

04

## 어텐션 구조와 트랜스포머 모델

## 04 어텐션 구조와 트랜스포머 모델

### ④ 오늘 학습을 통해 우리는

- Seq2seq 부터 Attention까지 RNN 모델의 한계를 극복하기 위한 다양한 시도를 알아봅니다.
- 딥러닝 역사에 획을 그은 Transformer 모델의 구조와 작동 방식을 살펴봅니다.
- BERT 모델이 등장하게 된 계기와 학습 방법에 대해 탐구합니다.

# 목차

어텐션 구조와  
트랜스포머 모델

- 01 Seq2Seq
- 02 Attention Mechanism
- 03 Transformer
- 04 BERT

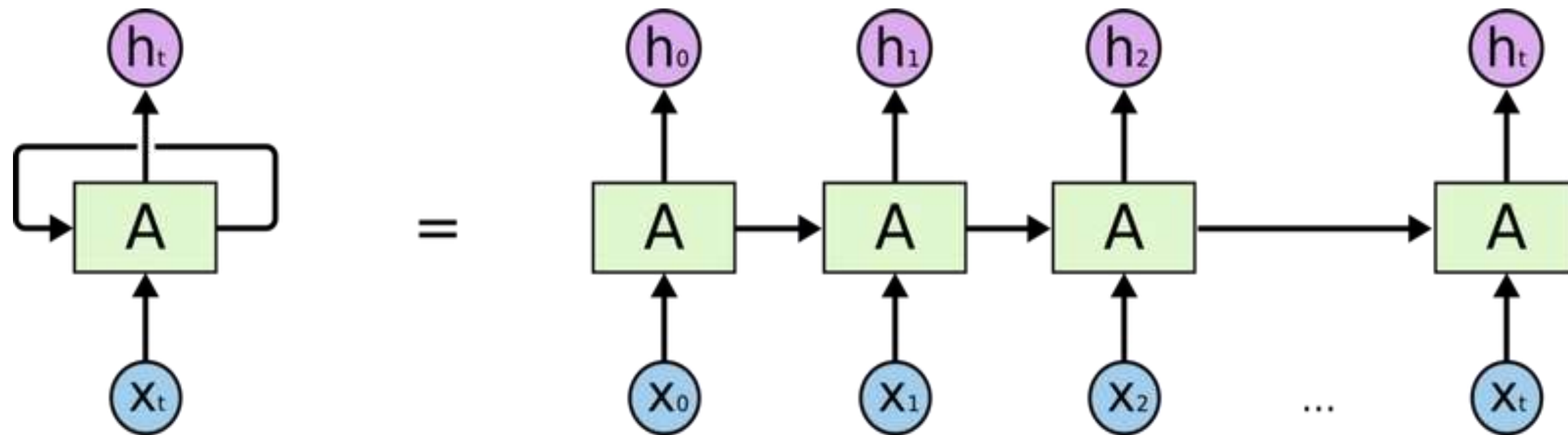
01

Seq2Seq (Sequence to Sequence)

## 02 자연어 모델의 변천사 - 순환신경망에서 Attention, Transformer로

### ☑ RNN의 장점

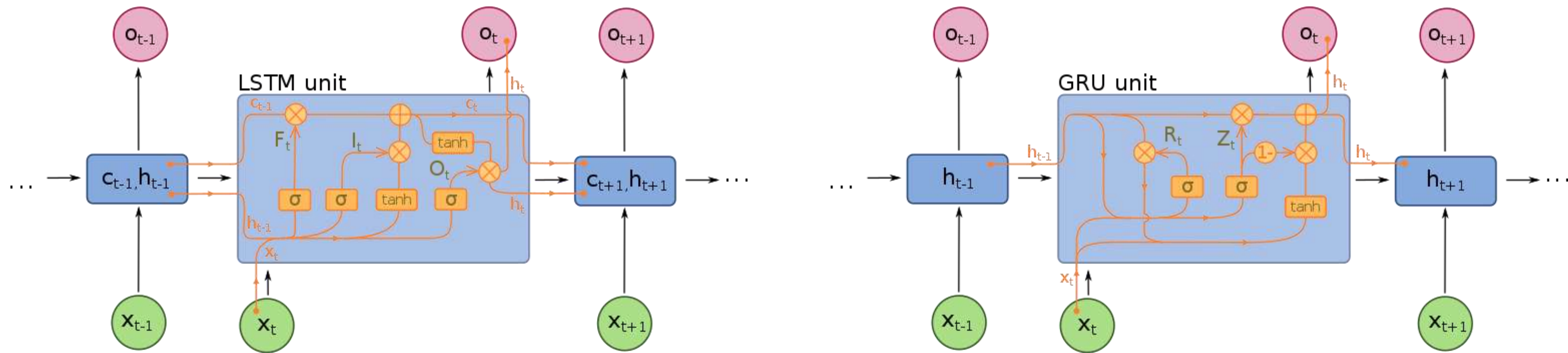
- 시계열 데이터와 같이 **순차적**인 특성을 필수로 갖는 데이터를 처리하기 위해 고안
- (입력의 차원, 출력의 차원)에 해당하는 **단 하나의 Weight**를 순차적으로 업데이트  
-> 자연어의 흐름(문맥) 파악에 적합



## 02 자연어 모델의 변천사 - 순환신경망에서 Attention, Transformer로

### ☑ RNN의 한계

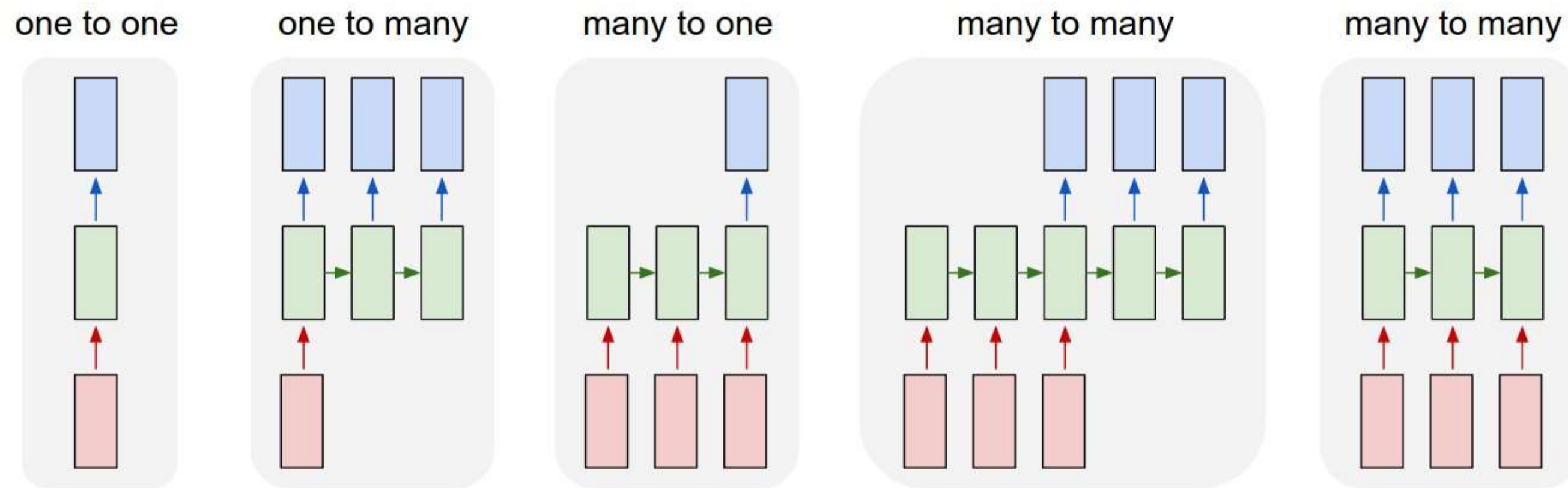
- 그러나, 위치상 거리가 있는 문자열일수록, 패턴파악이 어려워지는 현상 발생
  - 기울기 소실(Vanishing Gradient)
- 기울기 소실 현상 해소를 위해, LSTM, GRU 등의 모델이 등장



## 01 Seq2Seq (Sequence to Sequence)

### ④ RNN의 한계

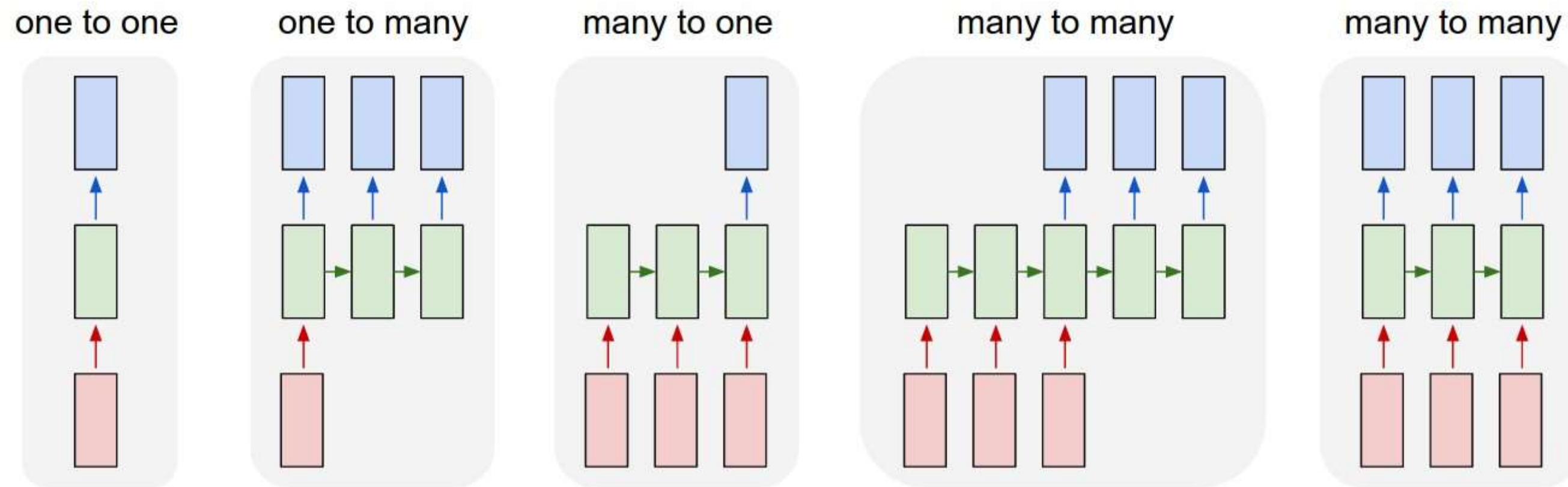
- 문장 생성 문제에서, 기존 RNN 기반의 모델은 구조적 한계를 지님
  - RNN을 이용한 텍스트 생성 모델은 Many-to-many 구조를 취함
  - 전통적인 RNN은 입력과 출력의 길이에 제한이 있음



## 01 Seq2Seq (Sequence to Sequence)

### ④ RNN의 한계

- 문장 생성 문제에서, 기존 RNN 기반의 모델은 구조적 한계를 지님
  - Input 단어의 수 = output 단어의 수
  - 이러한 한계는 번역에서 더욱 취약한 모습을 보임

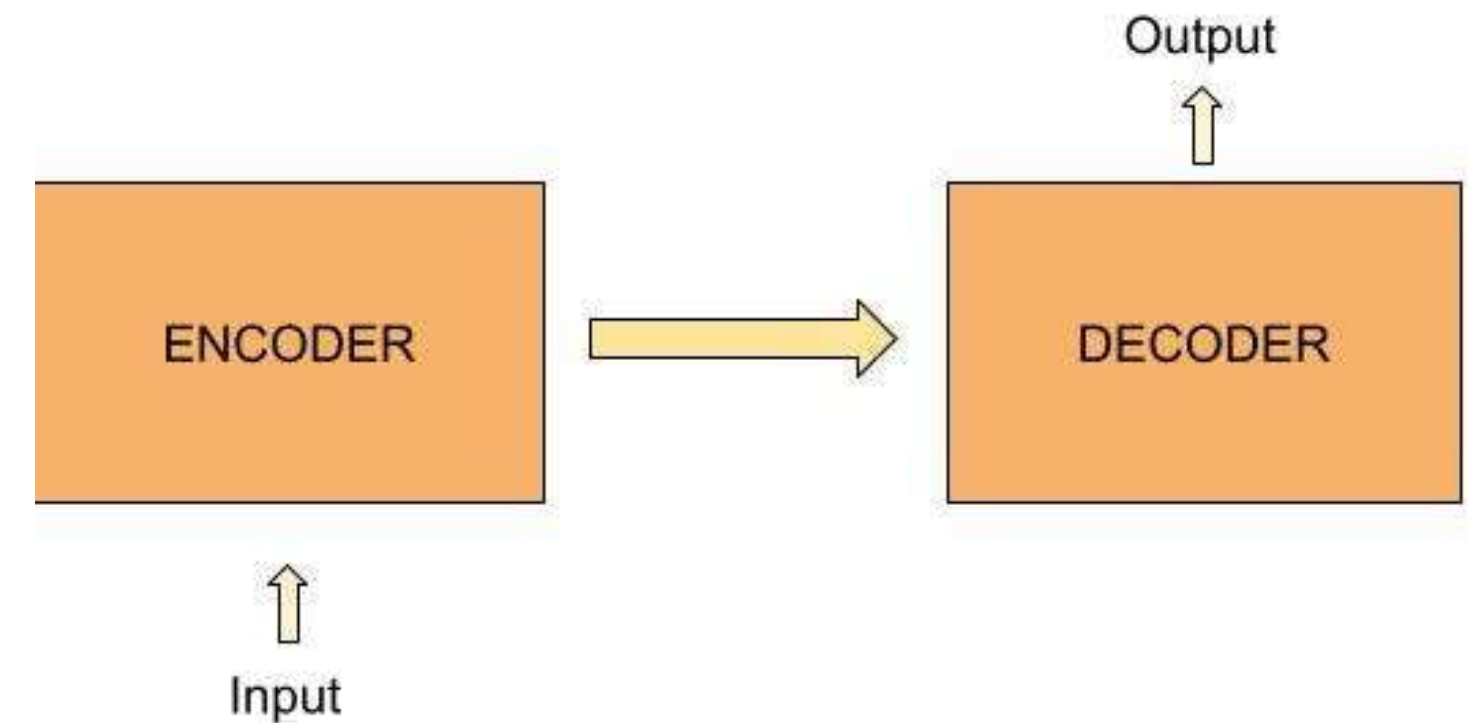




## 01 Seq2Seq (Sequence to Sequence)

### ④ Seq2Seq(Sequence to Sequence)의 기본 아이디어

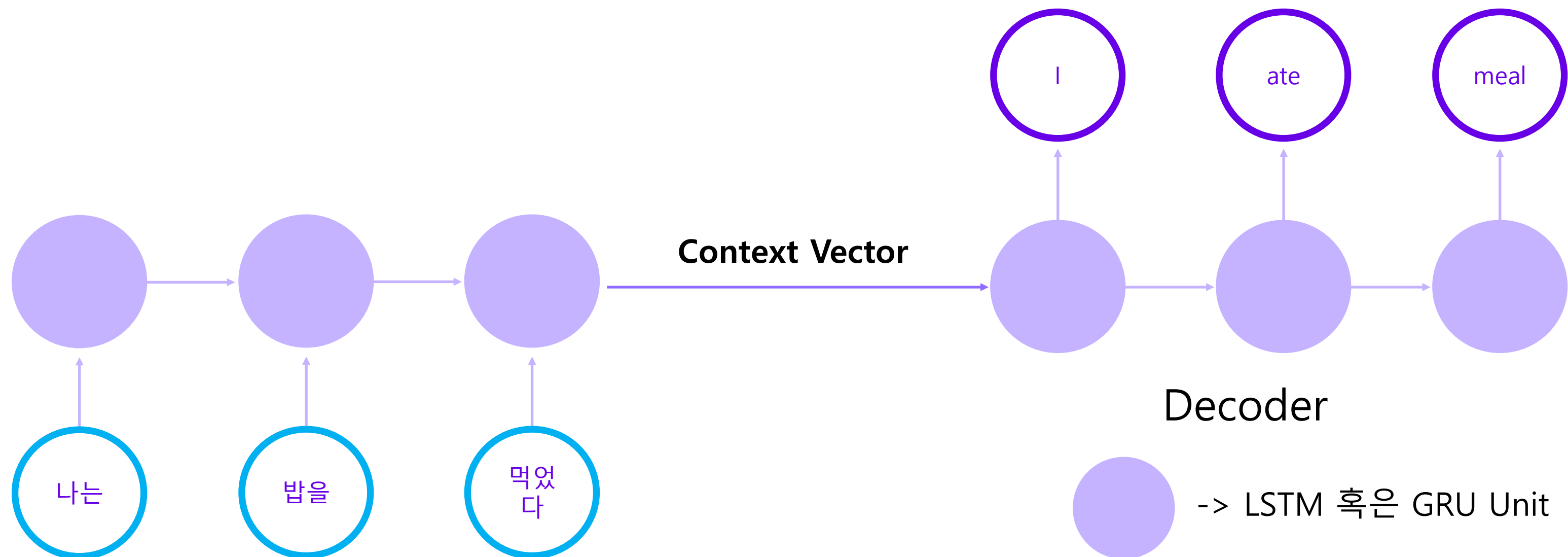
- 다양한 길이의 입력과 출력 시퀀스를 처리하기 위한 모델이 필요
  - 입력과 출력의 기능을 구조적으로 분리해볼까?
- 두 개의 RNN 구조 사용: 인코더와 디코더
  - 인코더에서는 문자열 입력만을 담당
  - 디코더에서는 문자열 생성만을 담당



## 01 Seq2Seq (Sequence to Sequence)

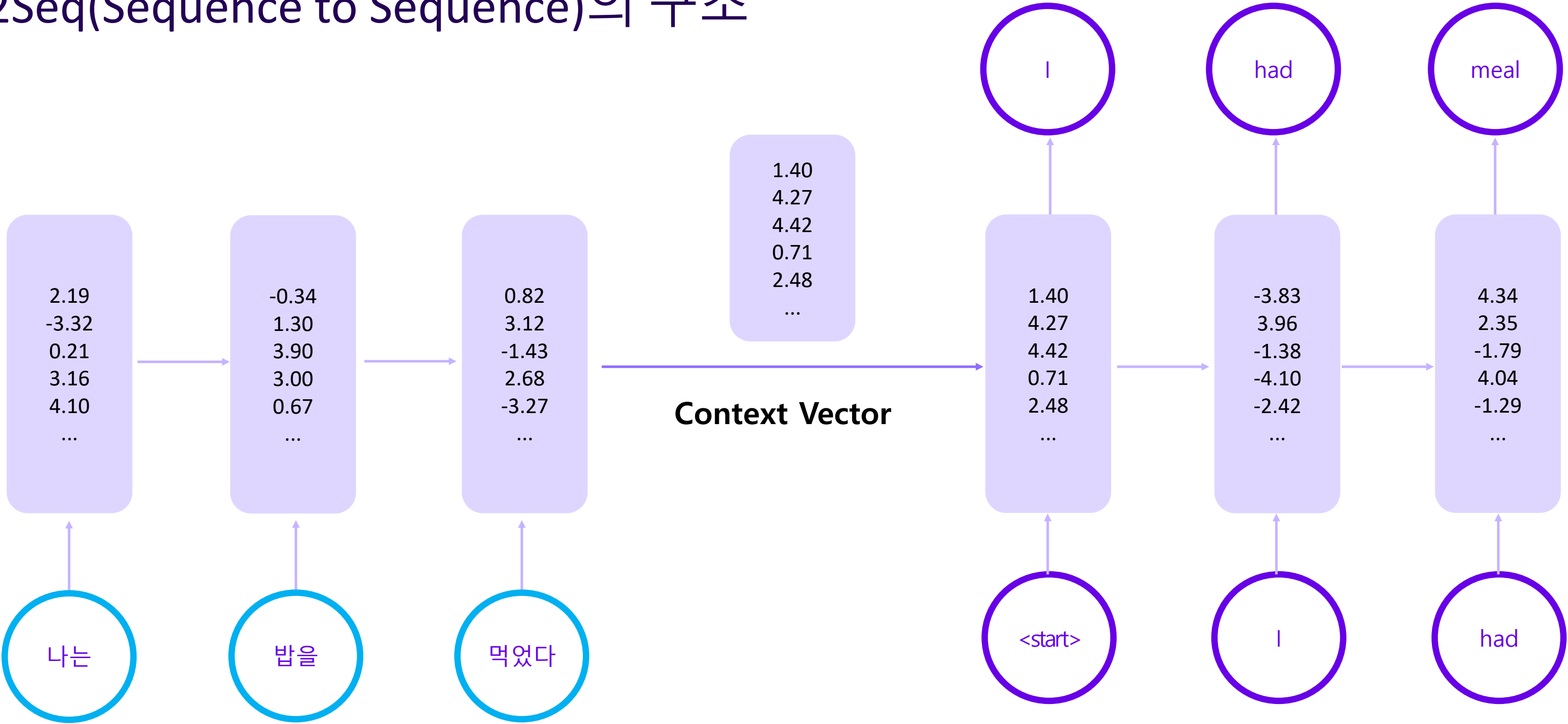
### ④ Seq2Seq(Sequence to Sequence)의 기본 아이디어

- 입력 시퀀스를 고정된 크기의 벡터로 변환
- 변환된 벡터를 기반으로 출력 시퀀스 생성



# 01 Seq2Seq (Sequence to Sequence)

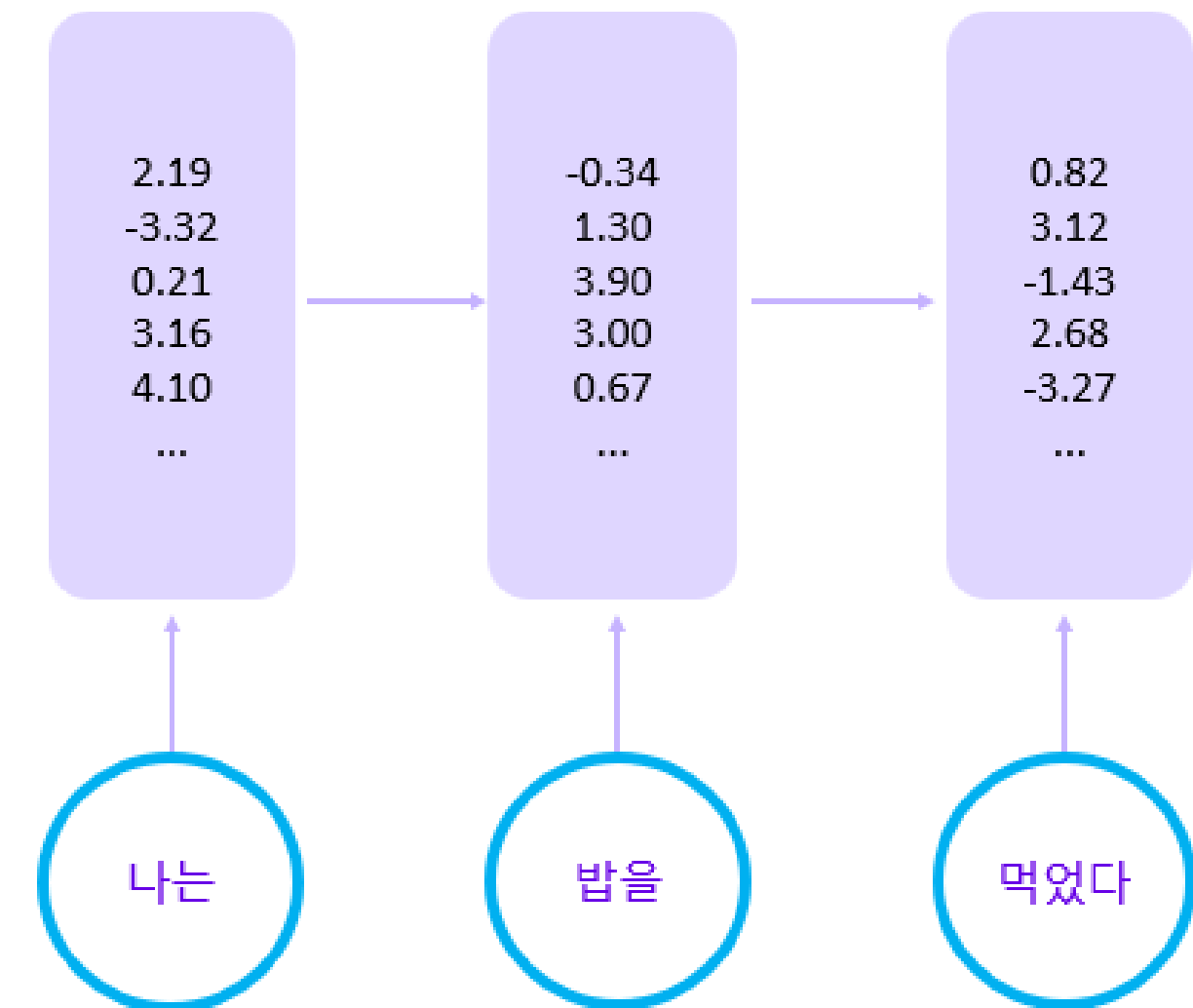
## ✔ Seq2Seq(Sequence to Sequence)의 구조



## 01 Seq2Seq (Sequence to Sequence)

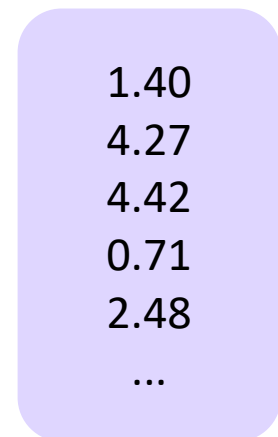
### ④ 인코더(Encoder)

- 입력 시퀀스를 받아들임
  - 입력된 문장을 이해하는 역할을 담당
- 여러 RNN셀을 통과하며 상태 정보를 축적
- 마지막 RNN셀의 상태를 **Context vector**로 출력



## 01 Seq2Seq (Sequence to Sequence)

### ☑ Context Vector



1.40  
4.27  
4.42  
0.71  
2.48  
...

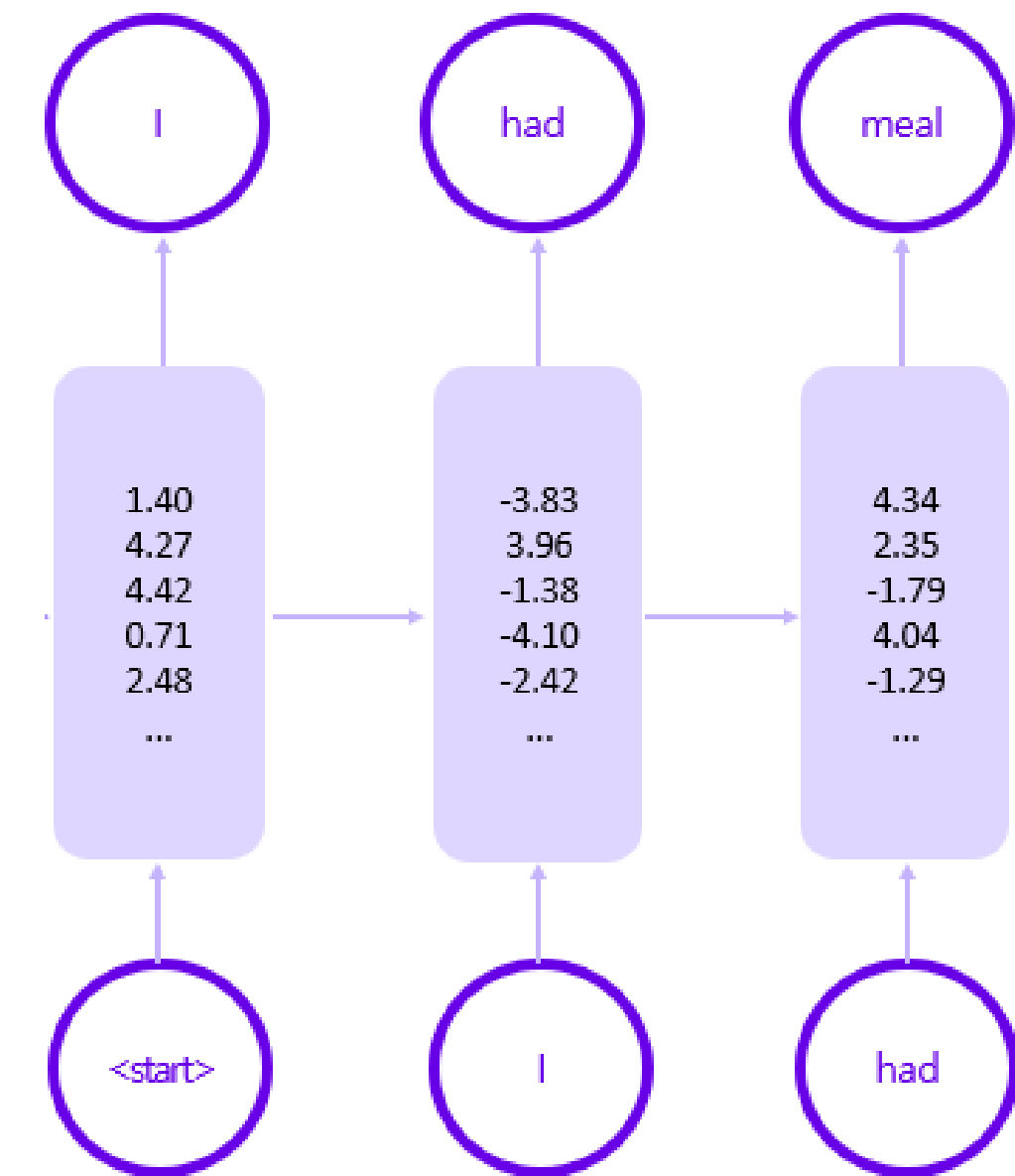
Context Vector

- 전체 문장을 축약한 고정된 크기의 벡터
- 인코더의 마지막 Hidden state를 전달받음
- 인코더가 읽은 텍스트의 문맥 정보가 포함
- 디코더가 문장을 생성할 때 참고하게 됨

## 01 Seq2Seq (Sequence to Sequence)

### ④ 디코더

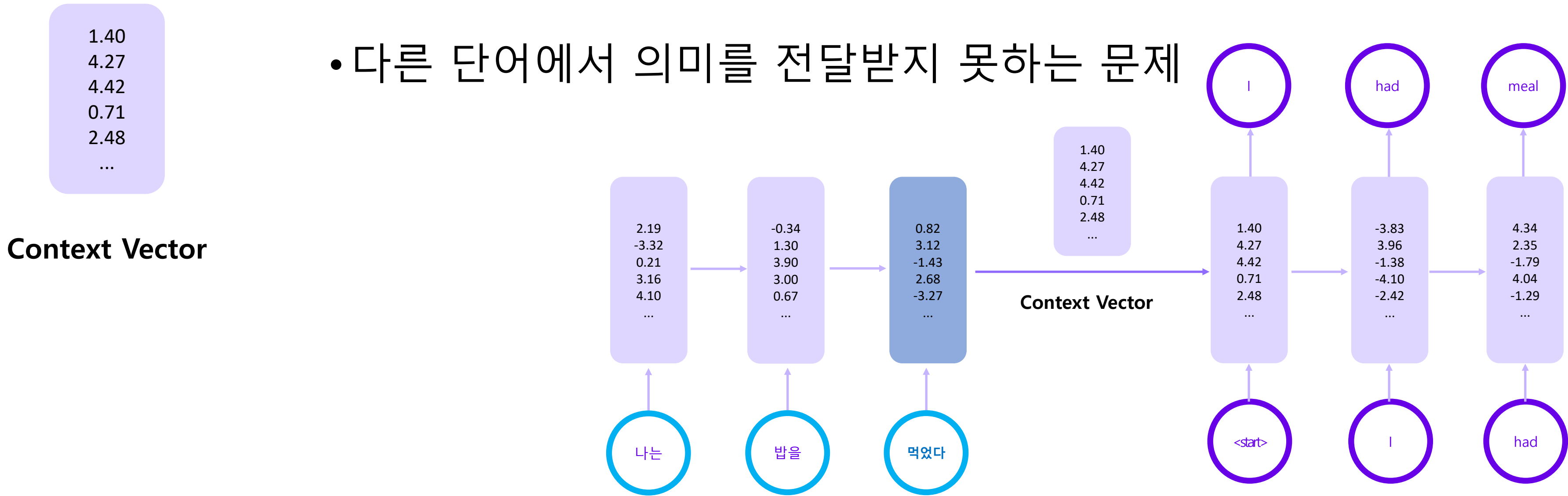
- 입력 문장에 대응되는 문장을 생성하는 역할
  - Language Model: 생성된 단어를 바탕으로 그 다음 단어를 예측하는 모델
- 문장 생성 시 Context 벡터를 참조함
  - Context vector를 초기 Hidden state로 사용



# 01 Seq2Seq (Sequence to Sequence)

## ☑ Context Vector의 한계

- 실제 번역기로 동작 가능
- Encoder의 **마지막 단어의 정보**를 가장 많이 기억
- 다른 단어에서 의미를 전달받지 못하는 문제



02

# Attention Mechanism



## 02 Attention Mechanism

### ④ Attention Mechanism의 등장 배경

- 기본 Seq2Seq 모델의 한계
  - Context vector의 한계
    - 고정된 크기의 컨텍스트 벡터로 모든 정보 압축
    - 마지막 단어에 의존성이 지나치게 높음
    - 긴 입력 시퀀스의 경우 정보 손실 발생 가능성(Vanishing gradient)
- RNN 기반 모델의 한계
  - Task 수행 시, 각 단어가 갖는 영향력을 정확하게 반영할 수 없음
    - 기존에는 텍스트 전처리를 통해 많은 부분 해결(TF-IDF, W2V 등)

## 02 Attention Mechanism

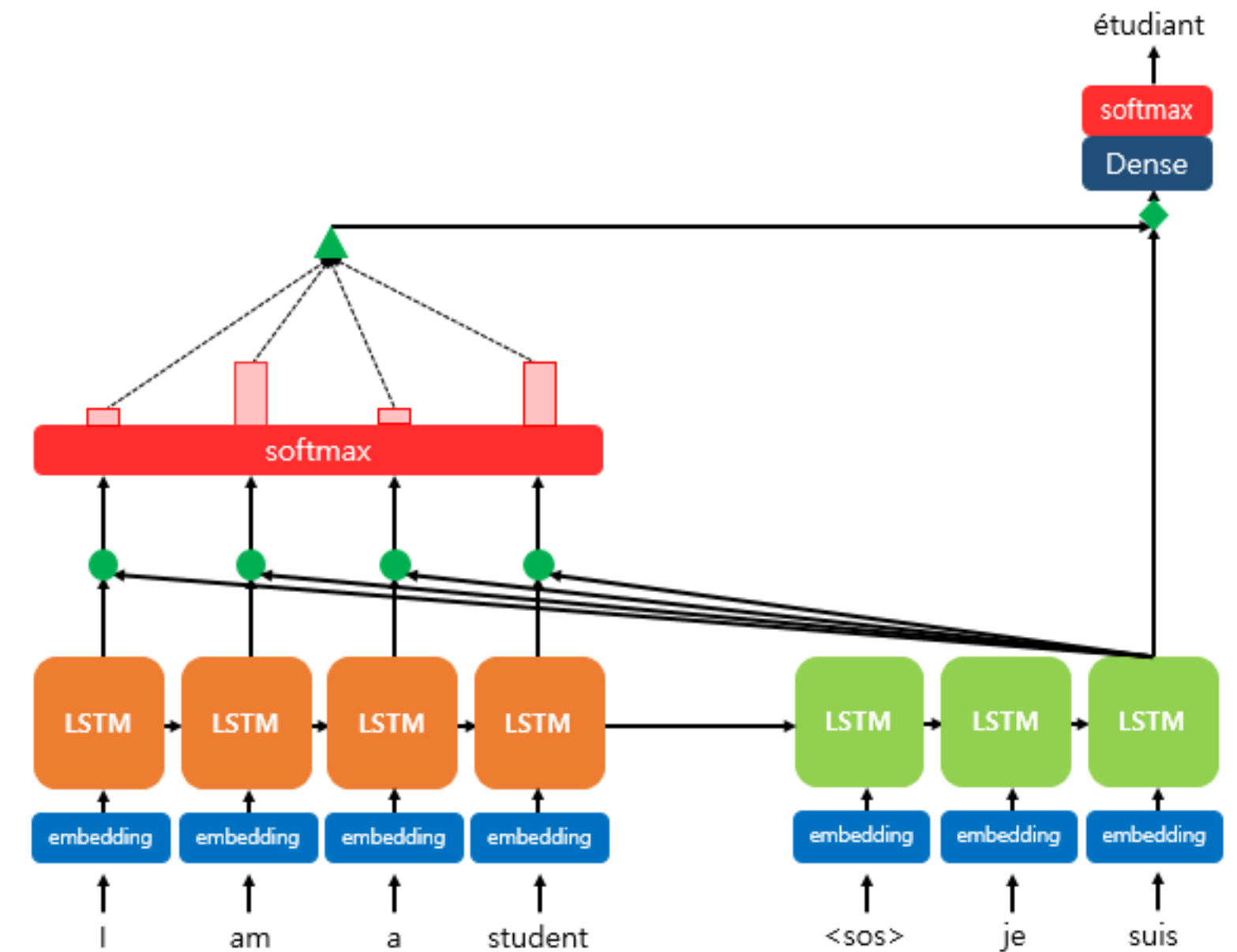
### ④ Attention Mechanism의 기본 아이디어

- 입력 시퀀스의 모든 부분이 동일한 중요성을 갖지 않음
- 디코더가 출력을 생성할 때 입력 시퀀스의 특정 부분에 "주목"
- 가중치를 통해 중요한 부분에 더 많은 주의를 기울임

## 02 Attention Mechanism

### ④ Attention의 작동 원리

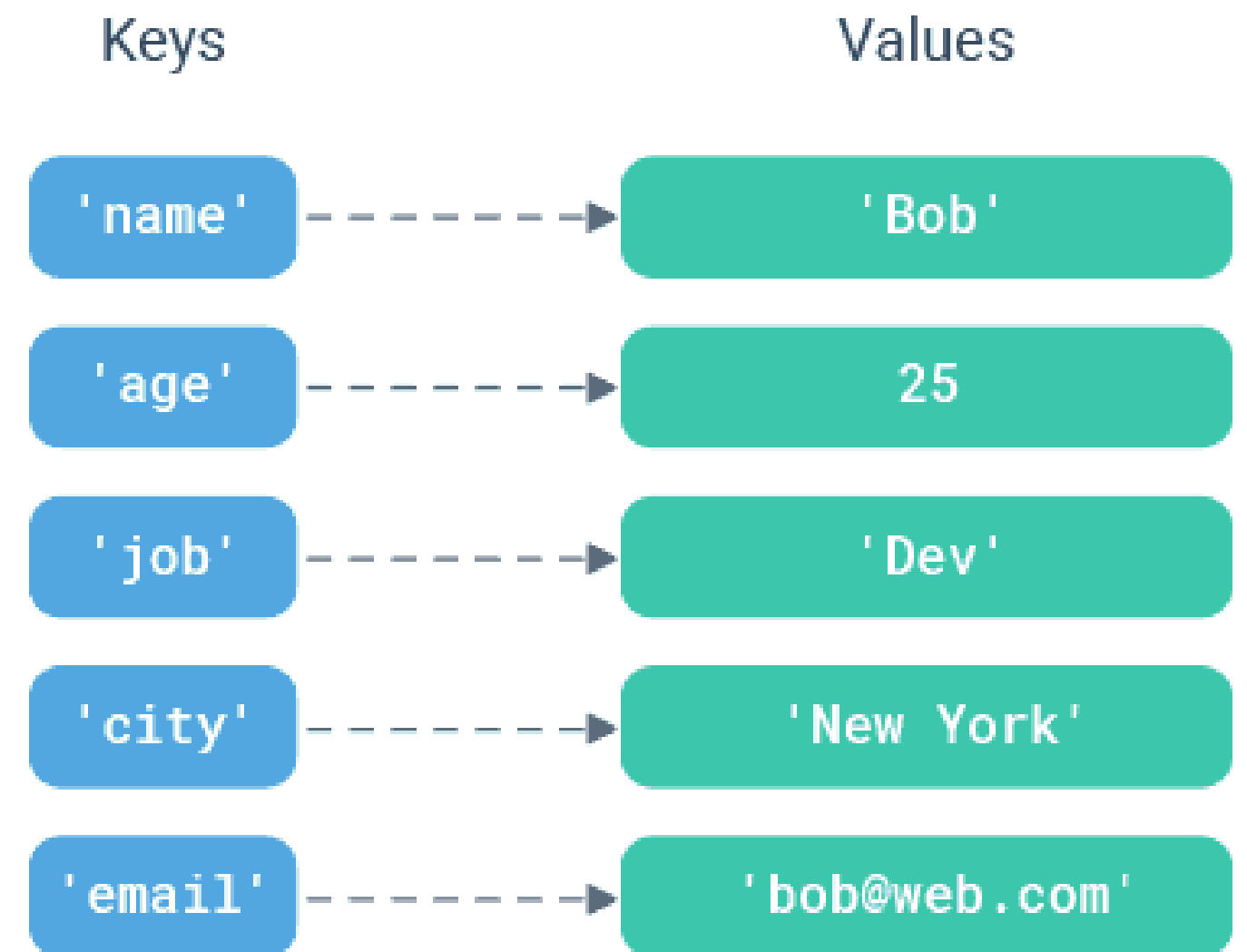
- 인코더는 입력 문장의 단어를 읽고 내용을 학습
- 각 입력 단어에 대한 Hidden state(**Key**)가 생성됨
- 디코더는 문장을 구성하는 단어(**Query**) 생성 시, 입력된 여러 단어 Hidden state의 기여도(**Value**)를 참고함



## 02 Attention Mechanism

### ④ Key, Value

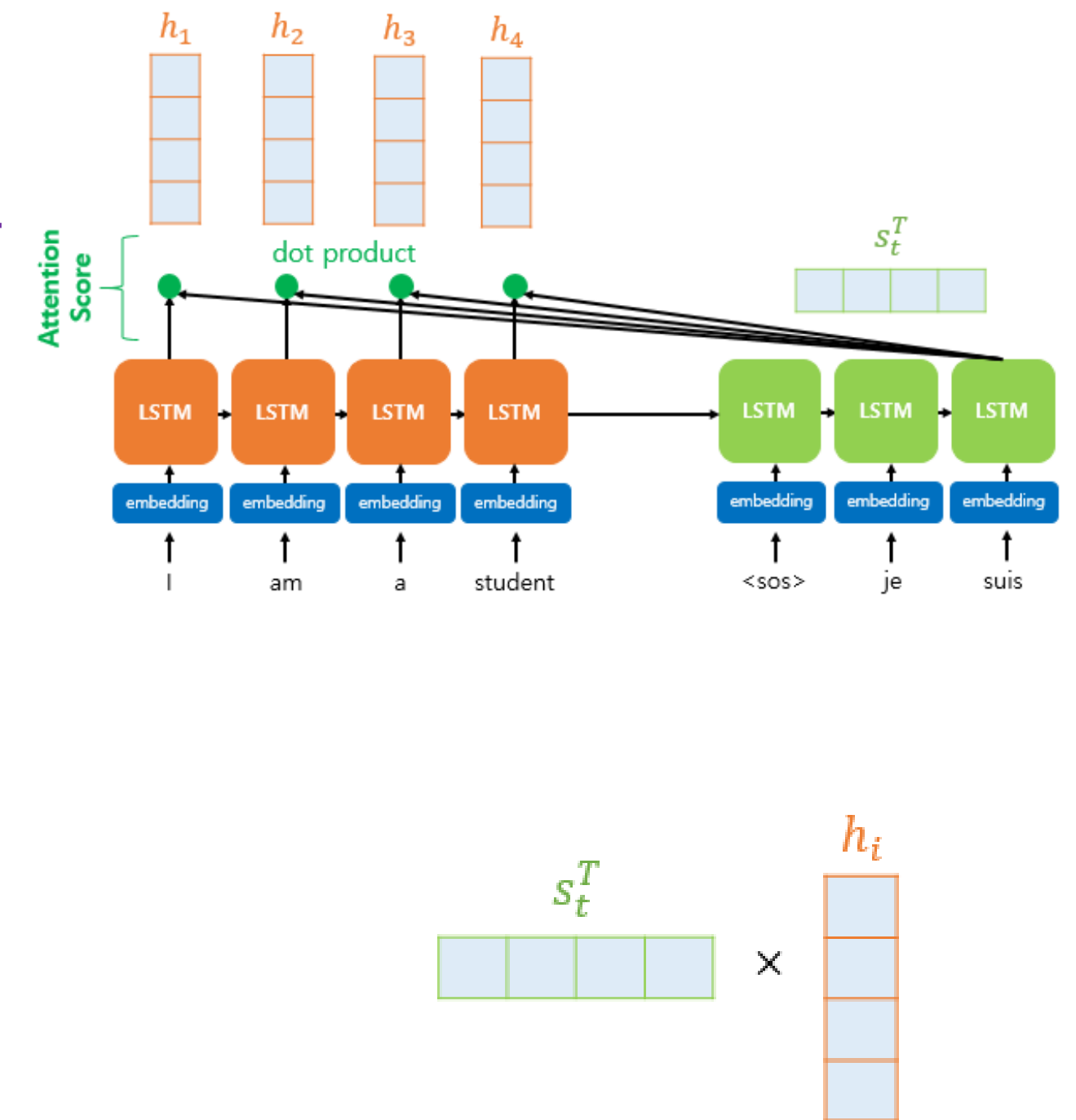
- Python 자료형 중 Dictionary에서는 데이터를 짝지어 저장
  - Key - Value
  - Key: 데이터를 호출하는 인덱스
  - Value: 호출한 인덱스의 실제 값
- 위 개념이 Attention에서 유사하게 사용



## 02 Attention Mechanism

### ④ Attention Score

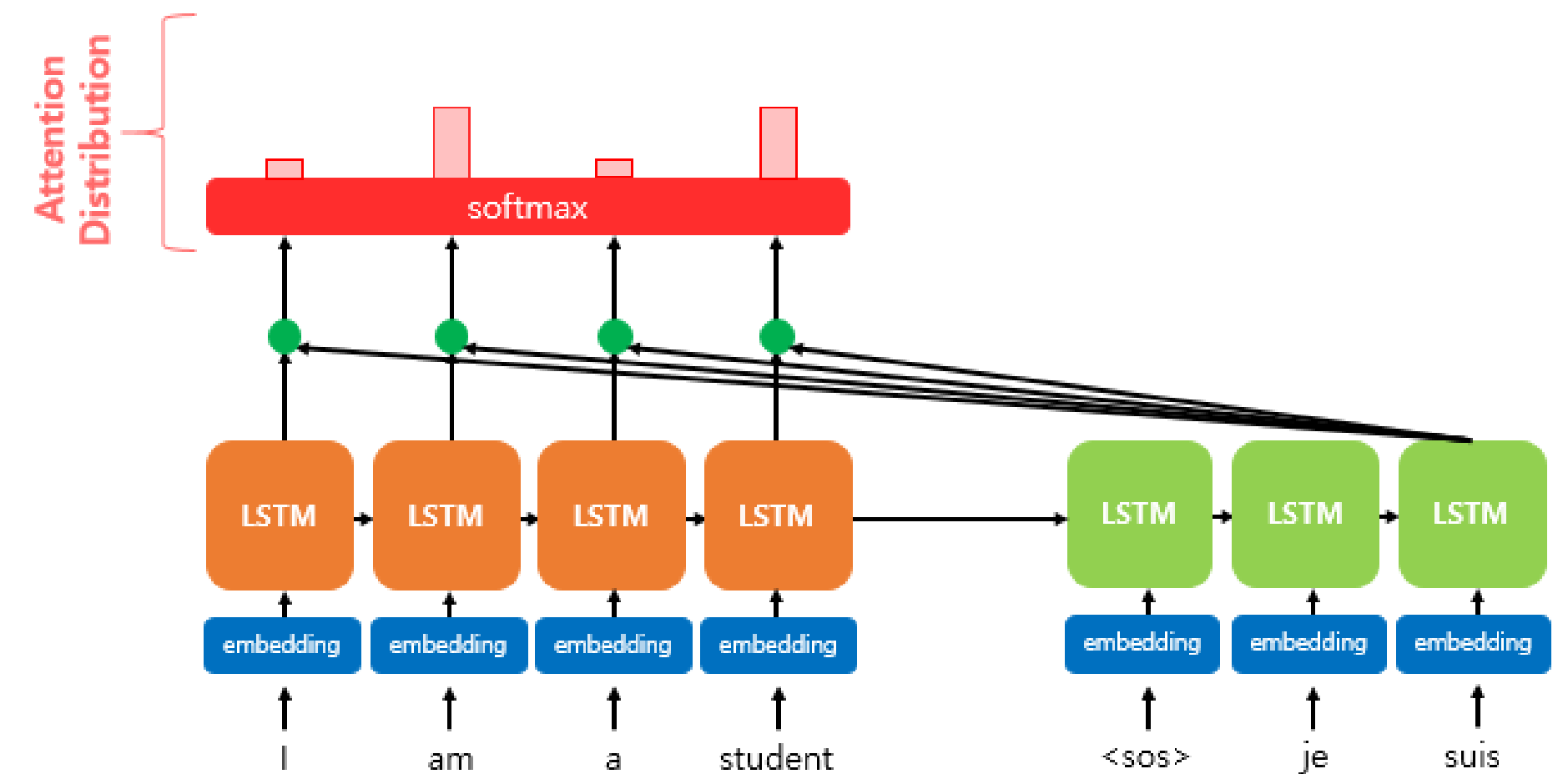
- 디코더에서 단어를 생성할 때, 인코더의 모든 Hidden state에 대해 각 요소가 디코더의 현재 Hidden state와 얼마나 유사한지를 측정한 점수 값
- 주로 내적을 통해 계산하며, 코사인 유사도 등 다른 연산을 사용하기도 함
  - Query와 Key의 내적 값



## 02 Attention Mechanism

### ④ Attention Distribution

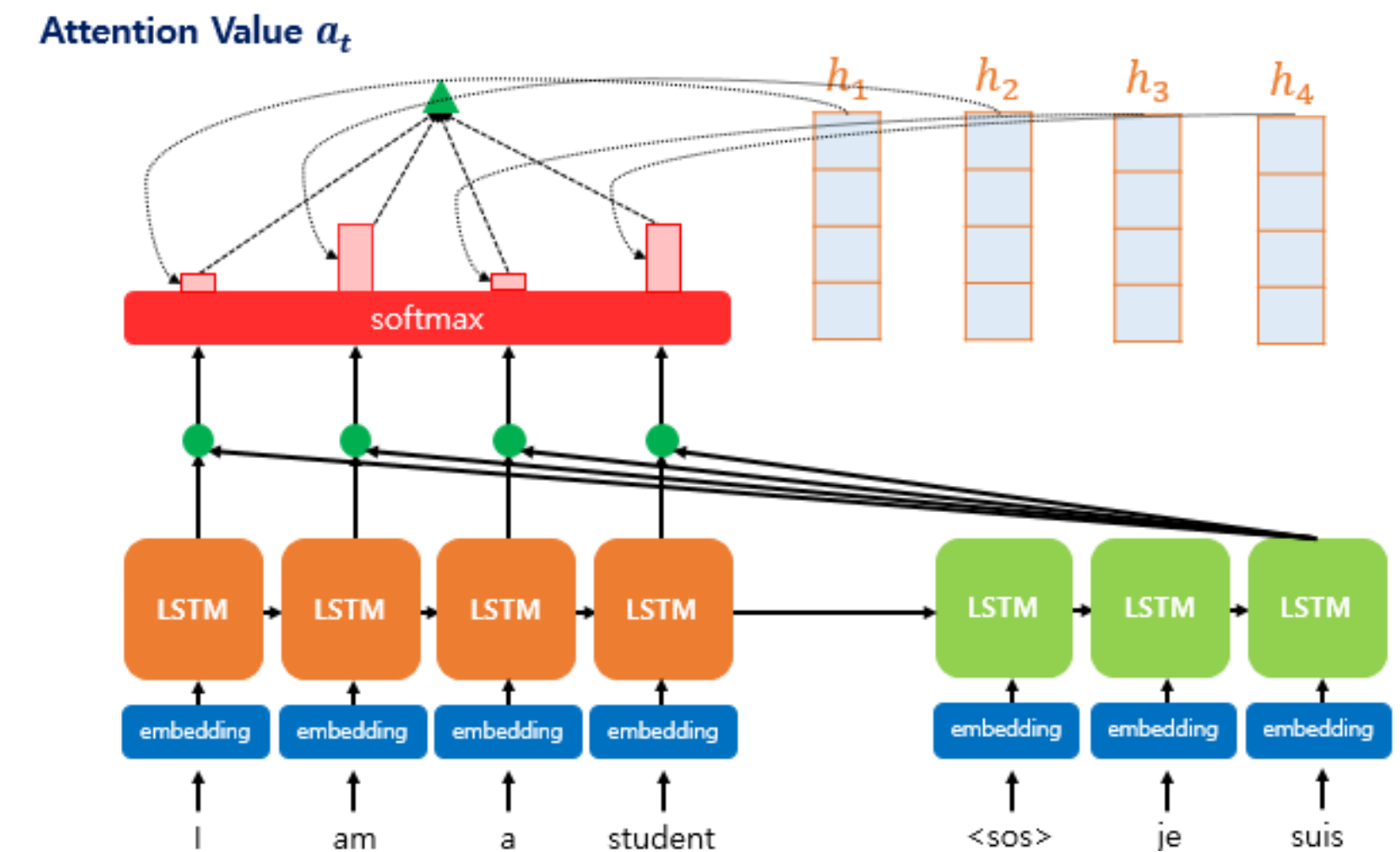
- 앞에서 구한 Attention score를 확률의 분포로 변환
  - 모든 인코더 Hidden state의 기여도가 확률 값으로 반환[0, 1]
- 각 기여도를 Attention weight라 함



## 02 Attention Mechanism

### ④ Attention Value

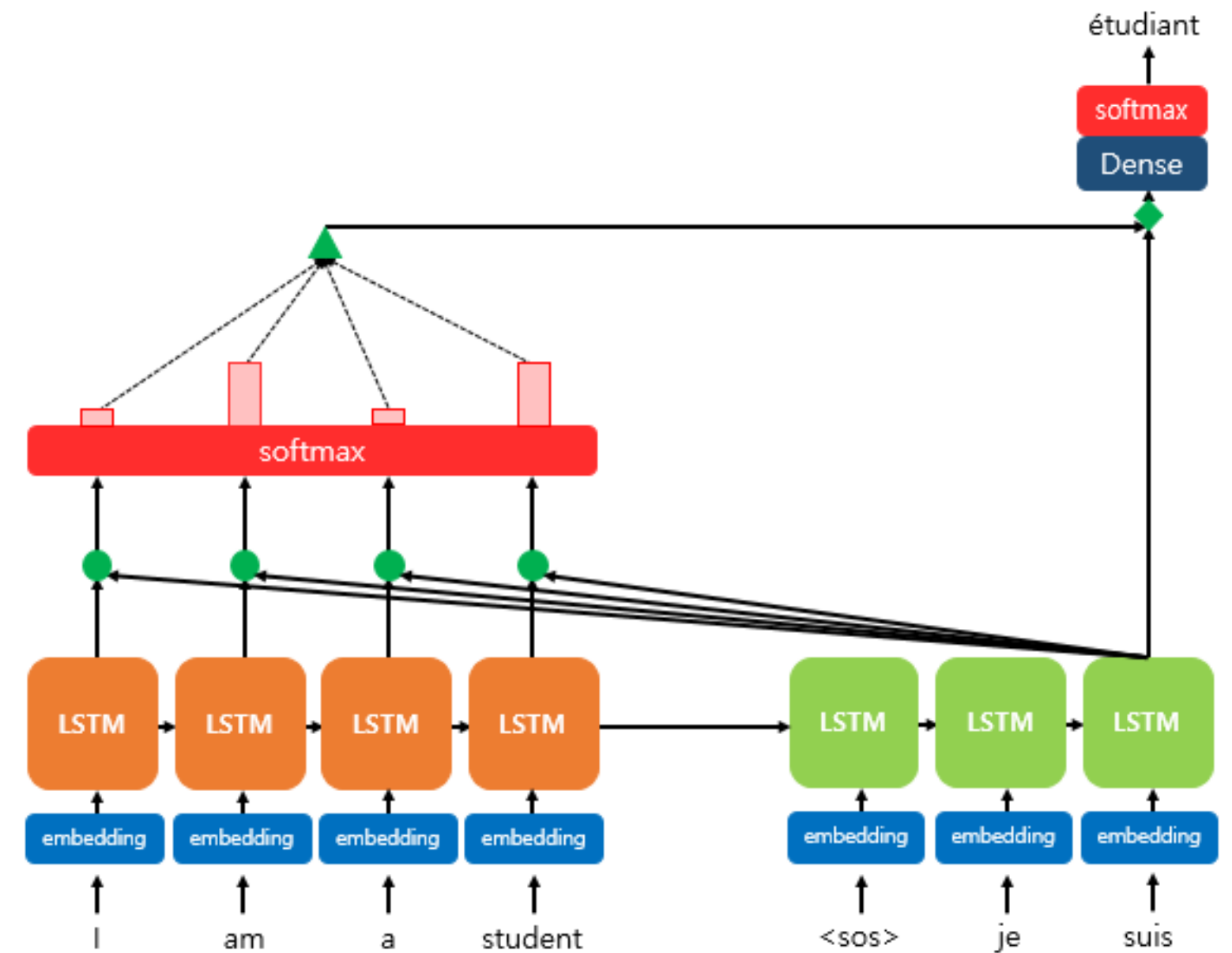
- 계산법
  - 각 인코더의 시점에서, 즉 매 단어마다
  - 각 단어의 Hidden state와
  - Attention distribution의 원소인 Attention weight를 곱한 후 모두 더함
- 의미
  - 디코더의 단어 생성 시 Hidden state 역할을 함
  - Context vector



## 02 Attention Mechanism

### ④ Attention의 결과

- 디코더는 주어진 컨텍스트 벡터를 사용해 출력 토큰을 생성
- 이 과정은 디코더의 각 단계마다 반복
- 각 단계에서의 주목 대상이 다름

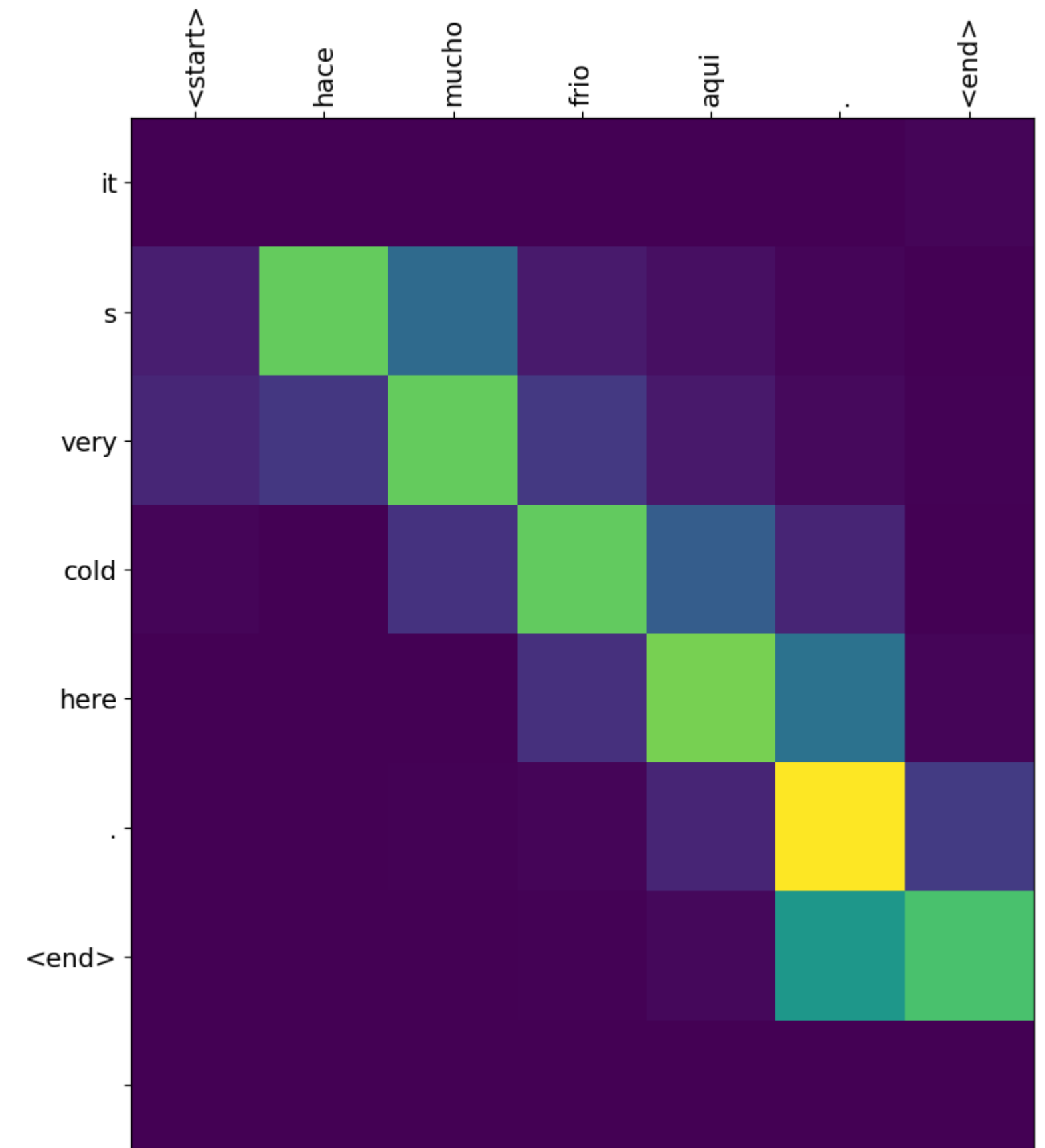




## 02 Attention Mechanism

### ④ Attention의 장점

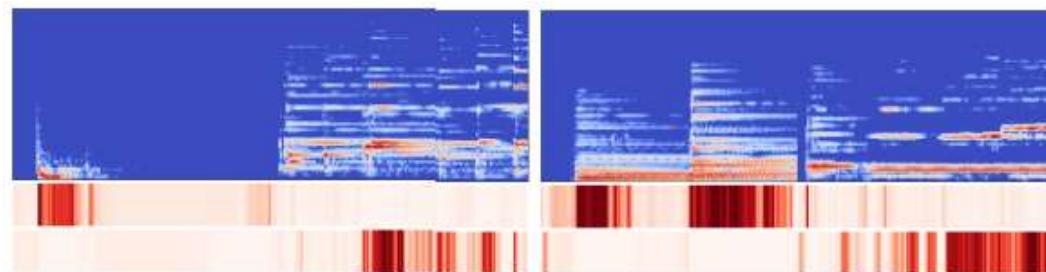
- 정보 손실 최소화: 긴 입력 시퀀스에서도 중요한 정보 유지
- 디코딩 시 입력의 특정 부분에 집중 가능
- 시각화를 통해 모델의 주목 포인트 확인 가능



## 02 Attention Mechanism

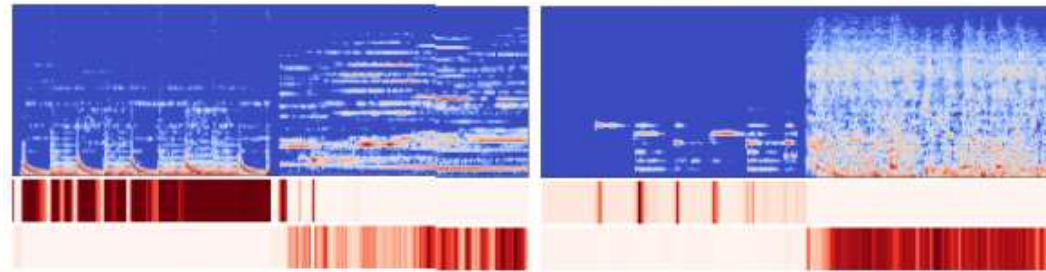
### ☑ Attention Mechanism 응용 사례

- Attention을 이용한 Music tagging & Image captioning



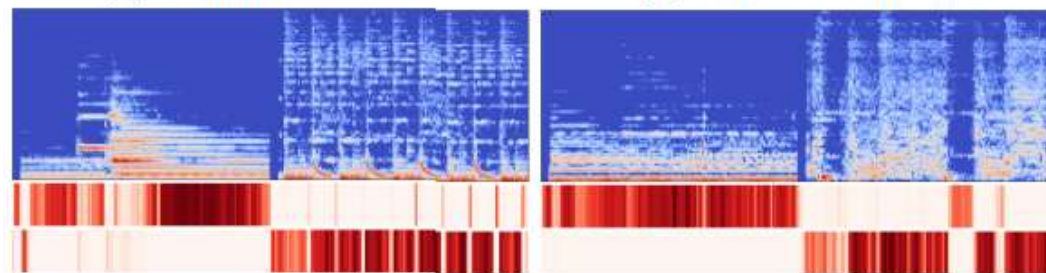
(c) Drums + Harp

(d) Piano + Flute



(e) Techno + Classic

(f) Classic + Metal



(g) Slow + Fast

(h) Quiet + Loud



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

03

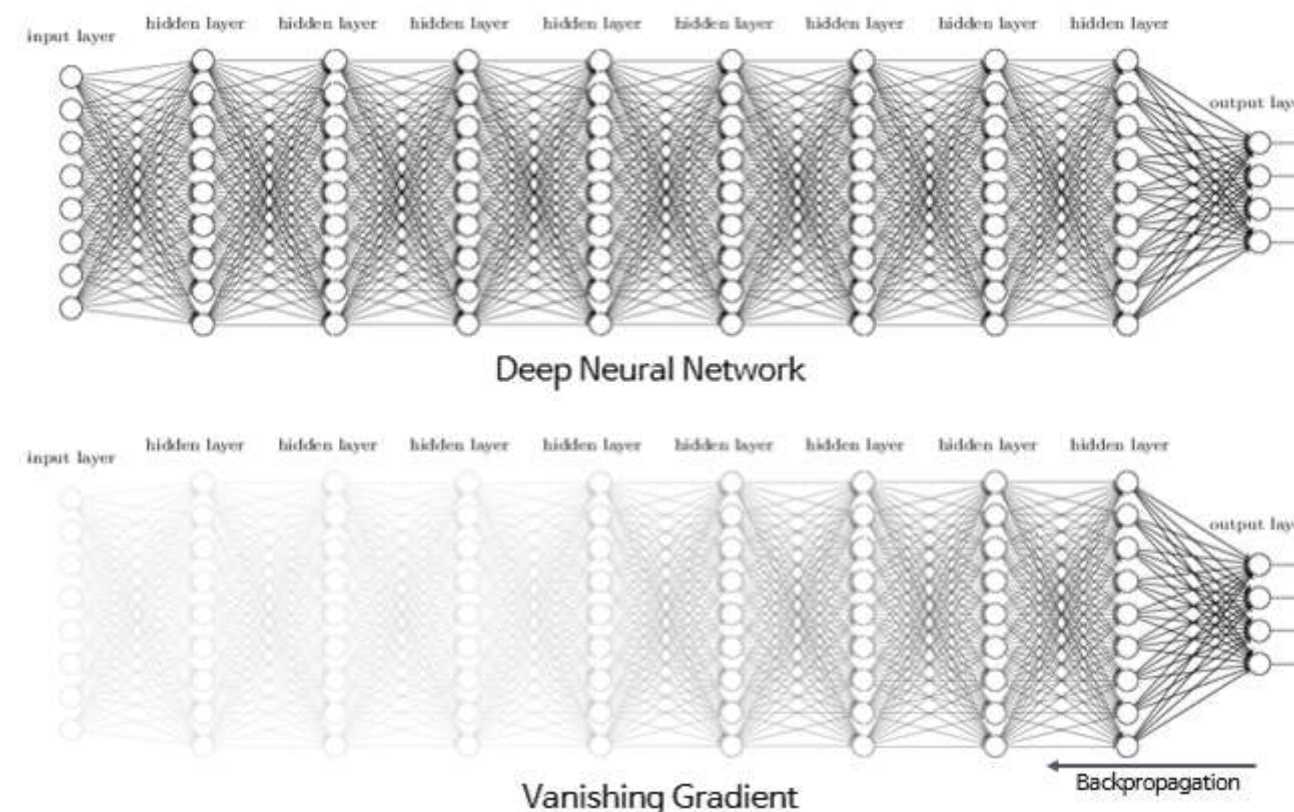
# Transformer



## 03 Transformer

### ④ RNN의 한계

- 연속적인 계산: 병렬 처리가 어려움
- 장기 의존성 문제: 긴 시퀀스에서 초기 정보의 손실
- 기울기 소실 및 폭발: 깊은 네트워크에서 학습의 어려움
- Seq2seq, Attention mechanism을 사용해도 본질적인 한계가 있음

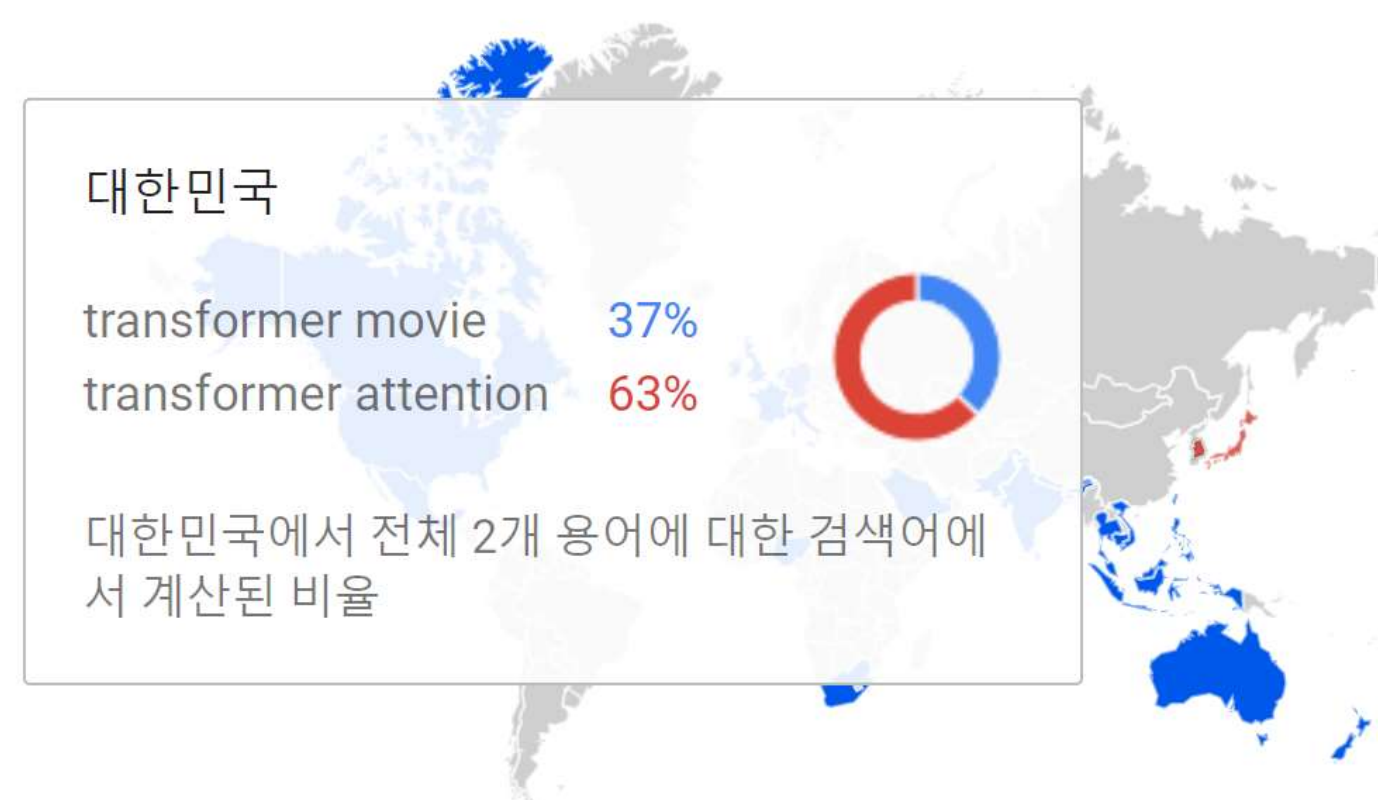


## 03 Transformer

### ④ Transformer

- Google에서 2017년 발표한 “Attention is All You Need” 논문의 핵심 인공지능 모델
- 후속 논문이 10만 건이 나올 정도로 자연어 처리 인공지능 업계를 발전시킴
- 유명 할리우드 영화제목과 같은 이름임에도, 경쟁력에서 밀리지 않고 있음

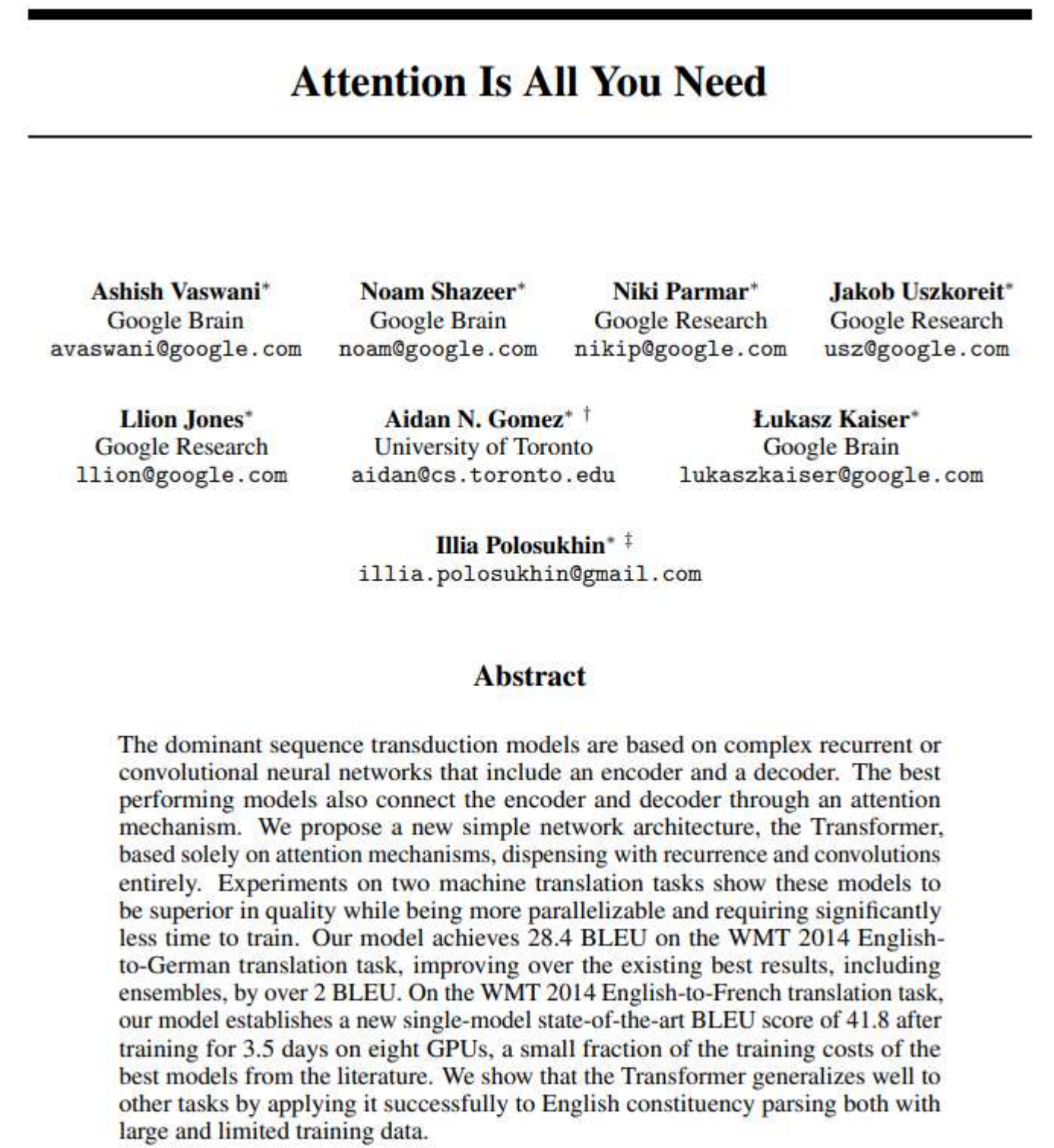
● transformer movie ● transformer attention



## 03 Transformer

### 👉 Attention is All You Need

- Google에서 2017년 발표한 논문
- RNN이나 CNN 없이 Attention만을 사용
- 병렬 처리 가능: 학습 속도 향상
- Self-Attention을 통해 시퀀스 내 모든 토큰 간의 관계 파악





## 03 Transformer

### ✓ Transformer

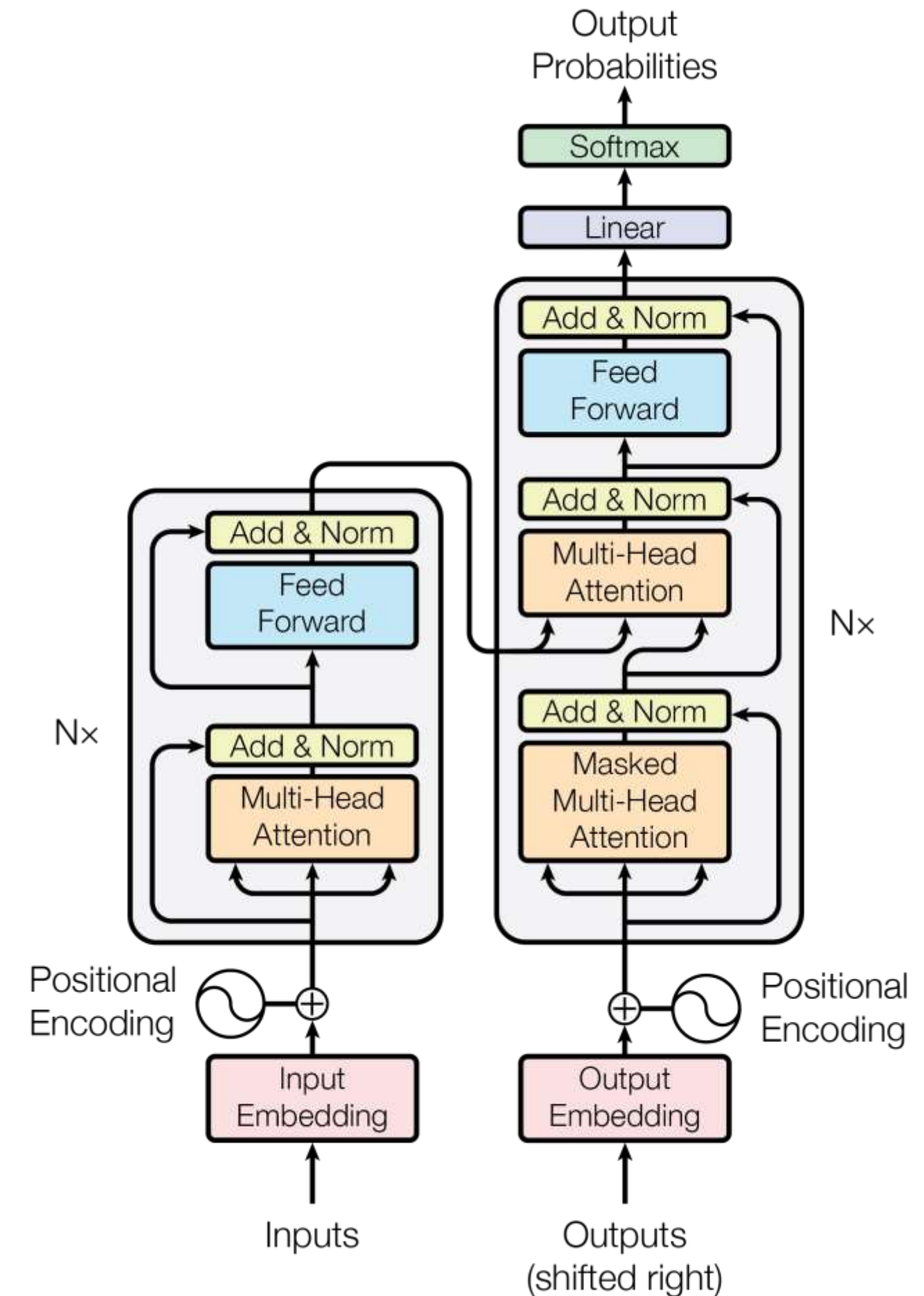
- 자연어 처리 번역 테스트에서 각 부문마다 SOTA(State of the Art) 달성
- BLEU score 41.8
- 후속 논문 10만 건

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	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	<b><math>3.3 \cdot 10^{18}</math></b>	
Transformer (big)	<b>28.4</b>	<b>41.8</b>	$2.3 \cdot 10^{19}$	

## 03 Transformer

### ☑ Transformer의 원리

- Positional Encoding
- Model Architecture - Encoder Decoder
- Self-Attention
- Multi-Head Attention
- Feed Forward Network

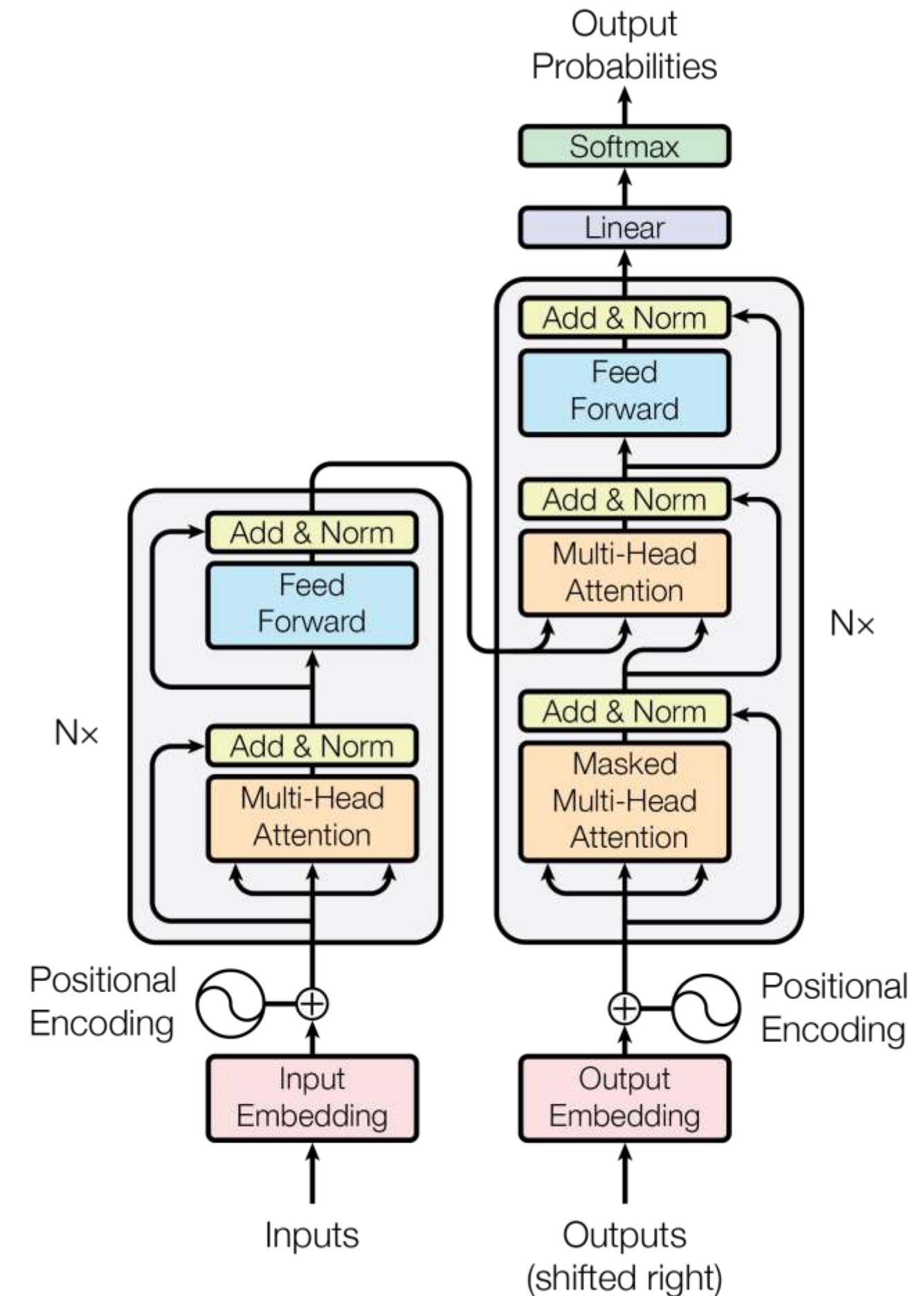




## 03 Transformer

### ☑ Transformer - Model Architecture

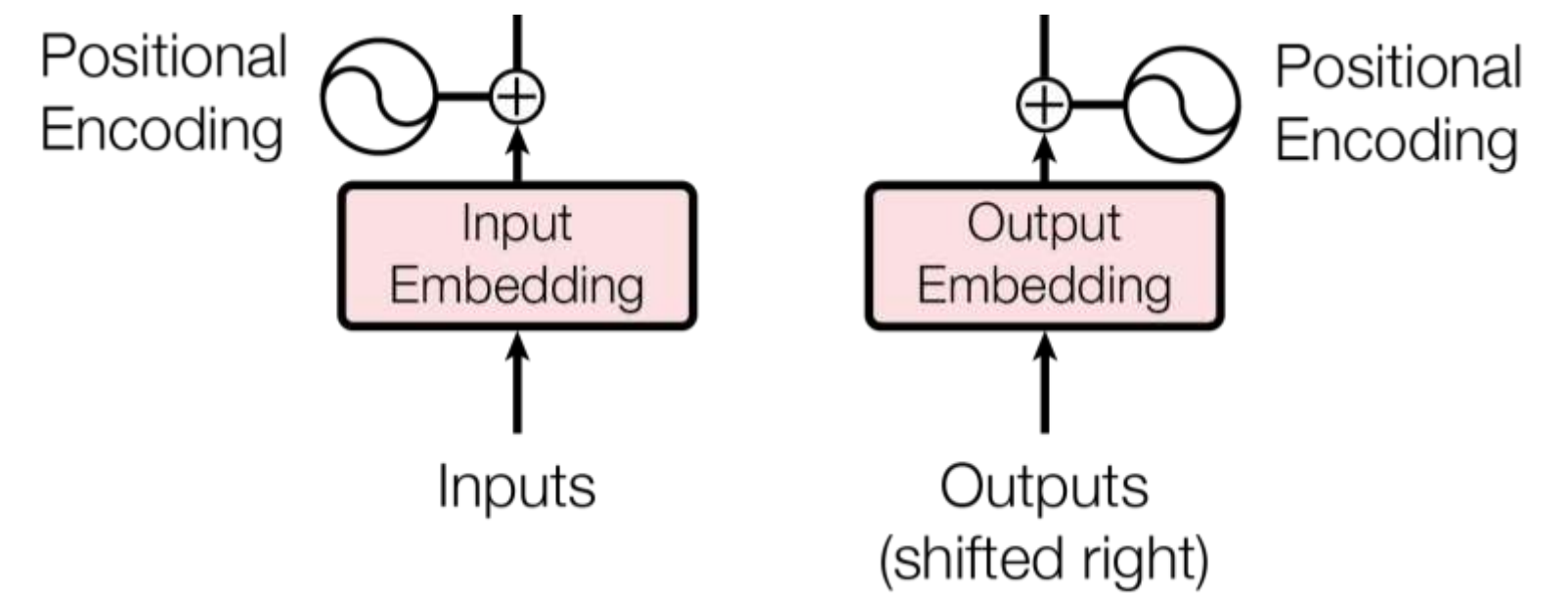
- 고도화된 여러가지 기법이 한꺼번에 적용됨
- 모델의 용량이 기존에 비해 큰 편
- RNN이 없음 -> Attention is All You Need
  - 문자열 데이터가 시계열 형태로 입력되지 않음



## 03 Transformer

### ④ Positional Encoding

- Transformer는 **순서 정보가 없는 구조**
  - 그러나 문자열 처리 시 순서 정보가 중요함
- 시퀀스 내 토큰의 위치 정보를 주입하기 위한 방법
- 고정된 벡터를 사용하여 각 토큰의 위치 정보를 인코딩

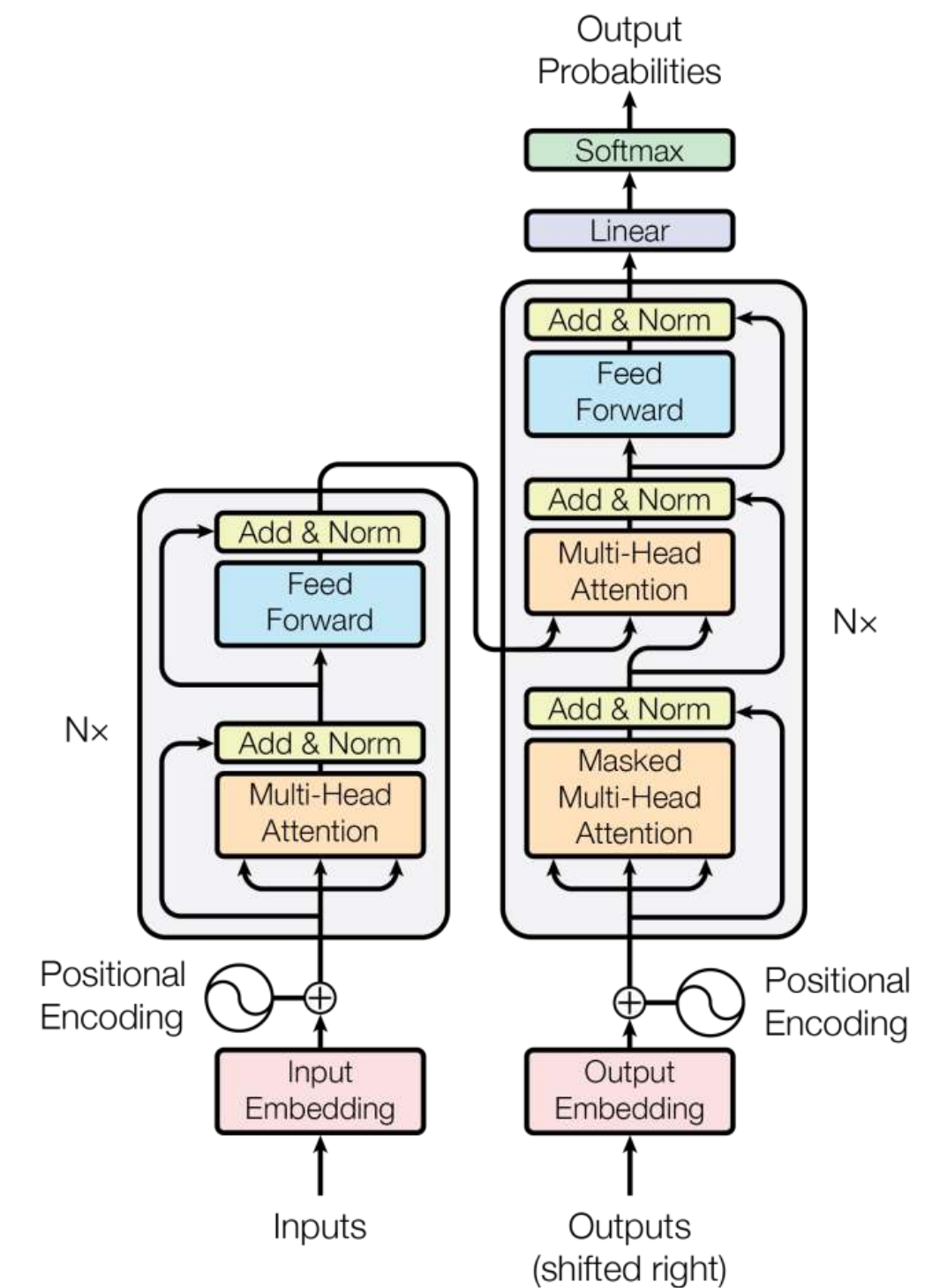


$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

## 03 Transformer

### ☑ Encoder – Decoder 구조

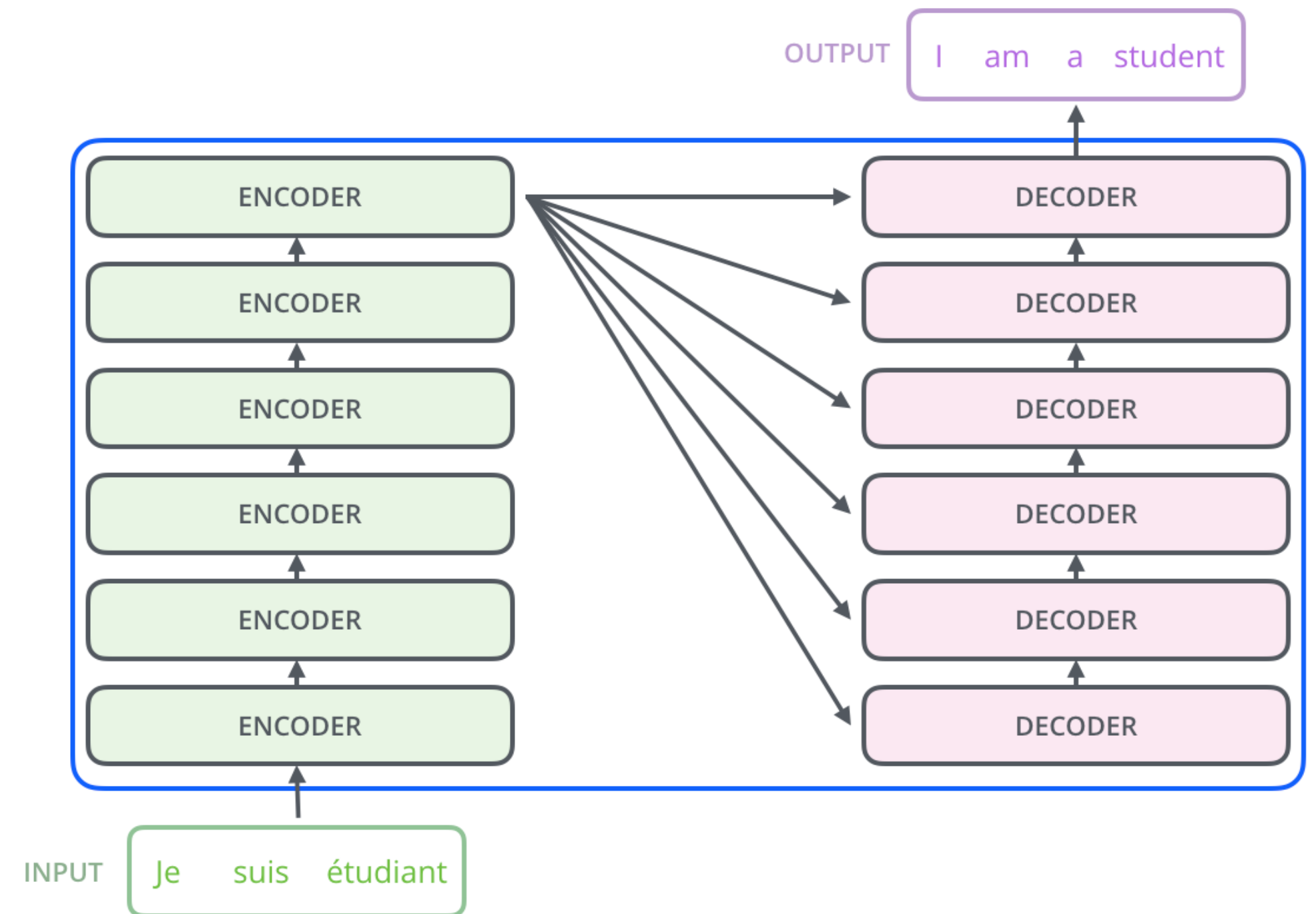
- 논문에서 공개된 Transformer 모델은 **번역기(Machine translation)**
  - English – French(WMT2014)
- 인코더
  - 영어 문장을 학습
- 디코더
  - 프랑스어 문장을 학습
  - 영어 문장의 문맥을 참조하여 **프랑스어 문장을 생성**



## 03 Transformer

### ☑ Encoder – Decoder 구조

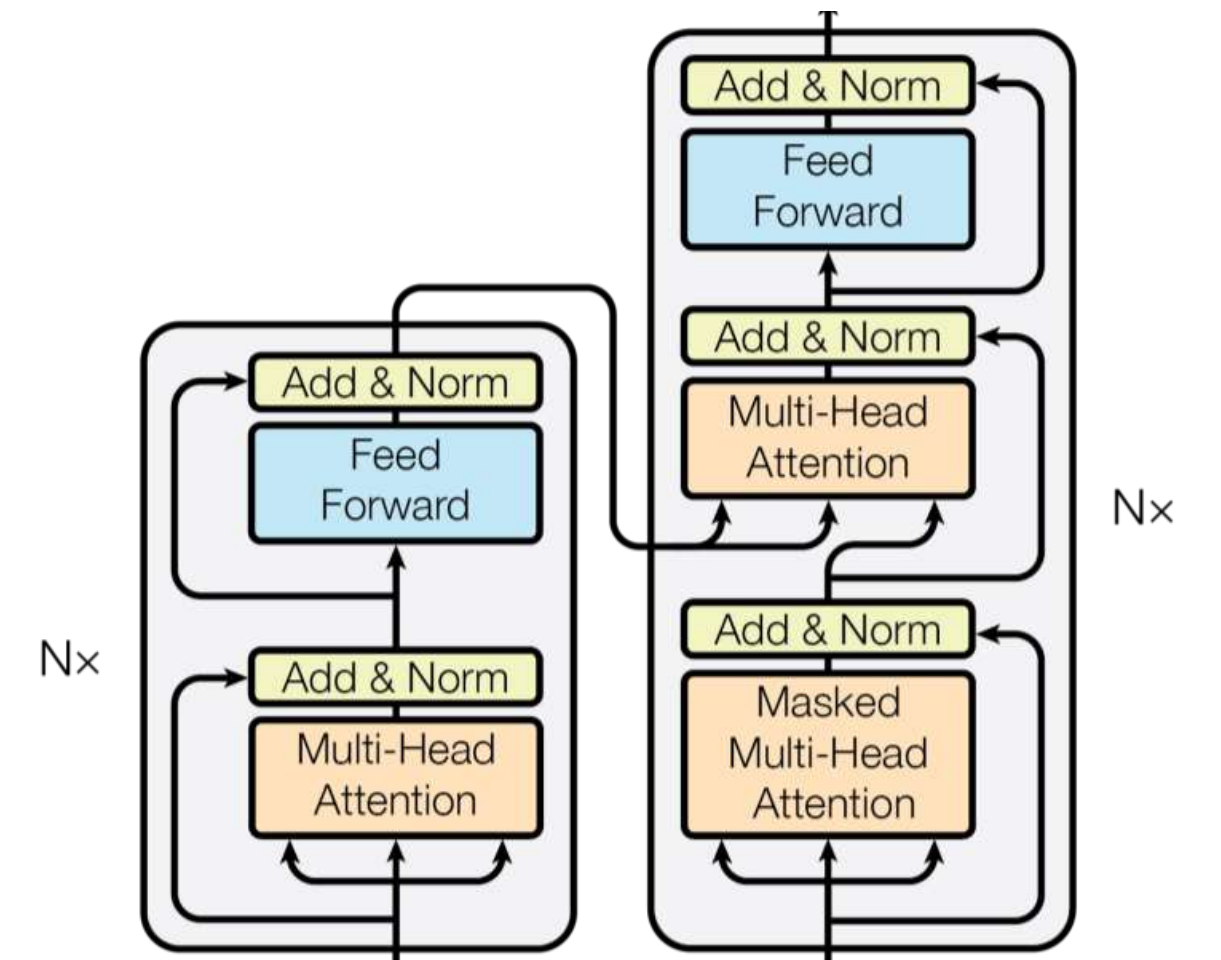
- 실제로, 한 모델에서 인코더와 디코더를 여러 층으로 쌓음
  - 논문 기준
  - 작은 모델: 6층
  - 큰 모델: 12층



## 03 Transformer

### ☑ Attention Block

- Attention Block들이 자연어 문장에서의 문맥을 학습
  - 인코더
    - Multi-head attention: 영어 문장 내 문맥 학습
  - 디코더
    - Masked multi-head attention: 프랑스어 문장 내 문맥 학습
    - Multi-head attention(위): 프랑스어 문장 생성 학습

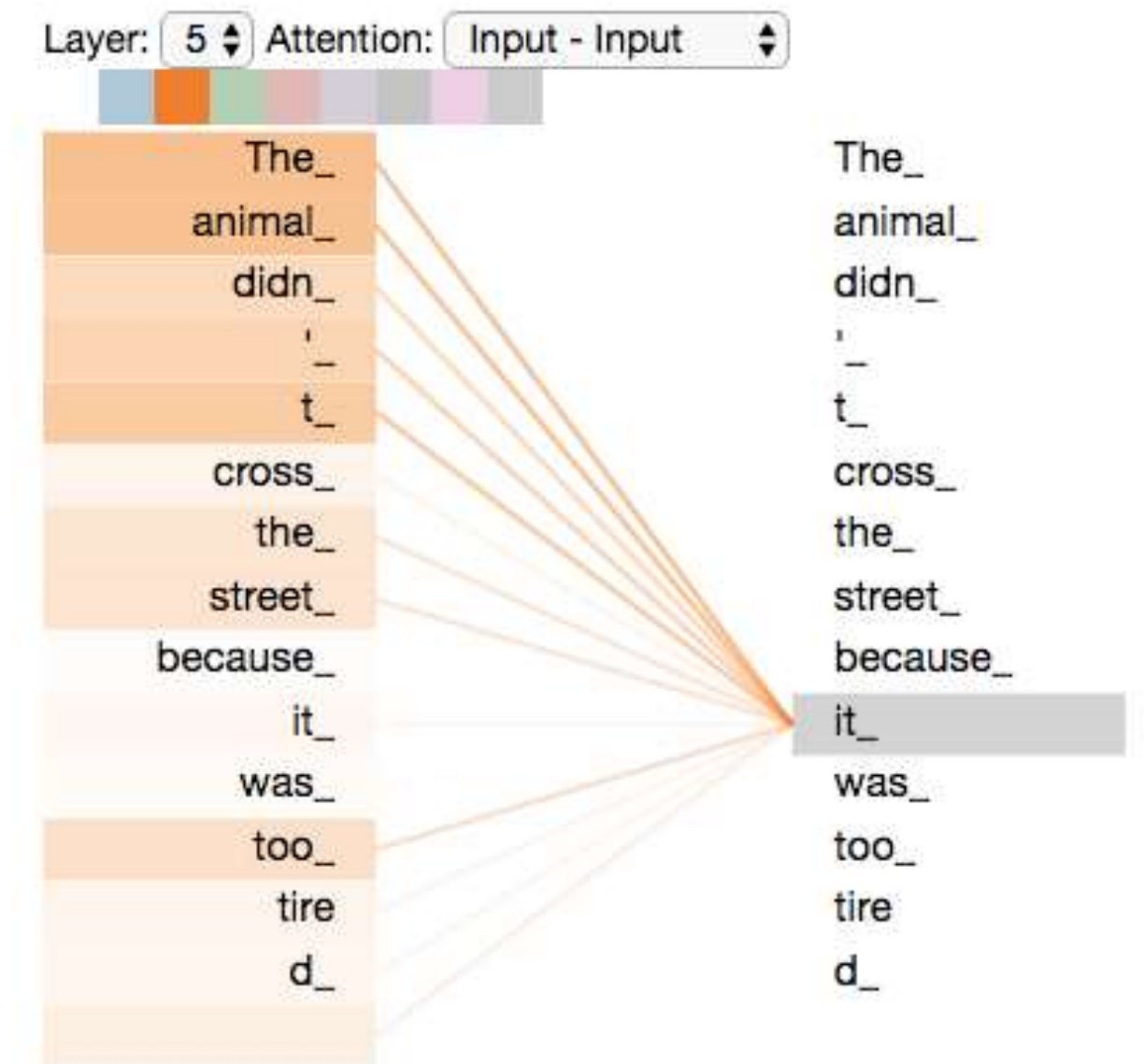




## 03 Transformer

### ☑ Self-Attention

- Transformer의 핵심 원리
  - 시퀀스 내 모든 토큰 간의 관계 파악
  - Query, Key, Value의 개념을 사용하여 attention 점수 계산
    - 앞서 배운 Attention 개념과 유사
  - 각 토큰에 대한 가중치를 계산하여 새로운 표현 생성



## 03 Transformer

### ④ Self-Attention v.s. Attention

- 기존 Attention

- Key와 Value는 인코더(입력 문장)의 Hidden state
- Query는 디코더(출력 문장)의 Hidden state
  - 즉 영-프 번역기에서 K, V는 영어 문장의 단어, Q는 프랑스어 문장의 단어에서 유래

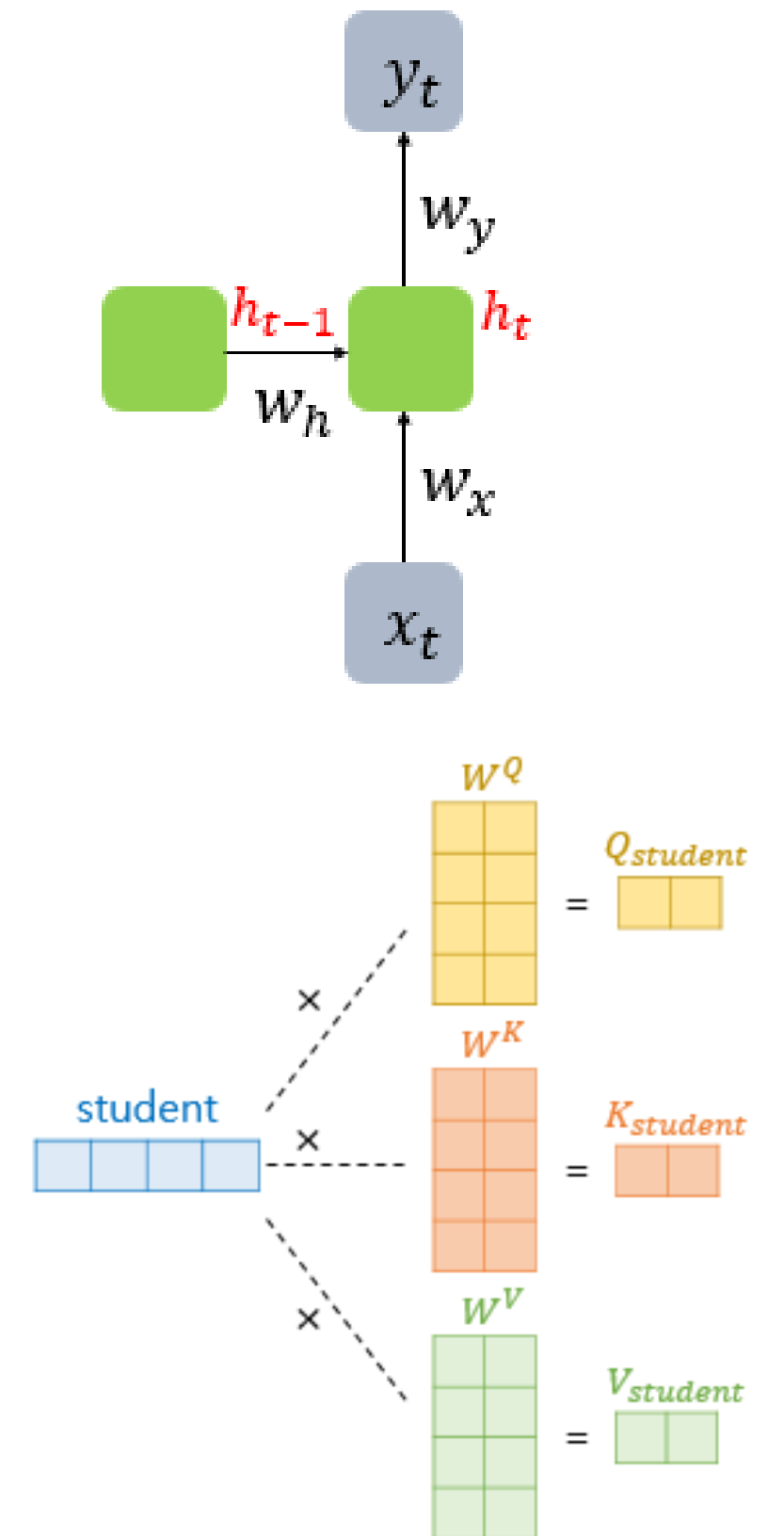
- Self attention

- Q, K, V 모두 동일한 문장에서 유래
  - 인코더의 경우 동일한 영어 문장에서 Q, K, V 모두 계산
  - 디코더에서도 프랑스어 문장에서 Q, K, V 모두 계산

## 03 Transformer

### ☑ Self Attention의 가중치

- RNN에서의 가중치:
  - 입력값을 읽어오는 가중치  $W_x$
  - 이전 시점을 받아 다음 시점으로 넘겨줄 때 연산되는 가중치  $W_h$
  - 경우에 따라 출력 값을 생성할 때 사용하는 가중치  $W_y$
- Self Attention에서의 가중치
  - 임베딩된 입력 문장에서  $Q, K, V$ 를 생성하기 위한 행렬
  - $W_Q, W_K, W_V$



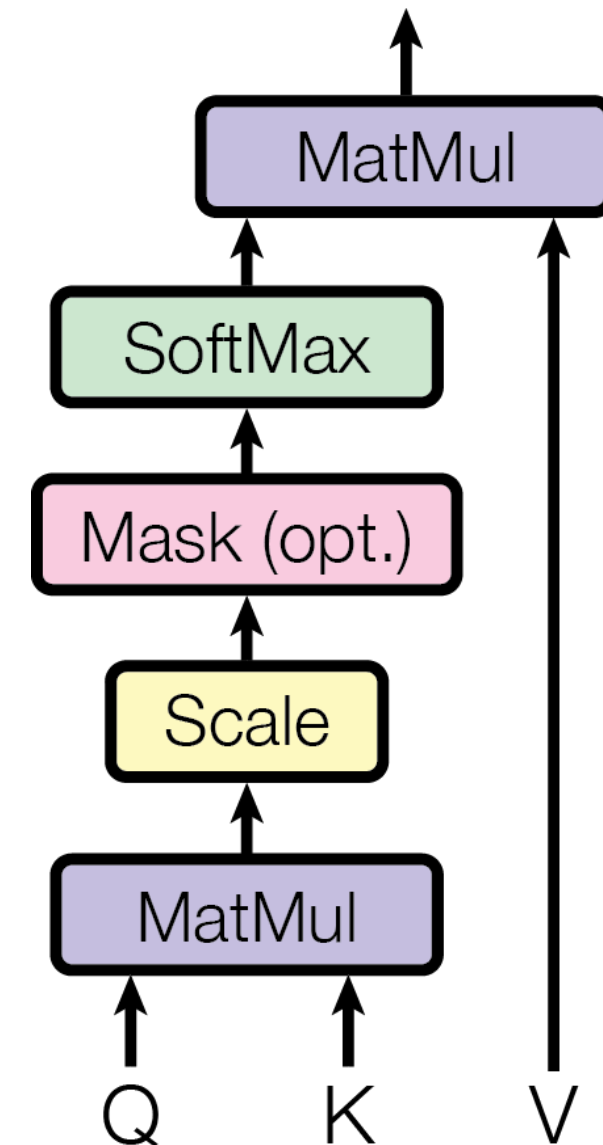


## 03 Transformer

### ☑ Self Attention

- 기존 Attention과 연산 방식에도 차이가 있음
  - Attention score
    - Q와 K의 내적은 동일하지만
    - K의 차원 수 제곱근으로 나누어 **Scaling** 진행
  - Attention value
    - 기존 Attention은 softmax의 결과와 V를 합했지만
    - Self attention에서는 이 둘을 **내적**함
  - 그런 이유로 **Scaled dot-product attention** 이라고도 함

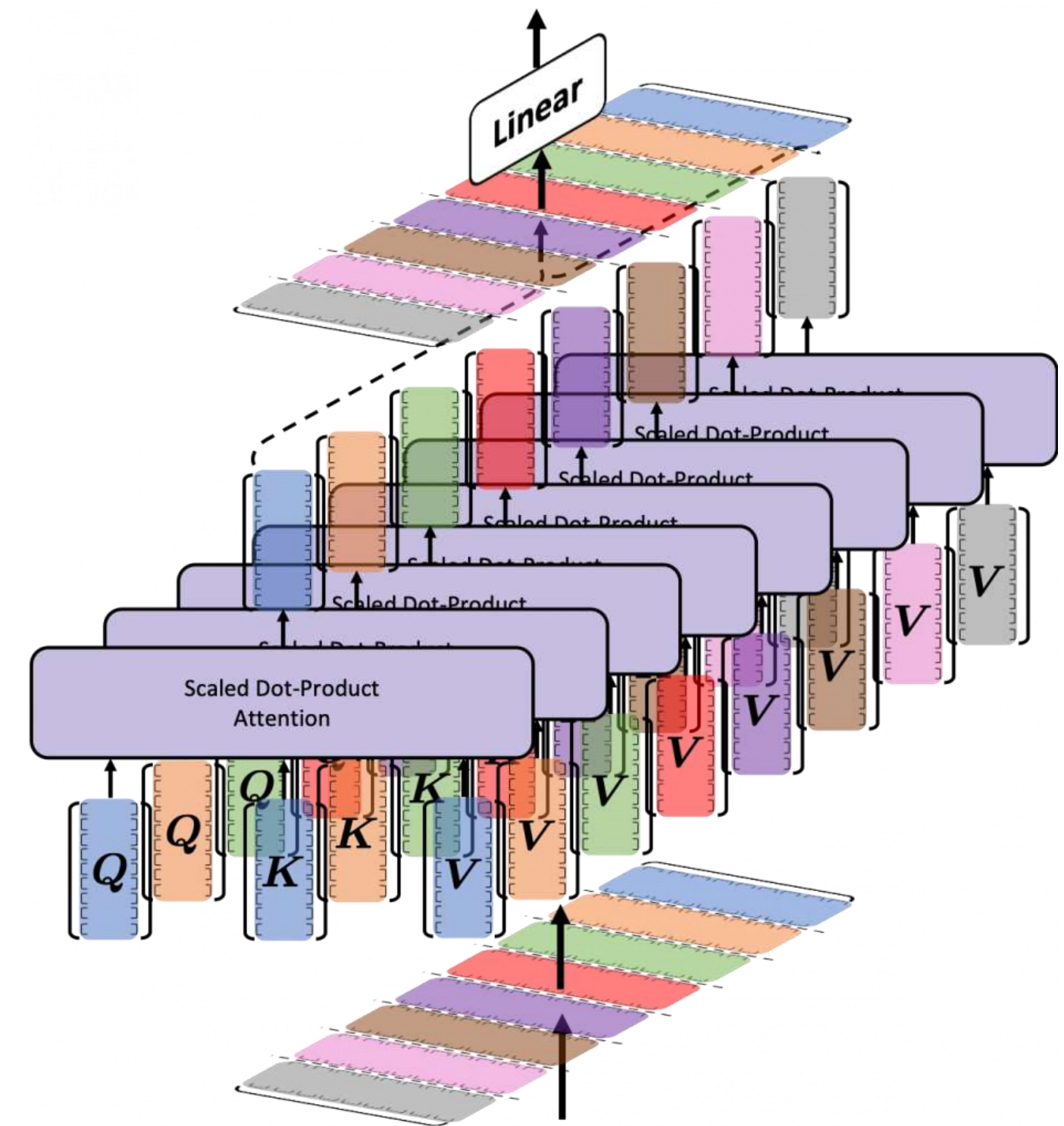
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



## 03 Transformer

### 👍 Multi-Head Attention

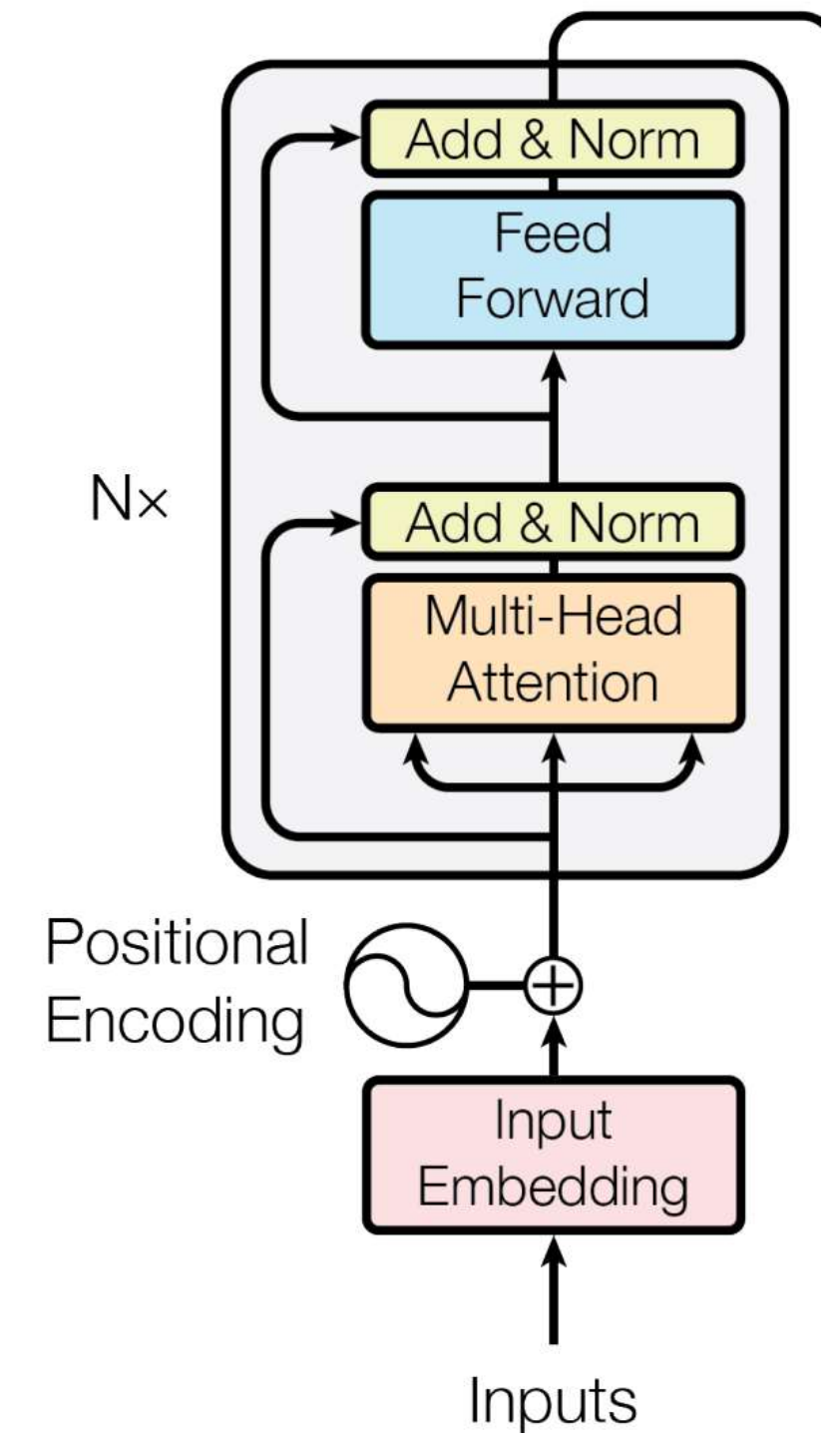
- 여러 개의 Attention 메커니즘 동시 사용
  - 병렬 연산 가능
  - 여러 모델을 동시 학습시키는 것과 유사(Ensemble)
- 각 Head는 다른 표현 공간에서의 Attention 정보를 캡처
  - 데이터에서 다양한 Feature를 추출할 수 있음
- 모든 Head의 결과를 결합하여 더 풍부한 정보 표현 생성



## 03 Transformer

### ☑ Multi-Head Attention(Encoder)

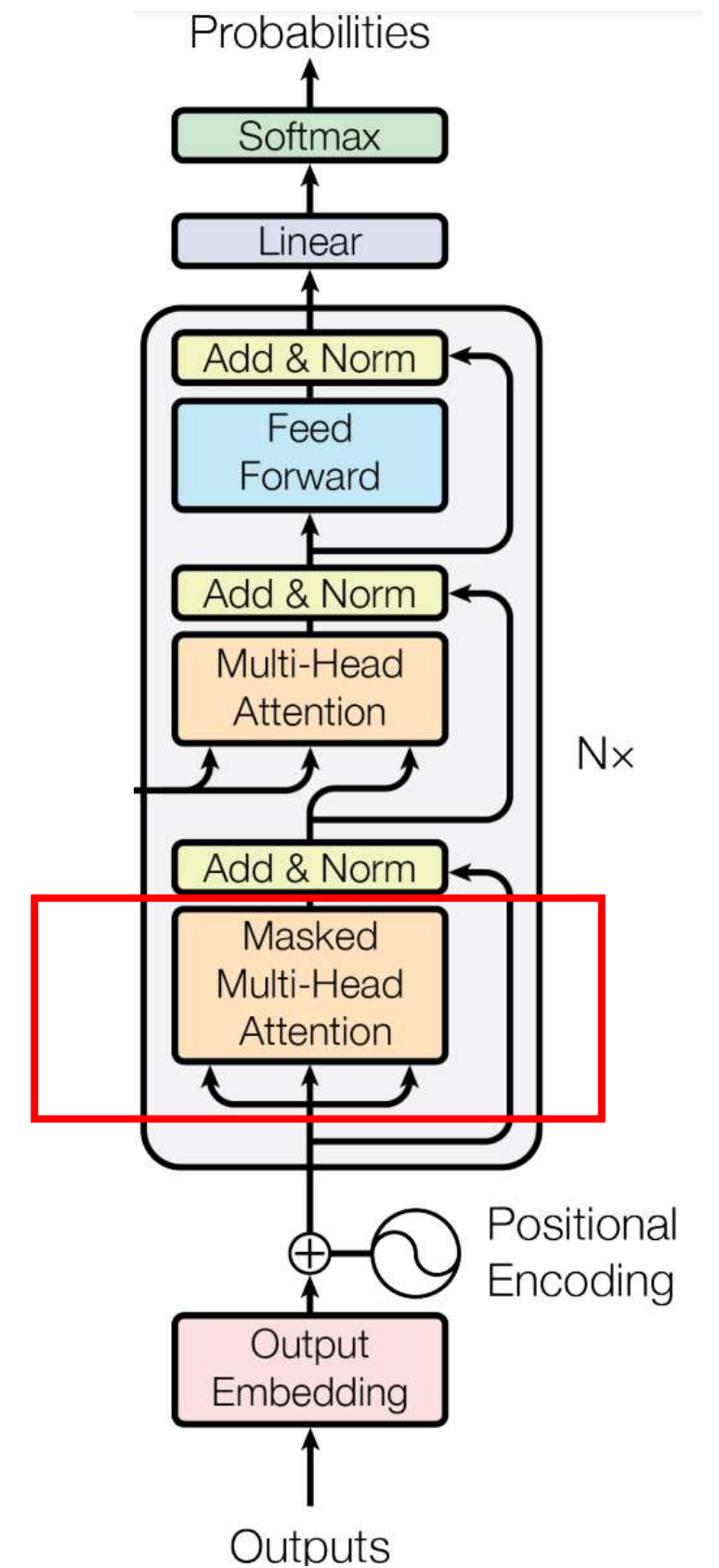
- 8개의 Head로 구성
- Self attention 연산을 통해 입력 문장의 정보를 학습
- 학습된 정보는 디코더의 Multi-Head Attention에 전달됨



## 03 Transformer

### ☑ Masked Multi-Head Attention(Decoder)

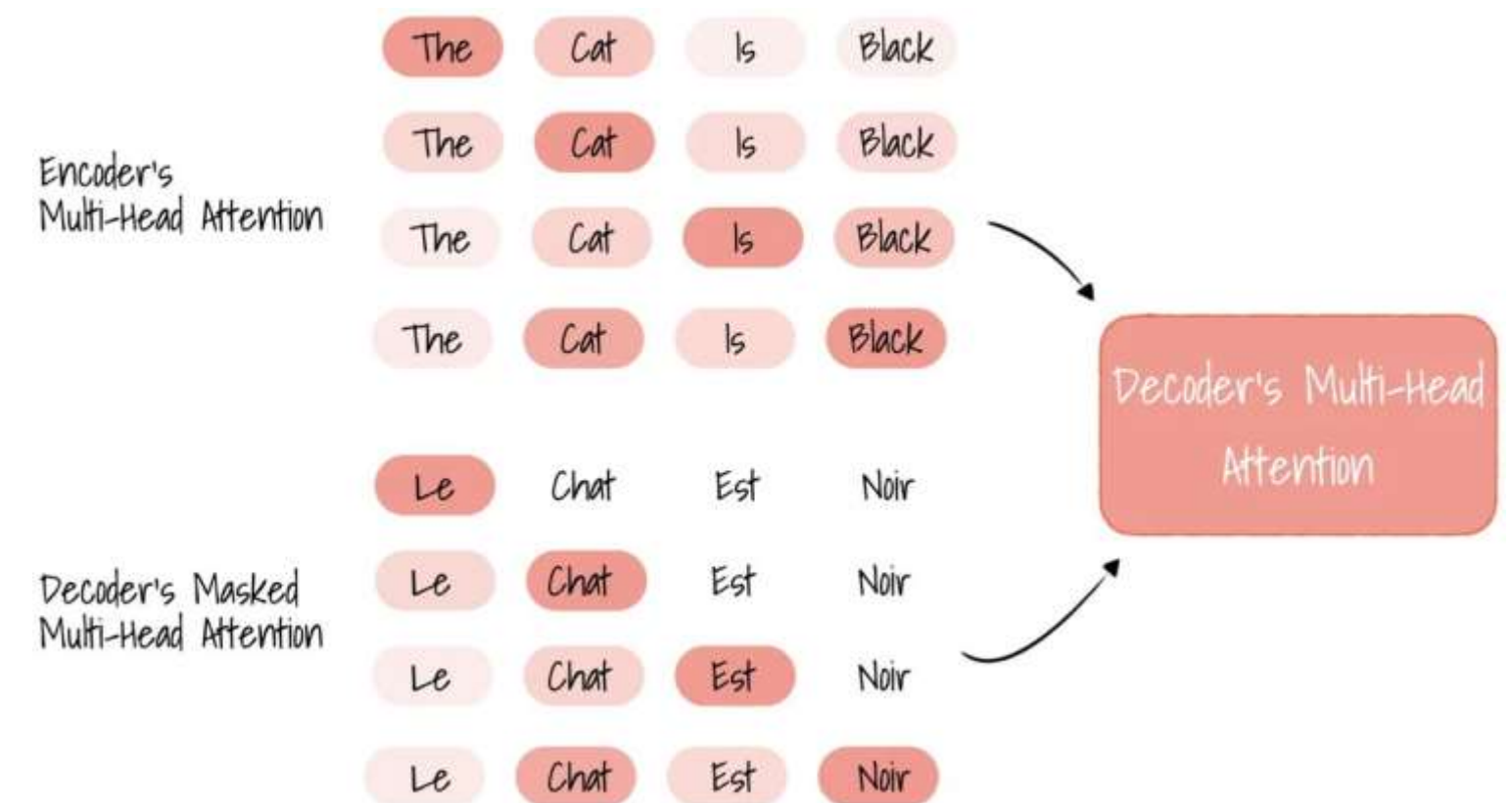
- 디코더에만 존재하는 유일한 구조
- 디코더는 LM(Language Model)의 역할을 수행해야 함
- 즉 앞 단어를 바탕으로 뒤 단어를 예측할 수 있어야 함
- 해당 기능을 구현할 수 있는 장치 필요



## 03 Transformer

### ④ Masked Multi-Head Attention(Decoder)

- 디코더에서 문장을 입력받을 경우, Self attention 계산 시 문장의 뒷부분을 의도적으로 가림
- 문장의 제일 앞 단어부터 하나씩 마스크를 해제하며 Self attention 계산
- 결과적으로 단어 간 순서 정보를 학습할 수 있음

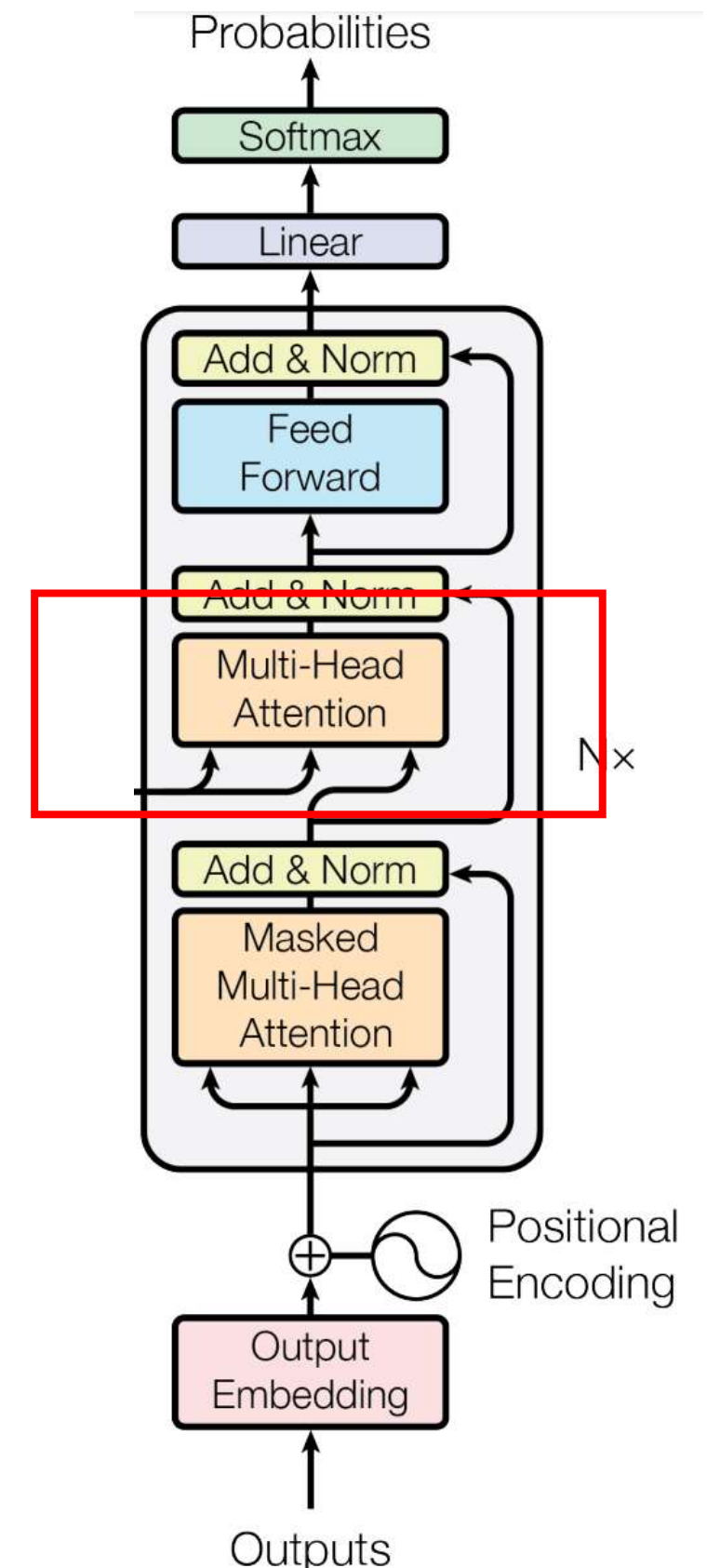




## 03 Transformer

### ☑ Multi-Head Attention(Decoder)

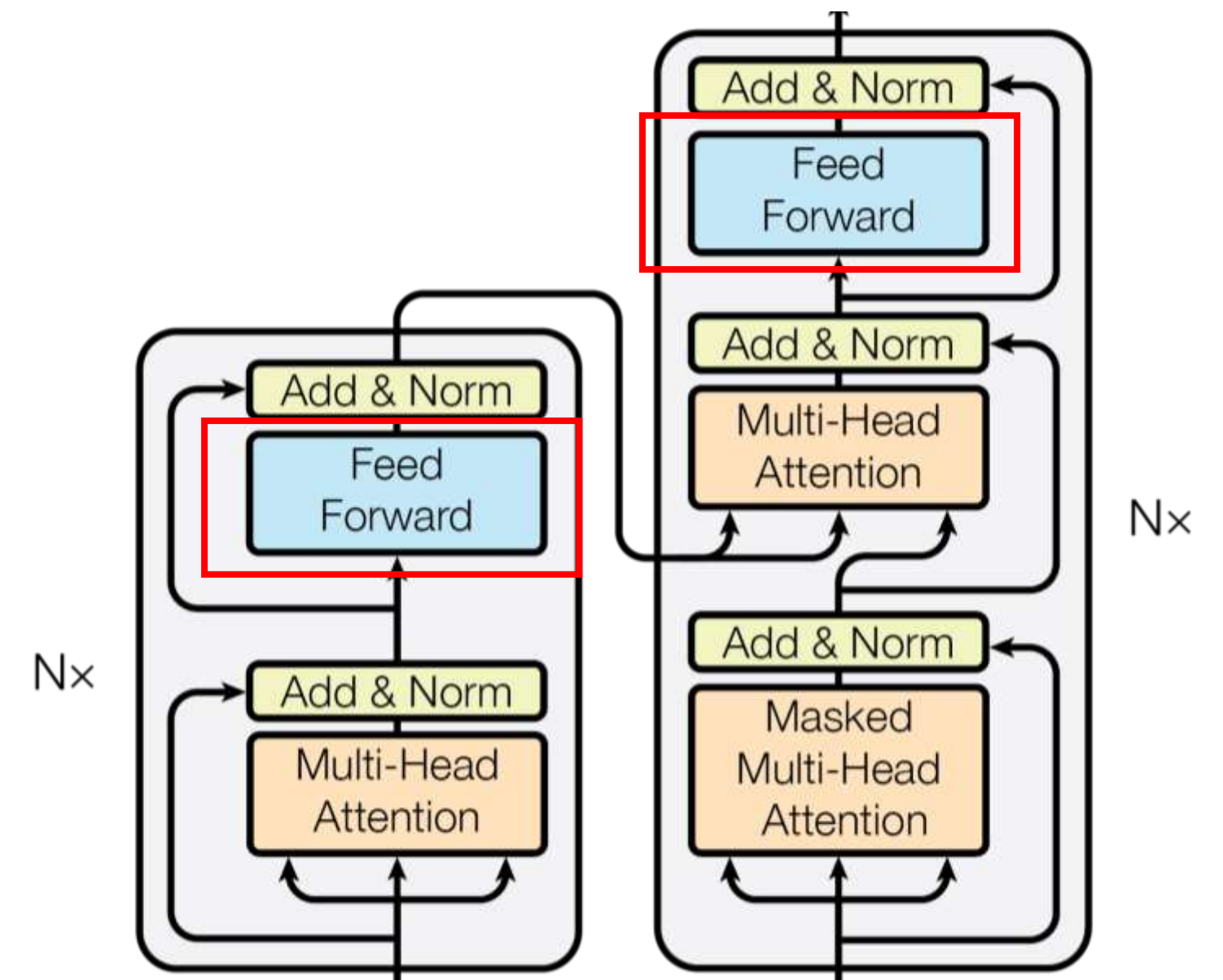
- 인코더의 Multi-head attention과 유사한 구조
- 다만, 이 과정에서  $K, V$ 는 인코더에서,  $Q$ 는 디코더에서 받음
- 사실상 이전 Attention 구조와 동일한 목적을 위해 연산
  - 디코더의 단어 생성( $Q$ )을 위해
  - 인코더의 Hidden state( $K, V$ )를 참조



## 03 Transformer

### ④ Feed-forward Networks

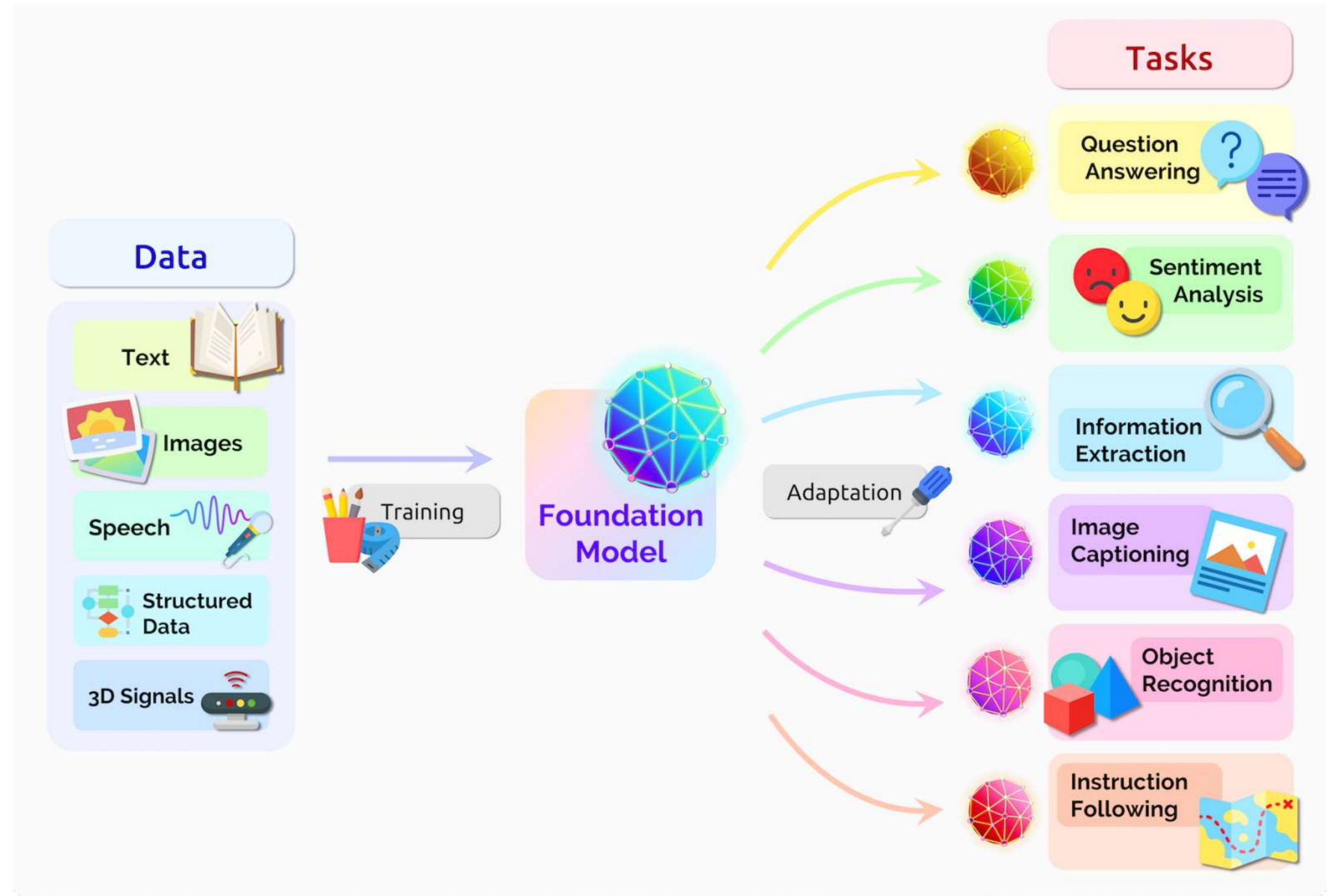
- 각 Self attention 후에 적용되는 신경망
- 동일한 구조가 **각 토큰에 독립적**으로 적용
  - **Position-wise** FFNN이라고도 함
  - 한 단어에 대한 의미 공간(Embedding vector)을 연산
  - **단어의 의미 파악**
- 비선형성 추가(ReLU) 및 토큰 표현의 변환



## 03 Transformer

### ④ Transformer의 성과

- NLP 분야의 급속한 성장
- 다양한 분야에서의 우수한 성능 입증
  - Vision, audio, RL
- 딥러닝 모델의 상용화 촉진





04

BERT

## 04 BERT

### ④ BERT의 기본 개념

- Bidirectional Encoder Representation from Transformers
- Transformer의 인코더 구조를 기반으로 한 모델
  - 양방향으로 문맥을 고려하여 토큰을 인코딩
  - 주어진 텍스트에서 일부 토큰을 마스킹하고 이를 예측하는 방식으로 사전 학습
- QA, 문장 분류 등의 다양한 NLP 문제에서 SOTA 달성

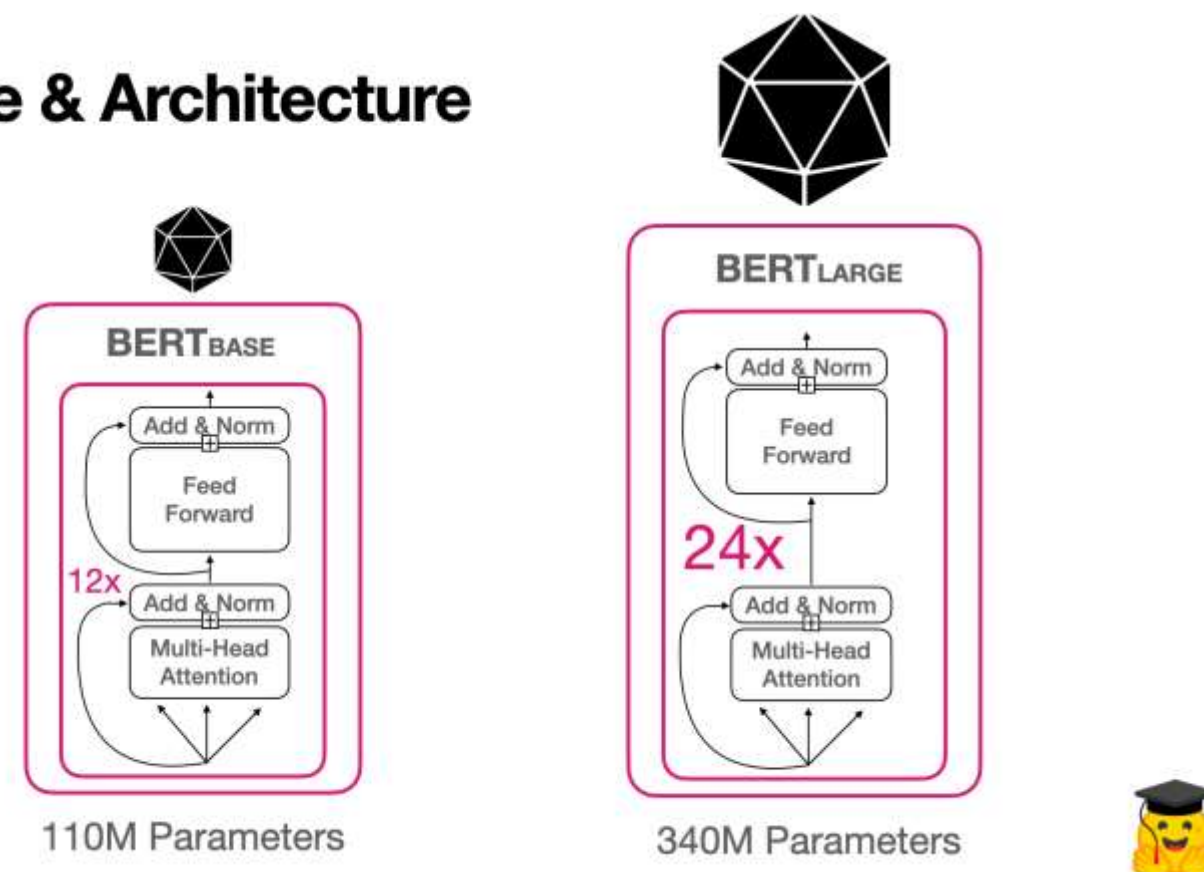


## 04 BERT

### 📌 BERT 구조

- Transformer의 인코더 구조를 기반
  - Base: 12층
  - Large: 24층
  - Multi-head Self-Attention, Feed-forward NN 등 Transformer와 동일

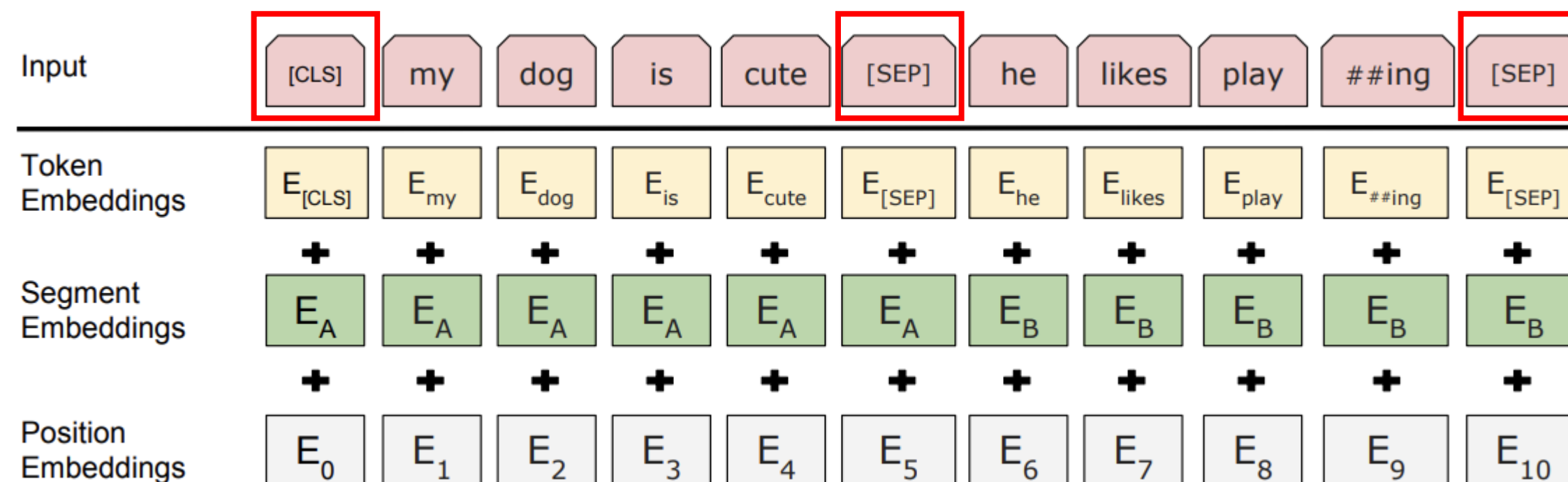
#### BERT Size & Architecture



## 04 BERT

### ☑ BERT 구조

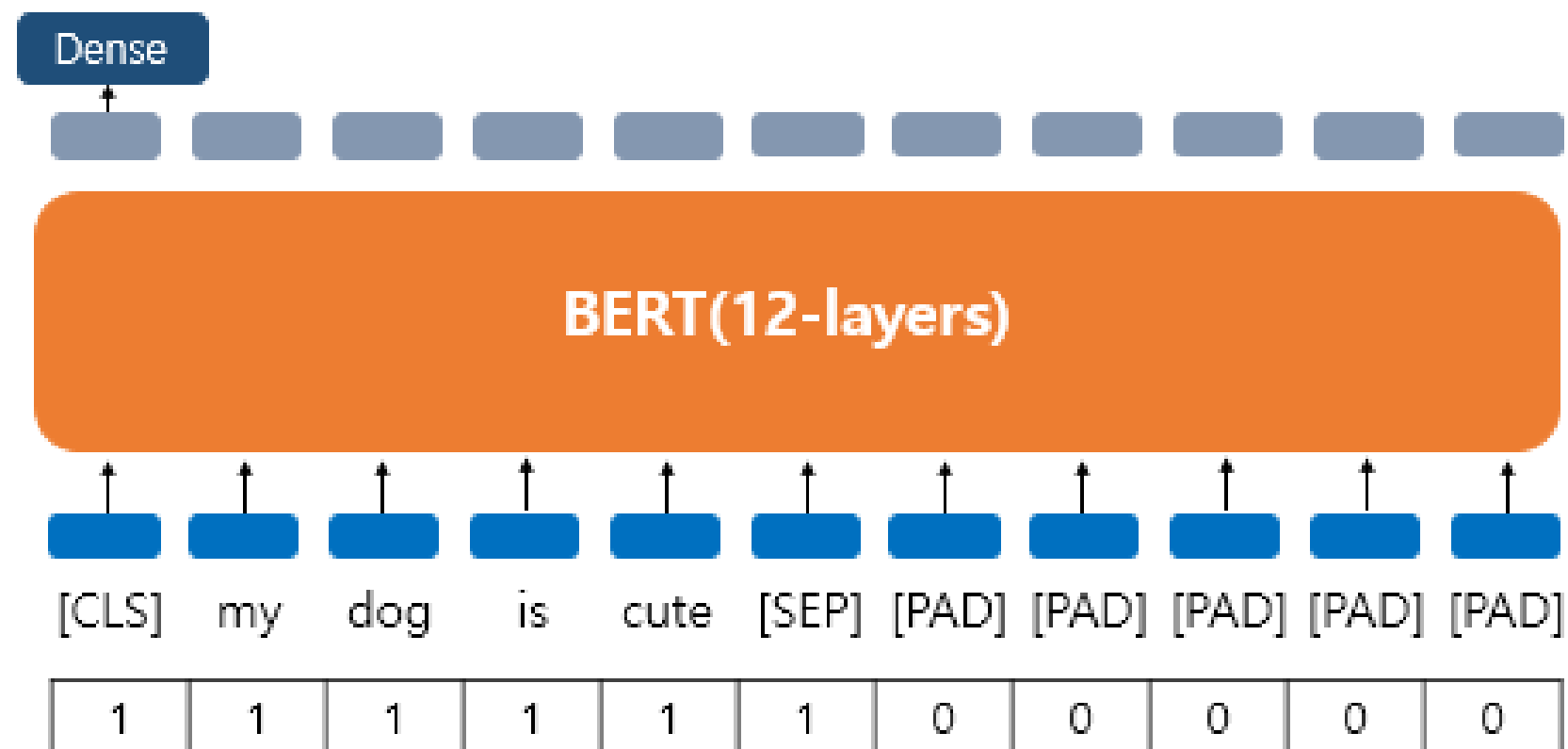
- 모델의 입력 구조에 다양한 Embedding 사용
  - Position embedding: Positional encoding과 유사하게 단어 위치를 태깅
  - Segment embedding: 문장과 문장을 구분하는 값(A or B)
    - 두 개의 문장이 결합되어 입력
  - Token embedding: 일반적인 토큰 임베딩 과정



## 04 BERT

### 👉 BERT 구조

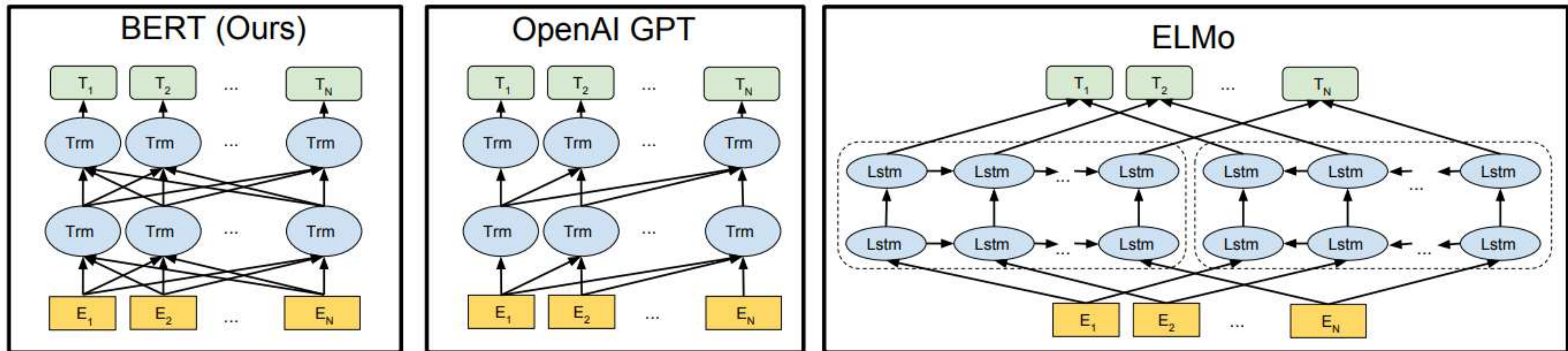
- Attention mask 사용
  - Attention mask는 Transformer의 디코더에서만 사용됨
  - BERT의 경우 문장을 생성하는 것은 아니지만,
  - 패딩 토큰이 불필요하게 연산에 투입되는 것을 방지하기 위해 마스크를 사용



## 04 BERT

### ☑ BERT의 특징 - 양방향 문맥 인식

- 이전 모델들은 주로 단방향(왼쪽에서 오른쪽 또는 그 반대) 문맥만 고려
- BERT는 양쪽 방향의 문맥을 동시에 고려하여 더 정확한 토큰 표현 생성



## 04 BERT

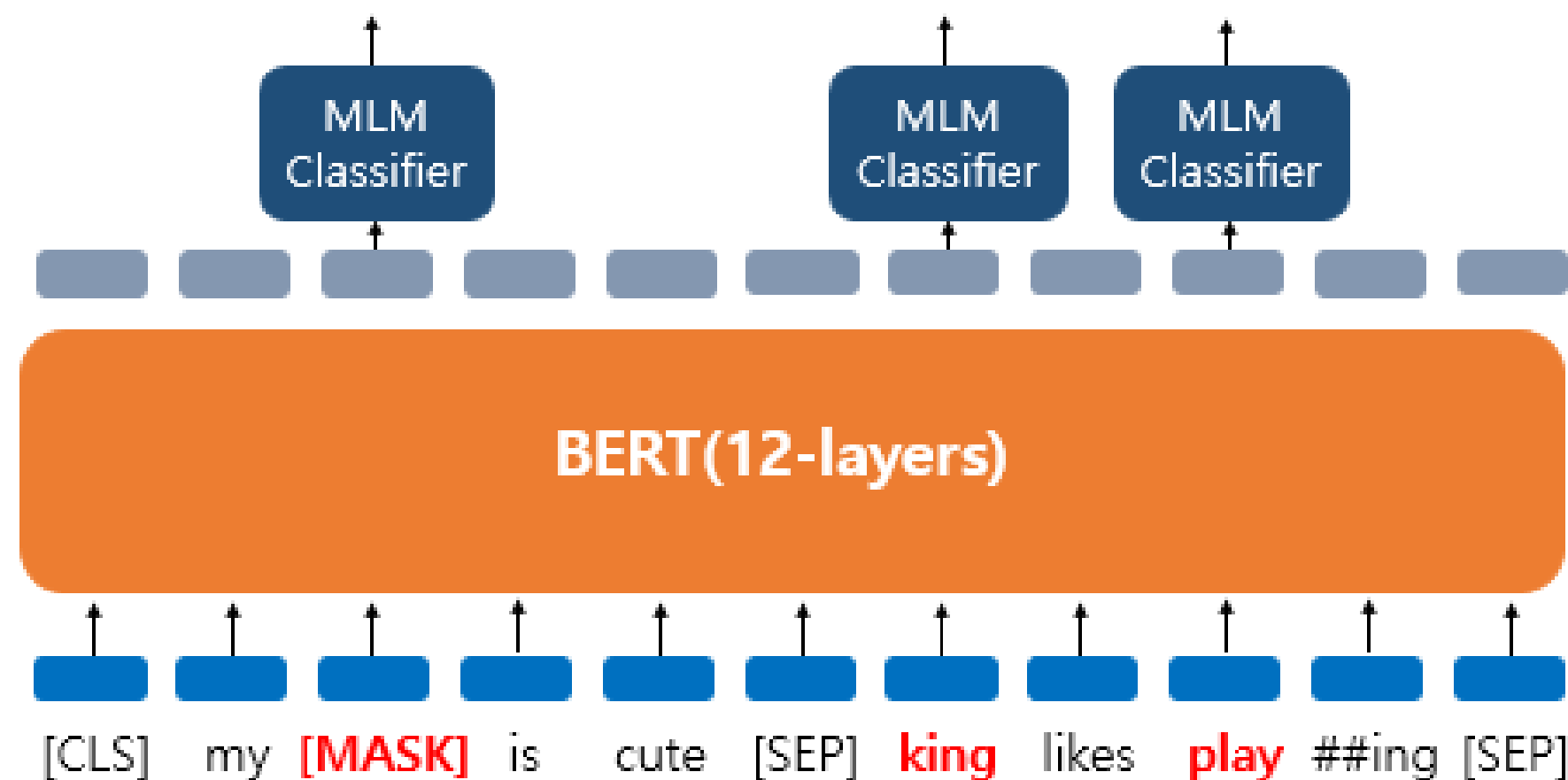
### ④ BERT의 특징 – 사전 학습(Pretraining)

- BERT에는 무한한 양의 지도학습 데이터가 투입
  - 지도학습의 형태로 비지도학습을 수행
  - 즉 레이블이 없는 데이터를 이용하여 지도학습을 수행
- 두 가지 방식의 사전 학습이 진행
  - MLM(Masked Language Model)
  - NSP(Next Sentence Prediction)

## 04 BERT

### ☑ BERT의 사전학습 - MLM(Masked Language Modelling)

- 일부(15%) 토큰을 임의로 마스킹하고 해당 토큰을 예측하는 방식으로 학습
- 이 과정을 통해 모델은 문맥을 기반으로 한 토큰의 의미를 깊게 이해

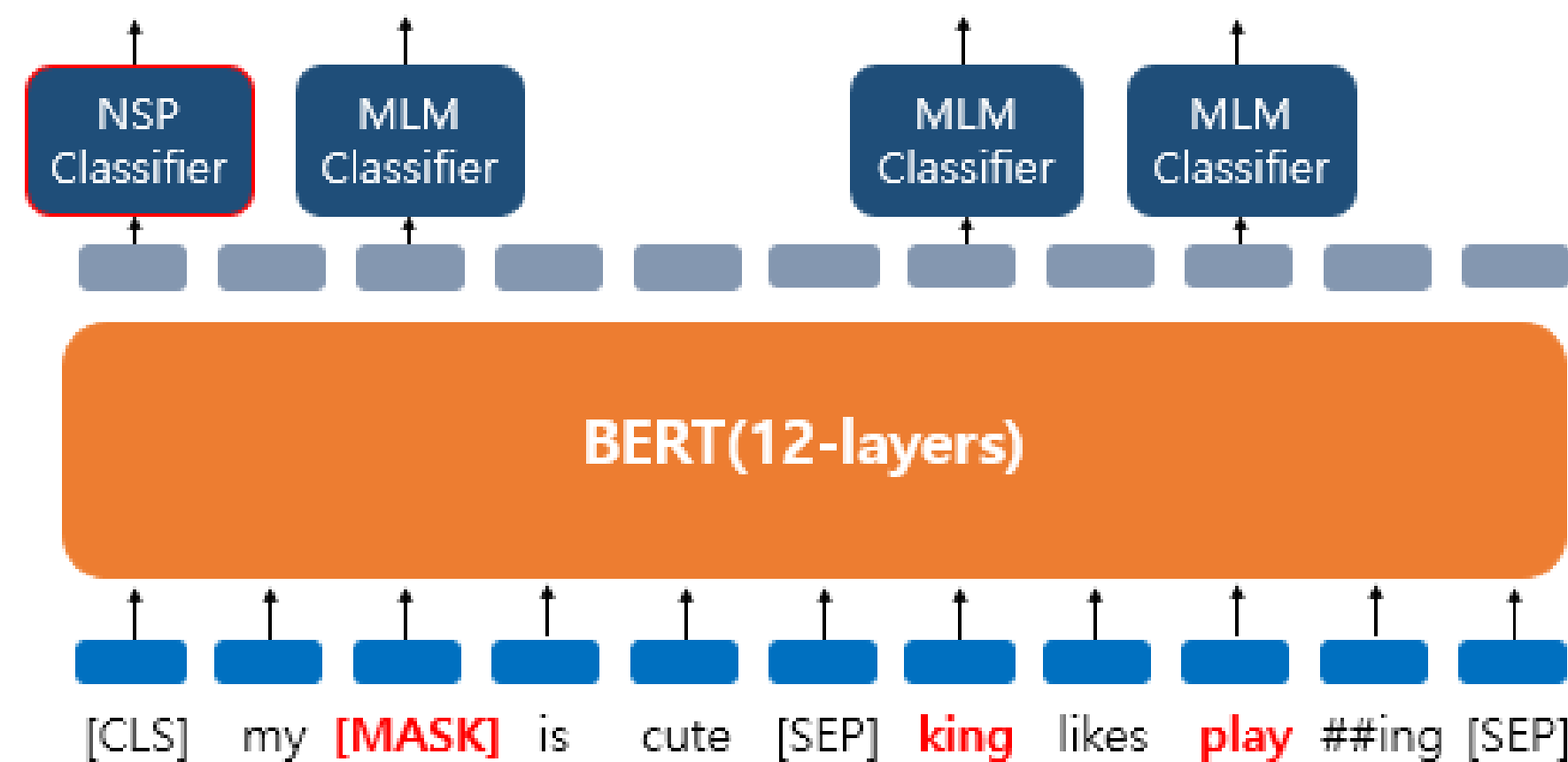




## 04 BERT

### ☑ BERT의 사전학습 – NSP(Next Sentence Prediction)

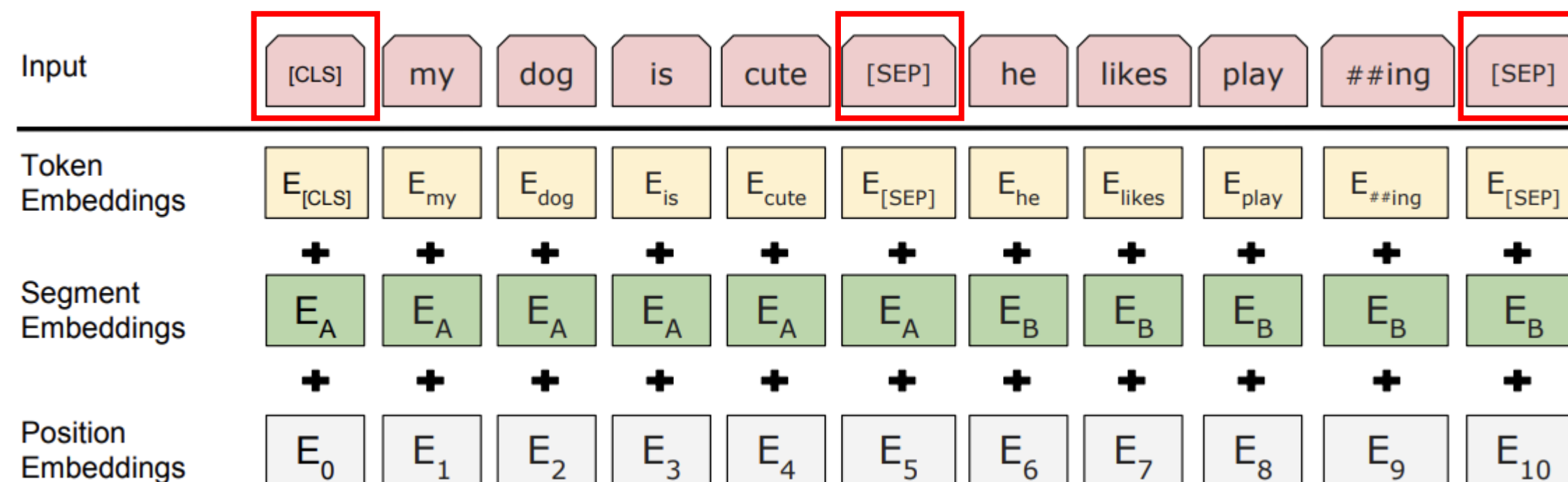
- 두 개의 문장이 붙은 상태로 모델에 입력됨
  - 문장을 구분하기 위해 [SEP] 토큰을 구분자로 사용
- 이진 분류를 통해, 두 문장의 연속성 여부를 학습함
  - [CLS]토큰의 위치에서 이진 분류 문제를 해결



## 04 BERT

### ☑ BERT의 사전학습 – NSP(Next Sentence Prediction)

- Segment embedding: 문장과 문장을 구분하는 값(A or B)
  - [SEP]토큰과 함께 사전학습 중 NSP 문제를 풀기 위한 장치
- 두 특수 토큰은 모델에 입력되기 전 전처리 단계에서 부착됨



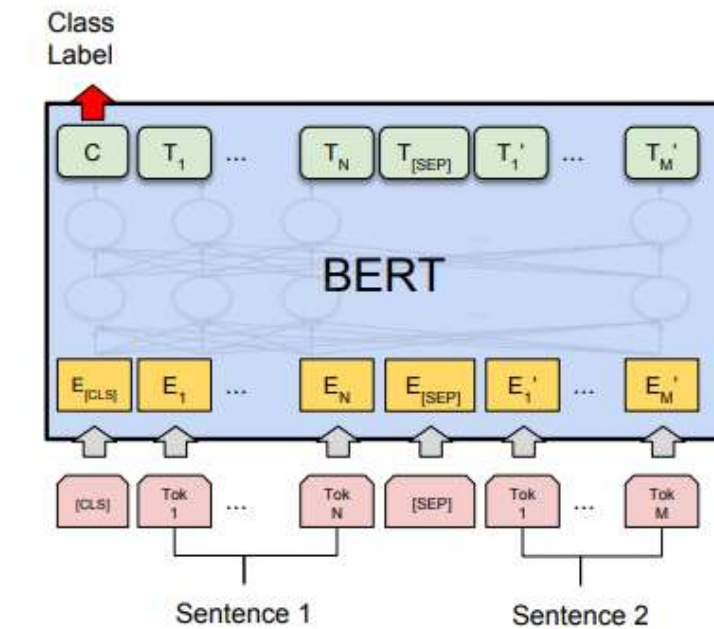
## 04 BERT

### ☑ BERT의 특징 - 전이 학습(Transfer learning and Fine-tuning)

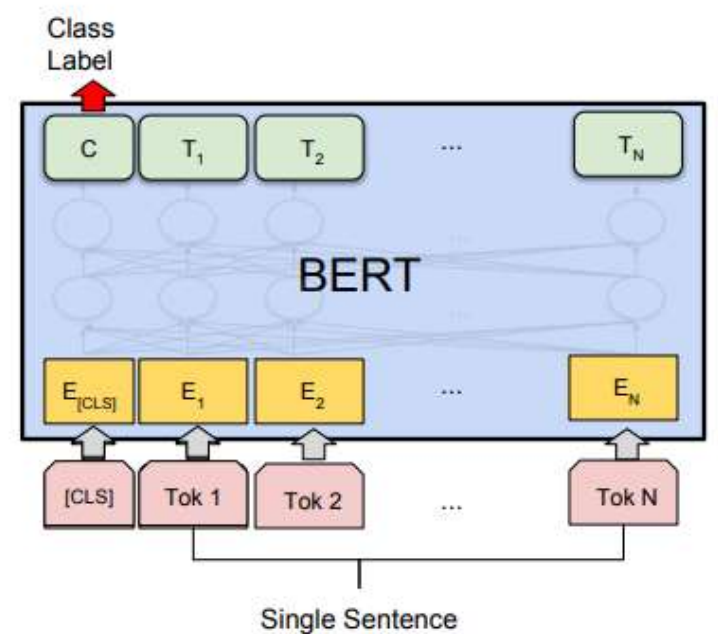
- BERT는 큰 텍스트 코퍼스에서 사전 학습 후, 특정 작업에 미세 조정 가능
  - Wikipedia(2.5B), BooksCorpus(8M)

• 11가지 NLP task에서 SOTA 달성

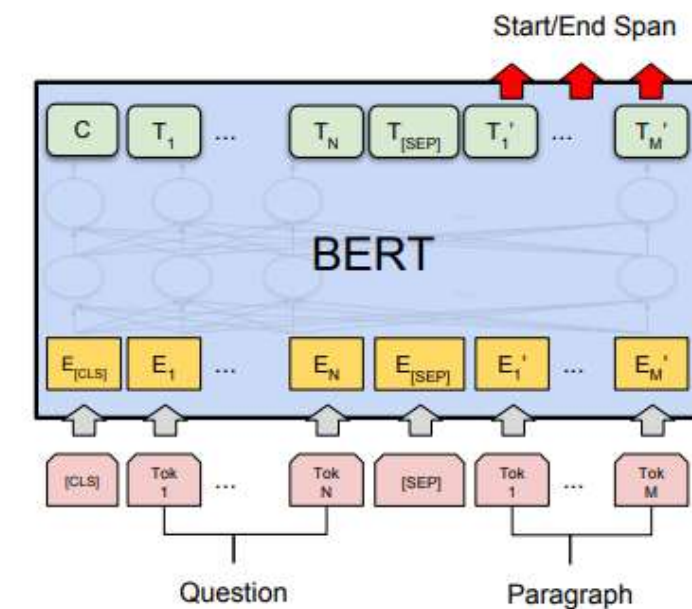
• 각 Task마다 모델 말단의 구조만 다름



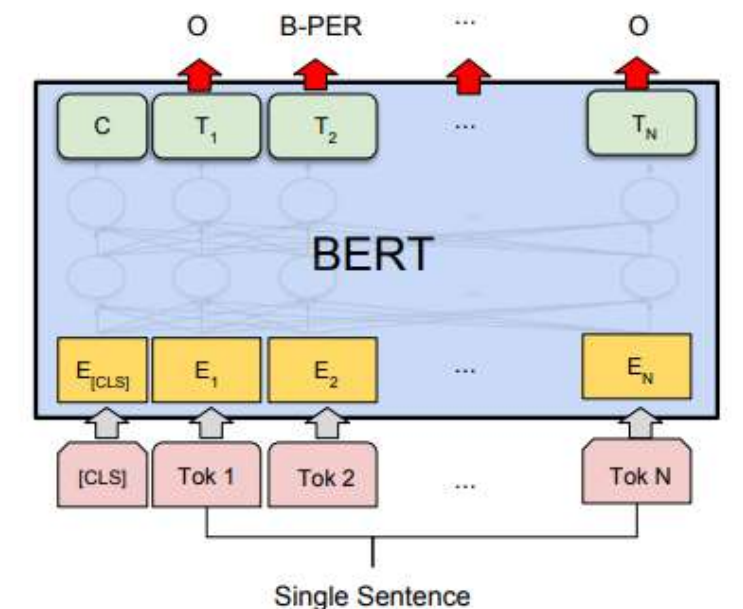
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(b) Single Sentence Classification Tasks:  
SST-2, CoLA



(c) Question Answering Tasks:  
SQuAD v1.1



(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER