

이론부터 실전까지 AI에이전트 완벽 마스터

-2장 트랜스포머-

이 하 늘(haneul@pel.sejong.ac.kr)

세종대학교 프로토콜공학연구실

목차

- seq2seq 모델과 어텐션 메커니즘
- 트랜스포머 모델 소개
- 트랜스포머 학습
- 내부 메커니즘 시각화
- 트랜스포머 활용

seq2seq 모델과 어텐션 메커니즘

• 신경망 기반 기계번역 모델의 발전

문제점 (1960년대)

당시 컴퓨터의 연산 능력 부족

학습 데이터 부족

문맥과 의미를 이해해야 하는
자연어 처리의 복잡성

해결 (1990년대)

GPU의 등장으로 인한 연산 능력 확보

인터넷의 보급으로 인한
텍스트 데이터 확보

(해결 필요!)

RNN
(1986)

문장을 순차적으로 읽는
신경망 구조

LSTM
(1997)

긴 문장을 RNN에 비해
잘 기억하도록 개선된 신경망 구조

seq2seq
(2014)

입력 문장을 출력 문장으로
변환하는 모델

Transformer
(2017)

Attention을 사용한
인코더-디코더 모델

seq2seq + Attention
(2015)

번역할 때 필요한 단어에 집중하도록 하는
메커니즘인 Attention이 추가됨

seq2seq 모델과 어텐션 메커니즘

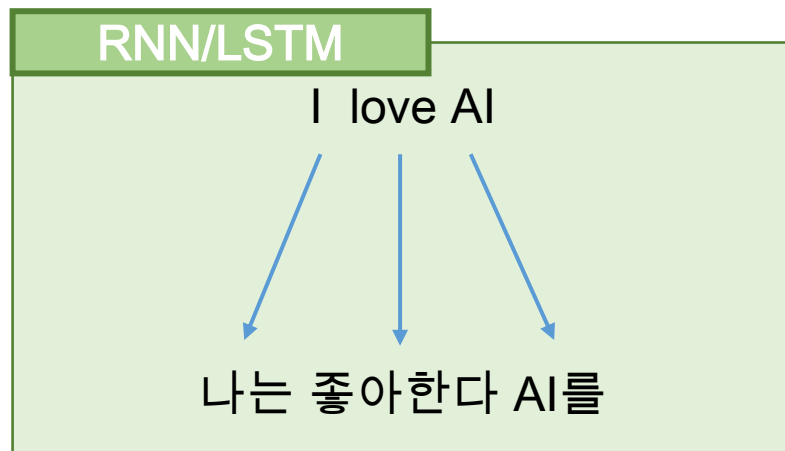
- seq2seq (sequence to sequence) 모델 (1/5)

- RNN 기반의 기계 번역 모델

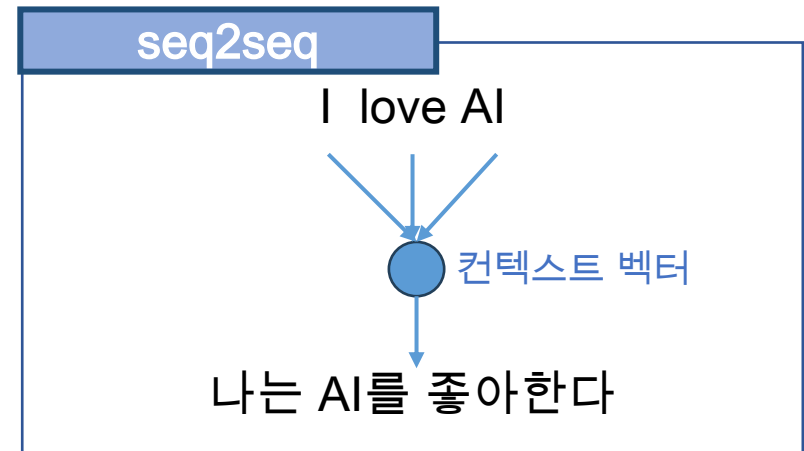
1) 컨텍스트 벡터(context vector):
번역하고자 하는 입력 문장의 정보를 압축한 벡터

- 특징

- 입력 문장과 출력 문장의 길이가 서로 달라도 번역 가능
 - e.g., 영어 5단어 → 한국어 8단어
- 입력 문장 전체를 하나의 컨텍스트 벡터¹⁾로 압축하고, 이를 이용해 새로운 문장을 생성



입력을 읽으면서 동시에 출력 생성



입력 문장 전체를 이해한 후 출력 생성

seq2seq 모델과 어텐션 메커니즘

• seq2seq 모델 (2/5)

1) 정보: 문장을 번역하는 데 필요한 의미와 문맥 정보
e.g., 문법(syntax), 단어의 의미(semantics), 단어 간의 관계(context) 등

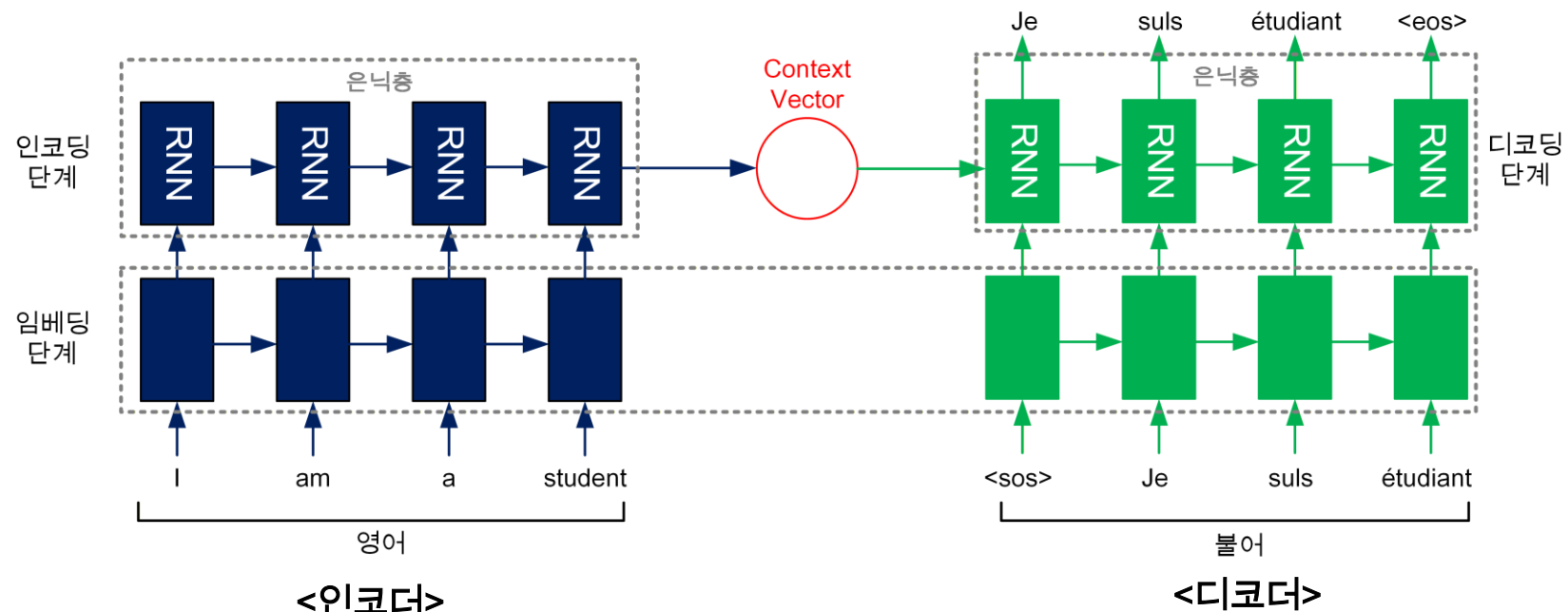
• 구성요소

• 인코더 (encoder)

- 번역하고자 하는 입력 문장의 정보¹⁾를 추출 및 정리하여 압축된 컨텍스트 벡터를 생성하는 모듈

• 디코더 (decoder)

- 인코더가 전달한 컨텍스트 벡터로부터 번역 문장을 생성하는 모듈



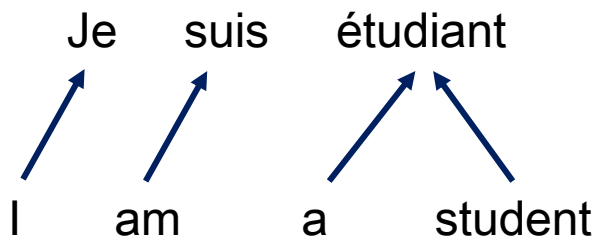
seq2seq 모델과 어텐션 메커니즘

- seq2seq 모델 (3/5)

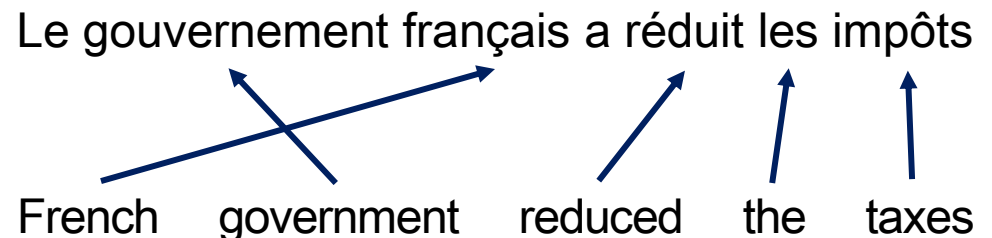
- 문제점 (1/3)

1. 정렬 문제

- 입력 문장과 출력 문장의 길이가 서로 상이하여 발생하는 문제로서, 출력(번역)된 각 단어가 어떤 입력 단어에 해당하는 지 알 수 없음
- 고정된 크기의 컨텍스트 벡터에 문장의 의미를 압축해서 답아야 하는 구조적 한계로 인해 발생



<단어 하나가 여러 단어를 포괄하는 일대다 문제>



<불필요한 단어 삽입 문제>

seq2seq 모델과 어텐션 메커니즘

• seq2seq 모델 (4/5)

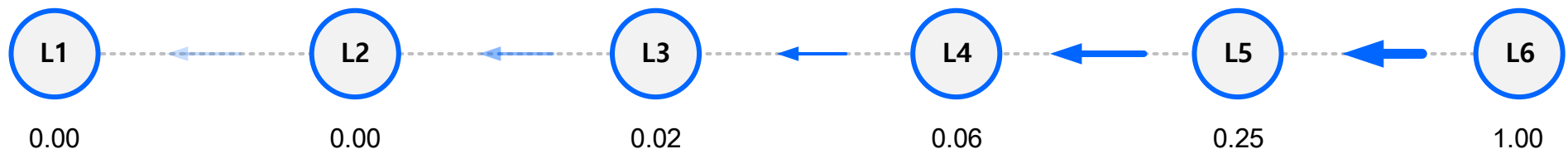
1) 역전파(backpropagation):

각 층의 가중치가 손실에 미치는 영향을 최소화하기 위해, 출력층에서 입력층의 방향으로 계산된 오차를 전달하는 과정

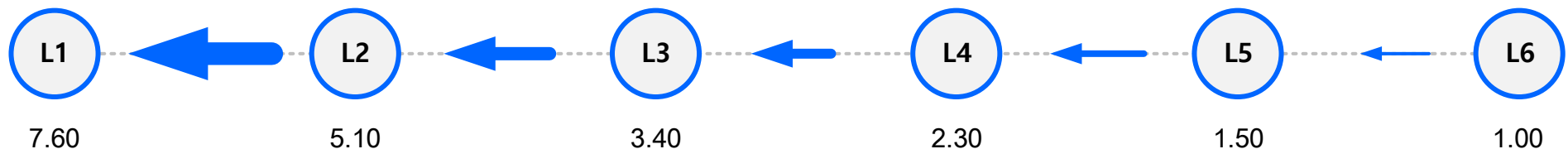
• 문제점 (2/3)

2. 기울기 소실 및 폭발 문제

- 역전파 과정¹⁾에서 입력층에 가까워질수록 기울기가 0에 가깝게 작아지거나(기울기 소실), 반대로 너무 증가하여(기울기 폭발) 학습 성능을 저하시키는 문제
 - 기울기 소실이 발생하면 문장 초반의 단어들이 제대로 학습되지 않아 긴 문맥을 파악할 수 없음
 - 기울기 폭발이 발생하면 학습 과정이 불안정해져 모델의 성능이 저하됨



<기울기 소실>



<기울기 폭발>

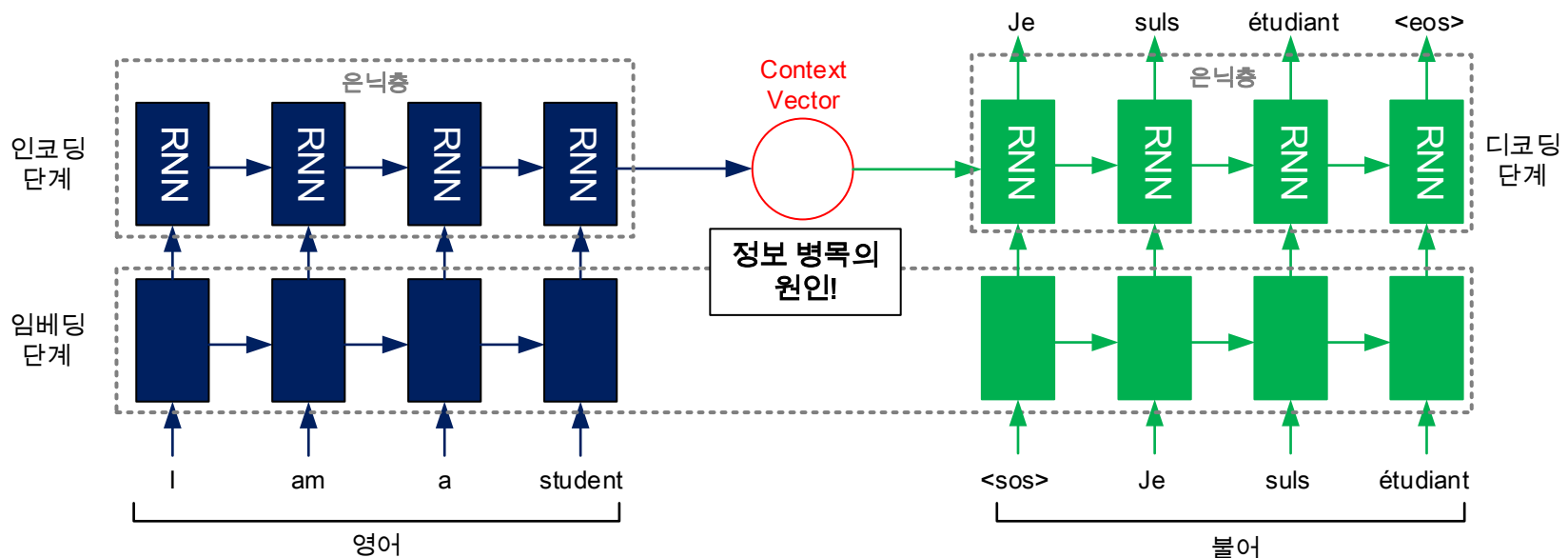
seq2seq 모델과 어텐션 메커니즘

- seq2seq 모델 (5/5)

- 문제점 (3/3)

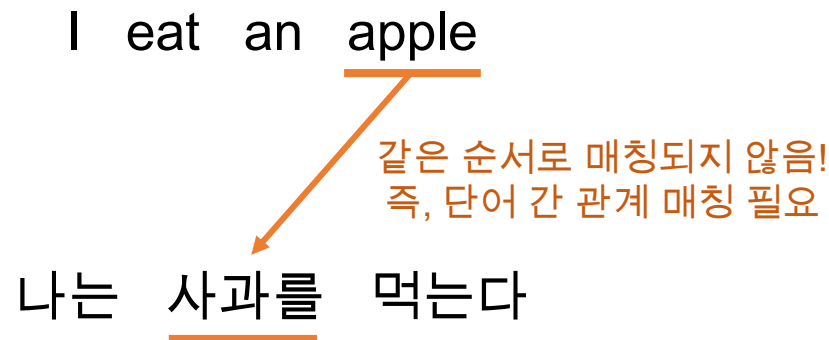
3. 정보 병목 현상

- 한정된 크기의 컨텍스트 벡터에 과거 정보를 저장하므로, 긴 문장을 처리할 경우 앞부분의 중요 정보를 잃는 문제(정보 병목 현상)가 발생
- e.g., 1,000개 단어로 이루어진 긴 문장을 512 차원의 고정된 크기의 컨텍스트 벡터에 압축하려 한다면, 정보가 유실될 수 밖에 없음



seq2seq 모델과 어텐션 메커니즘

- 어텐션 메커니즘 (attention mechanism) (1/7)
- 입력 문장에 존재하는 단어들과 번역된 문장의 단어들 간 관계를 학습하기 위해 고안된 메커니즘



- 특징
 - 입력과 출력 단어 간의 정렬 문제 해결
 - 번역 시 중요한 입력 단어에 더 큰 가중치를 부여하여, 정보 병목 현상 개선

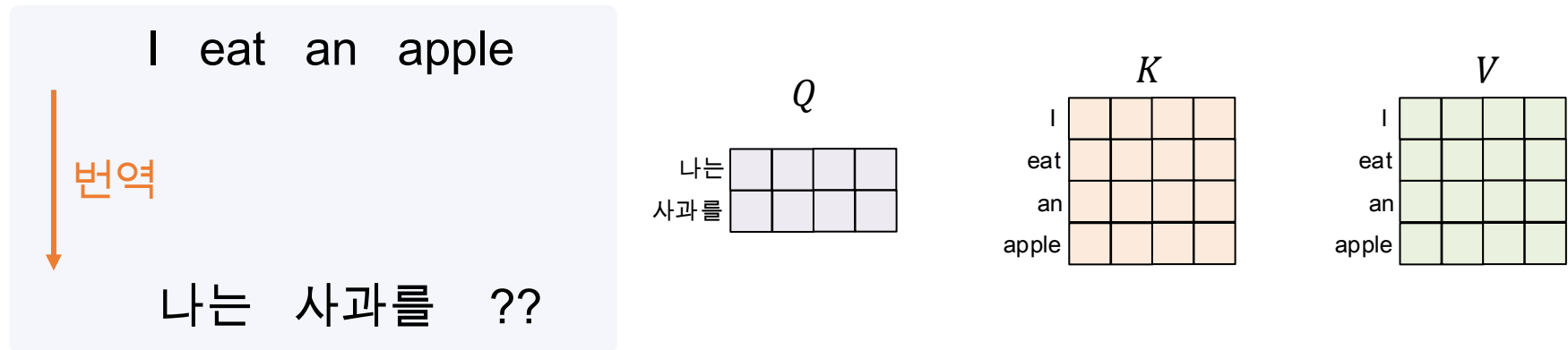
seq2seq 모델과 어텐션 메커니즘

• 어텐션 메커니즘 (2/7)

• X : 입력 문장을 벡터로 표현한 배열

• 핵심 개념

- Query(질의), Q : 현재 필요한 정보를 찾기 위한 질문을 나타내는 벡터
- Key(키), K : 각 정보가 어떤 특징을 가지고 있는지를 나타내며, Q 와 관련도를 비교하는 벡터
- Value(값), V : 실제로 참고하여 가져올 정보의 내용을 담고 있는 벡터



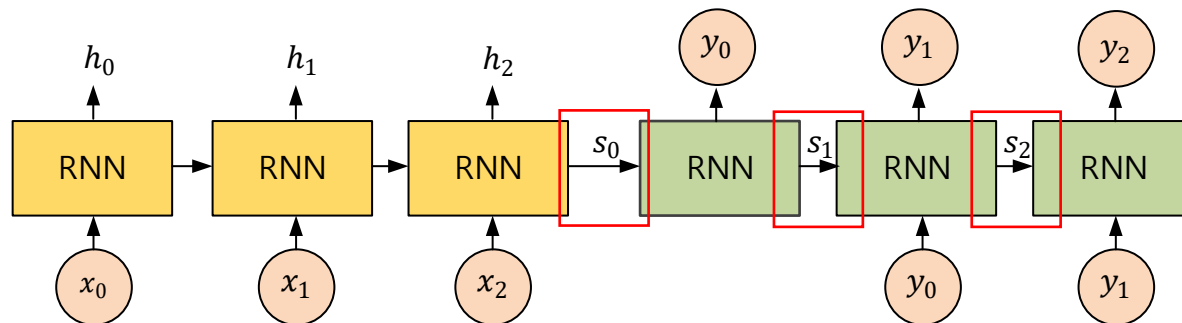
참고있는 정보(Query)와 색인(Key)과의 관련도를 계산(Comparison)하여, 관련도가 높은 실제 정보(Value)에 가중치를 부여(Aggregation)하는 방식

seq2seq 모델과 어텐션 메커니즘

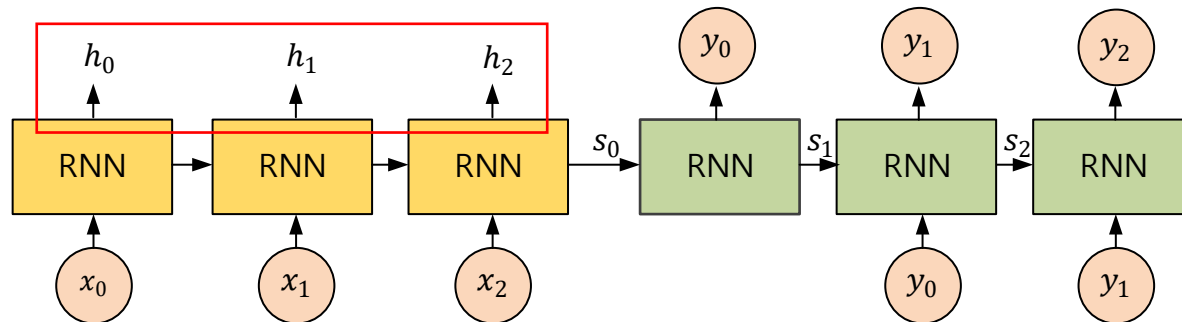
- 어텐션 메커니즘 (3/7)

- seq2seq 모델과 어텐션 메커니즘의 활용 (1/2)

- 현재 시점의 단어를 생성하기 전에 지금까지의 디코더 상태를 이용해 어떤 인코더 정보를 참고할지 결정
 - seq2seq에서는 디코더의 이전 은닉상태 값(s_{i-1})들을 Q 로 사용



- seq2seq에서는 인코더의 은닉상태 값(h_i)들을 K 와 V 로 사용

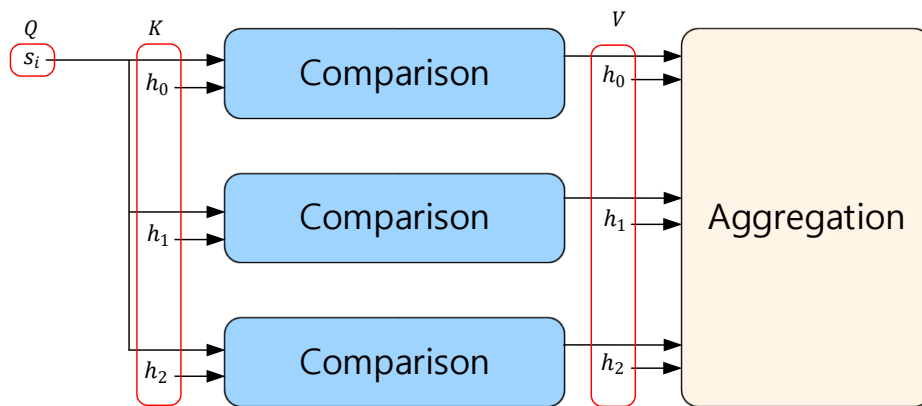


seq2seq 모델과 어텐션 메커니즘

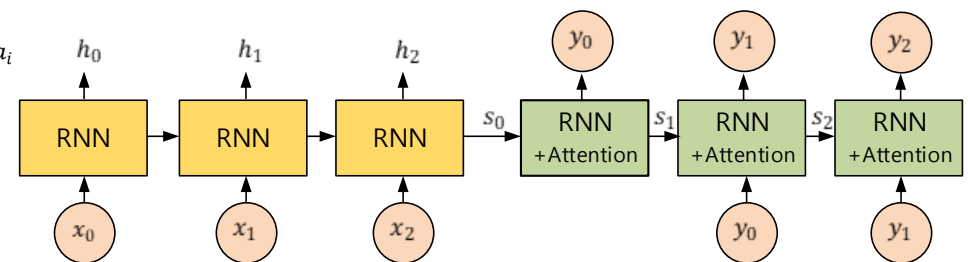
• 어텐션 메커니즘 (4/7)

• seq2seq 모델과 어텐션 메커니즘의 활용 (2/2)

- Q 와 인코더에 있는 모든 K 간의 관련도를 계산
- 해당 유사도들을 V 와 연산하여 어텐션 값(a_i)을 출력함



<어텐션 메커니즘의 개요도>



<seq2seq 모델에 어텐션 메커니즘을 활용하는 경우>

seq2seq 모델과 어텐션 메커니즘

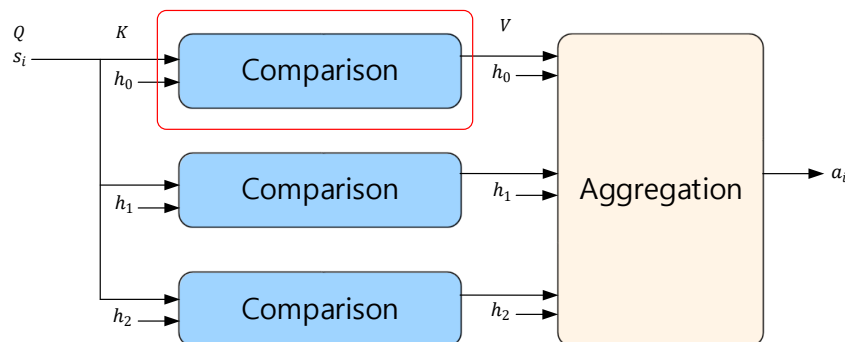
• 어텐션 메커니즘 (5/7)

1. 과정: 정렬 점수 계산

- 디코더에서 현재 번역(생성)하려는 단어와 관련성이 높은 인코더의 입력 단어를 찾기 위함
- Q 인 디코더의 이전 은닉 상태 s_{i-1} 과 K 인 인코더의 각 은닉 상태 h_j 를 $score()$ 로 비교하여 정렬 점수($e_{i,j}$)를 계산

$$e_{i,j} = score(s_{i-1}, h_j)$$

- $e_{i,j}$: 현재 출력 단어 생성 시 j 번째 입력 단어의 중요도를 나타내는 정렬 점수
- $score()$: 두 상태의 관련성(즉, 유사도)를 계산하는 함수
- s_{i-1} : 이전 시점의 디코더 은닉 상태
- h_j : j 번째 입력 단어의 인코더 은닉 상태



<정렬 점수($e_{i,j}$) 계산 예시>

입력 : I, eat, apple
출력 : 먹는다

I → 2.1
eat → 5.4
apple → 1.8

seq2seq 모델과 어텐션 메커니즘

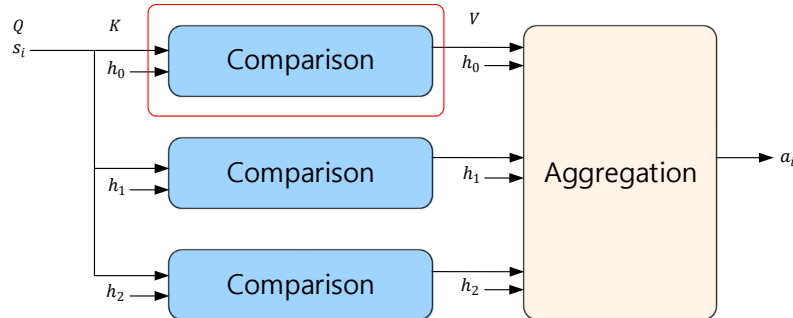
• 어텐션 메커니즘 (6/7)

2. 과정: 어텐션 가중치 계산

- Q 와 각 K 의 유사도를 비교한 정렬 점수들에 $\text{softmax}()$ 적용
- 점수들을 합이 1인 0과 1 사이의 확률 값으로 정규화
 - 현재 시점에서 각 입력 단어에 얼마나 집중할지를 나타내는 어텐션 가중치($a_{i,j}$)를 계산

$$a_{i,j} = \text{softmax}(e_{i,j}) = \frac{\exp(e_{i,j})}{\sum_{k=1}^t \exp(e_{i,k})}$$

- $a_{i,j}$: 어텐션 가중치
- $e_{i,j}$: 정렬 점수
- $\sum_{k=1}^t \exp(e_{i,k})$: 인코더의 첫 번째 단어($k=1$)부터 마지막 단어($k=t$)까지의 점수들을 지수화하여 모두 더한 값
- $\text{softmax}()$: 모든 정렬 점수($e_{i,j}$)를 0에서 1사이로 정규화



<어텐션 가중치($a_{i,j}$) 계산 예시>

I → 2.1 → 0.035
eat → 5.4 → 0.940
apple → 1.8 → 0.025

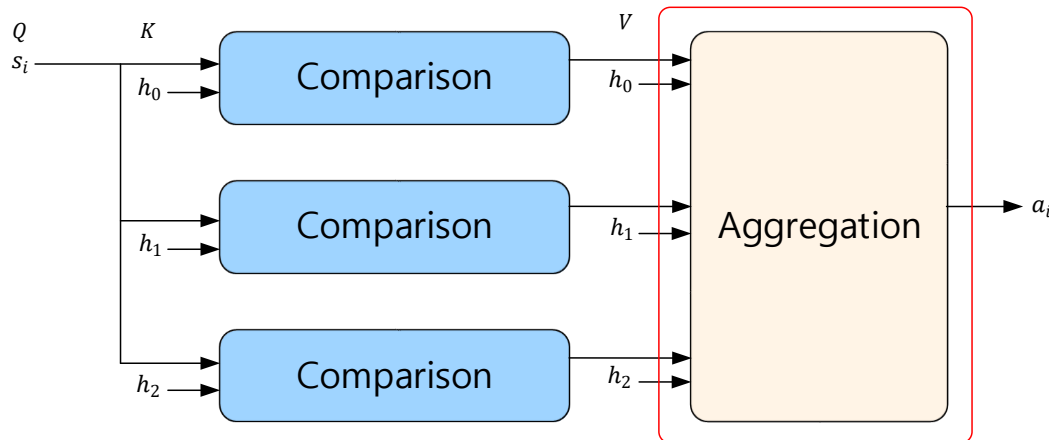
seq2seq 모델과 어텐션 메커니즘

• 어텐션 메커니즘 (7/7)

3. 과정: 가중합 계산

- 각 V 에 해당하는 인코더의 은닉 상태(h_j)에 앞서 구한 어텐션 가중치($a_{i,j}$)를 곱한 후 모두 더해 가중합(c_i)을 구함
 - 가중합을 컨텍스트 벡터로 사용함

$$c_i = \sum_{j=1}^T a_{i,j} \cdot h_j$$



- c_i : 컨텍스트 벡터, 가중합
- $a_{i,j}$: j 번째 입력 단어의 어텐션 가중치
- h_j : j 번째 입력 단어의 인코더 은닉 상태
- T : 입력 문장의 길이

<가중합(c_i) 계산 예시>

I → $0.035 \times h_1$
eat → $0.940 \times h_2$
apple → $0.025 \times h_3$

모두 덧셈

→ c_i

seq2seq 모델과 어텐션 메커니즘

- 셀프 어텐션 메커니즘(self-attention mechanism) (1/6)
 - 하나의 문장 안에서 단어들 사이의 관계를 파악해 특정 단어의 의미를 예측할 수 있는 어텐션 메커니즘

그

동물은

길을

건너지

않았다

왜냐하면

그것은

너무

피곤했기

때문이다

"그것은"이 무엇을 지칭하는지 판단할 때, 셀프 어텐션은 문장 내 "그", "동물은"과의 관련도를 높게 계산함

• 특징

- 문장 안에서 각 단어가 다른 모든 단어와 관련 있는 정도를 학습하여, 문맥에 따라 달라지는 단어의 의미를 예측함

구분	일반 어텐션	셀프 어텐션
대상	서로 다른 두 시퀀스 간의 관계 (입력 문장 - 출력 문장)	단일 시퀀스 내 내부 단어들 간의 관계 (입력 문장 - 입력 문장)
Query (Q)	디코더의 은닉 상태 (s_i)	자기 자신의 은닉 상태
Key (K), Value (V)	인코더의 은닉 상태 (h_j)	자기 자신의 은닉 상태
주요 목적	번역할 때 디코더의 특정 단어가 인코더의 어떤 단어를 참조할지 결정	문장 내에서 단어와 단어 사이의 문맥적/문법적 관계를 파악

seq2seq 모델과 어텐션 메커니즘

• 셀프 어텐션 메커니즘 (2/6)

• X : 입력 문장을 벡터로 표현한 배열

• 핵심 개념

Query (질의)

$$Q = X \cdot W^Q$$

현재 필요한 정보를 찾기 위한 질문을 나타내는 벡터

Key (키)

$$K = X \cdot W^K$$

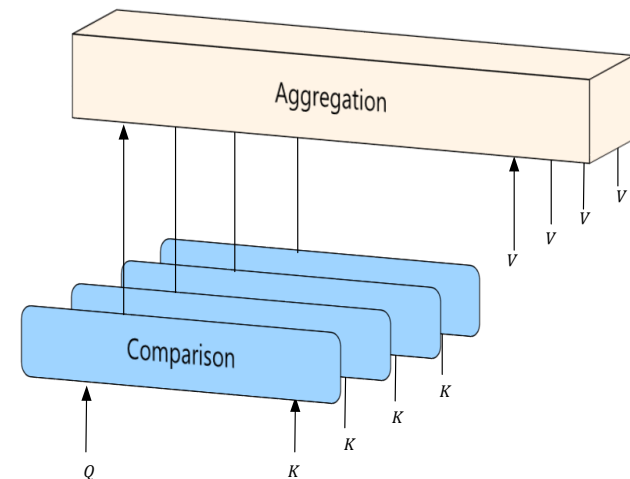
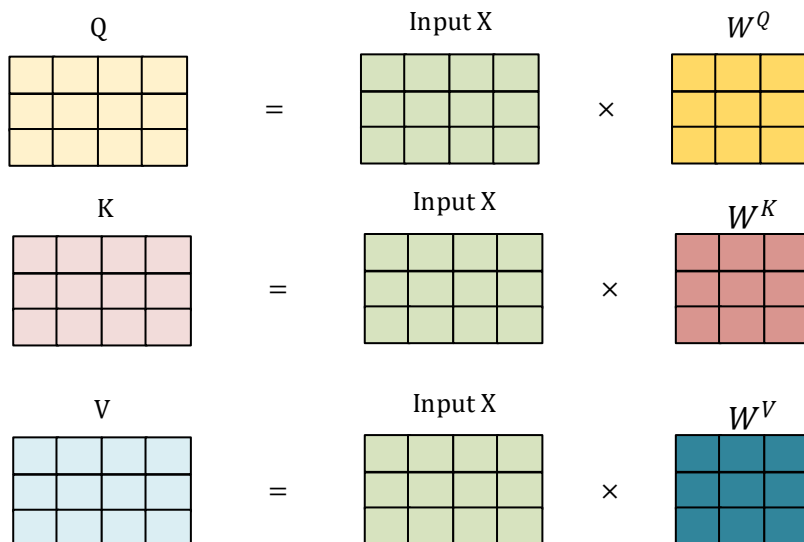
각 정보가 어떤 특징을 가지고 있는지를 나타내며, Q 와 관련도를 비교하는 벡터

Value (값)

$$V = X \cdot W^V$$

실제로 참고하여 가져올 정보의 내용을 담고 있는 벡터

찾고있는 정보(Query)와 색인(Key)과의 관련도를 계산(Comparison)하여, 관련도가 높은 실제 정보(Value)에 가중치를 부여(Aggregation)하는 방식



seq2seq 모델과 어텐션 메커니즘

• 셀프 어텐션 메커니즘 (3/6)

- d_k : 시퀀스의 크기
- K^T : key 행렬을 전치한 것

1. 과정: Query, Key, Value 생성

- 입력 문장의 각 단어를 Q, K, V 벡터로 변환

$$\begin{aligned}Q &= X \times W_Q \\K &= X \times W_K \\V &= X \times W_V\end{aligned}$$

• 예시

- 입력 시퀀스 벡터 $X = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

Query (질의)

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix}$$

Key (키)

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 4 & 4 & 0 \\ 2 & 3 & 1 \end{bmatrix}$$

Value (값)

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 2 & 0 \\ 0 & 3 & 0 \\ 1 & 0 & 3 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{bmatrix}$$

seq2seq 모델과 어텐션 메커니즘

• 셀프 어텐션 메커니즘 (4/6)

• K^T : key 행렬을 전치한 것

2. 과정: 어텐션 점수 계산

- Q 와 모든 K 를 비교해 관련도(QK^T)를 계산
- 각 단어가 문장 내 다른 단어와 관련 있는 정도를 계산

$$QK^T$$

• 예시

- 관련도 QK^T 계산 $Q = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix}$ $K = \begin{bmatrix} 0 & 1 & 1 \\ 4 & 4 & 0 \\ 2 & 3 & 1 \end{bmatrix}$ $K^T = \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix}$

$$QK^T = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix} \times \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 4 \\ 4 & 16 & 12 \\ 4 & 12 & 10 \end{bmatrix}$$

seq2seq 모델과 어텐션 메커니즘

• 셀프 어텐션 메커니즘 (5/6)

- d_k : 시퀀스의 크기 (벡터 차원의 크기)
- K^T : key 행렬을 전치한 것

3. 과정: 어텐션 가중치 계산

- $\text{softmax}()$ 를 통해 관련도를 0과 1사이로 정규화하여 가중치 계산

$$\text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$$

• 예시

- 어텐션 가중치 $\text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$ 계산 $QK^T = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix} \times \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 4 \\ 4 & 16 & 12 \\ 4 & 12 & 10 \end{bmatrix}$

$$\text{softmax}\left(\left[\frac{2}{\sqrt{3}}, \frac{4}{\sqrt{3}}, \frac{4}{\sqrt{3}}\right]\right) = [0.13613, 0.43194, 0.43194]$$

$$\text{softmax}\left(\left[\frac{4}{\sqrt{3}}, \frac{16}{\sqrt{3}}, \frac{12}{\sqrt{3}}\right]\right) = [0.00089, 0.90884, 0.09027]$$

$$\text{softmax}\left(\left[\frac{4}{\sqrt{3}}, \frac{12}{\sqrt{3}}, \frac{10}{\sqrt{3}}\right]\right) = [0.00744, 0.75471, 0.23785]$$

seq2seq 모델과 어텐션 메커니즘

• 셀프 어텐션 메커니즘 (6/6)

- d_k : 시퀀스의 크기
- K^T : key 행렬을 전치한 것

4. 과정: 출력 벡터 생성

- 어텐션 가중치를 V 와 곱하여 각 단어의 출력 생성

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

• 예시

- 출력 벡터 생성

$$softmax\left(\left[\frac{2}{\sqrt{3}}, \frac{4}{\sqrt{3}}, \frac{4}{\sqrt{3}}\right]\right) = [0.13613, 0.43194, 0.43194]$$

$$softmax\left(\left[\frac{4}{\sqrt{3}}, \frac{16}{\sqrt{3}}, \frac{12}{\sqrt{3}}\right]\right) = [0.00089, 0.90884, 0.09027]$$

$$softmax\left(\left[\frac{4}{\sqrt{3}}, \frac{12}{\sqrt{3}}, \frac{10}{\sqrt{3}}\right]\right) = [0.00744, 0.75471, 0.23785]$$

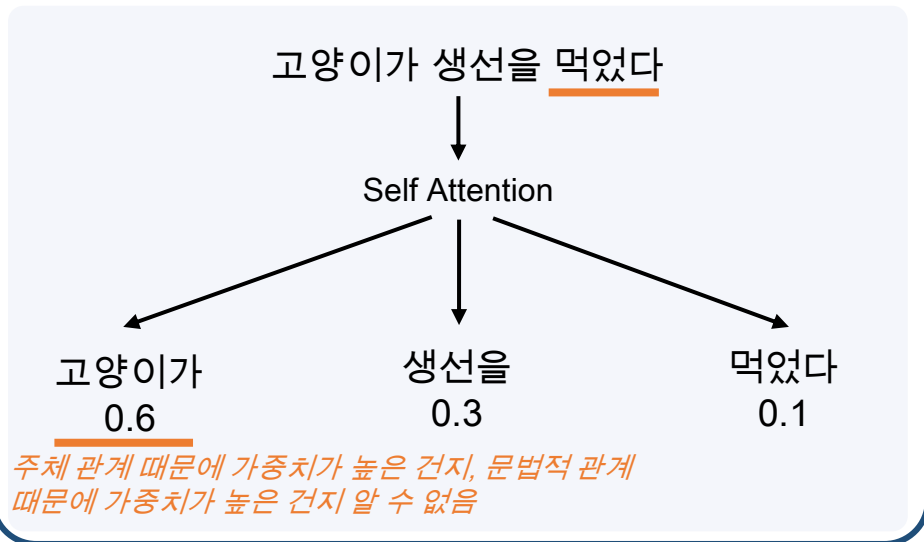
$$Attention(Q, K, V) = \begin{bmatrix} 0.13613 & 0.43194 & 0.43194 \\ 0.00089 & 0.90884 & 0.09027 \\ 0.00744 & 0.75471 & 0.23785 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{bmatrix} = \begin{bmatrix} 1.8639 & 6.3194 & 1.7042 \\ 1.9991 & 7.8141 & 0.2735 \\ 1.9926 & 7.4796 & 0.7359 \end{bmatrix}$$

seq2seq 모델과 어텐션 메커니즘

- 셀프 어텐션 메커니즘: 확장 (1/2)
 - 멀티헤드 셀프 어텐션
 - 하나의 셀프 어텐션을 여러 개의 헤드로 나누어 병렬로 수행

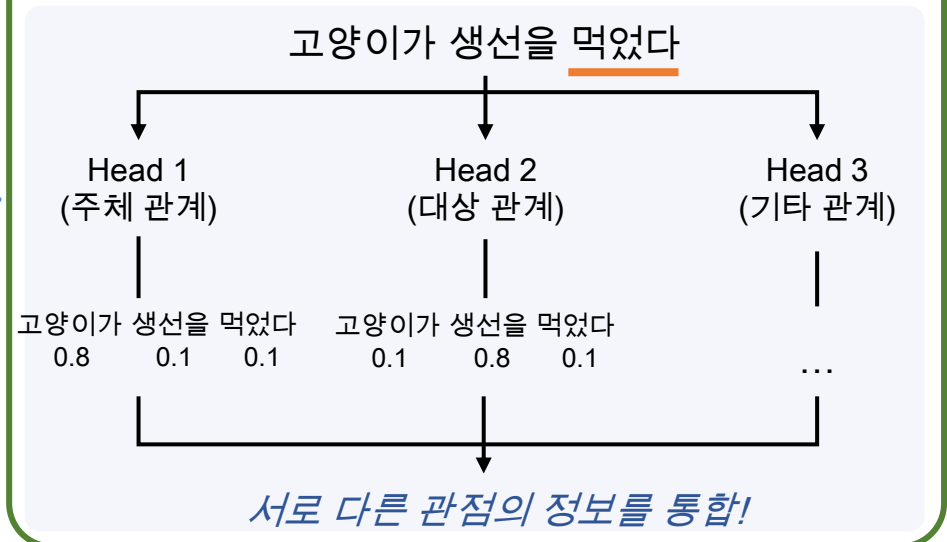
기존 셀프 어텐션

하나의 어텐션에서 모든 토큰 간 관계를 하나의 관점으로 계산



멀티헤드 셀프 어텐션

여러 어텐션 헤드가 서로 다른 관점에서 토큰 간 관계를 병렬로 학습

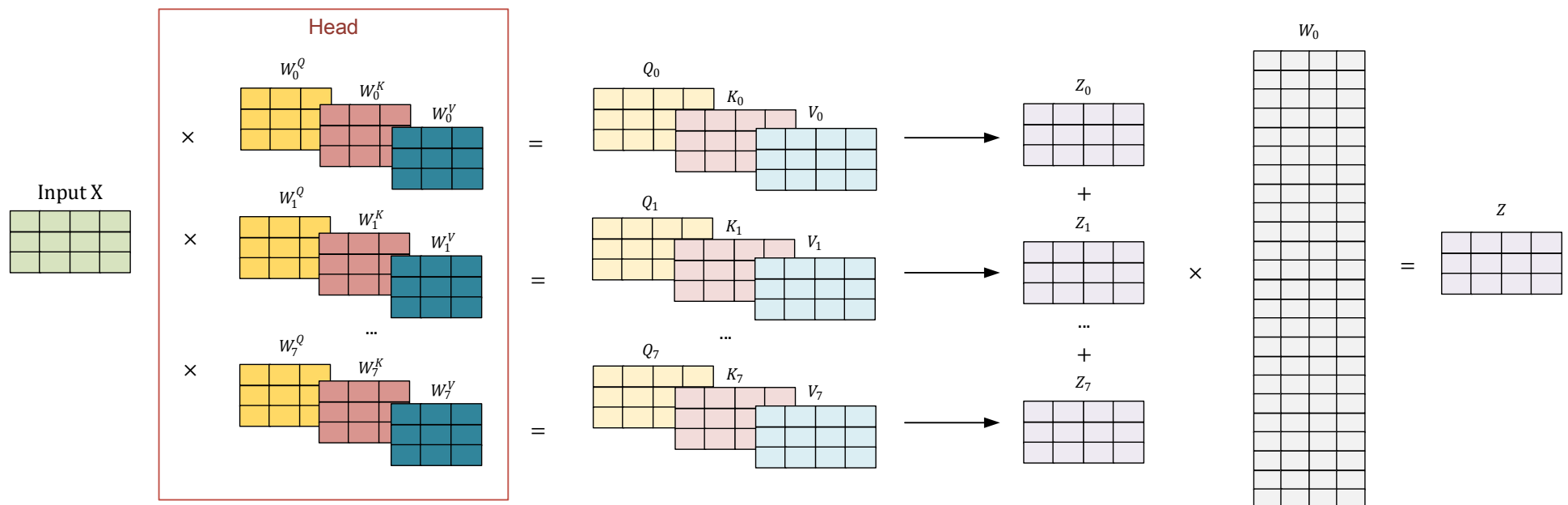


seq2seq 모델과 어텐션 메커니즘

- 셀프 어텐션 메커니즘: 확장 (2/2)

- 멀티헤드 셀프 어텐션: 과정

- 입력 행렬 X 를 헤드 수만큼 Q, K, V 로 각각 선형 변환
- 각 헤드별 병렬 어텐션 연산을 수행하여 맥락 벡터 Z_i 산출
- 각 맥락 벡터를 하나로 연결하여 가중치 행렬 W_0 를 곱해 최종 출력 Z 산출



트랜스포머 모델 소개

• 셀프 어텐션에도 불구하고 남아있던 문제들

모델이 문장의 의미를 제대로 파악하지 못하고 오류가 다수

모델이 학습 당시의 어휘집에 없는 단어를 처리하기 어려움

대명사 및 기타 문법적 형태를 다룰 때 오류 발생

긴 텍스트 문맥을 유지하기 어려움

학습·테스트 데이터셋의 도메인이 다를 경우, 일반화 능력 저하

RNN은 구조적 한계로 병렬 연산이 불가능해 반드시 순차적으로 계산 (연산 비효율성)



RNN을 개선하는 대신 제거하고, 멀티헤드 셀프 어텐션을 이용하자!

2016년, 구글

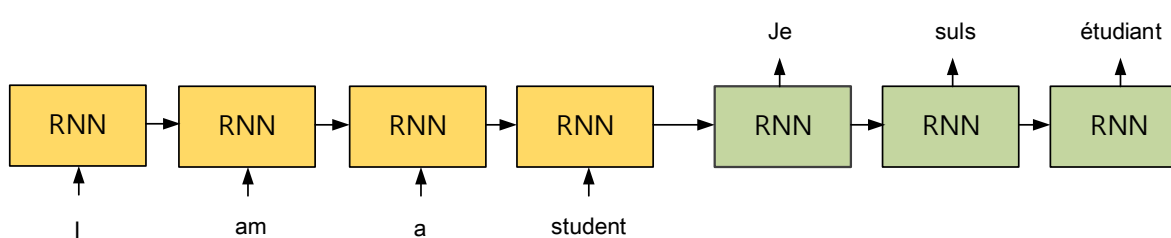
트랜스포머 모델 소개

- 정의

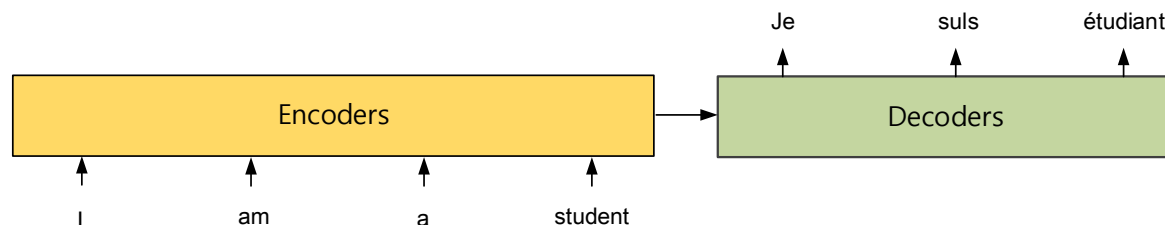
- 멀티헤드 셀프 어텐션을 여러 층으로 쌓아 올린 신경망 아키텍처

- 특징

- seq2seq 모델과 유사하게, 인코더-디코더 구조를 활용
- 인코더와 디코더를 여러 개 사용함



<seq2seq 모델의 개요도>



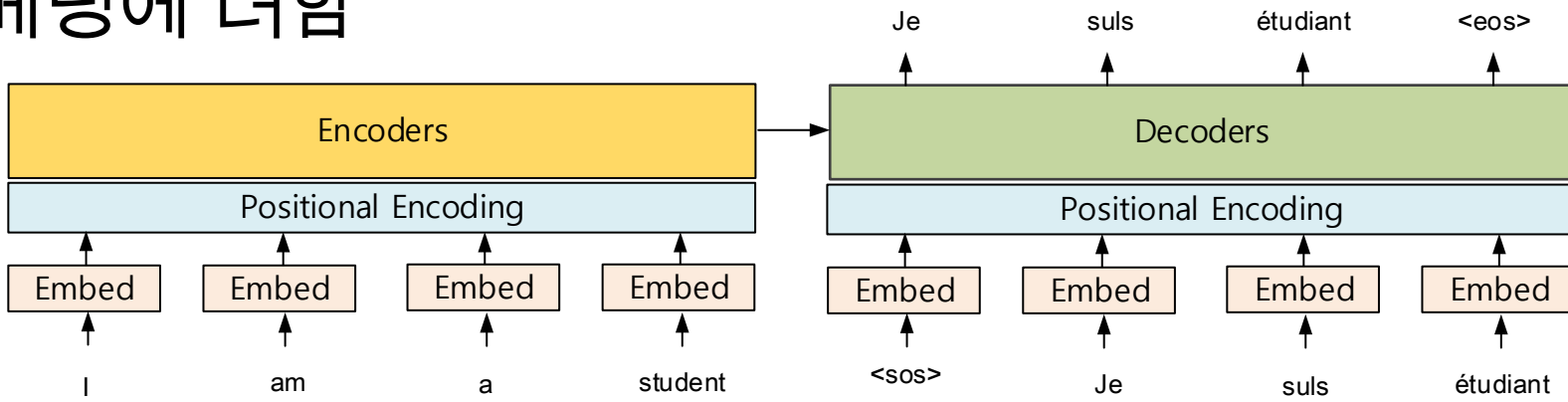
<트랜스포머 모델의 개요도>

구분	RNN 기반 모델	트랜스포머
처리 방식	토큰을 순서대로 하나씩 입력 (순차 처리)	모든 토큰을 한 번에 입력 (병렬 처리)
속도	느림 (반드시 순차적으로 계산)	빠름 (병렬 연산 가능)
예시	‘나’ → ‘는’ → ‘학생’ → ‘입니다’ 순서로 입력	[‘나’, ‘는’, ‘학생’, ‘입니다’]가 동시에 입력
순서 인식	구조상 자연스럽게 단어의 앞뒤 순서 파악 가능	병렬 입력이므로 순서 정보가 소실됨

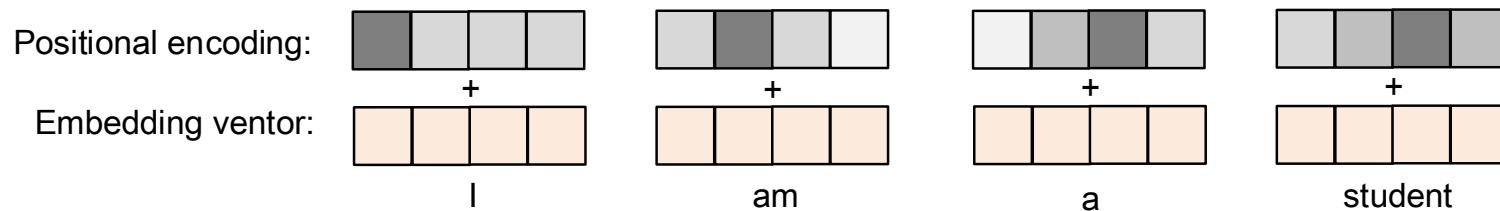
트랜스포머 모델 소개

• 위치 인코딩 (Positional Encoding) (1/2)

- 병렬 처리로 인해 사라지는 단어의 순서 정보를 보완
- 각 단어의 위치 정보를 나타내는 위치 벡터를 단어 임베딩에 더함



• 시각화 예시



트랜스포머 모델 소개

• 위치 인코딩 (Positional Encoding) (2/2)

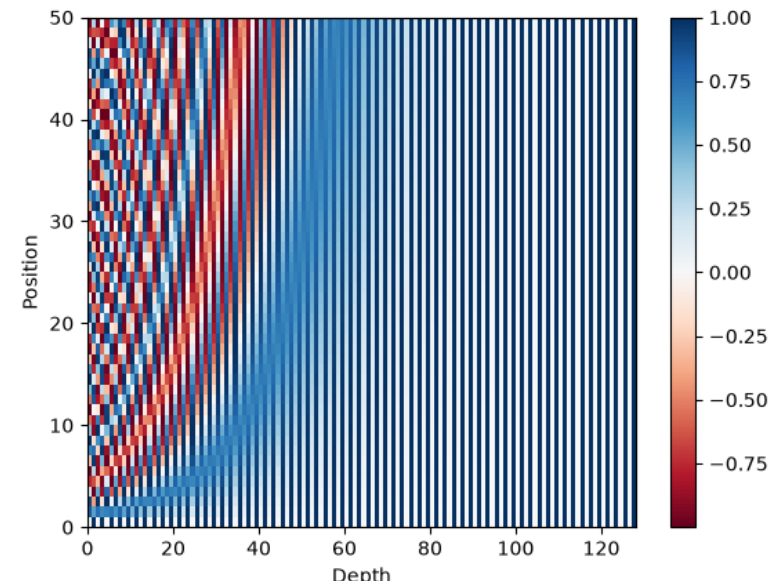
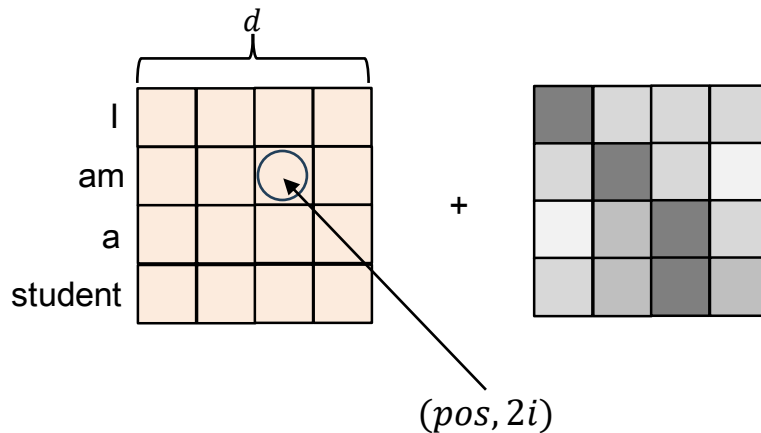
• 수식 표현

- 짝수 차원에는 $\sin$ 함수, 홀수 차원에는 $\cos$ 함수를 적용하여 위치별 고유한 패턴을 생성

- pos : 입력 문장에서의 토큰의 위치
- i : 임베딩 벡터의 차원 인덱스
- d : 임베딩 벡터의 전체 크기 차원

$$PE(pos, 2i) = \sin(pos / 10000^{\frac{2i}{d}})$$

$$PE(pos, 2i + 1) = \cos(pos / 10000^{\frac{2i}{d}})$$



<50 x 128 크기 위치 인코딩 시각화>

트랜스포머 모델 소개

- 트랜스포머 아키텍처

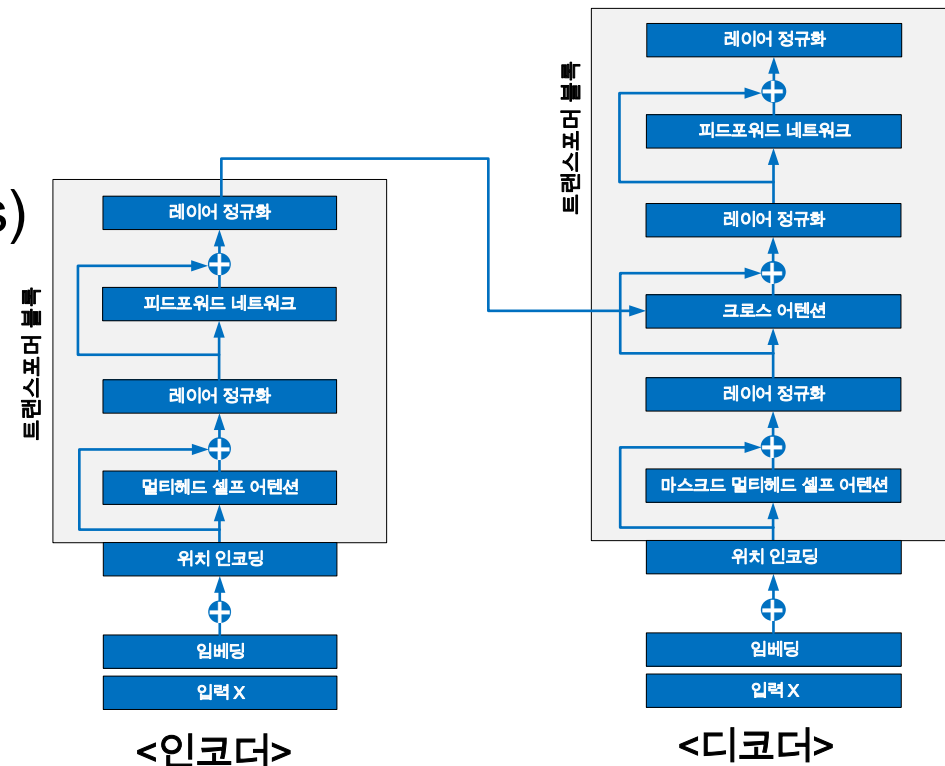
- 인코더-디코더 구조

- 트랜스포머 블록

- 트랜스포머에서 문맥 정보를 학습하는 기본 단위

- 주요 구성요소

- 멀티헤드 셀프 어텐션
- 피드포워드 네트워크 (FFN, Feed-Forward Networks)
- 잔차 연결 (Residual Connection)
- 레이어 정규화 (Layer Normalization)



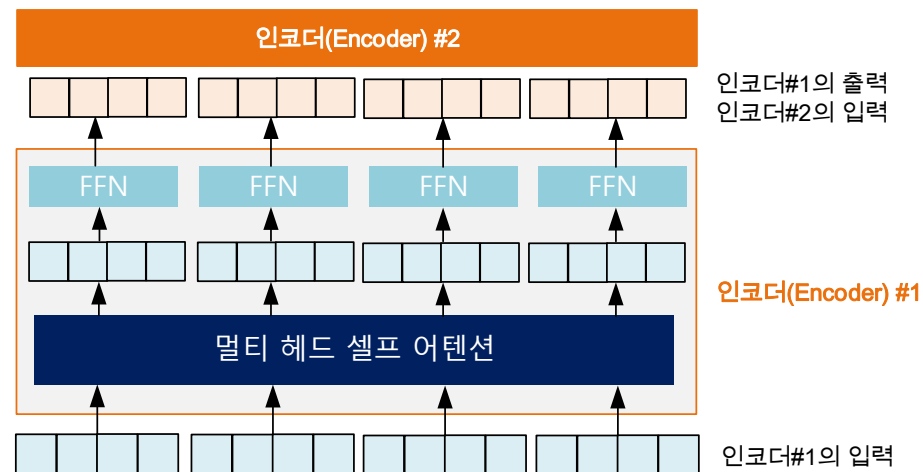
트랜스포머 모델 소개

- 인코더: 구성요소 (1/3)

- 피드포워드 네트워크 (FFN, Feed-Forward Networks)
 - 어텐션으로부터 파악한 정보를 각 토큰에 대해 독립적으로 비선형 변환하는 신경망 구조
 - 비선형 변환은 활성화 함수를 이용해서 이루어짐
 - 활성화 함수를 통한 비선형성을 통해 복잡한 관계와 특징 학습
- 수식 표현

- W_1/b_1 : 첫 번째 선형 레이어 가중치/편향
- W_2/b_2 : 두 번째 선형 레이어 가중치/편향

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

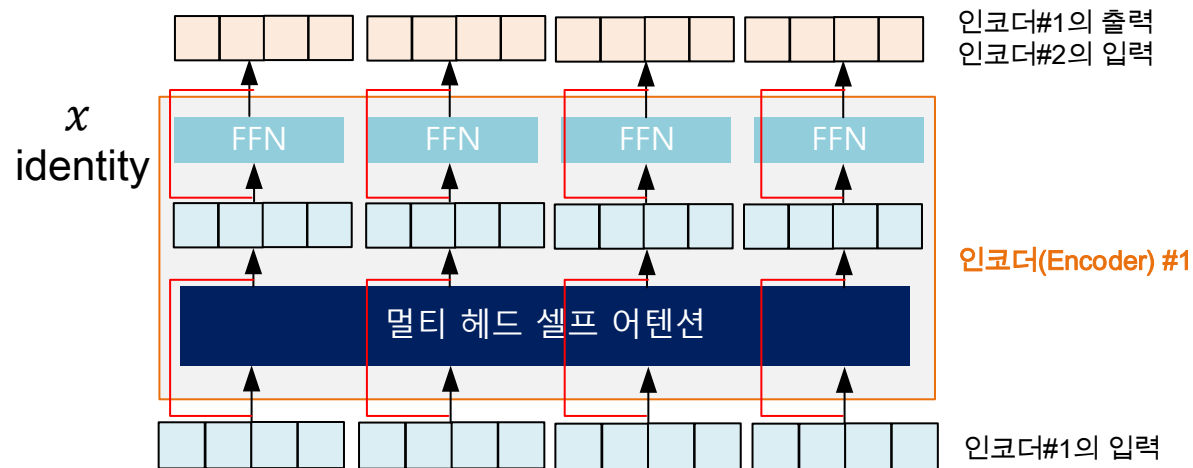
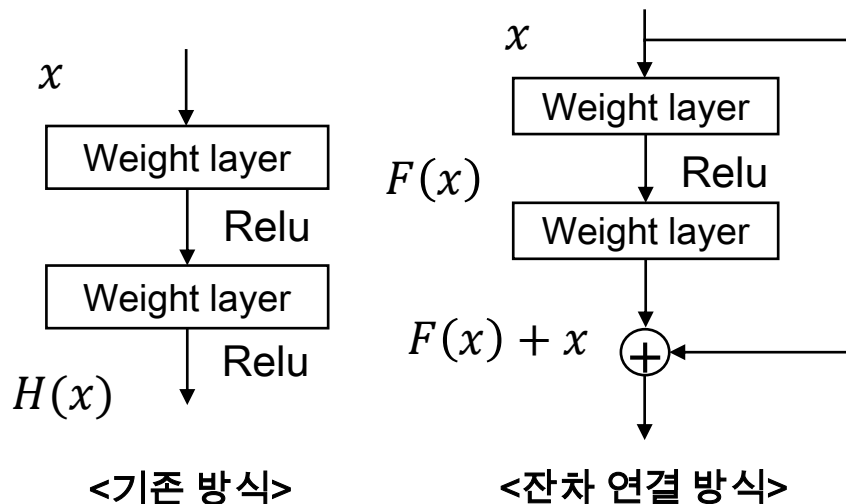


트랜스포머 모델 소개

- 인코더: 구성요소 (2/3)

- 잔차 연결 (Residual Connection)

- 각 층의 입력을 해당 층의 출력에 그대로 더하는 구조
- 네트워크가 깊어졌을 때 발생하는 정보 손실을 줄이고 기울기 소실 문제를 완화함



트랜스포머 모델 소개

• 인코더: 구성요소 (3/3)

• 레이어 정규화 (Layer Normalization)

- 딥러닝 모델 내부의 은닉층 값들을 일정한 범위 안에 유지하여 학습을 안정적으로 만드는 정규화 기법
 - 학습 중 출력값 분포의 변동성이 크면 학습이 불안정해지기 때문에 필요
- 수식 표현

① 입력 벡터의 평균과 표준편차 계산

$$\mu = \frac{1}{d} \sum_{i=1}^d x_i \quad \sigma = \sqrt{\frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2}$$

② 정규화

$$\hat{x} = \frac{(x - \mu)}{\sigma}$$

- | | |
|-----------------------|------------------------------------|
| • μ : 평균 | • γ : 스케일을 조절하는 학습 가능한 파라미터 |
| • σ : 표준편차 | • β : 위치를 조절하는 학습 가능한 파라미터 |
| • $\hat{x}$: 정규화된 벡터 | |
| • d : 벡터 차원 | |

③ 최종 변환

$$\text{LayerNormalization} = \gamma \hat{x} + \beta$$

- 정규화를 하면 모든 값을 일정한 분포로 맞추게 되나, 어떤 벡터는 값을 증가시키거나 줄이는 조절 작업이 필요함
- γ, β 는 학습 과정에서 함께 학습되는 파라미터로, 정규화 이후에도 모델이 필요한 표현력을 잃지 않도록 보정함
 - 표현력: 정규화로 인해 표현 방식이 제한되지 않고, 모델이 학습을 통해 각 특징의 상대적인 중요도와 기준값을 다시 조절할 수 있는 능력

트랜스포머 모델 소개

- 디코더: 구성요소 (1/2)

- 마스크드 멀티헤드 셀프 어텐션

- 현재 시점의 단어를 예측하고자 할 때, 입력 문장으로부터 미래 시점의 단어까지도 참고할 수 있는 현상을 보완
 - 모델이 예측하려는 다음 단어를 사전에 알고있다면, 데이터 유출에 해당함
- 아직 등장하지 않은 미래 토큰이 해당하는 위치에 음의 무한대($-\infty$)로 대체하는 마스크 추가

	<sos>	Je	suls	étudiant
<sos>		$-\infty$	$-\infty$	$-\infty$
Je			$-\infty$	$-\infty$
suls				$-\infty$
étudiant				

트랜스포머 모델 소개

- 디코더: 구성요소 (2/2)

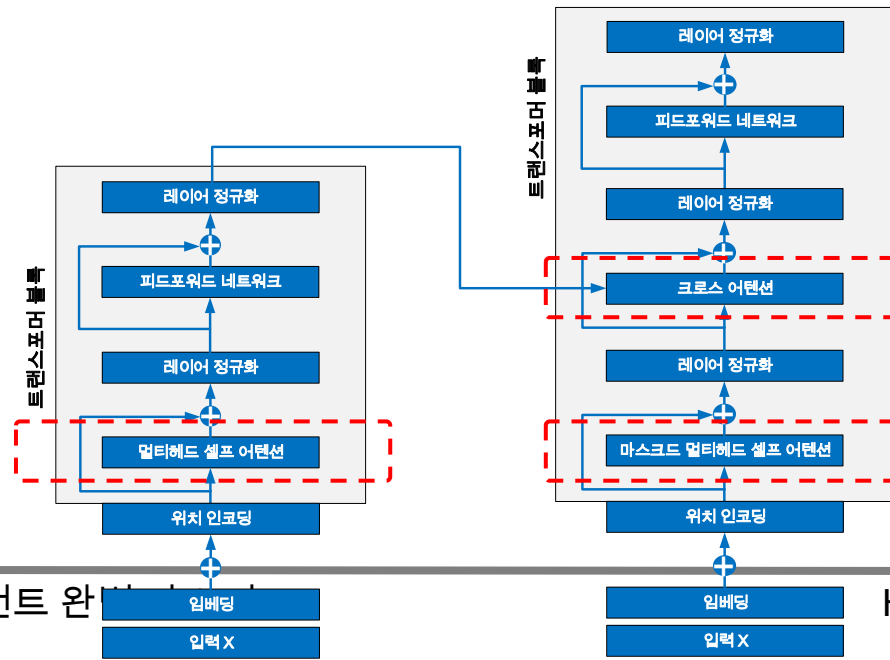
- 크로스 어텐션 (Cross-Attention)

- 셀프 어텐션과 달리, 인코더와 디코더 양쪽의 정보를 모두 활용하는 어텐션 메커니즘
 - Q, K, V 의 관계

인코더의 첫번째 어텐션 – Query, Key, Value: 인코더 행렬

디코더의 첫번째 어텐션 – Query, Key, Value: 디코더 행렬

디코더의 두번째 어텐션 - Query : 디코더 행렬 / Key, Value : 인코더 행렬

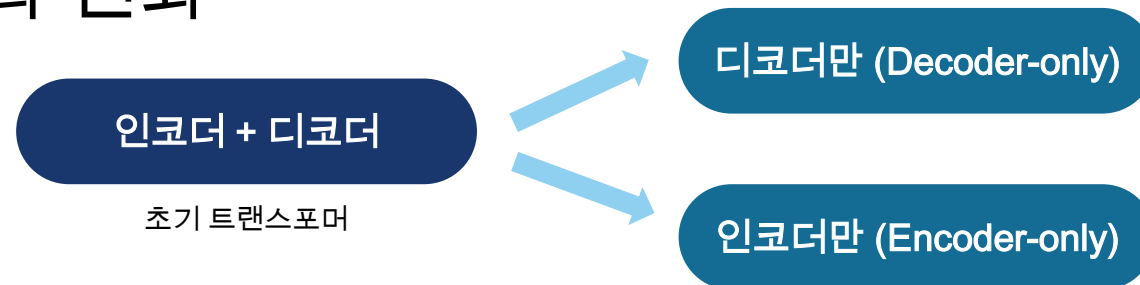


트랜스포머 학습

- 자기지도(self-supervised) 학습

- 관리자가 정답(label)을 일일이 만들어 학습시키지 않아도, 모델이 입력 데이터 자체에서 정답을 만들어 학습하는 방식

- 모델 구조의 변화



- 인과 언어 모델(Decoder-only)

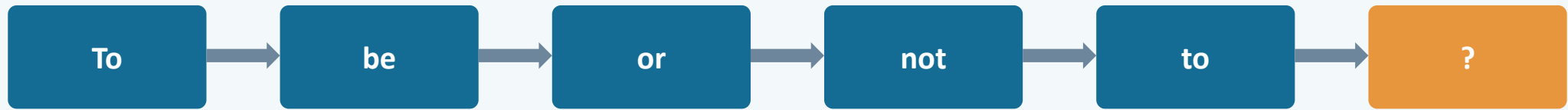
- 이전 토큰을 기반으로 다음 토큰을 생성하는 작업
- e.g., 텍스트 생성, LLM

- 양방향 언어 모델(Encoder-only)

- 입력의 의미와 문맥을 이해해야 하는 작업
- e.g., 요약, 감성 분석, 질의응답

트랜스포머 학습

- 언어 모델링 (Language Modeling)
 - 자기지도를 통해 문맥으로 다음 단어가 나올 확률을 추정하는 방식



- 수식 표현

- w_i : i 번째 단어
- $w_{1:i-1}$: 이전까지 등장한 모든 단어(문맥)

$$P(w|h) = P(w_n | w_{1:n-1}) = \prod_{i=1}^n P(w_i | w_{1:i-1})$$

- 예시

- ‘To be or not to’에서 다음 단어가 ‘be’일 확률 $P(\text{be} | \text{To be or not to})$

트랜스포머 학습

- 다음 토큰에 대한 확률 벡터 구하기 (1/4)

- 과정

1. 입력 임베딩 및 위치 인코딩

- 각 토큰을 벡터로 변환

입력 문장: “나는 오늘 학교에”

나는 → [0.2, -0.4, ...]

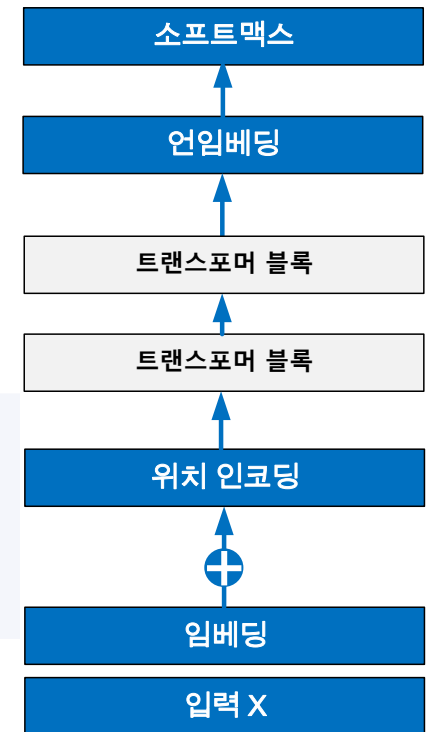
오늘 → [0.7, 0.1, ...]

학교에 → [-0.3, 0.8, ...]

2. 트랜스포머 블록

- 입력 토큰들이 트랜스포머 블록을 통과하며 문맥 정보를 반영한 표현으로 변환

“나는 오늘 학교에 ____”



트랜스포머 학습

• 다음 토큰에 대한 확률 벡터 구하기 (2/4)

• 과정

3. 언임베딩

- 각 토큰에 대해 다음 토큰으로 선택될 가능성을 판단하는 점수인 로짓으로 변환

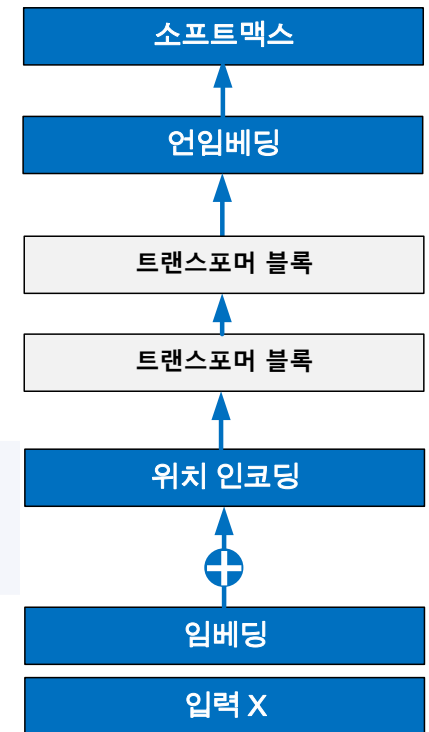
N개의 점수 (N: 어휘 개수): $[-1.2, 0.8, 3.7, -0.4, 1.2, \dots]$

↑
로짓(Logits) 벡터

4. 소프트맥스 연산

- 로짓을 $\text{softmax}()$ 에 넣어 확률 분포로 변환

갔다	0.72
간다	0.15
있다	0.05
먹었다	0.02
...	



트랜스포머 학습

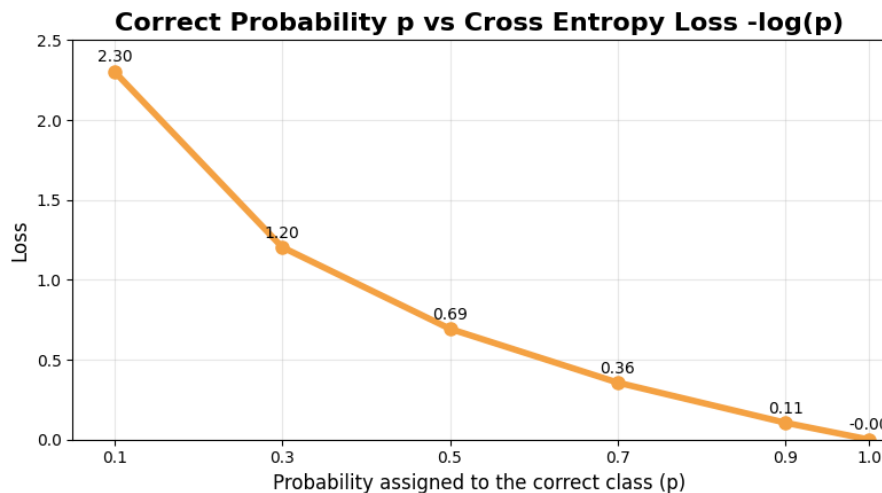
• 다음 토큰에 대한 확률 벡터 구하기 (3/4)

5. 교차 엔트로피 손실 계산

- 모델이 예측한 다음 단어의 확률과 실제 다음 단어의 확률 간 차이를 최소화 하기 위해 활용하는 손실 함수
- 수식 표현

$$L_{CE} = - \sum_{w \in V} y_t[w] \log \hat{y}_t[w]$$

- V : 모델이 알고 있는 전체 단어(토큰)의 집합
- $y_t[w]$: 현재 위치 t 에서의 정답 벡터 (원핫 벡터)
- $\hat{y}_t[w]$: 모델이 예측한 확률



I	eat	an	apple	the
0	0	0	1	0

$$L_{CE} = -\log P(\text{apple})$$

모델이 정답에 높은 확률을 부여하면 손실은 작아지고,
낮은 확률을 부여하면 손실이 커짐

트랜스포머 학습

- 다음 토큰에 대한 확률 벡터 구하기 (4/4)
- 교사 강요 (Teacher Forcing)
 - 디코더에 입력을 제공하기 위해 예측 결과 대신 실제 정답 시퀀스를 다음 입력으로 강제로 사용함
 - 교사 강요를 사용하지 않는다면 오류가 전파될 수 있음

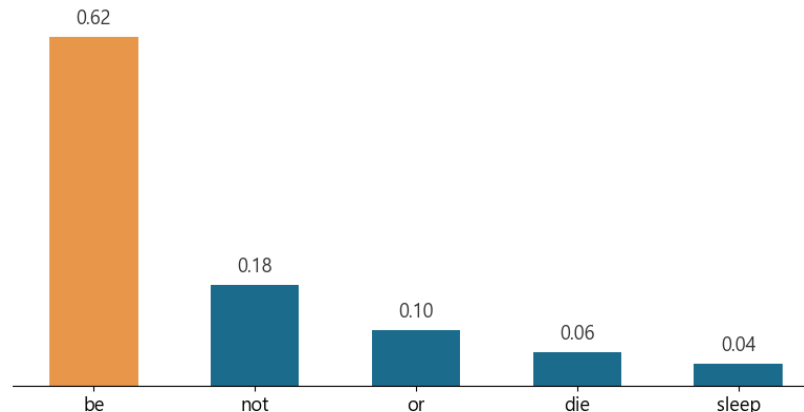


트랜스포머 학습

- 다음 토큰 선택하기 (1/2)
- 그리디 디코딩 (Greedy Decoding)
 - 확률 벡터를 얻은 후 주어진 문맥으로부터 다음 단어를 예측할 때 가장 확률이 높은 단어를 선택하는 방식

$$w_i = \operatorname{argmax}_{w \in V} P(w \mid w <_i)$$

- 특징
 - 결과가 예측 가능하고 정형화됨
 - 확률이 가장 높은 단어 선택이 항상 좋다고 보장할 수 없음



<다음 단어 확률 분포 예시>

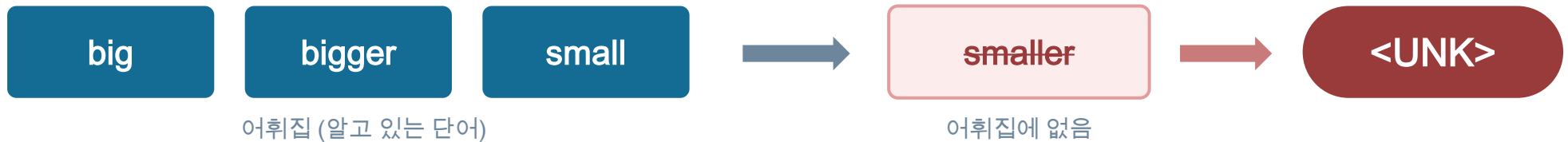
트랜스포머 학습

- 다음 토큰 선택하기 (2/2)
 - 비결정적 샘플링 (Decoding)
 - 그리디 디코딩의 한계를 보완한 다음 토큰 선택 방식
 - 디코딩, 자기회귀 생성, 인과언어 모델링이라고도 함
 - 샘플링 기법 종류

샘플링 기법	핵심 원리	특징
랜덤 샘플링	다음 토큰을 전체 확률분포에서 무작위로 선택	드물고 특이한 단어가 선택되어 문장이 부자연스러울 수 있음
Top-k 샘플링	확률 상위 k 개 후보만 남기고 재정규화 후 선택	k 값으로 후보 범위를 직접 통제
Top-p 샘플링	누적 확률이 특정 비율 p 에 도달할 때까지 후보 선택	Top-k보다 유연하게 다양한 단어 선택 가능
온도 샘플링	소프트맥스 전 로짓을 온도 $t(0\sim 1)$ 로 나눔	$t \rightarrow 0$: 0에 가까울수록 가장 확률이 높은 단어의 확률이 1에 가까워짐 $t > 1$: 덜 예측 가능한 결과를 원하는 경우 설정

트랜스포머 학습

- 미지 단어 문제 (Unknown Word Problem)
- 고정된 어휘집의 한계
 - 학습 데이터에 없는 단어(e.g., 신조어, 오타자)를 접하면 토큰 <UNK>를 할당
 - <UNK>가 많아지면 실제 정보가 손실되어 모델이 문맥을 제대로 이해하지 못하게 됨



트랜스포머 학습

- 미지 단어 문제: 서브워드 단위의 토큰화
 - 형태소·문법 규칙을 단어보다 더 작은 단위로 나누어 토큰라이저가 인식할 수 있게 한다면 정교한 처리와 일반화 가능
- 바이트 페어 인코딩 (BPE, Byte-Pair Encoding) (1/2)
 - 현대 LLM에서 가장 널리 사용되는 서브워드 토큰화 방법
 - 특징
 - 어휘집에 10만 개의 토큰만 있어도 조합을 통해 수백만 개의 단어 표현 가능
 - 어휘집에 없는 단어도 이미 있는 서브워드의 조합으로 만들 수 있음

트랜스포머 학습

- 미지 단어 문제: 서브워드 단위의 토큰화
- 바이트 페어 인코딩 (BPE) (2/2)
 - 흔한 단어와 드문 단어의 적용
 - 흔한 단어의 경우에는 단어 전체를 어휘집에 추가함
 - e.g., “the”는 등장 빈도가 높으므로 어휘집에 통째로 추가
 - 드문 단어의 경우에는 단어 전체를 어휘집에 추가하는 대신, 단어를 분리하여 어휘집에 추가함
 - e.g., “react”는 등장 빈도가 비교적 낮으므로, “re”와 “act”로 분리하여 어휘집에 추가

트랜스포머 학습

- 기존의 인과 언어 모델
 - 텍스트 생성에 특화되어 문장을 왼쪽에서 오른쪽으로 순차적으로 읽음
 - 특정 단어를 처리할 때 그 오른쪽에 있는 단어를 알 수 없음
- 양방향 모델 (BERT, Bidirectional Encoder Representations from Transformers)
 - 주어진 문장 전체를 이해하는 작업이 필요할 때 활용

문장: “나는 오늘 영화를 봤다.”

영화를 -> “나는 오늘”, “봤다”

구분	인과 언어 모델 (Causal LM)	양방향 언어 모델 (BERT)
구조	디코더	인코더
문맥 참조	왼쪽(이전) 문맥만	양쪽 문맥 모두
대표 작업	텍스트 생성	질의응답, 요약, 번역 등 이해·분류 작업

트랜스포머 학습

- 마스크드 언어 모델링
 - 입력 문장의 일부 단어를 [MASK]로 가린 후, 앞뒤 문맥을 이용하여 가려진 단어를 예측하는 자기지도 학습 방식
- 특징
 - 문장 내의 토큰을 무작위로 마스킹함
- 동작 방식
 - 일부 토큰을 마스크로 가리고, 나머지 문맥으로 해당 토큰을 예측
 - 전체 문맥(앞+뒤)이 주어진 상태에서 <MASK> 자리에 올 단어를 예측

나는

오늘

<MASK>

를

봤다

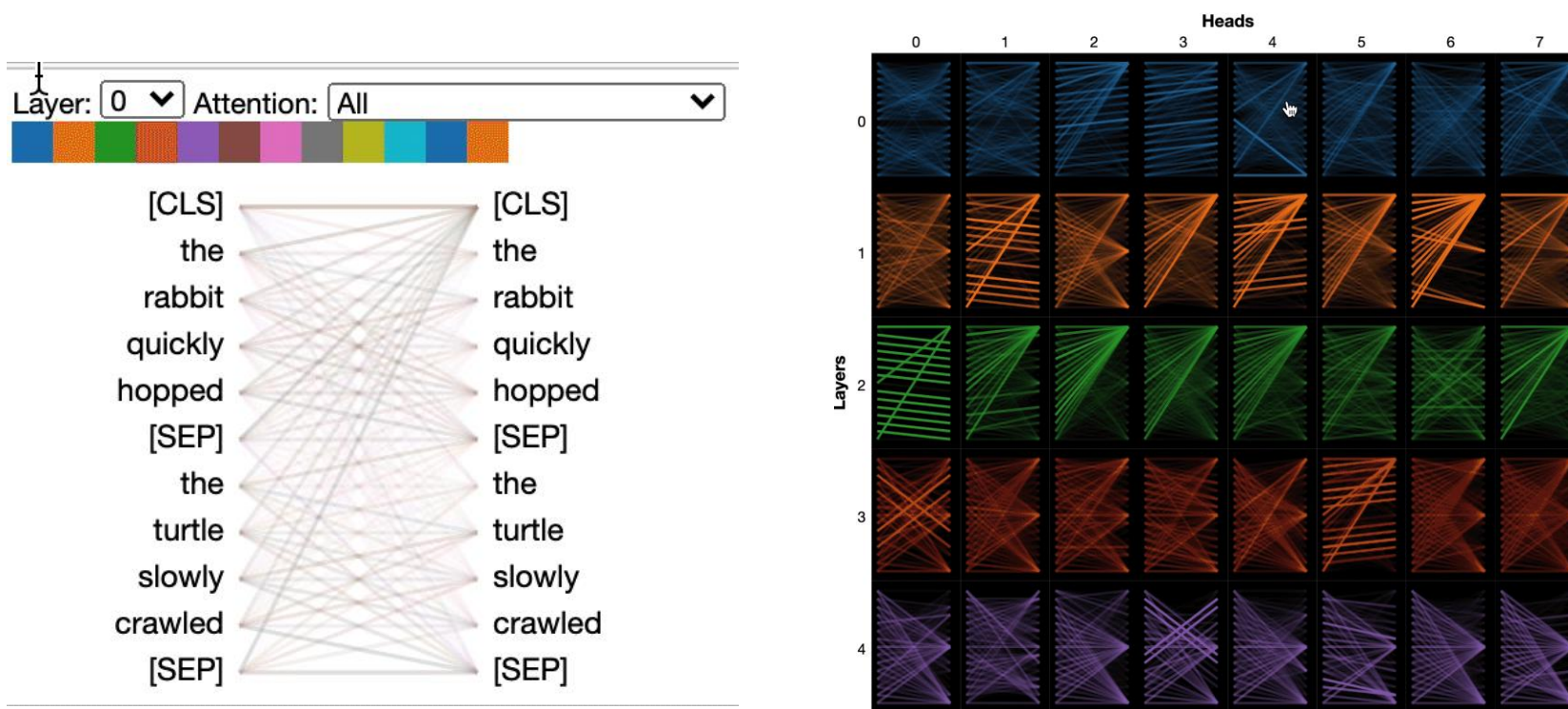
트랜스포머 학습

- 입력 구분을 위한 특수 토큰
 - [CLS]: 모든 입력의 맨 앞에 붙는, 입력 시작을 알리는 토큰
 - [SEP]: 입력 내에서 서로 다른 문장을 분리하는 토큰

"[CLS] 영화는 언제 개봉했나요? [SEP] 2023년에 개봉했습니다. [SEP]"

내부 메커니즘 시각화

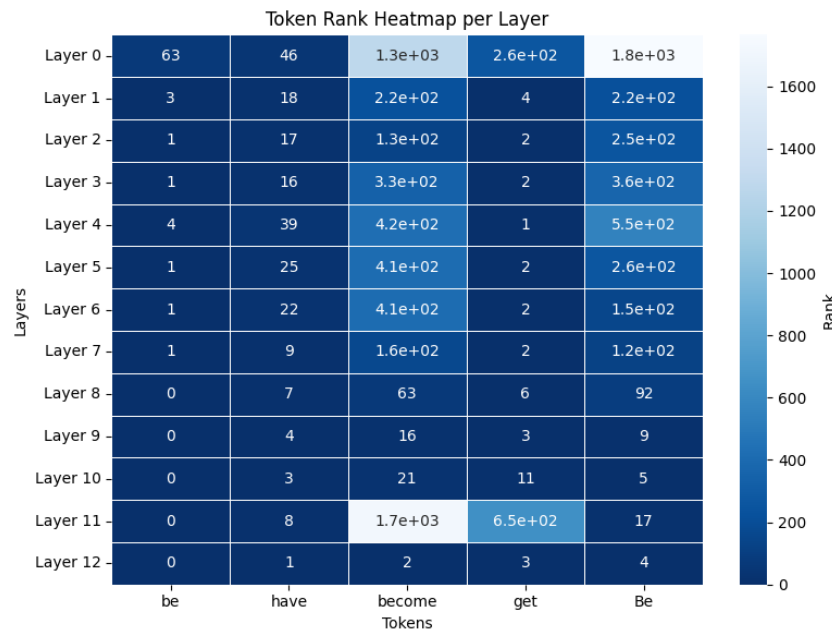
- 어텐션 (Attention) 가중치 분석
 - 각 단어가 어떤 단어와의 관계를 참고하는지 확인
 - 선이 진할수록 어텐션 가중치가 높음



<출처> <https://github.com/jessevig/bertviz>

내부 메커니즘 시각화

- 레이어별 다음 토큰 예측 순위(Rank) 변화
 - 초기 레이어에서는 단순한 특징을, 후반 레이어에서는 의미 중심의 특징을 학습
 - 레이어를 거칠수록 다음 토큰 후보의 순위가 점차 안정적으로 변화



1. 세로축: 트랜스포머 레이어
2. 가로축: 마지막 레이어에서 선택된 상위 후보 토큰
3. 히트맵 상의 숫자: 각 토큰의 예측 순위

<GPT-2에서 "To be or not to"를 입력했을 때의 토큰 예측 히트맵>

내부 메커니즘 시각화

- 뉴런 활성화 패턴 분석

- 비음수 행렬 분해(NMF, Non-negative matrix factorization)를 활용하여 뉴런을 여러 활성화 요소로 분해하여 분석
 - 비음수 행렬 분해: 음수가 아닌 값으로 이루어진 행렬을 여러 개의 작은 행렬로 분해하는 방법
- 토큰 생성 과정에서 어떤 요소가 활성화되는지 확인 가능

<소수의 활성 요소 분석>



<여러 활성 요소 분석>



트랜스포머 활용

- 사전학습, 전이학습
 - 사전학습(pre-training) 단계
 - 방대한 텍스트로부터 언어의 일반적 구조 규칙을 학습
 - 전이학습(transfer learning) 단계
 - 사전 학습된 능력을 원래 목적과 다른 새 작업에 활용
 - 파인튜닝(fine tuning)
 - 사전 학습된 모델이 익힌 일반 지식을 특정 작업에 맞게 조정
 - 모델의 상단에 새 파라미터 집합(한 두개의 레이어)을 추가하고, 해당 파라미터만 경사하강법으로 학습
 - 모델의 내부 구조를 훼손하지 않음

트랜스포머 활용

- 파인튜닝 과정

1. 마지막 레이어 제거

- 원래 작업에 특화된 최종 레이어를 잘라냄

2. 새 레이어 추가

- 무작위로 초기화된 새 레이어를 상단에 추가

3. 학습 데이터 통과

- 새 작업 예제가 동결된 모델 전체를 순전파로 통과

4. 새 레이어만 업데이트

- 역전파를 통해 새로 추가한 레이어의 가중치만 학습

트랜스포머 활용

- 파인튜닝 과정

- Decoder-only 모델의 파인튜닝을 위한 데이터 구성
 - 디코더 기반 모델에서는 인코더가 없어 입력과 출력을 하나의 시퀀스로 구성해야 함
 - 예시
 - 번역 작업

소스 (Source)

I am a student

구분자 (Delimiter)

<to-fr>

타겟 (Target)

Je suis étudiant

- 요약 작업

소스 (Source)

article

구분자 (Delimiter)

<summarize>

타겟 (Target)

summary

트랜스포머 활용

- 지식 증류(Knowledge Distillation) (1/2)
 - 대규모 모델의 지식을 흡수하여 특정 작업에 활용하는 기법
 - 주요 개념
 - 교사 모델(Teacher Model)
 - 지식을 전달하는 모델
 - 학생 모델(Student Model)
 - 교사 모델의 로짓을 모방하도록 학습한 경량 모델

트랜스포머 활용

• 지식 증류(Knowledge Distillation) (2/2)

• 과정

1. 데이터 전처리

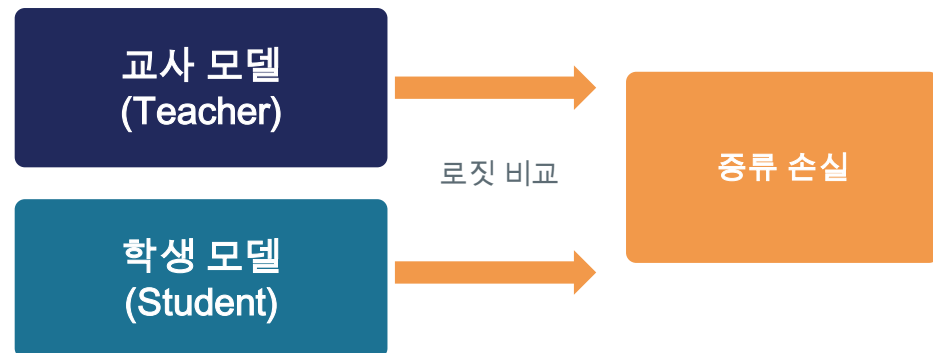
- 각 모델에 적합한 토큰라이저를 활용해 전처리

2. 모델 파인튜닝

- 모델을 특정 작업에 맞게 파인튜닝해 교사 모델을 생성

3. 학생 모델 생성

- 교사 모델보다 작은 사전학습 모델을 학생 모델로 학습
- 증류 손실(KL Divergence) 함수를 이용해 교사 모델의 로짓과 학생 모델의 로짓을 비교해 차이를 계산



트랜스포머 활용

• 파인튜닝 vs 지식 증류

구분	파인튜닝	지식 증류
적합한 상황	학습 데이터가 적어 처음부터 학습하기 어려운 경우	계산 자원이 제한된 상황에서 경량 모델이 필요한 경우
핵심 아이디어	사전 학습된 모델에 새 레이어만 학습	큰 교사 모델의 로짓을 작은 학생 모델이 모방

Thanks!

이 하 늘(haneul@pel.sejong.ac.kr)

부록#1

• 50 x 128 크기 위치 인코딩 시각화 (1/2)

```
1. import tensorflow as tf
2. import numpy as np
3. import matplotlib.pyplot as plt

4. class PositionalEncoding(tf.keras.layers.Layer):

5.     def __init__(self, position, d_model):
6.         super(PositionalEncoding,
7. self).__init__()
8.         self.pos_encoding =
9. self.positional_encoding(position, d_model)

10.     def get_angles(self, position, i, d_model):
11.         angles = 1 / tf.pow(
12.             10000,
13.             (2 * (i // 2)) / tf.cast(d_model,
14. tf.float32))
15.         return position * angles

16.     def positional_encoding(self, position,
17. d_model):
18.         angle_rads = self.get_angles(
19.             position=tf.range(position,
20. dtype=tf.float32)[:], tf.newaxis],
```

```
21. i=tf.range(d_model, dtype=tf.float32)[tf.newaxis, :],
22.         d_model=d_model
23.     )

24. sines = tf.math.sin(angle_rads[:, 0::2])
25. cosines = tf.math.cos(angle_rads[:, 1::2])

26.     angle_rads = np.zeros(angle_rads.shape)
27.     angle_rads[:, 0::2] = sines
28.     angle_rads[:, 1::2] = cosines

29.     pos_encoding = tf.constant(angle_rads)
30.     pos_encoding = pos_encoding[tf.newaxis, ...]

31.     print(pos_encoding.shape)

32.     return tf.cast(pos_encoding, tf.float32)

33.     def call(self, inputs):
34.         return inputs +
35. self.pos_encoding[:, :tf.shape(inputs)[1], :]
```

부록#1

- 50 x 128 크기 위치 인코딩 시각화 (2/2)

```
31. sample_pos_encoding = PositionalEncoding(50, 128)
32. plt.pcolormesh(
33.     sample_pos_encoding.pos_encoding.numpy()[0],
34.     cmap='RdBu'
35. )
36. plt.xlabel('Depth')
37. plt.xlim((0, 128))
38. plt.ylabel('Position')
39. plt.colorbar()
40. plt.show()
```

부록#2

• GPT-2에서 “To be or not to”를 입력했을 때의 토큰 예측 히트맵 (1/2)

```
1. import torch
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. import seaborn as sns

5. from transformers import GPT2Tokenizer,
   GPT2LMHeadModel

6. model_name = "gpt2"# 모델 불러오기

7. tokenizer =
   GPT2Tokenizer.from_pretrained(model_name)
8. model = GPT2LMHeadModel.from_pretrained(
9.     model_name,
10.    output_hidden_states=True
11. )

12. model.eval()

13. text = "To be or not to"# 입력 문장

14. inputs = tokenizer(text, return_tensors="pt")

15. with torch.no_grad(): # Forward
16.     outputs = model(**inputs)

17. hidden_states = outputs.hidden_states
```

```
18. final_hidden = hidden_states[-1]
   # 마지막 레이어에서 Top5 후보 선택

19. final_logits =
   model.lm_head(final_hidden)

20. final_probs = torch.softmax(
21.     final_logits[:, -1, :],
22.     dim=-1
23. )

24. topk = torch.topk(final_probs, k=5)

25. candidate_ids = topk.indices[0].tolist()

26. candidate_tokens = [
27.     tokenizer.decode([i]).strip()
28.     for i in candidate_ids
29. ]

30. print("Top5 Tokens")
31. print(candidate_tokens)
```

부록#2

• GPT-2에서 “To be or not to”를 입력했을 때의 토큰 예측 히트맵 (2/2)

```
32. rank_matrix = [] # 각 레이어마다 Rank 계산
33. for hidden in hidden_states:
34.     logits = model.lm_head(hidden)
35.     probs = torch.softmax(
36.         logits[:, -1, :],
37.         dim=-1
38.     )
39.     sorted_ids = torch.argsort(
40.         probs,
41.         descending=True)[0]
42.     ranks = []
43.     for token_id in candidate_ids:
44.         rank = (
45.             (sorted_ids == token_id)
46.             .nonzero(as_tuple=True)[0]
47.             .item()
48.         )
49.         ranks.append(rank)
50.     rank_matrix.append(ranks)
```

```
51. df = pd.DataFrame(
52.     rank_matrix,
53.     columns=candidate_tokens
54. ) # DataFrame
55. df.index = [
56.     f"Layer {i}"
57.     for i in range(len(df))
58. ]
59. print(df)
60. plt.figure(figsize=(8,6)) # Heatmap
61. sns.heatmap(
62.     df,
63.     annot=True,
64.     cmap="Blues_r",
65.     linewidths=0.5,
66.     cbar_kws={"label":"Rank"}
67. )
68. plt.title("Token Rank Heatmap per Layer")
69. plt.xlabel("Tokens")
70. plt.ylabel("Layers")
71. plt.tight_layout()
72. plt.show()
```