

Paper review

Transformer

카피바라팀

박현아, 배누리, 김호정, 전사영

2024.11.19

목차



01

Chapter 1

자연어 처리
딥러닝 모델 변천 과정

02

Chapter 2

LSTM의 한계 및
Seq2Seq

03

Chapter 3

Attention

04

Chapter 4

Transformer

05

Chapter 5

코드 구현

Chapter 1. 자연어 처리 딥러닝 모델 변천 과정

LSTM (1997)

- Long Short-Term Memory
- RNN에서 장기적 기억을 더 잘하도록 변환된 신경망



Seq2Seq (2014)

- 시퀀스 입력을 통해 다른 시퀀스 출력을 얻도록 고안된 모델
- 기계번역에 주로 사용되는 모델



Attention (2015)

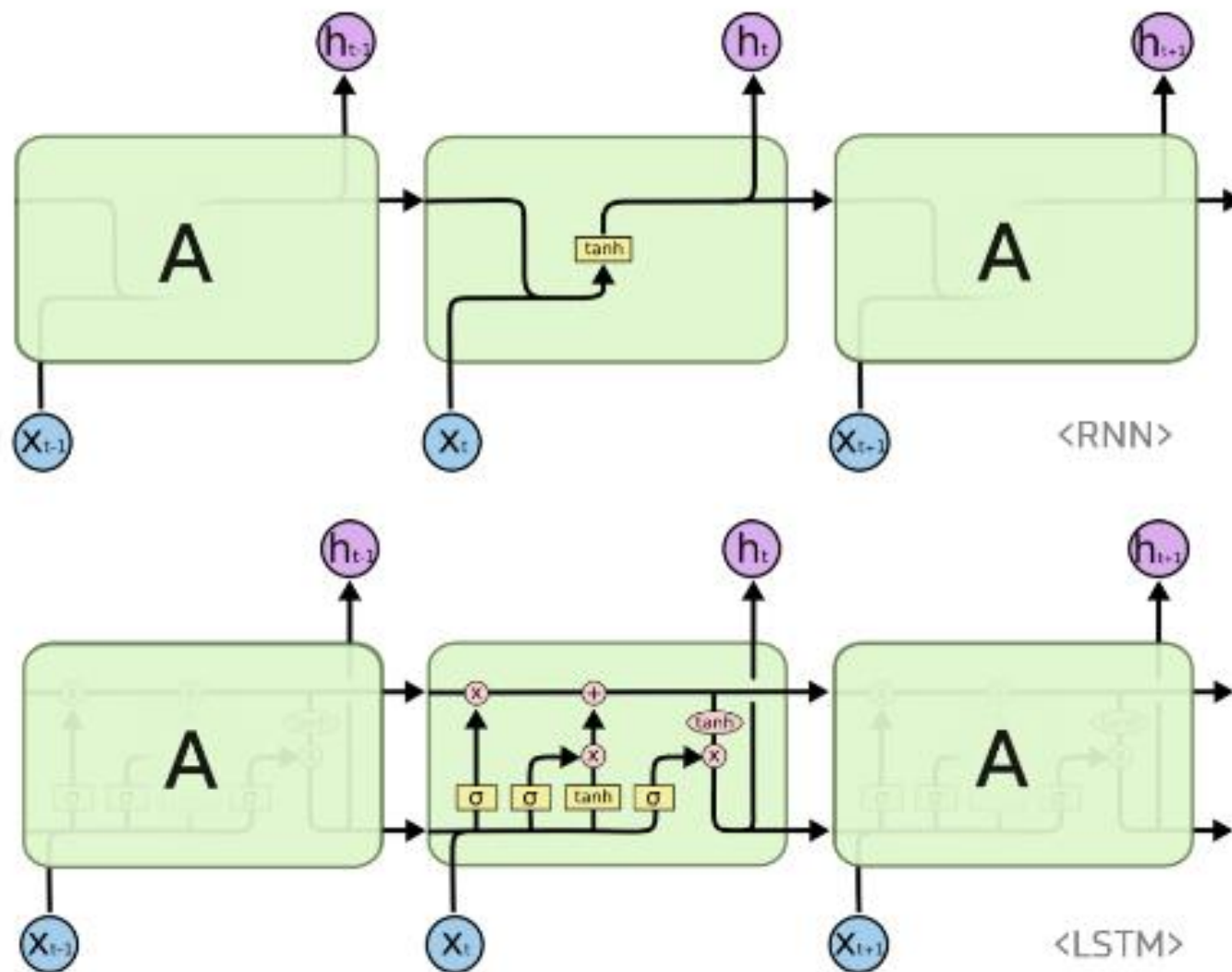
- Bottleneck 문제를 보완
- 전체 입력 시퀀스에서 '주의'를 기울여야 하는 부분에 초점을 맞춰 학습



Transformer (2017)

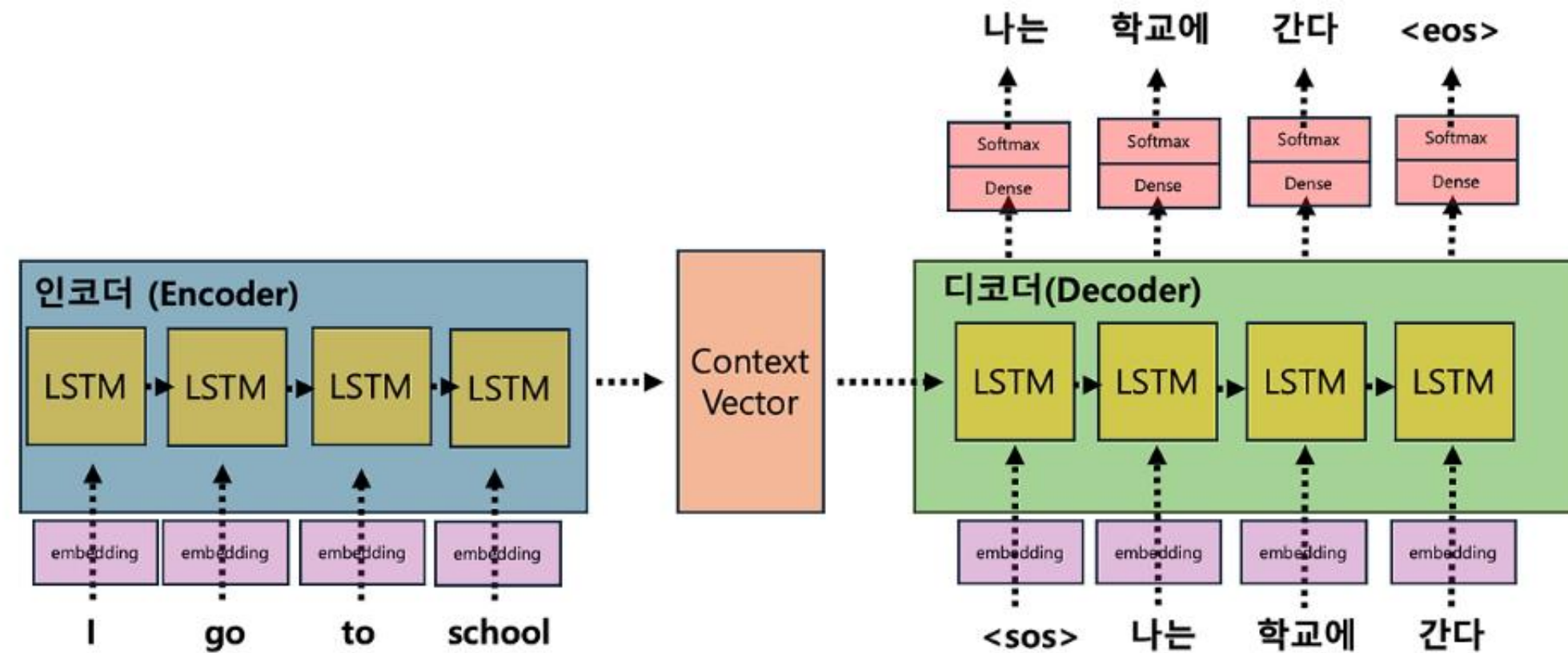
- Attention을 활용하여 구성한 Encoder-Decoder 모델
- BERT, GPT의 기반 모델

Chapter 2. LSTM의 한계



- 단일 시퀀스를 입력 받고, 해당 시퀀스를 처리하여 결과를 내는 구조임.
- 기본적인 LSTM 구조에서는 입력 시퀀스와 출력 시퀀스가 동일한 길이를 가져야 처리할 수 있으며 한 번에 하나의 값을 출력하는 방식.
- 시퀀스 데이터를 처리하는 데 매우 효과적이지만, 입력과 출력 길이가 크게 다른 경우에는 모델이 충분히 모든 정보를 반영하지 못할 수 있음.
- 입력 시퀀스가 길거나 복잡할 경우, 각 타임 스텝의 정보를 잊어버리거나 학습에 어려움을 겪을 수 있음.

Chapter 2. Seq2Seq



- Seq2Seq는 두 개의 LSTM 네트워크로 구성되며, 인코더(Encoder)와 디코더(Decoder)로 나누어 시퀀스 변환 문제를 처리함.
- 인코더(Encoder): 입력 문장의 모든 단어들을 순차적으로 입력받은 후 모든 단어 정보들을 압축해서 하나의 벡터로 만들어 디코더로 전송함.
- 디코더(Decoder): 디코더는 컨텍스트 벡터를 받아서 번역된 단어를 한 개씩 순차적으로 출력함.
- 한계 : 컨텍스트 벡터의 고정적인 크기로 Bottleneck(병목현상)이 일어날 수 있음.


Chapter 3. Attention

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

- Query : 특정 단어(토큰)의 정보를 요청하는 역할.
- Key : 각 단어의 특징을 나타냄. 쿼리가 다른 단어와 관련성이 있는지를 판단할 때, 키를 이용함.
- Value: 그 단어의 실제 정보를 담고 있으며, 어텐션 점수에 따라 최종 출력에 반영되는 정보.
- Query와 Key 사이의 유사도를 측정하여 각 입력이 얼마나 중요한지를 계산함.
- 이 유사도는 주로 내적으로 계산됨. 즉, Query와 Key의 내적 결과가 높을수록 해당 입력이 더 중요한 것으로 간주됨.

Chapter 3. Attention

예시 : 나는 학생 입니다

나는 
Q



I



am



a



student

K

W_q

W_k

W_v

Chapter 3. Attention

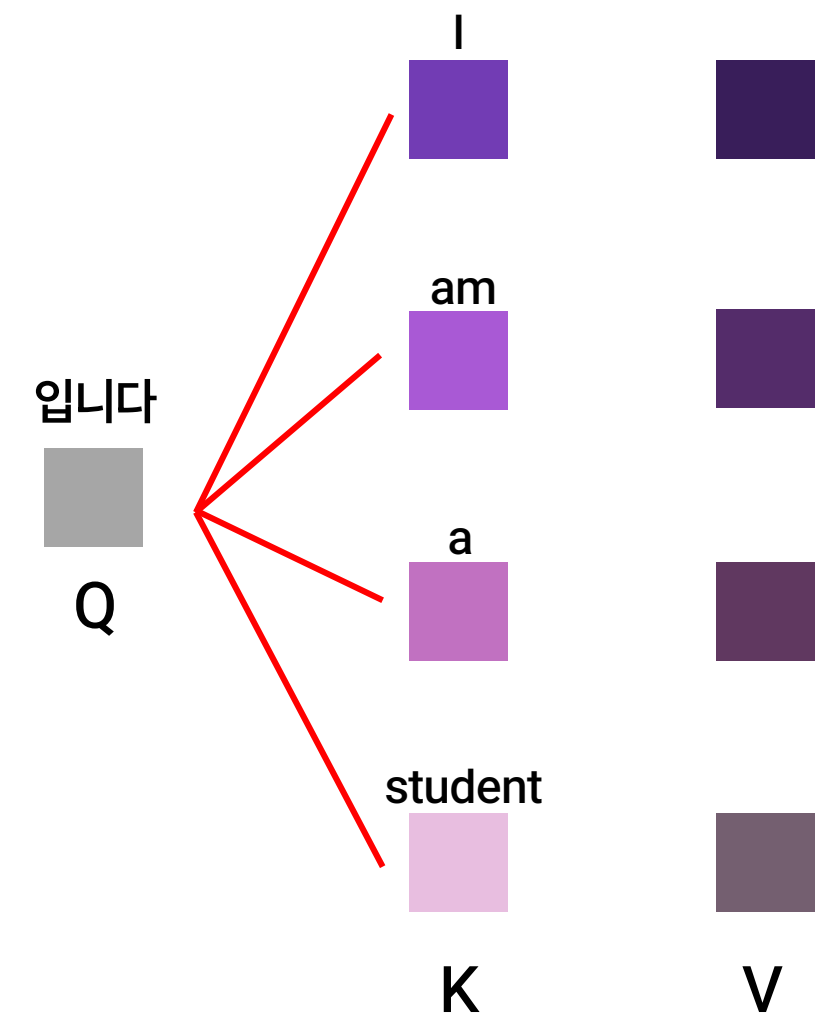
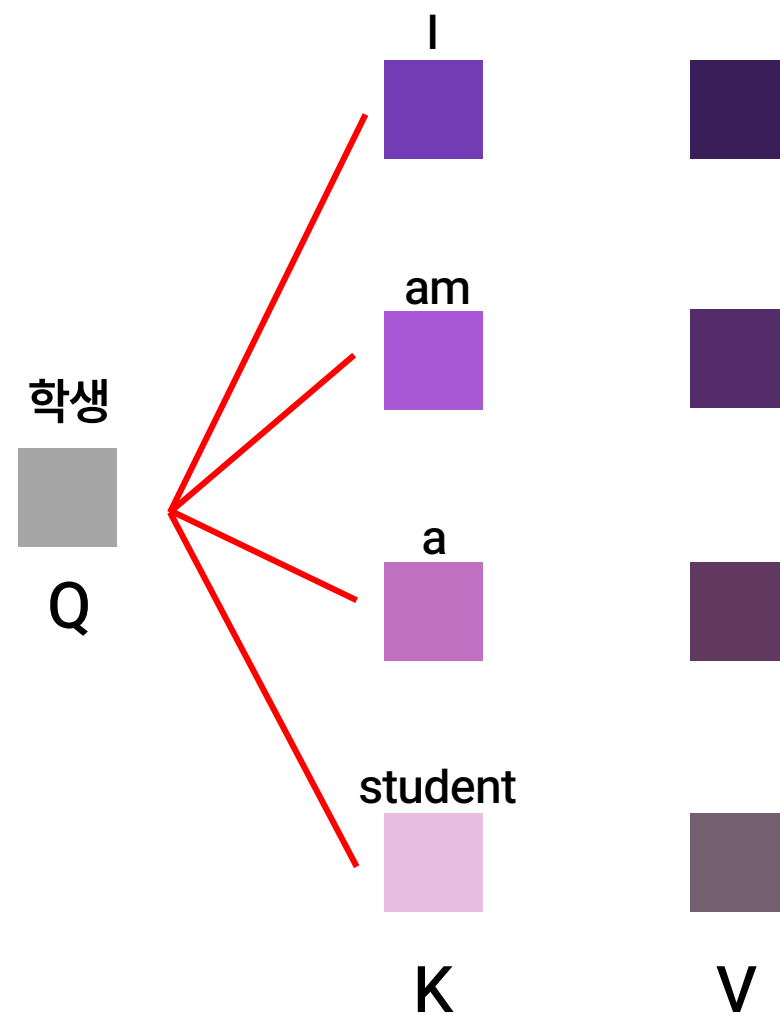
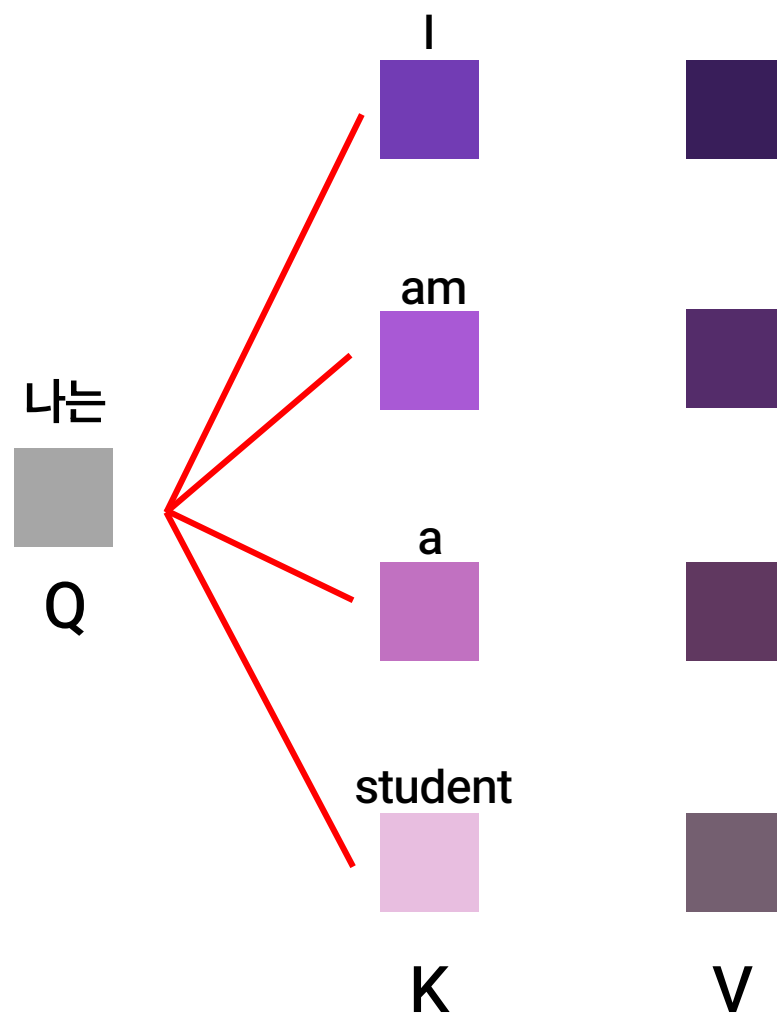
예시 : 나는 학생 입니다



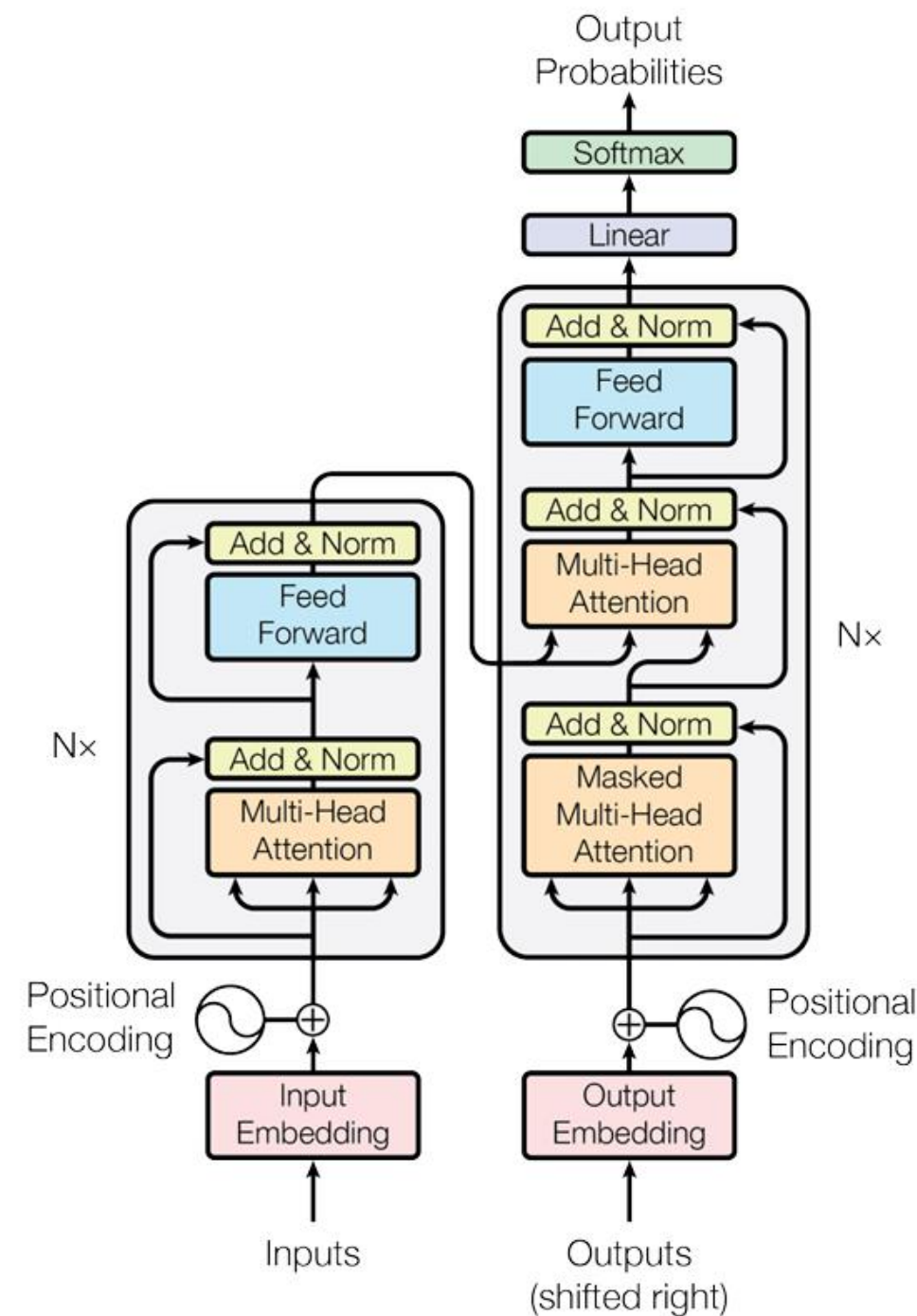
$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Chapter 3. Attention

예시 : 나는 학생 입니다



Chapter 4. Transformer

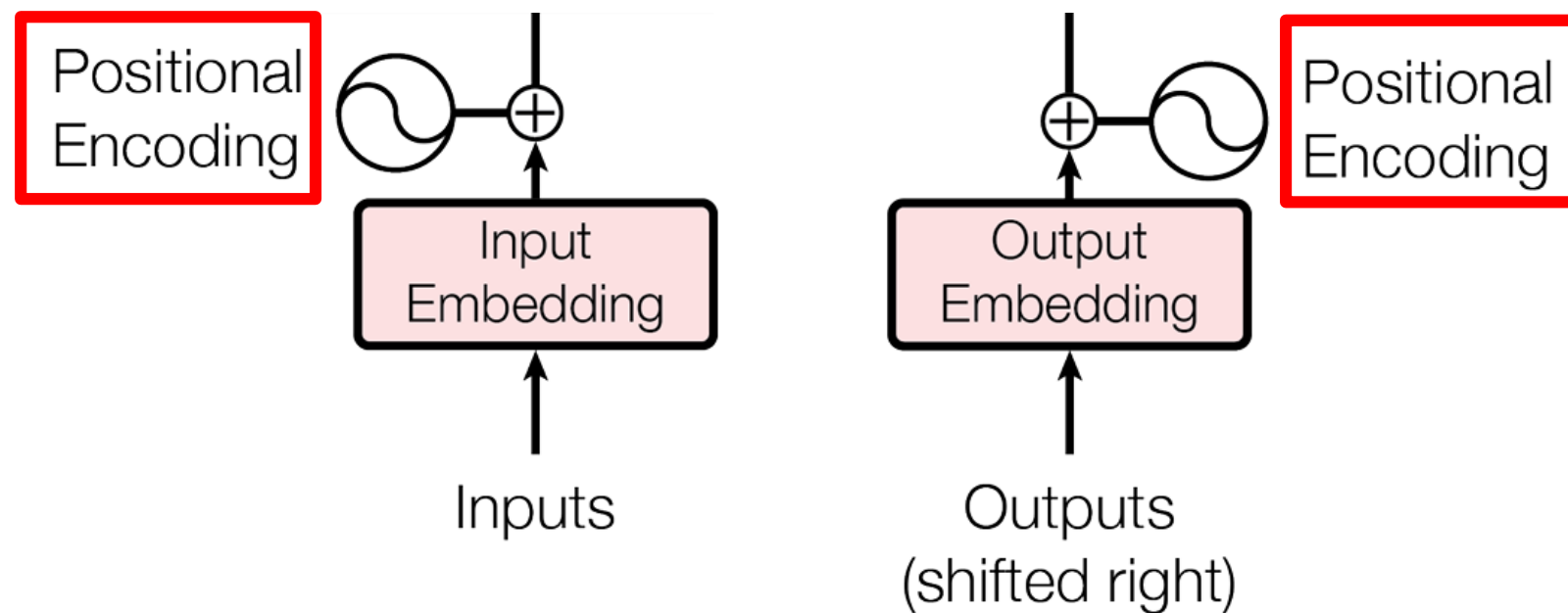


동작과정

- 입력 시퀀스 처리: 입력된 단어에 포지셔널 인코딩이 더해져 단어의 순서를 고려
- 인코더 변환: 입력 시퀀스는 여러 층의 Self-Attention과 Feed-Forward 네트워크를 통과하며 정보가 변환
- 디코더 활용: 변환된 인코더 출력이 디코더로 전달되고, 디코더는 이를 바탕으로 출력 시퀀스를 생성
- 최종 출력 생성: 디코더에서 Self-Attention과 Encoder-Decoder Attention 레이어를 거쳐 최종 출력값이 만들어짐

Chapter 4. Transformer

Positional Encoding

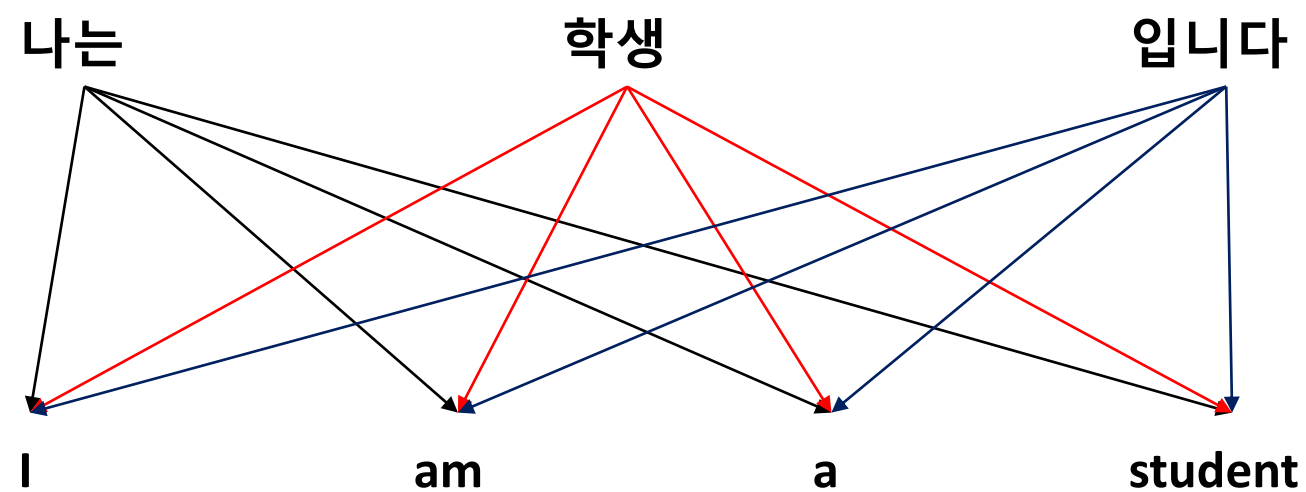


$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

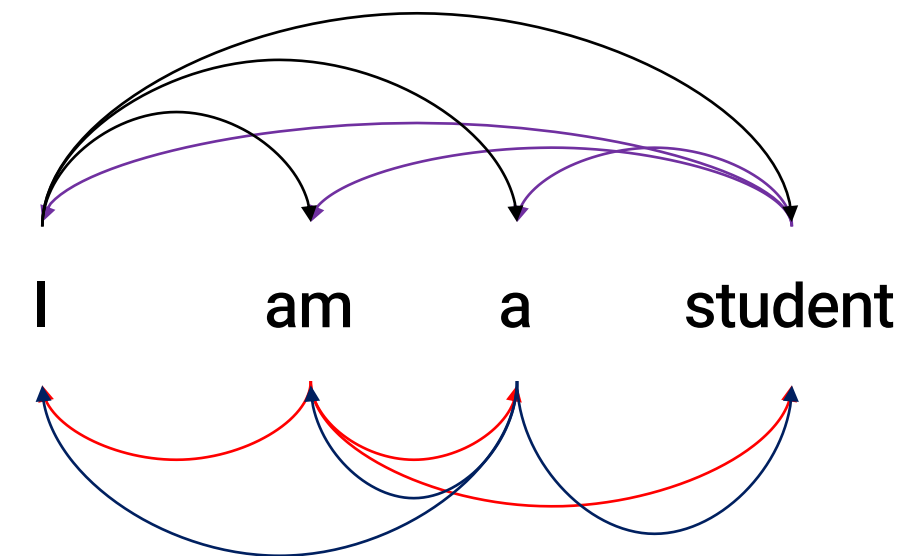
- 트랜스포머는 단어 순서를 직접 학습하지 못하기 때문에, 위치 정보를 벡터로 표현하는 Positional Encoding을 사용
- 사인/코사인 함수로 생성된 값들을 입력 임베딩 벡터에 추가하여 각 단어의 위치를 반영
- 이를 통해 모델이 문장 내 단어들의 상대적 위치를 이해하고, 문장의 의미를 보다 효과적으로 해석

Chapter 4. Transformer

Self Attention



<Attention>

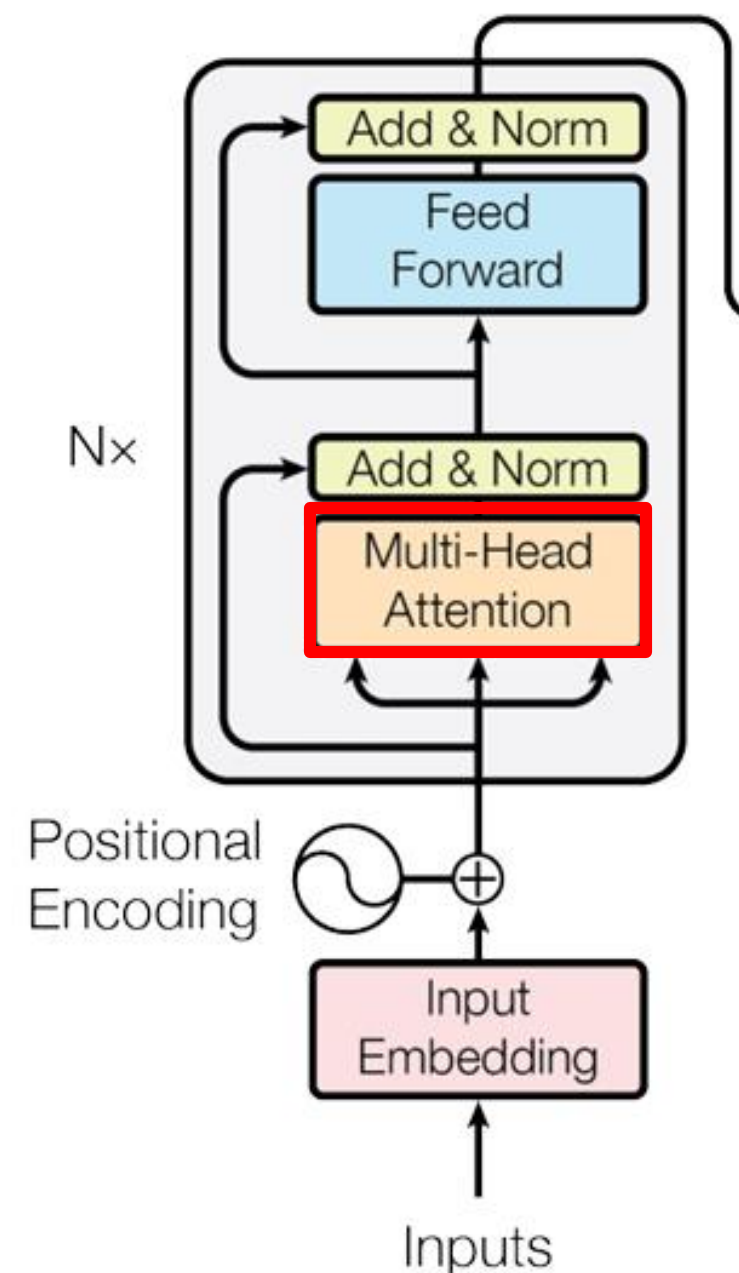


<Self-Attention>

- 한 문장 안에서 각 단어가 다른 단어와 어떻게 연관되어 있는지 파악함
- 모든 입력 단어를 동시에 참조하기 때문에, 단어 간의 장기 의존성을 잘 처리할 수 있으며 병렬로 계산이 가능하여 빠른 처리 속도를 제공함

Chapter 4. Transformer

Multi-Head Attention



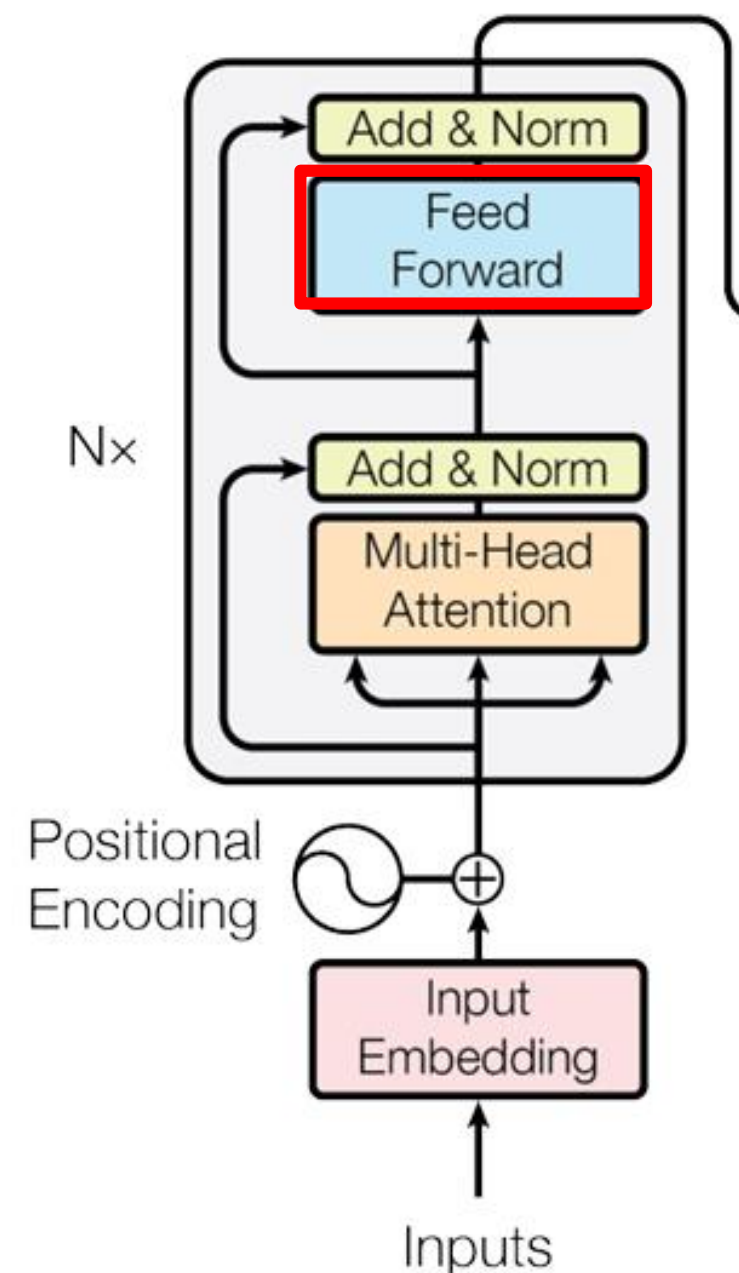
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

- Self-Attention 메커니즘을 병렬로 여러 번 수행하는 구조
- 각 Head는 자기만의 Query, Key, Value에 대해 Self-Attention을 수행하여 중요도를 계산함
- 각 Head에서 나온 Attention 결과는 모두 결합(concatenate)되어 하나의 벡터로 합쳐지고 선형 변환을 거쳐 최종 출력이 만들어짐

Chapter 4. Transformer

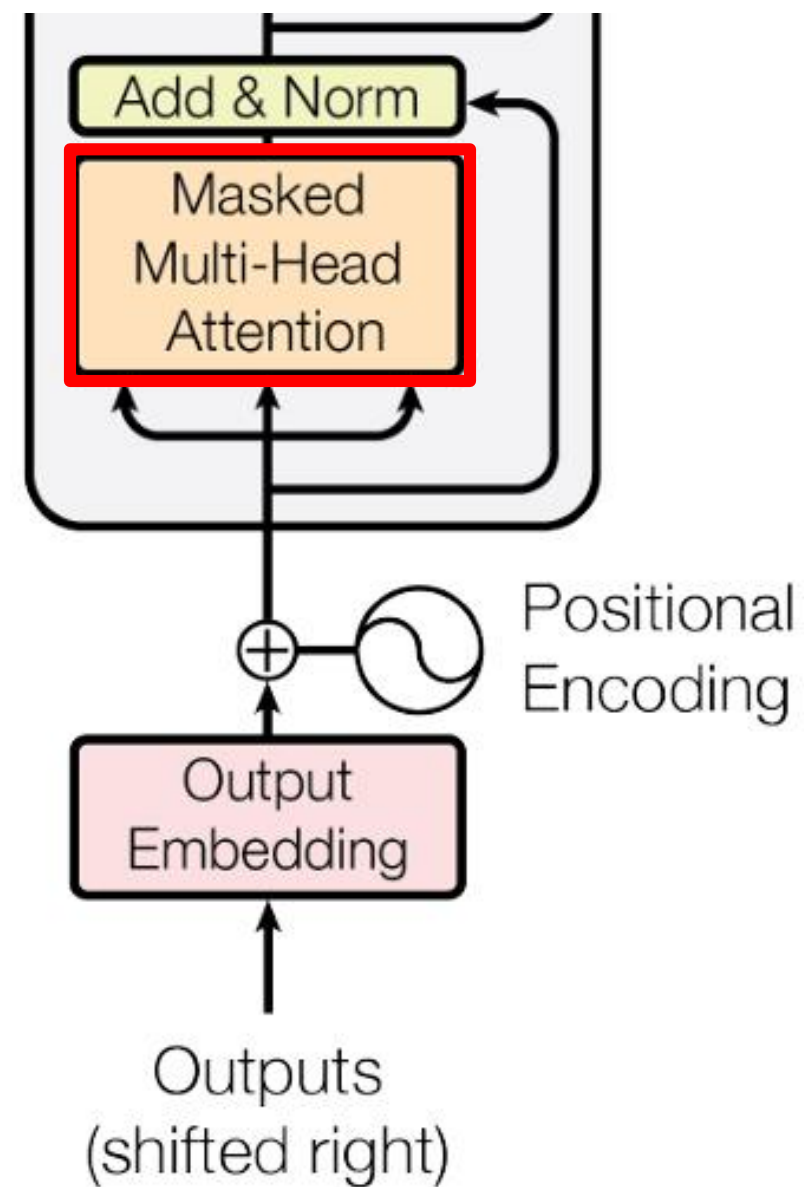
Feed Forward



- 입력 처리: 어텐션 결과를 입력으로 받아 선형 변환을 수행
- 첫 번째 선형 변환 (Fully Connected Layer): 입력 벡터의 차원을 확장하여 고차원 공간에서 정보를 표현
- 비선형 활성화: ReLU와 같은 활성화 함수를 적용하여 비선형성을 추가
- 두 번째 선형 변환: 확장된 차원을 다시 원래 차원으로 축소하여 출력으로 전달

Chapter 4. Transformer

Masked Multi-Head Attention



	I	am	a	student
I	10	$-\infty$	$-\infty$	$-\infty$
am	0	10	$-\infty$	$-\infty$
a	-5	-0.1	10	$-\infty$
student	10	-7	-8	10

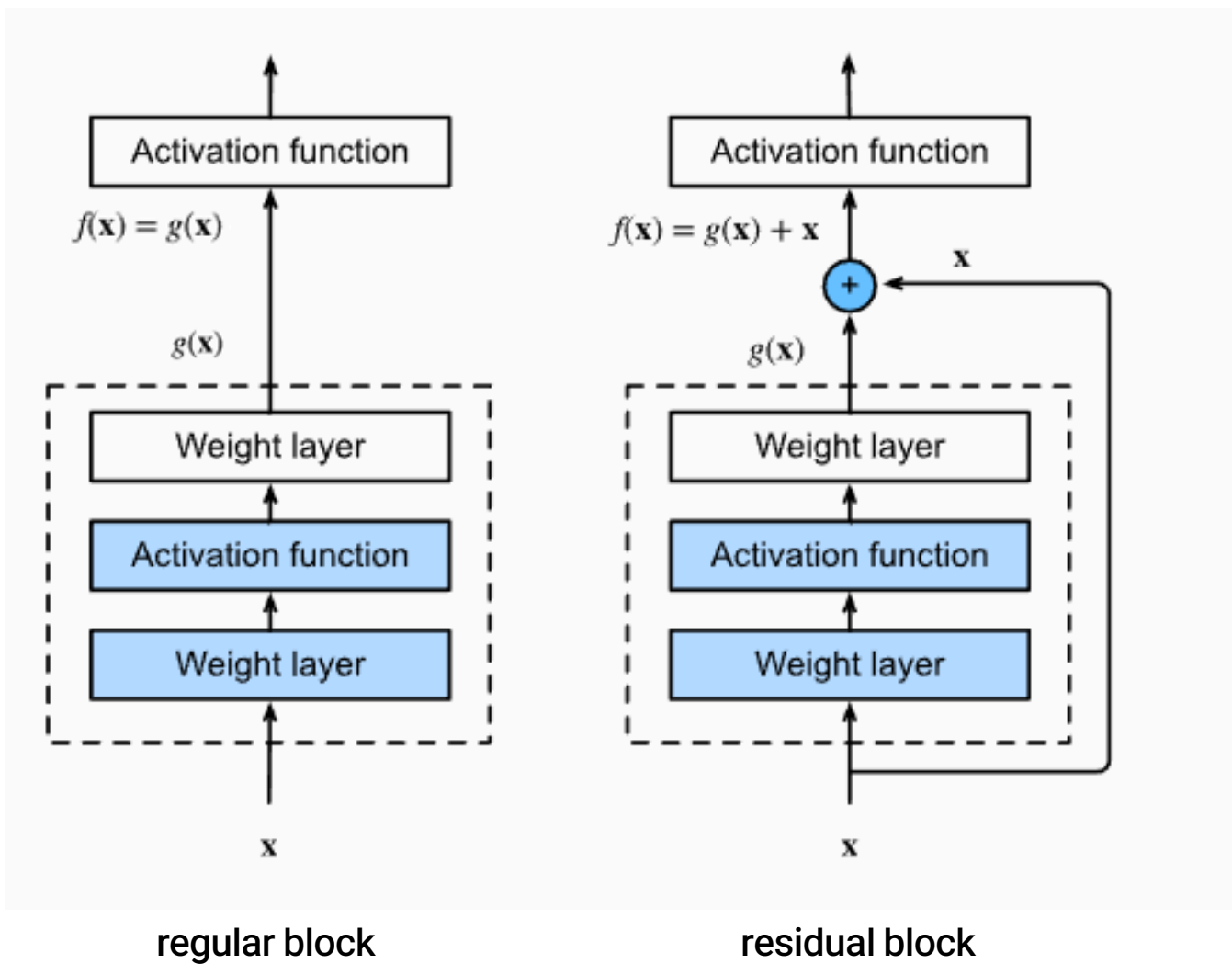
Softmax →

	I	am	a	student
I	1.0	0.0	0.0	0.0
am	0.5	1.0	0.0	0.0
a	0.3	0.4	1.0	0.0
student	1.0	0.2	0.1	1.0

- 디코더에서 사용되며, 미래 단어를 참조하지 못하도록 마스킹을 적용
- 마스크된 부분에는 큰 음수 값을 더해 Softmax 결과가 0이 되도록 처리
- 이를 통해 모델은 현재 시점까지의 정보만 사용해 다음 단어를 예측할 수 있음

Chapter 4. Transformer

Residual Block



구분	Regular Block	Residual Block
구조	단순한 순차적 레이어 (가중치, 활성화 함수 통과)	입력을 레이어에 통과시키고, 입력을 출력에 더하는 잔차 연결 포함
출력 계산식	$f(x) = g(x)$	$f(x) = g(x) + x$
기울기 소실 문제	네트워크가 깊어질수록 기울기 소실 문제 발생 가능	잔차 연결 덕분에 기울기 소실 문제 완화
깊은 네트워크 학습	깊은 네트워크에서 성능 저하 가능성	매우 깊은 네트워크에서도 성능 유지 가능

Chapter 5. 코드 구현

```
from transformers import pipeline

# 1. 감정 분석 파이프라인 생성
classifier = pipeline("sentiment-analysis")

# 2. 텍스트 입력
texts = [
    "I love this product! It's amazing and works perfectly.",
    "This is the worst experience I've ever had. Completely disappointed."
]

# 3. 감정 분석 수행
results = classifier(texts)

# 4. 결과 출력
for text, result in zip(texts, results):
    print(f"텍스트: {text}")
    print(f"분석 결과: {result}")
    print("-" * 50)
```

Chapter 5. 코드 구현

결과

텍스트: I love this product! It's amazing and works perfectly.

분석 결과: {'label': 'POSITIVE', 'score': 0.9998775720596313}

텍스트: This is the worst experience I've ever had. Completely disappointed.

분석 결과: {'label': 'NEGATIVE', 'score': 0.9998016953468323}



감사합니다

Q & A
