Self-Attention For Generative Models

Wongyu Kim (김원규)

Internet Computing Laboratory
Department of Computer Science
Yonsei University

2020.09.24

RNN vs CNN

RNN

- ✓ Sequential -> inhibit parallelization
- ✓ Long and Short Dependencies Problem(attention 없다 가정)
- ✓ We want to model hierarchy

CNN

- ✓ Trivial to parallelize
- ✓ Local Dependencies
- ✓ Computation linear or logarithmic
- ✓ Long-distance Dependencies require many layers

Self-Attention

따라서 우리는 두 개의 장점을 섞은 Self-Attention 필요

Attention is Cheap!

대부분 (dim > length) 인 상황이 많음 => Self-Attention이 좋을 가능성이 높음

FLOPs

Self-Attention	$O(length^2 \cdot dim)$	$= 4 \cdot 10^9$
RNN (LSTM)	$O(length \cdot dim^2)$	$= 16 \cdot 10^9$
Convolution	$O(length \cdot dim^2 \cdot kernel\_width)$	$= 6 \cdot 10^9$

length=1000 dim=1000 kernel_width=3

Machine Translation: WMT-2014 BLEU

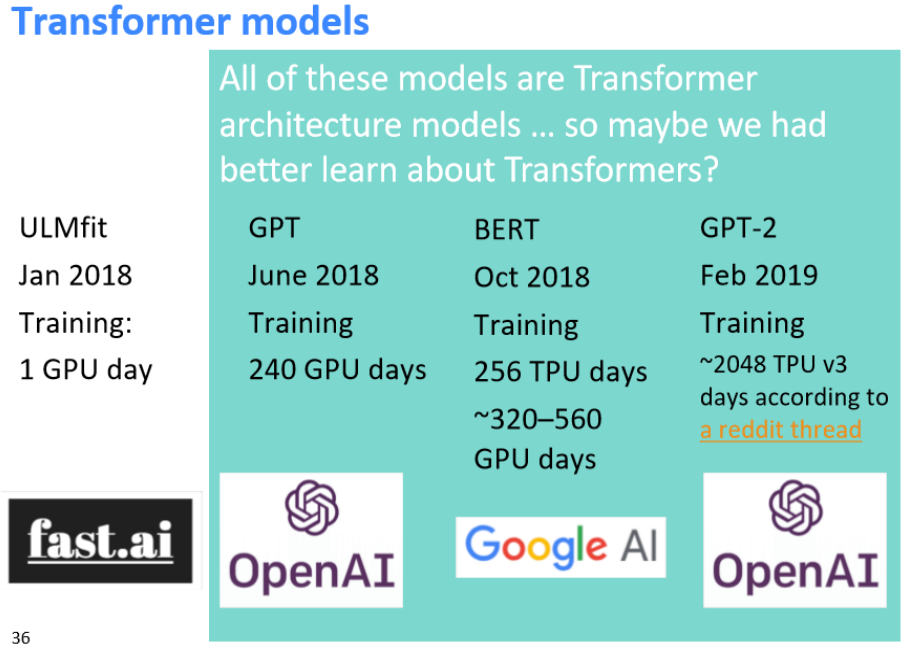
WMT에서도 좋은 성능을 보이고 있음

	EN-DE	EN-FR
GNMT (orig)	24.6	39.9
ConvSeq2Seq	25.2	40.5
Transformer*	**28.4**	**41.8**

*Transformer models trained >3x faster than the others.

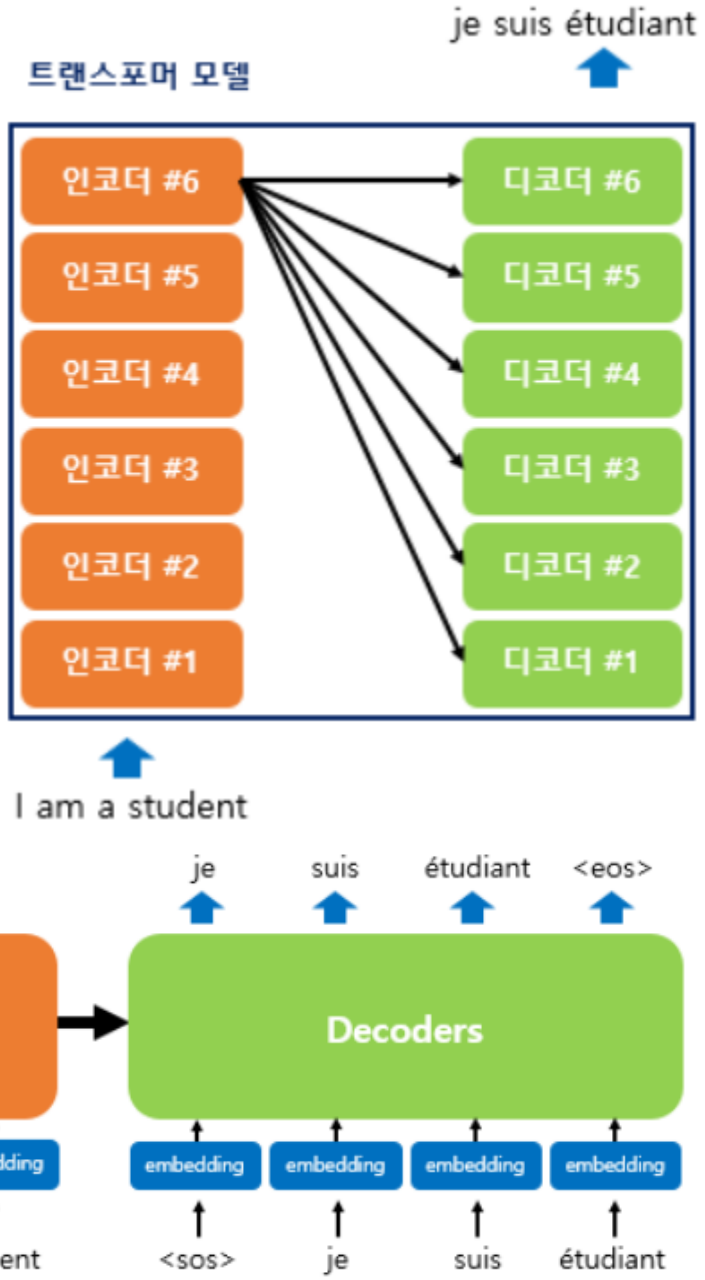
Transformer

- ✓ Parallelization Computation
- ✓ Attention 기능은 유지
  -> long range dependency 해결
- ✓ 인코더-디코더 구조는 유지
- ✓ 처음의 target task는 NMT 였음

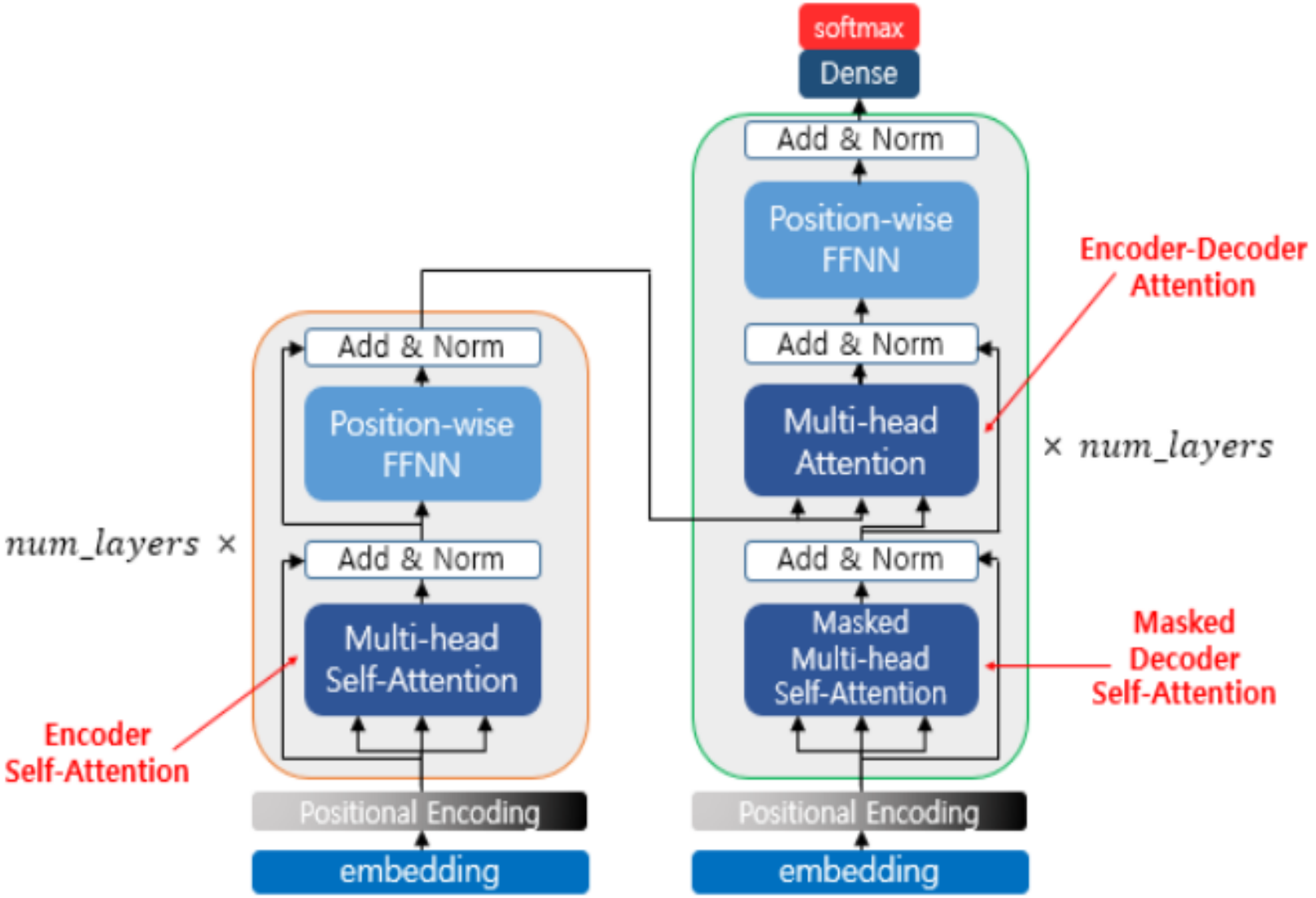


Transformer

- ✓ <sos> & <eos> symbols
- ✓ 인코더의 마지막 output
  -> 각 디코더와 cross attention



Transformer



Transformer - encoder

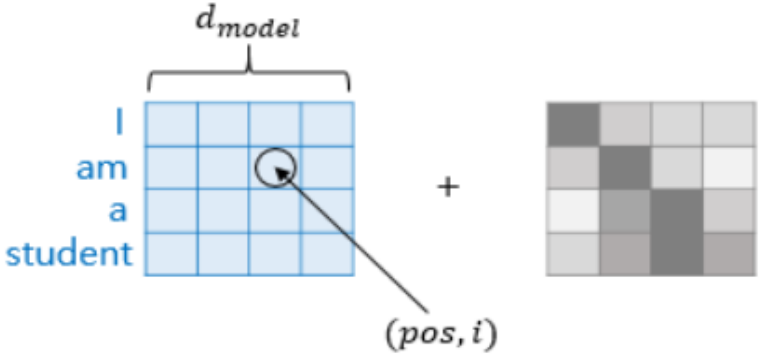
✓ Positional Encoding

기존 RNN - recurrent -> 자연스럽게 위치 정보 획득

Transformer - 인위적으로 위치 정보 주입

$$PE_{(pos,\,2i)} = sin(pos/10000^{2i/d_{model}})$$

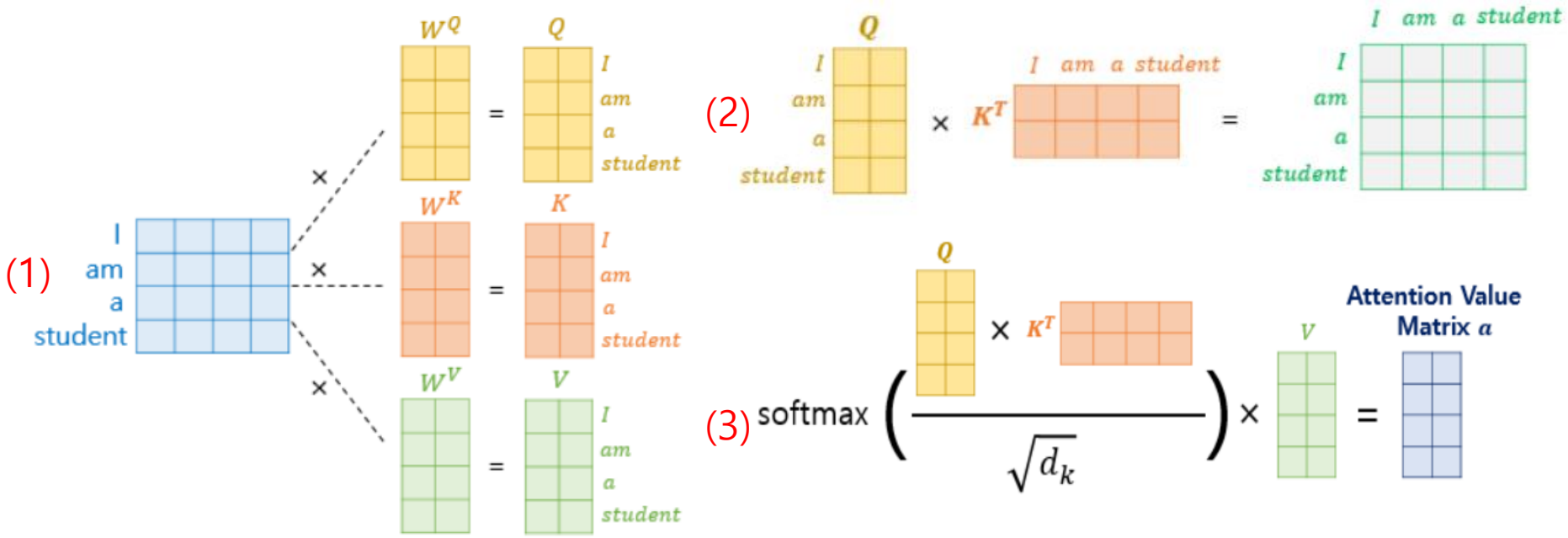
$$PE_{(pos,\,2i+1)} = cos(pos/10000^{2i/d_{model}})$$



i가 홀수, 짝수에 따라 & pos 값에 따라 값이 달라짐

Transformer – Multi-Head Self-Attention

- ✓ Query, Key, Value가 모두 같은 형태
- ✓ 기본 어텐션 메커니즘을 따름
  dot_product(Query, Key) = Attention Distributions
  weighted_sum(Attention Distributions * Values) = Attention Value

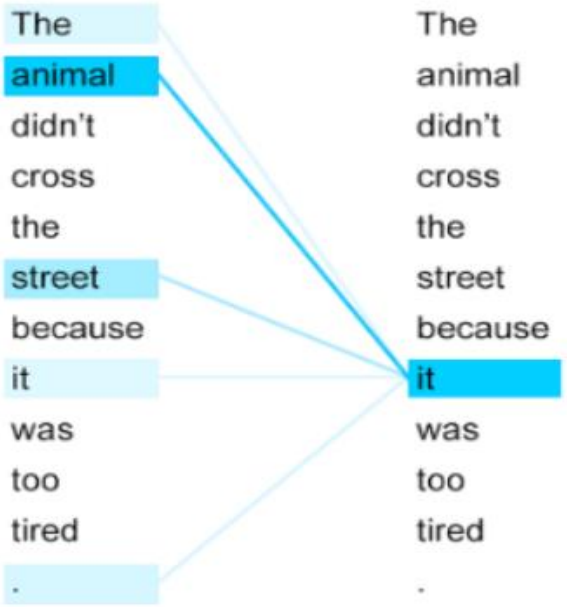


1 cycle이 1 head = 1 attention 이라 함

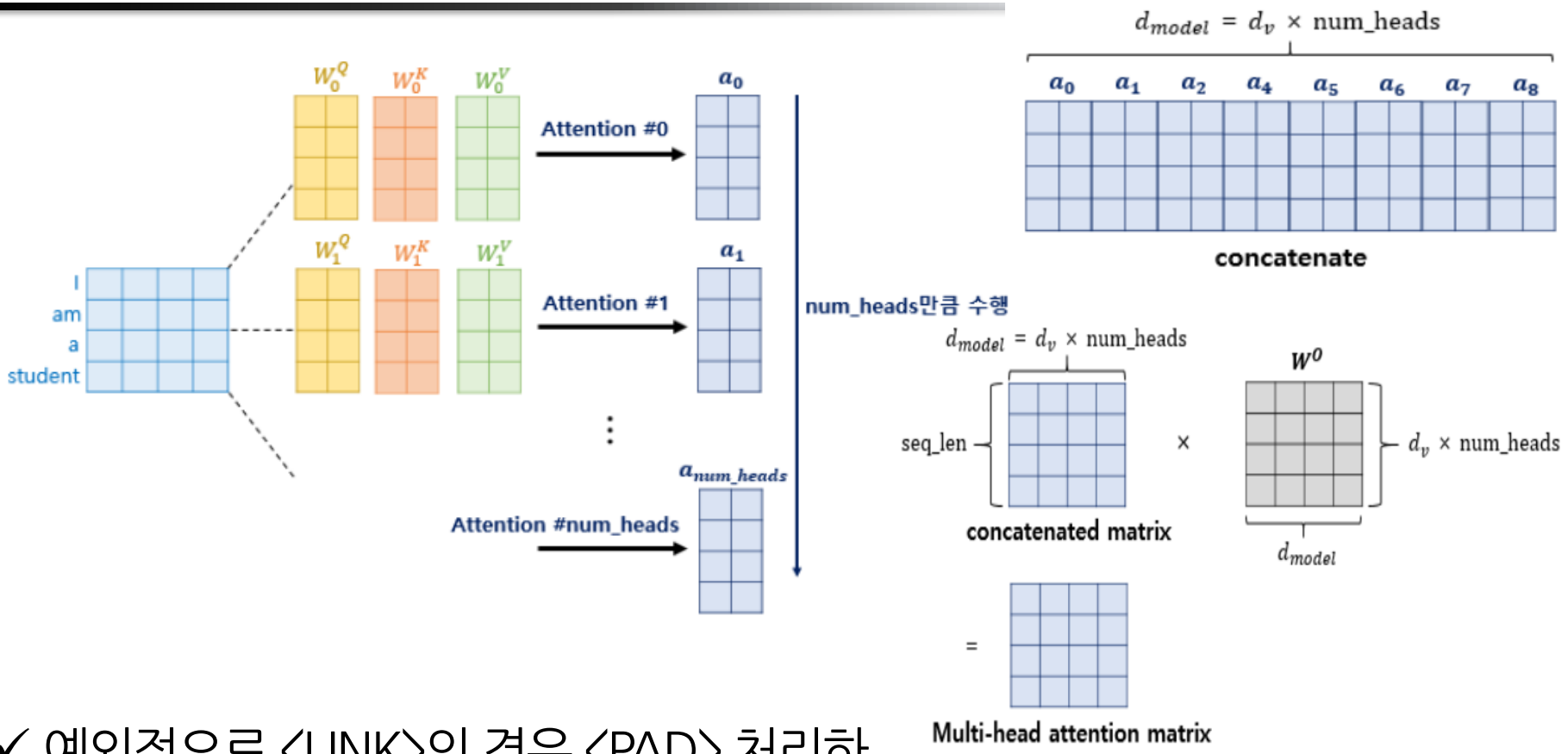
- ✓ d-model = 512, d-k = d-model / num_heads = 512 / 8 = 64

Transformer – Multi-Head Self-Attention

- ✓ it이 animal에 관련 있는지 street에 관련 있는지 스스로 깨닫는 구조
- ✓ 첫 head가 it과 animal 사이의 관계를 주의 깊게 봤다면 두 번째 head는 it과 tired의 관계를 주의 깊게 보는 것 (여러 시점에서 보도록 하기 위해 multi heads 사용)
- ✓ Masking을 하지 않는 이유: 인코더는 입력을 알고 있고 단지 정보를 압축만 하면 되고 따라서 구조상 masking을 전혀 할 필요가 없음



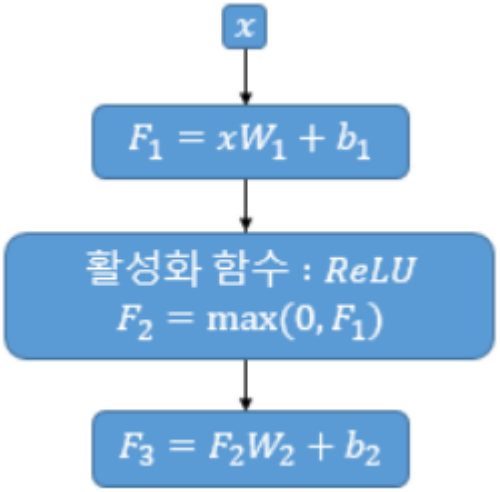
Transformer – Multi-Head Self-Attention

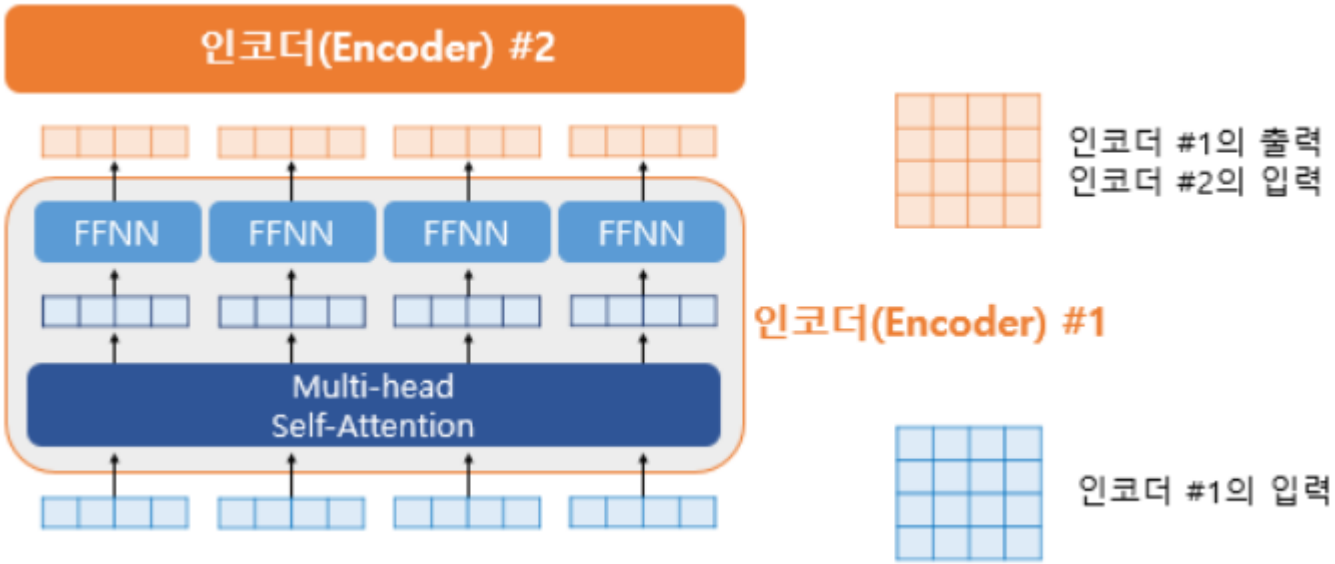


✓ 예외적으로 <UNK>의 경우 <PAD> 처리하여 어텐션 분포 값을 0으로 하여 의미 없게 함

Transformer – Feed Forward Neural Network

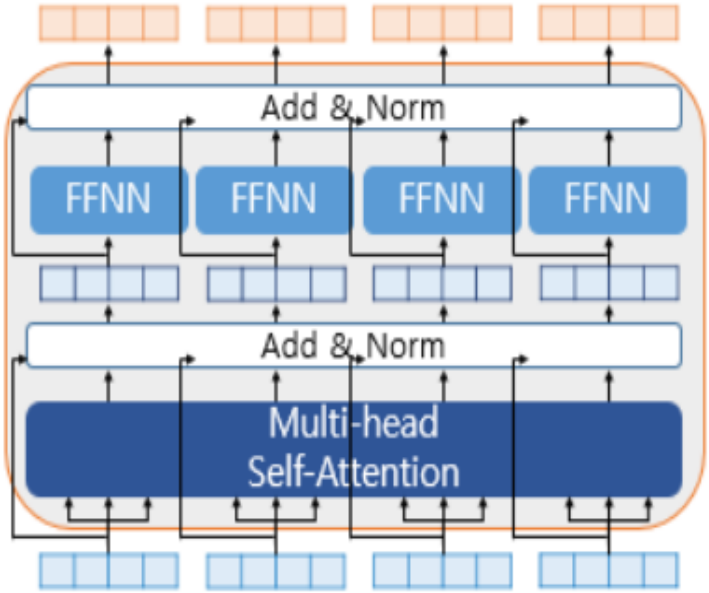
- ✓ FFNN은 인코더 디코더 공통
- ✓ 인코더와 디코더의
  멀티헤드 모듈과 FFNN 모듈의 입력 & 출력 크기는
  모두 (seq_len, d-model)





Transformer – Residual Connection & Layer Normalization

- ✓ 아까 크기가 같기 때문에 skip connection이 가능
- ✓ Layer norm은 레이어 입력을 mean 0 & variance 1의 분포로 바꾸게 하는 것 (batch normalization 처럼)

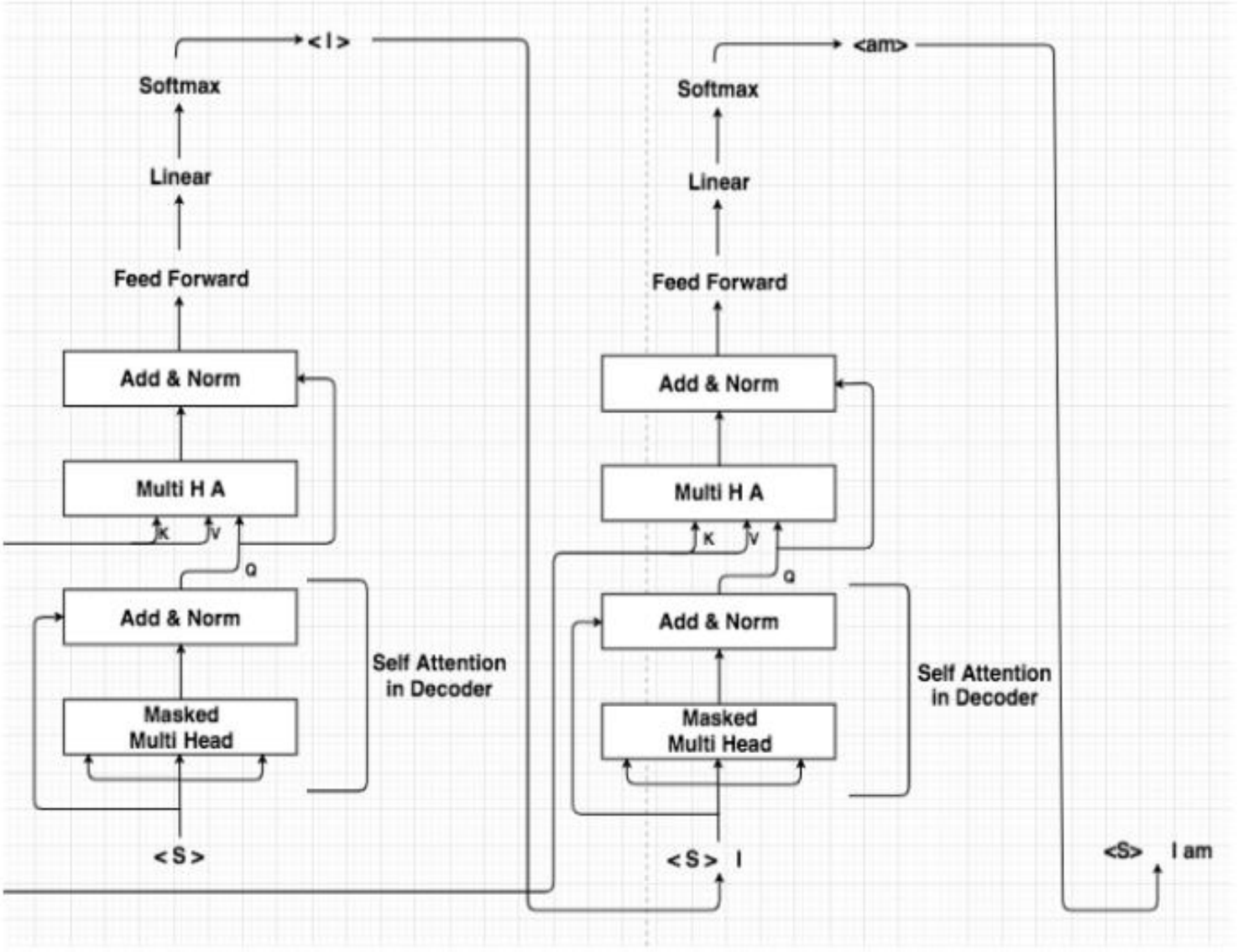


인코더(Encoder) #1

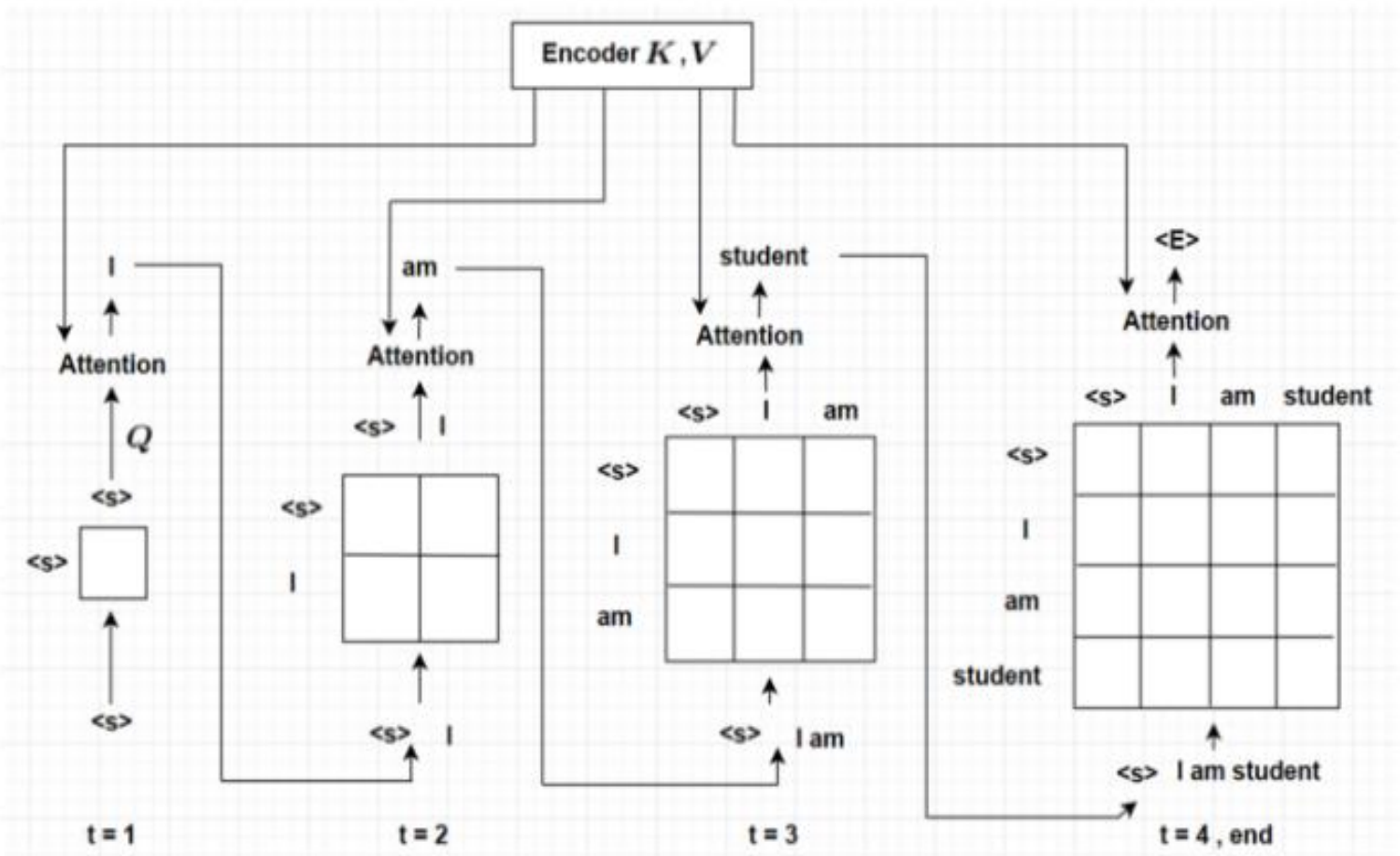
Transformer – Decoder

- ✓ Masking 이유: 이전 정보만 input으로 넣기 위해, 뒷 부부은 -inf로 해서 유사도 측정에 영향을 주지 않음 = 어텐션 작동 안함
- ✓ Inference 시에는 greedy or beam search
- ✓ 중간에 cross-attention은 self attention이 아니라 encode의 최종 결과가 Key & Value로 작동

Transformer – Decoder



Transformer – Decoder



참고 문헌

한글 블로그 자료

- https://wikidocs.net/31379
- https://pozalabs.github.io/transformer/
- https://namhandong.tistory.com/48
- https://medium.com/platfarm/%EC%96%B4%ED%85%90%EC%85%98-%EB%A9%94%EC%BB%A4%EB%8B%88%EC%A6%98%EA%B3%BC-transfomer-self-attention-842498fd3225

CS224n-winter 2019-syllabus-lecture 14

- https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/
- https://www.youtube.com/watch?v=QEw0qEa0E50&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=14

Questions & Answers

Thank you