



TTT에 오신 여러분, 환영합니다!!

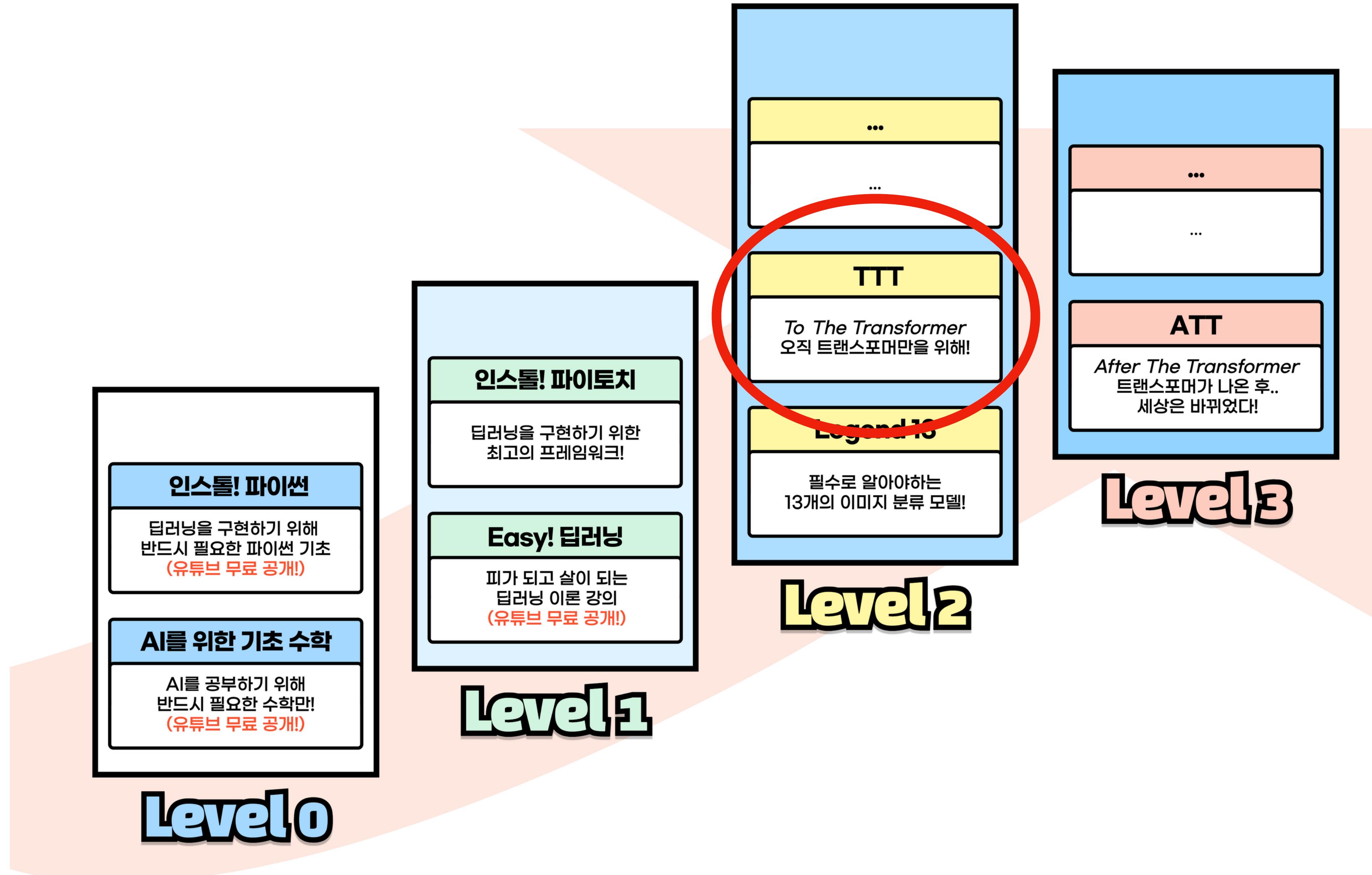


- 줌 미팅이다 보니 아무래도 현장 강의보다는 집중력이 떨어질 수 있습니다.
- **마이크를 켜놓고 참여하시는 것을 기본으로 합니다.** (주변 환경이 시끄럽다면 제외, 캠은 선택적)
 - 강제 집중(?)되는 효과를 보실 수 있습니다. 😊
 - 또, 필기할 것이 중간중간 있으니 줄릴 틈 없을 겁니다. (태블릿 PC 등, 필기할 수 있게 미리 준비해 놓으세요!)
- 질문은 생기면 바로바로 해주세요.
 - 마이크 권장, 채팅도 가능합니다.
- **1 시간당 2~3명씩 돌아가면서 집중 케어 들어갑니다.**
 - 지목받으신 분들은 마이크 켜주시면 됩니다. 그룹 과외라고 생각해 주세요.
 - 나머지 분들도 계속해서 마이크 켜놓고 적극 참여해 주세요!
- 쉬는 시간에도 질문이 가능하니, 체크만 해두고 일단 수업을 끝나오신 다음,
쉬는 시간에 돌아서 질문하시는 것도 좋습니다.
 - 이해 안 됐다고 해서 물잡고 있느라 뒤 내용 놓치지 말자는 뜻입니다!
 - 수업이 끝나고도 질문 시간 30분 주어집니다. 또, 단톡방에서도 질문 가능합니다.

< TTT: To The Transformer >

트랜스포머를 향해…!

헉펜하임 딥러닝 커리큘럼

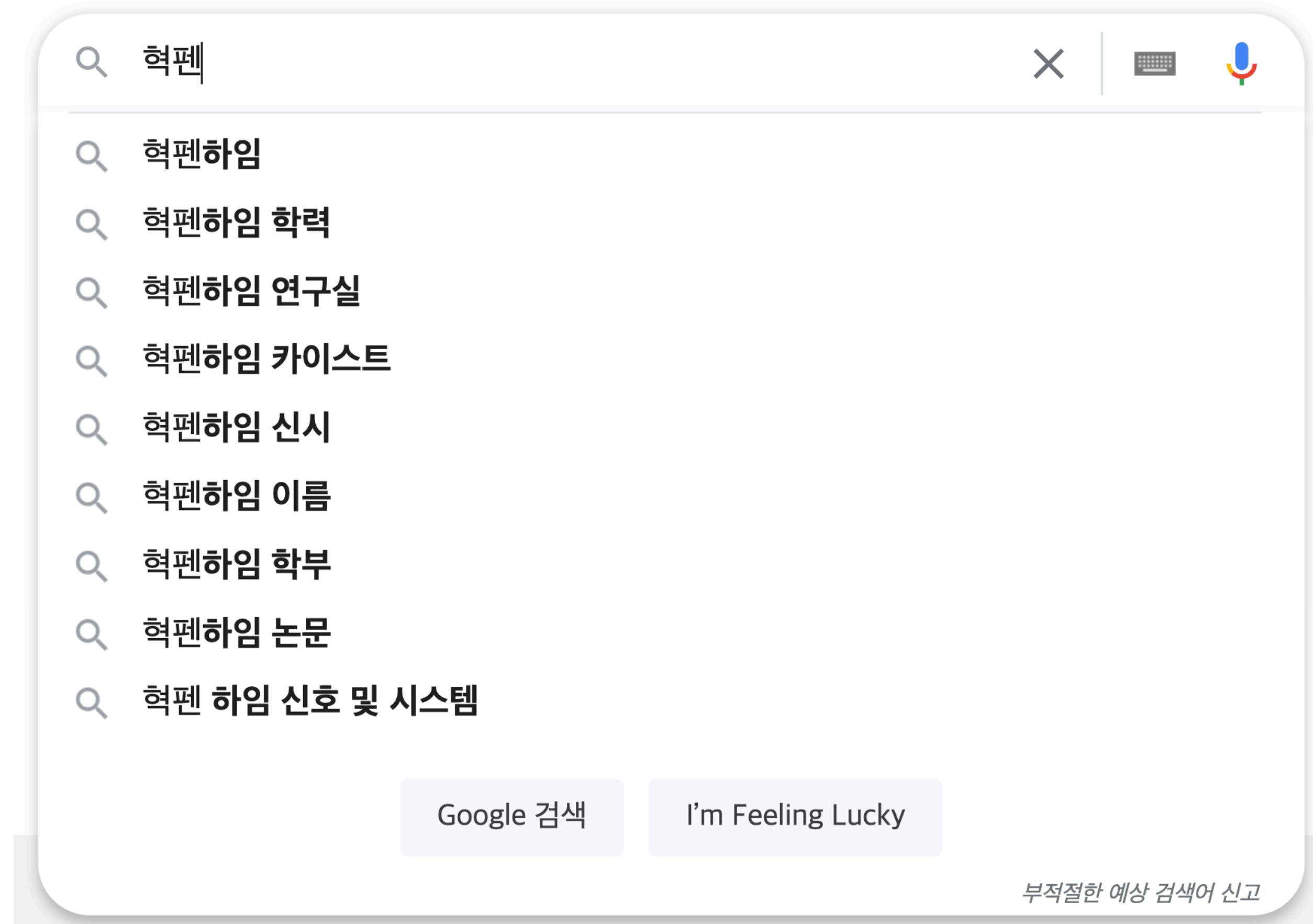


배우는 것들

- 1. RNN 이론
- 2. RNN의 구조적 한계
- 3. Seq2seq의 개념과 attention
- 4. RNN+attention의 문제점과 트랜스포머의 self-attention
- 5. 트랜스포머 이론
- 6. 한→영 번역 AI 만들기

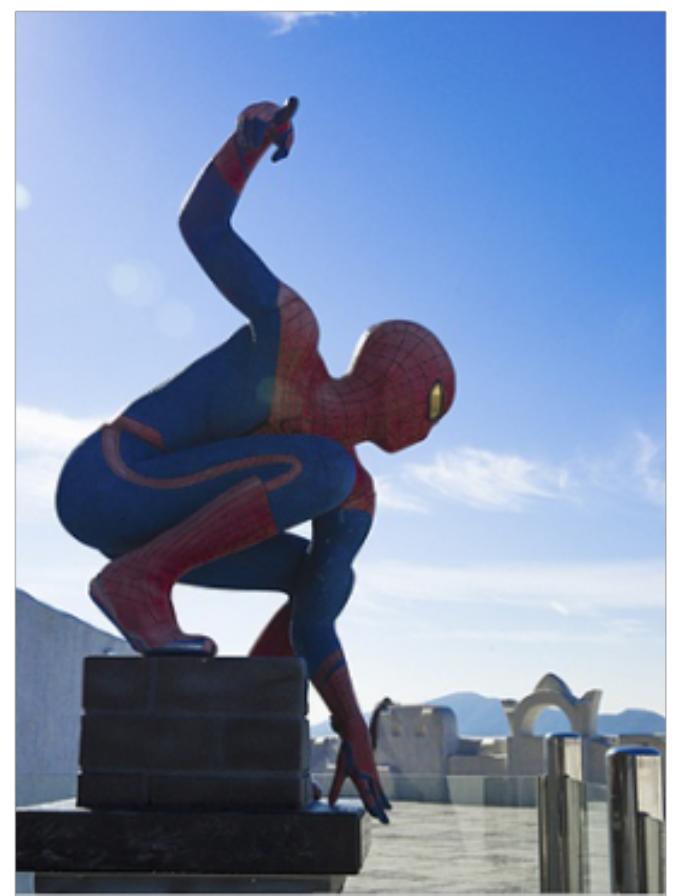
RNN 리뷰

RNN



연속적인 데이터

RNN 응용



[감상평] 원래부터 캐릭터의 팬이라면 매우 감동했을 수 있지만 글쎄...

Not Good!

주식 시장 요약 > 테슬라

932.00 USD

+222.01 (31.27%) ↑ 지난 6개월

폐장: 2월 10일 오전 8:51 GMT-5 • 면책조항

개장 전 거래 911.94 -20.06 (2.15%)

NASDAQ: TSLA

+ 팔로우

1일 | 5일 | 1개월 | 6개월 | 연중 | 1년 | 5년 | 최대



연속적인 데이터? (번역 예시)

- 단어 → 숫자로 바꾸자

저는

[1, 0, 0]

x_1

강사

[0, 1, 0]

x_2

입니다

[0, 0, 1]

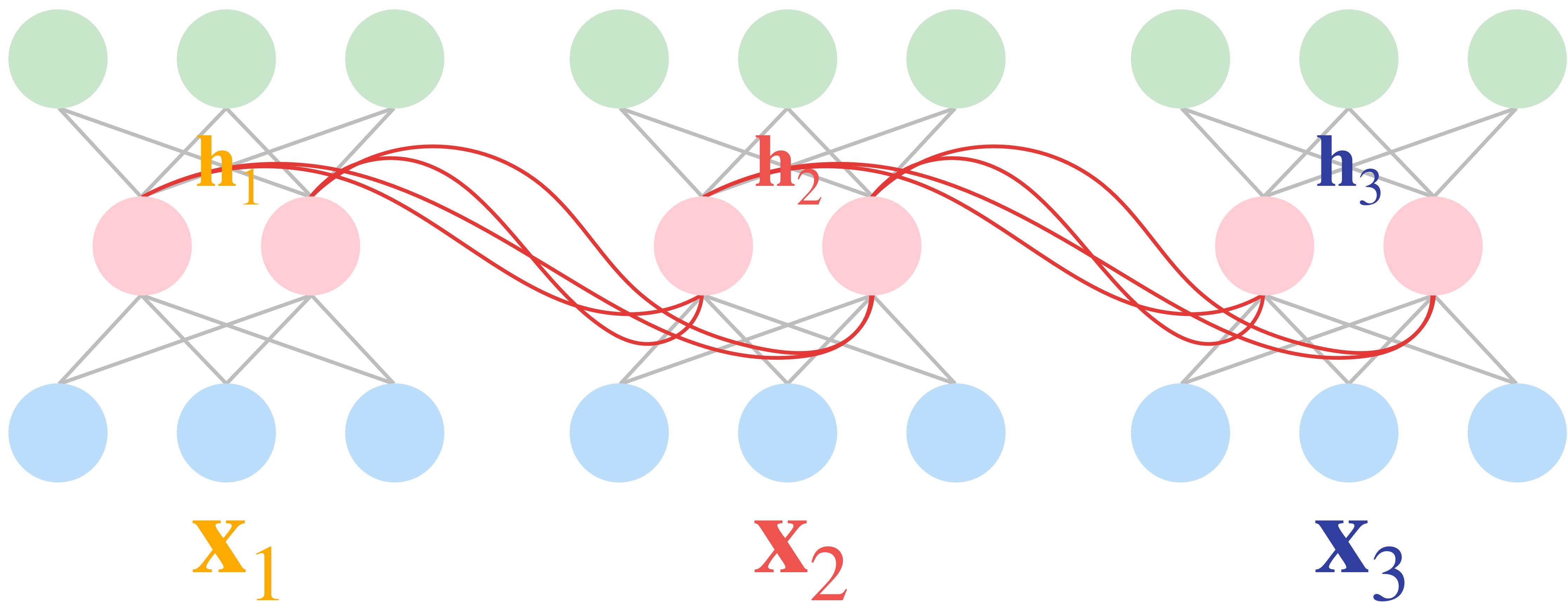
x_3

- MLP에 하나씩 **순차적으로** 넣어보자! (RNN의 접근법)

RNN 동작 방식

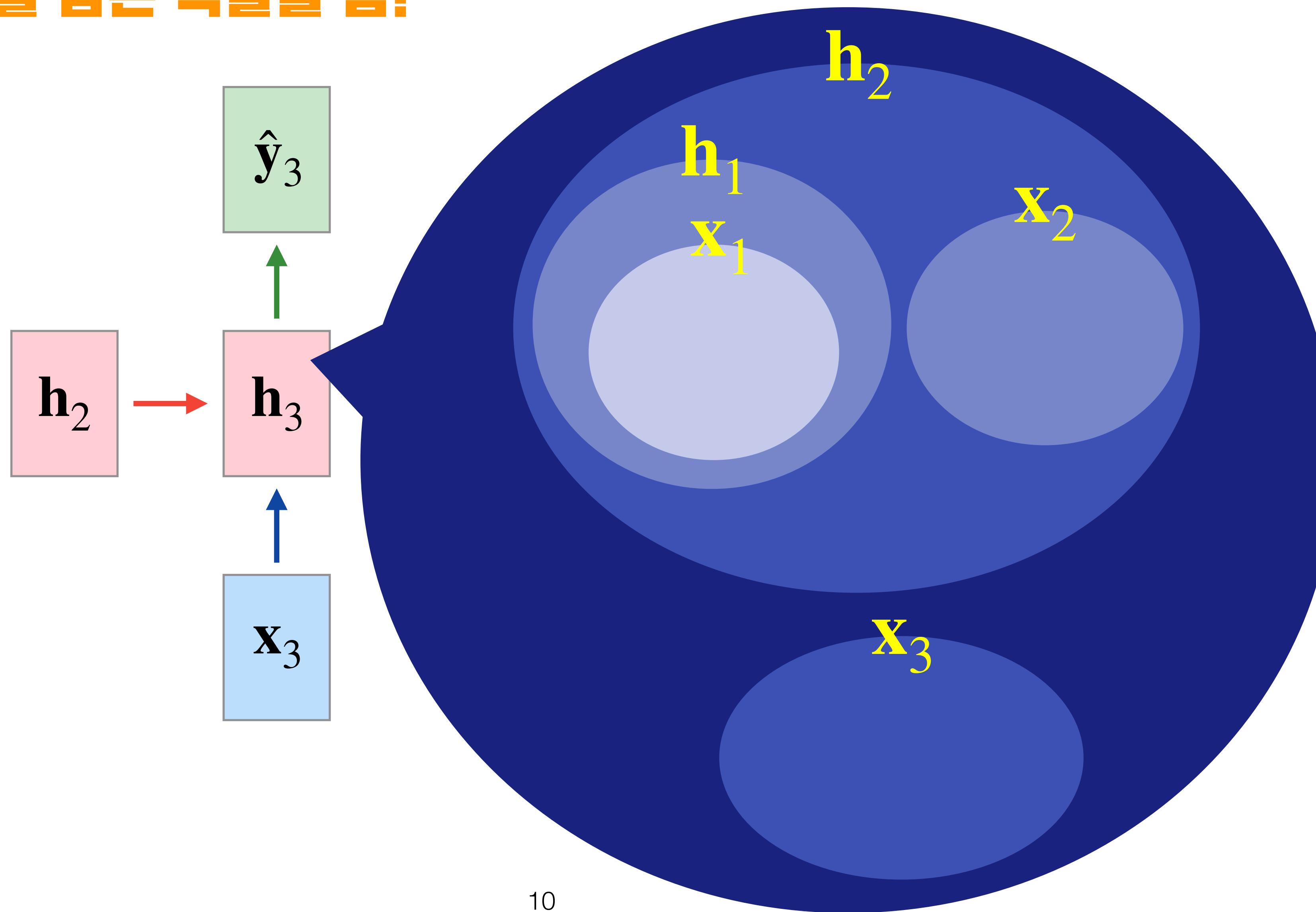
RNN: Recurrent Neural Network

- 빨간색 connection도 그냥 우리가 배운 connection임
- 이렇게 함으로써 얻는 효과는 무엇일까요?



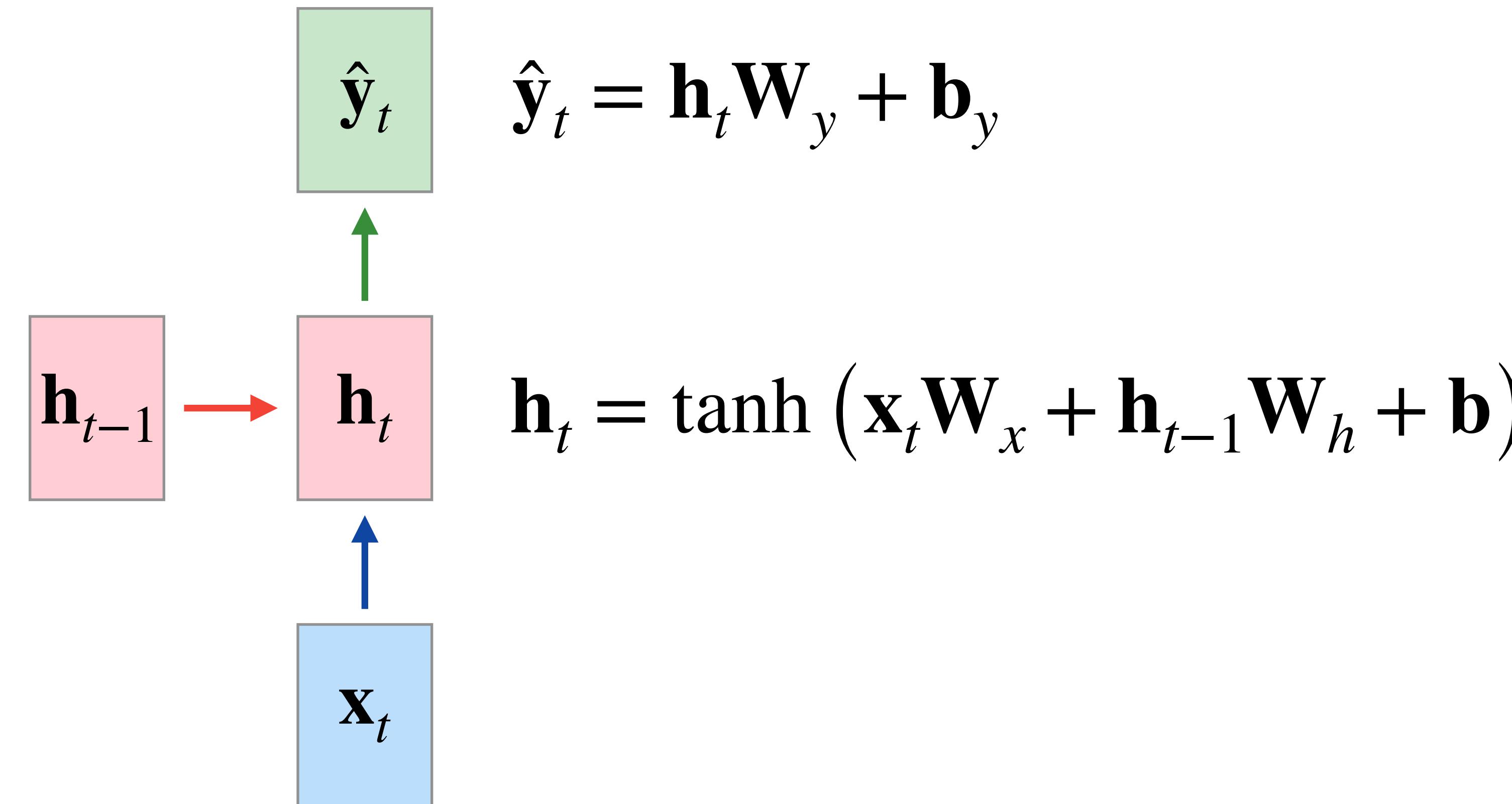
RNN은 이전 정보를 담아낸다

- **h가 이전 정보를 담는 역할을 함!**

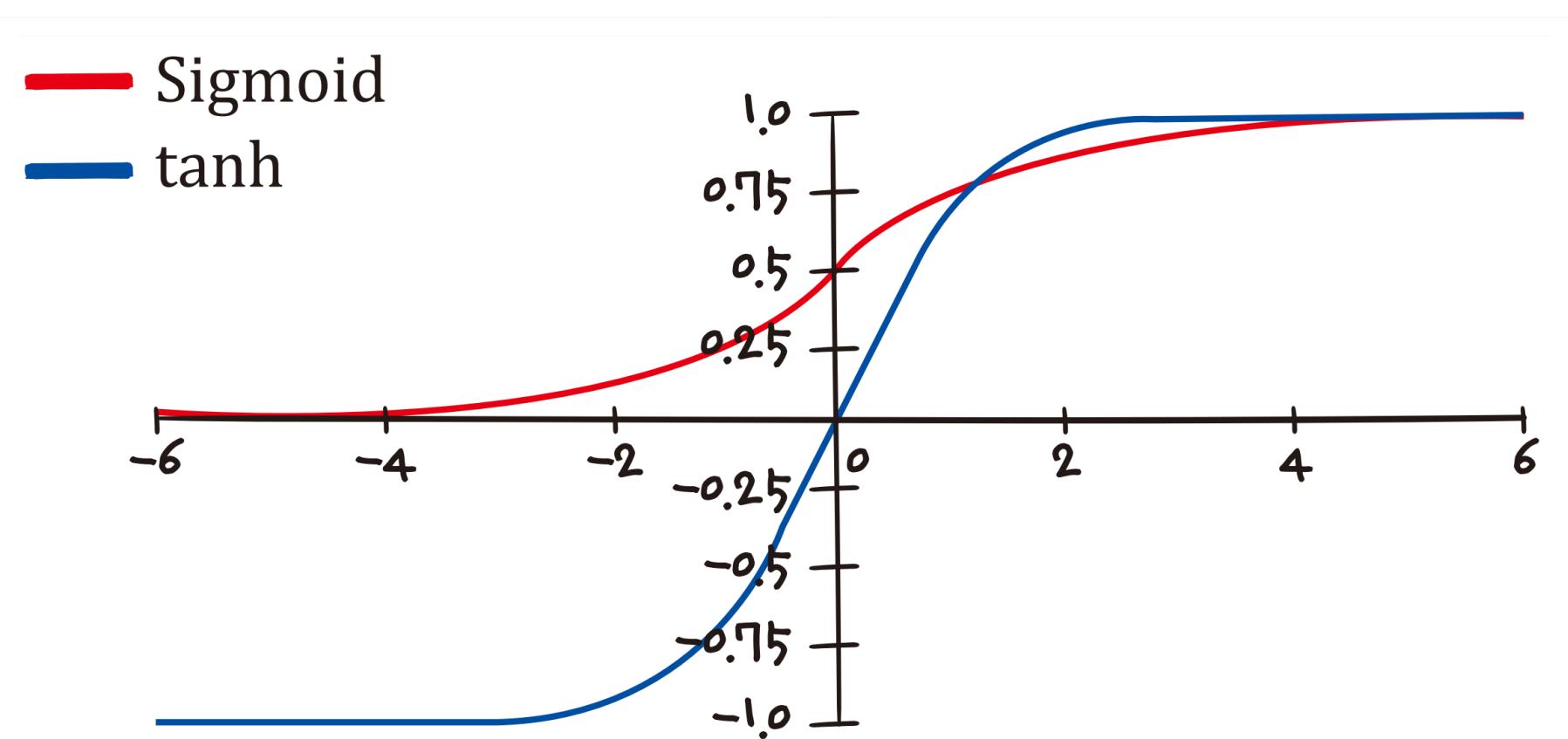
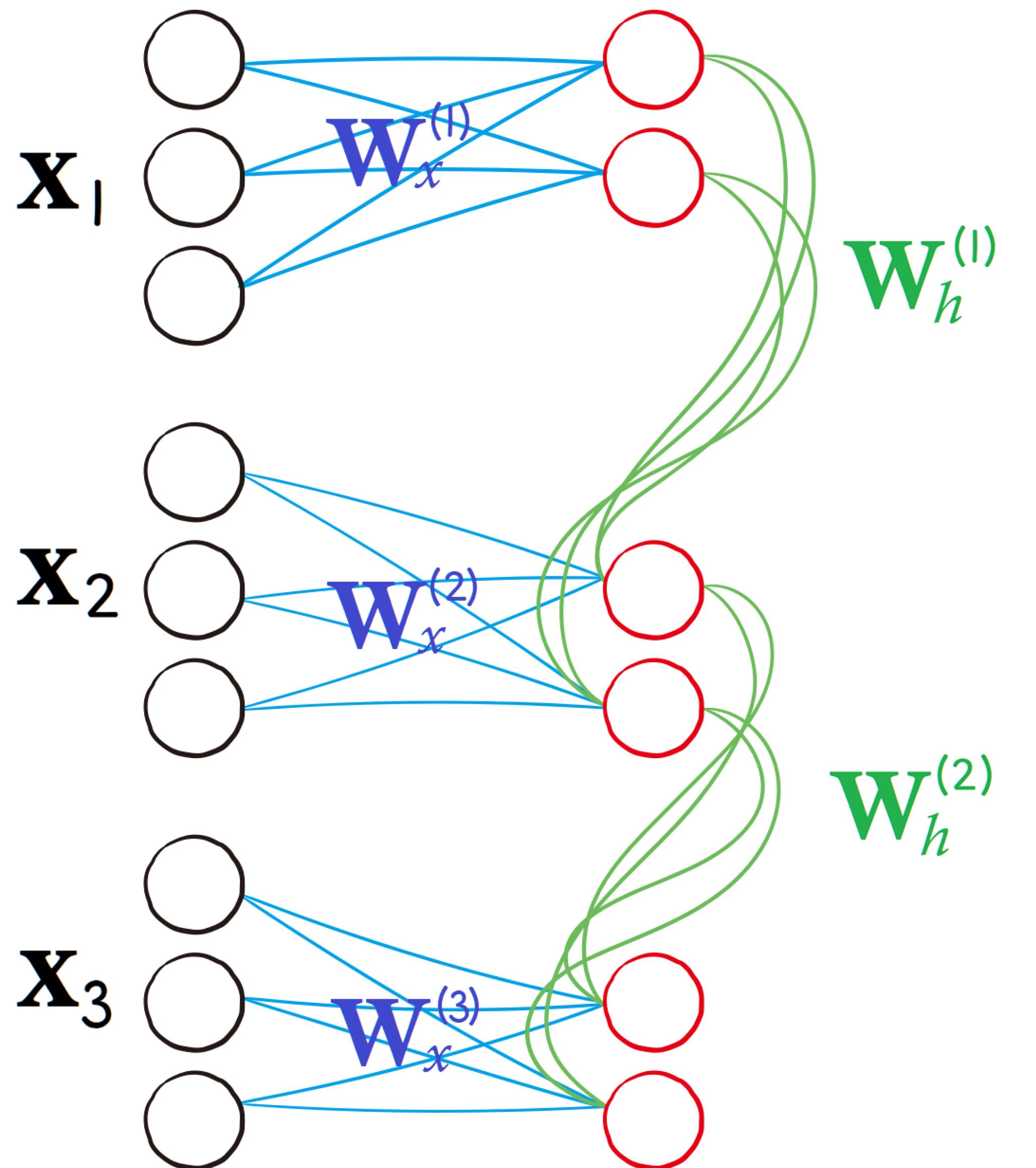


RNN 수식

- 화살표를 지날 때 FC 통과한다고 생각

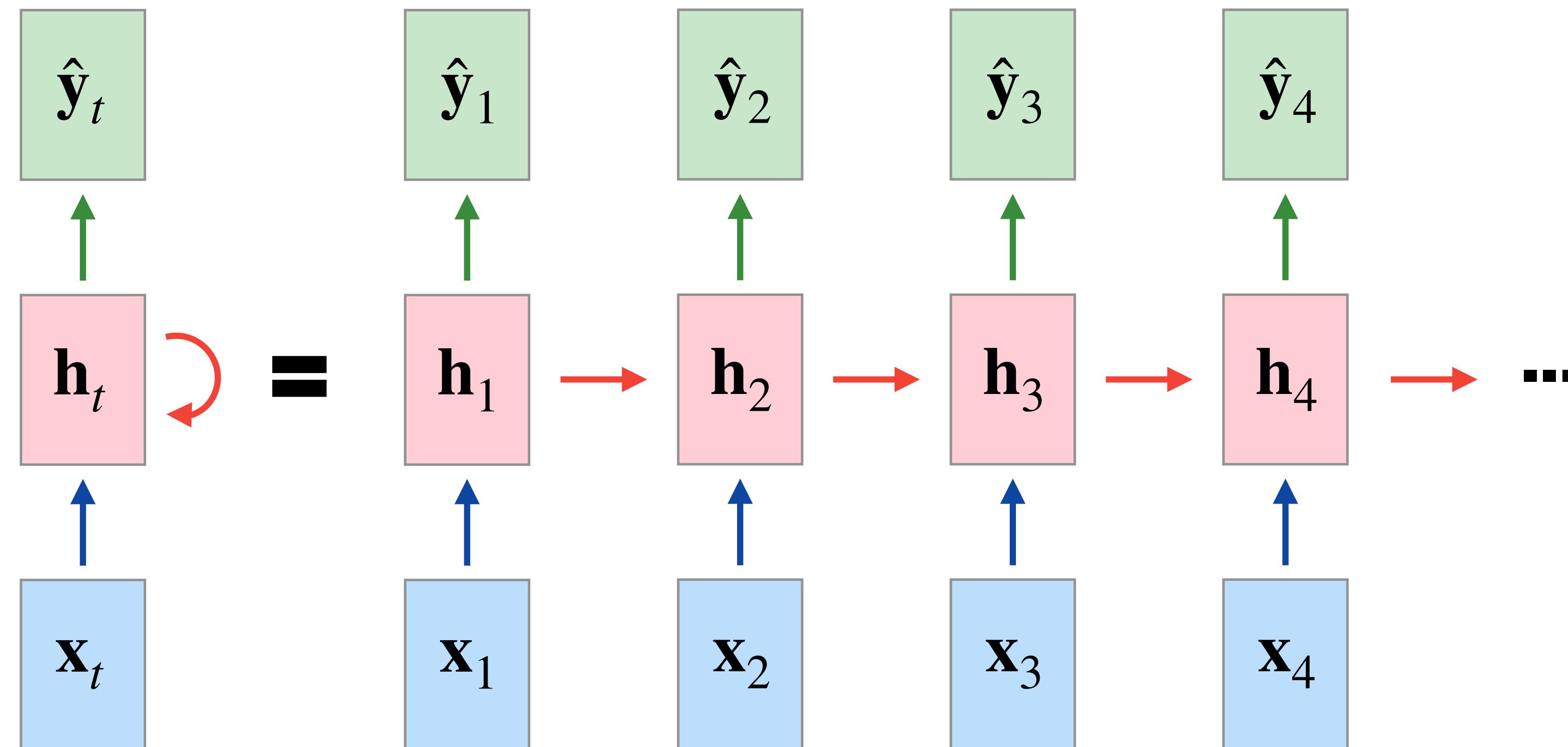


W_x, W_h, W_y, b, b_y 는 시점에 따라 다르지 않음!



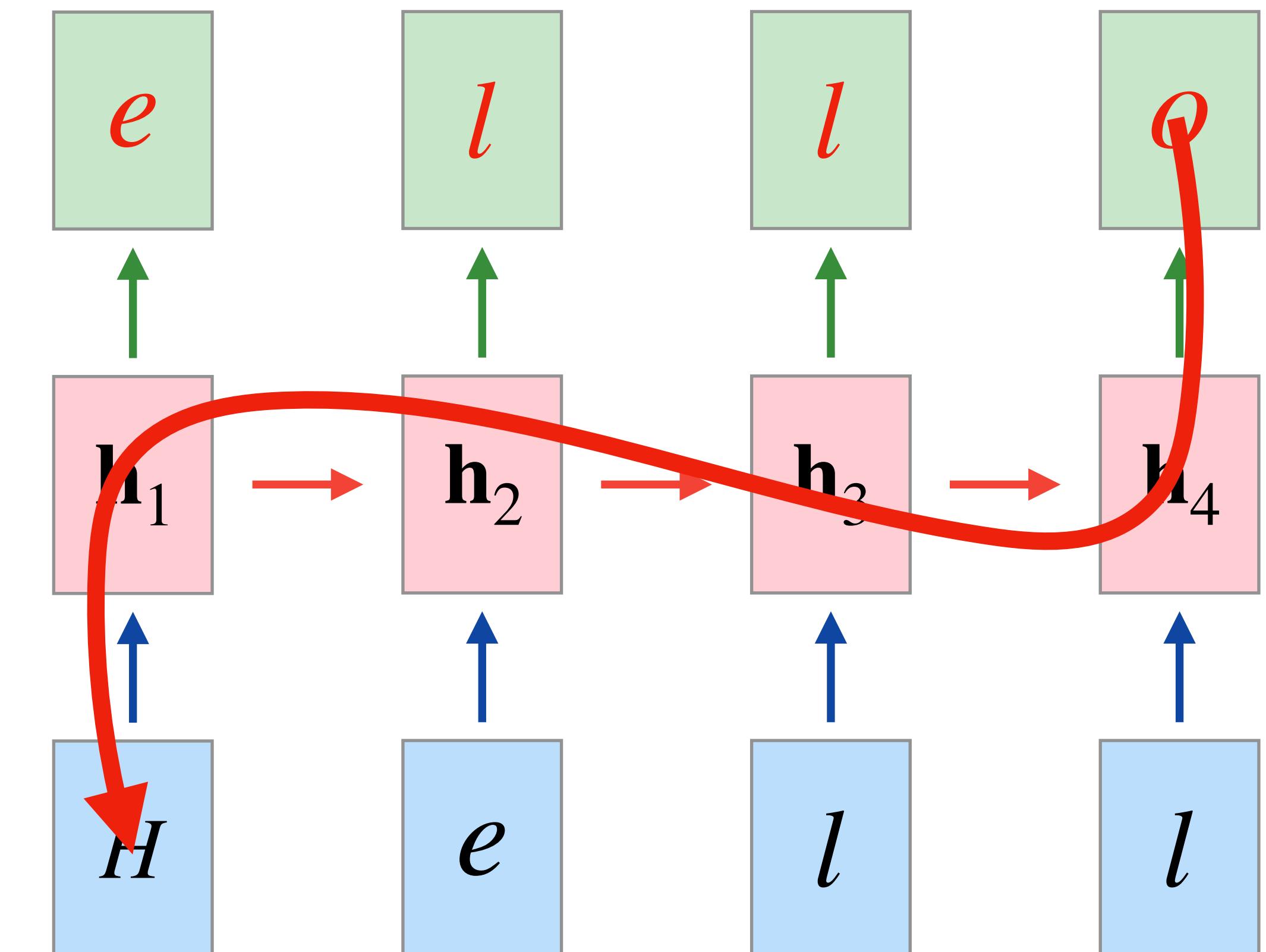
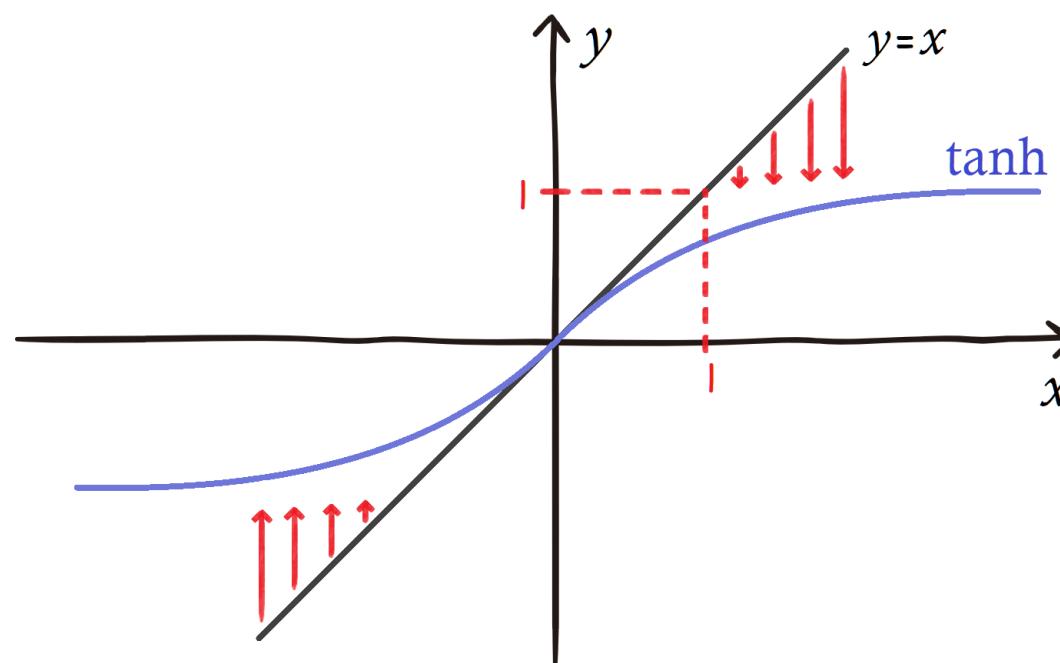
RNN 펼쳐보기

- **취향 따라 표현하세요~**



자동 완성 AI 만들기 (Hell → ello) (next token prediction: GPT)

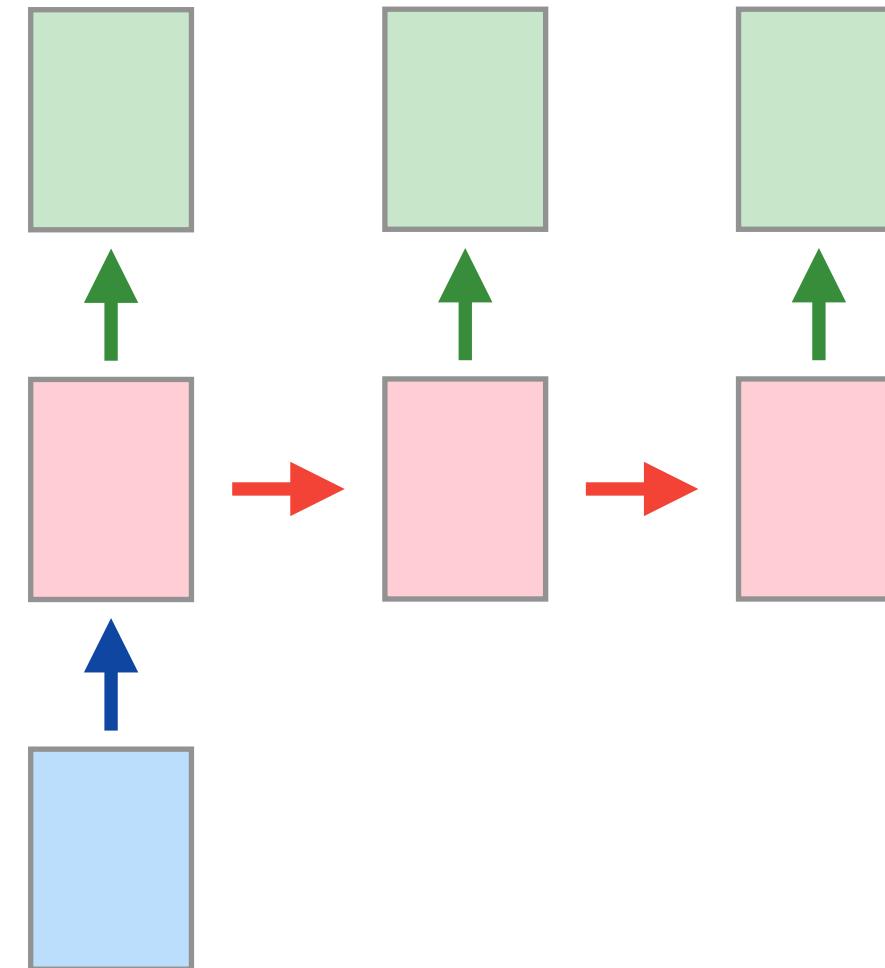
- 다음에 어떤 알파벳(One-hot encoded)이 나와야 할까?
 - => 다중 분류네!
 - => \hat{y} 을 softmax 통과시켜서 Cross-entropy 계산
 - => ello 네개 글자에 대한 Cross-entropy를 평균내서 Loss로 사용하면 되겠네~
- 근데.. o가 나오게 하기 위해 H가 gradient에 미치는 영향력?
- 멀수록 잊혀진다...! (back propagation)
- 또, 갈수록 뭉개진다...! (forward propagation)
(둘 다 tanh 때문)
- 이 둘 모두 RNN의 구조적 한계



RNN의 여러가지 유형

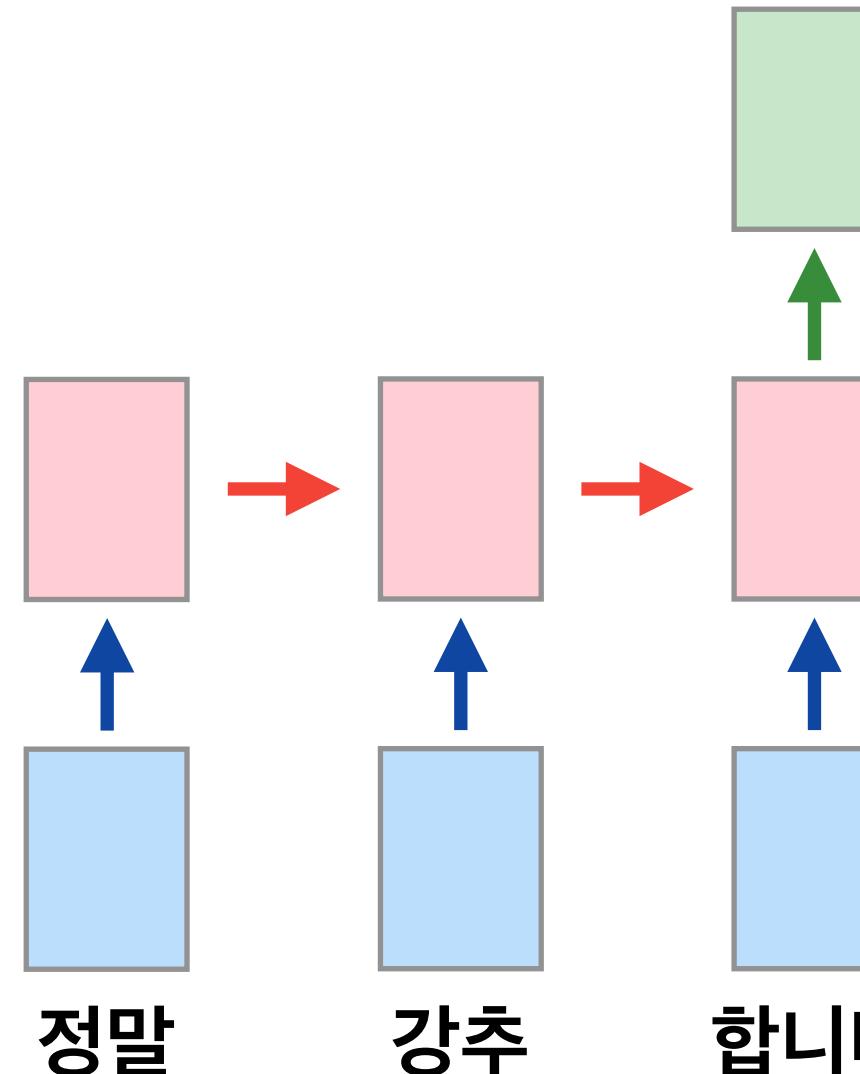
< One to Many >

Man is playing ~



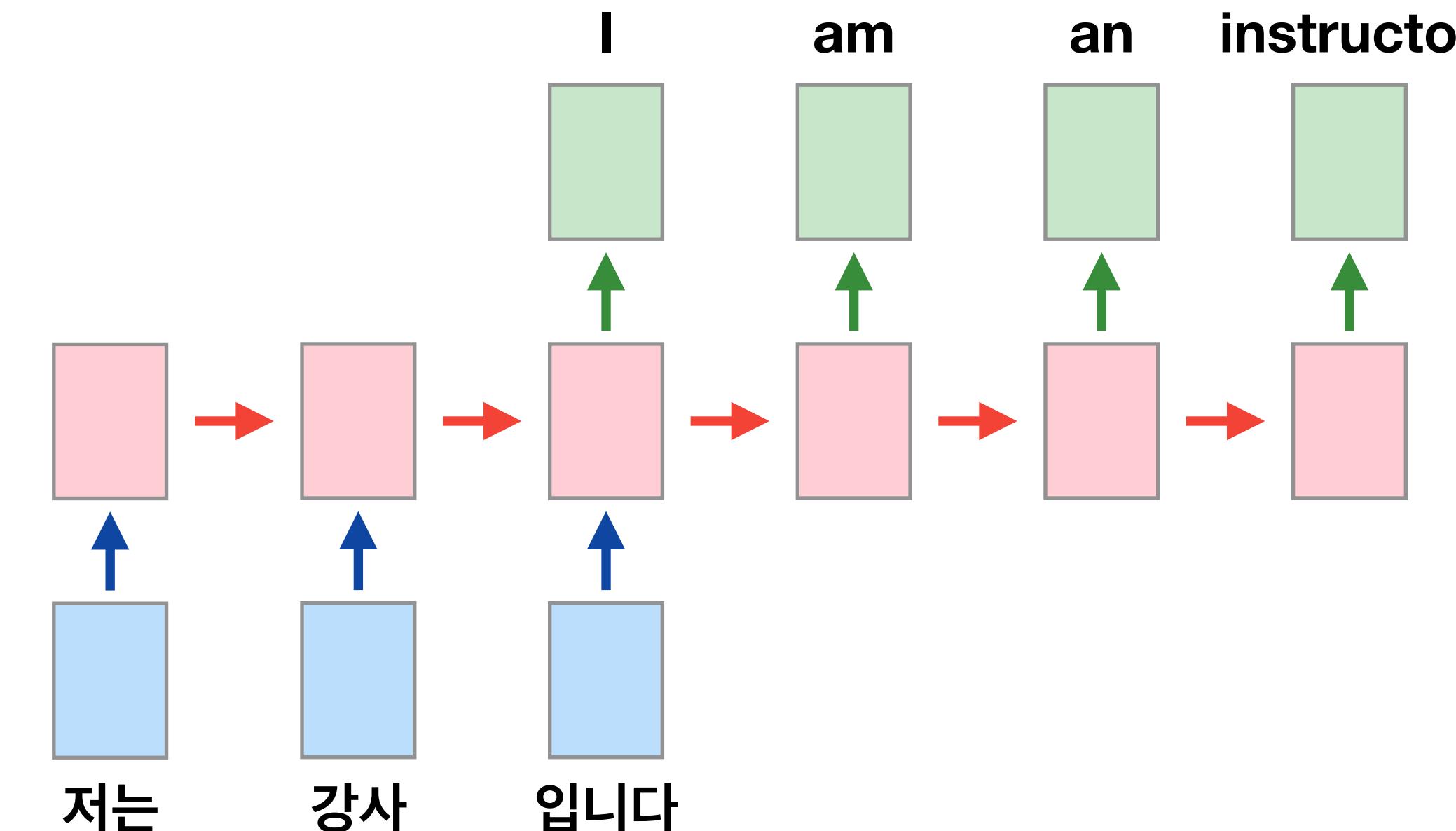
< Many to One >

긍정 지수: 0.98



< Many to Many >

I am an instructor

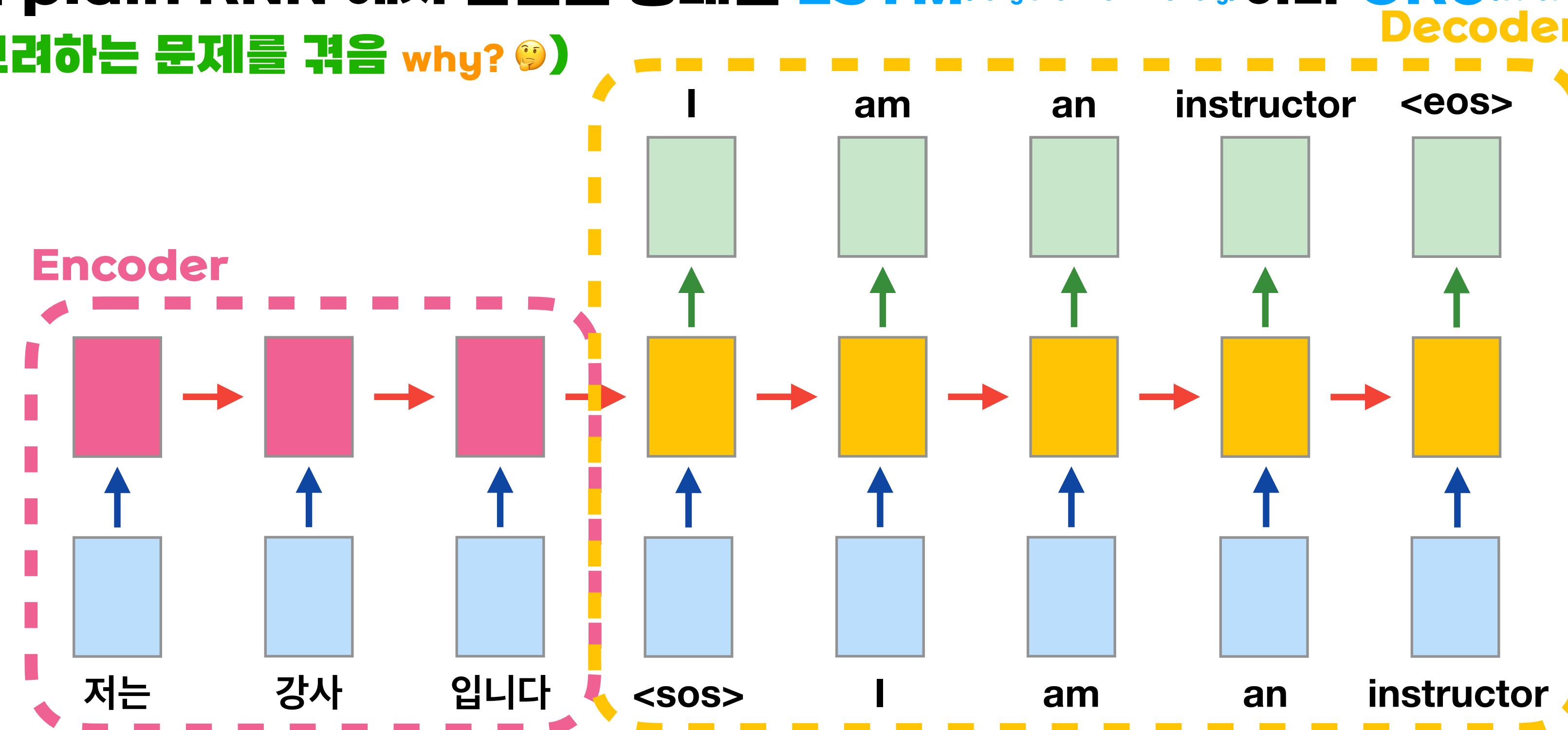


번역기는 이 구조 보단 seq2seq 구조로.. ->



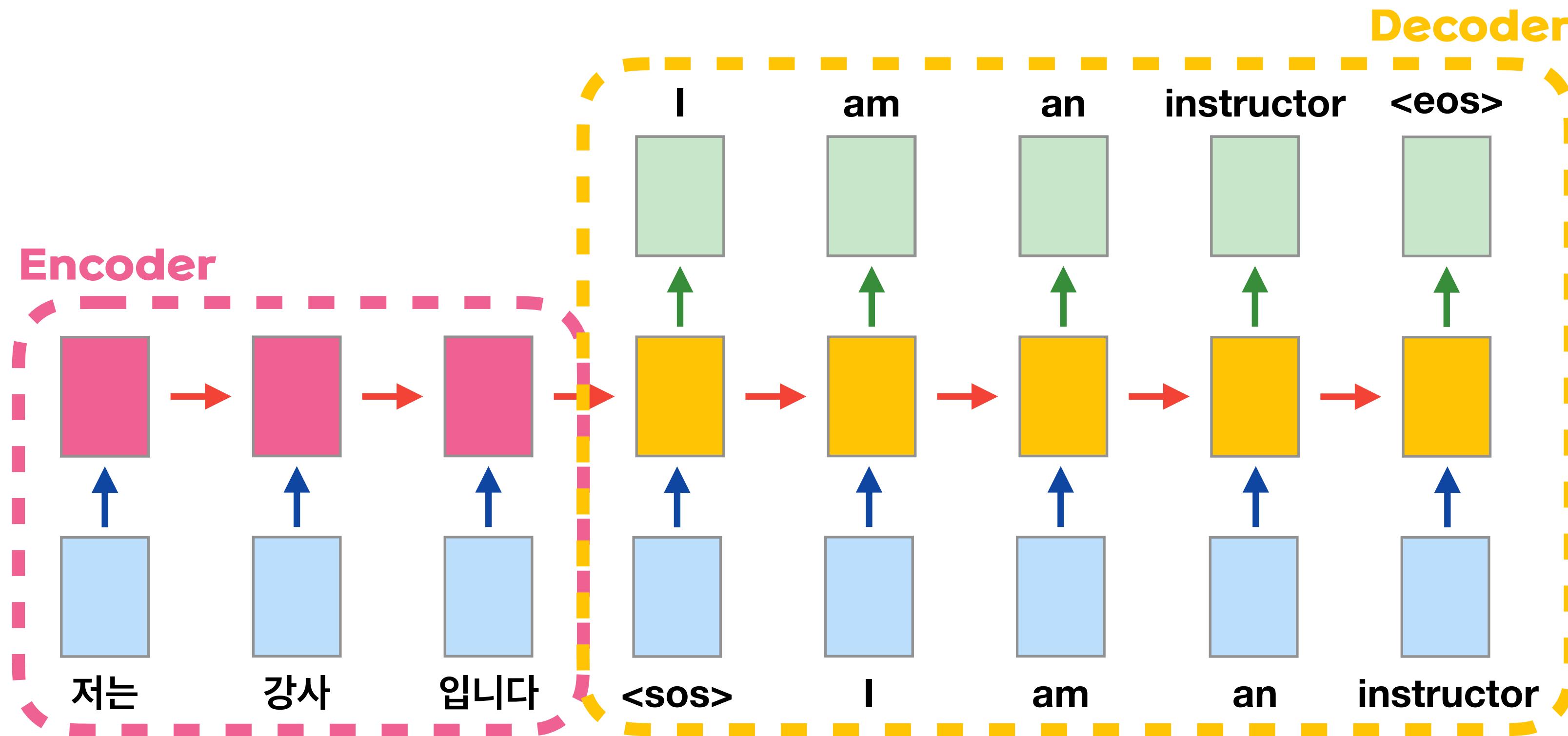
seq2seq

- Encoder, Decoder 두 파트로 구성! 이 둘은 서로 다른 RNN이다.
- Decoder는 무슨 Predictor?
그럼 Encoder의 역할은? _____ 를 잘 만들자!
- 학습 시엔 teacher forcing(지도 학습이라 가능), test 땐 출력 나온 것을 입력으로 사용!
- Encoder의 마지막 h 를 decoder의 첫 h 로 사용. 즉, 예전 h_3 를 context vector로 사용한 것
- 하나하나 cell은 plain RNN에서 발전된 형태인 LSTM_(Long Short Term Memory)이나 GRU_(Gated Recurrent Unit) 주로 사용
(이 둘도 멀면 멀 고려하는 문제를 겪음 why? 😐)



01 seq2seq 구조의 문제점

- 1. **멀수록 잊혀진다** (enc, dec 모두에 해당하는 문제)
- 2. **context vector**에 마지막 단어의 정보가 가장 뜨렷하게 담긴다
 - 첫 단어 정보 << 마지막 단어의 정보 (갈수록 뭉개진다) 인 채로 담겨있고 그런 context vector로 decoder가 번역하다보니.. 마지막 단어를 제일 열심히 본다!



Beautiful insights for RNN - 1 -

- 왜 plain RNN은 번역기로는 잘 안쓰일까?
 - 마지막 단어를 가장 중요시 본다? 인간의 사고 방식이 아님!
 - LSTM, GRU가 있다지만..근본적인 해결은 아님!
 - 이전 정보를 얼마나 끌고갈지, 현재 정보를 얼마나 담을지를 학습
- 트랜스포머는 왜 성공했나 (논문 제목: **Attention is all you need**)
 - Attention은 트랜스포머 이전에 이미 있던 개념
RNN → RNN + attention → 트랜스포머로 발전
 - 어떤 단어를 “주목”할지를 학습한다..!

Beautiful insights for RNN - 2 -

- **RNN + attention의 문제:**
멀수록 잊혀진다(decoder) + 의미를 제대로 못 담은 h에 attention 한다..!
 - **이게 왜 문제냐면, “쓰다” (x7)라는 단어의 뜻을 이해하려면 “돈을”, “모자를”, “맛이”, “글을” (x1)과 같이 멀리 있는 앞 단어를 봐야 알 수 있는데 h7에는 x1이 뭉개진 채로 들어가 있으니 x7의 “참 의미”를 못 담고 있다..! 심지어 영어라면 뒤를 봐야 할 텐데 뒤 단어들은 아예 담지도 않음.. (bidirectional RNN 써야 함!.. 근데 이것도 “거리”에 영향받는다는 건 여전..)**
그런 h에 attention을 하니 주목해야 할 것을 제대로 골랐더라도 번역을 성공적으로 하기 어렵다!
- **트랜스포머는 attention을 적극 활용,
self-attention을 통해 RNN을 완전히 버렸다..!**
 - **1. Decoder가 마지막 단어만 열심히 보는 문제 (갈수록 뭉개진다) <- E-D Attention으로 해결!**
 - **2. 디코더의 gradient의 불균형적 가중합 (멀수록 잊혀진다) <- 디코더 Self-Attention으로 해결!**
 - **3. 의미를 제대로 못 담은 h에 attention (갈수록 뭉개진다) <- 인코더 Self-Attention으로 해결!**

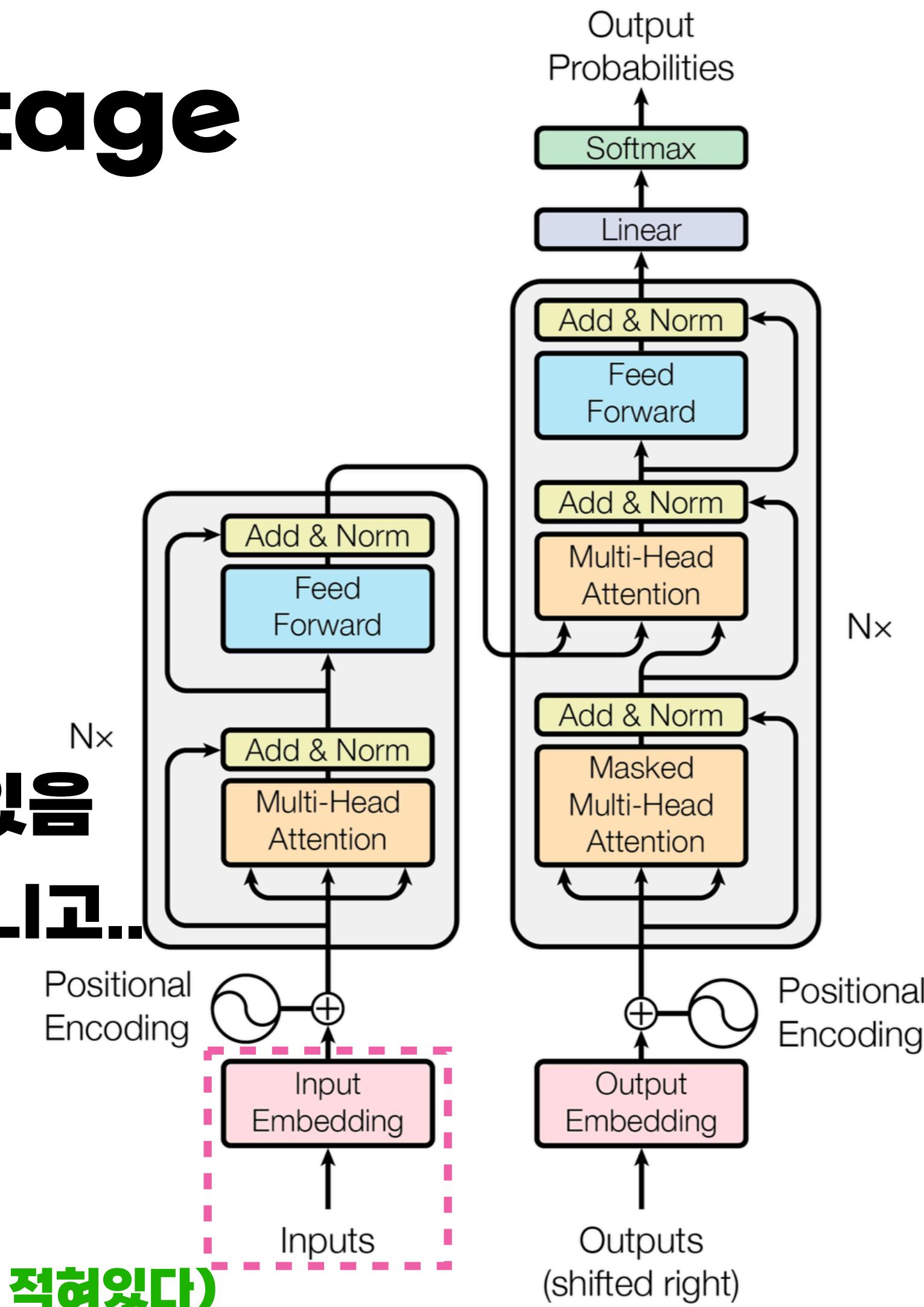
Transformer – Attention is all you need (2017.06)

Transformer – Attention is all you need (2017.06)

- 자연어 처리 (Natural Language Processing) 분야를 지배한 모델 (번역기 예시로 설명해볼게요)
- self-attention을 이용, 각 단어를 숫자로 잘 바꿨다! (잘? 의미를 잘 담음!)
- self-attention을 한마디로 표현하면 그냥 내적을 이용한 weighted sum!
 - 어떤 단어들과 함께 등장했는지를 살펴보고 (attention) 워드 임베딩 벡터에 의미를 잘 담음!
 - 입력 문장도, 출력 문장도 self-attention을 사용해서 단어를 숫자로 잘 바꿔 놓는다
- 또, Encoder-Decoder Attention을 통해 번역할 때 어떤 단어를 주목해야 할지를 학습시켰다
- CNN은 conv를 사용해서 이미지에서 어떤 특징을 추출할지를 학습시켰다면
트랜스포머는 내적과 가중합을 사용해서 다음 단어를 예측할 때 어떤 단어를 주목할지를 학습시켰다!
- 단순히 어떤 연산을 적용했을 뿐인데 이런 해석이 가능하다는 것이 놀라운 점!

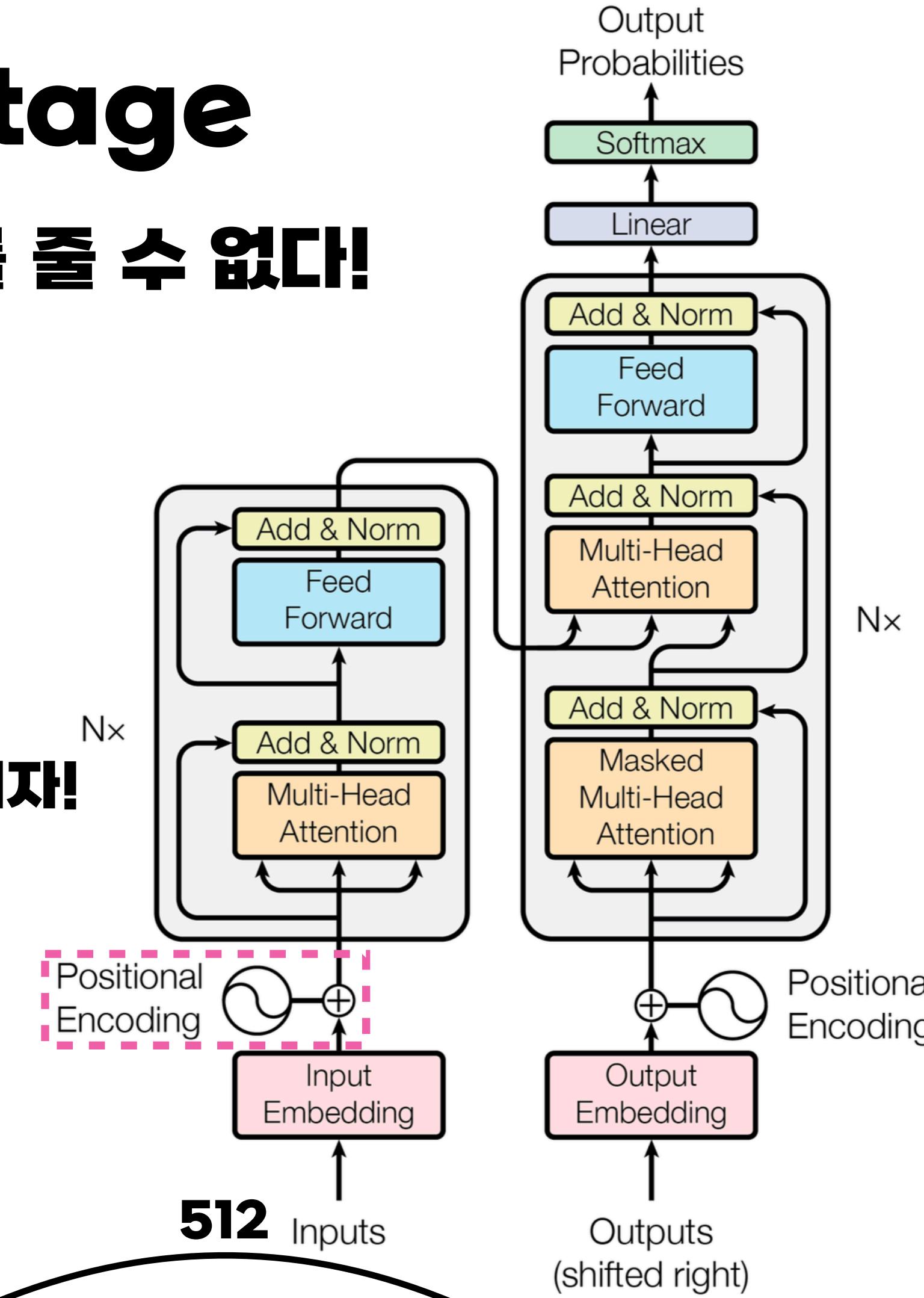
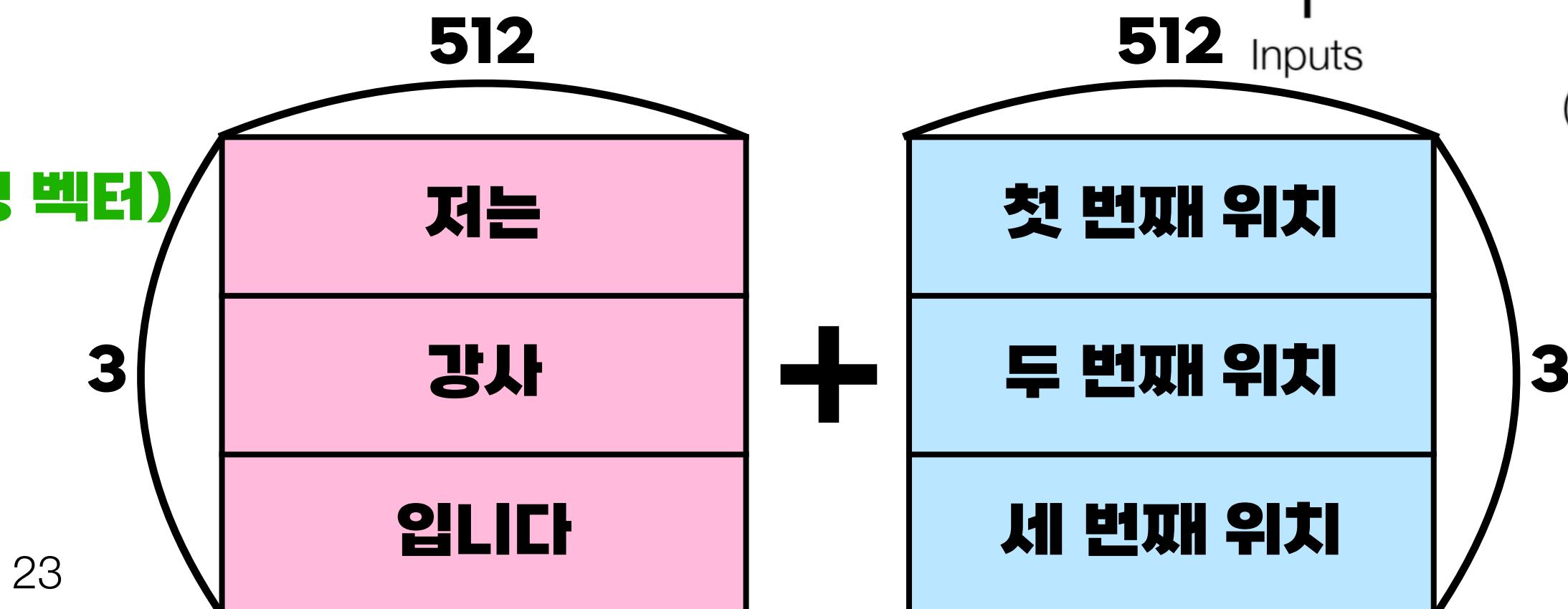
Transformer – The first stage

- 이미지에선 Input shape이 개별 행렬 NLP에선 개단자
 - 연습! $32 \times 50 \times 512$ 의 의미는..?
 - 예를 nn.Linear(512, 64)에 통과하면 결과 shape?
 - 근데 문장마다 단어의 수가 다를텐데 어떻게 한 tensor에 담지..?
 - <pad> 토큰! (이것도 하나의 단어로 취급, one-hot 되어 있다)
 - 맨 처음 Inputs는 $32 \times 50 \times 7851$ 이런 식으로 one-hot 되어있음
 - 즉, 실제로 저는 = [1 0 0], 강사 = [0 1 0], 입니다 = [0 0 1] 이게 아니고.
저는 = [0 0 0 ... 1 ... 0 0 0] (1482 번째에 1 나머지는 0)
강사 = [0 0 0 0 0 ... 1 ... 0] (5821 번째에 1 나머지는 0)
입니다 = [0 ... 1 ... 0 0 0 0 0] (243 번째에 1 나머지는 0)
이런 식으로 되어있다는 뜻! (실제 구현에서는 32×50 으로, 1482와 같은 인덱스만 적혀있다)
 - 따라서, 위에서 7851? 내가 써먹을 수 있는 한글 단어의 총 개수!
 - $32 \times 50 \times 7851$ 을 nn.Linear(7851, 512) 통과(행 뽑기)시키면 그게 워드 임베딩 벡터! ($32 \times 50 \times 512$)
(실제 구현에서는 nn.Embedding 사용)



Transformer - The first stage

- 하지만, 이렇게만 하면 RNN 과 달리 단어의 순서에 대한 정보를 줄 수 없다!
 - 단어의 위치를 one-hot ($32 \times 50 \times \text{max_len}$) 해서
`nn.Linear(max_len, 512)` 통과시키면 그게 위치 임베딩 벡터!
 - 예를 들어, 첫 번째 위치 = [1 0 0 0 ... 0]
두 번째 위치 = [0 1 0 0 ... 0]
세 번째 위치 = [0 0 1 0 ... 0] 얘네를 `nn.Linear` 통과시키자!
 - 여기서 `max_len`은 모델이 받아들일 최대로 고 문장의 길이!
(하이퍼파라미터임. 낙낙잡으면 됨. ex) `max_len=100`로 잡자
 - 즉, `nn.Linear(100, 512)` 통과시키면
 $32 \times 50 \times 100 \Rightarrow 32 \times 50 \times 512$
(얘도 실제 구현에서는 `nn.Embedding` 사용)
 - 이제 둘을 더하면 (워드 임베딩 벡터 + 위치 임베딩 벡터)
`self-attention`할 준비 완료!



근데 왜 순서 정보가 중요할까요?

- “아니 다 오르는데 왜 테슬라만 떨어져??”

VS

“아니 테슬라만 오르는데 왜 다 떨어져??” (비현실)

- 만약 위치 임베딩이 없었다면 ‘테슬라’ 가 어디에 있어도
‘테슬라’라는 단어의 one-hot vector는 똑같기 때문에 똑같은 값으로 임베딩 될 것!

=> 즉, 위치가 달라졌음에도 다른 의미인 것을 모른다! (현실은 폭락의 아이콘이면서!! 😢)

=> 따라서, 순서 정보를 알려줘야!

- 하지만, 몇 번째 행에 해당 워드 임베딩 벡터가 있느냐로 이미 순서 정보를 가지고 있는데
위치 정보를 왜 굳이 만들어 줘야 하나요..?라는 질문이 있었어요.

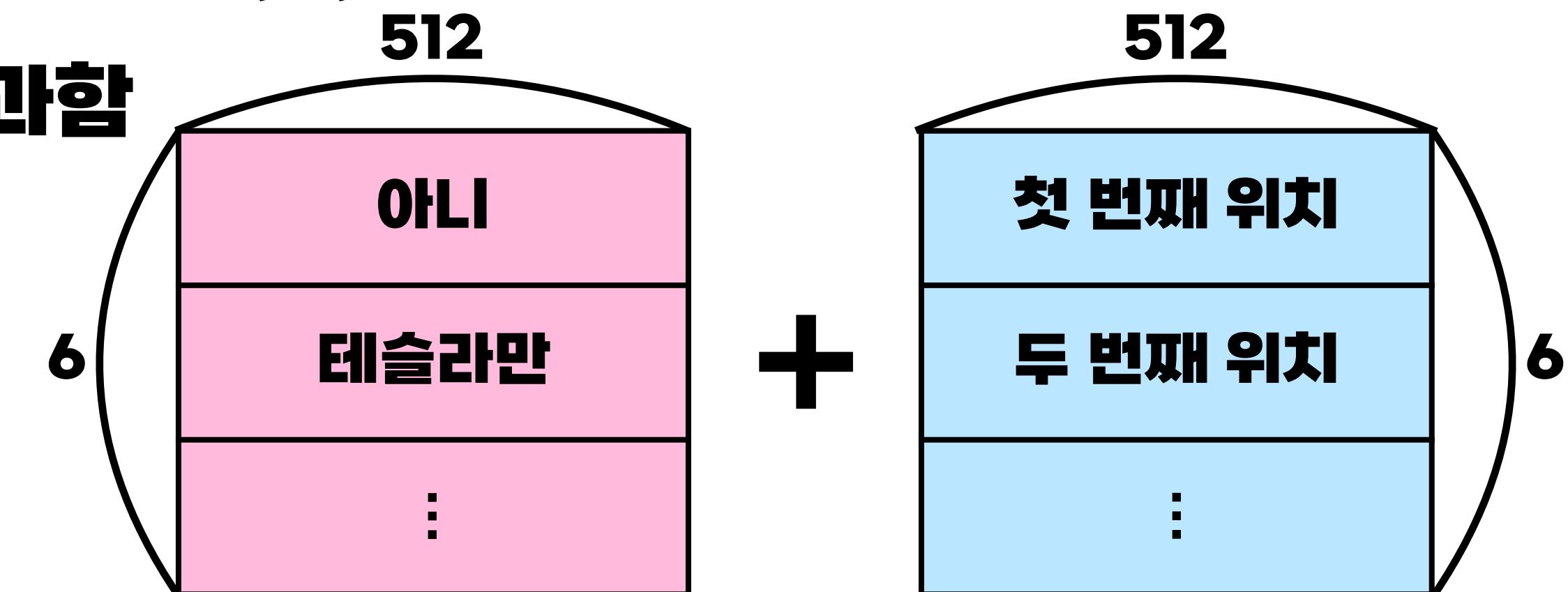
- 일단 모든 시점의 단어들이 동일한 레이어를 통과함

(이건 RNN도 마찬가지이고 함)

- 또, 어텐션에서 섞여버림!

(아니 + 다 + ... vs

아니 + 테슬라만 + ... 이런 식! ∴ 위치 구별 안 됨)



1. Positional Embedding

- 처음에 보여드린 `nn.Linear(max_len, 512)` 통과시키는 방식은 말하자면
테슬라(워드 임베딩 벡터) + **다섯번째**(위치 임베딩 벡터 <- 학습 파라미터로!)
와 같이 위치 정보를 아예 따로 학습시키자는 것!
- 위치에 따라 어떤 벡터를 더해줄지 내가 고민하기 귀찮아. 그냥 AI 님이 알아내!**
- 구현 디테일) 단어 정보를 더 고려 하게끔 워드 임베딩 벡터에 $\sqrt{512}$ 를 곱하고
위치 임베딩 벡터와 더함. ($\sqrt{512} \times$ 테슬라 + **다섯번째**)

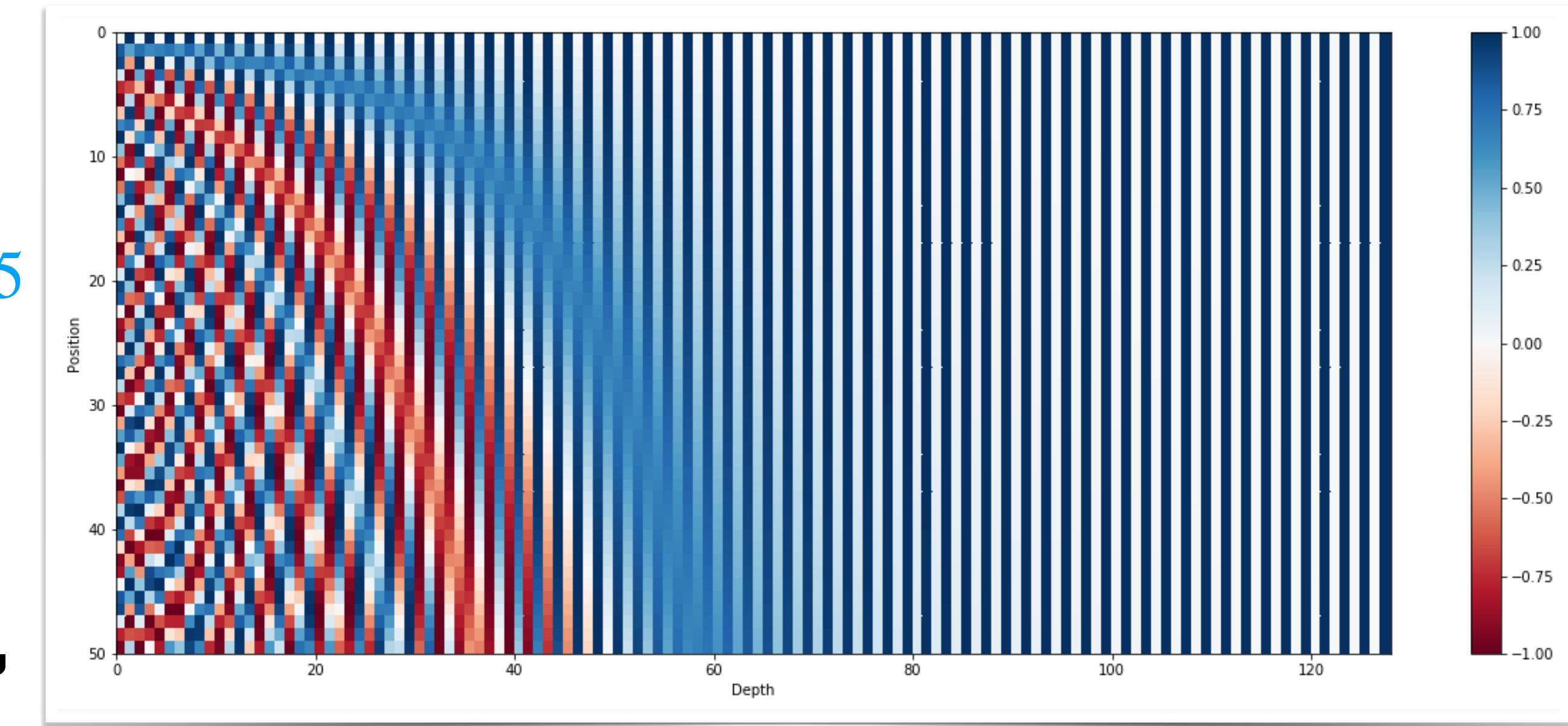
2. Positional Encoding

- 트랜스포머 논문에서는 전 슬라이드 설명처럼 학습시키진 않았고 위치별로 다른, 고정된 벡터 사용

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), pos \text{ 는 단어 위치},$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right), i = 0, \dots, 255$$

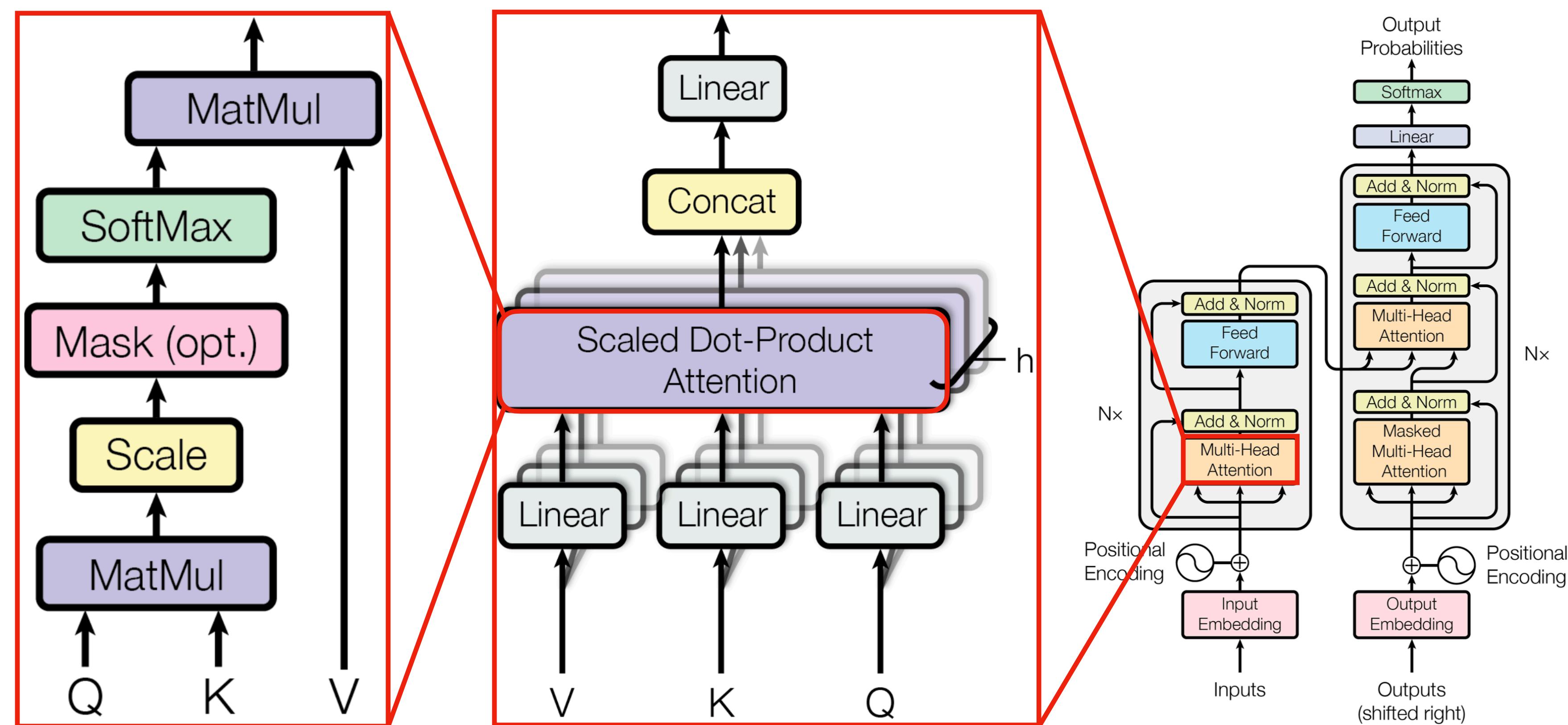
- **sin 값, cos 값이 번갈아 나오는 특이한 형태**
- 간단히 말하자면, 그림에서
0 번째 가로줄을 0 번째 단어의 임베딩 벡터에,
1 번째 가로줄을 1 번째 단어의 임ベ딩 벡터에
에 더해주는 것!



reference: https://kazemnejad.com/blog/transformer_architecture_positional_encoding

트랜스포머가 학습하는 layer는?

- 어떤 단어를 더 볼지를 학습시키기 위해 **내적**을 사용한다!
- 그럼 어떤 단어를 더 볼지를 AI가 학습해서 알아내야 할 텐데..
내적은 파라미터가 필요한 연산이 아님..! 그럼 대체 뭘 학습 파라미터로 삼는가..?
- **내적할 워드 임베딩 벡터를 선형 변환(=이동시키는 행위)하는 FC layer가 학습된다!**



Query & Key & Value

두 번째 단어에 대한 self-attention:

$$\mathbf{h}_2^{\text{new}} = \langle \mathbf{h}_2^Q, \mathbf{h}_1^K \rangle \mathbf{h}_1^V + \langle \mathbf{h}_2^Q, \mathbf{h}_2^K \rangle \mathbf{h}_2^V + \langle \mathbf{h}_2^Q, \mathbf{h}_3^K \rangle \mathbf{h}_3^V$$

Query & Key & Value

- The first stage에서 만든 **32x50x512**를 (여기부턴 편의상 **1x3x512**로 설명할게요)

fc_q = nn.Linear(512, 64) 여기도

fc_k = nn.Linear(512, 64) 여기도

fc_v = nn.Linear(512, 64) 여기도

통과시켜 Q, K, V를 각각 따로 얻음 why? 🤔

(그림에서는 통과 시키기 전도 QKV로 표기되어 있음)

- Query** : 물어볼 기준 단어 벡터

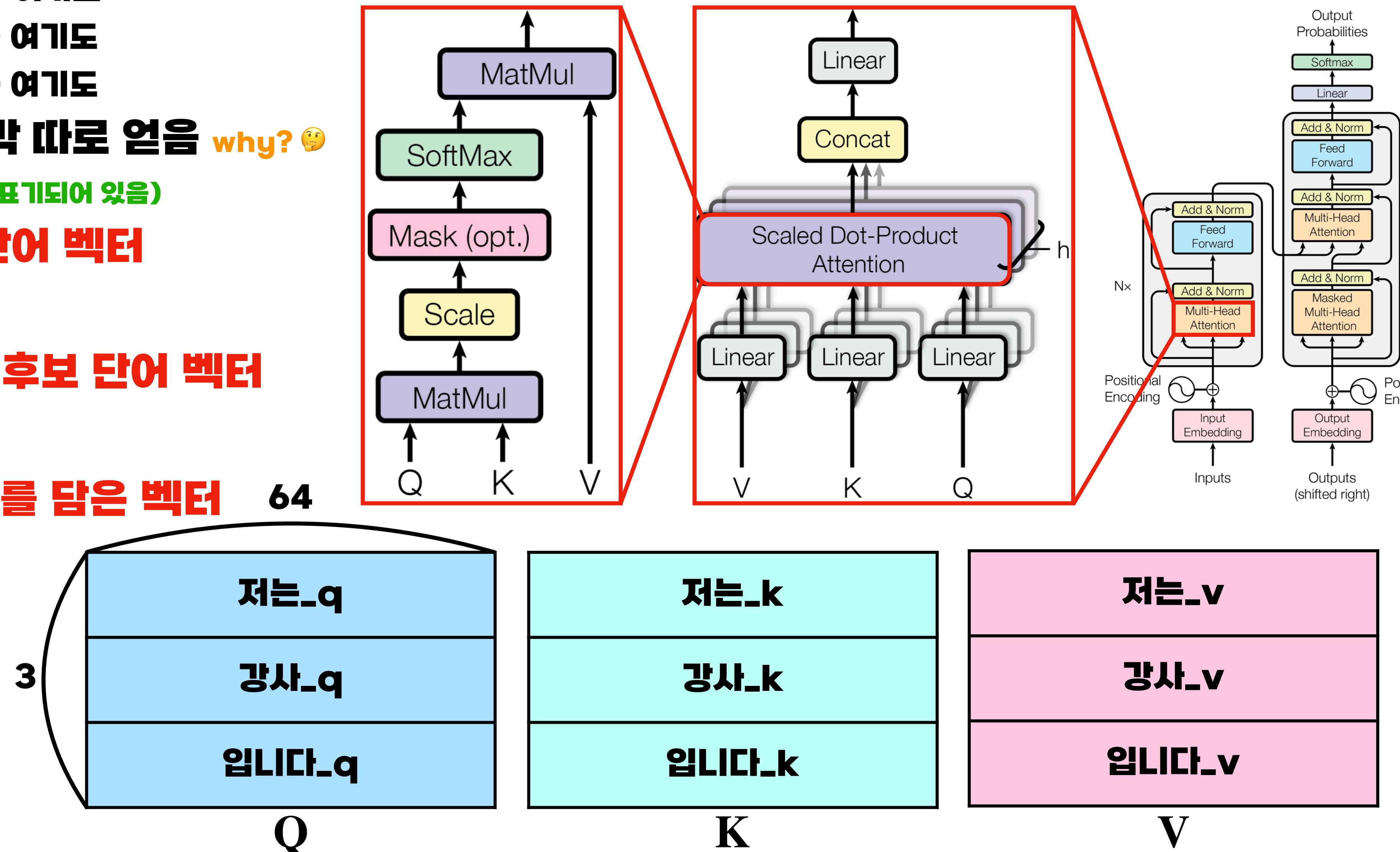
- 질문을 잘하자!

- Key** : Query에 답할 후보 단어 벡터

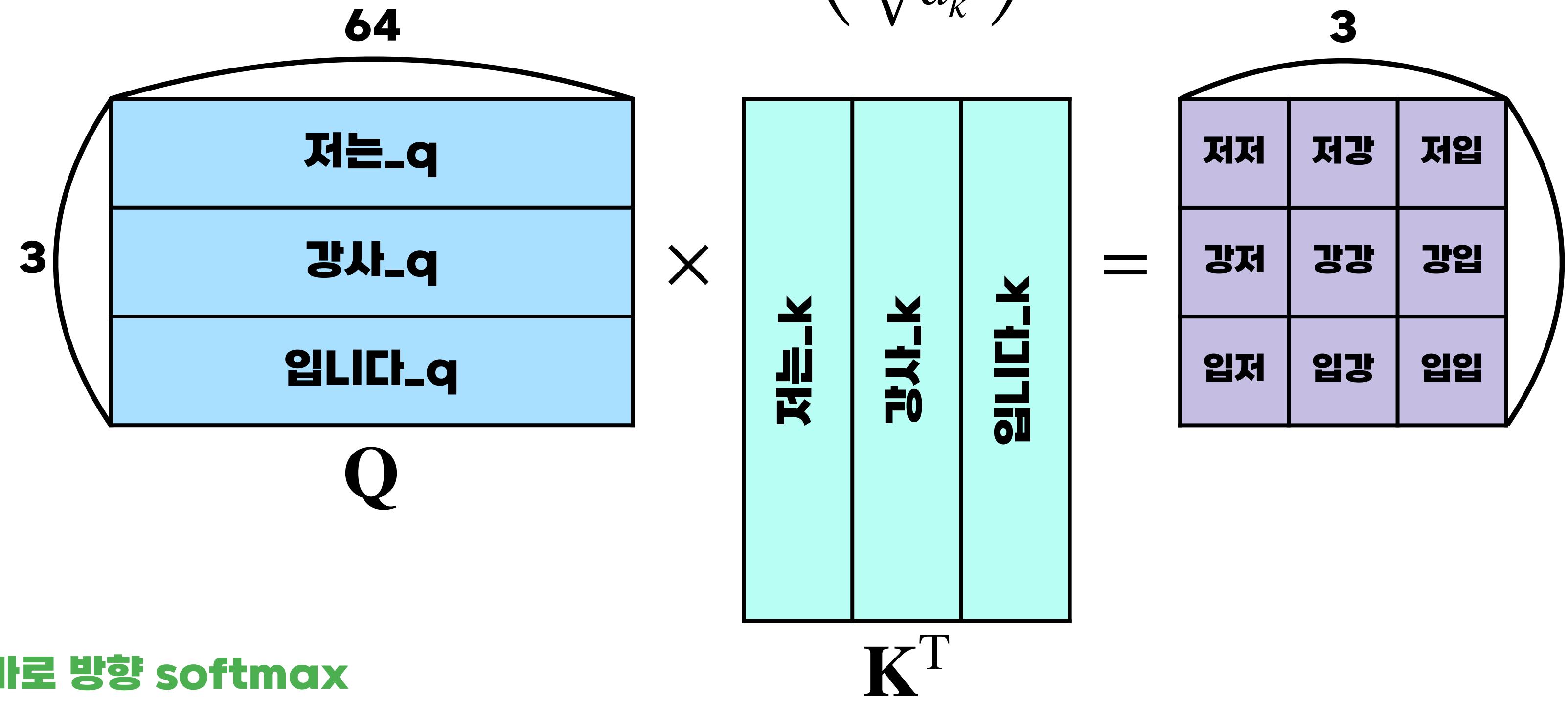
- 답변을 잘하자!

- Value** : 키 단어의 의미를 담은 벡터

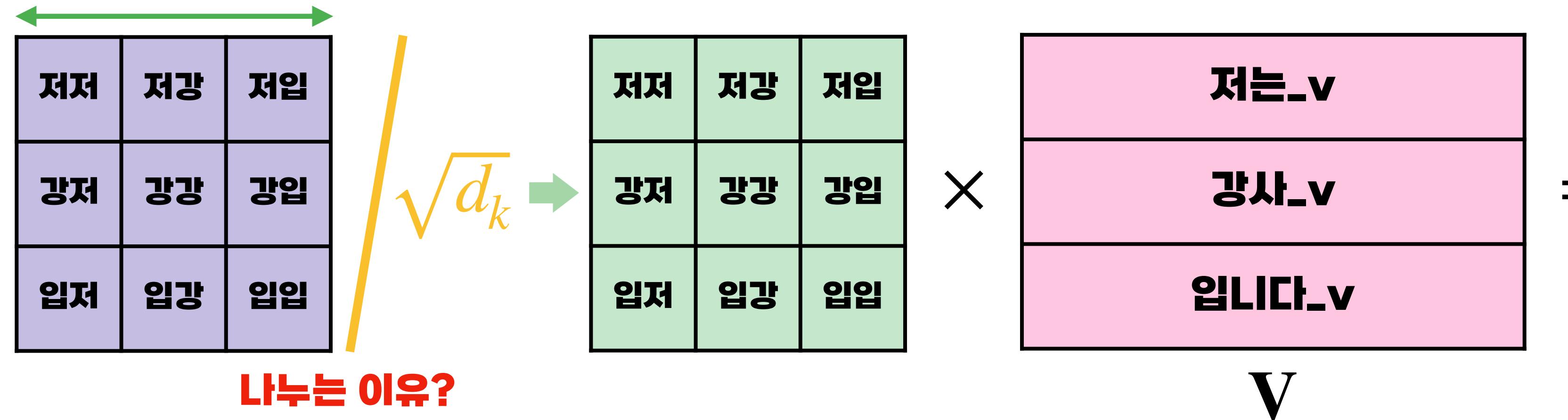
- 표현을 잘하자!



- $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V}$ 만 이해하면 끝! ($d_k = 64$)

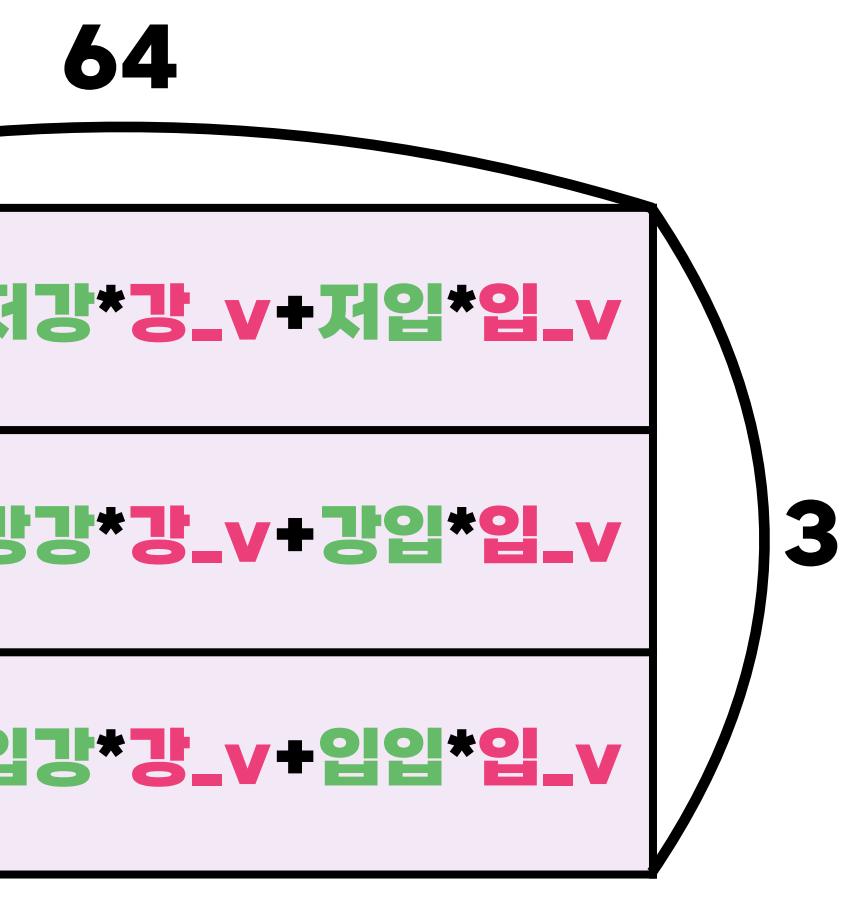
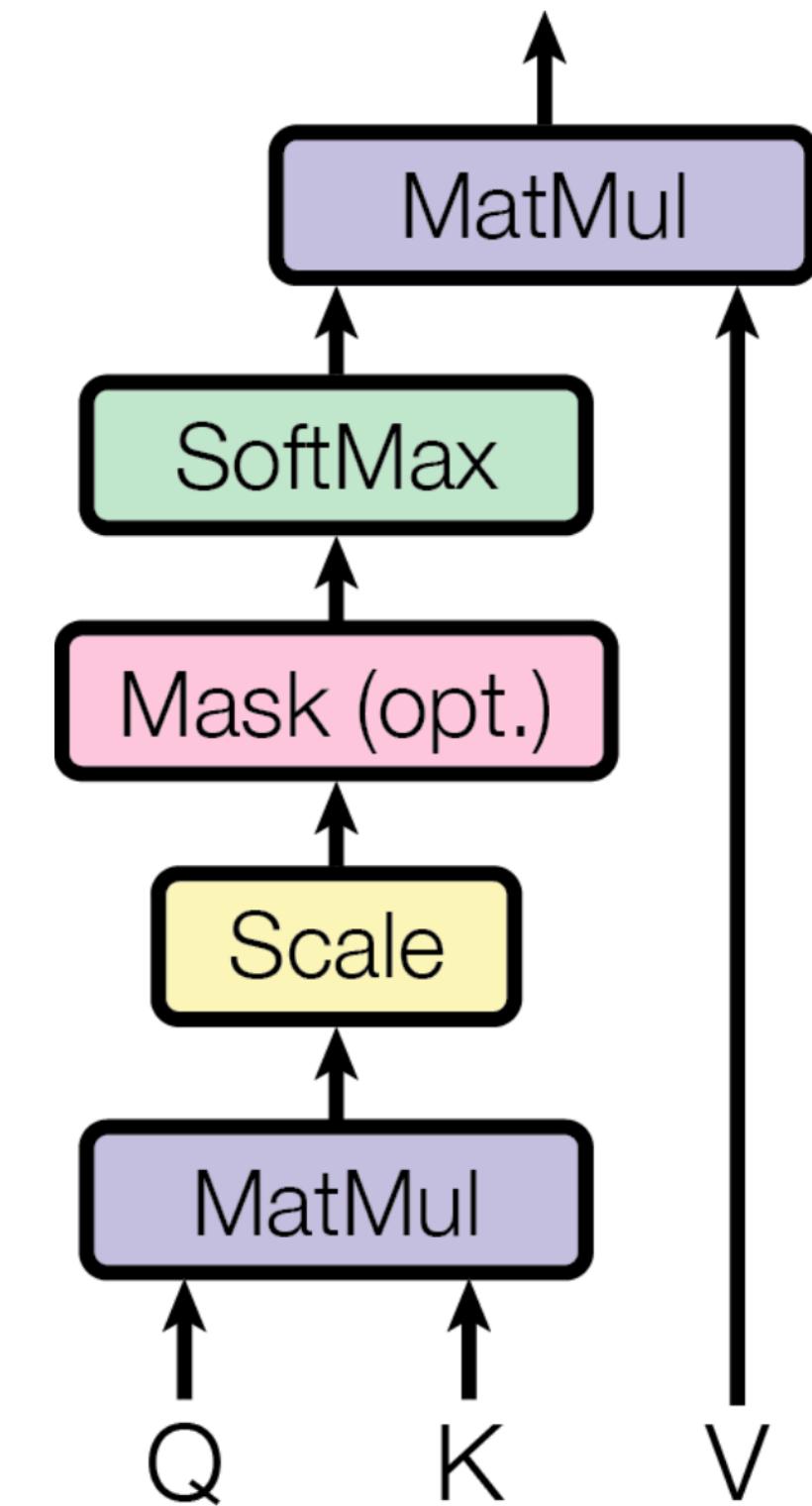


가로 방향 softmax



나누는 이유?

d_k 를 수록 내적의 분산이 자꾸 커져서 $\sqrt{d_k}$ 로 나눠 줌으로써 softmax 미분이 작아지는 것 방지



Multi-Head Attention

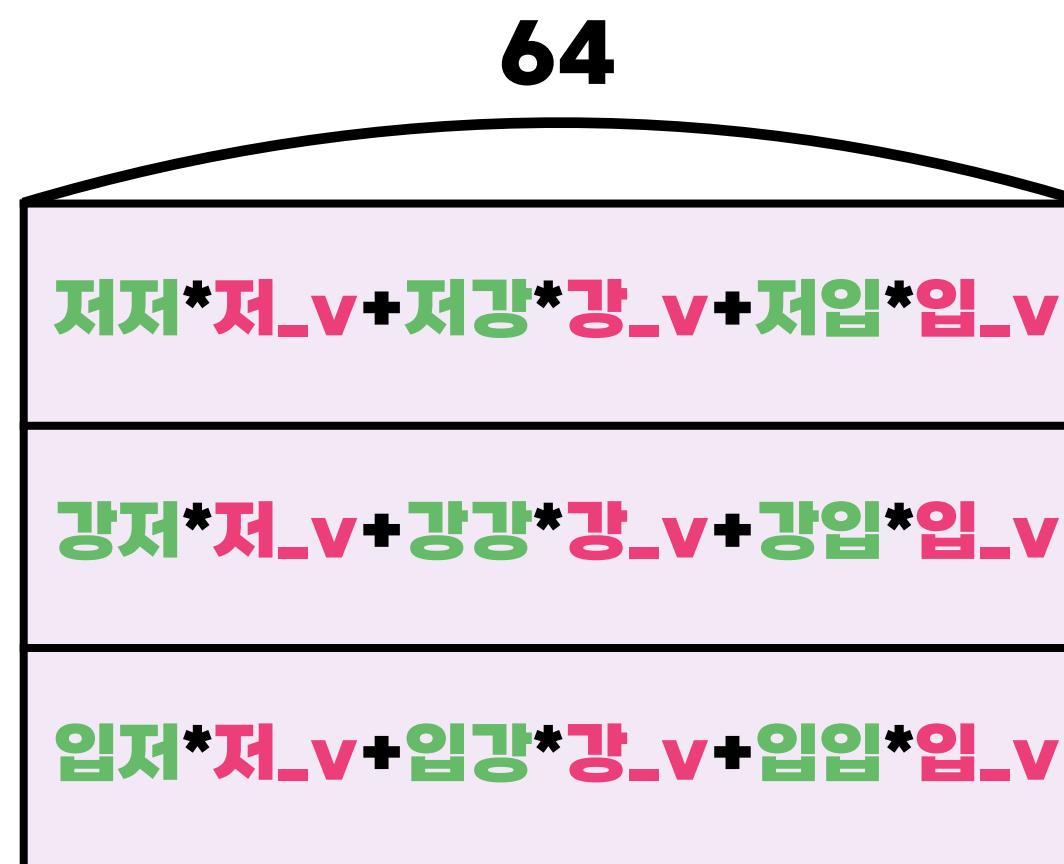
- 이 행위를

`fc_q2 = nn.Linear(512, 64), fc_q3 = nn.Linear(512, 64), ... fc_q8 = nn.Linear(512, 64)`
`fc_k2 = nn.Linear(512, 64), fc_k3 = nn.Linear(512, 64), ... fc_k8 = nn.Linear(512, 64)`
`fc_v2 = nn.Linear(512, 64), fc_v3 = nn.Linear(512, 64), ... fc_v8 = nn.Linear(512, 64)`

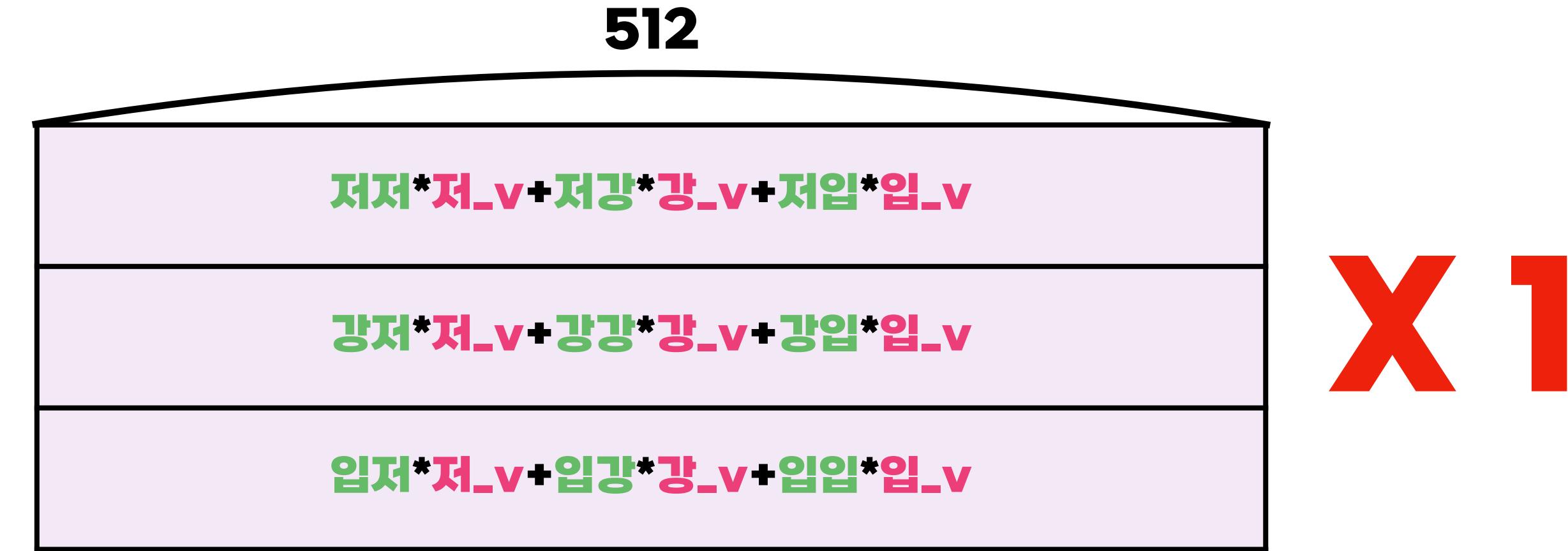
에 대해서도 똑같이 해주자!

- 이것이 바로 Multi-Head Attention!

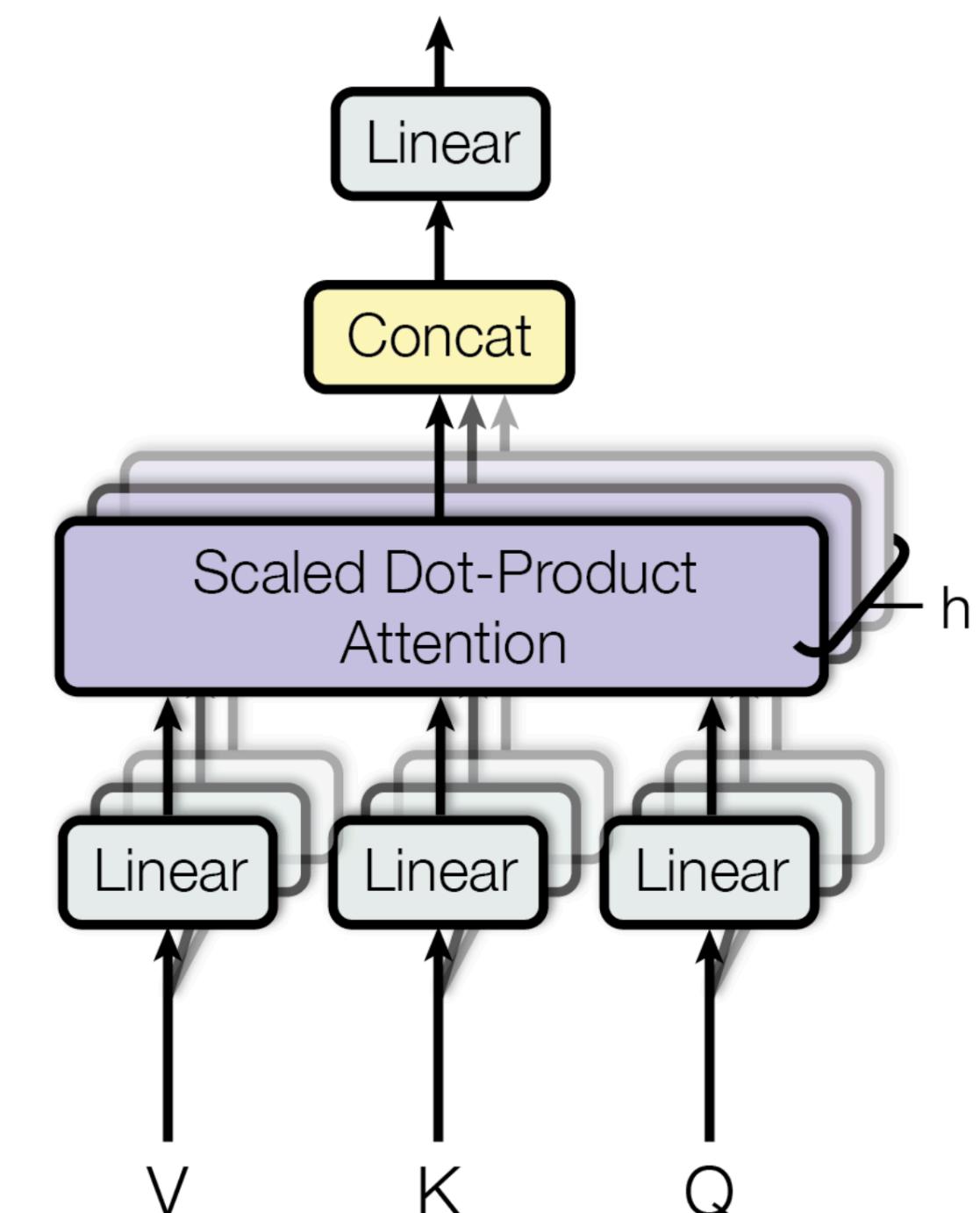
- 이렇게 해서 얻는 효과는 무엇일까요?



X 8 VS



- 이제 $1 \times 3 \times 64$ 짜리 여덟 개를 가로로 concat $\rightarrow 1 \times 3 \times 512$ 를 얻음
 \rightarrow 맨 위에 `nn.Linear(512, 512)` 통과 \rightarrow 도로 $1 \times 3 \times 512$ (들어온 입력과 같은 사이즈!)



Multi-Head Attention

- Head마다 다른 단어를 주목해서 본다..

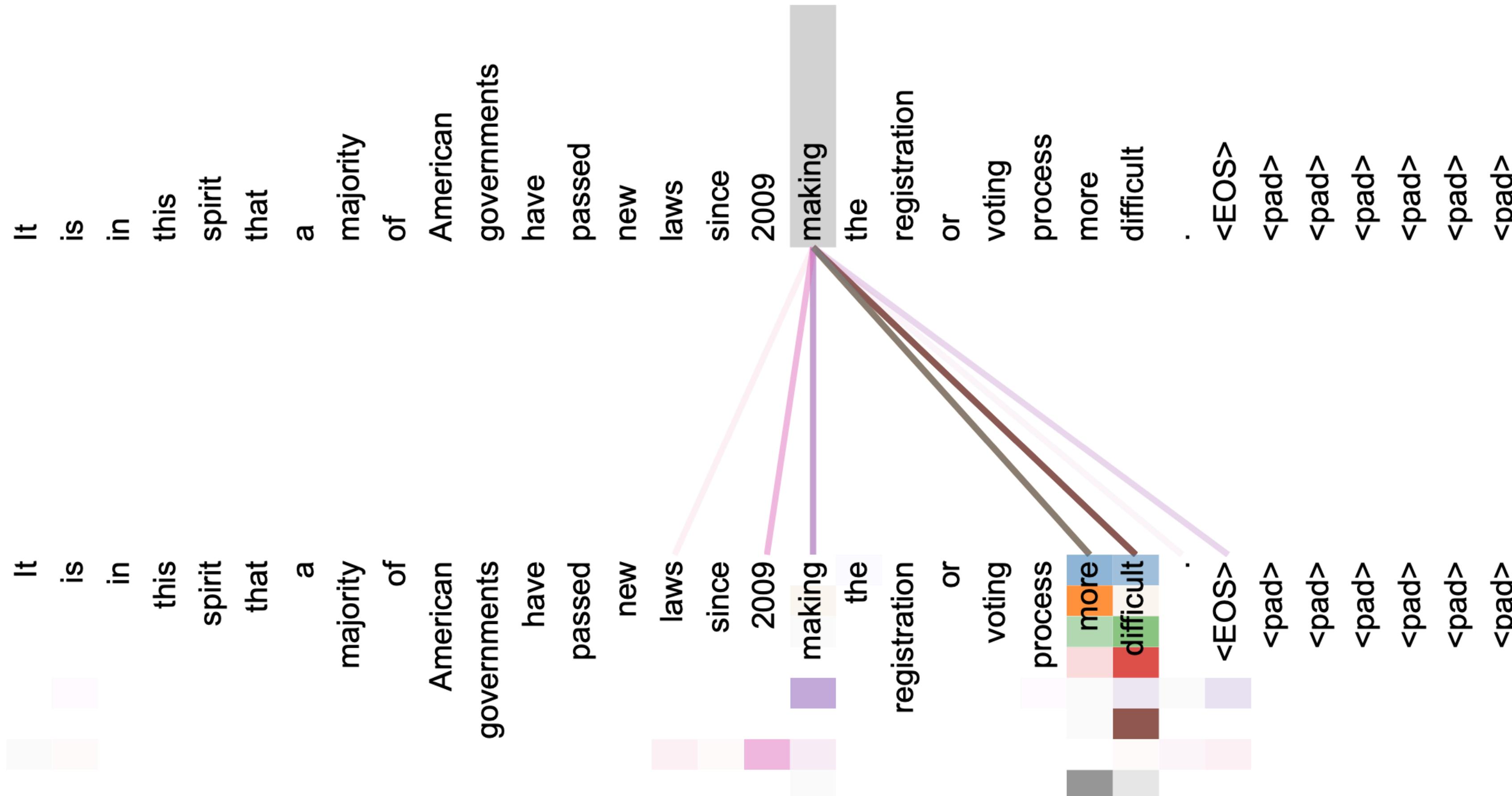


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention **in layer 5 of 6**. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.

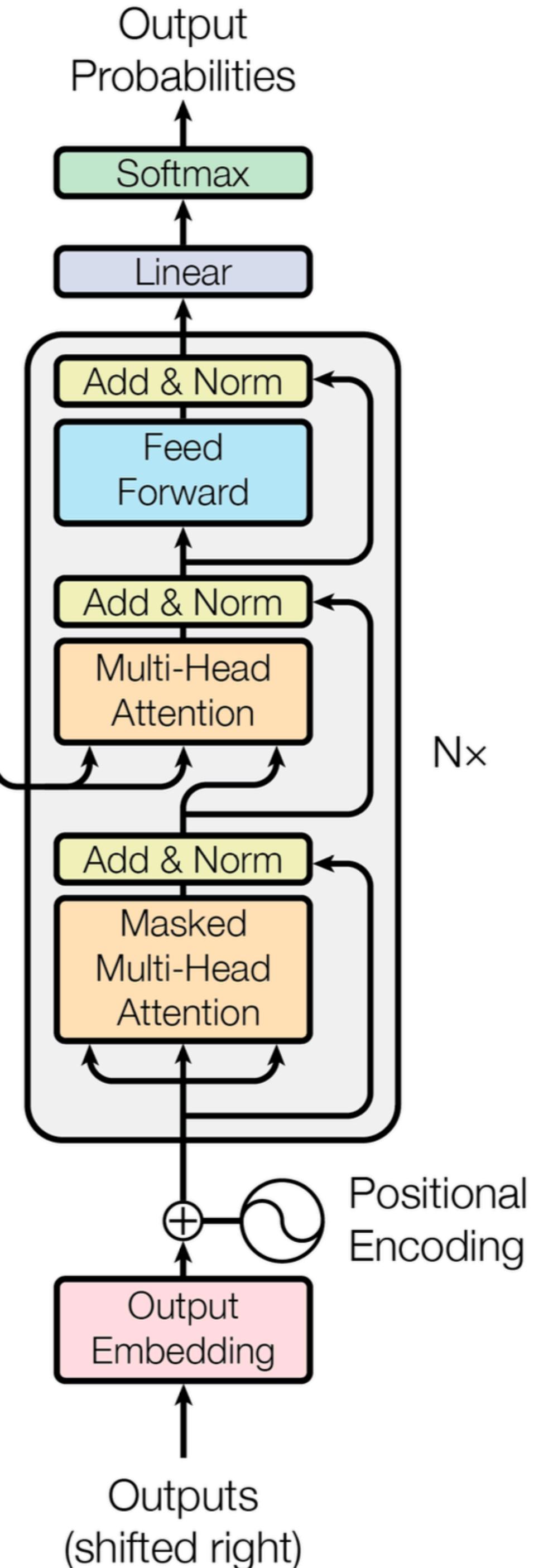
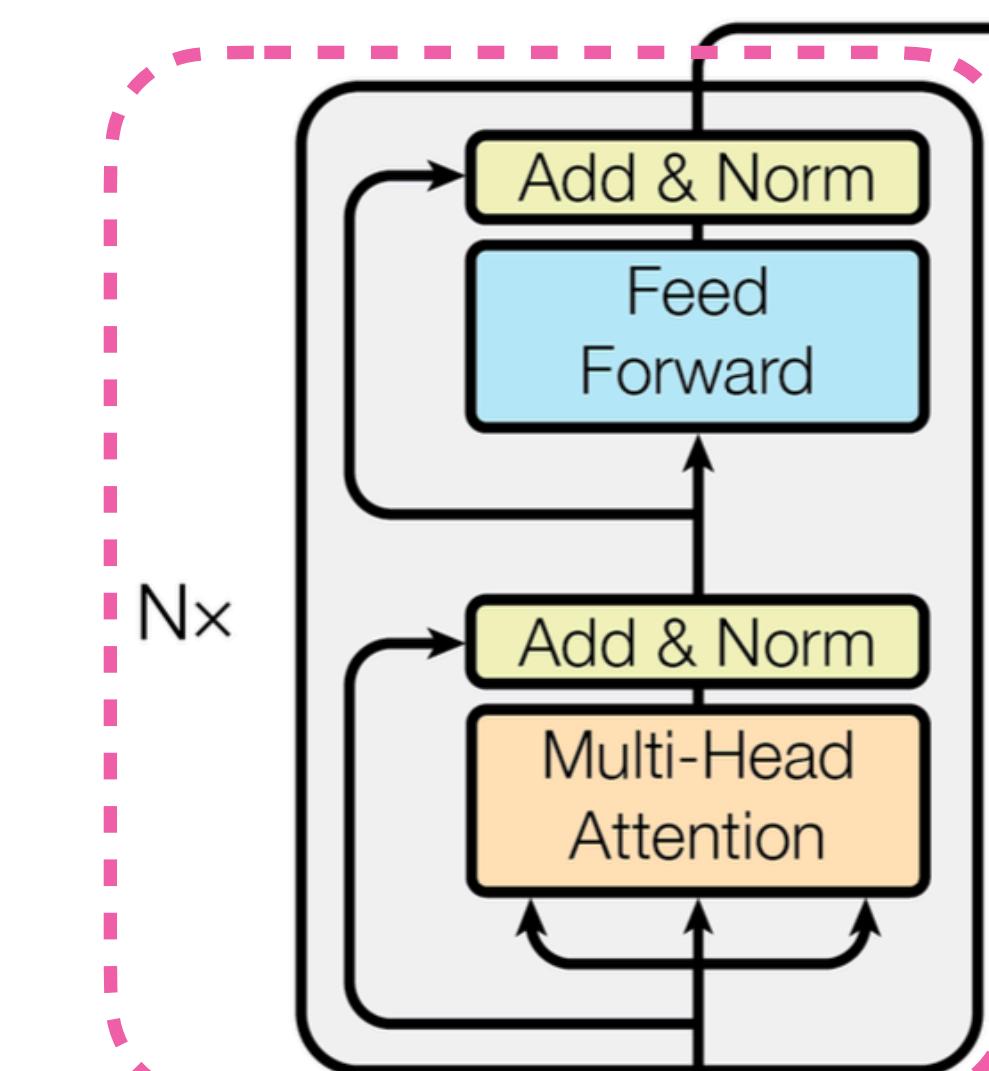
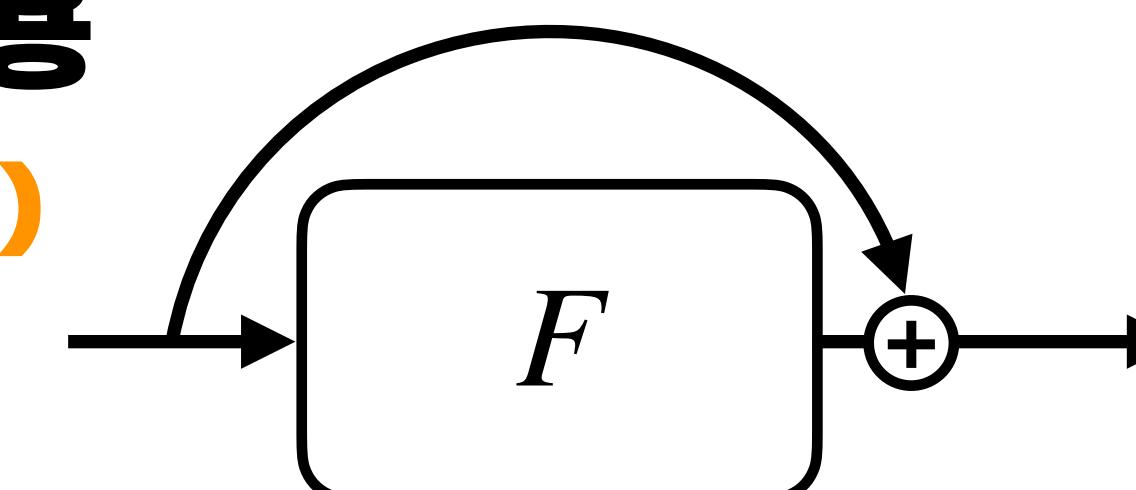
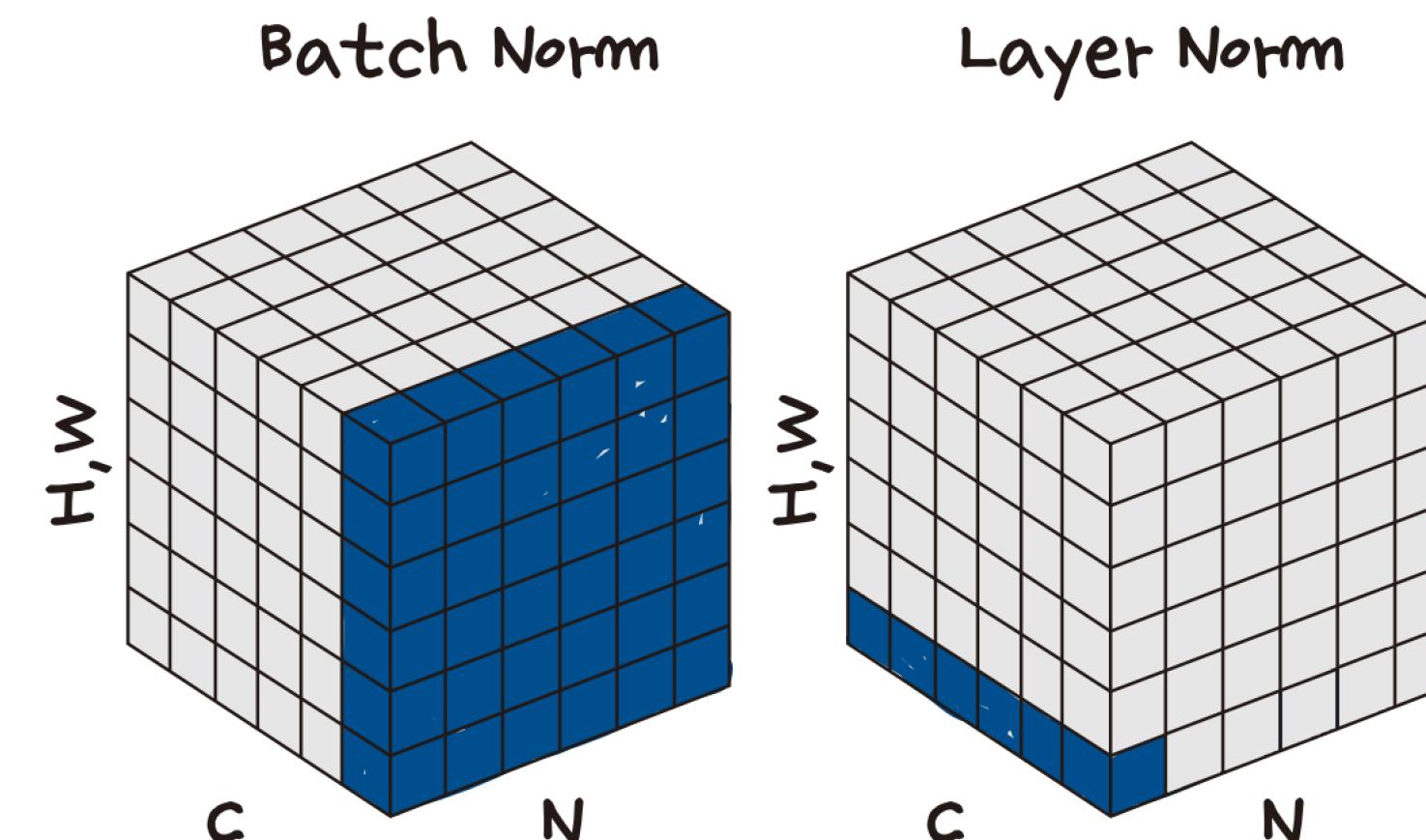
Encoder 전체 구조

- # • ResNet 의 skip-connection 사용

(잔차 학습: 차이 벡터를 만들어 톡톡! 보태기 개념)

- # • Layer Normalization 사용

- 즉, 512개 샘플에 대해 normalization!

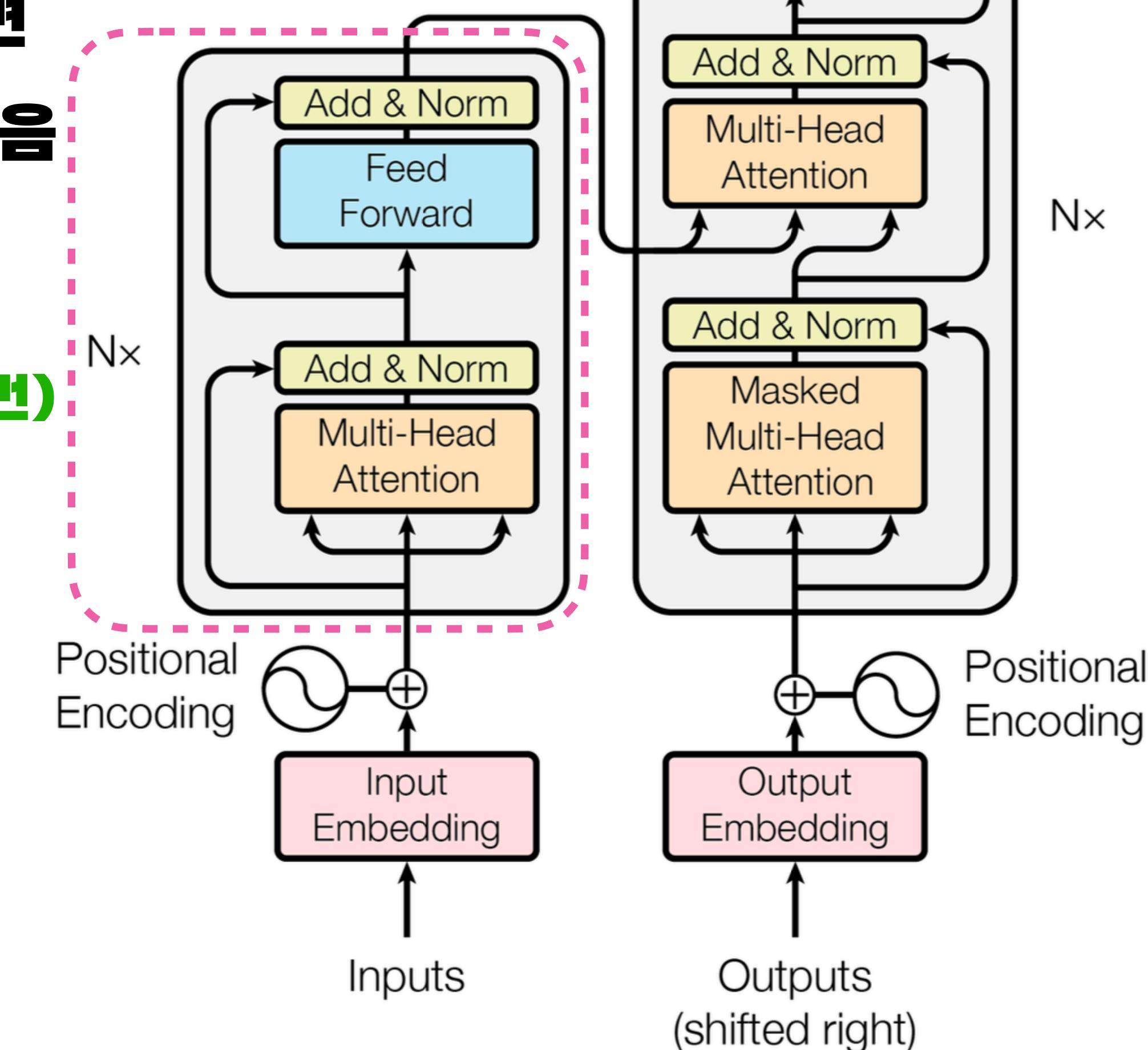
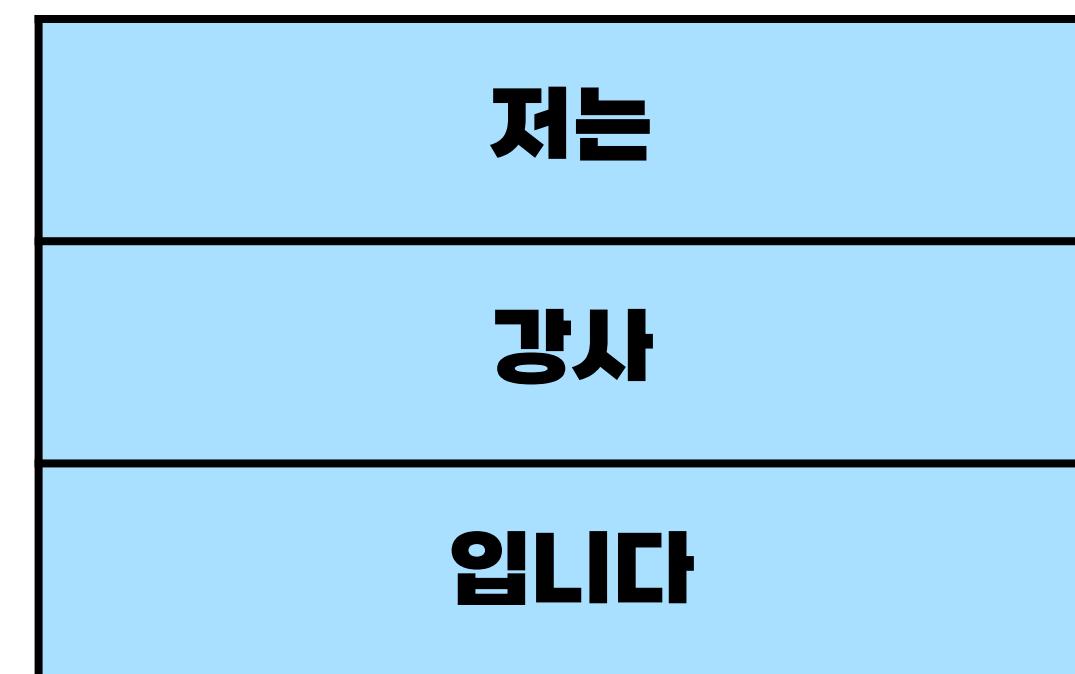


Encoder 구조 분석

- 어텐션은 가로세로 둘 다, MLP는 가로로만 연산 들어감

- 그러면 왜 굳이 MLP도 썼을까요?

- 1. 두 블록 각각의 목적(역할)을 생각해 봅시다
- 2. Universal Approximation Theorem 측면
- MLP를 안 썼더니 성능이 안 좋아지더라는 연구가 있음
(<https://arxiv.org/pdf/2103.03404>)
(👉 근데 놀랍게도 skip-connection 없앴을 때가 가장 최악)
- 오히려 MLP만 쓰는 모델도 있음 (세로 한 번, 가로 한 번)
(MLP-Mixer: <https://arxiv.org/pdf/2105.01601>)

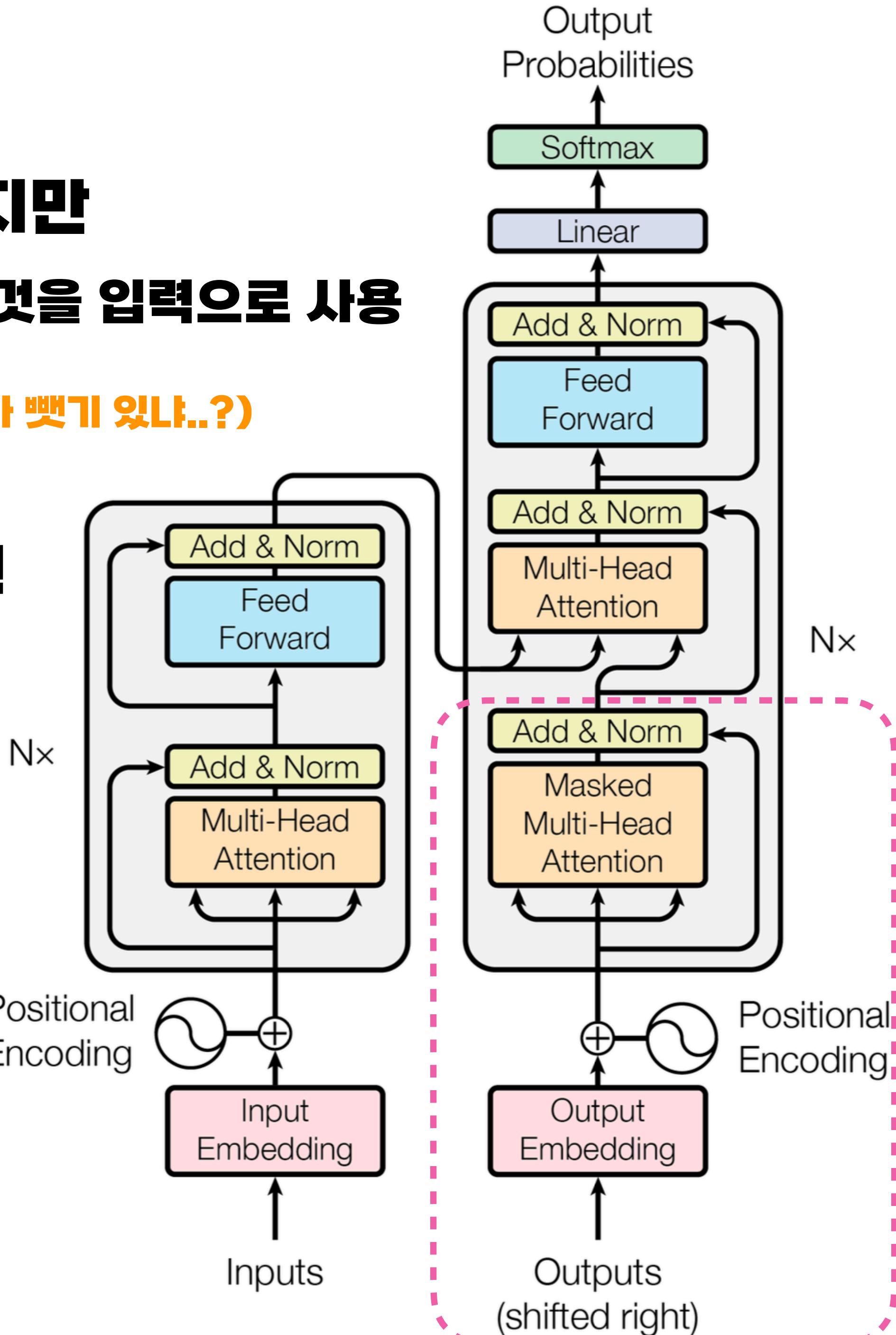
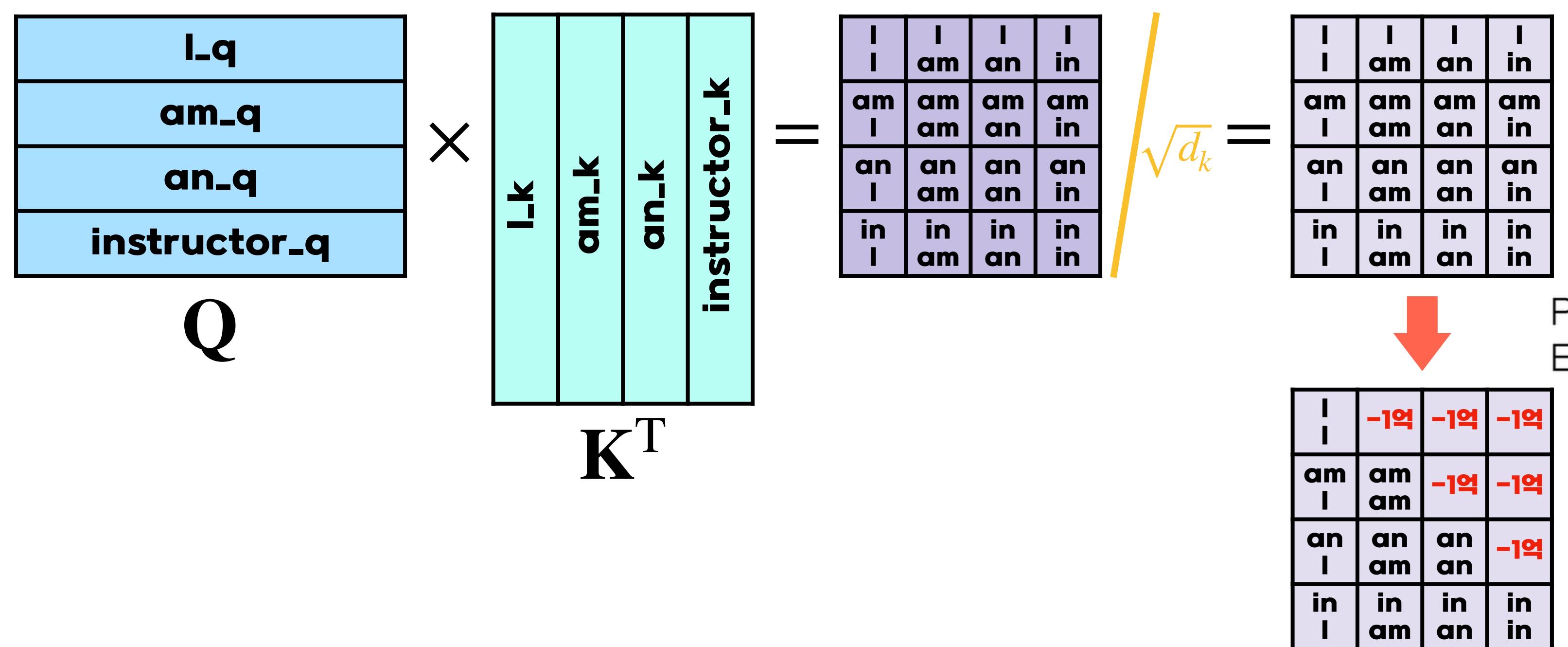


Masked MHA?

- 일단 입력 들어가는 방식은 Encoder 와 동일하지만

학습 시엔 **teacher forcing**(지도 학습), test 땐 출력 나온 것을 입력으로 사용
(seq2seq 처럼)

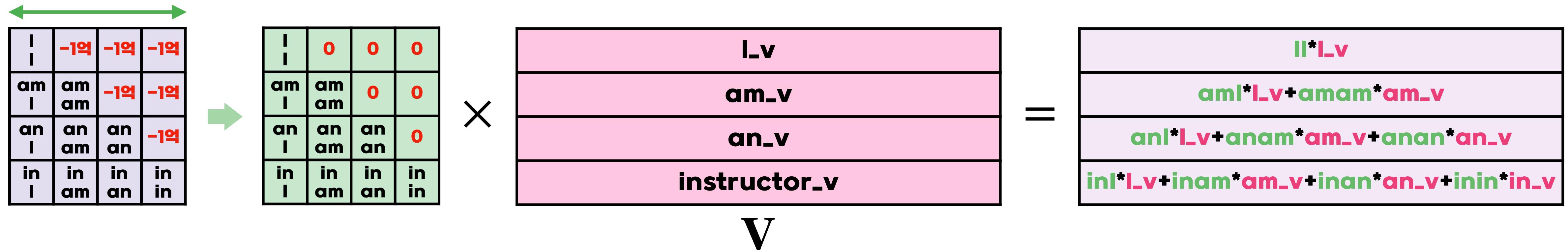
- MHA에 정답 문장을 집어넣고 그냥 attention 한다면..?
- 뒷 단어를 못 보게 (**런닝 못하게**) **Masking**이 필요하다!
 - softmax 통과 직전에 엄청 작은 음수로 바꿔치기!



Masked MHA?

- 현재 단어 포함 이전 단어들만 참조한다! (softmax 하고 나서 0으로 바꾸면 안 되는 이유는?)

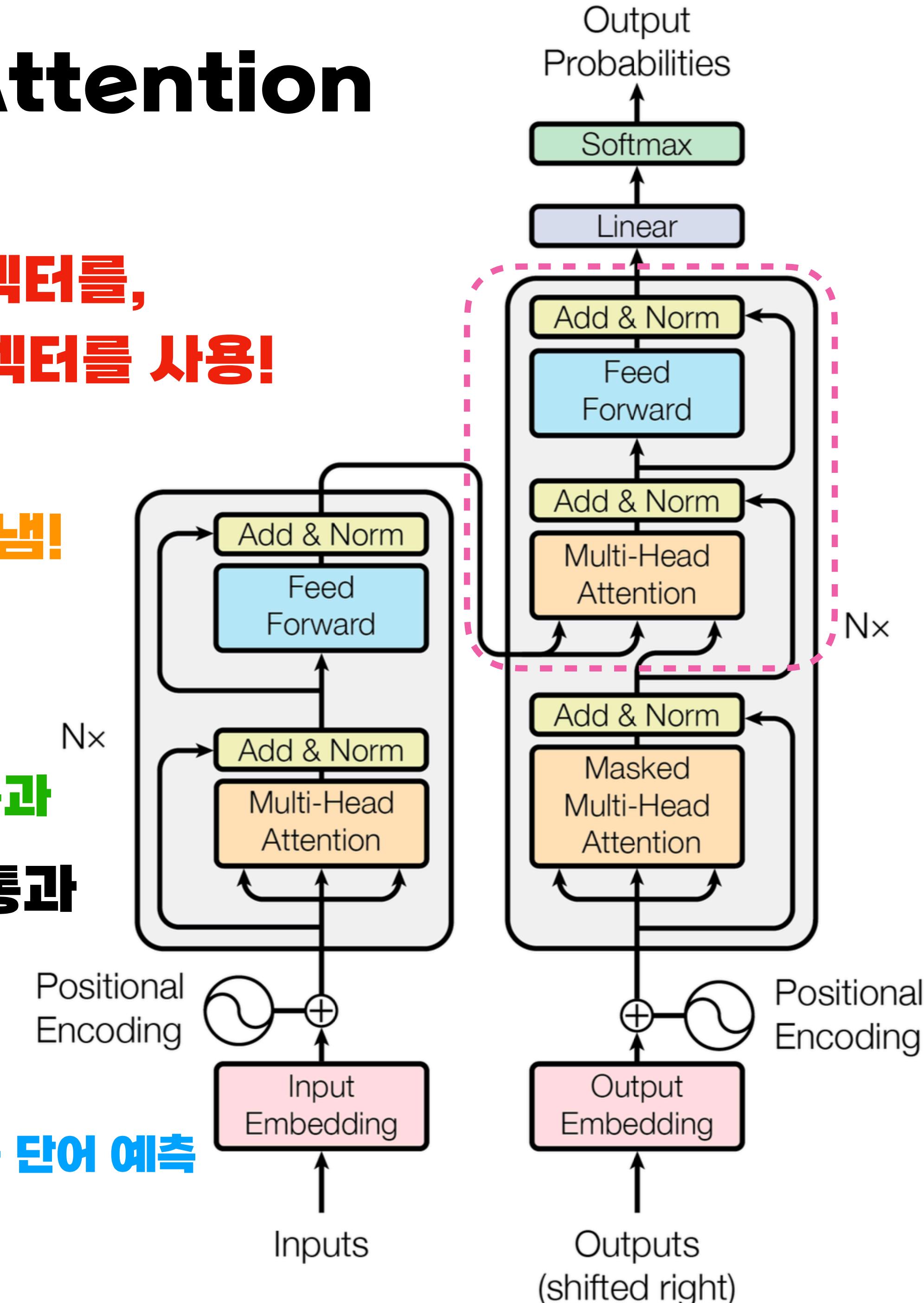
가로 방향 softmax



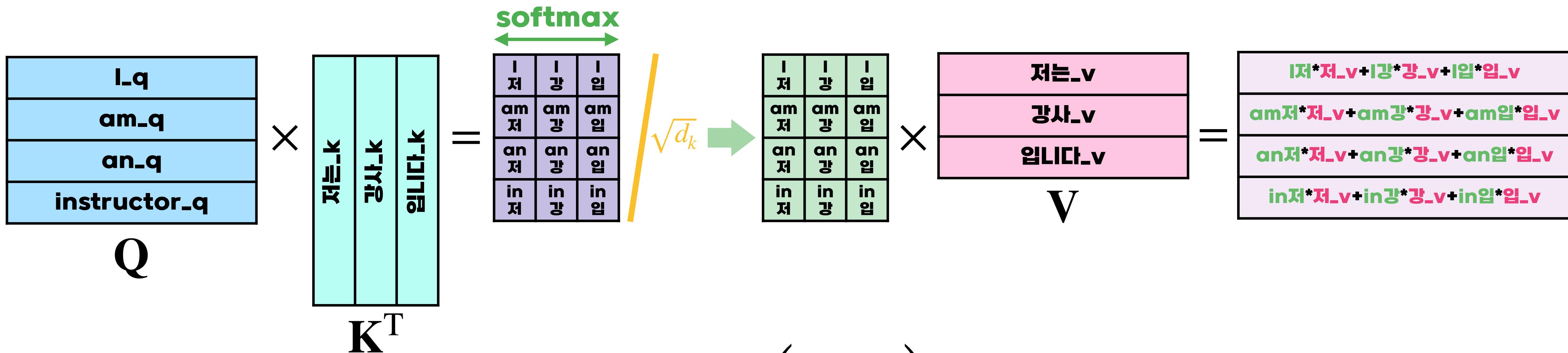
- 연습) am 입장에서 어떤 단어는 참조하고 어떤 단어는 참조하지 않나요?

Encoder-Decoder Attention

- 똑같은 MHA 모듈을 사용하는데
**Q로는 해당 Decoder layer에서 얻은 임베딩 벡터를,
KV로는 마지막 Encoder layer의 출력 임베딩 벡터를 사용!**
- 이렇게 하면 다음 단어가 뭐가 출력 돼야할지를
출력 문장의 Q로 물어보고 입력 문장의 KV를 보고 알아냄!
- 근데 이 Q는 MHA를 통과한 놈이니까
출력 문장의 각 단어에 대한 의미를 잘 담은 Q임!
 - 그렇게 context 벡터를 완성하고 [Add & Norm] 통과
 - 그리고 나서 여기도 같은 구조의 Feed Forward 통과
 - 즉, 아래 순서로 통과
self_atten → enc_dec_attn → FF
 - 먼저 디코더 문장 파악 → 입력 문장에서 뭘 주목할지 보고 → 다음 단어 예측
 - 이것이 Decoder Layer 하나이고 얘도 6번 반복!



Encoder-Decoder Attention 식으로도 확인!

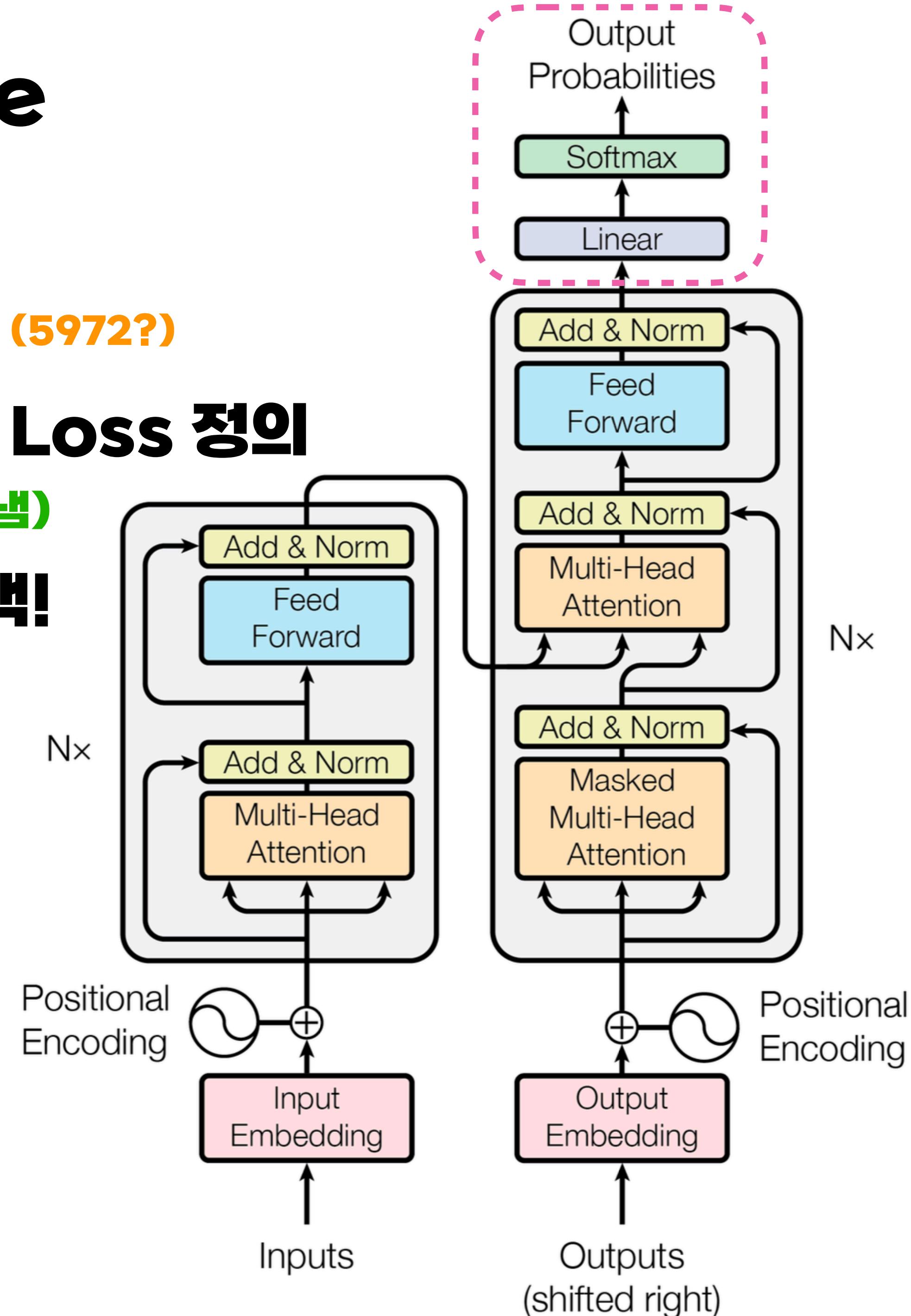


$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

정말 마법의 수식..!

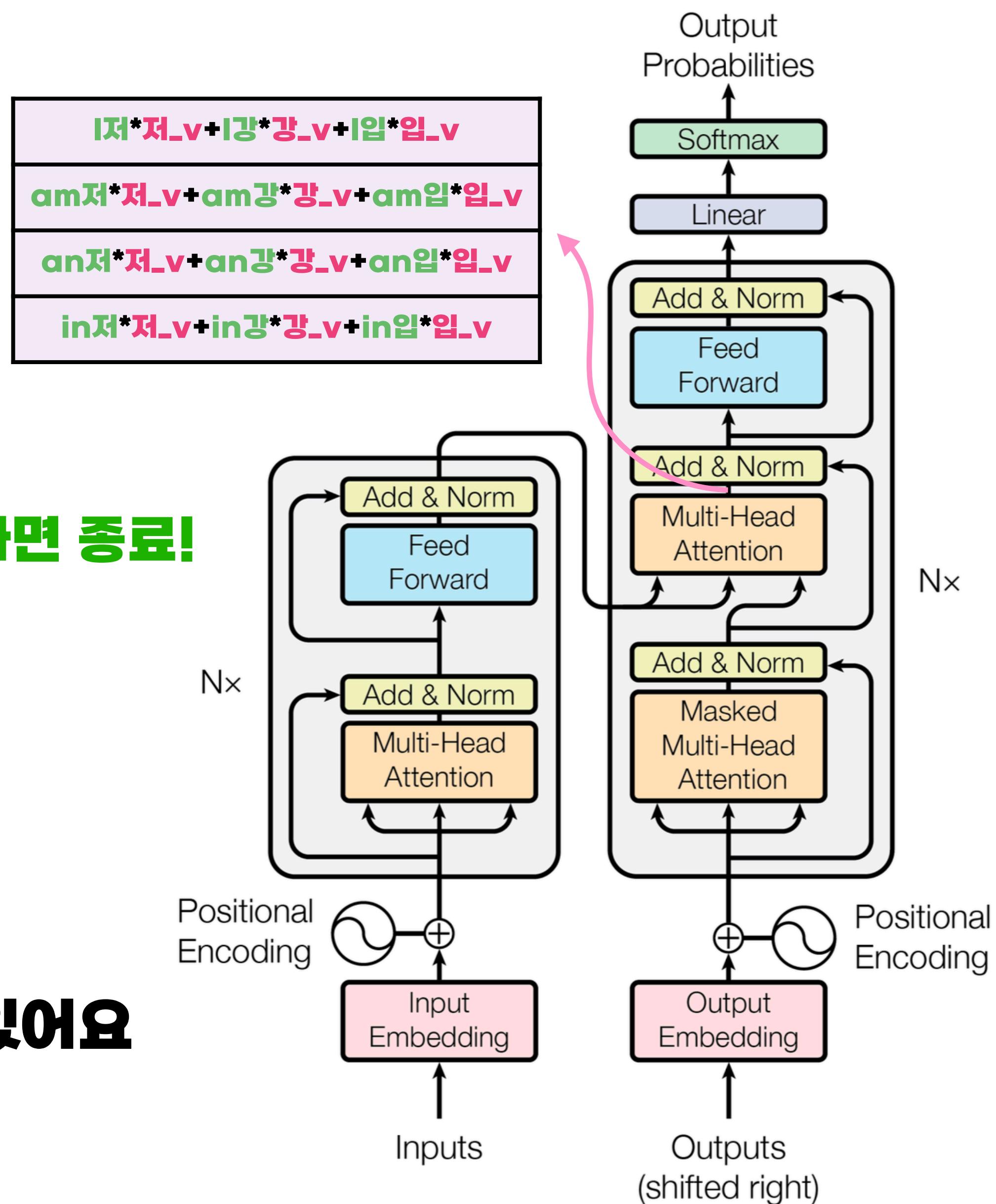
The last stage

- Decoder의 마지막 Layer의 출력을 사용
- 그냥 nn.Linear(512, 5972) 통과시키면 끝! (5972?)
- softmax를 통과 시키고 Cross-Entropy로 Loss 정의
(Hello->ello, Seq2seq 예시와 동일하게 전체 시점에 대해 평균냄)
- 추론 시에는 가장 높은 확률에 해당하는 단어 선택!



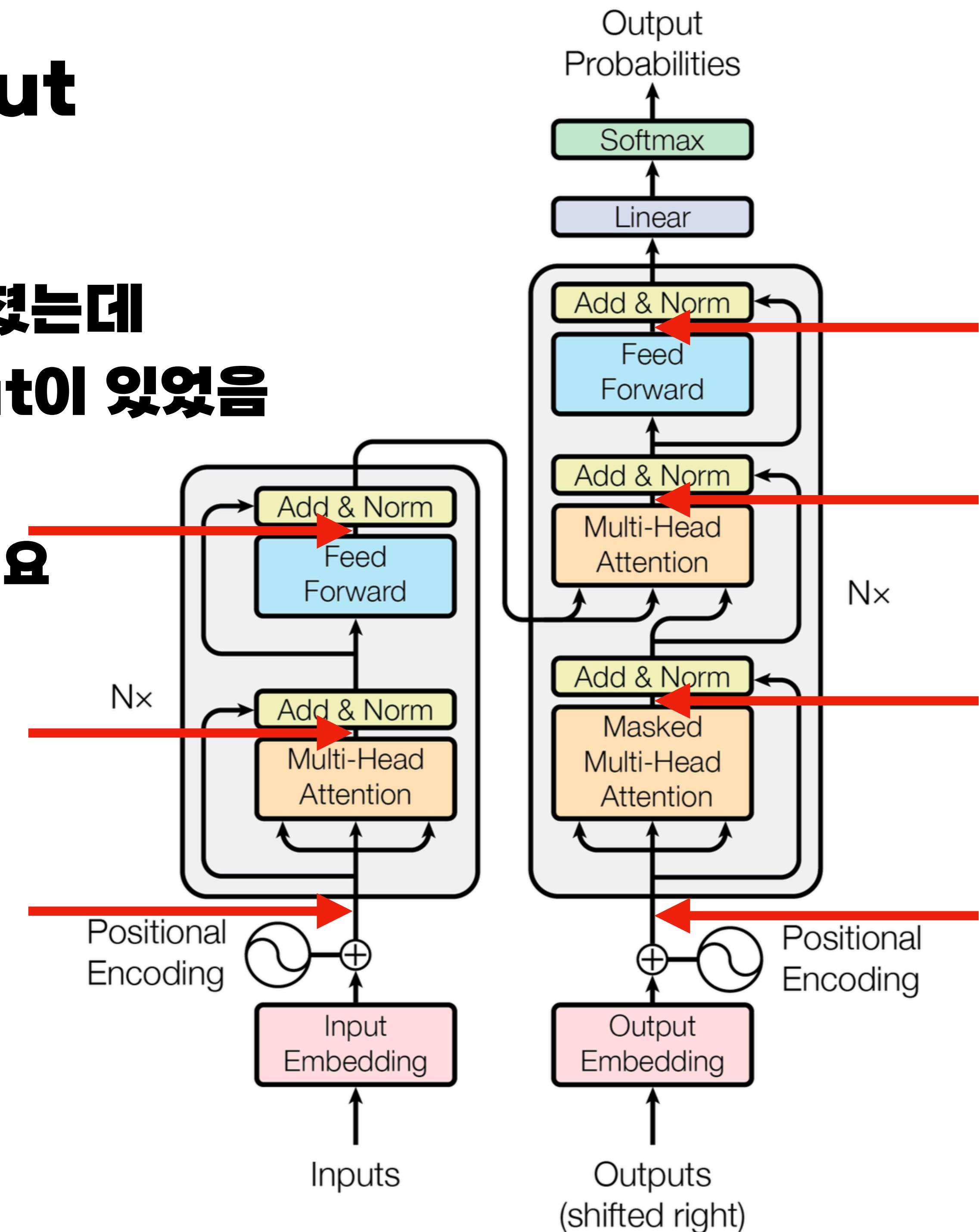
실제 추론(test 때)은 어떻게 이루어지나

- **추론할 때는**
 - 1. <sos> 하나 넣고 뽑고
 - 2. <sos> 두 단어 넣고 am 뽑고
 - 3. <sos> I am 세 단어 넣고 an 뽑고
 - 4. <sos> I am an 네 단어 넣고 instructor 뽑고
 - 5. <sos> I am an instructor 넣고 <eos> 나왔다면 종료!
- **Next Token Prediction!**
- **주의!) 추론 시에도 똑같이 masking을 합니다**
- **미래 단어를 참조하지 않으므로**
- **4. 에서 이전 단어 번역 결과가 달라지진 않아요**
- **이렇게 하기도 하지만 beam search 같은 것도 있어요**

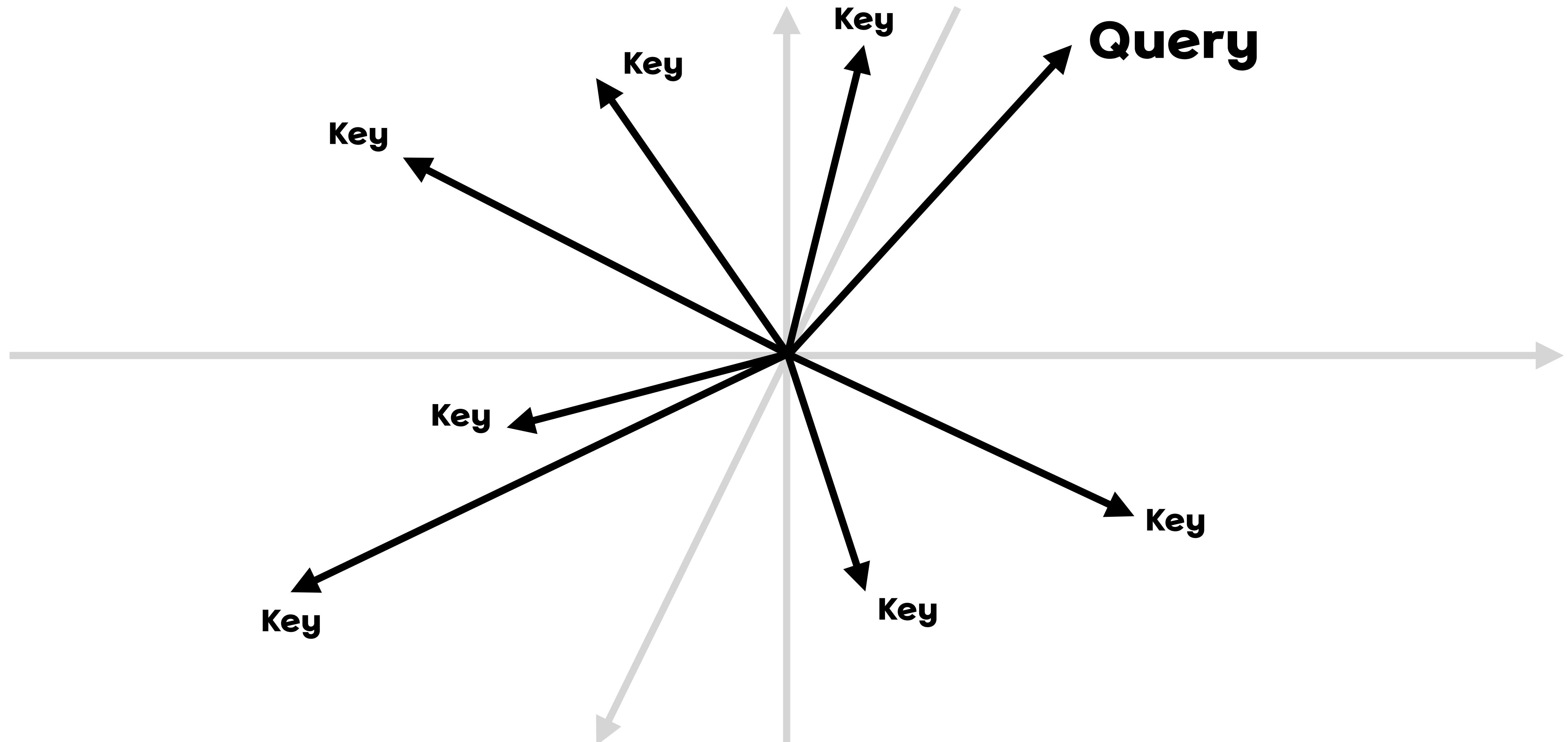


Dropout

- 총 7군데, $P_{\text{drop}} = 0.1$ 으로 했다고 함
- 논문이 arXiv 에 v1~v5 까지 수정을 거쳤는데
v4까지는 Attention 안에도 dropout이 있었음
(attention dropout)
- 구현할 때는 Feed Forward 에도 추가할게요
(overfitting 방지)

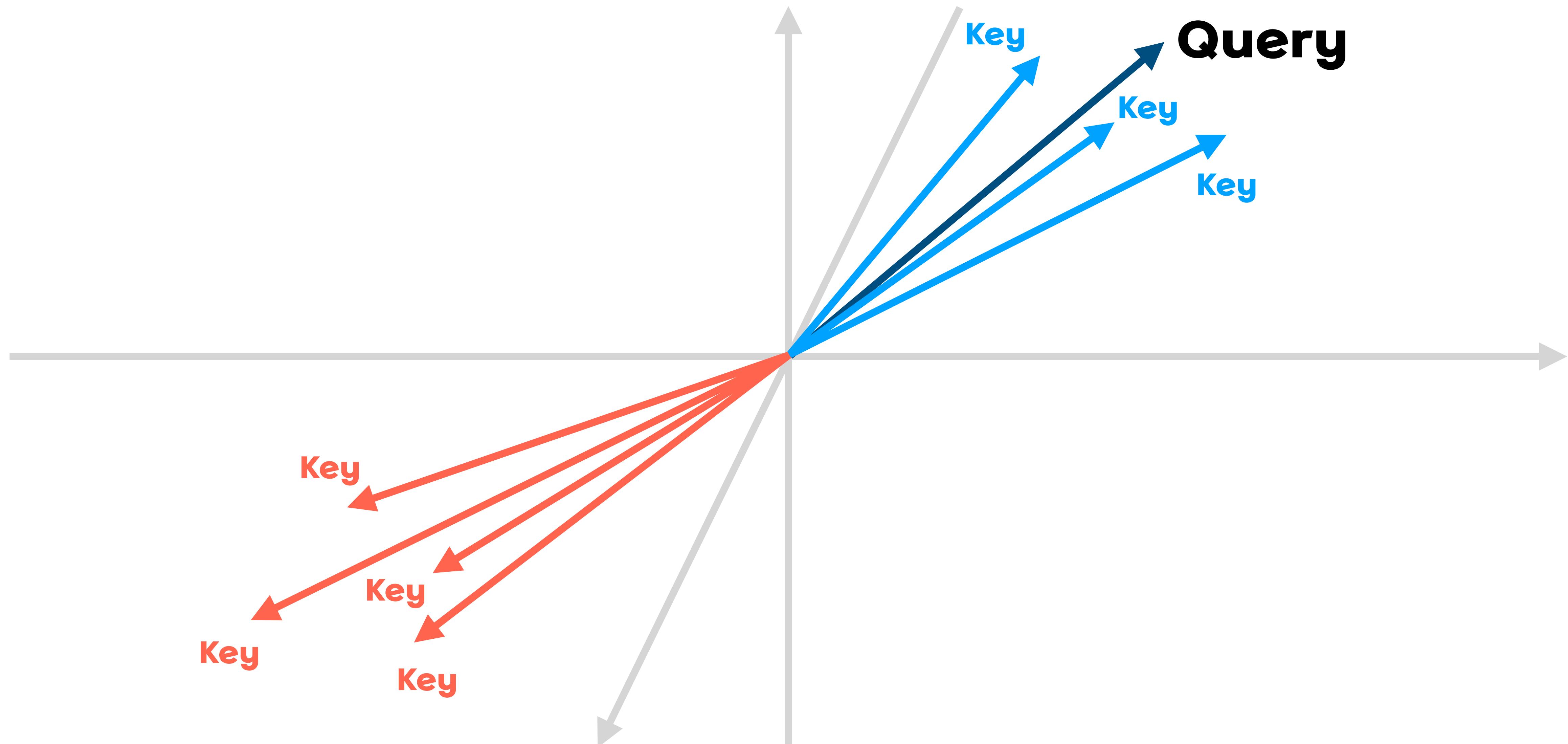


결국 임베딩 벡터를 어디로 이동시킬지를 학습하는 것!



결국 임베딩 벡터를 어디로 이동시킬지를 학습하는 것!

- 자연스럽게 어떤 단어를 “**주목**”해야 번역을 잘 할지를 알아서 알아낸다



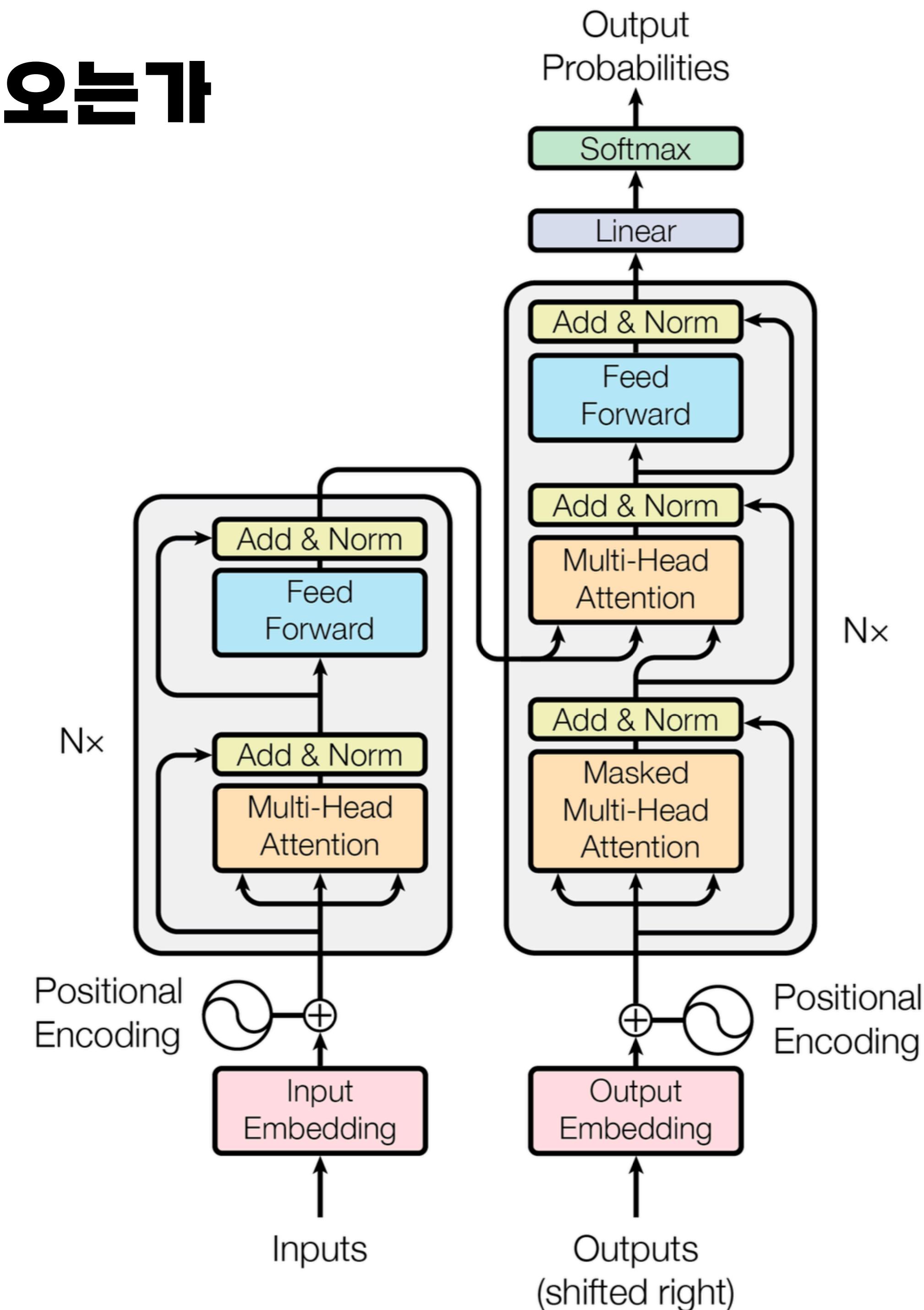
결국 임베딩 벡터를 어디로 이동시킬지를 학습하는 것!

- Skip-connection이 있으니, 각 block은 차이 벡터만을 학습한다!
(살짝살짝 톡톡톡! 치면서 이동시킴)
- “쓰다”라는 단어가 input embedding을 통과할 땐 주변 단어는 전혀 참고하지 않은 채로 위치하게 되는데 self-attn하면서 주변 단어를 봄하면서 있어야 할 위치로 이동된다 (톡톡톡하면서!)
- 즉, 돈을 쓰는 건지 모자를 쓰는 건지에 따라 그 의미에 맞는 위치로 가게 된다!
- 따라서, 똑같은 “쓰다”라는 단어도 최종 임베딩 벡터의 위치가 문맥에 따라 다름!!
(파라미터는 고정임에도!) Key



비선형성은 어디에서 오는가

- Feed Forward 말고는 ReLU가 없음!
- 비선형성이 너무 적은거 아닌가..?
- MHA가 이미 비선형적임! why? 🤔
- $Q = XW_Q \& K = XW_K \& V = XW_V$
- 따라서, $QK^T = XW_Q W_K^T X^T$ 만 봐도..
- 물론 softmax에서도 비선형성이 있고~



Evaluation

- 일단, 이미지 분류에서처럼 번역에서도 accuracy로 평가한다면??
 - “우리는 식당에서 맛있는 음식을 먹었다.” 가 정답 문장일 때
 - 머신이 “식당에서 맛있는 음식을 먹었다.” 라고 추론했다면?
- 정확도 0 %
- 따라서, 번역 문제에 있어서는 다른 evaluation metric (평가 지표)들을 사용
- PPL (Perplexity) 과 BLEU score를 배워봅시다.

Evaluation – Perplexity (PPL) (헷갈려하는 정도..?)

- 정의: N 개 토큰으로 이뤄진 문장에 대한 PPL은

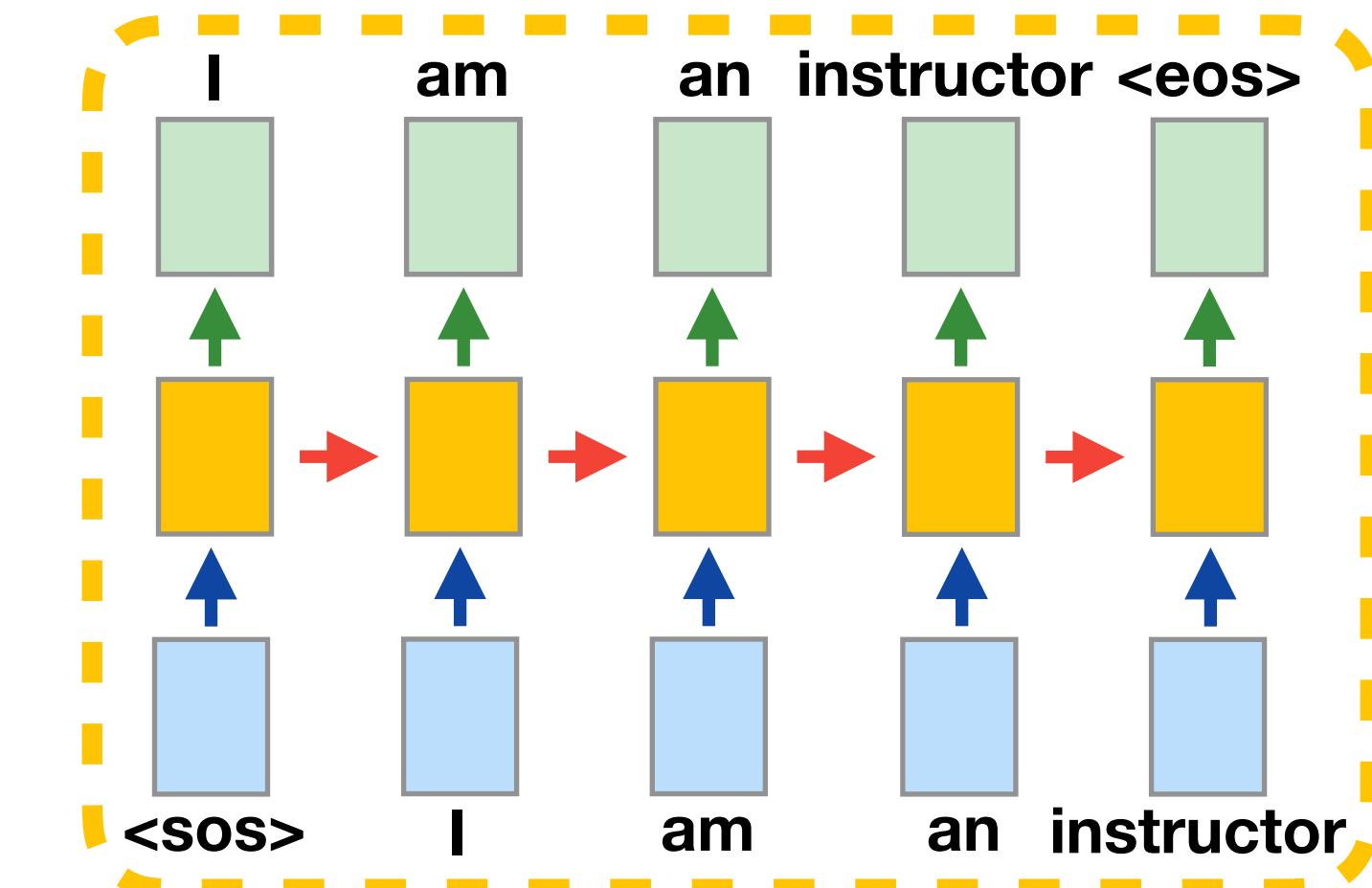
정답 문장에 대한 확률 $^{-\frac{1}{N}} = P(w_1, w_2, w_3, \dots, w_N)^{-\frac{1}{N}} = \left(P(w_1) \prod_{i=2}^N P(w_i | w_1, w_2, \dots, w_{i-1}) \right)^{-\frac{1}{N}}$ (기하 평균의 역수. 작을수록 좋다)

- 예를 들어, 100단어 중에 골라야 하는데 $N = 3, P = 1/100$ (이건 그냥 100개 중에 찍은 것) 이면 PPL=100 이 나옴
=> 이건 평균적으로 100 가지 중에 뭘 골라야할지 “헷갈려서” 고민했다는 뜻!!
- 한편, 우리의 cross-entropy를 생각해 보면.. (Teacher forcing 가정)

$$L = \frac{1}{N} \sum_i \text{Cross-Entropy} = \frac{1}{N} \left(-\log P(w_1) + \sum_{i=2}^N -\log P(w_i | w_1, w_2, \dots, w_{i-1}) \right) = \log \left(P(w_1) \prod_{i=1}^N P(w_i | w_1, w_2, \dots, w_{i-1}) \right)^{-\frac{1}{N}}$$

- 디코더 구조 생각해보면, 이전 단어들을 통해 현재 나와야 할 정답 단어에 대한 확률을 loss에 사용 decoder
- 즉, 쉽게 말해서 그냥 loss에 exp 취하면 PPL 됨!
- 아래와 같이 n-gram을 적용해볼 수도! (예시는 3-gram PPL)

$$\left(\prod_{i=1}^N P(w_i | w_1, w_2, \dots, w_{i-1}) \right)^{-\frac{1}{N}} \cong \left(\prod_{i=1}^N P(w_i | w_{i-2}, w_{i-1}) \right)^{-\frac{1}{N}}$$



Evaluation - BLEU (Bilingual Evaluation Understudy) score

- 번역에 있어서는 PPL 보단 BLEU가 좀 더 신뢰성이 있다.

0과 1 사이 값 나옴 ↗

$$BLEU = BP \cdot \prod_{n=1}^N p_n^{w_n}$$

p_n : n-gram precision

w_n : weight, 합 = 1

BP : Brevity (짧음) penalty

- n-gram precision? 연속한 n개 단어가 정답 문장에 존재하는지 여부로 계산 (같은 위치일 필요 X)

- 정답: 훌륭한 강사와 훌륭한 수강생이 만나면 명강의가 탄생한다

- 예측: 훌륭한 강사와 훌륭한 수강생이 함께라면 명강의가 만들어진다

- 1-gram precision : $\frac{\text{정답 문장에 존재하는지 여부의 합계}}{\text{예측 문장의 1-gram 개수}} = \frac{5}{7}$ (해당 1-gram이 존재하면 +1하고 다음으로 가기)

- 2-gram precision : $\frac{\text{정답 문장에 존재하는지 여부의 합계}}{\text{예측 문장의 2-gram 개수}} = \frac{3}{6}$

- 3-gram precision : $\frac{\text{정답 문장에 존재하는지 여부의 합계}}{\text{예측 문장의 3-gram 개수}} = \frac{2}{5}$

- 4-gram precision : $\frac{\text{정답 문장에 존재하는지 여부의 합계}}{\text{예측 문장의 4-gram 개수}} = \frac{1}{4}$

- N 은 정해주는 것 (default가 4) $\Rightarrow \prod_{n=1}^N p_n^{w_n} = \left(\frac{1}{28}\right)^{\frac{1}{4}} = 0.4347$

Evaluation - BLEU (Bilingual Evaluation Understudy) score

- 왜 n-gram? 만약 Unigram 만 쓴다면..
 - 정답: 훌륭한 강사와 훌륭한 수강생이 만나면 명강의가 탄생한다
예측: 만들어진다 강사와 훌륭한 명강의가 훌륭한 수강생이 함께라면
 - 순서만 섞였다면 똑같은 정밀도?! 그래서 n-gram을 쓰는 것!
- 아직 보정할 게 남았습니다. 아래에서 Unigram precision은?
 - 정답: 훌륭한 강사와 훌륭한 수강생이 만나면 명강의가 탄생한다
예측: 훌륭한 강사와 훌륭한 수강생이 훌륭한 강사와 훌륭한 강의를 만든다
 - 7/9? 단어의 순서 자체는 나쁘지 않지만.. 이상한 문장인데 점수가 높다?!
👉 Clipping이 필요! (optional 한 게 아님)
 - 즉, 정답 문장에 해당 gram이 존재하는 개수까지만 늘려라! 보정 후에는 4/9!
 - 2, 3, 4-gram precision 계산할 때도 clipping 적용

Evaluation - BLEU (Bilingual Evaluation Understudy) score

- 이제 마지막 관문 BP: Brevity penalty, 짧음 패널티?
- 이렇게 번역했다면 좋은걸까?
 - 정답: 훌륭한 강사와 훌륭한 수강생이 만나면 명강의가 탄생한다
예측: 수강생이 만나면 명강의가 탄생한다
 - $\prod_{n=1}^N p_n^{w_n} = 1 \dots$
 - 너무 간결한 나머지 중요한 정보(저와 여러분 같이 훌륭한 수강생이 만나야 함)가 빠졌음에도..
 - 이를 막기 위해 $BP = \begin{cases} 1 & \text{if } c \geq r \\ e^{1-r/c} & \text{if } c < r \end{cases}$ 도입! 0.4724 으로 낮아진다..!
(r: reference(정답) 문장 길이, c: candidate(예측) 문장 길이)
 - 길게 번역하는 건 괜찮은데 (어차피 n-gram precision에서 점수 낮아짐)
너무 짧게 해서 정보가 날아가지는 못하게 하자는 것! (AI 너 말이 짧다..?)

One More Thing..

- 진짜 마지막 관문: 여러 문장에 대한 BLEU score를 계산할 때는..

$$\text{n-gram precision } p_n = \frac{\sum_{\text{sentence}} \text{정답 문장에 존재하는지 여부의 합계}}{\sum_{\text{sentence}} \text{예측 문장의 n-gram 개수}}, \quad BP = \begin{cases} 1 & \text{if } \Sigma c \geq \Sigma r \\ e^{1-\Sigma r/\Sigma c} & \text{if } \Sigma c < \Sigma r \end{cases}$$

- 아마도 짧은 문장에는 4-gram 0I 00I 나오고 해서 그런 들쭉날쭉함을 방지할 수 있지 않을까..
- 논문에서는 이렇게 구한 BLEU score에 100을 곱해서 0~100점으로 표기함
- 그럼 마지막 예시 풀어봅시다! ($N = 4$, equal weight 가정)
 - 정답 1: 훌륭한 강사와 훌륭한 수강생이 만나면 명강의가 탄생한다
예측 1: 훌륭한 강사와 훌륭한 수강생이 훌륭한 강의를 만든다
 - 정답 2: 이것은 두 번째 문장입니다
예측 2: 이것은 문장입니다
 - 정답은?? ↗

모델 변형해가며 영어 → 독일어 성능 비교

- 안 쓰여 있으면 base model의 값
- dev는 development set, validation set과 같은 개념
- (A): $h \times d_k = 512$ 는 그대로 두고 h 를 바꿔가며 실험

(head가 너무 많아지면 오히려 성능 떨어짐)

- (B): d_k 만 줄여봤을 땐 성능 감소
- (C): 파라미터 수 키우니 성능 향상
- (D): Dropout 덕에 overfit 방지

- (D): ϵ_{ls} 는 label smoothing 값
⇒ PPL 커지지만 BLEU 증가

- (E): positional encoding을
positional embedding으로,

즉, 학습파라미터로 놓았을 때

⇒ 별 차이 없음!

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)				2						6.11	23.7	36
				4						5.19	25.3	50
				8						4.88	25.5	80
				256		32				5.75	24.5	28
				1024		128				4.66	26.0	168
				1024		128				5.12	25.4	53
				4096		4.75				26.2	90	
				5.77		24.6						
(D)							0.0			4.95	25.5	
							0.2			4.67	25.3	
							0.0	5.47		25.7		
							0.2	4.92		25.7		
(E)	positional embedding instead of sinusoids									4.33	26.4	213
big	6	1024	4096	16			0.3		300K			

Transformer - Summary

- 한마디로 요약하면 **attention** 오지게 쓴 모델..!
- 학습되는 건 결국 **FC 레이어!** (선형 변화)
- **내적과 가중합이라는 단순한 연산을 적용했을 뿐인데 여러가지 해석이 가능**
 - 어떤 단어들과 함께 등장했는지를 살펴보고 워드 임베딩 벡터에 의미를 잘 담음! (**self-atten.**)
 - 어떤 단어를 집중해서 봐야 번역을 잘 할지를 학습했다 (**enc-dec atten.**)
- ‘**미분 가능성**’만 따지면 **또 다른 연산 적용해서 또 다른 해석이 가능할 것**
- 너무 잘 동작해서 이미지 처리 등, 여러가지 문제에도 트랜스포머가 적용됨
- 정말 **attention! all we need**인가? 아직까진 깨지지 않고 있다..

 리뷰 이벤트 

- 100자 이상의 정성 리뷰를 남겨주신 분들께 녹화본 제공!
 - 네이버 스토어에 남겨주시면 되고 주문 내역 몰 수 있는 링크 카톡방에 올려놓겠습니다!
 - 띄어쓰기도 포함이라 2분이면 쓸걸요..? (첫 줄에 [라이브 수강자 대상 녹화본 제공 이벤트 참여]라고 써주세요)
 - 강의에서 가장 인상적이었던 것, 뭘 기대했고 뭘 얻게 됐는지 등.. 자유롭게 써주세요!
 - 예시:
[라이브 수강자 대상 녹화본 제공 이벤트 참여]
트랜스포머 아직 공부 안 한 뇌 삽니다.. 너무 재밌게 들었습니다 ㅎㅎ
- 사진까지 남겨주시면 더욱 좋습니다! 😊
- 필기하신 것 혹은 인상 깊었던 슬라이드, 코드를 폰으로 찍어서 올려주세요!
- 캡쳐본은 네이버에서 자동 필터링 됩니다 😢
- 다음 주말까지는 올려주세요!
- 톡방 제 프로필 눌러서 1:1 톡으로 후기 캡쳐해서 보내주시면 녹화본 제공 드립니다!