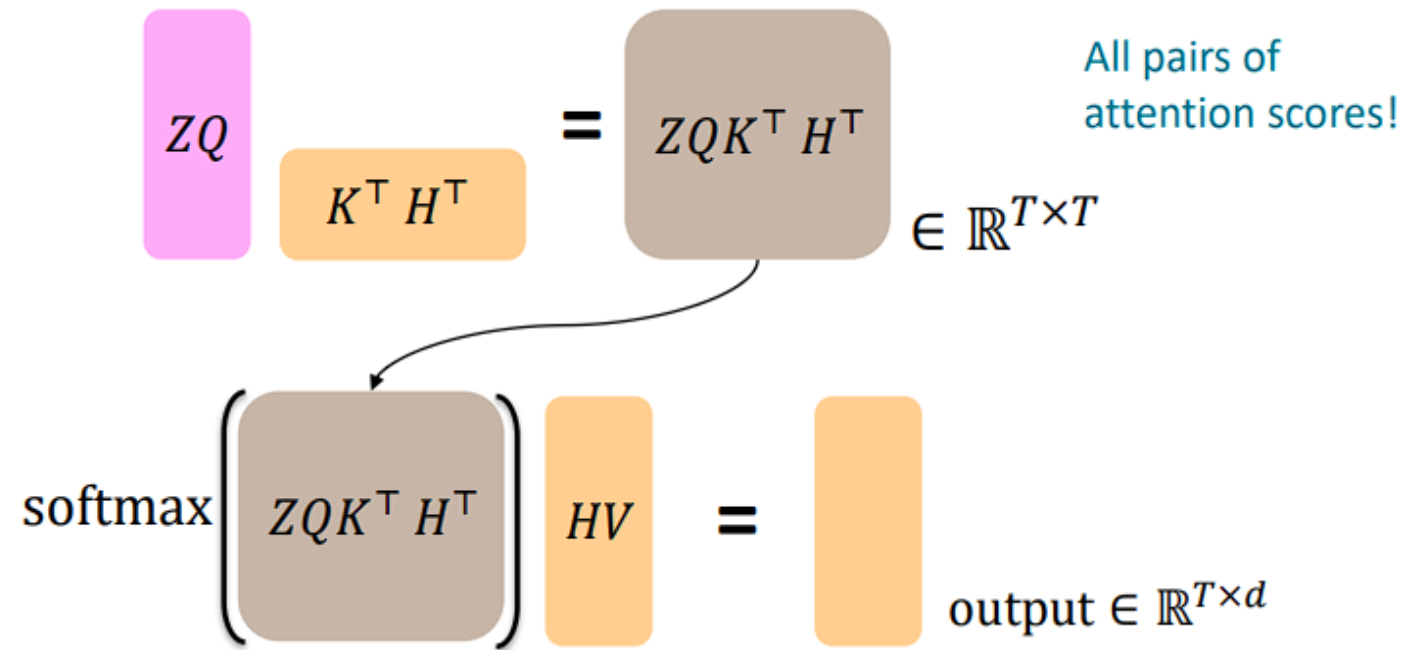


The Transformer Decoder: Cross-attention

- Decoder는 self-attention이 아니라 Encoder의 내용도 확인해야 한다.
 - $h_1, \dots, h_T; h_i \in \mathbb{R}^d$: Transformer Encoder의 output vector
 - $z_1, \dots, z_T; z_i \in \mathbb{R}^d$: Transformer Decoder의 input vector
- Query Vector는 Decoder로부터 생성한다.
 - $q_i = Qz_i$
- Key Vector와 Value Vector는 Encoder로부터 생성한다.
 - $k_i = kh_i, v_i = Vh_i$
- 처음 Seq2Seq 모델에서 attention을 활용하는 방법과 비슷하다!

The Transformer Decoder: Cross-attention

- Query와 Key를 dot product
 - $ZQ(HK)^T \in \mathbb{R}^{T \times T}$
- Softmax에 통과시킨 뒤에 Value와 matrix multiplication
 - $\text{softmax}(ZQ(HK)^T) \times HV \in \mathbb{R}^{T \times d}$



출처: CS224n lecture 7

Great Results with Transformers

First, Machine Translation from the original Transformers paper!

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

출처: CS224n lecture 7

Great Results with Transformers

Next, document generation!

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

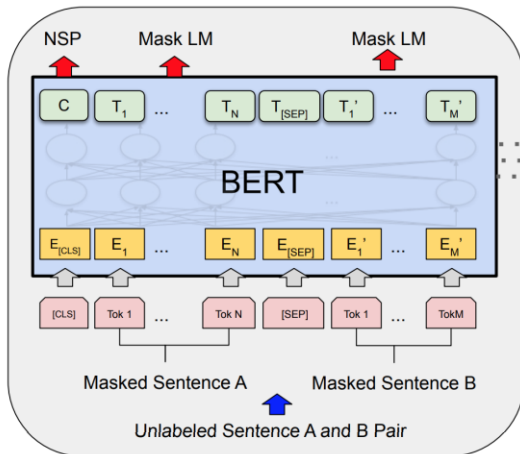
The old standard

Transformers all the way down.

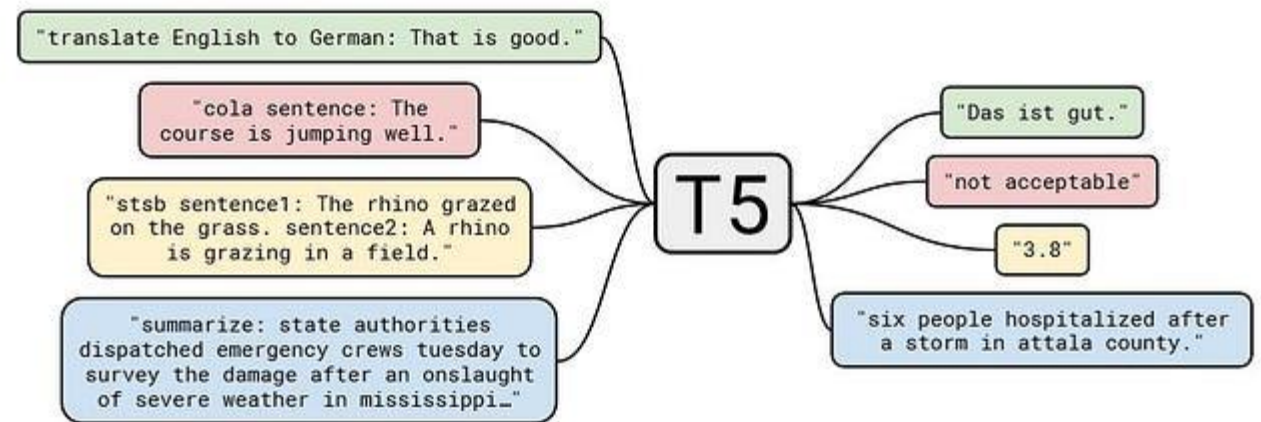
출처: CS224n lecture 7

Why Transformer has Great Results?

- 이처럼 Transformer가 좋은 성능을 보여주는 이유 중 하나는 병렬화를 통해 빠르게 Pretraining을 할 수 있기 때문이다.
- Pretraining에도 여러가지 종류가 있는데 그 방법과 여러가지 모델에 대해서는 다음에 배우게 될 기회가 (아마) 있을 겁니다!



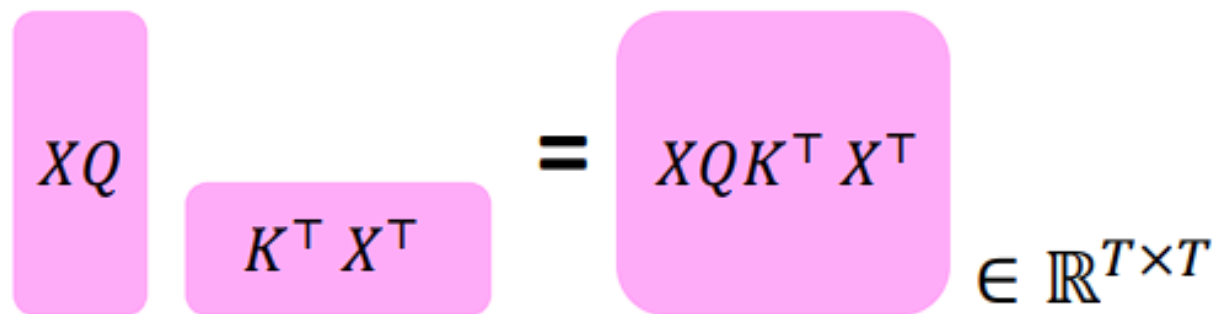
출처: <https://etc.cuit.columbia.edu/news/basics-language-modeling-transformers-bert>
@ Korea Univ.



출처: <https://pub.towardsai.net/create-indonesian-recipe-generator-by-fine-tuning-t5-bart-and-gpt-2-a7fc0551190e>

Drawbacks of transformer

- Quadratic computation
 - Attention의 연산량은 Sequence 길이의 제곱에 비례한다.
 - RNN은 선형적으로 비례한다는 점을 고려하면 문장이 길어질수록 비용이 많이 드는 문제점이 있다.

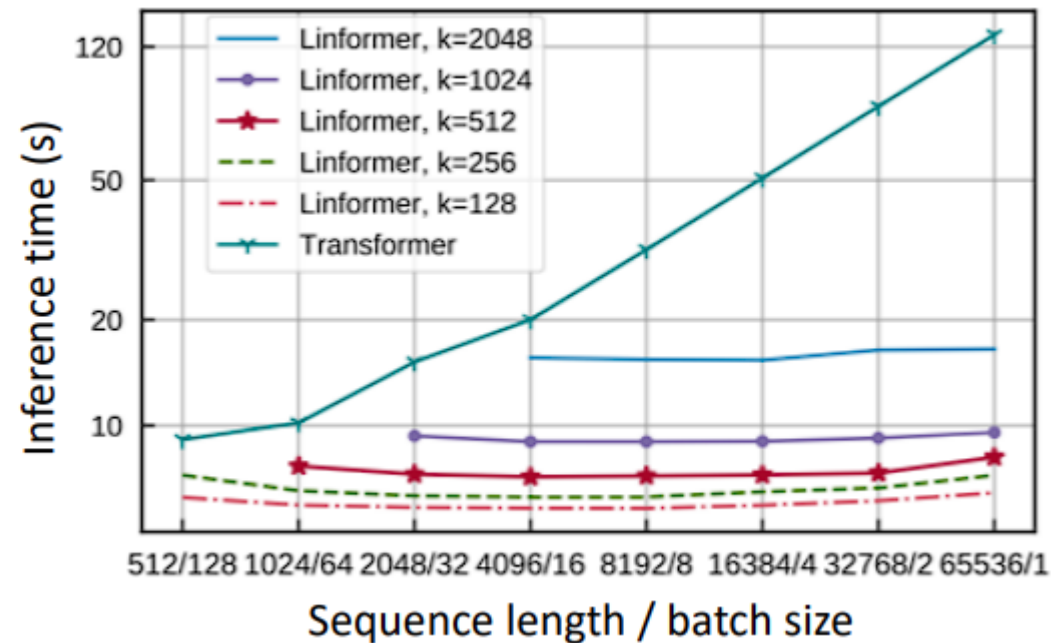
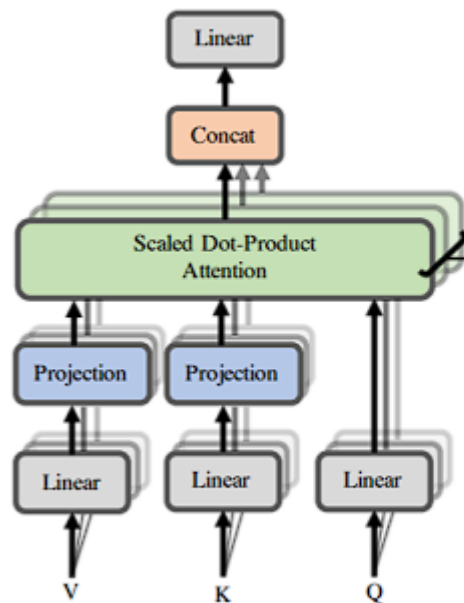

$$\begin{matrix} XQ \\ K^T X^T \end{matrix} = XQK^T X^T \in \mathbb{R}^{T \times T}$$

Need to compute all
pairs of interactions!
 $O(T^2 d)$

출처: CS224n lecture 7

Works on improving on quadratic self-attention cost

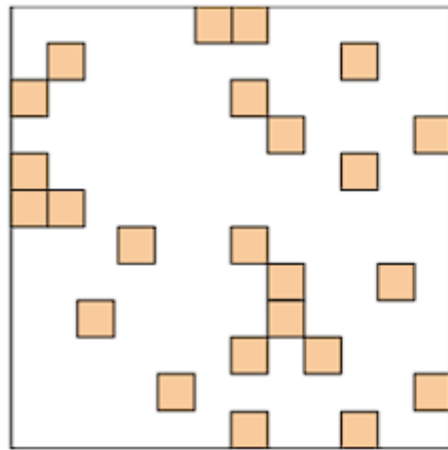
- Linformer
 - Sequence length의 차원을 Linear layer를 통해 작은 차원으로 축소시켜 attention을 계산



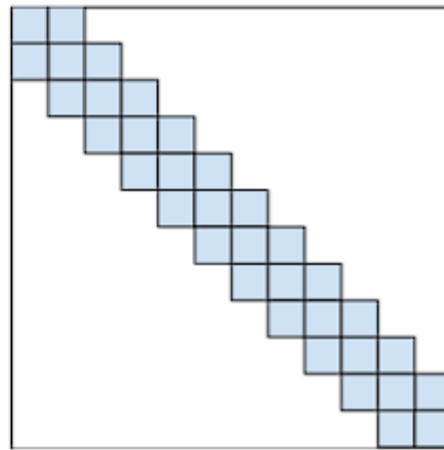
출처: CS224n lecture 7

Works on improving on quadratic self-attention cost

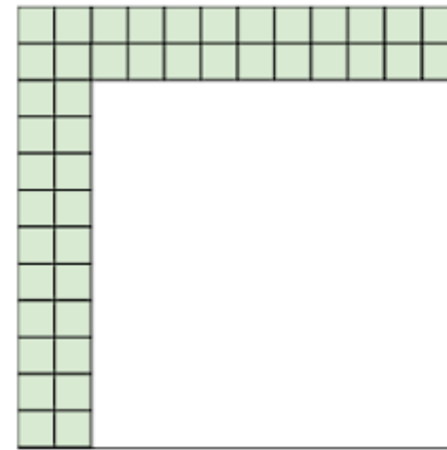
- BigBird
 - 일부 중요한 관계 사이의 attention만을 계산



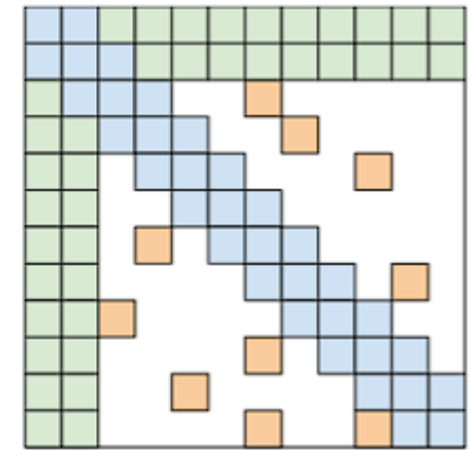
(a) Random attention



(b) Window attention



(c) Global Attention



(d) BIGBIRD

출처: CS224n lecture 7

Discussion questions

- 이미지 분야에서도 Transformer를 활용한 모델이 존재합니다. Transformer의 특징을 고려했을 때 Transformer를 이미지 분야에 어떻게 적용할 수 있을까요?
- 지난 시간에 배운 RNN과 비교했을 때, Transformer가 RNN을 대체할 수 있었던 이유는 무엇일까요?

감사합니다.