Attention

〈배경〉 - Machine Translation Task

- 개요
  - Input : Sequence of words (번역할 문장들)
  - Model : 번역 모델
  - Output : Sequence of words (번역된 문장들)

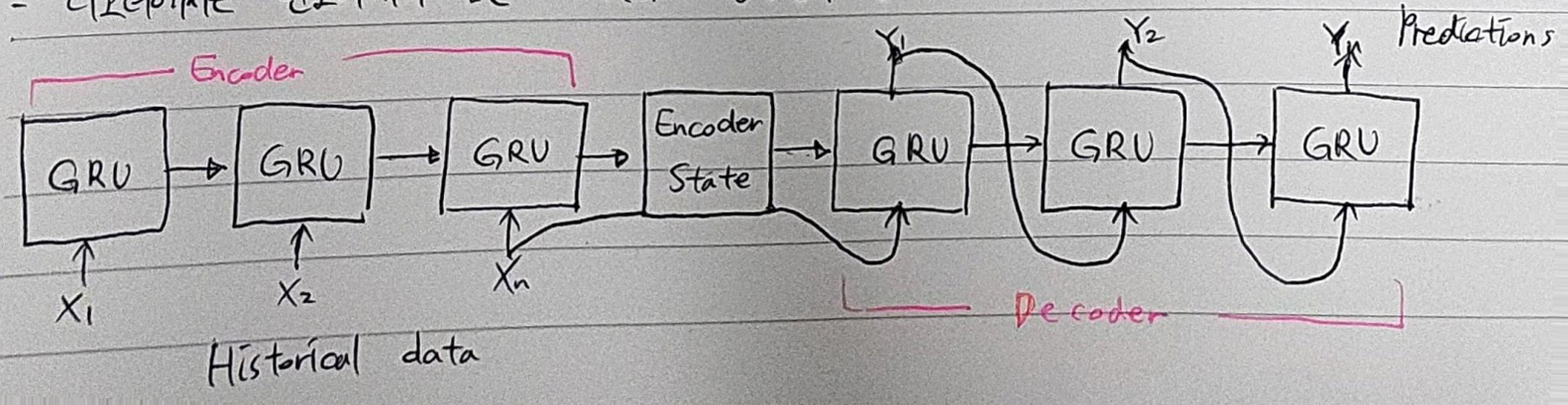
- Word Embedding
  - 텍스트는 이미지처럼 수치적인 값으로 표현되는 데이터가 아님
  - 단어를 수치적인 자료 (ex. 벡터)로 표시해야 하는데 이를 word embedding 이라고 함
  - Word embedding 의 방법은 목적이나 모델에 기반하여 매우 다양함
    ↳ 똑같은 단어라도 다른 모델에 따라 임베딩된 벡터가 다름
  - 대용량의 데이터로 잘 학습된 모델의 word embedding 결과를 가져다 사용

⊕ 희소표현 (Sparse Representation) : 벡터 또는 행렬의 값이 대부분이 0으로 표현되는 방법
  ↳ 공간적 낭비 발생

⊕ 밀집 표현 (~~Density~~ Dense Representation) : 0과 1만이 아니라 실수값을 적용 → 벡터의 차원 ↓
  ↳ 단어를 밀집 표현으로 나타내는 방법을 word embedding 이라고 함

ex. Word → Word Embedding → Dimensionality Reduction → Visualization of word embeddings in 2D

- Seq2Seq
  - LSTM or GRU cell로 구성된 인코더 - 디코더 구조의 신경망 모형
  - Input : Sequence of text ($\mathbb{R}^{T \times d_w}$ 행렬)
  - Output : Sequence of text ($\mathbb{R}^{T' \times d_w}$ 행렬)
  - 인코더에서 번역할 문장을 차례대로 입력받아 디코더로 넘길 피처 (feature, $h \in \mathbb{R}^{1 \times d_m}$)를 학습함
  - 디코더에서는 인코더에서 넘긴 피처 (h)를 입력받아 순차적으로 번역 단어를 생성함



〈Attention〉

- Seq2Seq

1. T개의 단어로 이루어진 Input 시퀀스 ($\in \mathbb{R}^{T \times d_w}$)
2. 하나의 벡터 $h \in \mathbb{R}^{1 \times d_m}$인 인코더의 결과물
3. ($T \rightarrow 1$)의 압축. 즉 하나의 고정된 크기의 벡터에 모든 정보를 압축하려고 하니까 정보 손실이 발생하고, 정확도가 떨어짐

⇒ 해결책 : T시점의 모든 정보를 다 사용하되 특정 시점들에 집중 → Attention

↳ 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라, 해당 시점에서 예측해야할 단어와 연관이 있는 부분을 좀 더 집중해서 보는 것

↳ 번역이라는 작업의 특수성에서 발생된 아이디어

- Attention Score

- non-normalized attention ~~score~~ weight $\xrightarrow{\text{softmax}}$ attention weight
- 디코더의 t시점을 기준으로 인코더 시점들의 attention score 계산

q_t : 디코더 t시점의 LSTM cell의 hidden state 벡터

k_i, $i \in \{1, \cdots, T\}$: 인코더 i시점의 LSTM cell의 hidden state 벡터

$q_t^T k_i$: attention score (스칼라)

$s_t = [q_t^T k_1 \cdots q_t^T k_T]^T$: 디코더의 t시점과 인코더의 모든 시점의 attention score 벡터

$w_t = \text{softmax}(s_t) \in \mathbb{R}^T$: 디코더의 t시점과 인코더의 모든 시점의 attention weight 벡터 → 집중 비율

$w_t^T 1 = 1$ (가중치의 합 = 1)

- attention score ($q_t^T k_i$)는 두 벡터간의 내적으로 비슷할수록 큰 값을 가짐
- 디코더의 t ~~[illegible]~~ 시점의 q_t와 비슷할수록 인코더의 i시점은 큰 가중치를 가짐

• Attention value

- 인코더의 hidden state와 attention weight의 가중 평균
- 디코더의 t시점을 기준으로 인코더 시점과의 attention value 계산

$$\begin{cases} K = [k_1, \cdots, k_T]^T \in \mathbb{R}^{T \times d_m} & : \text{인코더 hidden state들을 쌓은 행렬} \\ c_t^T = w_t^T K \in \mathbb{R}^{1 \times d_m} & : \text{attention value 벡터 (컨텍스트 벡터라고도 불림)} \end{cases}$$

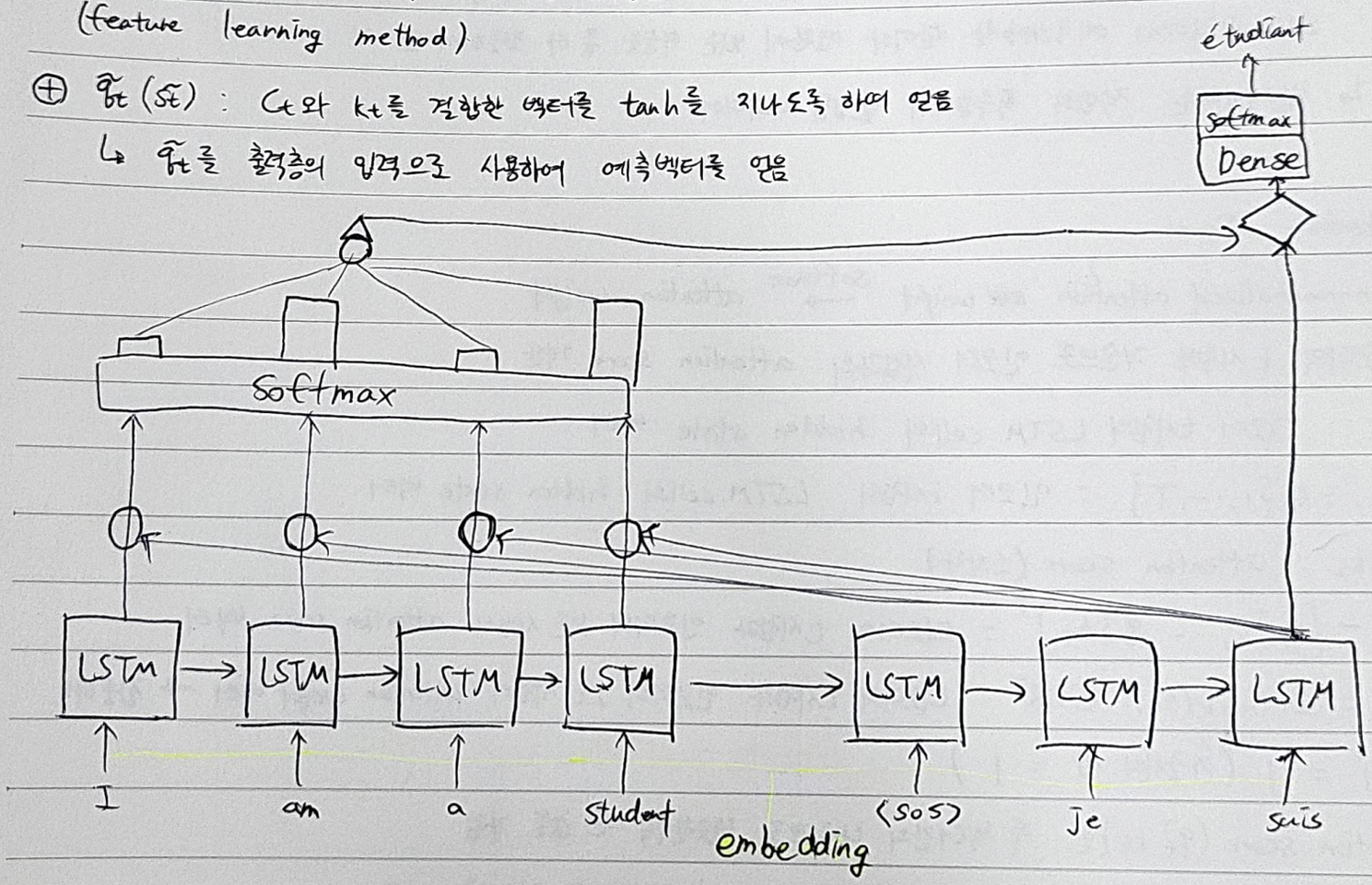
⇒ Attention value는 디코더의 t시점과의 연관성을 기반으로 입력정보를 사용하여 새롭게 생성한 피처

- c_t와 k_t를 결합한 벡터를 가지고 기존의 LSTM과 동일한 과정 수행

* Attention은 input과 output 시점간의 연관성을 고려한 피처학습 방법 (feature learning method)

⊕ $\tilde{q}_t$ ($\tilde{s}_t$) : c_t와 k_t를 결합한 벡터를 tanh를 지나도록 하여 얻음

↳ $\tilde{q}_t$를 출력층의 입력으로 사용하여 예측벡터를 얻음



Seq2Seq의 attention 방법 예시

<Transformer>

• 요약

Seq2Seq 는 인코더와 디코더를 포함하는 복잡한 RNN과 CNN을 기반으로 한다.

그리고 성능이 좋은 모델은 Attention을 이용해서 인코더와 디코더를 연결한다.

이 때 Attention을 RNN과 CNN을 보조하는 용도가 아니라 Attention 만으로 인코더와 디코더를 만들어서 새로운 간단한 네트워크 아키텍처가 바로 Transformer 라고 한다.

• 배경

- LSTM과 Convolution 기반의 모델에서 순차적인 계산량은 너무 많고 비효율적임
- LSTM과 Convolution 층을 버리고 병렬화가 가능한 연산 구조를 가진 모델을 구성
- Self-attention은 병렬화가 가능하고 이미 많은 자연어 처리 분야에서 성공적으로 쓰이고 있었음

• Positional encoding

- Transformer는 기존의 모델들의 순차적인 연산을 비판하면서 그 구조를 과감하게 없앰
  ↳ Positional encoding 이라는 대안을 사용

$$\begin{cases} PE(pos, 2i) = \sin\left(\dfrac{pos}{10000^{2i/d_m}}\right) \\ PE(pos, 2i+1) = \cos\left(\dfrac{pos}{10000^{2i/d_m}}\right) \end{cases}$$

pos : 단어의 시점
i : 단어 임베딩 벡터에서의 인덱스

↳ 이렇게 해주는 이유 : RNN의 경우 단어의 위치에 따라 단어를 순차적으로 입력 받아서 처리하므로 각 단어의 위치 정보를 가질 수 있었음
but 트랜스포머는 순차 입력 X → 단어의 위치 정보를 다른 방식으로 알려줄 필요가 있음

- positional encoding 값으로 만든 행렬을 임베딩된 행렬에 더함
  ↳ 같은 단어라도 위치에 따라 다른 임베딩 벡터를 가짐

• Self - attention

- 서로 다른 input과 output 간의 연관성이 아닌 동일한 문장 안에서 단어들 간의 연관성을 통해 새로운 피처 생성
- Self-attention을 하기 위해서는 하나의 임베딩 행렬(X)이 각각 다른 FFN(Feed-Forward Network)을 거쳐 query(Q), key(K), value(V)를 생성

$$\begin{cases} Q = XW_Q \\ K = XW_K \\ V = XW_V \end{cases}$$

- Q와 K를 가지고 attention score를 계산하고, attention weight를 계산
- V의 attention weight 만큼의 가중평균으로 새롭게 피처 생성

⊕ Attention vs Self-attention

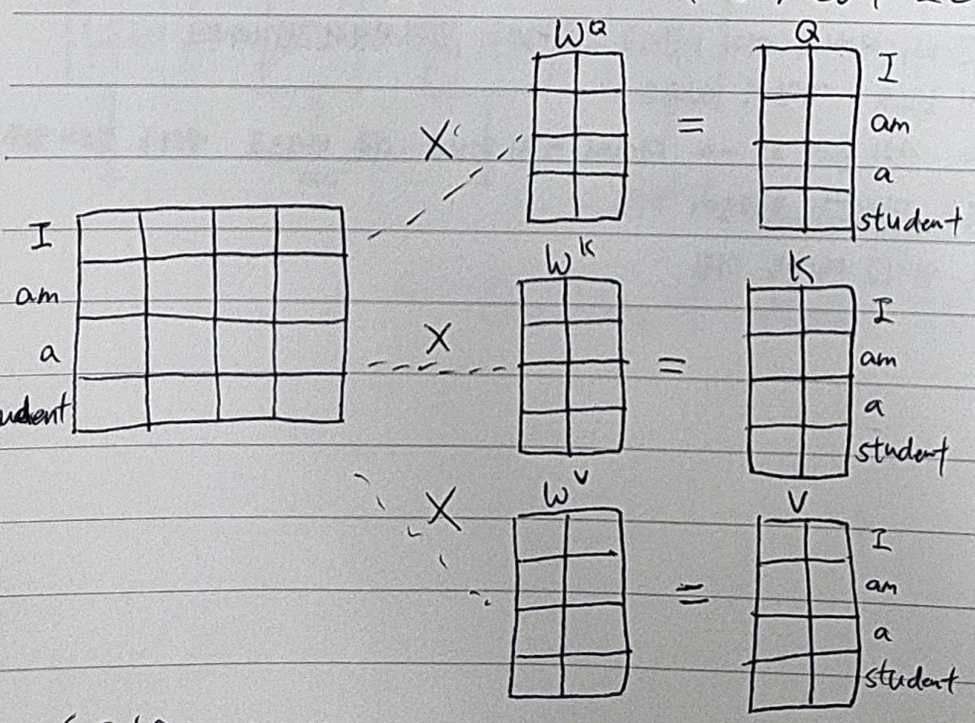
Attention 함수는 주어진 쿼리(Query)에 대해서 모든 키(key)와의 유사도를 각각 구해서 이 유사도를 가중치로 하여 키와 맵핑되어 있는 각각의 값(value)에 반영해주고, 유사도가 반영된 값을 모두 가중합하여 리턴

⇒ Q : t 시점의 디코더 셀에서의 은닉 상태

K : 모든 시점의 인코더 셀에서의 은닉 상태들

V : " " " "

but self-attention에서는 Q, K, V가 모두 "입력 문장의 모든 단어 벡터들"로 동일함



(self-attention에서 query, key, value 예시)

• Seq2Seq 의 Attention 과 Transformer 의 Self-attention 의 차이점

1. Seq2Seq 에서는 디코더 t시점의 attention 연산을 위해서는 t-1 시점까지 attention 연산이 선행되어야 함

→ 디코더의 모든 시점에서 동시에 진행될 수 없어서 t시점 hidden state (벡터)에서만 계산함

2. Transformer 는 순차적인 연산이 없어서 모든 시점에 대해서 동시에 attention을 진행할 수 있음

→ t시점의 벡터만이 아닌 모든 시점의 (1, ---, T) 행렬을 가지고 계산

- Query
  - Attention : 디코더의 t시점의 hidden state (벡터)
  - Self-attention : 모든 시점의 단어 행렬 X와 W_Q의 곱 (행렬)
- Key
  - Attention : 인코더의 모든 시점의 hidden state를 모은 행렬 (행렬)
  - self-attention : 모든 시점의 단어 행렬 X와 W_K의 곱 (행렬)
- Value
  - Attention : 인코더의 모든 시점의 hidden state를 모은 행렬 (행렬)
  - Self-attention : 모든 시점의 단어 행렬 X와 W_V의 곱 (행렬)

• Self-attention 연산 과정

$$\text{Attention}(Q, K, V) = \underbrace{\text{softmax}\Big(\overbrace{\frac{QK^T}{\sqrt{d_m}}}^{\text{attention score}}\Big)}_{\text{attention weight}} V$$

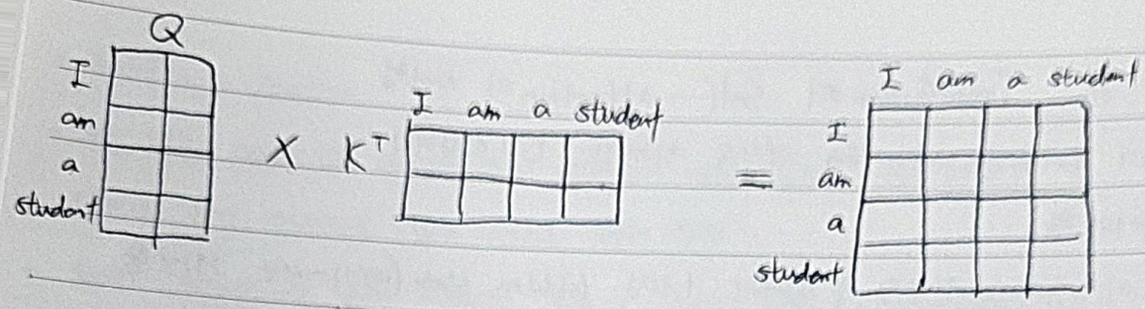
- 여기서 d_m은 행렬곱의 연산값이 너무 커지는 것을 방지하기 위한 조정 상수
- $QK^T/\sqrt{d_m} \in \mathbb{R}^{T\times T} = [s_1, \cdots, s_T]^T$: 모든 시점간의 attention score를 담은 행렬
- $W = \text{softmax}(QK^T/\sqrt{d_m}) = [w_1, \cdots, w_T]^T \in \mathbb{R}^{T\times T}$
- Attention (Q, K, V)의 첫번째 행 : $c_1 = \sum_{t=1}^{T} w_{1,t} v_t$, $\sum_{t=1}^{T} w_{1,t} = 1$

• Self-attention 의 의미

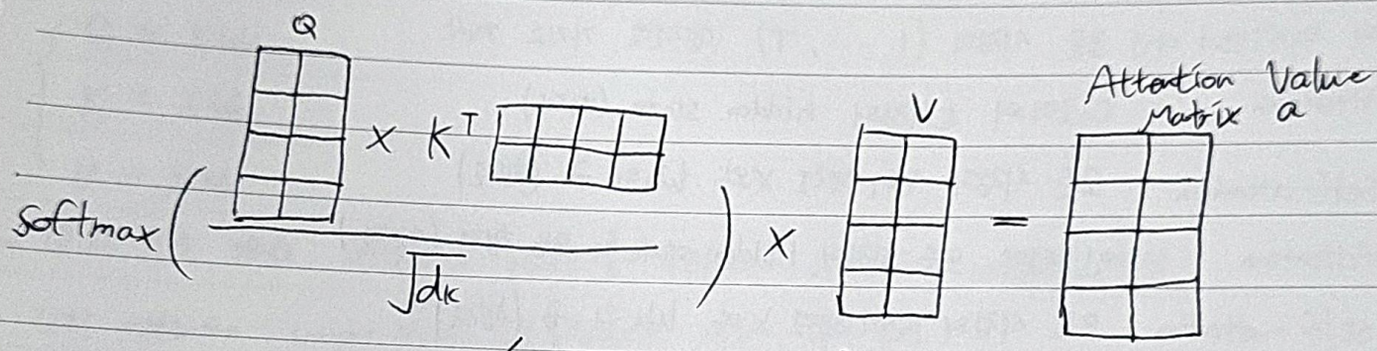
- 내부 단어들간의 연관성 (문맥)을 고려한 피처 생성

ex. Tom likes a dog. He also likes a cat.

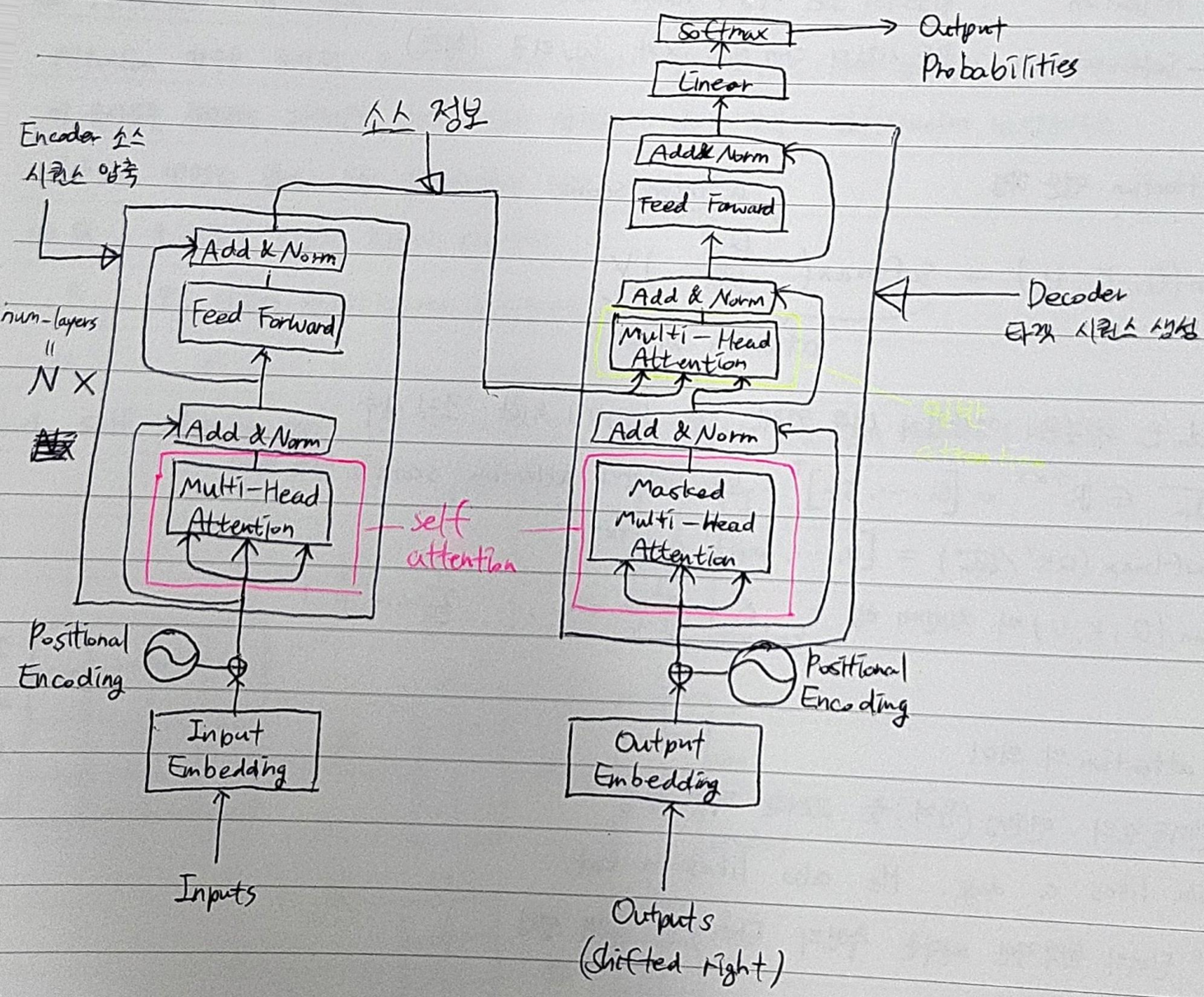
⇒ "He" 단어의 문맥적인 의미를 주변의 단어들을 통해 생성



(Self-attention에서 attention score 예시)



(attention 연산 과정)



• Multihead attention

- Attention 병렬화
- Attention 연산은 합결을 임베딩 차원의 방향으로 나눠서 연산 수행 가능
- 나눠서 연산을 수행한 다음 합쳐서 다음 작업 수행

〈 관련 연구 〉

• Attention 방법은 새로운 피처생성 방법으로 기계 번역 뿐만 아니라 비전, 시계열 예측 등의 방법에서도 많이 사용됨

1. On the Relationship between Self-attention and Convolutional Layers (ICLR)
   - Self-attention이 convolution 연산을 포함하는 연산임을 밝힌 연구
2. Set Transformer (ICML)
   - Input으로 집합을 받아 예측을 수행하는 Transformer 모델
3. MQTransformer (Arxiv)
   - 시계열 예측을 위한 Transformer 모델