Transformer : Attention Is All You Need

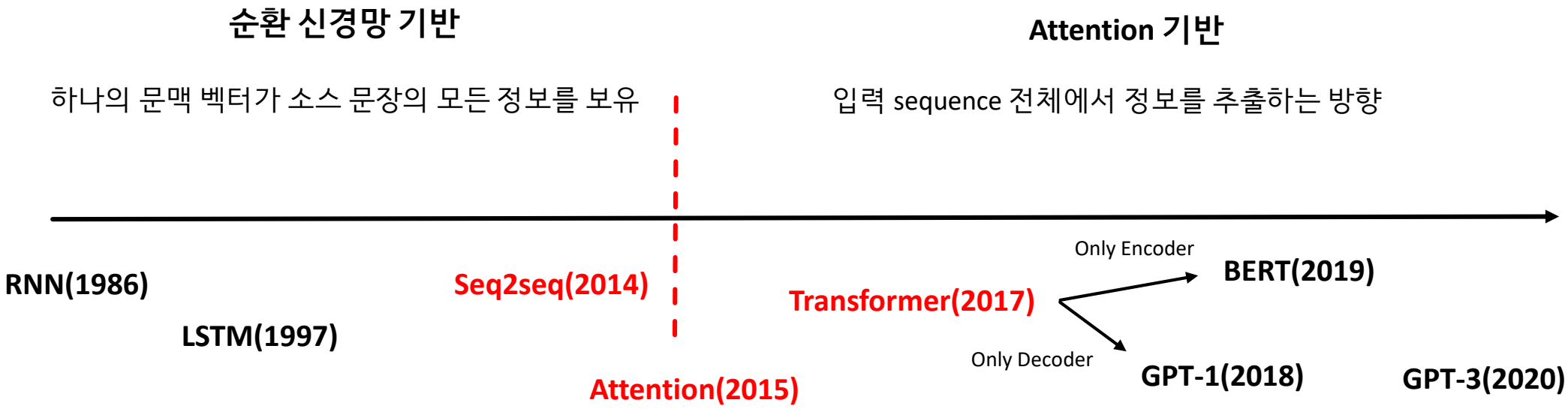
(Vaswani et al. 2017, NeurIPS Proceedings)

2022.01.24

문구영

고려대학교
KOREA UNIVERSITY

Contents

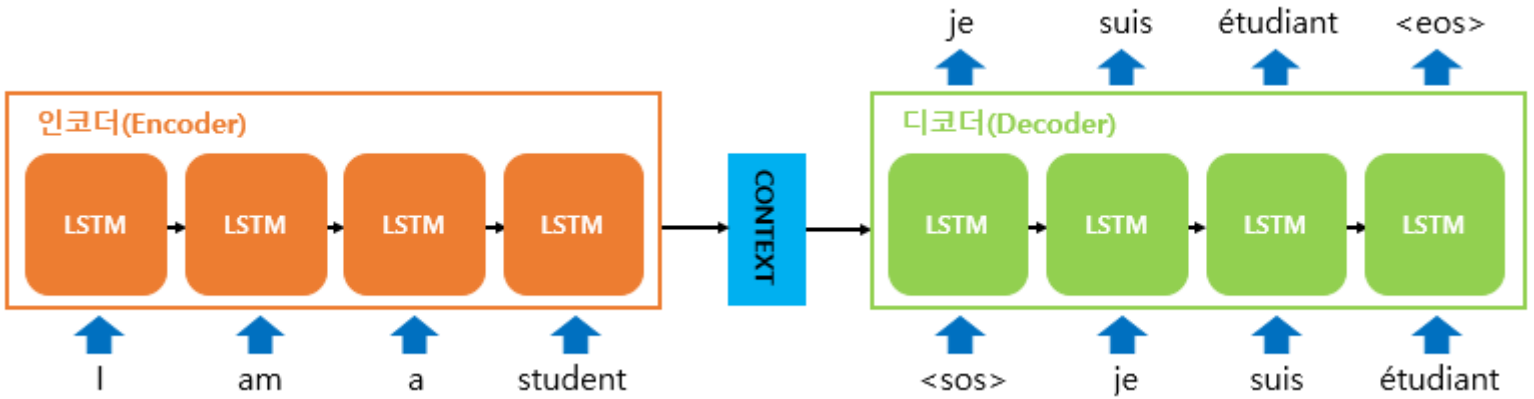


1. Seq2seq의 Encoder-Decoder 구조의 이해 및 한계
2. Seq2seq의 문제점을 개선한 Attention 이해
3. Encoder-Decoder 구조의 순환 신경망 계층을 전부 Attention 계층으로 대체한 Transformer 이해

1. Background : Language Model with Seq2seq

Seq2Seq

- 한 시계열 데이터를 다른 시계열 데이터로 **mapping**
- 입력 시퀀스로부터 다른 도메인의 시퀀스를 출력하는 분야에서 사용되는 모델 ex) 챗봇, 기계 번역, Q&A
- 순환 신경망으로 구성된 **Encoder-Decoder** 구조 활용



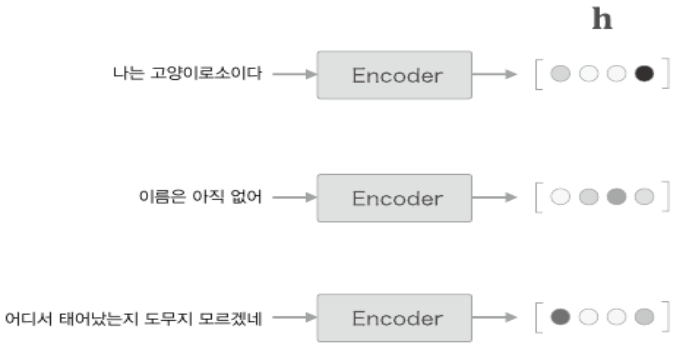
* <sos>, <eos> : 문장의 시작과 끝을 알리는 special token

1. Encoder가 입력 문장을 **encoding** (정보를 어떤 규칙에 따라 변환)
2. Encoder의 출력인 **Context**에는 번역에 필요한 정보가 응축되어 있음 (**고정된 길이**의 문맥 벡터)
3. Decoder는 해당 Context 정보를 바탕으로 번역 문장을 생성 (**decoding**)

1.1 Seq2Seq : Encoder & Decoder

Encoder

- 임의 길이의 sequence를 고정된 길이의 encoded 벡터로 변환



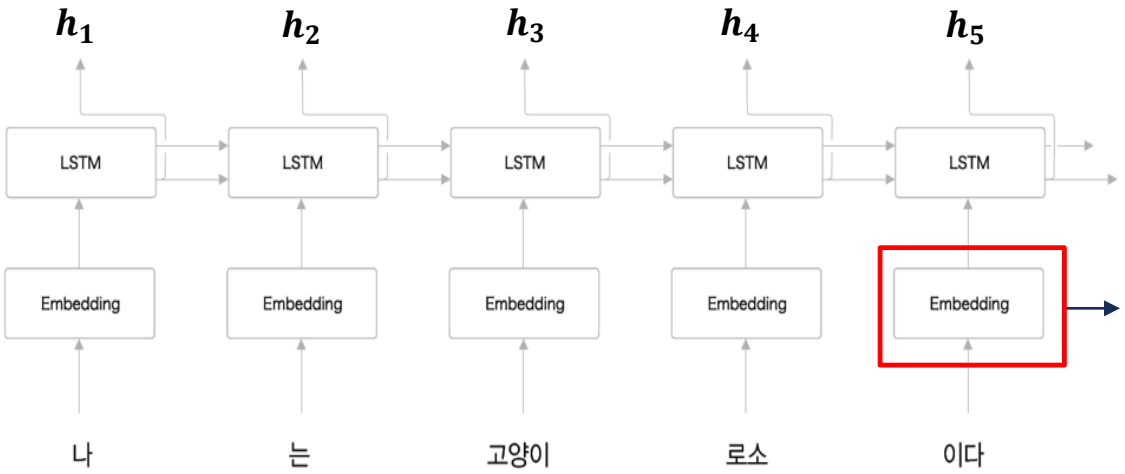
$\mathbf{x} = (x_1, \cdots, x_{T_x})$, → 단어(토큰) 벡터들의 **sequence**로 표현된 입력 문장

$h_t = f(x_t, h_{t-1})$ → 특정 time step t에서의 **hidden state** 출력

(t 시점 단어, t-1 시점의 hidden state를 입력으로 받음)

$c = q(\{h_1, \cdots, h_{T_x}\})$, → **문맥 벡터** (순환 신경망 계층의 최종 은닉 상태 출력)

* f, q : 비선형 활성 함수



Context

단어의 벡터 표현 ex) 왕 + 여자 = 여왕

- 딥러닝 네트워크가 입력으로써 활용할 수 있도록 유의미한 representation으로 변환

```
# Embedding
net1 = Embedder(TEXT.vocab.vectors)
x = batch.Text[0]
x1 = net1(x)  # 단어를 벡터로

print("입력 텐서 크기: ", x.shape)
print("Embedding된 출력 텐서 크기: ", x1.shape)
```

```
입력 텐서 크기:  torch.Size([24, 256])
Embedding된 출력 텐서 크기:  torch.Size([24, 256, 300])
```

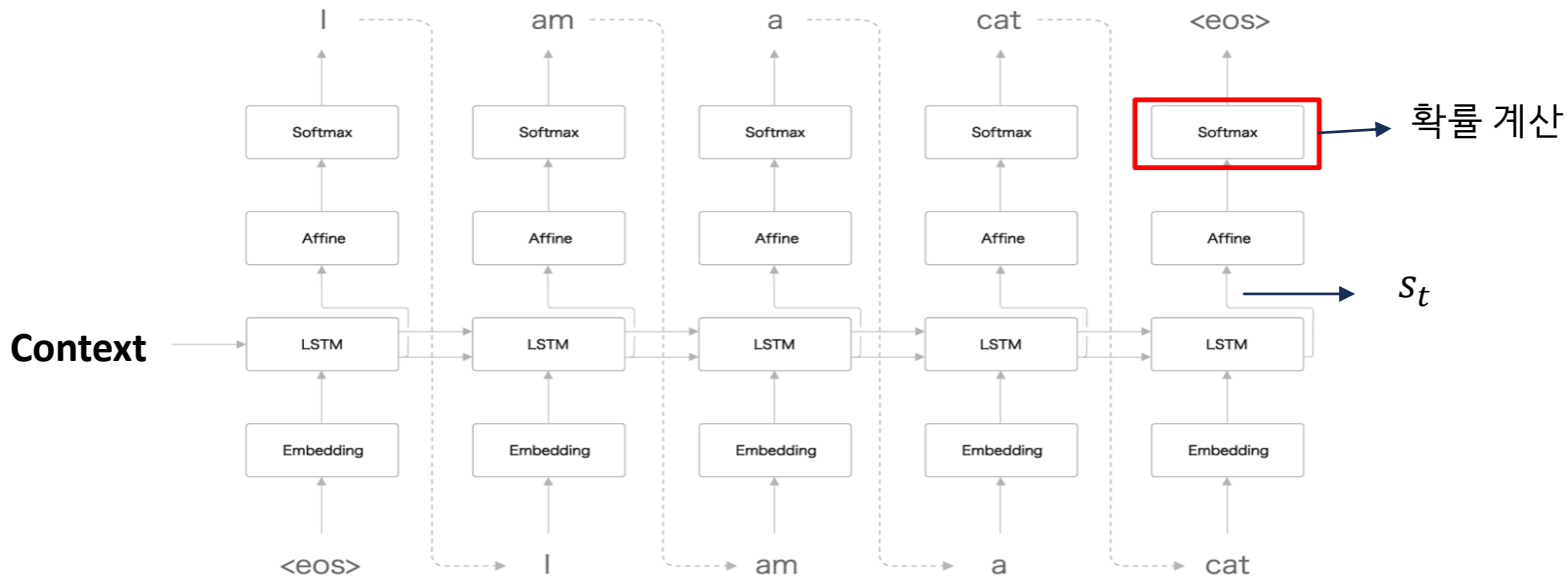
1.1 Seq2Seq : Encoder & Decoder

Decoder

- Encoding된 Context 벡터를 활용하여 출력 sequence 생성 $\mathbf{y} = (y_1, \cdots, y_{T_y})$.
- **특정 t 시점에 등장할 단어의 확률을 예측** (Context, t-1시점 까지 예측된 단어들, t 시점의 hidden state가 주어졌을 때)

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c),$$

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c), \quad P(y_1 \mid c) \cdot P(y_2 \mid y_1, c) \cdot P(y_3 \mid y_2, y_1, c) \cdot \cdots \cdot P(y_t \mid y_1, y_2, \cdots, y_{t-1}, c)$$



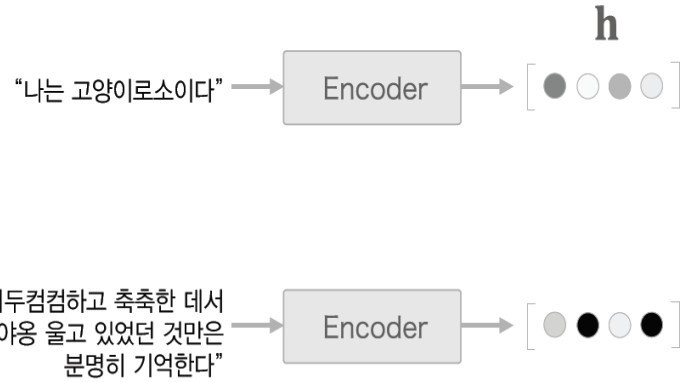
2. Motivation of Attention : Limitations of Seq2Seq

Encoder

1. 순환 신경망을 활용하여 **고정된 길이의 context vector**에 입력 sequence의 정보를 압축
2. 하나의 Context vector가 소스 문장의 모든 정보를 가지고 있어야 함
3. 입력 sequence의 길이에 관계 없이 정보를 고정된 길이의 벡터로 밀어넣음

→ 입력 sequence의 길이가 길어지면 **병목 현상(bottleneck) 발생, 기울기 소실 문제**

그림 8-1 입력 문장의 길이에 관계없이, Encoder는 정보를 고정 길이의 벡터로 밀어 넣는다.



Decoder

1. 인간은 문장을 번역할 때 입력과 출력의 여러 단어 중 어떤 단어끼리 서로 관련되어 있는가 라는 **global한 대응 관계**를 고려함
2. Seq2Seq에서는 Encoder의 **마지막 hidden state 출력만을 Context로써 Decoder에 전달**함
   - Encoder의 순환 신경망 계층의 모든 hidden state 출력들을 고려하지 않음

2.1 Motivation of Attention : Improvement for Seq2Seq

Solution - Encoder

- **하나의 고정된 길이 벡터 제약 무시**

  → Encoder의 시점(단어) 별로 순환 신경망 계층의 모든 hidden state 출력들을 이용 (return_sequences=True)

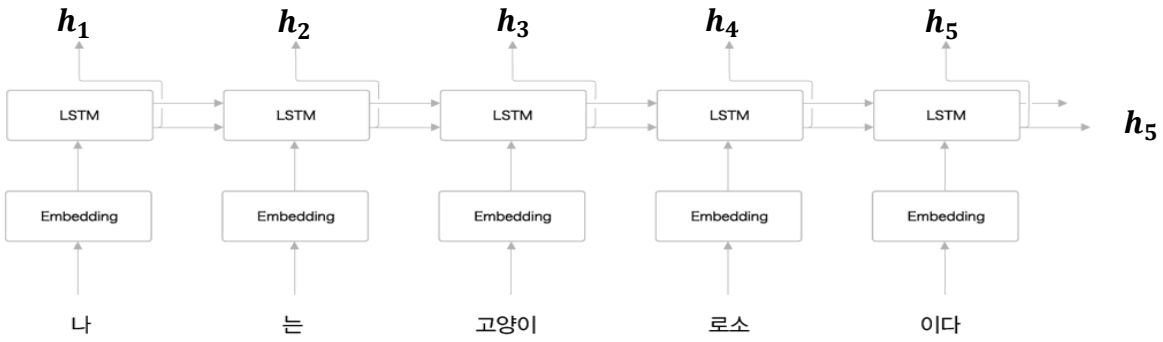
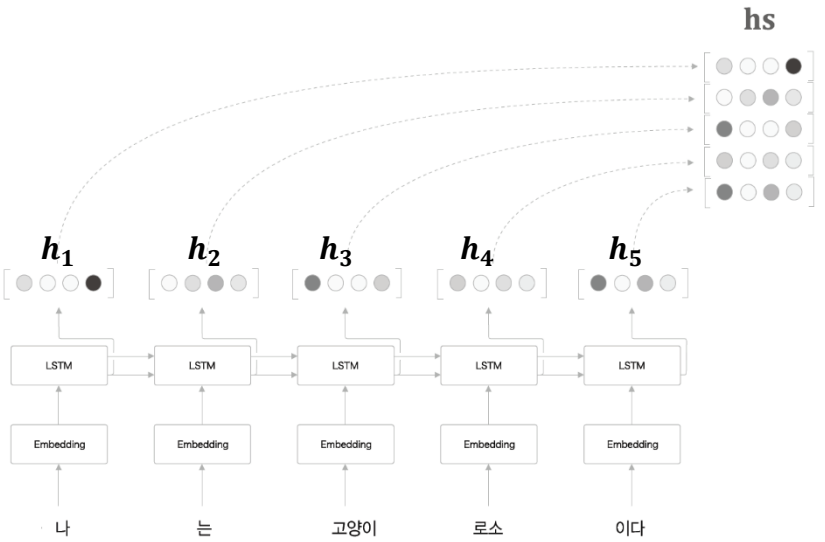


그림 8-2 Encoder의 시각별(단어별) LSTM 계층의 은닉 상태를 모두 이용(**hs**로 표기)



- hs는 단어 수 만큼의 벡터를 포함, 개별 행 벡터는 해당 단어에 대한 정보를 많이 보유

→ Decoder가 hs를 전달 받으면 입력 sequence의 모든 정보를 반영할 수 있게 됨 (global)

2.1 Motivation of Attention : Improvement for Seq2Seq

Solution - Decoder

- Encoder가 전달하는 hs를 전부 활용할 수 있도록
- 단, 가중치를 고려하여 입력 sequence의 모든 hidden state 중 **중요도가 높은 원소에 선택&집중**을 할 수 있도록

→ 현재 예측 시점에서 필요한 정보에 주목하여, 그 정보로부터 시계열 변환 수행

그림 7-8 Decoder를 구성하는 계층

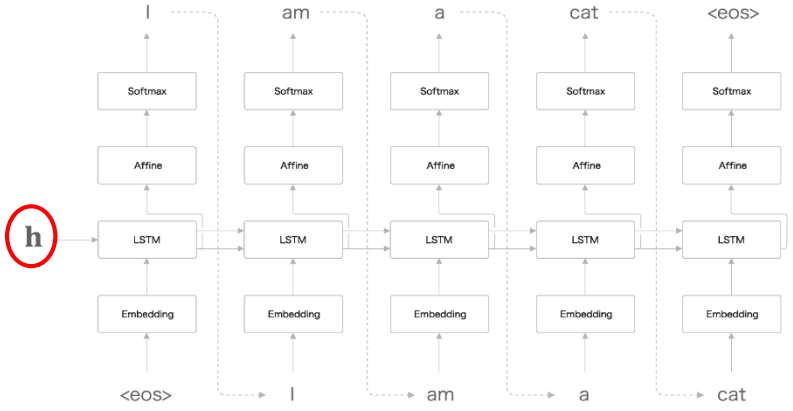
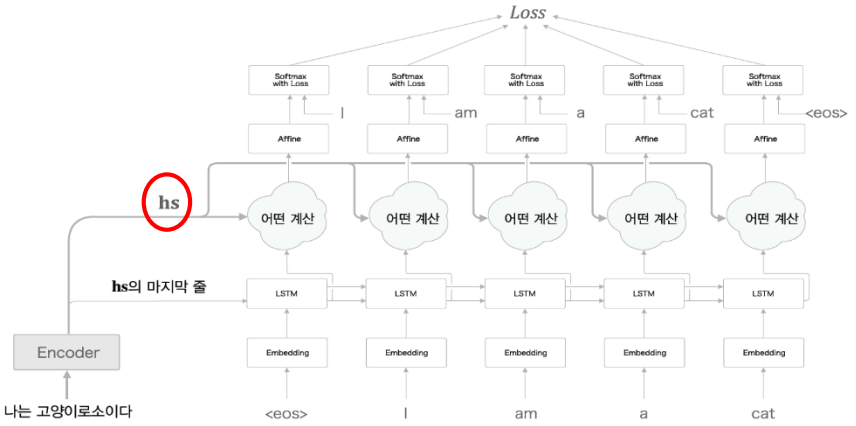


그림 8-6 개선 후의 Decoder의 계층 구성

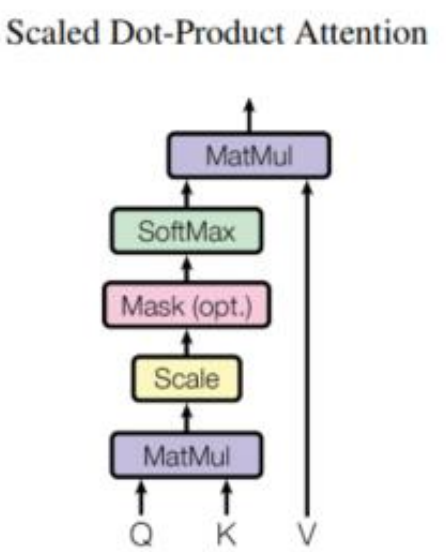


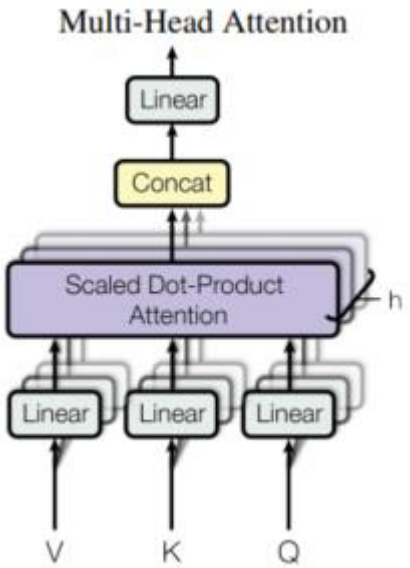
* h : Encoder의 마지막 순환 신경망 계층의 hidden state 출력
* hs : Encoder의 모든 순환 신경망 계층의 hidden state 출력들

'어떤 계산'을 통해 입력과 출력의 단어들 중 어떤 요소들끼리 서로 관련되어 있는가 라는 대응 관계를 seq2seq에 학습

3. Attention

- Seq2Seq : **Alignment** 학습 (대응 관계) ex) 나 = I, 고양이 = cat
- 입력과 출력의 여러 단어 중 **어떤 단어끼리 서로 관련되어 있는가** 라는 **Soft Alignment** 학습을 자동으로 도입
- Sequence를 구성하는 개별 단어 토큰들에 대한 **Query, Key, Value** 벡터들을 활용하여 Attention 연산 수행





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

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$MultiHead(Q,K,V) = Concat(head_1, \dots, head_h)W^O$$

- Q, K, V 벡터들의 행렬 연산으로 구성
- **Multi head** : 서로 다른 가중치 활용

3.1 Attention - Query Key Value

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

```
emb_size = 768  # 개별 단어 벡터의 차원
num_heads = 8

# embedding된 입력 텐서를 받아 linear projection 수행
keys = nn.Linear(emb_size, emb_size)
queries = nn.Linear(emb_size, emb_size)
values = nn.Linear(emb_size, emb_size)
```

- **Query** : 물어보는 주체, **현재 처리하고자 하는 단어 벡터 (**Decoder의 이전 layer의 hidden state)

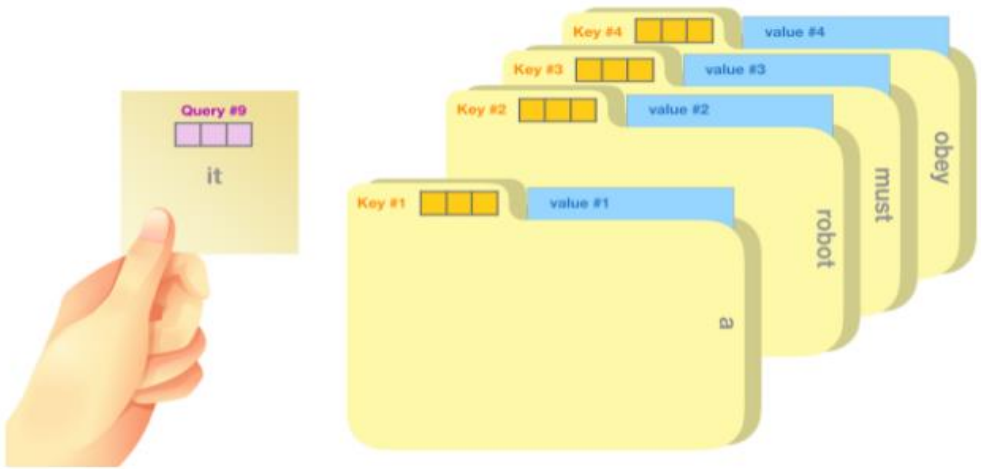
  → **Attention의 영향을 받는 Decoder의 토큰**

- **Key** : Attention 연산을 수행할 **sequence 내의 모든 단어들 (Encoder의 모든 hidden state 출력들)**

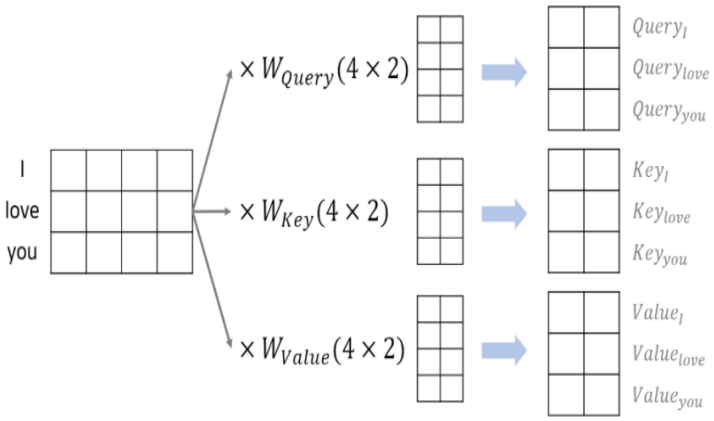
  → **영향을 주는 Encoder 토큰들**

- **Value** : Key와 연결된 실제 단어 벡터 (Encoder의 모든 hidden state 출력들)

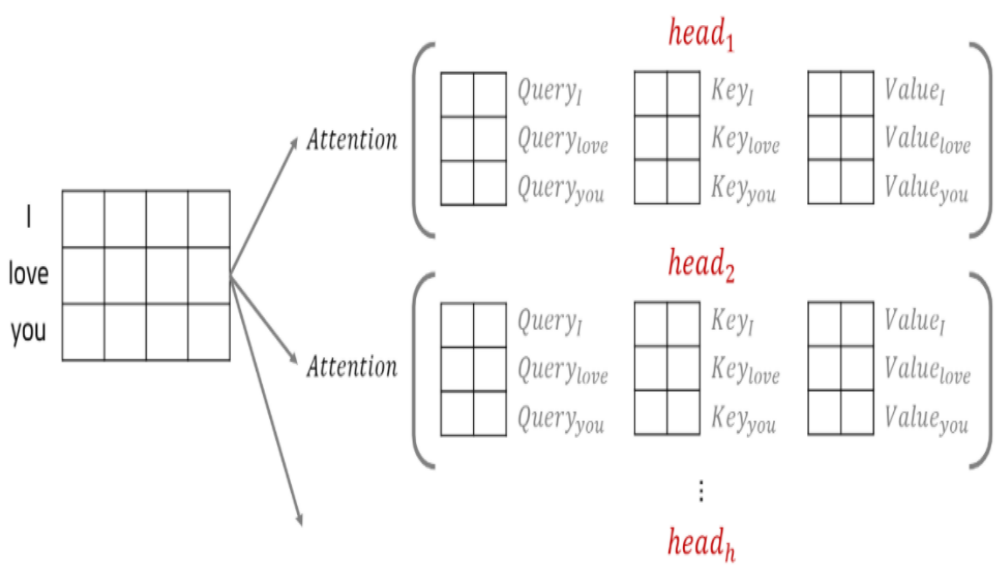
  → **그 영향에 대한 가중치(Attention map)이 곱해질 Encoder 토큰들**

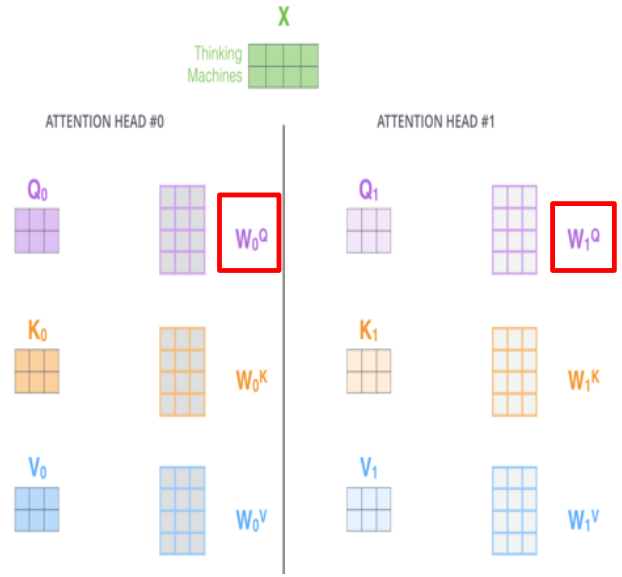


3.1 Attention – Query Key Value



Scaled-dot

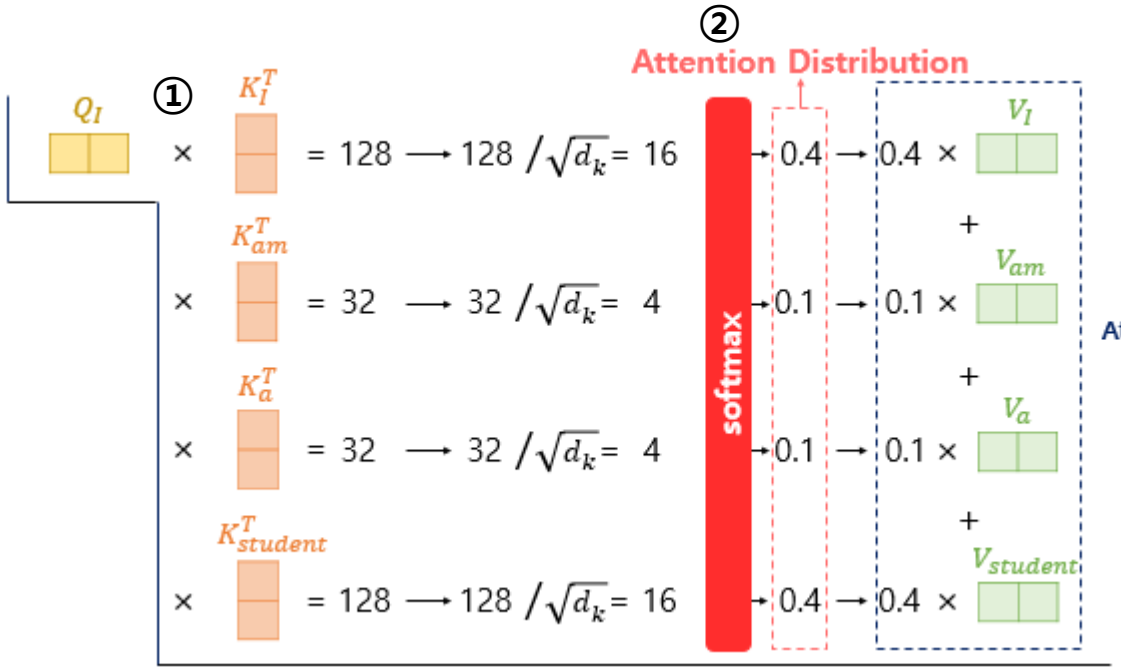




Multi-Head

- 각 head는 서로 다른 h개의 layer를 거침 (h개의 서로 다른 컨셉을 네트워크가 구분해서 학습)
  → **다양한 representation subspace** 제공 (모델의 능력 확장)

3.2 Attention - Operation



특정 단어에 대한 Context Vector

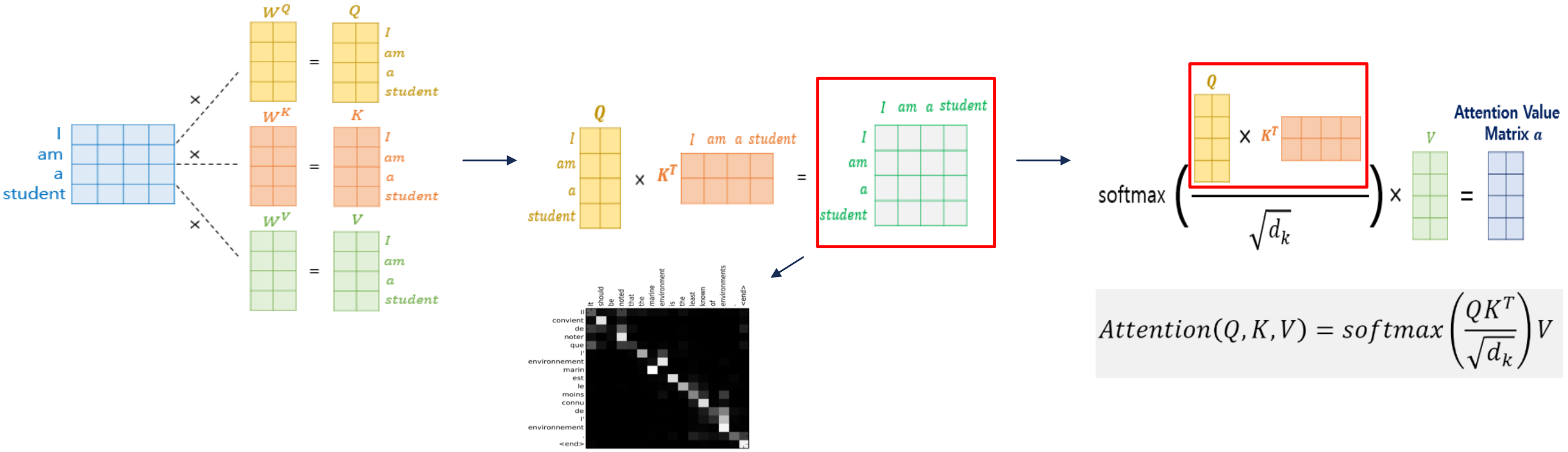
(입력 Sequence의 중요한 단어들에 초점을 맞춘)

① 유사도, 에너지 계산 – 어떤 key에 대해 높은 가중치를 갖는지 ('주목' 해야 하는 단어)

ex) I am a teacher : I (Query)가 {I, am, a, teacher} 각각의 단어들 (keys)에 대해 얼마나 **연관성**이 있는 지 계산

② Softmax를 거쳐 확률값으로 표현된 후 Value에 **Attention weight**로써 곱해짐

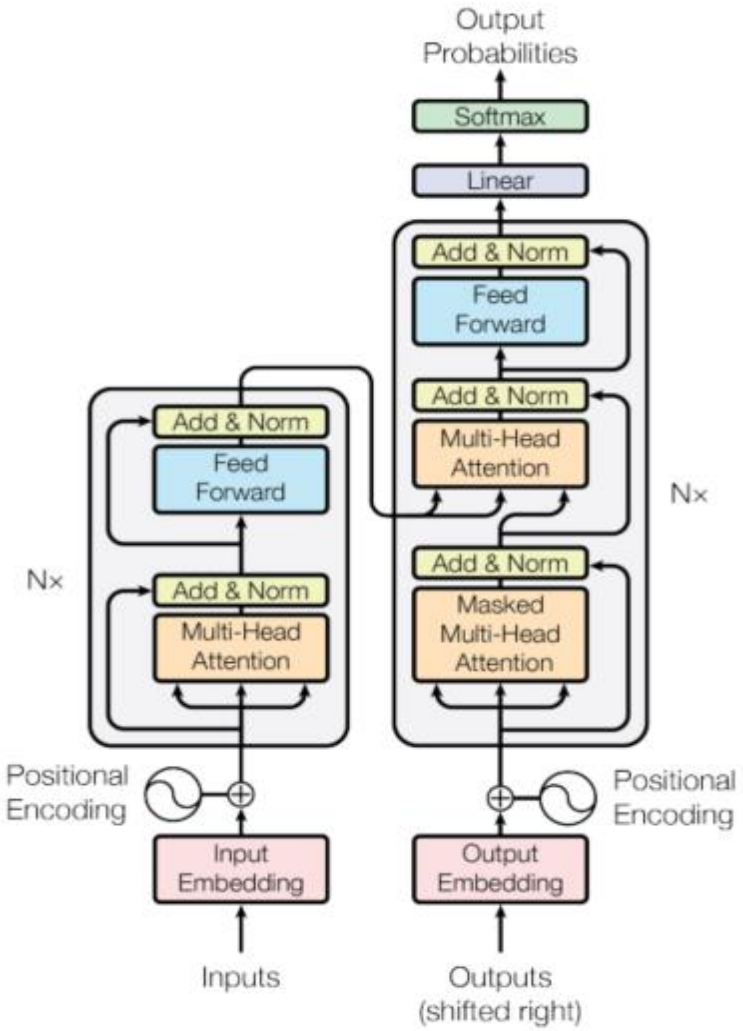
3.3 Attention – Overall Process



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

1. Query, Key 벡터들의 내적 ($\sqrt{d_k}$로 나누면 Attention score 행렬)
2. Softmax를 거쳐 **Attention distribution** 계산
3. 해당 분포를 Value에 곱해 모든 단어에 대한 Attention value 행렬 계산
4. 1~3 과정을 h개의 head마다 반복

4. Transformer - Architecture



Encoder

- N=6, 각 block은 2개의 sub-layer로 구성

  (**Multi-Head Self Attention & Position wise F.C feed-forward)**
- Residual connection 사용 - $x + sublayer(x)$

Decoder

- N=6, Encoder의 sub-layers 사이에 다른 layer 추가

  - Encoder stack의 hidden state 출력들에 대해 **Multi-Head Attention** 수행
- Residual connection 사용

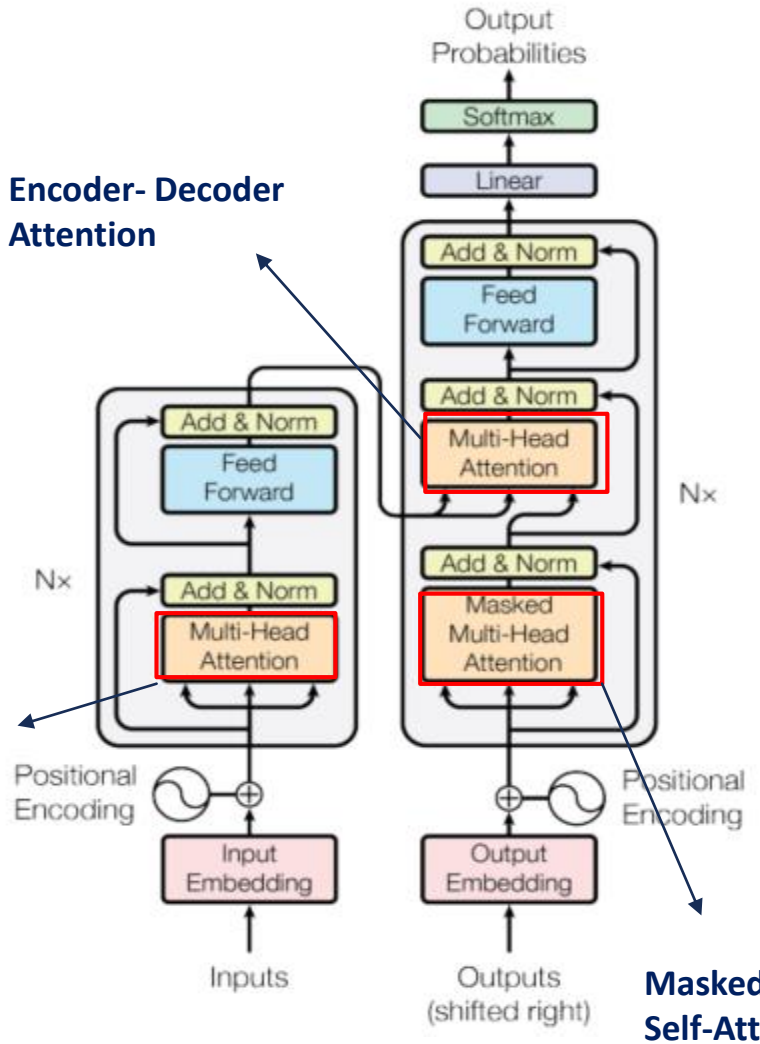
Position-wise Feed-Forward Networks

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$ → 서로 다른 layer마다 다른 가중치

Positional Encoding

- 순환 신경망 구조를 사용하지 않는 대신 **Sequence내 단어들의 위치 정보를 반영**
- 입력에 대한 embedding 벡터에 add

4.1 Transformer – Attention Operations



Encoder- Decoder Attention

Encoder Self Attention

Masked Decoder Self-Attention

Encoder Self-Attention

- Attention 연산을 자기 자신에 대해 수행
- **입력 sequence의 global representation 학습**

Masked Decoder Self-Attention

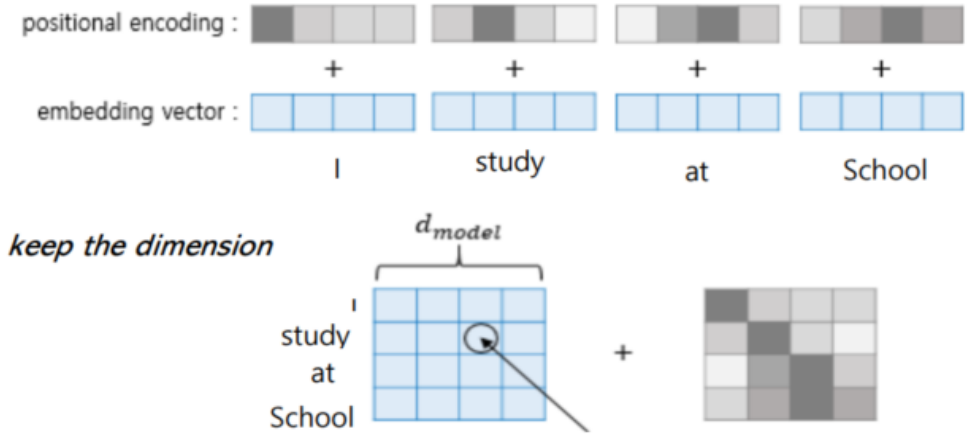
- 문장의 전체적인 표현 학습
- 단, **Mask를 사용해 입력 이후 시점의 Encoder 단어들은 참고하지 못하도록**

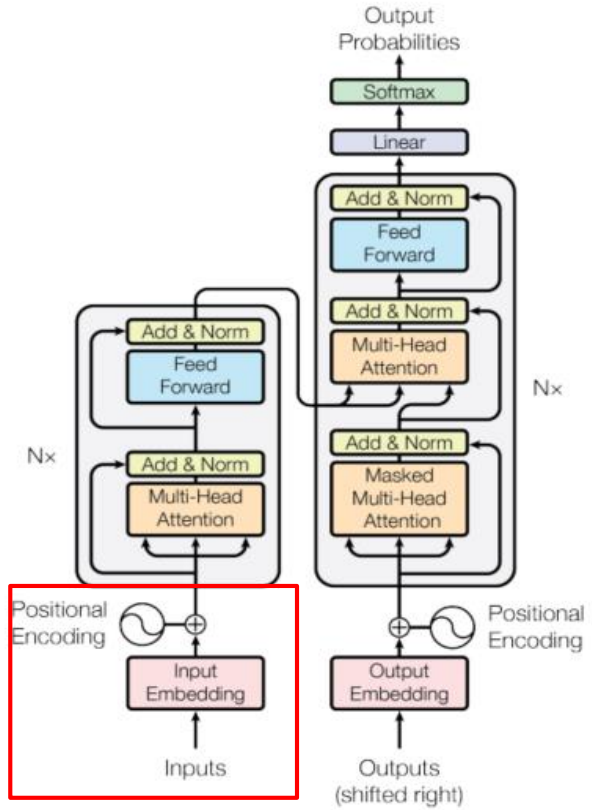
Encoder-Decoder Attention

- **출력 단어들이 입력 단어들 중 어떠한 정보(단어)에 초점을 맞추는 지**
- Decoder의 Query가 Encoder의 Key, Value 참고

① Positional Encoding

- Transformer는 sequence의 단어들을 순서대로 입력 받지 않음
- Sequence를 구성하는 각 단어들에 대한 embedding vector에 **positional information**을 더함






```
positions = nn.Parameter(torch.randn((img_size // patch_size) **2 + 1, emb_size))
```

→ Random하게 초기화된 상태에서 Transformer의 반복적인 학습을 통해 갱신, 학습됨

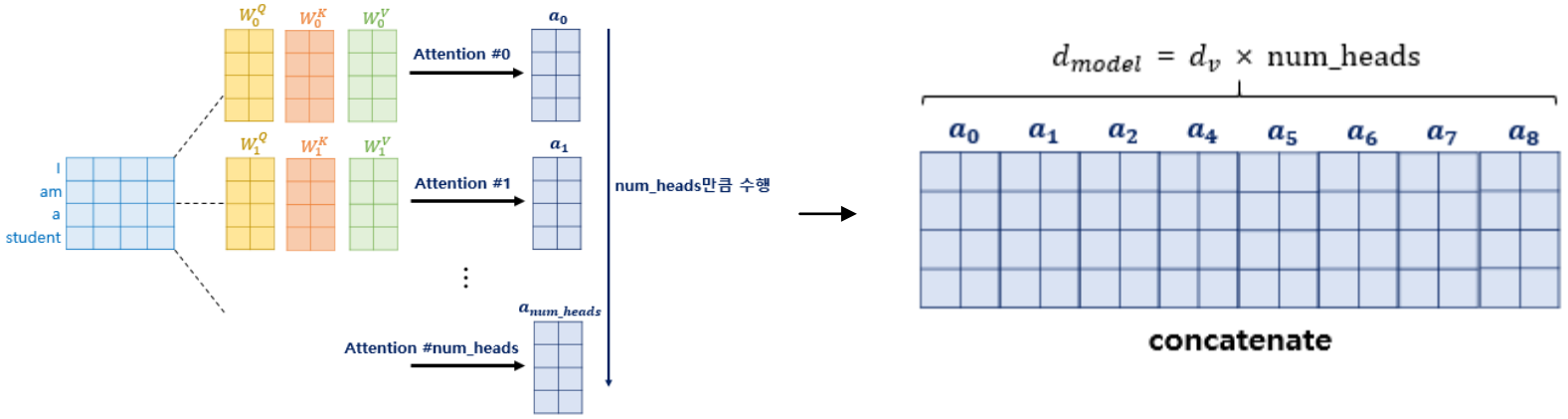
② Multi-Head Self Attention

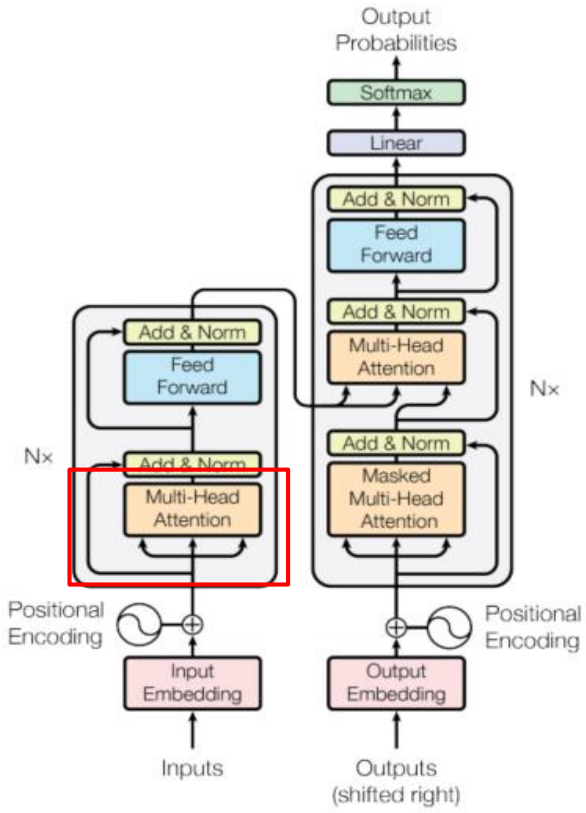
- 입력 sequence 자기 자신에 대한 Attention을 통해 **global representation** 학습
- 긴 입력 sequence를 효율적으로 처리하지 못하는 seq2seq의 한계 극복

A boy who is looking at the tree is surprised because it was too tall.

A boy who is looking at the tree is surprised because it was too tall.

→ 입력 sequence 내에서 모든 단어들간의 **연관성** 계산





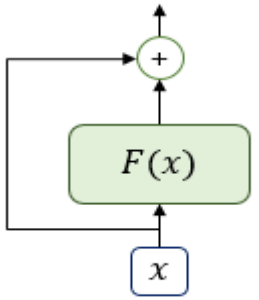
- **서로 다른 시각으로 정보들을 수집**

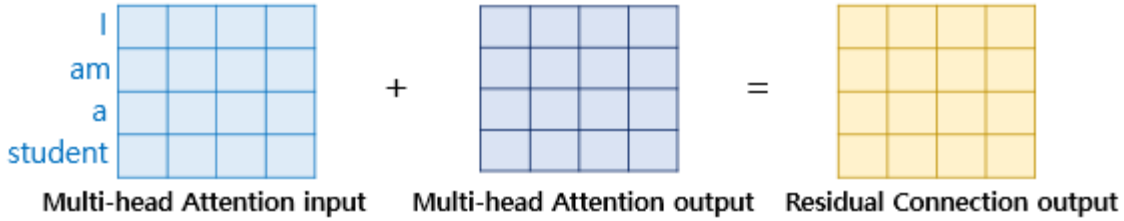
③ Residual Connection & Layer Normalization

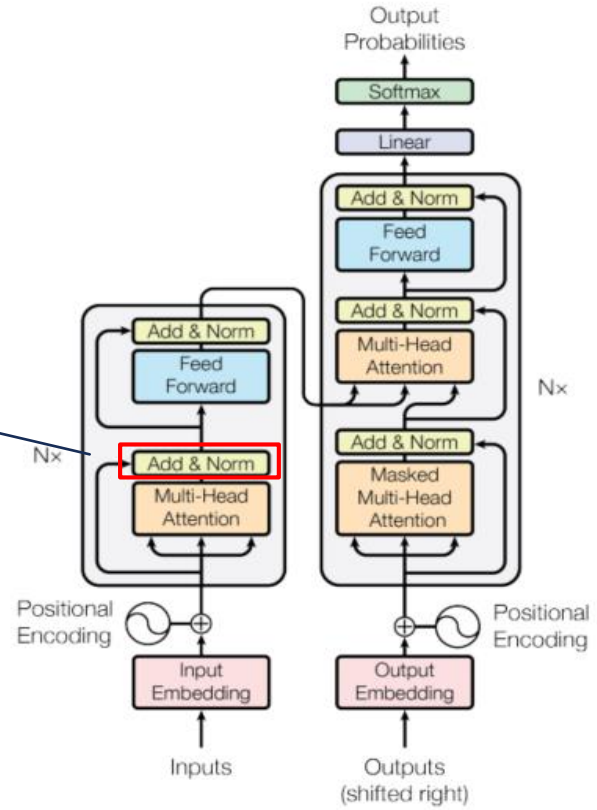
$$H(x) = x + F(x)$$

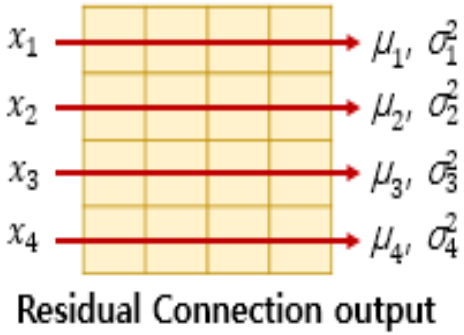
- Add function

$$H(x) = x + Multi - head\ Attention(x)$$









Normalization

$$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

Layer Normalization

$$ln_i = \gamma \hat{x}_i + \beta = LayerNorm(x_i)$$

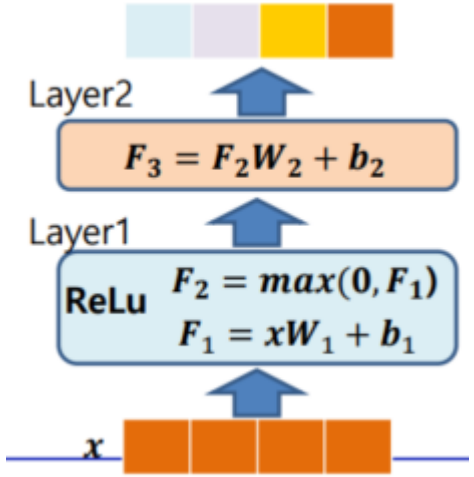
*** r, B는 학습되는 scaling parameter**

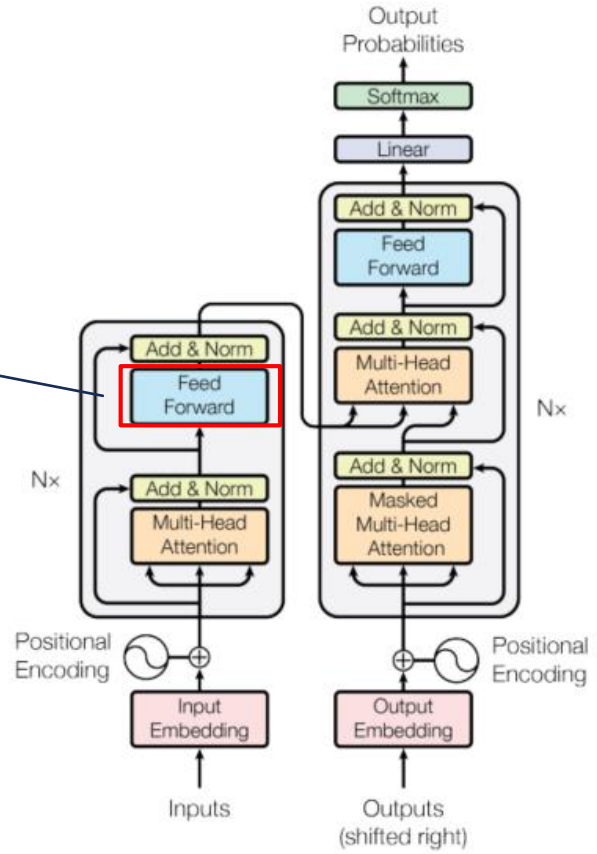
→ Residual connection을 거친 결과에 정규화 적용

→ layer를 거치면서 발생하는 scale 변화를 방지 (안정적인 학습)

④ Position-Wise FFNN

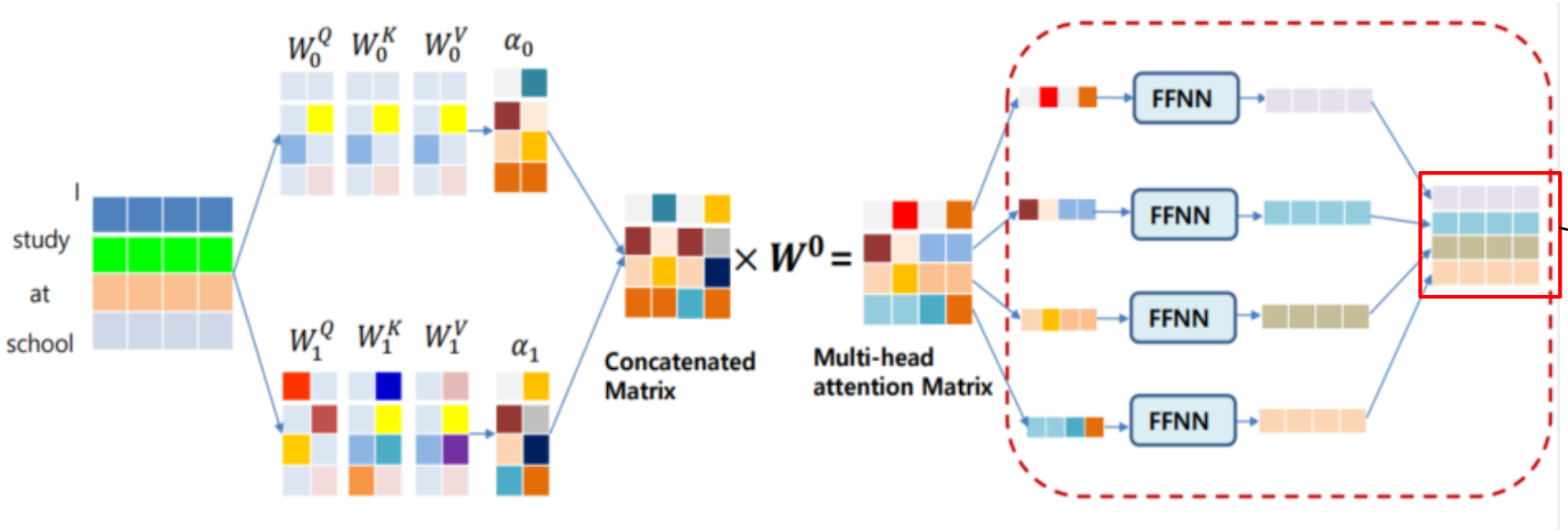
$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

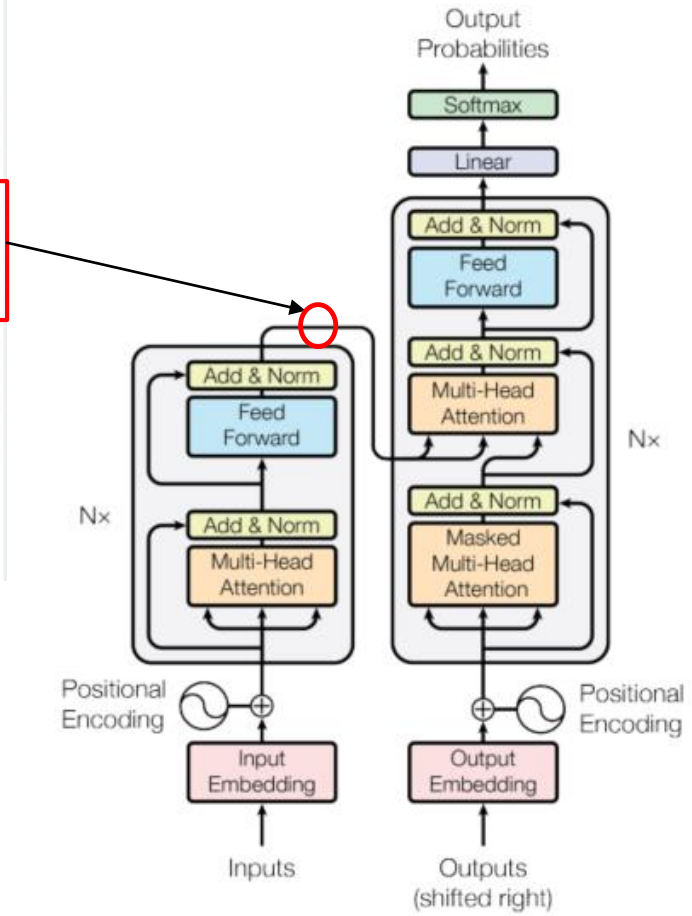




- 각 layer는 동일한 parameter 공유
- 서로 다른 layer는 서로 다른 parameter 보유

4.2 Transformer – Encoder (Summary)





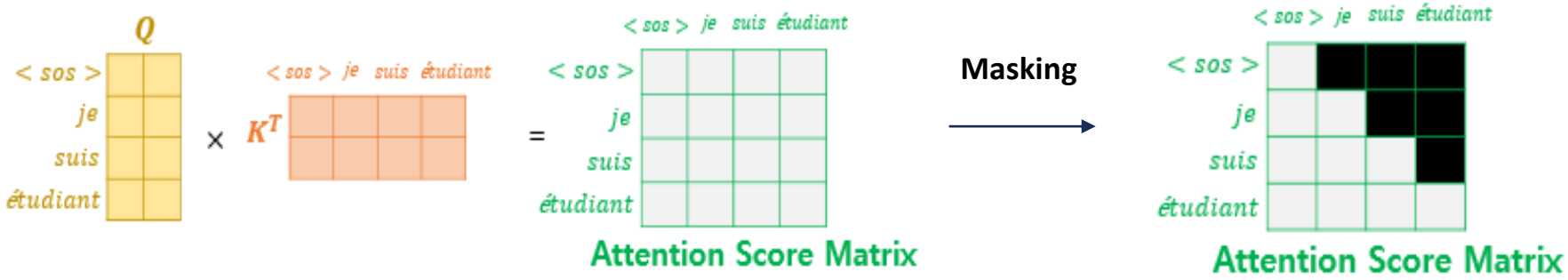
- Encoder의 출력을 Decoder에 전달
  - 입력 sequence를 구성하는 단어들에 대한 중요도를 고려한 embedding, representation
  - Encoder만을 활용하여 다양한 NLP 태스크를 수행하기도 함 (BERT)

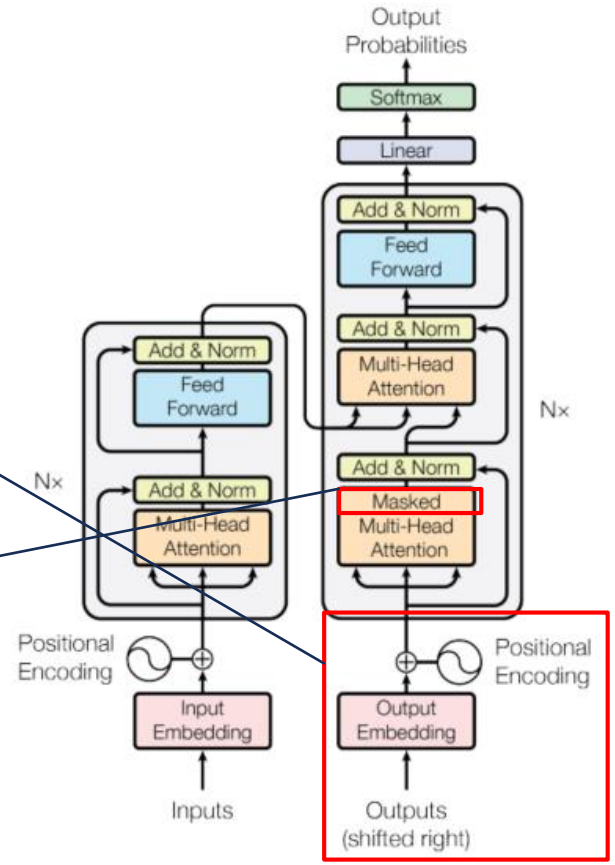
① Embedding & Positional Encoding

- Decoder는 정답 **sequence (번역된 문장)를 한꺼번에 입력으로 받음**
- 입력 받은 sequence 행렬로부터 각 시점의 출력을 정확히 예측하도록 학습됨 - **Teacher Forcing**

② Look-ahead mask

- **Decoder의 미리 보기 방지**
  - 현재 시점에 대한 예측에서 현재 시점보다 미래에 있는 단어들을 참고하지 못하도록
- Sequence를 구성하는 입력 단어를 매 시점마다 순차적으로 입력 받는 순환 신경망과 달리 Transformer는 입력을 한번에 받기 때문에 필요





③ Masked Decoder Self-Attention

- Decoder 입력 sequence에 대한 Attention 연산 수행 – **Masking 정보를 고려**하여 선택적으로 수행

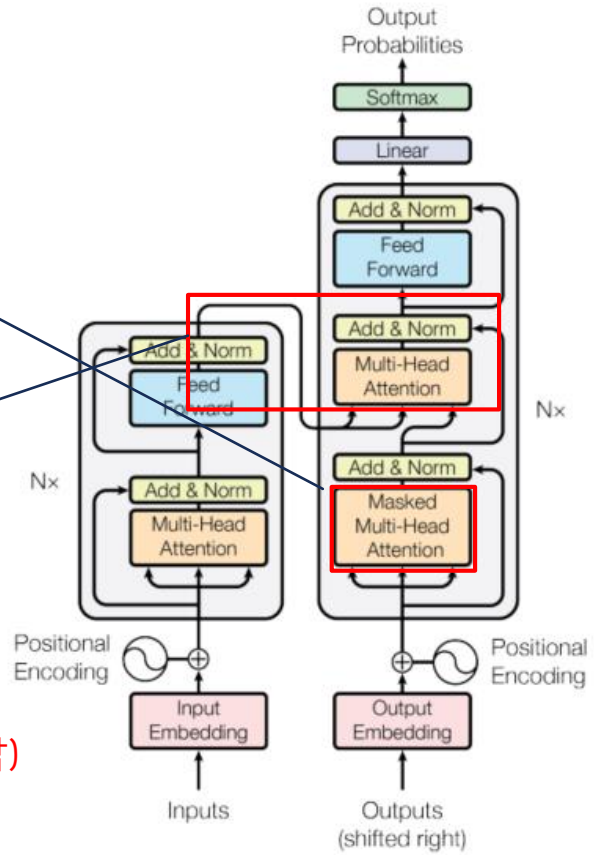
④ Encoder – Decoder Attention

- **Encoder가 전달한 정보에서 보다 중요한 원소에 주목**

  ① Encoder의 모든 hidden state 출력과 Decoder가 예측하고자 하는 시점의 입력 hidden state 와의 **연관성** 계산 (Attention score, 가중치)

  ② 위에서 구한 가중치를 이용하여 **가중합** 계산 (보다 중요한 원소에 **주목**하는 연산)

현재 시점의 타깃 예측에 있어 입력 sequence 중 어떤 부분들이 중요한 지 판단 (attention score) & 활용 (가중합)



4.3 Transformer – Decoder (Summary)

그림 8-18 Attention 계층을 갖춘 Decoder의 계층 구성

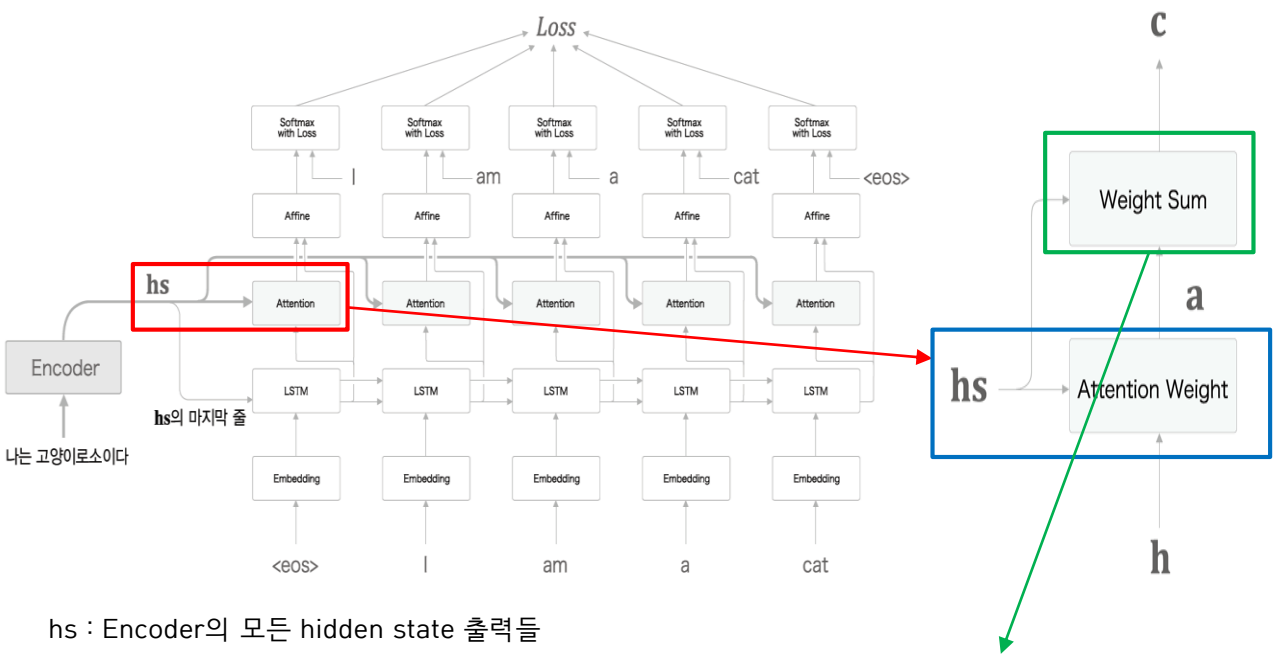


그림 8-13 내적을 통해 hs의 각 행과 h의 유사도를 산출(내적은 dot 노드로 그림)

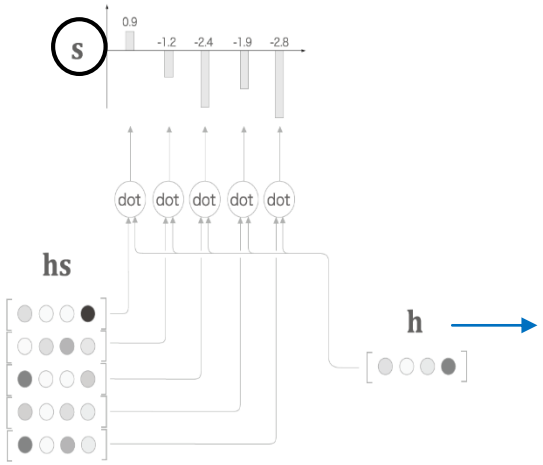
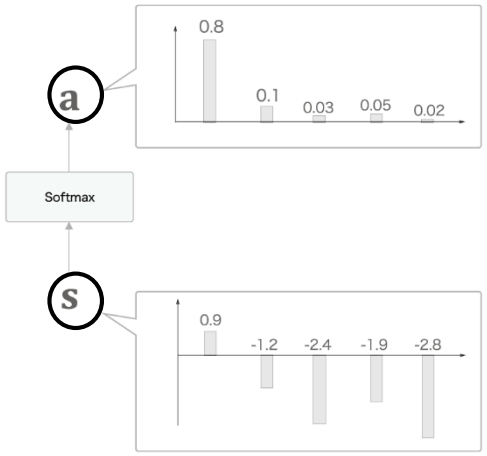


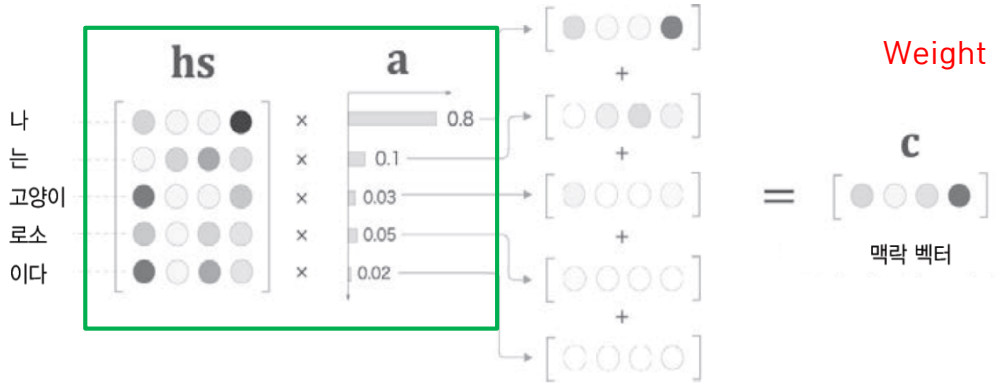
그림 8-14 Softmax를 통한 정규화



a : 각 단어들의 중요도 (Attention weight)

- hs : Encoder의 모든 hidden state 출력들
- h : Decoder의 특정 시점에서의 hidden state

그림 8-8 가중합을 계산하여 '맥락 벡터'를 구한다.



Weight sum : 각 단어들의 중요도를 고려하여 선택&집중

미분 가능한 연산

Summary

Overview

- **Method** : Encoder-Decoder 구조에서 일반적으로 사용되는 순환 신경망 계층을 전부 Attention 계층으로 대체
- **Objective** : **Attention을 도입하여 Seq2seq의 문제점을 효율적으로 해결** (Multi-head 병렬화를 통한 효율적 학습)

Key Insight

- Decoder가 Encoder의 모든 원소들 중 보다 중요한 부분에 **주목**할 수 있도록 함
- Attention weight를 활용한 가중합 연산으로 모델의 **선택과 집중** 능력을 미분 가능하도록 모델링

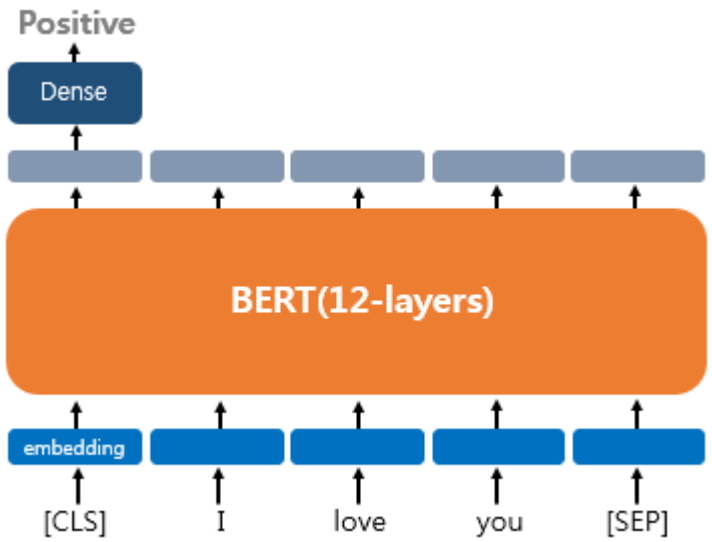
Main Contribution

- NLP 분야의 획기적 발전
- 이후 다양한 언어 모델들의 기반 제공 - **BERT, GPT**
- Computer Vision 분야에서도 Transformer 구조를 도입하고자 하는 다양한 연구 시도 (Vision Transformer)

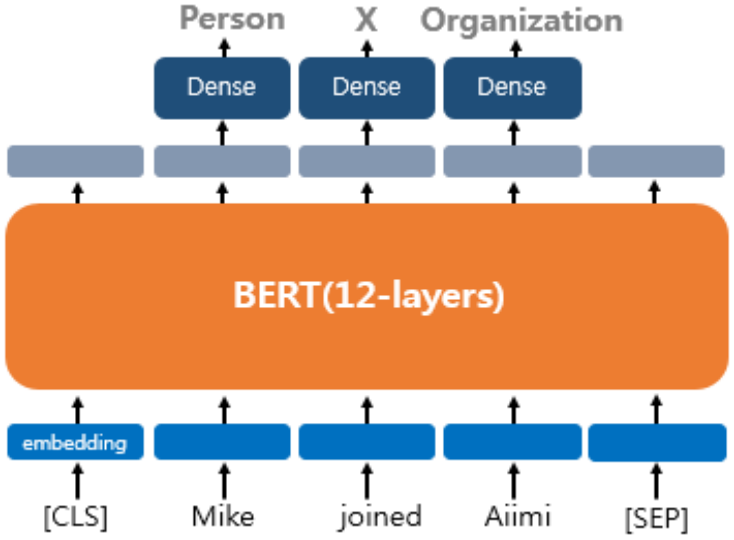
BERT (Bidirectional Encoder Representations from Transformers)

- **Transformer의 Encoder를 활용**하여 기존의 Word Embedding보다 단어의 의미를 잘 반영하는 벡터 representation 제공 (pre-trained)
- 해당 representation에 추가적인 sub layer를 쌓음으로써 다양한 NLP 태스크를 공통 architecture에 기반하여 해결

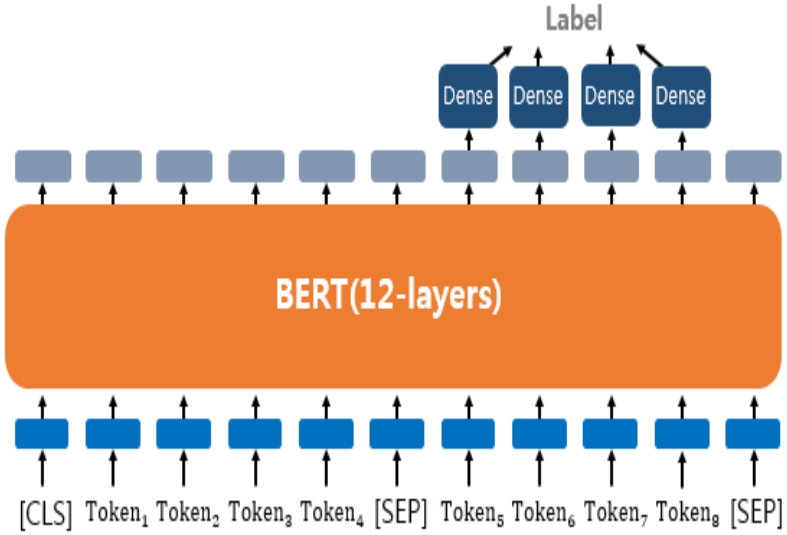
Sentiment Analysis



Named Entity Recognition



Question & Answering



Appendix

참고 문헌

- https://wikidocs.net/31379
- https://github.com/gymoon10/Paper-Review/tree/main/NLP
- https://github.com/gymoon10/Plant-Growth-Prediction/blob/main/Related%20works/Vision_Transformer.ipynb