

Transformer Paper Review

Attention Is All You Need - Ashish Vaswani et al.

2025.10.14 | 삼석사와아이들 | 김재현 / 이문용 / 어지유 / 김현진 / 박재익

목차

- | | | |
|----|---------------------|-----|
| 01 | Transformer 이전 RNN | 00p |
| 02 | Transformer | 00p |
| 03 | Code Implementation | 00p |

CHAPTER 1

Transformer 이전 RNN

Long Short-Term Memory - Hochreiter, Sepp, and Jürgen Schmidhuber.

Sequence to Sequence Learning with Neural Networks - Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le.

TRANSFORMER PAPER REVIEW

SIMPLE RNN

TRANSFORMER 이전 RNN

04

핵심 아이디어

1) Memory

이전 셀의 출력을 또 다른 입력으로 받아 과거의 정보를 기억

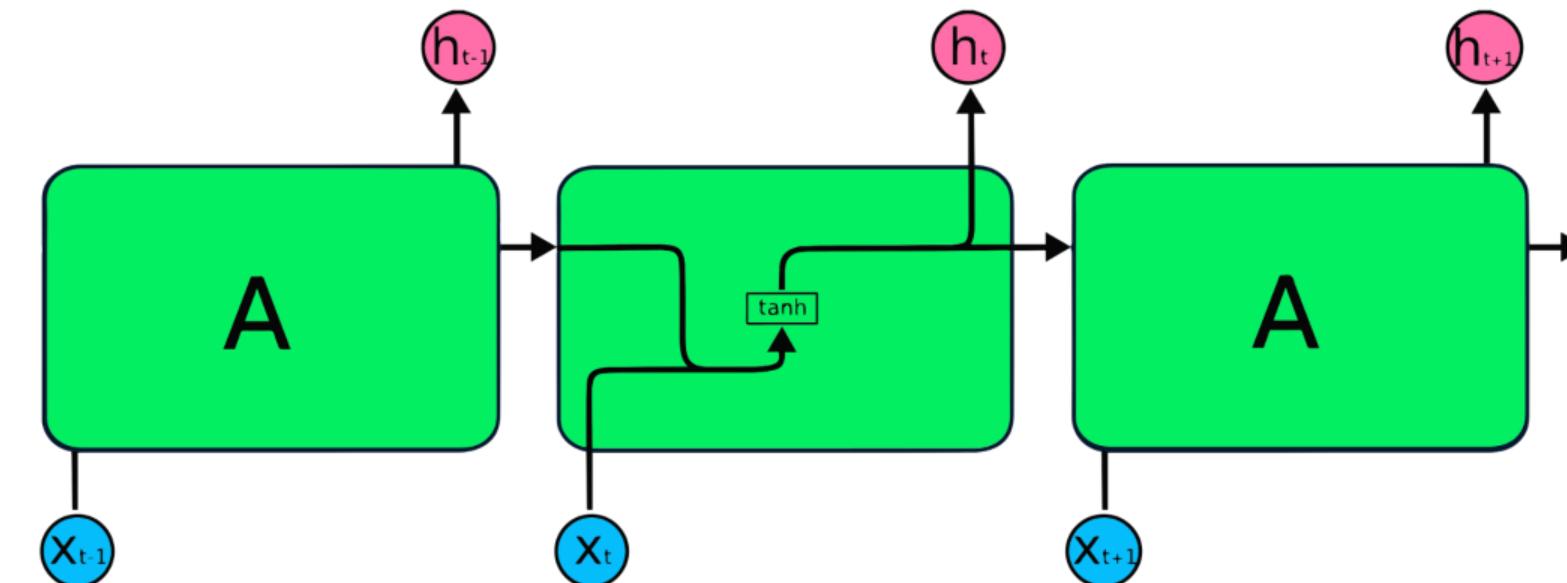
한계

1) 긴 layer에서의 기울기 소실

긴 layer 반복적 곱연산으로 인해 기울기 소실

2) 병렬 처리 불가

순차처리가 필수적이기 때문에 병렬처리 불가



LSTM

핵심 아이디어

1) Simple RNN의 기울기 소실 문제 해결

이전 hidden layer의 출력을 또 다른 입력으로 받아 과거의 정보를 기억

2) 게이트

입력 게이트(i): 새로운 정보와 결합

망각 게이트(f): 과거 정보에 대한 제거

출력 게이트(c): 현재 셀의 정보

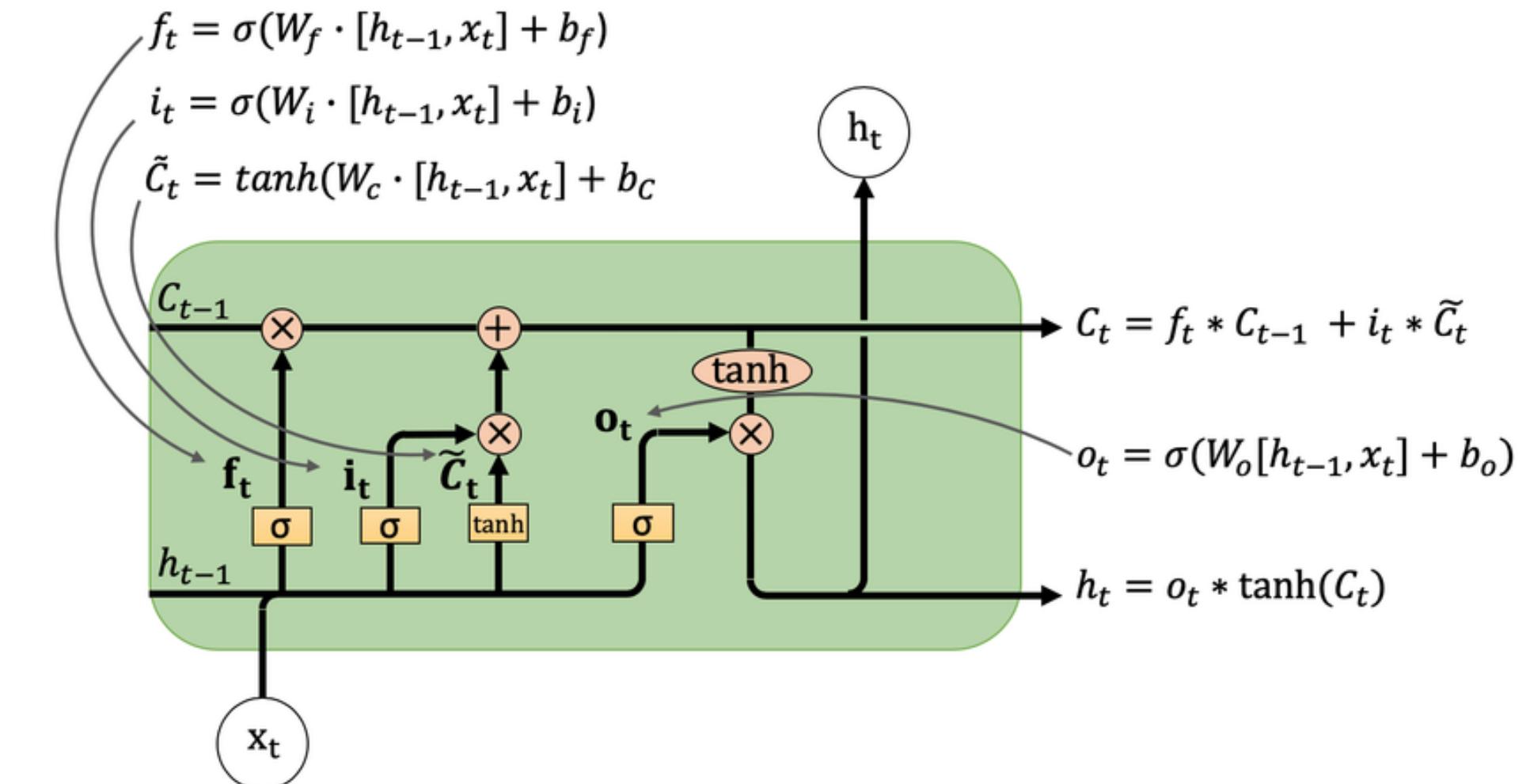
3) 효과

장기 의존성 학습 용이: 기울기가 셀을 따라 안정적

선택적 기억: 불필요 정보는 제거(f)/유용정보 저장(i)

표현력 상승: 다양한 시계열/NLP에서 성능 향상(번역,

음성 등)



LSTM

한계

1) 병렬 처리 불가

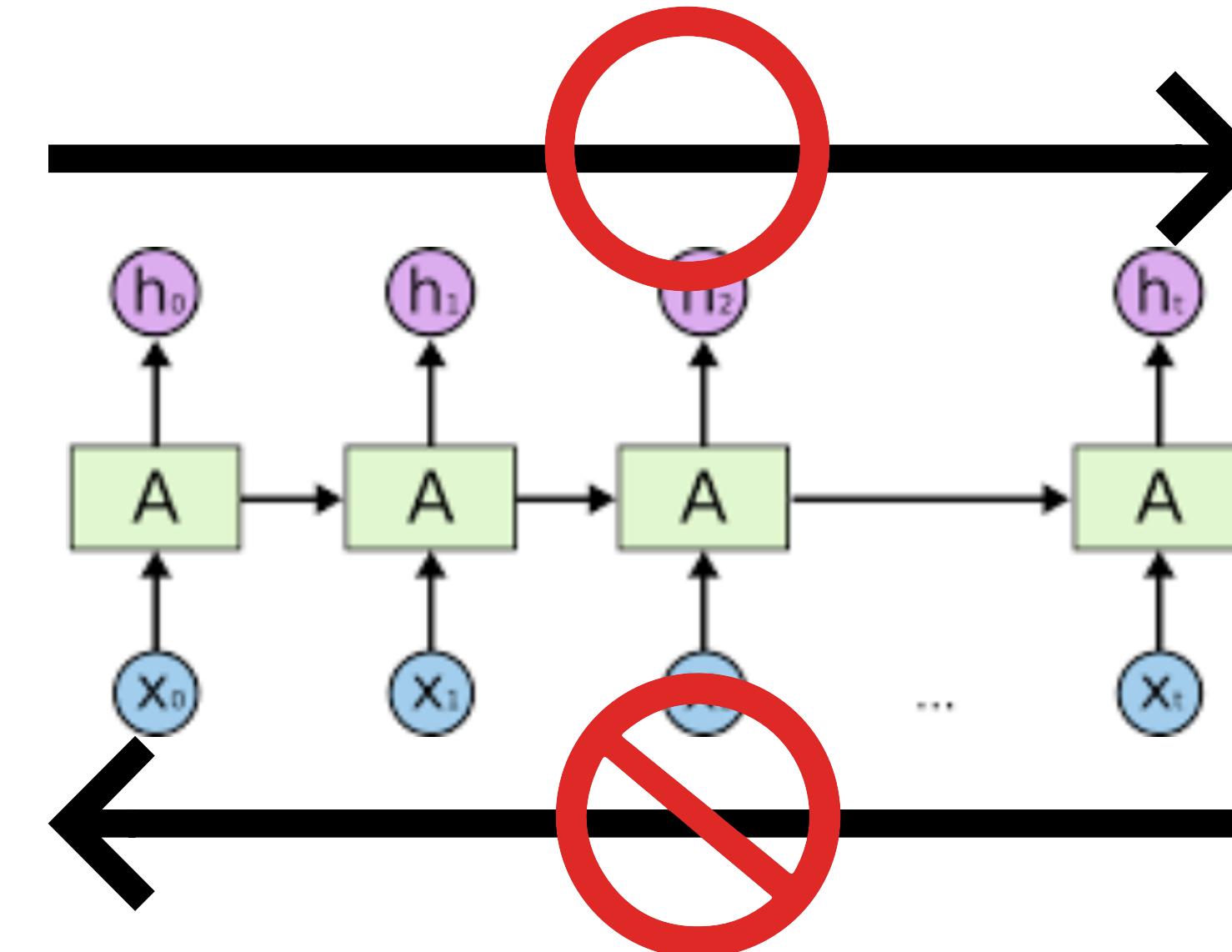
여전히 Simple RNN과 같이 이전 셀에 의존적이기 때문에 순차처리가 필수적이므로 병렬 처리가 불가

2) 파라미터·연산량 부하

각 셀마다 내부적으로 연산 횟수가 많아 처리 속도 저하

3) 해석 가능성 한계

게이트 값으로 대략적 경향은 보이지만, 토큰 간 명시적 연결(누가 누구에 주목했는지)을 보여주기 어려움



SEQ2SEQ

TRANSFORMER 이전 RNN

07

핵심 아이디어

1) Encoder - Decoder 구조

입력을 RNN(Encoder)로 벡터 z 에 압축 - RNN
(Decoder)이 z 를 시작 신호로 받아 한 토큰씩 생성

2) 입력 시퀀스 역순(reverse) 학습

초기 성능 강화 - 장거리 의존성 완화

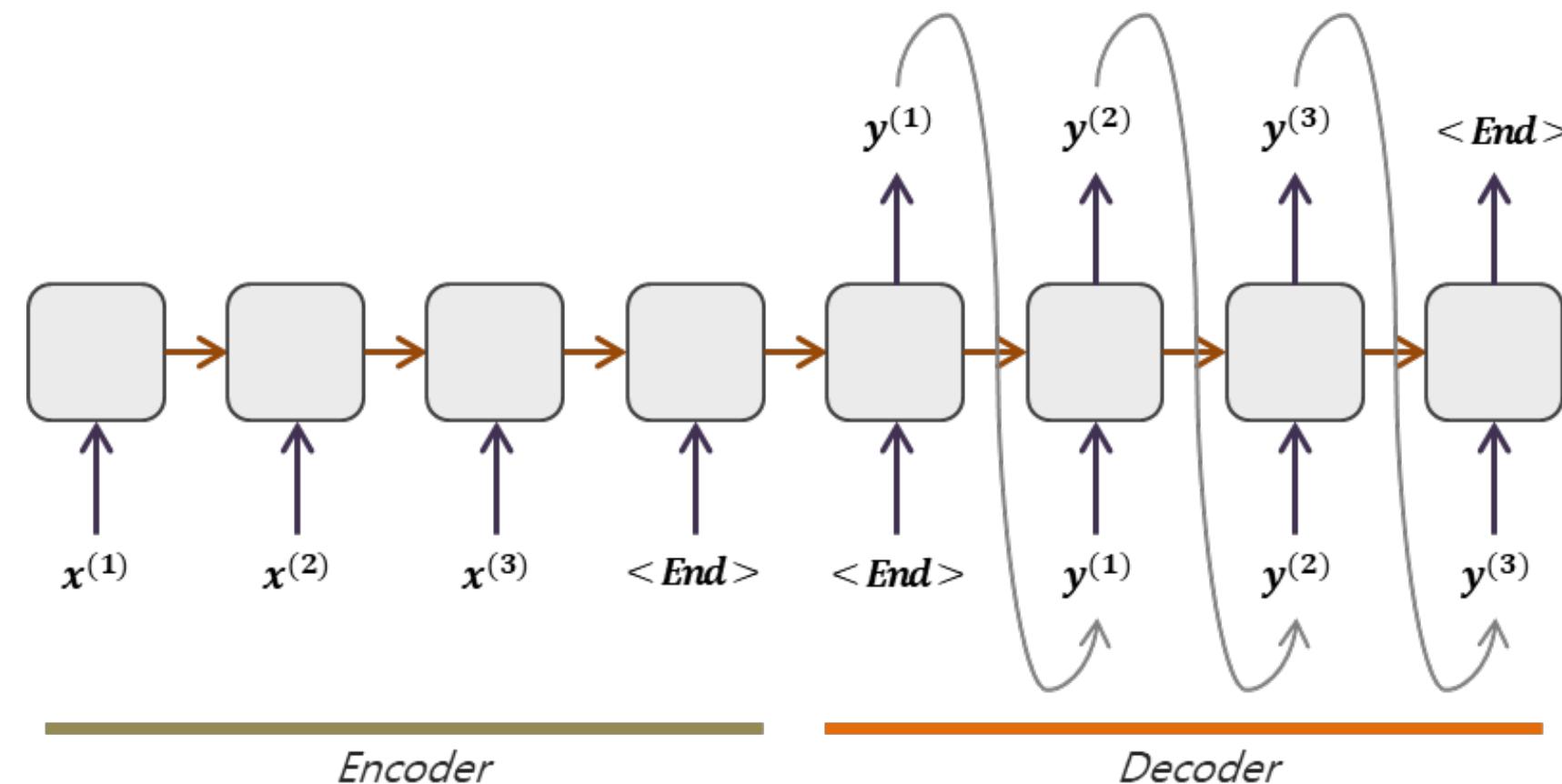
3) 입력 길이 불일치 문제 해결

인코더-디코더 분리로 입력/출력 길이 불일치 문제 해결

4) 효과

길이 유연성: 다양한 길이의 시퀀스 매핑을 단순·일관된
프레임으로 처리

모듈성 강화: 양방향 인코더, 다양한 디코더 전략 등 확장
용이



SEQ2SEQ

TRANSFORMER 이전 RNN

08

한계
(Transformer로 넘어가는 다리)

1) 고정 벡터 병목

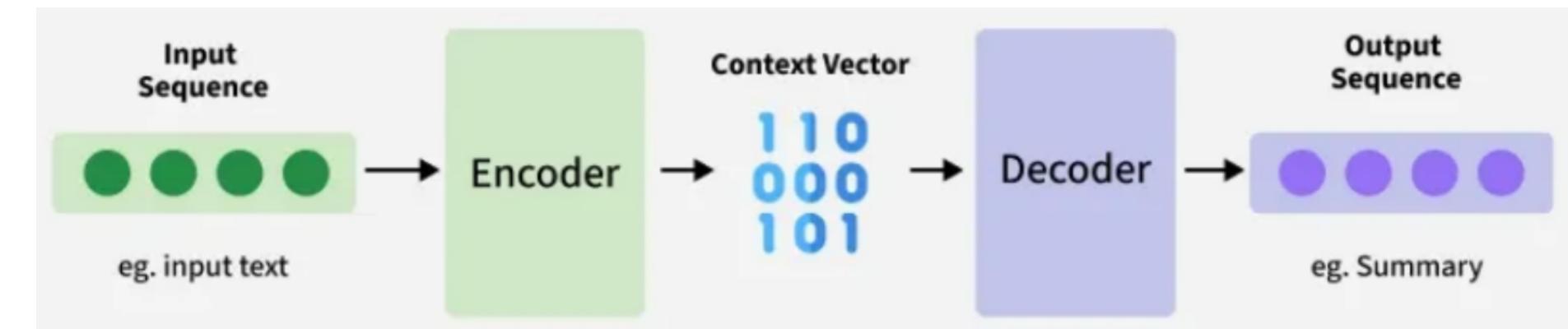
Encoder를 통과한 시퀀스가 고정된 차원의 Context Vector로 변환되면서 긴 시퀀스는 병목을 겪고 정보를 손실

2) 장거리 의존성 취약

RNN의 근본적 문제점 - 앞에 있는 단어일수록 정보가 손실됨

3) 병렬 처리 불가

Encoder - Decoder 구조는 여전히 RNN으로 구성되었기 때문에 병렬처리가 불가



CHAPTER 2

Transformer

Attention Is All You Need - Ashish Vaswani et al.

TRANSFORMER PAPER REVIEW

TRANSFORMER

TRANSFORMER

10

핵심 아이디어

1) Self-Attention

토큰 간 직접 참조: 각 토큰의 Q/K/V 를 만들어 유사도로 가중합

2) Multi-Head Attention

여러 Head가 서로 다른 표현 하위 공간/관계(문법, 의미, 위치 등)를 동시에 포착

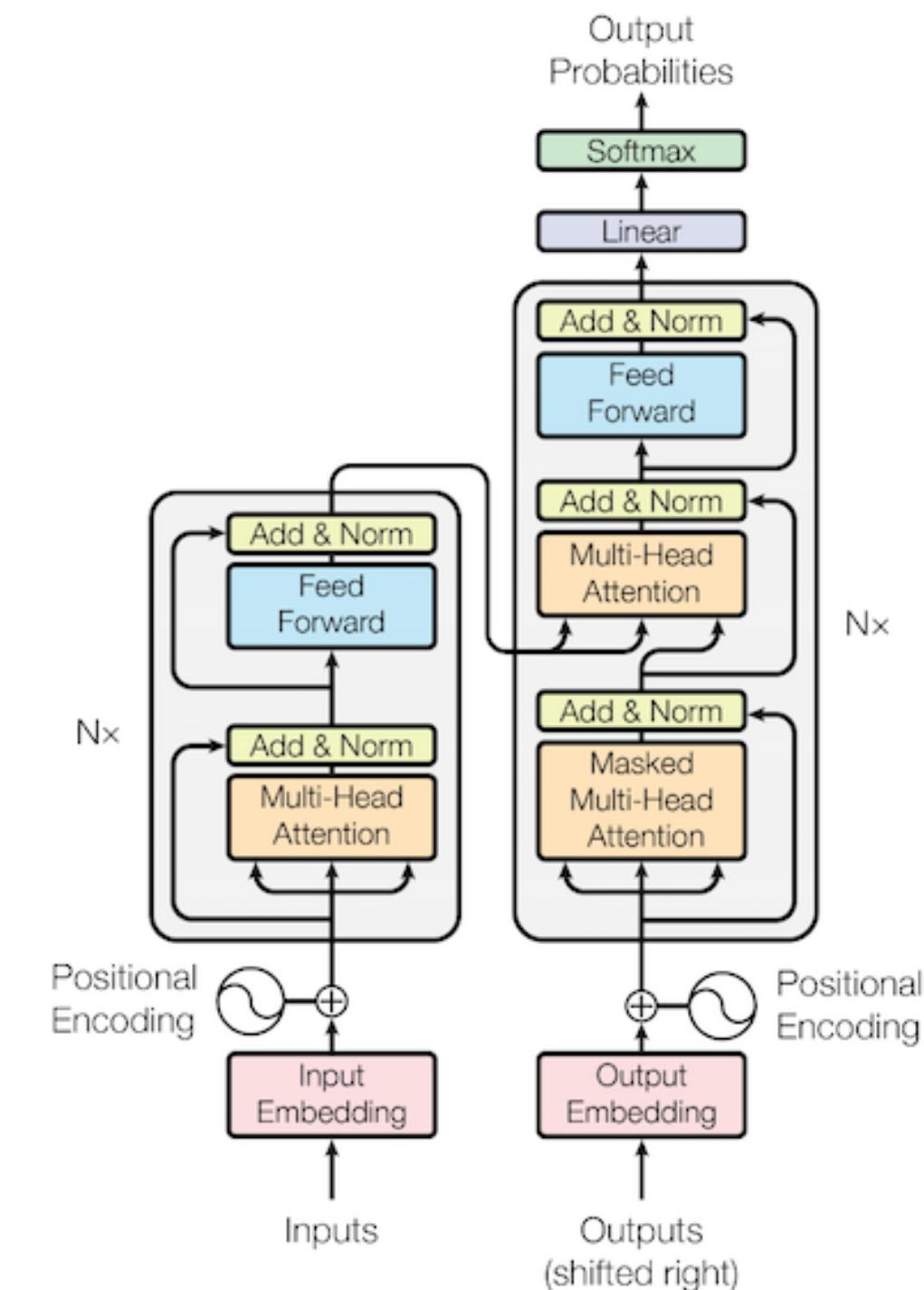
3) 전 구간 병렬 처리

이전까지 고질적인 문제였던 병렬 처리를 RNN을 제거하고 Attention 메커니즘을 채택하여 해결

4) Codec 구조

Encoder: Self-Attn + FFN

Decoder: Masked-Self-Attn + Cross-Attn + FFN



TRANSFORMER PAPER REVIEW

TRANSFORMER

Token Embedding

1) 개념

토큰(단어·서브워드·기호 등)을 고정 크기 벡터로 바꿔 주는 단어표(lookup) 층

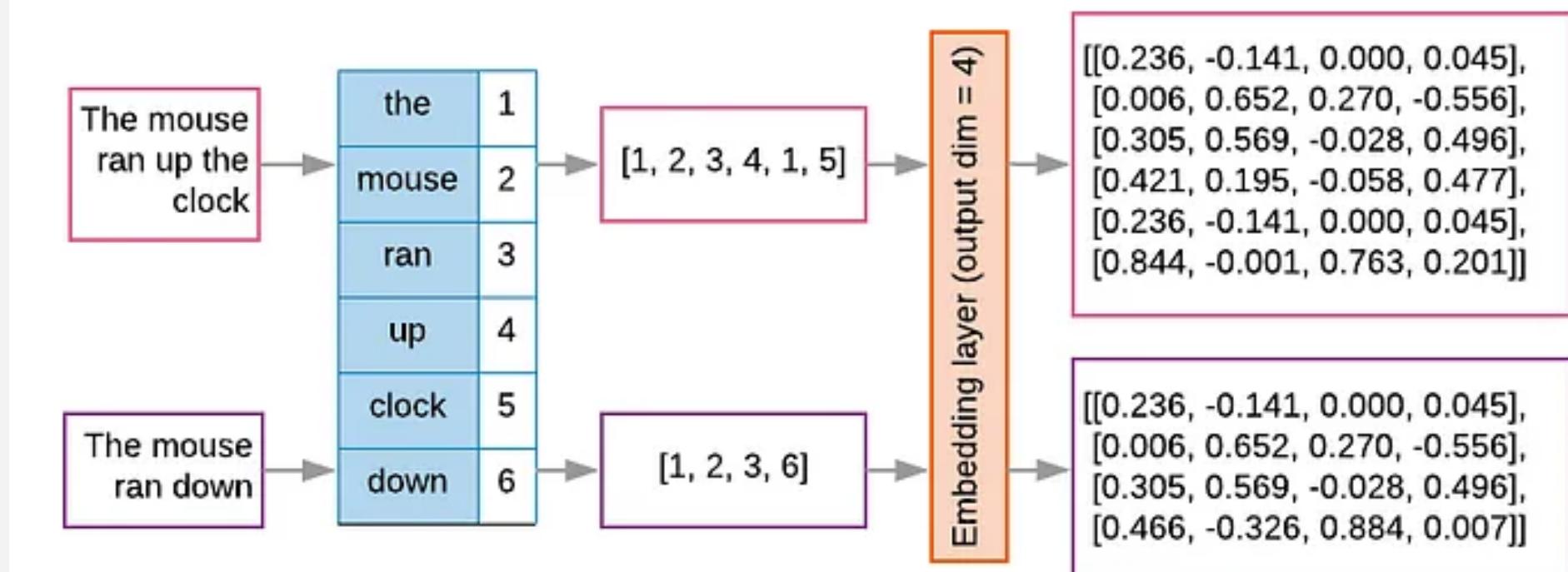
2) 방법

입력으로 들어온 프롬프트 문장의 단어를 사전 생성되어 있는 임베딩 테이블에서 찾아 벡터화시킴

원 논문:

WMT'14 En-De ≈ 37,000 토큰

WMT'14 En-Fr ≈ 32,000 토큰



TRANSFORMER

Positional Encoding

1) 개념

Self-Attention에는 순서 개념이 없으니, 토큰의 위치 정보를 별도로 주입

2) 방법

짝수 채널에는 $\sin(pos, 2i)$, 홀수 채널에는 $\cos(pos, 2i+1)$ 로 위치 정보를 원값(토큰 임베딩된 벡터)에 더함

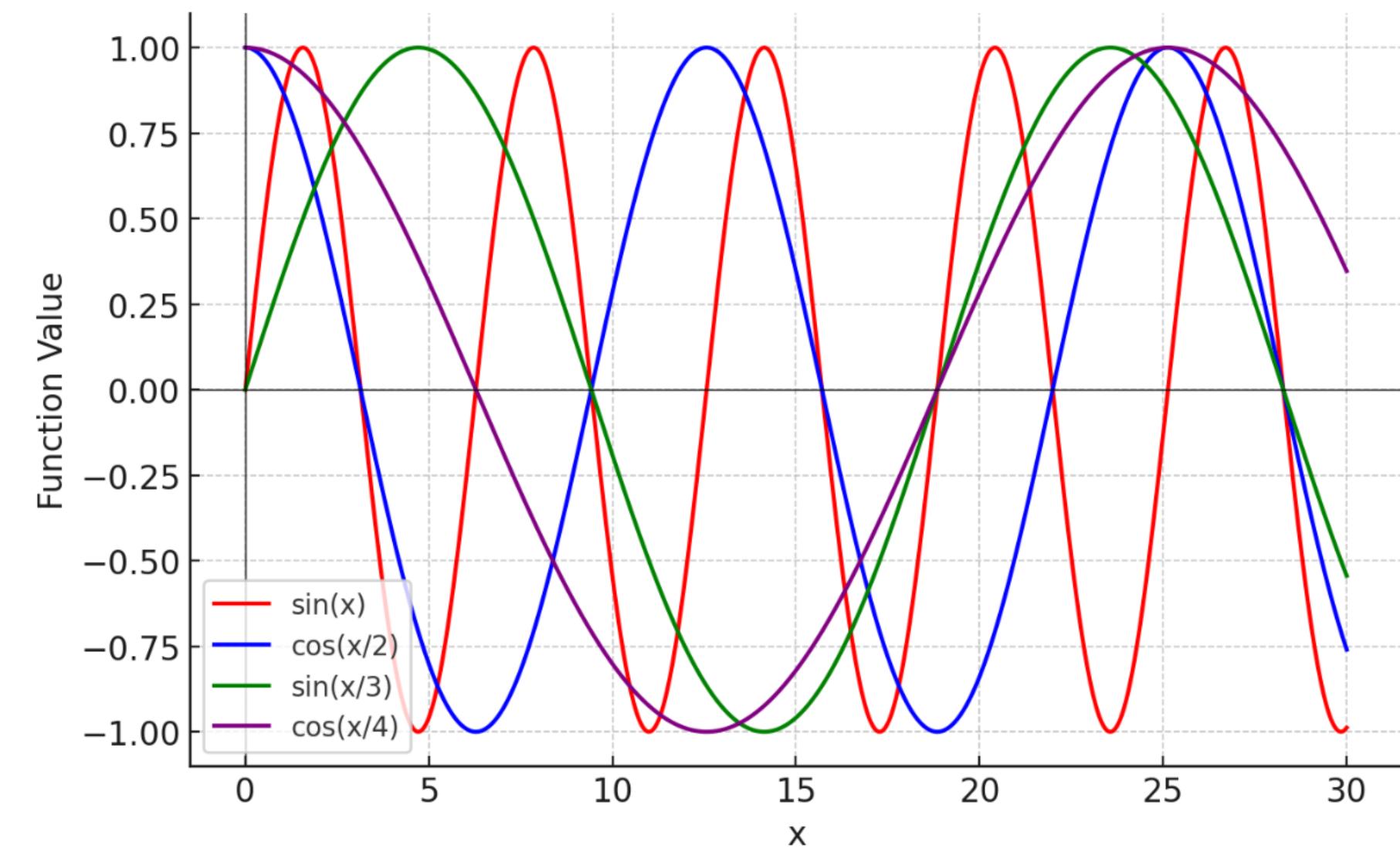
ex) $x = (\text{batch}, \text{seq_len}, d_{\text{model}})$

-> d_{model} 이 채널

3) 효과

상대 위치 학습 용이: 항등식으로 상대적 이동이 선형결합으로 표현 → 어텐션이 거리/방향을 쉽게 포착

표현력·분리성 강화: 정보가 섞이지 않고 명확히 분리된 표현



TRANSFORMER

Self-Attention

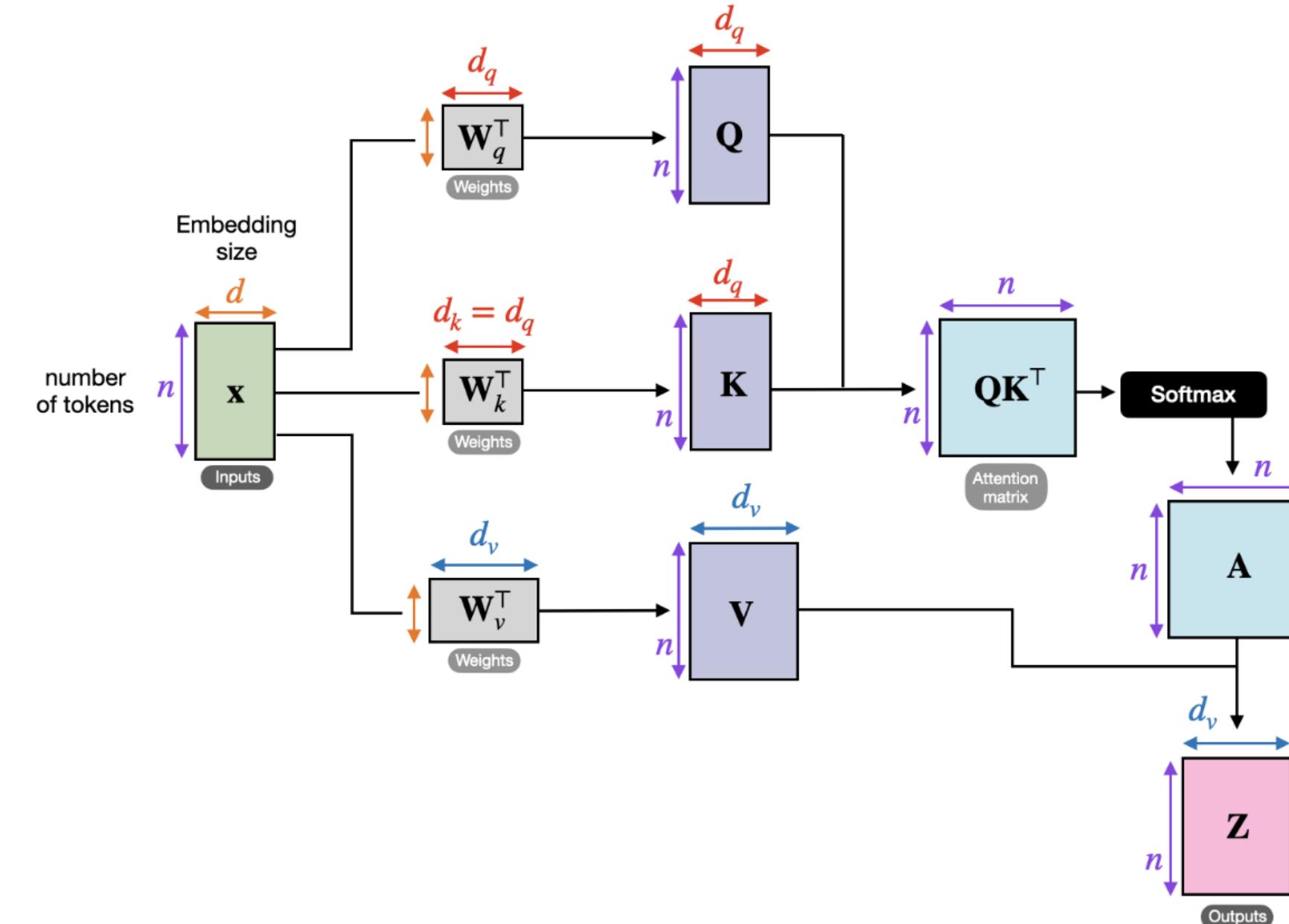
1) 개념

문장(시퀀스) 안의 각 토큰이 다른 모든 토큰을 “참조(attend)”하여 유사도에 따라 정보를 가중 평균해 받는 연산

2) 효과

토큰 간 직접 연결: 멀리 떨어진 단어라도 한 번의 매트릭스 곱으로 직접 연결

속도 상승: RNN처럼 순차로 전파하지 않고 동시에 여러 토큰의 상관관계를 파악할 수 있고, 전 구간 병렬 처리가 가능하여 속도가 상승



TRANSFORMER

Multi-Head Attention

1) 개념

서로 다른 파라미터를 가진 h 개의 헤드가 각각 Q/K/V를 만들고 Scaled Dot-Product Attention을 독립적으로 계산

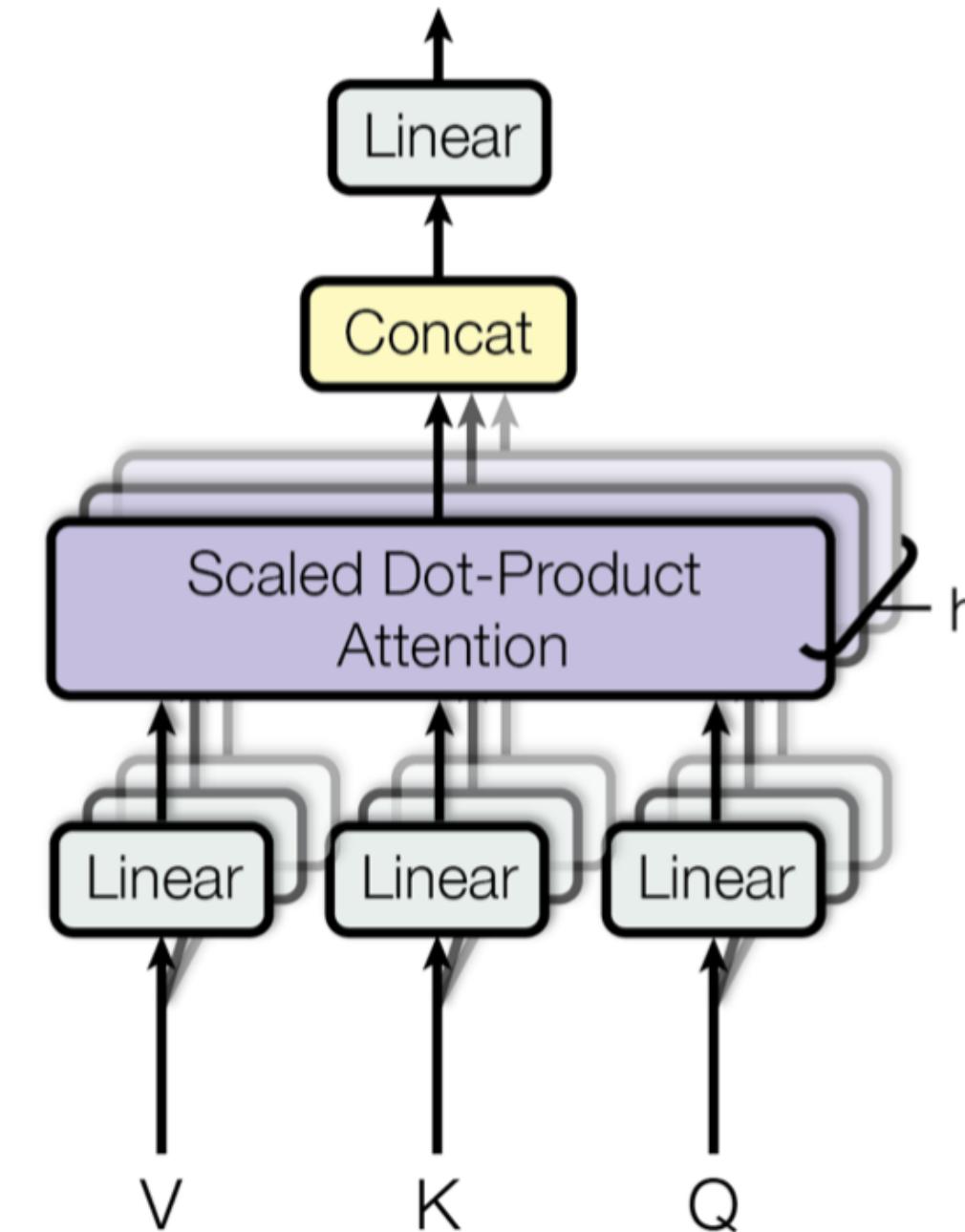
2) 효과

표현 하위공간 분할: 각 헤드가 다른 서브스페이스/패턴 (문법, 의미, 위치, 상관구조)을 학습 → 다양한 관계를 동시에 집약

장거리·다관계 동시 캡처: 떤 헤드는 장거리 의존, 다른 헤드는 지역 문맥/구문 등 서로 다른 스케일을 포착

학습 안정화: 한 큰 헤드보다 여러 작은 헤드가 softmax 포화를 덜 일으키고, 그라디언트 흐름이 비교적 안정

용량 확장성: head 수 h 를 늘리면 동일 FLOPs 내에서 관점 수를 늘리는 식의 용량 조절이 가능



TRANSFORMER

Scaled Dot-Product Attention

1) 개념

쿼리-키 점곱 점수를 $\frac{1}{\sqrt{d_k}}$ 로 스케일한 뒤 소프트맥스해 가중치로 쓰고, 그 가중치로 값 V 를 가중 평균하는 연산

2) 효과

기울기 소실 방지: 차원 d 가 커질수록 $q \cdot k$ 의 분산이 커져 점수가 쉽게 과도한 크기가 됨

학습 안정성: 점수 분포의 스케일을 안정화

temperature 조절: 스케일이 작을수록 소프트맥스가 덜 날카로움 \rightarrow 과신 예방(softmax 포화 예방)

Attention(Q, K, V)

$$= \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} + M \right) V$$

Scale

$\frac{e^{(Z_i/T)}}{\sum_j^n e^{(Z_j/T)}}$ = softmax T(temperature)처럼 동작

T	p(0)	p(4)	p(8)	p(16)	p(20)
1	0.000000	0.000000	0.000006	0.017986	0.982008
차갑게 0.5	0.000000	0.000000	0.000000	0.000335	0.999665
2	0.000040	0.000295	0.002178	0.118903	0.878584
5	0.011454	0.025490	0.056730	0.280984	0.625342
뜨겁게 10	0.058619	0.087449	0.130458	0.290339	0.433136

TRANSFORMER

Skip Connection

1) 개념

입력 x 를 변환 $F(x)$ 의 출력에 그대로 더해(또는 연결해) 주는 경로

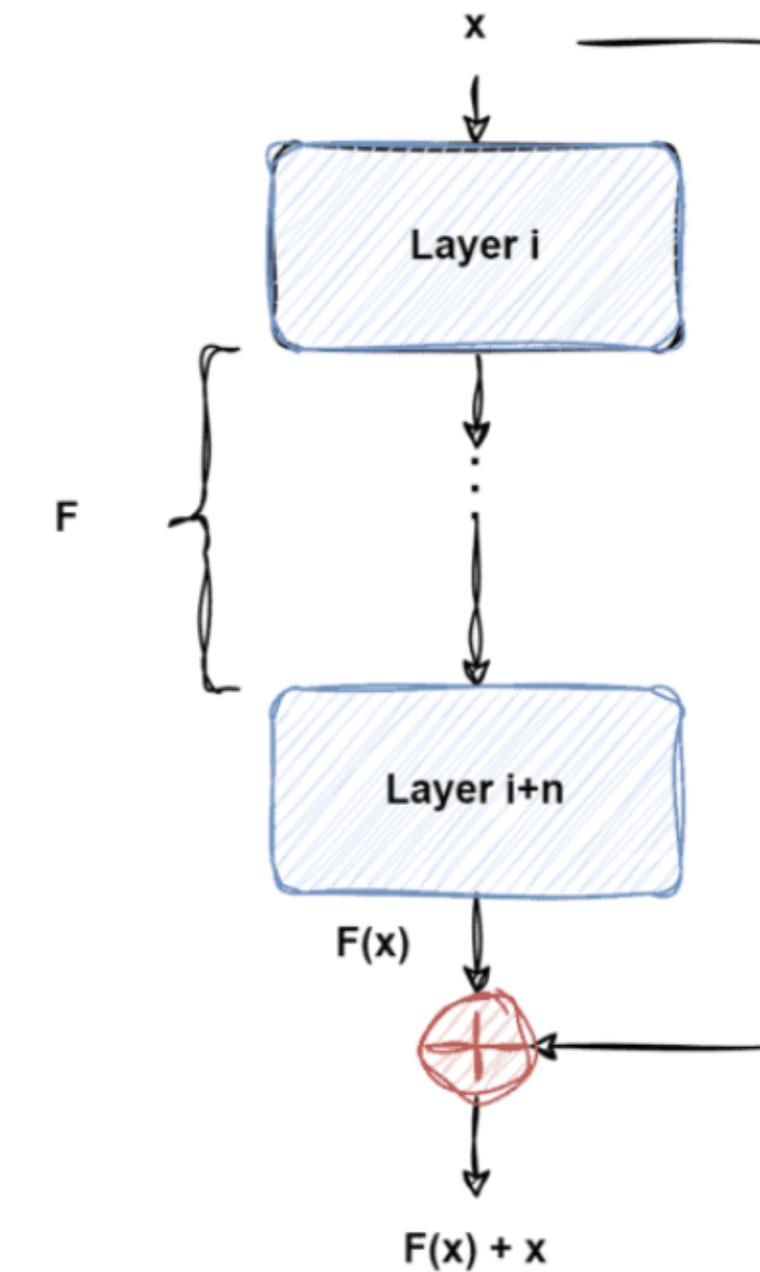
2) 효과

기울기 소실 개선: 항등 경로가 있어 역전파가 막히지 않음(소실/폭주 완화)

최적화: 아무것도 못 배우면($F(x)=0$) 최소한 나빠지지 않는 항등해가 존재 \rightarrow 깊게 쌓아도 성능 붕괴 방지

표현력 증가: 기존 표현 x 를 보존한 채 필요한 변화(잔 차)만 학습

일반화/규제 효과: 과도한 왜곡을 억제하고, 학습이 더 보수적으로 진행되는 경향



TRANSFORMER

Add + Layer Normalization

1) 개념

Add(Residual Connection)로 입력을 더해 안정적 경로를 만들고, Norm(LayerNorm)으로 스케일을 정리해 학습을 안정화하는 장치

2) 효과

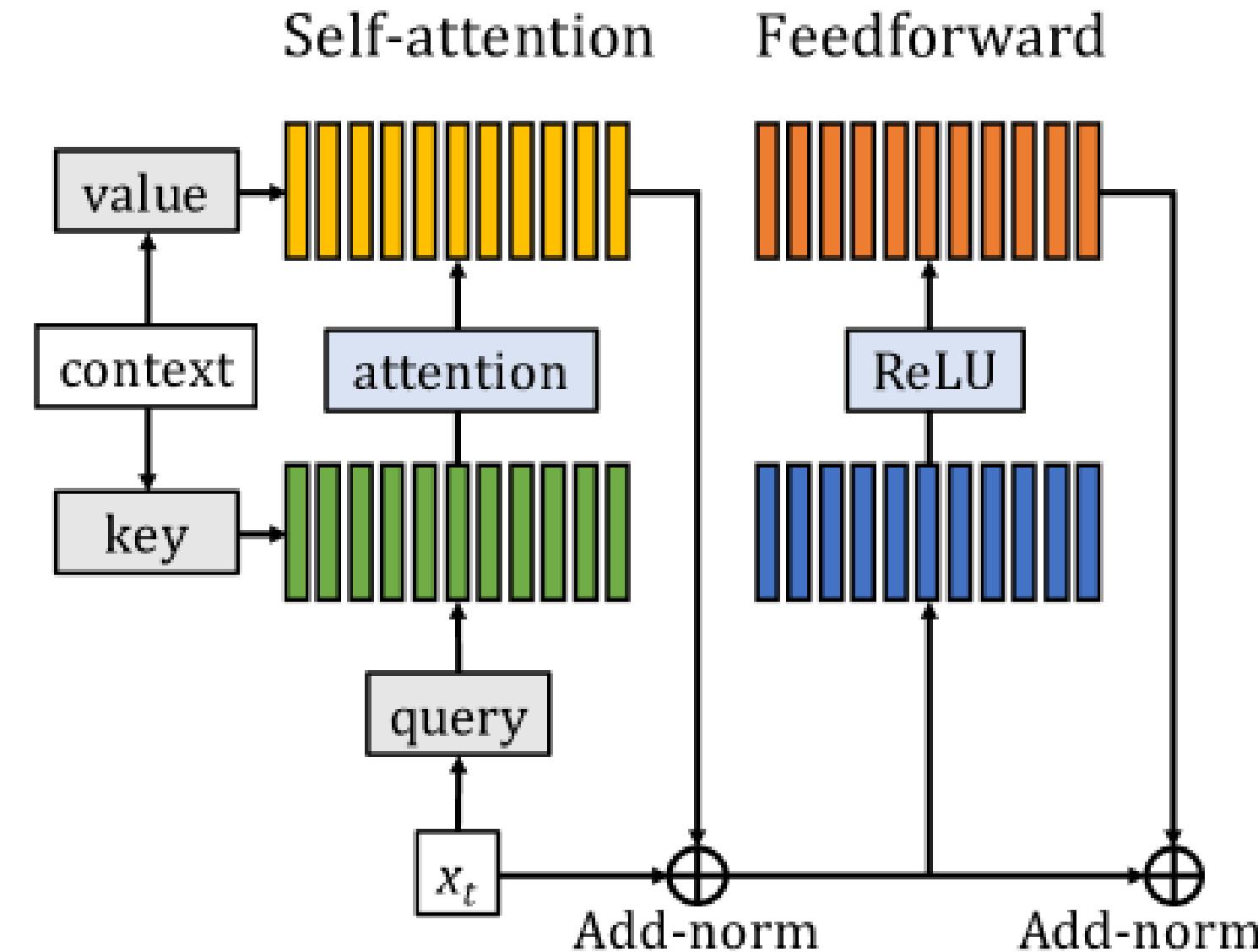
Add: 항등 경로 확보로 기울기 소실/폭주 완화, 수렴 가속

Norm: 각 위치(토큰)별로 평균·분산을 정규화, 스케일/분산 드리프트 억제, 학습률·초기값 민감도 감소

3) LayerNormalization

Post-LN(초기 모델): 수렴 속도가 빠를 수 있으나 깊어질 수록 불안정해질 수 있음

Pre-LN(최근 모델): 아주 깊은 네트워크에서 안정적, 최근 모델들의 표준



TRANSFORMER

Feed Forward Network

1) 개념

어텐션이 “누구를 볼지”를 정하는 관계 연산이라면, FFN은 각 토큰을 깊게 가공하는 특징 변환기

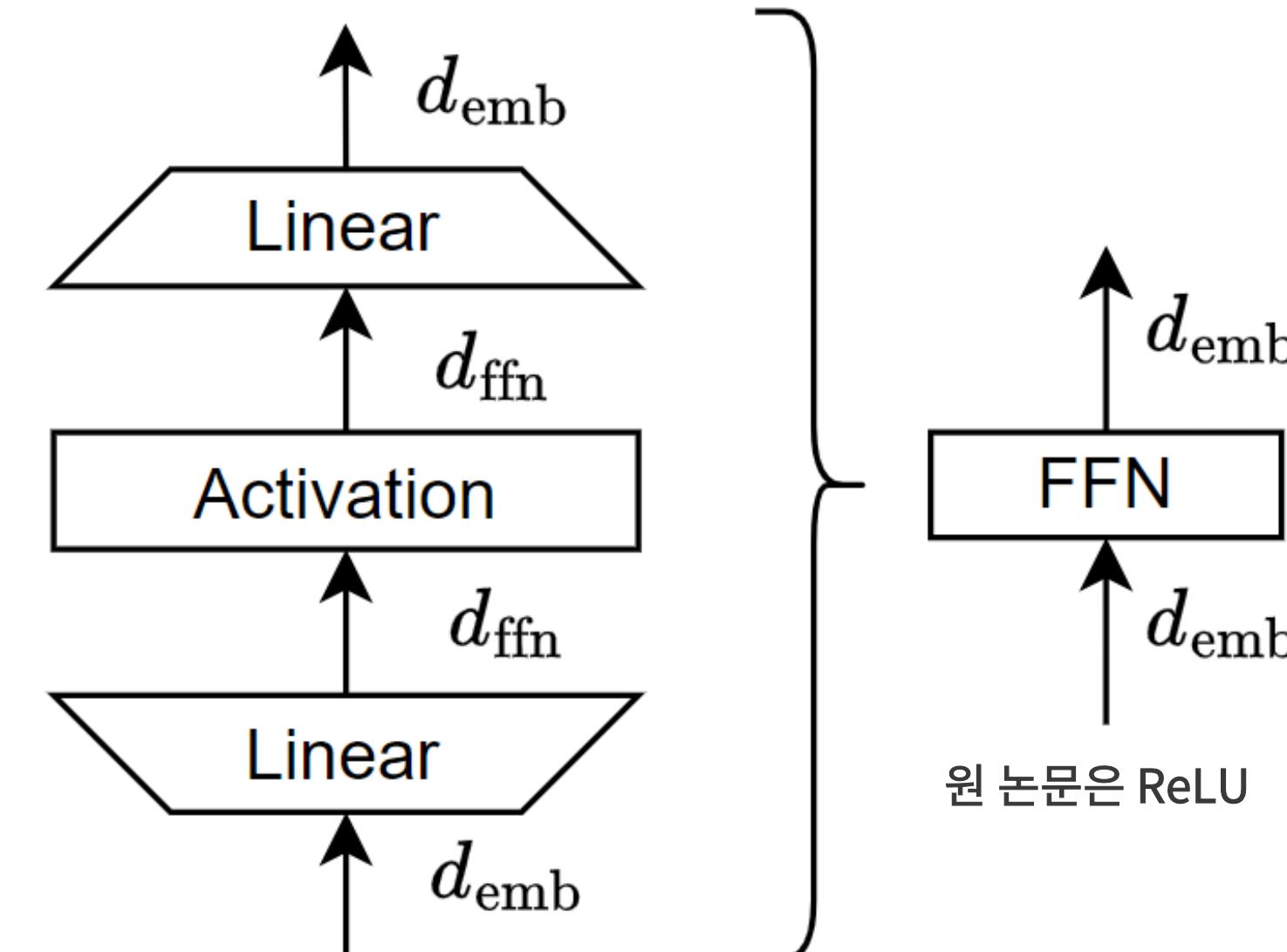
Linear + ReLU/GeLU/Swish 등 비선형 활성화 함수

2) 효과

표현력 증가: 채널 확장으로 표현력을 더 세밀하게 담을 수 있음

비선형성 주입: 선형 합성의 한계를 넘어 비선형 결정 경계를 형성

잔차·정규화와의 시너지: 그라디언트 흐름 안정 + 학습 수렴 속도 개선



TRANSFORMER

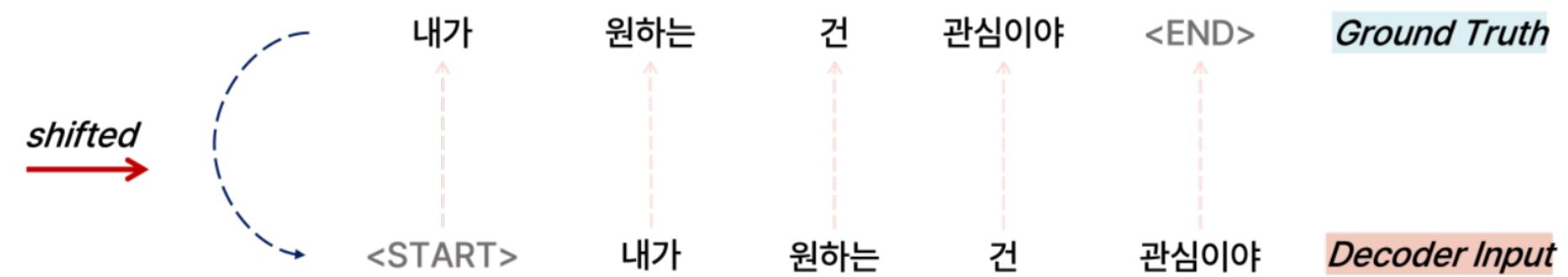
Right Shift

1) 개념

현재 토큰을 예측할 때 입력이 이전까지만 보게 맨 앞에 어떤 표현을 넣어 한 칸을 오른쪽으로 이동시키는 것
-> right shift로 입력을 y_{t-1} 로 바꾸면, 모델은 항상 이전 정보만 보고 현재 y_t 를 추론해야 함

2) 효과

인과성 유지 & 정렬: 시점 t의 입력이 y_{t-1} , 예측 대상이 y_t 로 “한 칸씩 맞물려” 정렬
자기 복사 방지: 디코더 입력에 정답 y_t 를 그대로 넣으면 '그대로 내보내기' 같은 치팅 경로가 생김
병렬 처리 가능: 한 토큰씩 돌리지 않고, 전체 시퀀스를 한 번에 넣어서 전 시점의 손실을 일괄 계산(단 미래 정 보는 mask로 차단)



TRANSFORMER

Masking

1) 개념

MHSA 이후 가려야 할 칸에 $-\infty$ (또는 아주 큰 음수)를 더함 \rightarrow softmax 후 정확히 0 \rightarrow 곱연산 후 0

2) 종류

causal masking: 디코더 self-attn에서 미래 위치를 가림

padding masking: 배치 정렬용 <PAD> 토큰을 보지도/보이게 하지도 않게 함(인코더/디코더 self & cross 모두 적용)

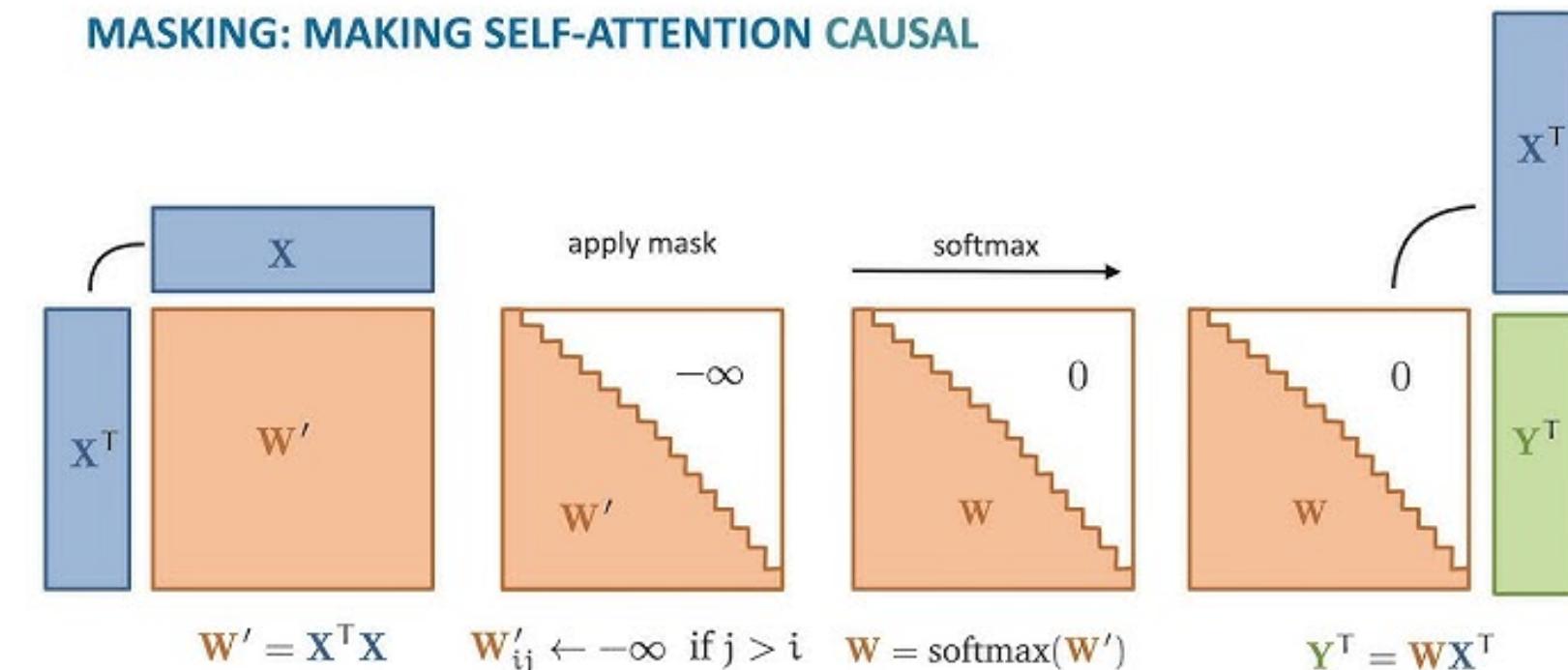
Loss mask: y_{pred} 라벨의 <PAD>를 ignore_index 처리해 loss 계산에서 제외

3) 효과

정보 누설 방지: 미래 토큰을 보면 안 됨

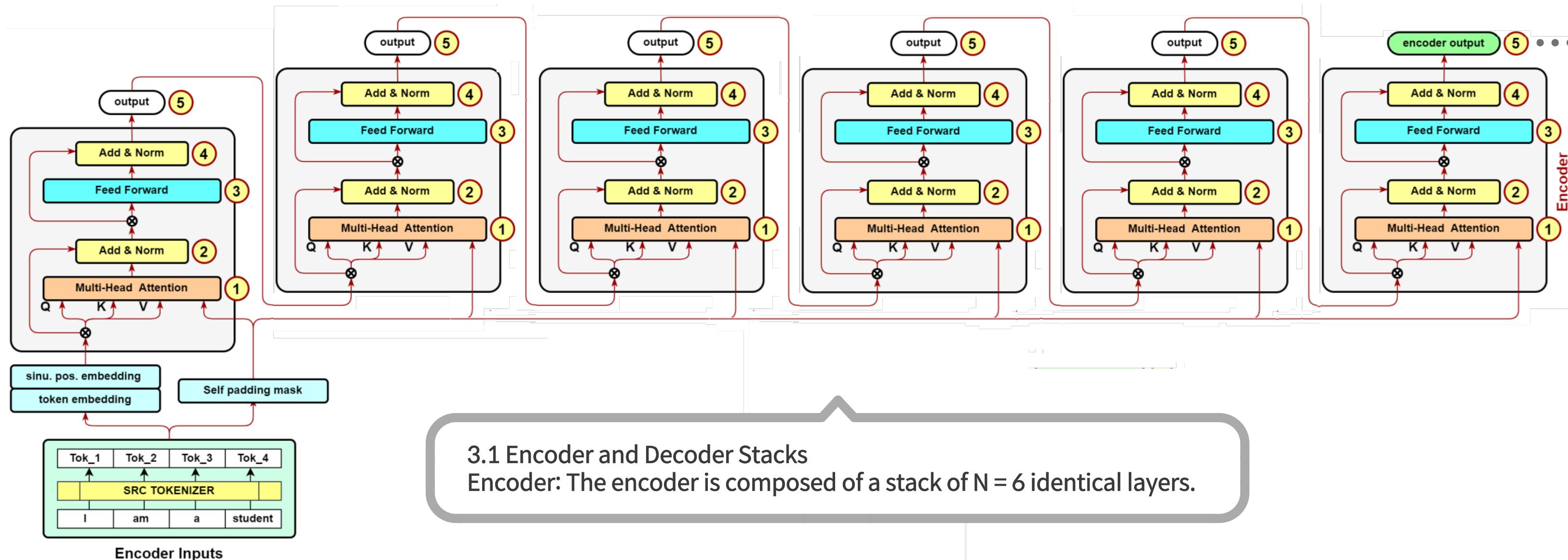
길이 다른 문장 안정 처리: PAD가 어텐션을 오염시키거나 손실을 부풀리지 않게 함

MASKING: MAKING SELF-ATTENTION CAUSAL



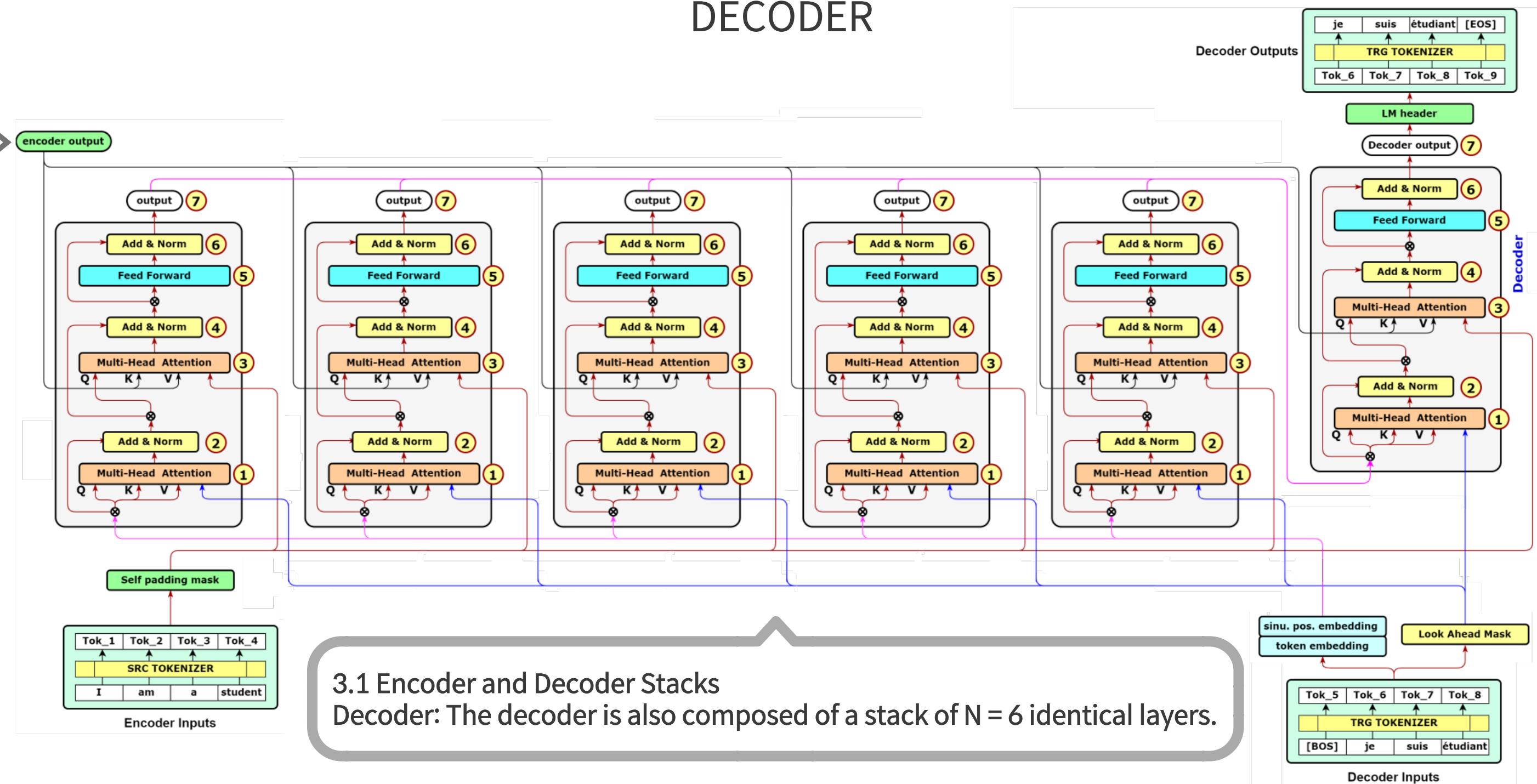
TRANSFORMER

ENCODER



TRANSFORMER

DECODER



3.1 Encoder and Decoder Stacks

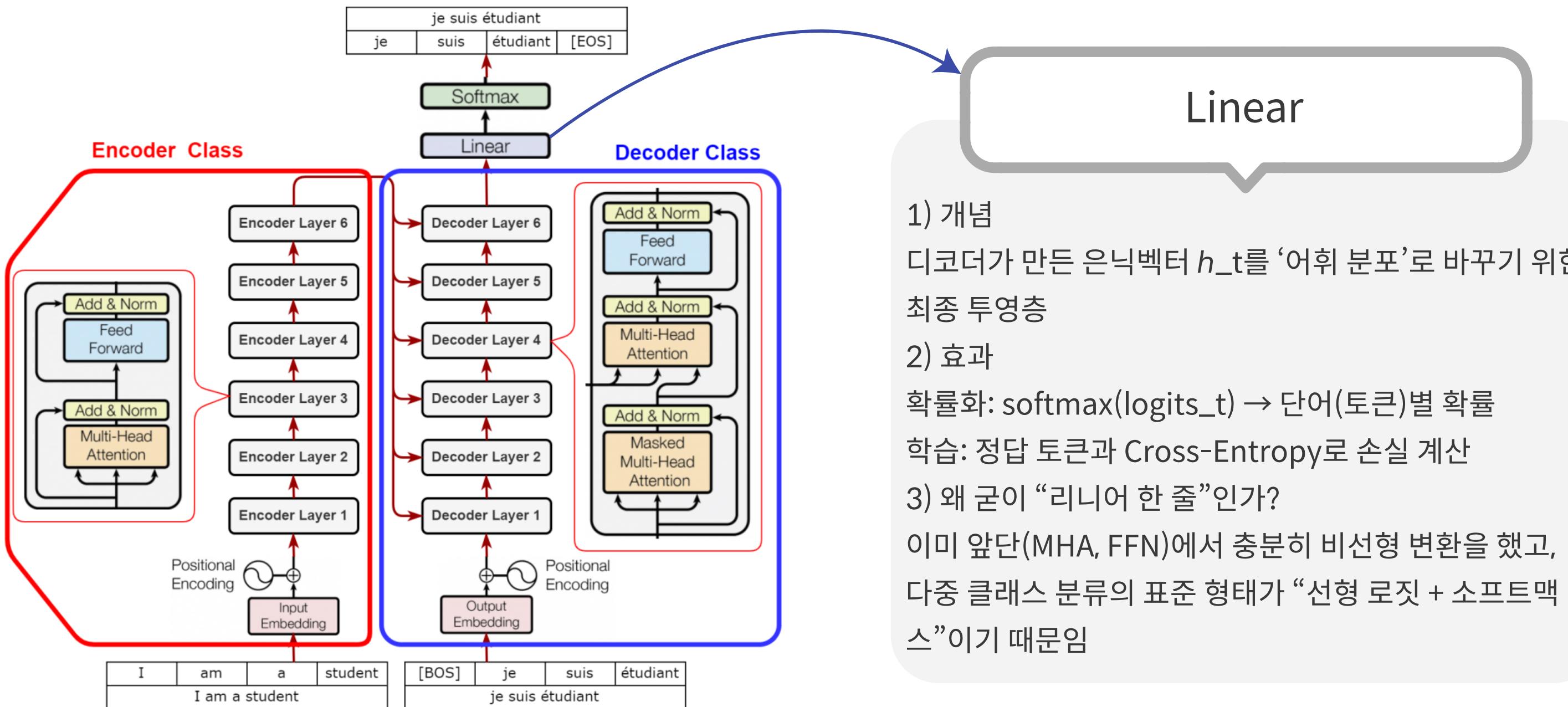
Decoder: The decoder is also composed of a stack of $N = 6$ identical layers.

TRANSFORMER

23

TRANSFORMER

ARCHITECTURE



TRANSFORMER PAPER REVIEW

CHAPTER 3

Code Implement

TRANSFORMER PAPER REVIEW

TRANSFORMER

Code Implement

Outputs

REFERENCES

- Elman, Jeffrey L. 1990. “Finding Structure in Time.” *Cognitive Science* 14 (2): 179–211.
- Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. “Long Short-Term Memory.” *Neural Computation* 9 (8): 1735–1780.
- Werbos, Paul J. 1990. “Backpropagation Through Time: What It Does and How to Do It.” *Proceedings of the IEEE* 78 (10): 1550–1560.
- Williams, Ronald J., and David Zipser. 1989. “A Learning Algorithm for Continually Running Fully Recurrent Neural Networks.” *Neural Computation* 1 (2): 270–280.
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. “Attention Is All You Need.” In *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS 2017)*.
- Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. 1994. “Learning Long-Term Dependencies with Gradient Descent Is Difficult.” *IEEE Transactions on Neural Networks* 5 (2): 157–166.
- Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. 2013. “On the Difficulty of Training Recurrent Neural Networks.” In *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*.

감사합니다.
QnA