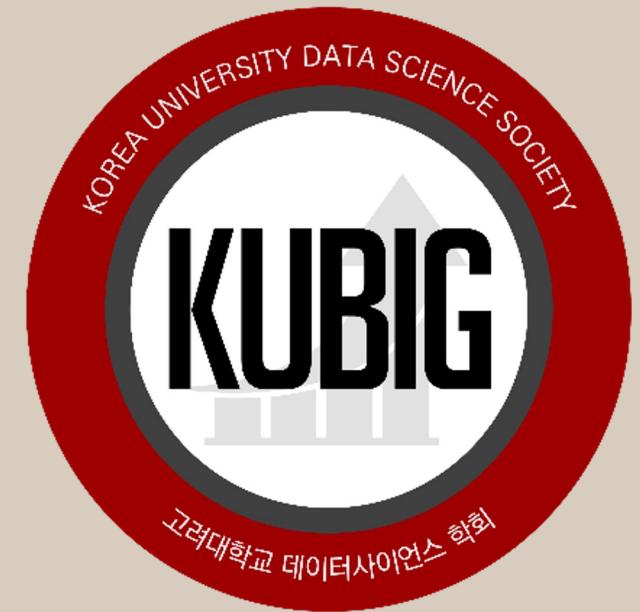


KUBIG 24-W
겨울방학 BASIC STUDY SESSION

NLP SESSION WEEK4



Today's Session Leader: 17기 김희준

01 Announcement, 복습과제 우수 코드 review

02 seq2seq

03 Attention

04 Transformer

05 예습과제 우수 코드 review, Announcement

01 Announcement, 우수 복습과제 Review



카톡방이 아닌 슬랙 채널 적극 활용!

보다 원활한 소통을 위해 슬랙 단체 DM 방을 개설하셔도 좋습니다(교수님이나 세션
리더 초대할 필요 X)

KUBIG CONTEST 2월 29일 예정.

팀별 중간발표 2월 15일 session 종료 직후 팀당 3~5분 분량으로 간단히 발표

서연님

WEEK3
복습과제2

네이버 쇼핑 리뷰 감성분석
with RNN, LSTM, GRU

화면공유 하셔서 5분 내외로 가볍게 리뷰해주시면 됩니다!

02 seq2seq

특정 도메인의 문장을 다른 도메인의 문장으로

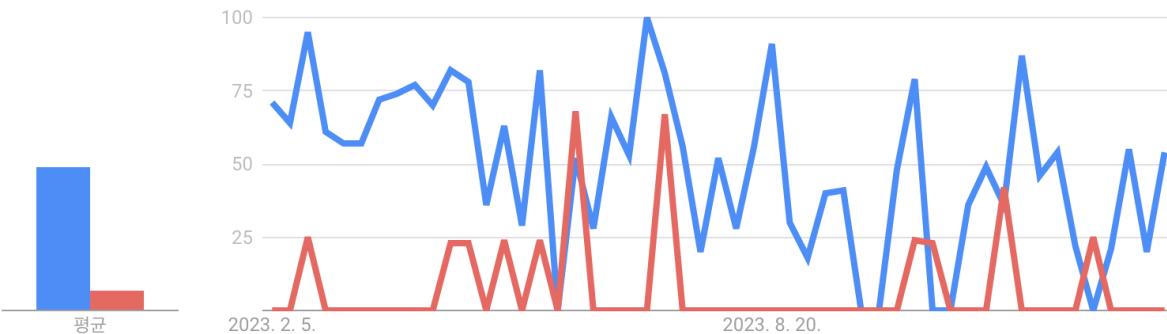
2-0. Why is it important?

왜, Attention과 Transformer가 중요한가?

Attention Is All You Need

시간 흐름에 따른 관심도 변화 ②

☰ <> ⌂



친구가 뉴진스 어텐션이라고 보내줌
아놔

2/5



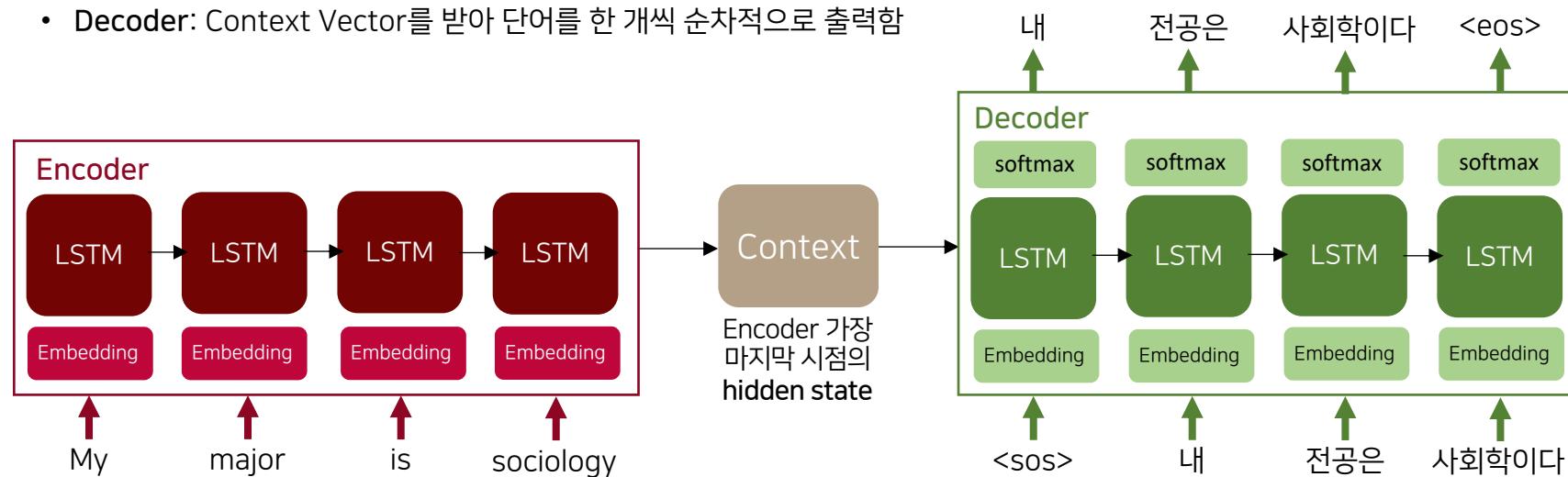
seq2seq

Sequence to Sequence: 일련의 시퀀스(문장)을 다른 도메인의 시퀀스로 변환해주는 알고리즘. Ex) 기계 번역, 챗봇



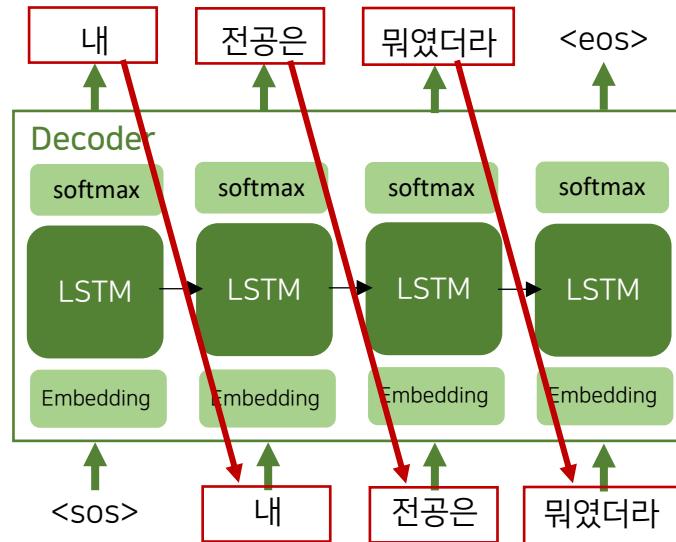
[seq2seq의 구조]

- Encoder: 입력 문장의 모든 단어를 순차적으로 입력 받음
- Context Vector: 인코더에서 입력 받은 단어 정보를 압축한 하나의 벡터
- Decoder: Context Vector를 받아 단어를 한 개씩 순차적으로 출력함



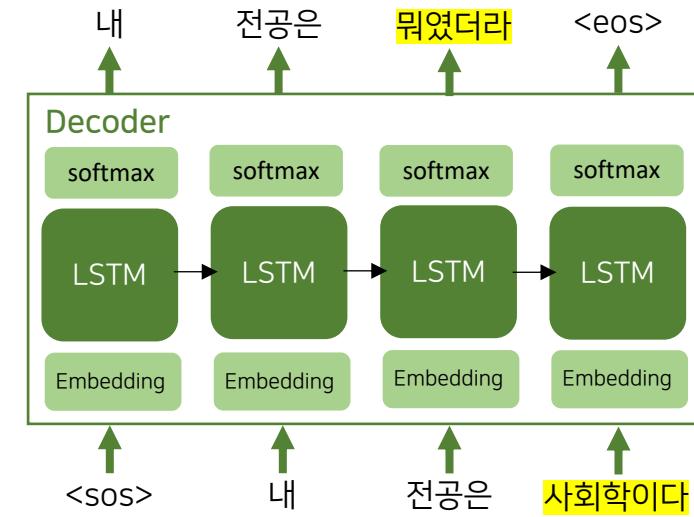
2-1. seq2seq

test



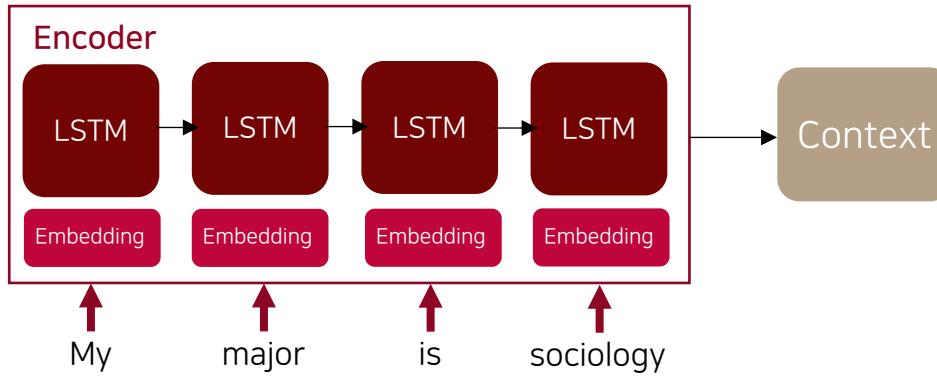
t-1 시점의 예측값은 t 시점의 input으로 쓰임

train



Decoder cell의 input은 모두 실제값으로 고정
= 교사 강요(Teacher forcing)

2-2. seq2seq Code



- `input_dim` = input 데이터의 vocab size = one-hot vector의 사이즈
- `emb_dim` = embedding layer의 차원
- `hid_dim` = hidden state의 차원 (= cell state의 차원)
- `n_layers` = LSTM layer 개수
- `dropout` = 사용할 dropout의 양 (overfitting 방지)
- `n_directions` = 1 (cf. bidirectional RNN의 경우 : `n_directions`=2)

```
class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, hid_dim, n_layers, dropout):
        super().__init__()

        self.hid_dim = hid_dim
        self.n_layers = n_layers

        # embedding: 입력값을 emd_dim 벡터로 변경
        self.embedding = nn.Embedding(input_dim, emb_dim)

        # embedding을 입력받아 hid_dim 크기의 hidden state, cell 출력
        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers, dropout=dropout)

        self.dropout = nn.Dropout(dropout)

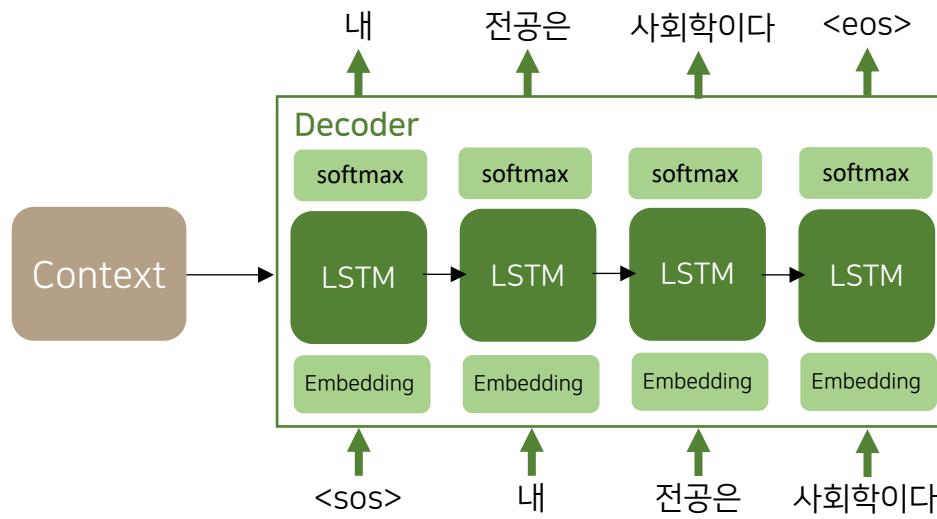
    def forward(self, src):
        # src: [src_len, batch_size]
        embedded = self.dropout(self.embedding(src))

        # 초기 hidden state는 영행렬
        outputs, (hidden, cell) = self.rnn(embedded)

        # output: [src_len, batch_size, hid dim * n directions]
        # hidden: [n layers * n directions, batch_size, hid dim]
        # cell: [n layers * n directions, batch_size, hid dim]

        return hidden, cell
```

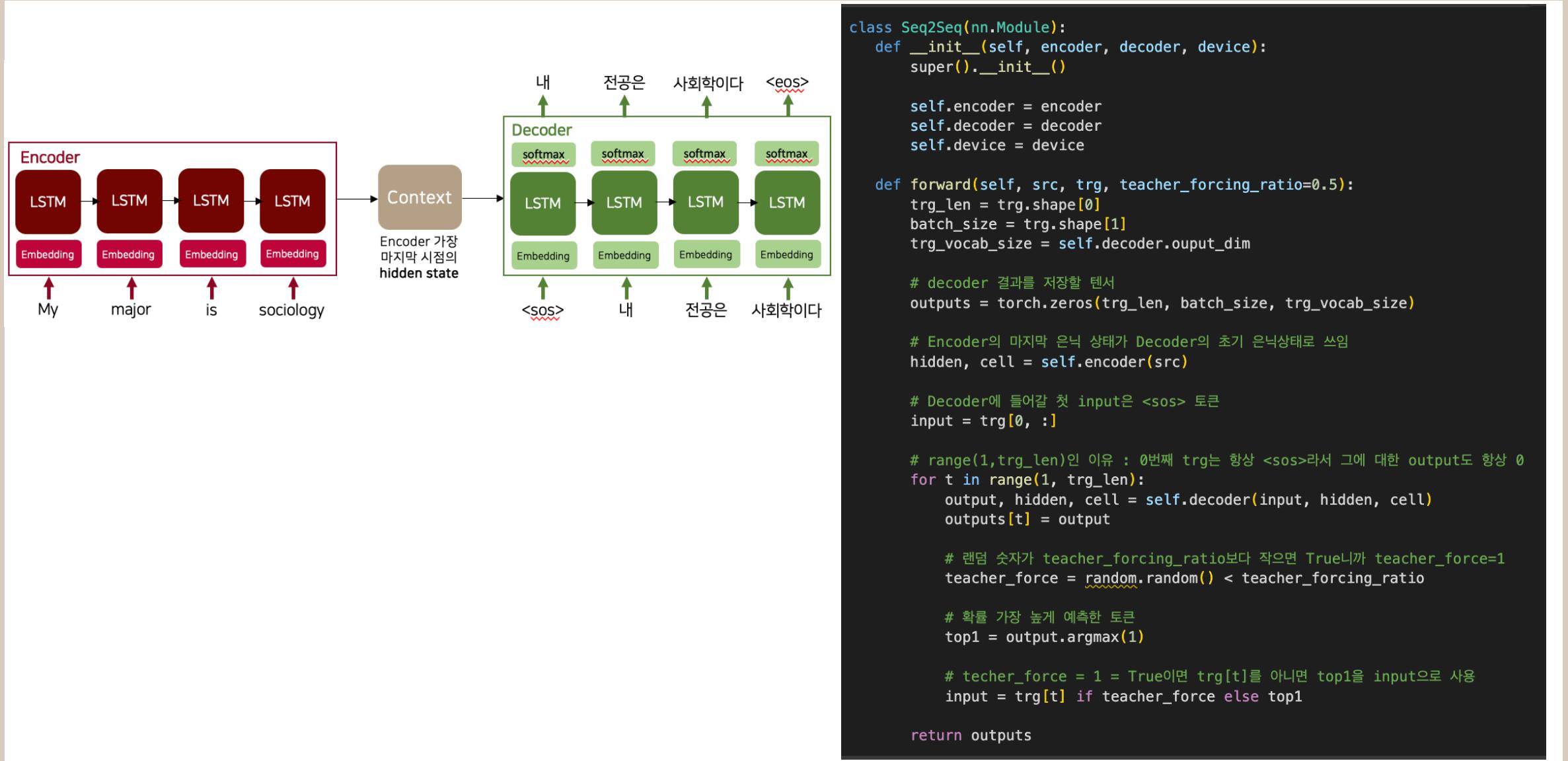
2-2. seq2seq Code



- `input_dim` = input 데이터의 vocab size = one-hot vector의 사이즈
- `emb_dim` = embedding layer의 차원
- `hid_dim` = hidden state의 차원 (= cell state의 차원)
- `n_layers` = LSTM layer 개수
- `dropout` = 사용할 dropout의 양 (overfitting 방지)
- `n_directions` = 1 (cf. bidirectional RNN의 경우 : `n_directions`=2)

```
class Decoder(nn.Module):  
    def __init__(self, output_dim, emb_dim, hid_dim, n_layers, dropout):  
        super().__init__()  
  
        self.output_dim = output_dim  
        self.hid_dim = hid_dim  
        self.n_layers = n_layers  
  
        self.embedding = nn.Embedding(output_dim, emb_dim)  
        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers, dropout=dropout)  
        self.fc_out = nn.Linear(hid_dim, output_dim)  
        self.dropout = nn.Dropout(dropout)  
  
    def forward(self, input, hidden, cell):  
        # input = [batch size]  
        # hidden, cell = encoder의 결과값(hidden, cell)  
  
        # input = [1, batch size]  
        input = input.unsqueeze(0)  
  
        # embedded = [1, batch size, emb dim]  
        embedded = self.dropout(self.embedding(input))  
  
        output, (hidden, cell) = self.rnn(embedded, (hidden, cell))  
  
        # output = [seq len, batch size, hid dim * n directions]  
        # hidden = [n layers * n directions, batch size, hid dim]  
        # cell = [n layers * n directions, batch size, hid dim]  
  
        # prediction = [batch size, output dim]  
        prediction = self.fc_out(output.squeeze(0))  
  
        return prediction, hidden, cell
```

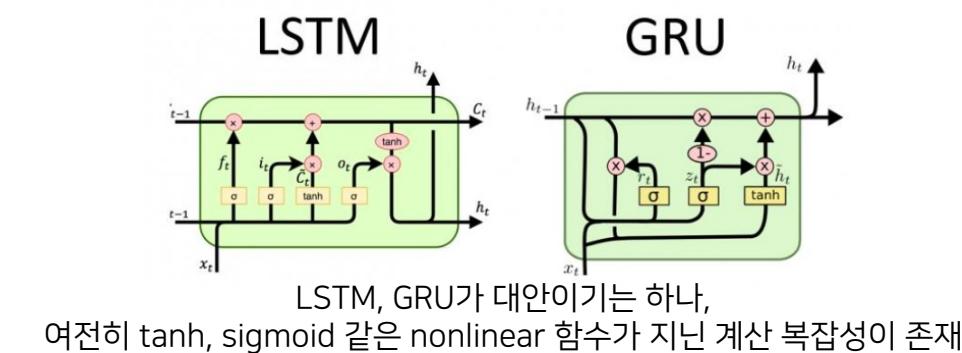
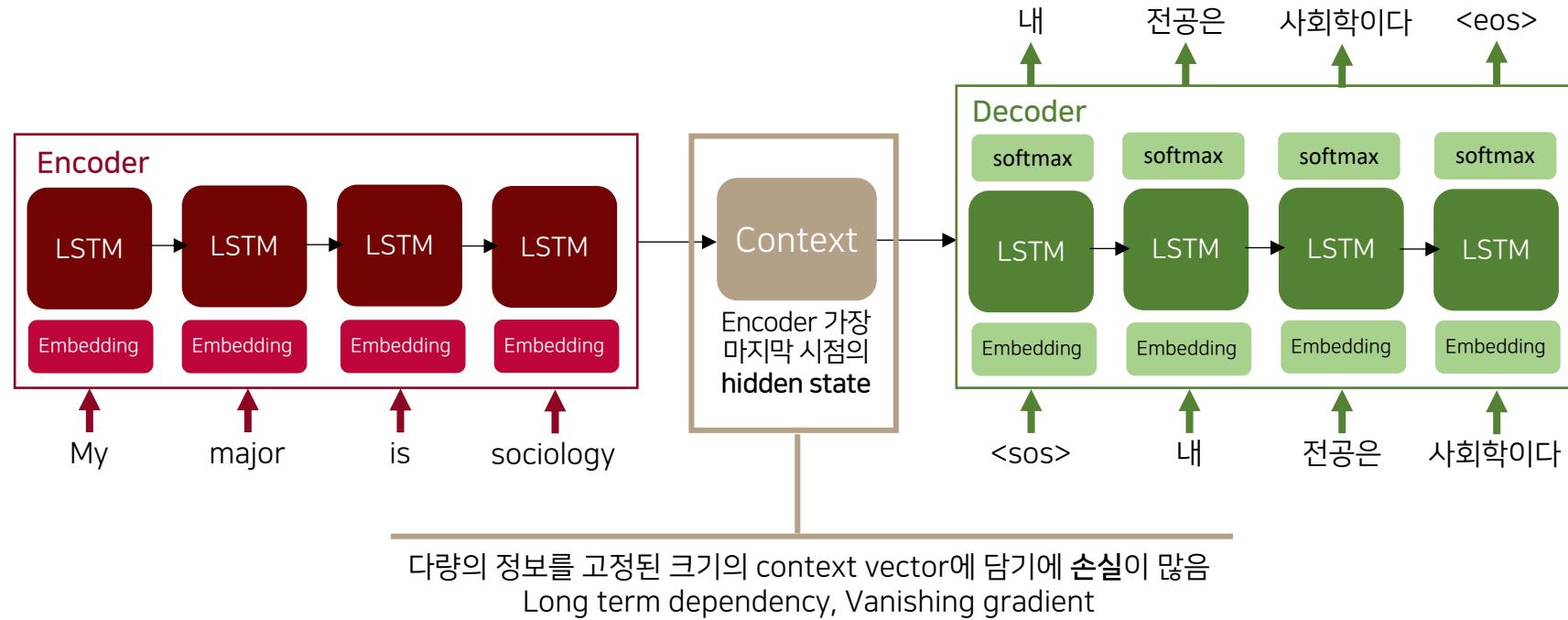
2-2. seq2seq Code



03 Attention

특정 단어에 “집중”하게 하려면?

3-1. Limitation of seq2seq



3-2. Attention

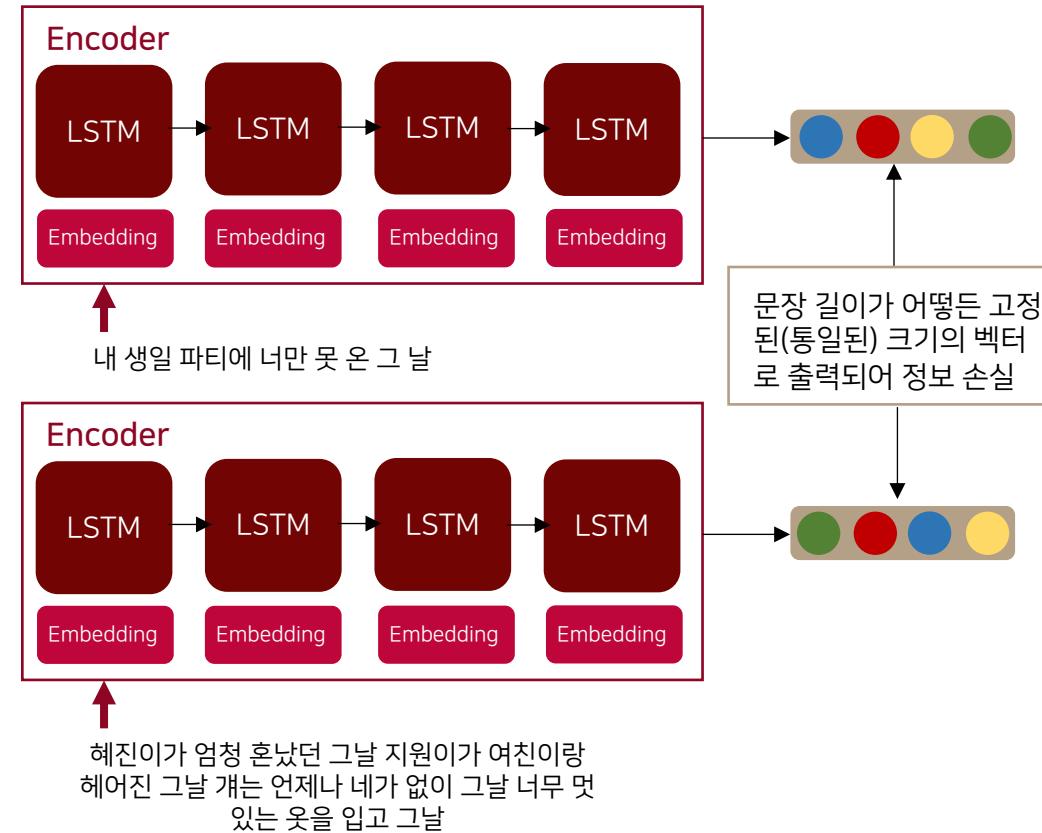
Sol) Attention:

Encoder의 최종 hidden state만 참고하는 것이 아니라 모든 time step의 hidden state를 참고함

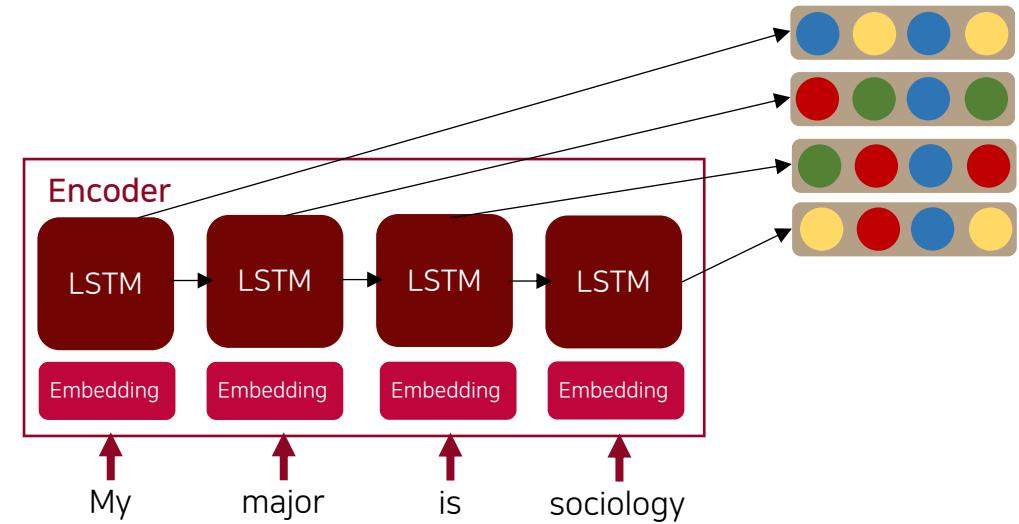
Decoder의 매 시점마다 그 시점에서 예측해야 할 단어와 연관이 있는 encoder의 hidden state만 집중(attention)해서 참고함



seq2seq



Attention



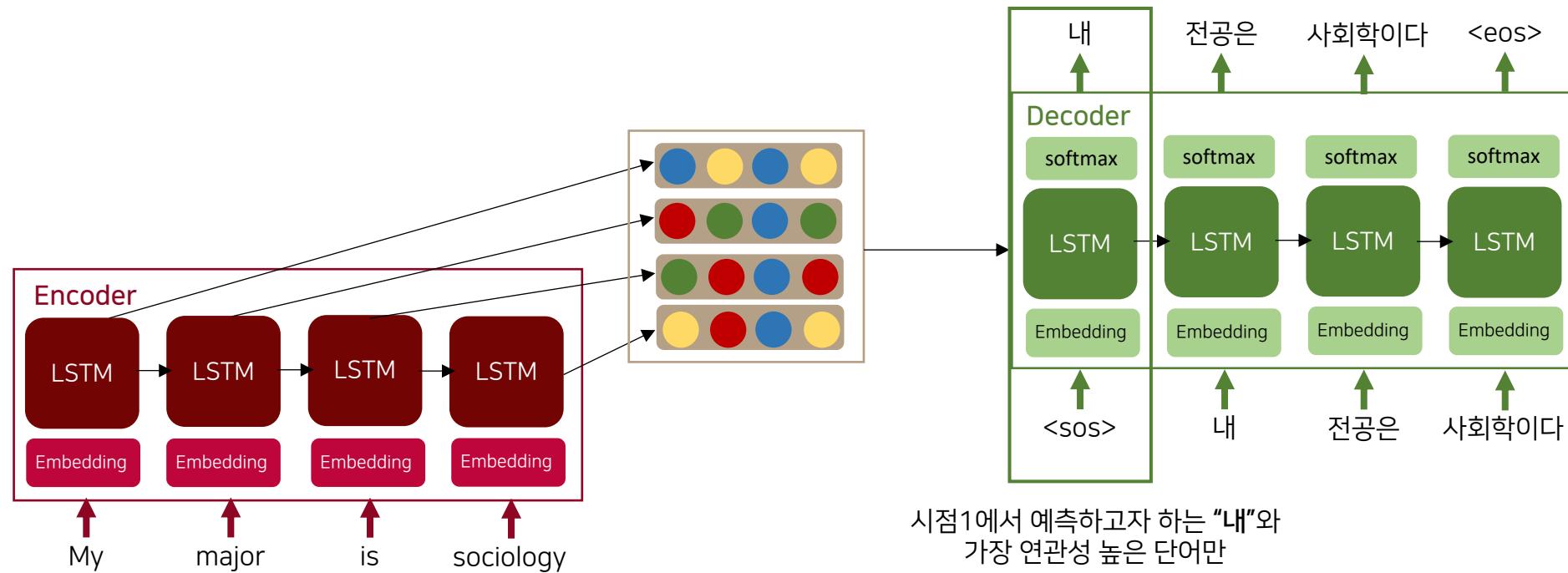
Encoder의 최종 hidden state만 참고하는 것이 아니라
매 time step의 hidden state를 참고함
참고) biLSTM이 더욱 효과적으로 쓰이나, 편의상 도식화에선 단방향으로

3-2. Attention

Sol) **Attention**:

Encoder의 최종 hidden state만 참고하는 것이 아니라 매 time step의 hidden state를 참고함

Decoder의 매 시점마다 그 시점에서 예측해야 할 단어와 연관이 있는 hidden state만 집중(attention)해서 참고함

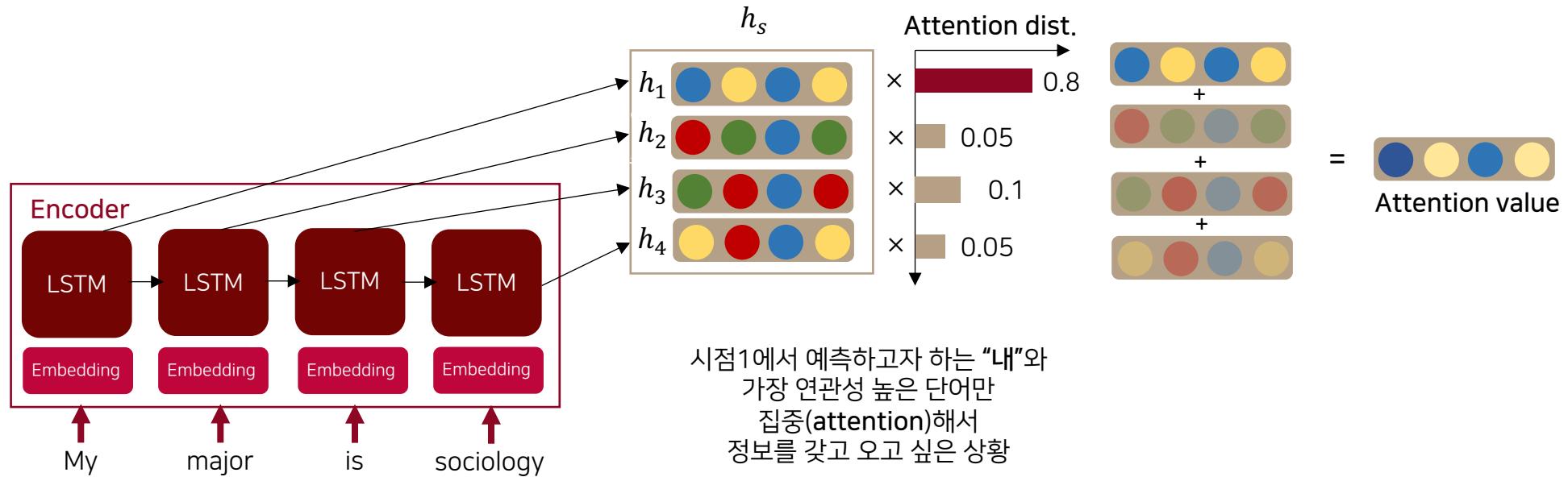


시점1에서 예측하고자 하는 “내”와
가장 연관성 높은 단어만
집중(attention)해서
정보를 갖고 오고 싶은 상황

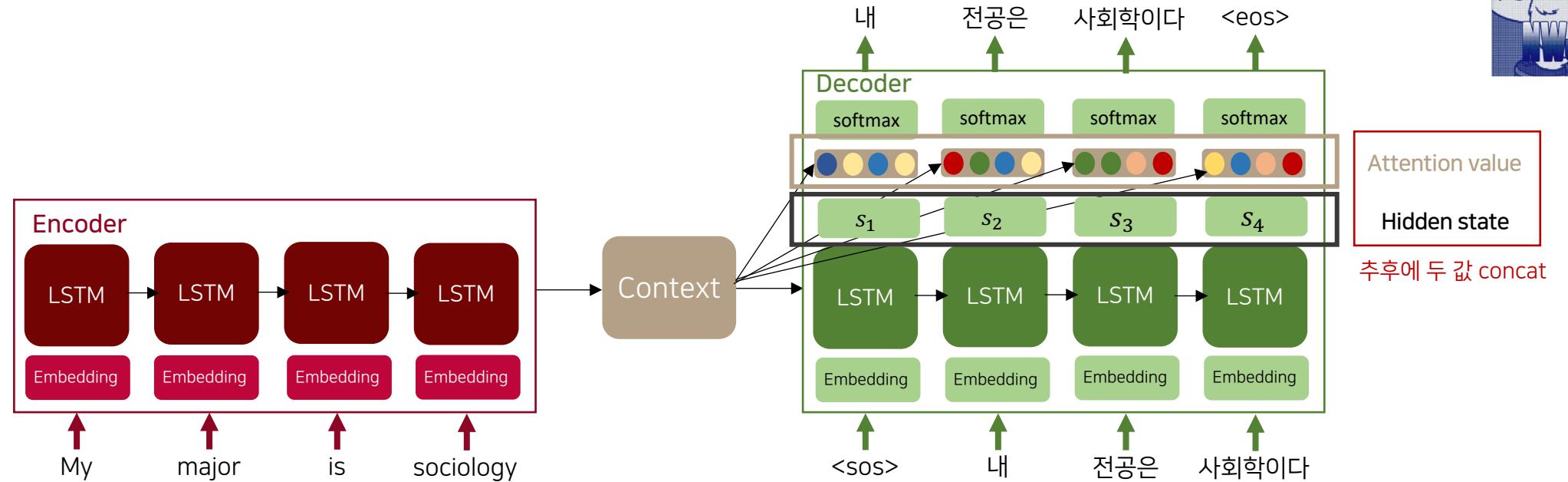
Sol) **Attention**:

Encoder의 최종 hidden state만 참고하는 것이 아니라 매 time step의 hidden state를 참고함

Decoder의 매 시점마다 그 시점에서 예측해야 할 단어와 연관이 있는 hidden state만 집중(attention)해서 참고함



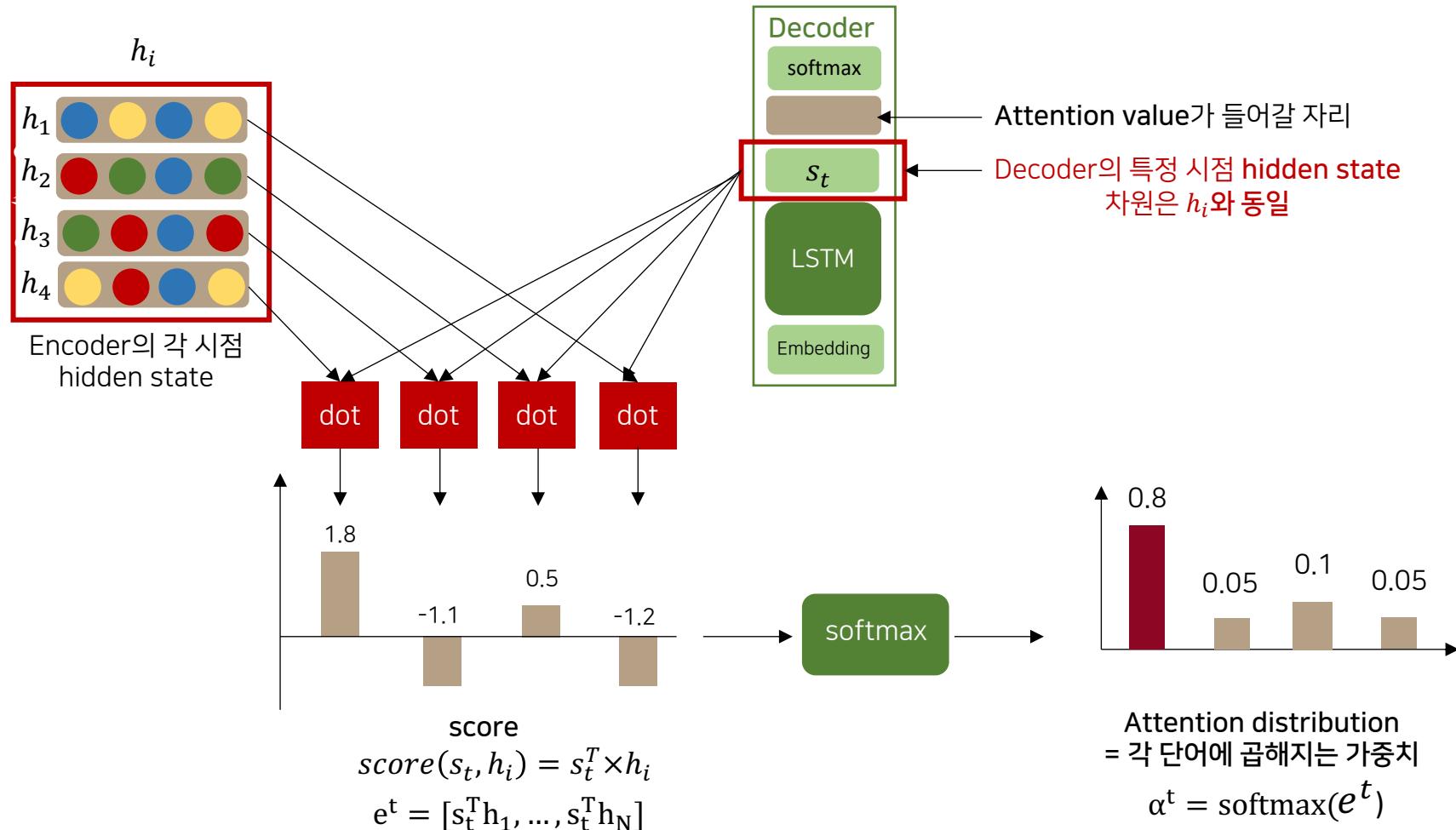
3-2. Attention



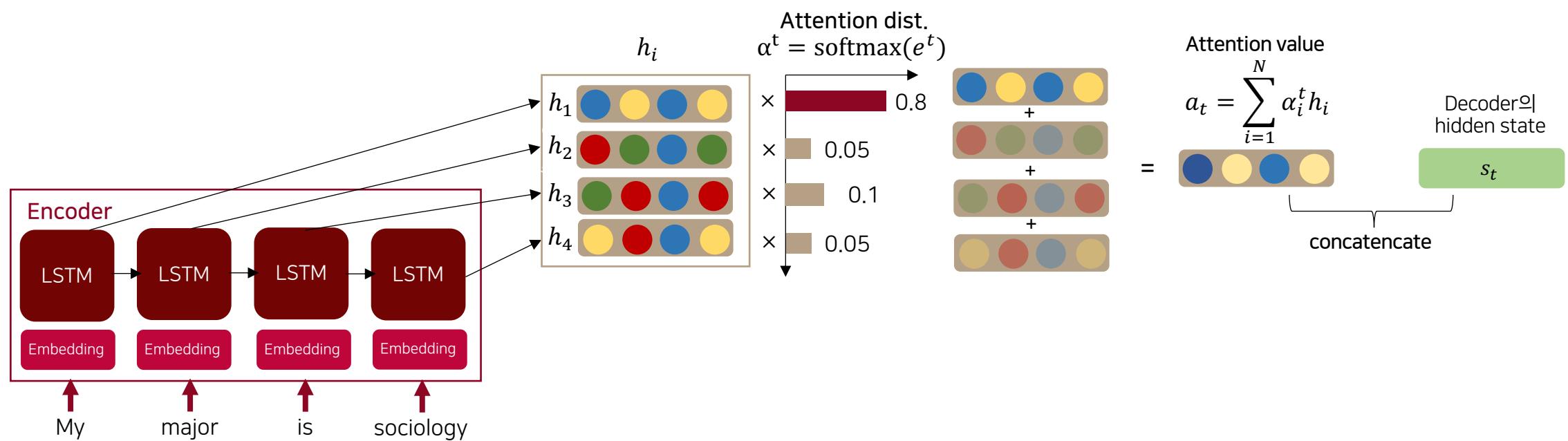
Decoder의 특정 시점에 input으로 들어가는 것들
1) hidden state
2) Attention value
3) Embedding vector

3-3. Attention score, distribution, value

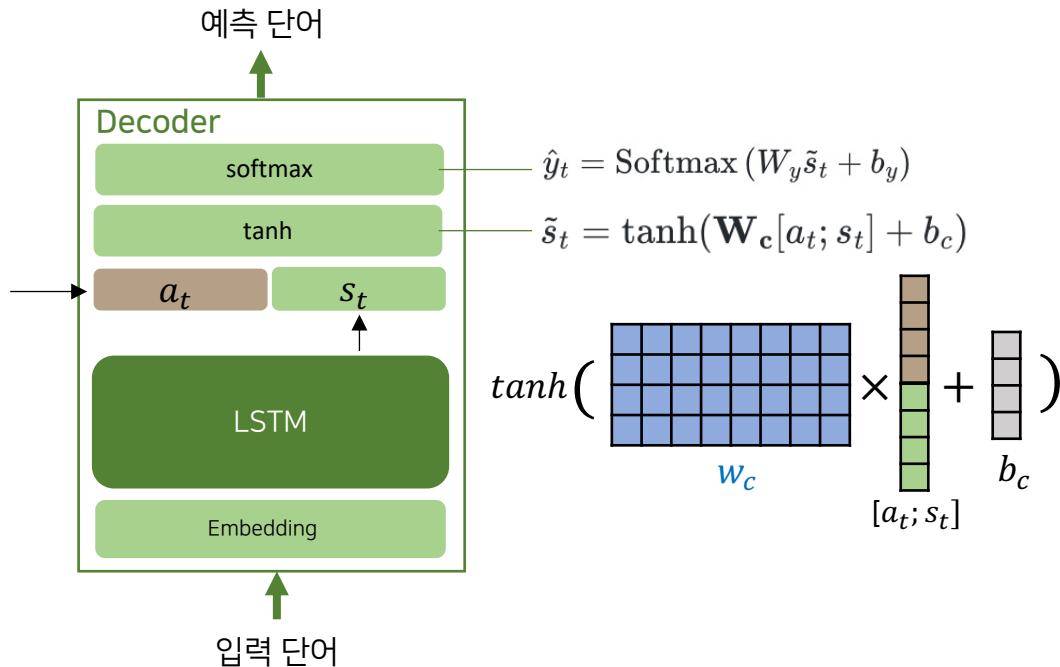
각 단어에 곱하는 가중치(attention distribution)는 어떻게 구하는가?



3-3. Attention score, distribution, value



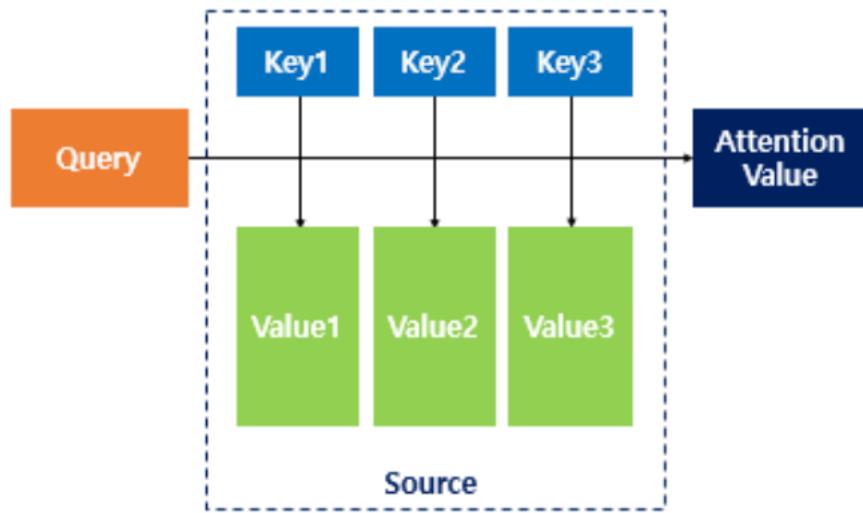
3-3. Attention score, distribution, value



- Score를 구하는 방식에 따라 attention 종류가 나뉨
- 앞선 예시에선 dot 방식을 사용하였음

이름	스코어 함수	Defined by
dot	$score(s_t, h_i) = s_t^T h_i$	Luong et al. (2015)
scaled dot	$score(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{n}}$	Vaswani et al. (2017)
general	$score(s_t, h_i) = s_t^T W_a h_i$ // 단, W_a 는 학습 가능한 가중치 행렬	Luong et al. (2015)
concat	$score(s_t, h_i) = W_a^T \tanh(W_b[s_t; h_i]), score(s_t, h_i) = W_a^T \tanh(W_b s_t + W_c h_i)$	Bahdanau et al. (2015)
location - base	$\alpha_t = \text{softmax}(W_a s_t)$ // α_t 산출 시에 s_t 만 사용하는 방법.	Luong et al. (2015)

3-4. Query, Keys, Values

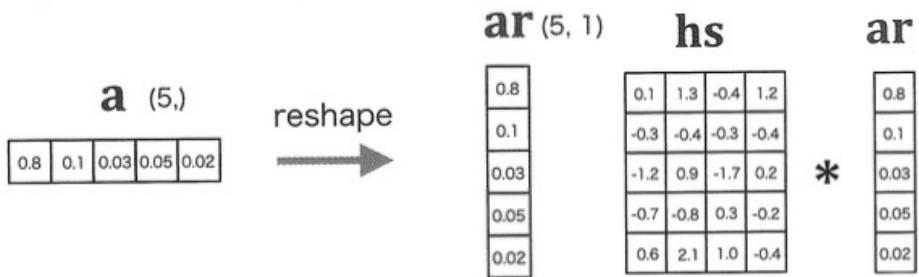
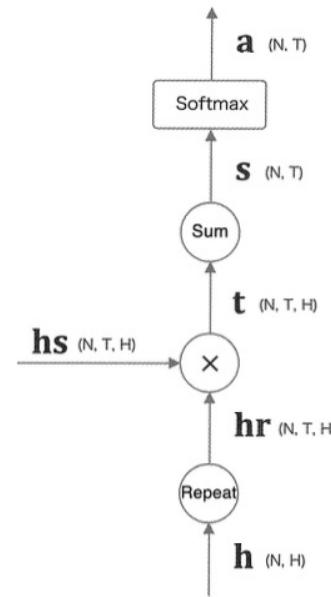


$$\text{Attention}(Q, K, V) = \text{Attention Value}$$

- Q(Query): t 시점의 decode의 hidden state
- K(Keys): 모든 시점의 encoder의 hidden state(유사도 비교 대상)
- V(Values): 모든 시점의 encoder의 hidden state(가중치를 곱해줄 대상)

사진: 딥러닝을 이용한 자연어 처리 입문

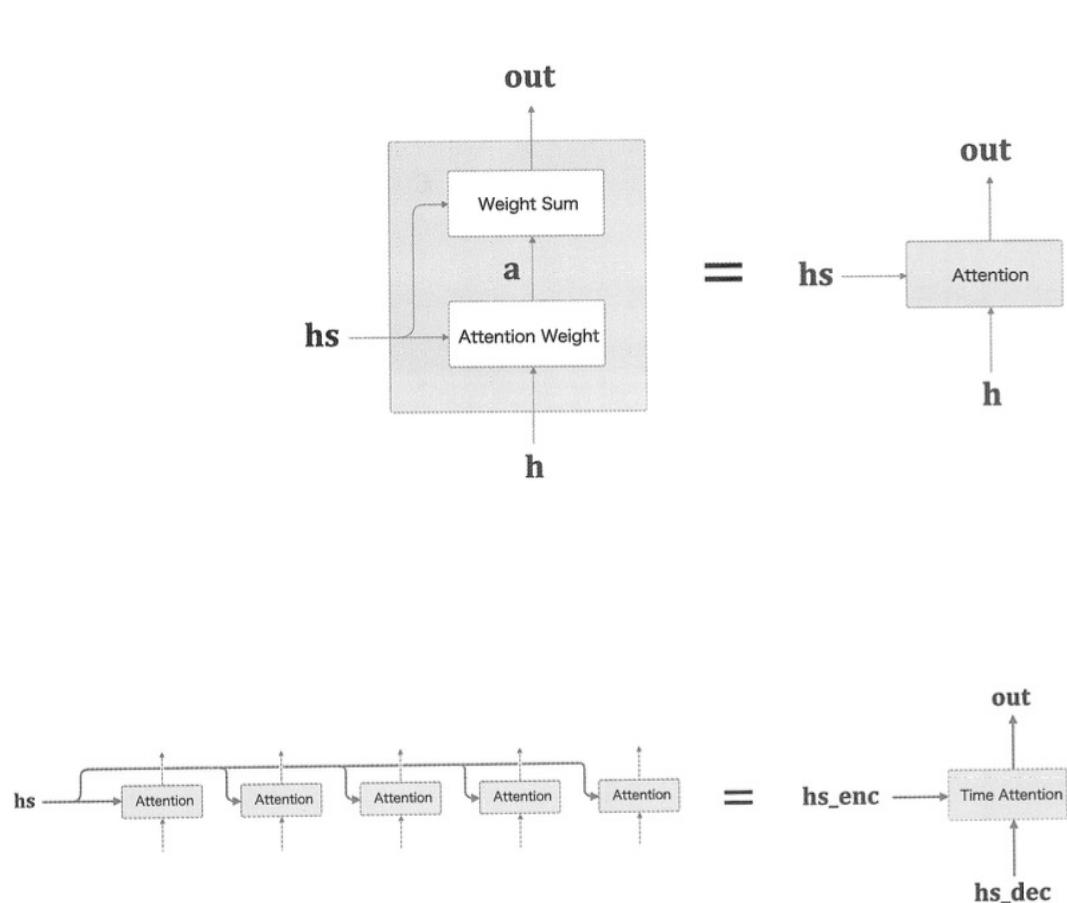
3-5. Attention Code



```
class AttentionWeight:  
    def __init__(self):  
        self.params, self.grads = [], []  
        self.softmax = Softmax()  
        self.cache = None  
  
    def forward(self, hs, h):  
        N, T, H = hs.shape  
  
        hr = h.reshape(N, 1, H).repeat(T, axis=1)  
        t = hs * hr  
        s = np.sum(t, axis=2)  
        a = self.softmax.forward(s)  
  
        self.cache = (hs, hr)  
        return a
```

```
class WeightSum:  
    def __init__(self):  
        self.params, self.grads = [], []  
        self.cache = None  
  
    def forward(self, hs, a):  
        N, T, H = hs.shape  
  
        ar = a.reshape(N, T, 1)  
        t = hs * ar  
        c = np.sum(t, axis=1)  
  
        self.cache = (hs, ar)  
        return c
```

3-5. Attention Code



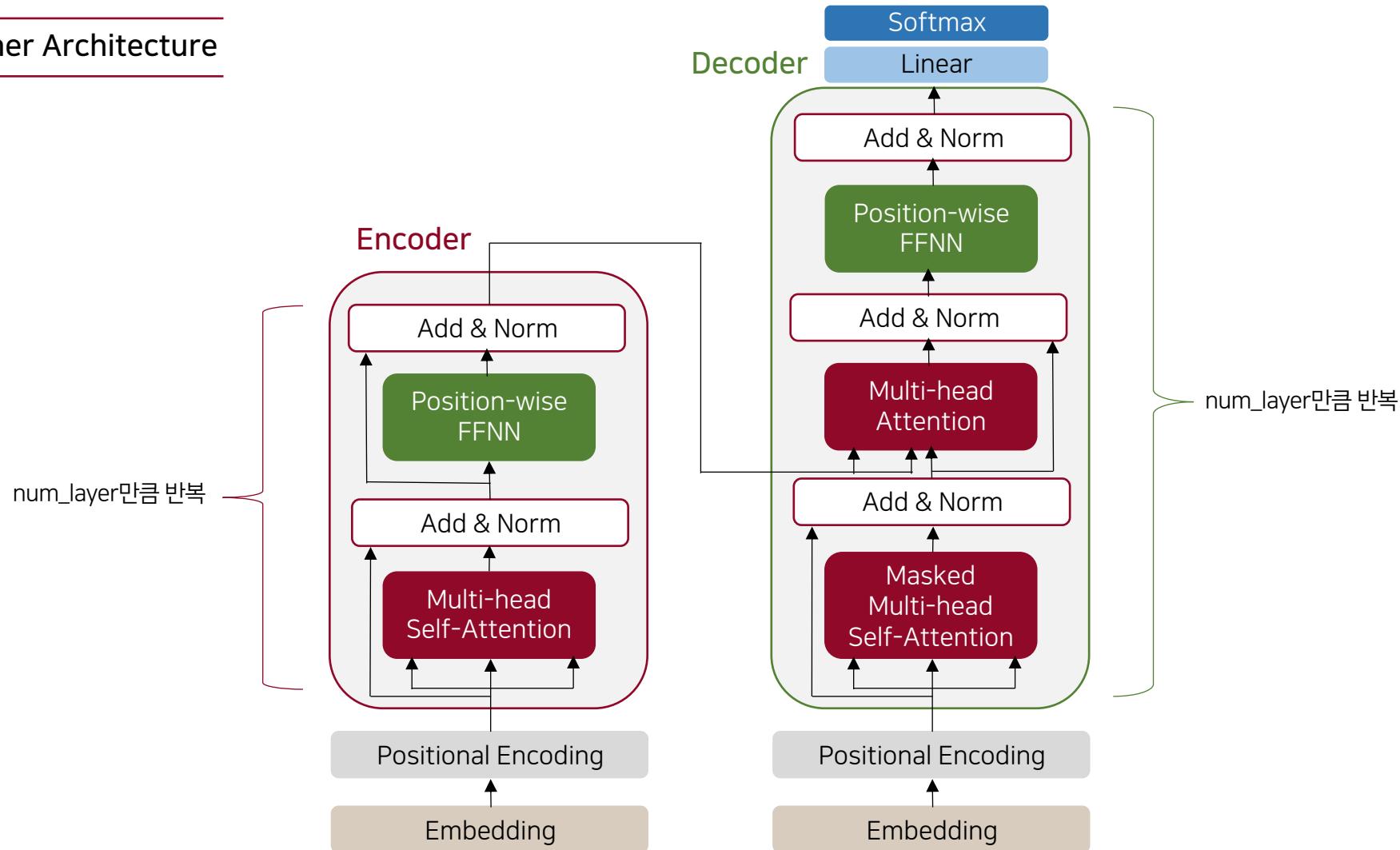
```
class Attention:  
    def __init__(self):  
        self.params, self.grads = [], []  
        self.attention_weight_layer = AttentionWeight()  
        self.weight_sum_layer = WeightSum()  
        self.attention_weight = None  
  
    def forward(self, hs, h):  
        a = self.attention_weight_layer.forward(hs, h)  
        out = self.weight_sum_layer.forward(hs, a)  
        self.attention_weight = a  
        return out  
  
class TimeAttention:  
    def __init__(self):  
        self.params, self.grads = [], []  
        self.layers = None  
        self.attention_weights = None  
  
    def forward(self, hs_enc, hs_dec):  
        N, T, H = hs_dec.shape  
        out = np.empty_like(hs_dec)  
        self.layers = []  
        self.attention_weights = []  
  
        for t in range(T):  
            layer = Attention()  
            out[:, t, :] = layer.forward(hs_enc, hs_dec[:, t, :])  
            self.layers.append(layer)  
            self.attention_weights.append(layer.attention_weight)
```

04 Transformer

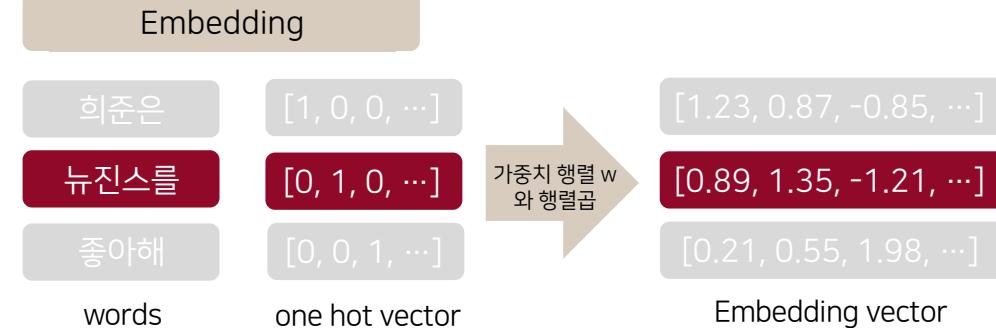
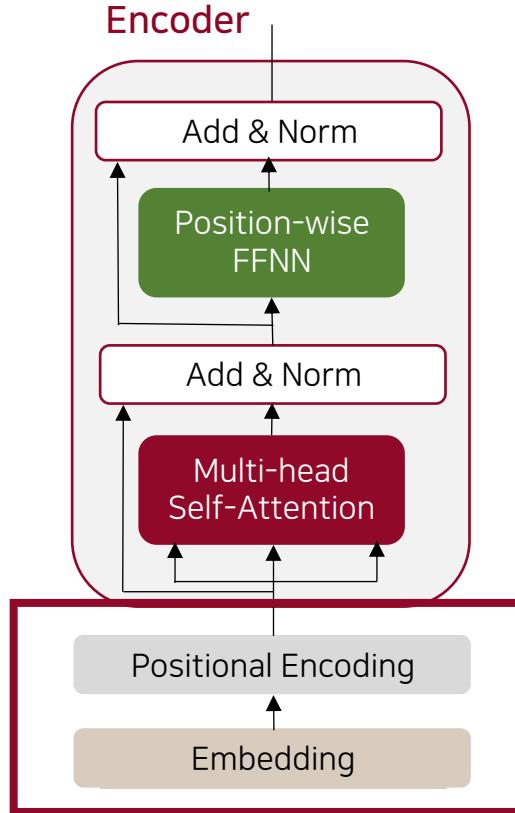
Attention만으로 encoder와 decoder 만들기

4-1. Transformer Architecture

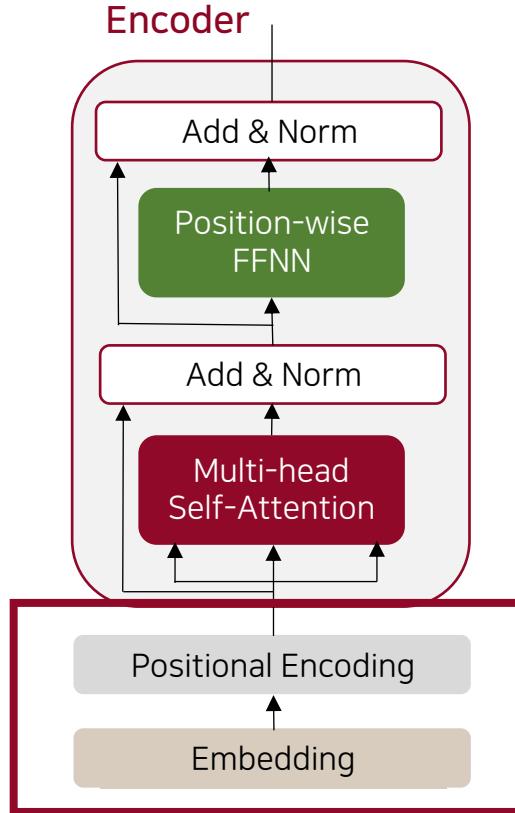
Transformer Architecture



4-2. Embedding Layer



4-2. Embedding Layer



Positional Encoding

- Seq2seq: 각 word token이 순서대로 입력됨
- Transformer: word token이 한꺼번에 입력됨. 순서 정보가 보존되지 않음

순서 정보가 약간이라도 뒤틀리면 아예 다른 의미로 변모함

- Although I did **not** get good grades in the Chinese character test, I was able to graduate
- Although I did get good grades in the Chinese character test, I was **not** able to graduate



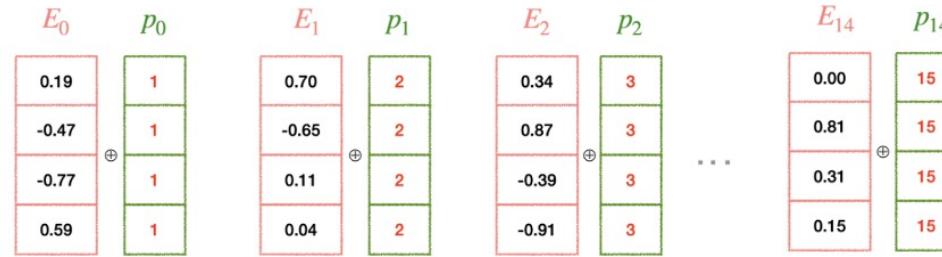
참고) Concatenate가 아닌 sum 연산을 하는 이유

- Concatenate 연산 시 의미 벡터와 위치 벡터가 서로 각자의 차원 공간을 갖게 됨
- 따라서 정보가 섞이는 혼란을 피할 수는 있지만, 메모리, 비용 문제가 발생
- 논문 publish 당시엔 GPU power가 지금만하지 않았으므로 비용 문제를 회피하려 sum 연산한 것으로 보임

4-2. Embedding Layer

방법1) 시퀀스 크기에 비례하여 커지는 정수 벡터 부여

- 임베딩 벡터에 비해 위치 정보가 비대하게 커짐



방법3) sine, cosine 주기함수를 통해 방법1,2의 단점 보완

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

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

this is my car

E_0	p_0	E_1	p_1	E_2	p_2	E_3	p_3
0.19	0.2	0.70	0.09	0.34	-0.1	0.69	-0.3
-0.47	0.6	-0.65	0.7	0.87	0.8	0.87	0.9
-0.77	0.98	0.11	0.80	-0.39	0.71	-0.39	0.65
0.59	0.01	0.04	0.02	-0.91	0.03	0.44	0.04

방법2) 한 시퀀스 당 첫 토큰에 0, 마지막 토큰에 1 부여하고, 그 사이를 정규화

- 같은 위치더라도 시퀀스 길이에 따라 다른 벡터값이 부여됨

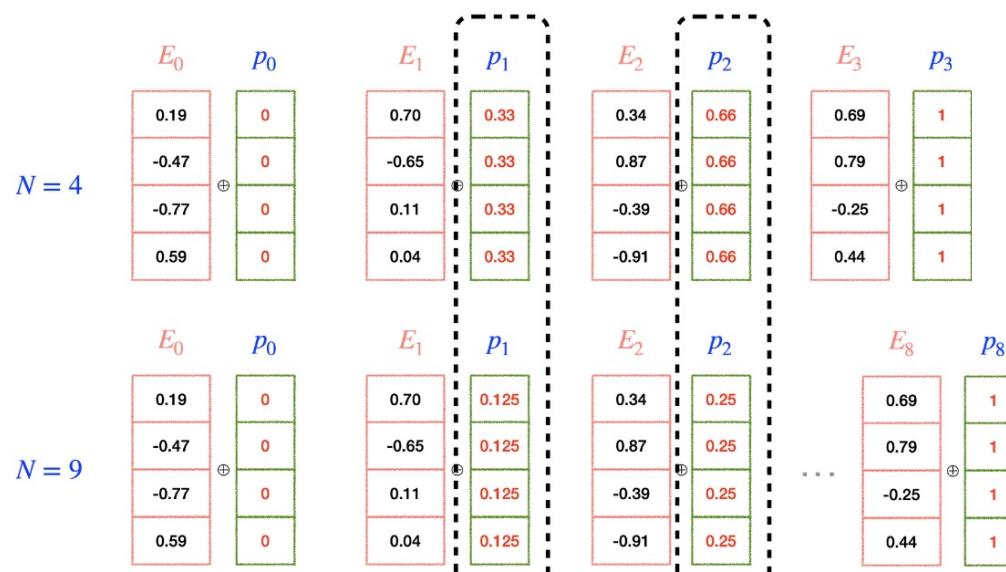
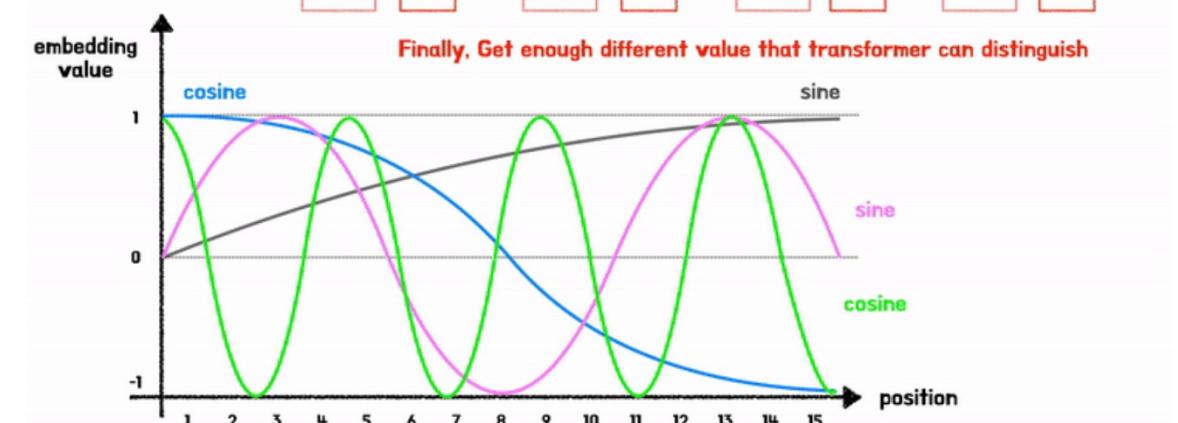
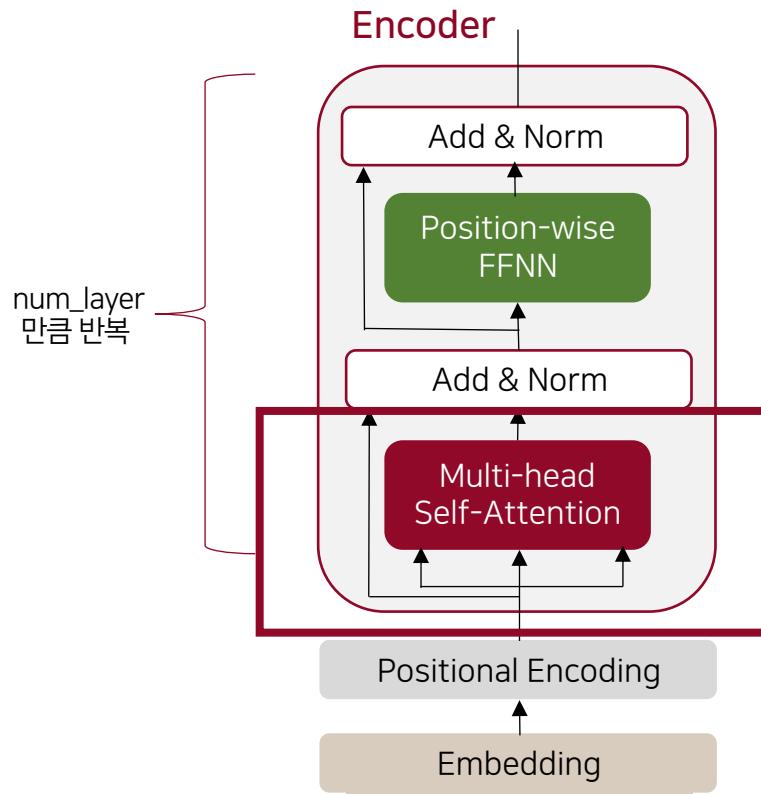


사진: <https://www.blossominkyung.com/deeplearning/transformer-positional-encoding>



4-2. Encoder



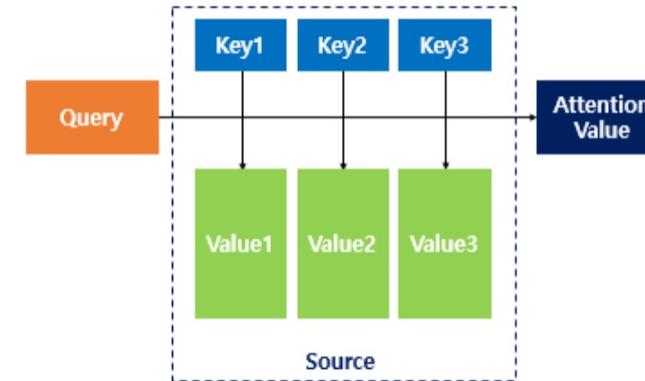
Multi-head
Self-Attention

What is Self-Attention?

Encoder의 token들

heejun loves newjeans and IVE too

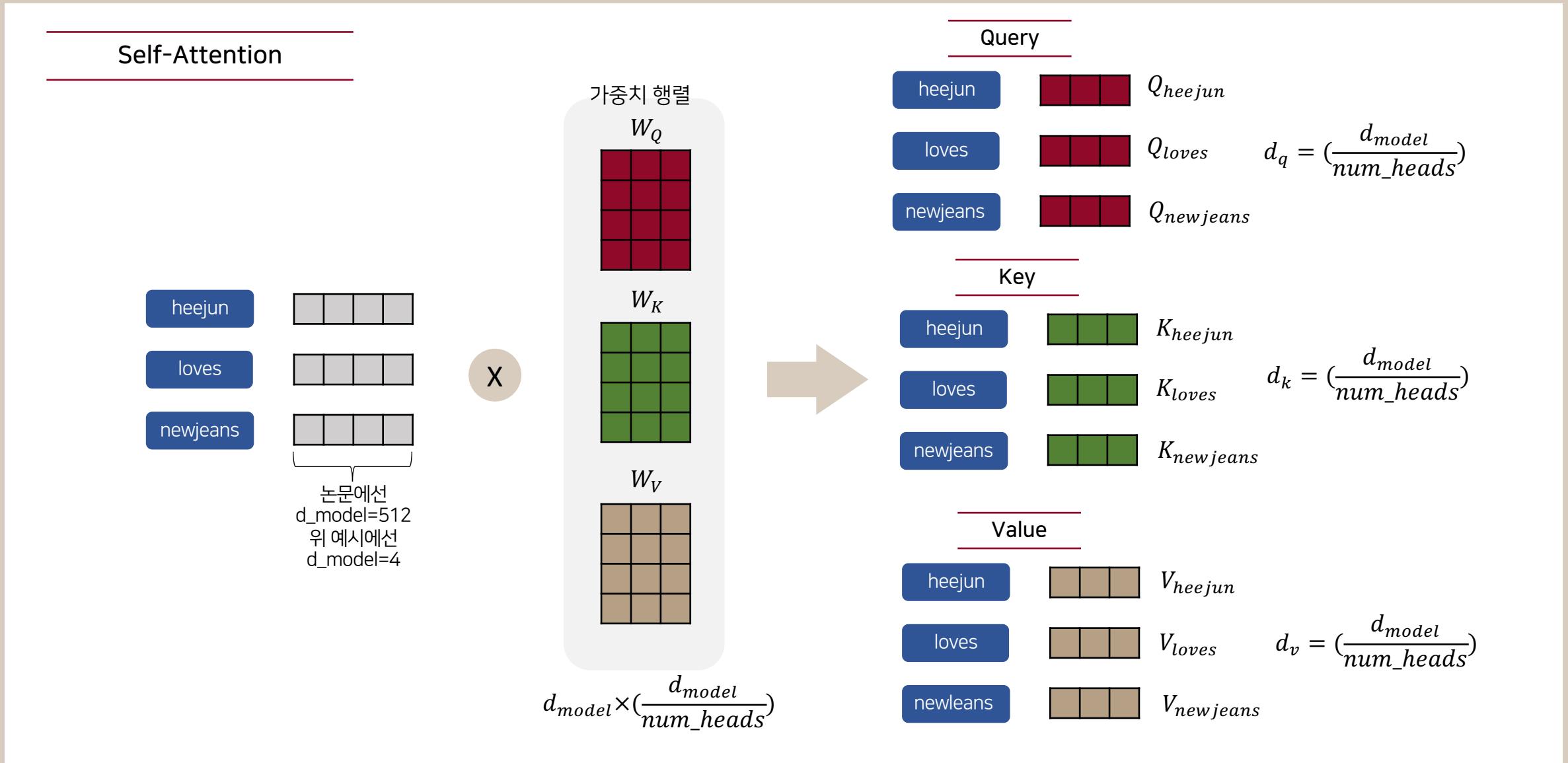
Encoder의 token들 간의 유사도도 파악할 필요가 있음
=> Self-Attention 활용



Self Attention의 경우, Q,K,V의 출처가 모두 같음!
출처가 같을 뿐, 아예 같은 값이란 의미는 아님.

- Q(Query): 입력 문장의 모든 단어들
- K(Keys): 입력 문장의 모든 단어들
- V(Values): 입력 문장의 모든 단어들

4-2. Encoder



Self-Attention

$$Q \quad \begin{matrix} \text{heejun} \\ \text{really} \\ \text{loves} \\ \text{newjeans} \end{matrix} \quad \text{seq_len} \times d_q$$

$$k^T \quad \begin{matrix} \text{heejun} \\ \text{really} \\ \text{loves} \\ \text{newjeans} \end{matrix}$$

÷

$\sqrt{d_k}$

dot-product attention이 아닌
scaled dot-product attention 사용
(내적 값이 커짐에 따른 기울기 소실을 방지)

$$\begin{matrix} \text{heejun} \\ \text{really} \\ \text{loves} \\ \text{newjeans} \end{matrix} \quad \begin{matrix} \text{heejun} \\ \text{really} \\ \text{loves} \\ \text{newjeans} \end{matrix}$$

Attention Score Matrix

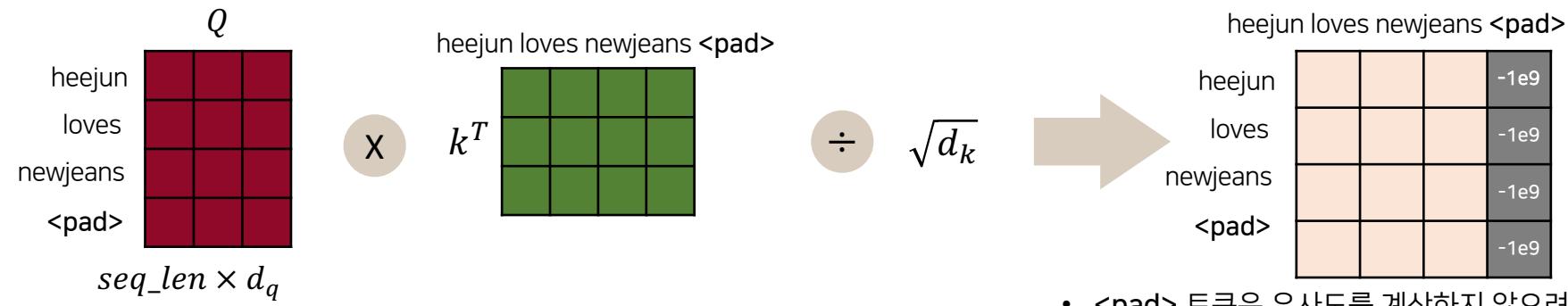
softmax

$$\text{softmax} \left(\frac{Q \times k^T}{\sqrt{d_k}} \right) \times V = a$$

Attention Value Matrix

$seq_len \times d_v$

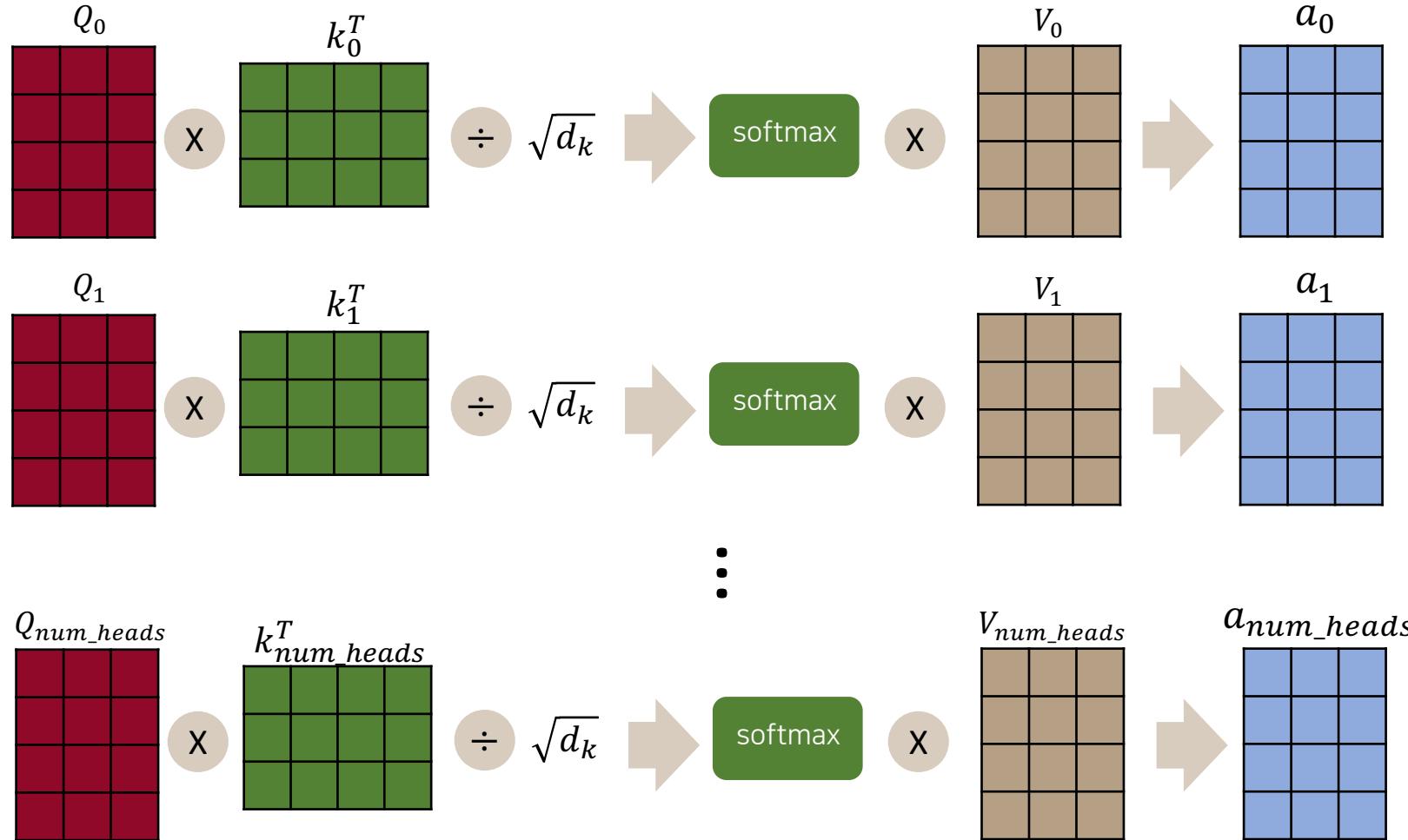
Self-Attention



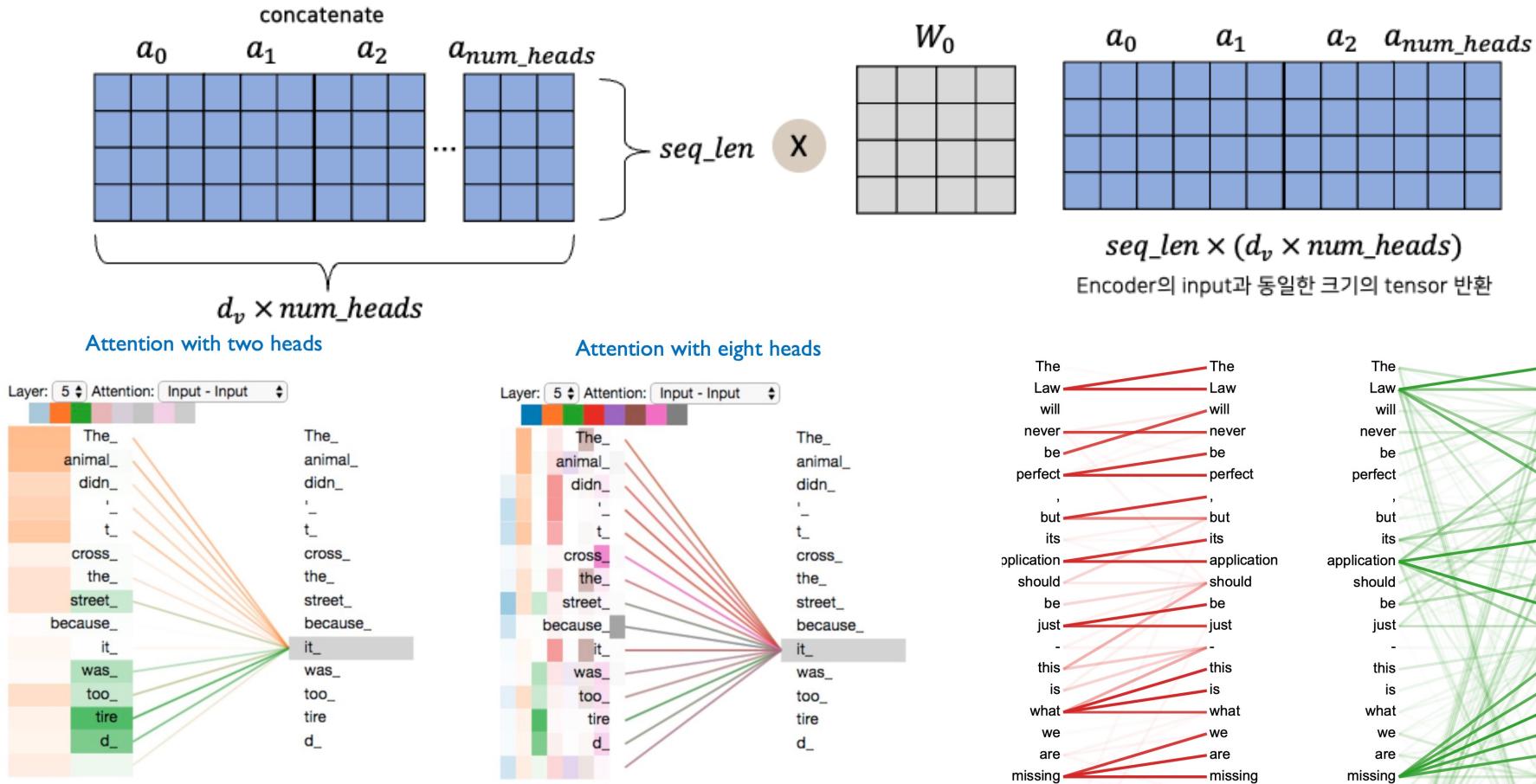
- <pad> 토큰은 유사도를 계산하지 않으려 Masking
- Token이 위치한 열(Key)에 Masking
- 매우 작은 음수값을 넣으면 softmax 함수를 거쳐 0으로 치환됨

4-2. Encoder

Multi-head Attention

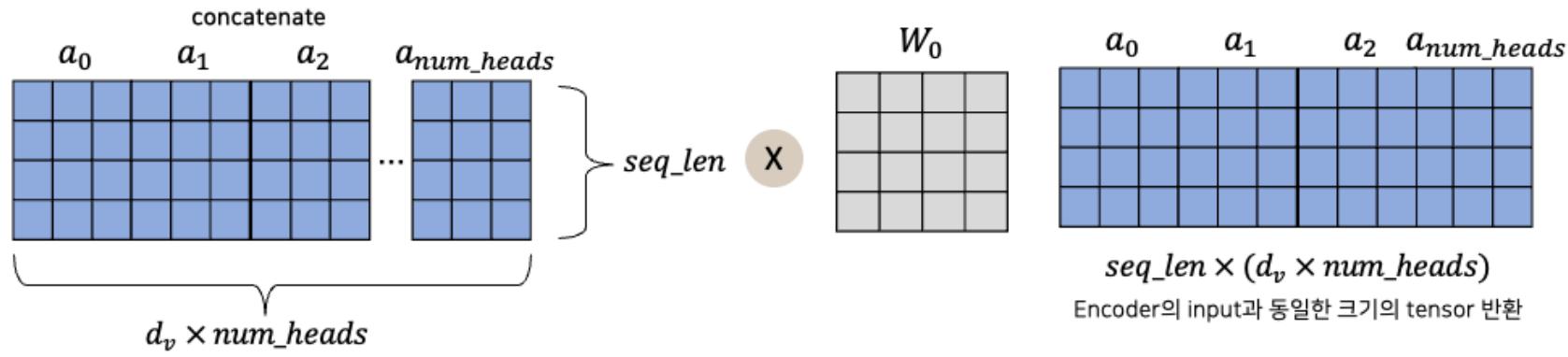


Multi-head Attention



- 서로 다른 attention-head가 집중하는 관점이 다름
- Multi-head로써 더 풍부한 attention이 가능해짐

Multi-head Attention



Which do you like better, coffee or tea?

- 문장 타입에 집중하는 어텐션

Which do you like better, coffee or tea?

- 명사에 집중하는 어텐션

Which do you like better, coffee or tea?

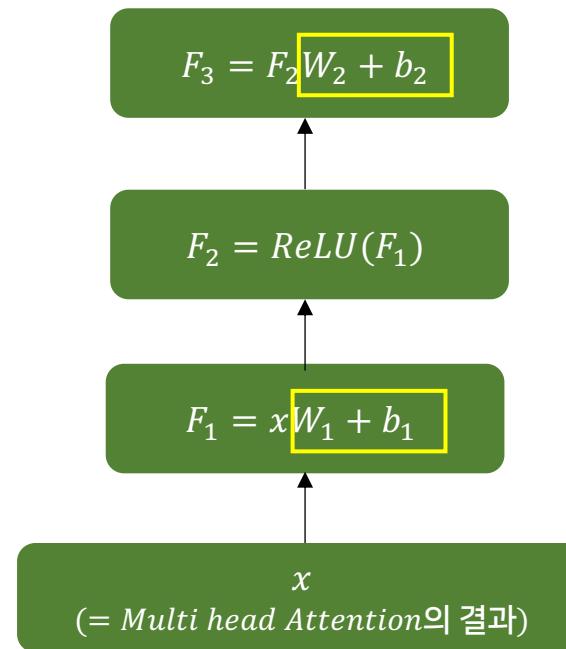
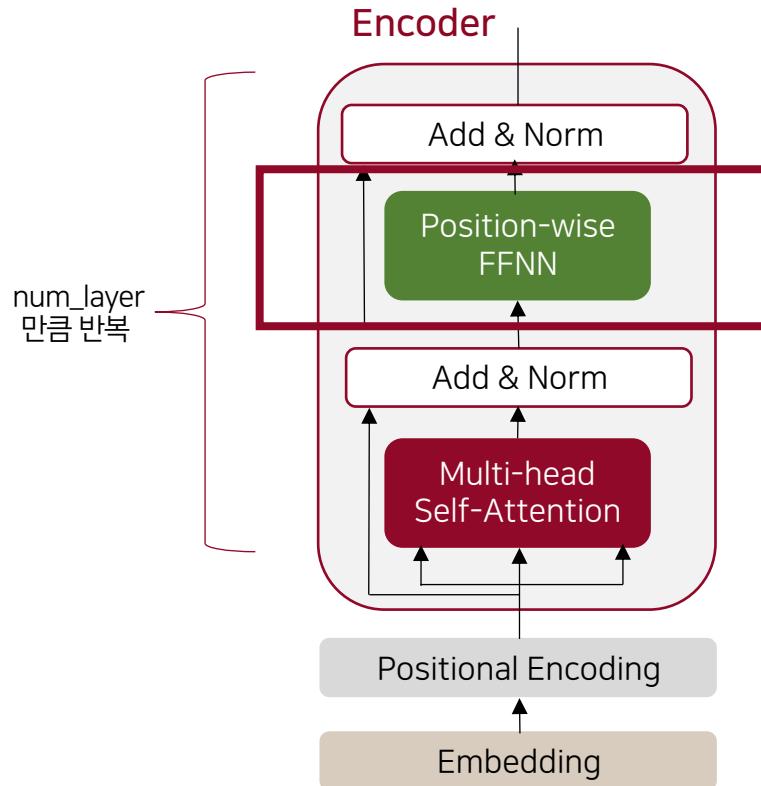
- 관계에 집중하는 어텐션

Which do you like better, coffee or tea?

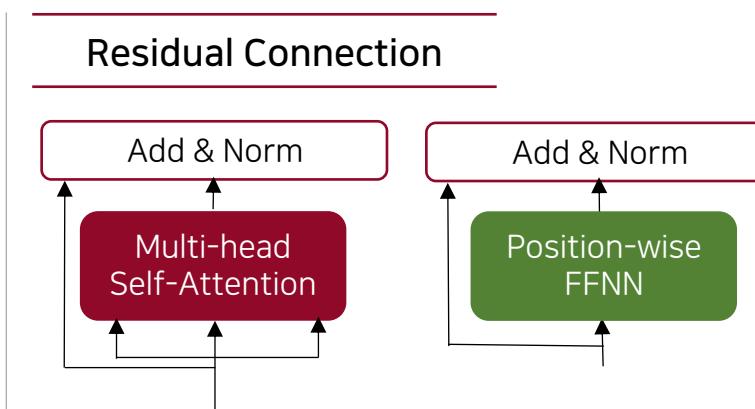
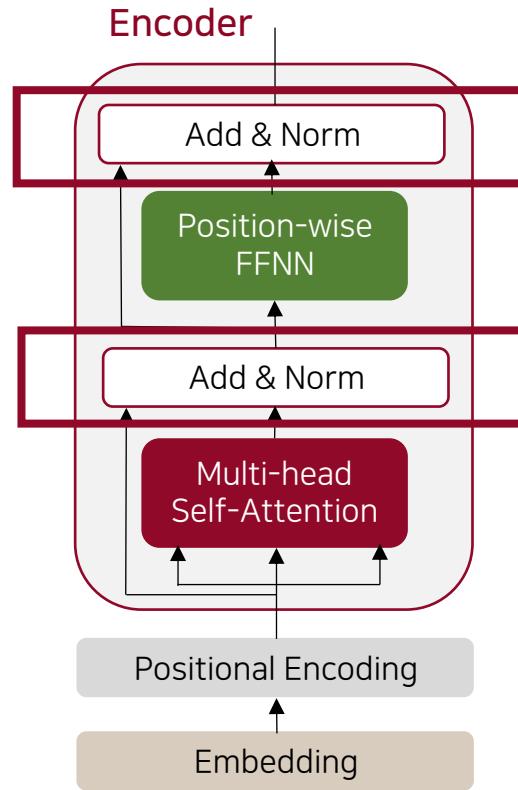
- 강조에 집중하는 어텐션

- 서로 다른 attention-head가 집중하는 관점이 다름
- Multi-head로써 더 풍부한 attention이 가능해짐

4-2. Encoder



- 여러 개의 attention-head는 각각의 관점에 따라 정보가 편향되어 있음. 이것들을 균등하게 섞는 역할 수행
- 각 가중치들은 하나의 인코더 내에서는 모든 문장에서 동일하게 share
- 다른 인코더 층끼리는 다른 가중치를 share

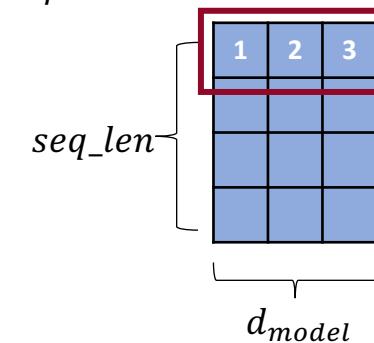


$$H(x) = x + \text{Multi-head Attention}(x)$$

Layer Normalization

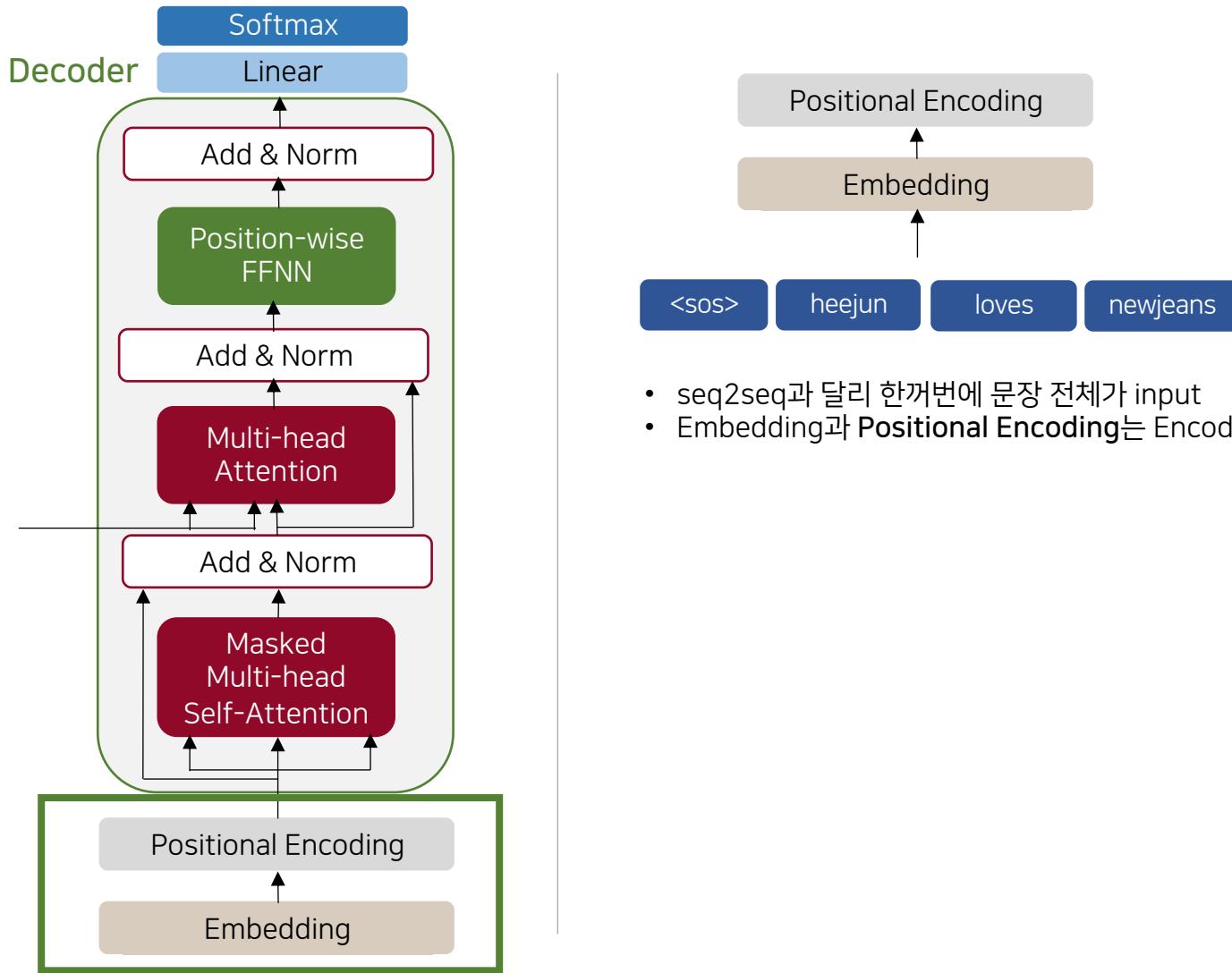
$$\text{LayerNorm}(x_i) = \gamma \hat{x}_i + \beta$$

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



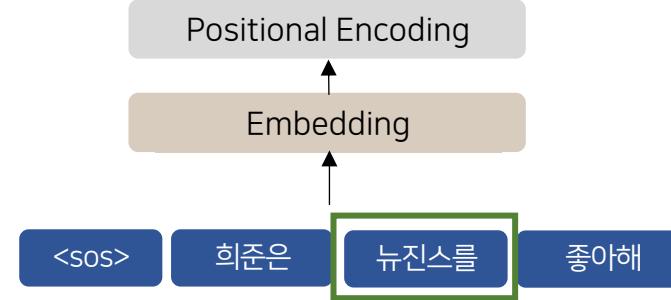
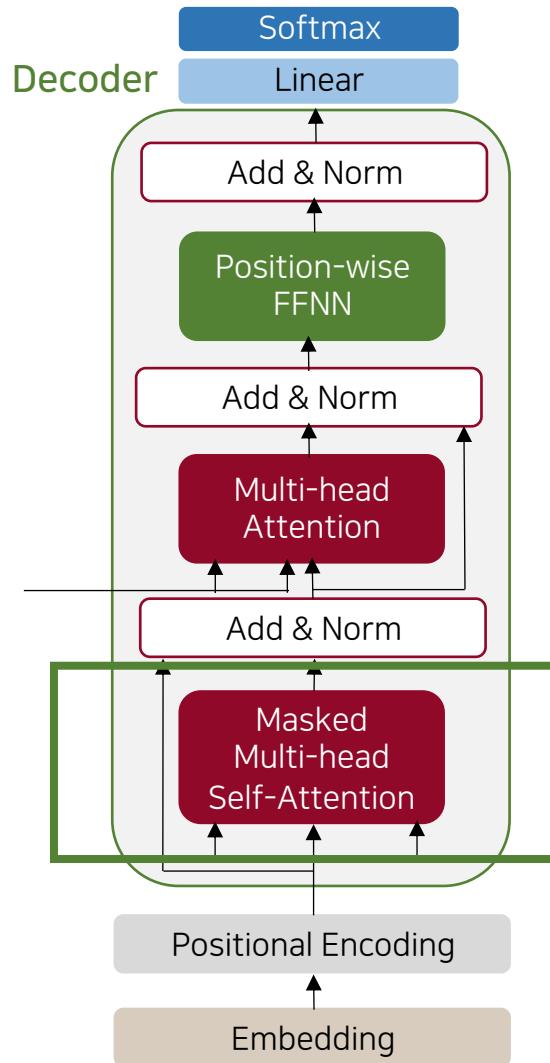
$$\gamma \left[\frac{1 - E(x_1)}{\sqrt{Var(x_i) + \epsilon}} \quad \frac{3 - E(x_1)}{\sqrt{Var(x_i) + \epsilon}} \right] + \beta$$

4-3. Decoder



- seq2seq과 달리 한꺼번에 문장 전체가 input
- Embedding과 Positional Encoding는 Encoder에서와 동일함

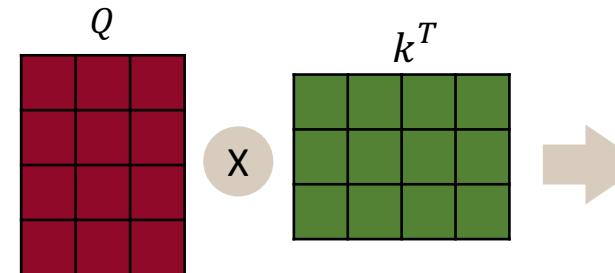
4-3. Decoder



<뉴진스를>을 예측해야 하는 시점을 가정해보자.

- Seq2seq라면 주어진 input은 <sos>, <희준은>
- Transformer는 미래 시점인 <좋아해>까지도 이미 알고 있음
- 미래의 단어를 미리 참고하지 못하게 Masking할 필요가 있음.

Look-ahead Mask

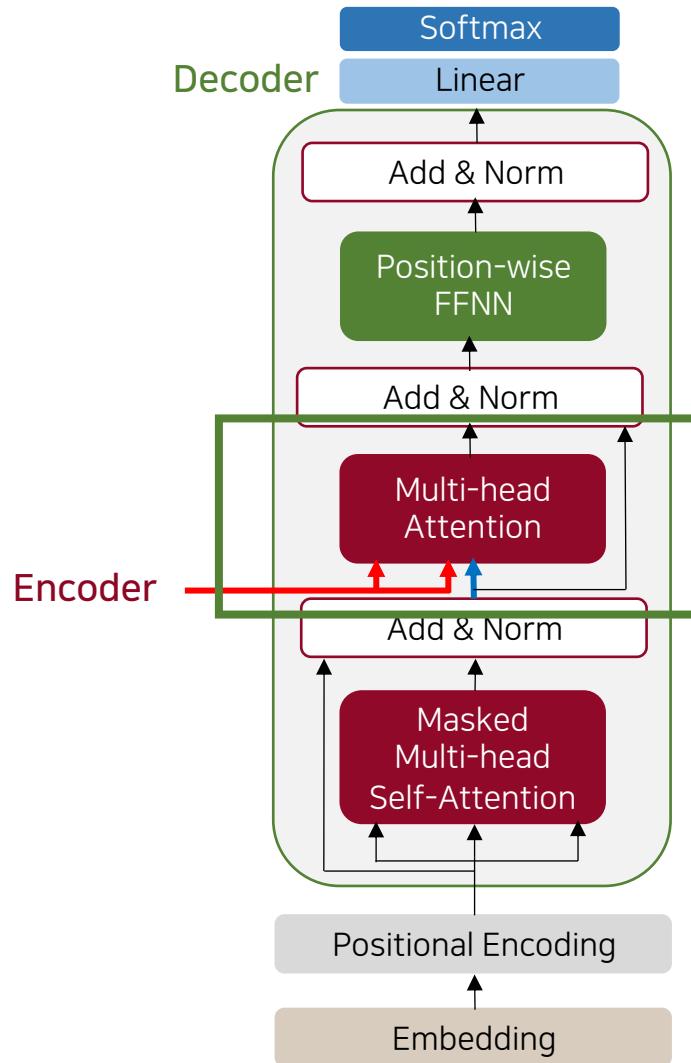


The **Attention Score Matrix** is a 4x4 grid representing the attention scores between tokens. The columns represent the input tokens (<sos>, 희준은, 뉴진스를, 좋아해) and the rows represent the output tokens (<sos>, 희준은, 뉴진스를, 좋아해). The diagonal elements are -1e9, indicating self-attention. The off-diagonal elements are 0, except for the row corresponding to '뉴진스를', which has non-zero values for the last three columns.

<sos>	-1e9	-1e9	-1e9
희준은		-1e9	-1e9
뉴진스를			-1e9
좋아해			

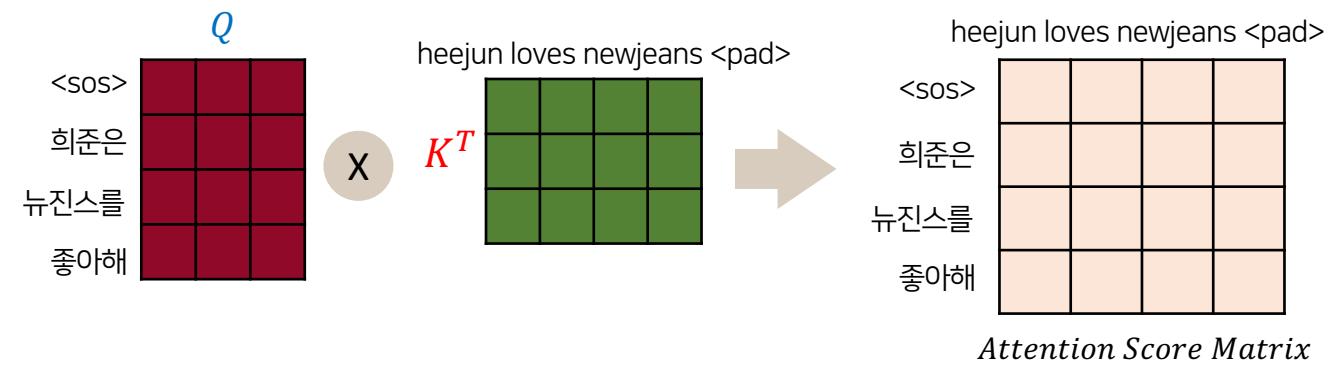
Attention Score Matrix

4-3. Decoder



Self Attention이 아니기 때문에, Q, K, V 값이 달라짐

- **Q(Query)**: decoder의 masked multi-head attention의 결과 행렬
- **K(Keys)**: encoder의 keys 행렬
- **V(Values)**: encoder의 Values 행렬



05 Announcement

Week2 예습과제 Review, week3 예복습 과제 안내, week4 진도 안내

5-1. 우수 예습과제 Review

주희님

WEEK3
예습과제1

Attention 기계번역

화면공유 하셔서 5분 내외로 가볍게 리뷰해주시면 됩니다!

5-2. Week4 예, 복습과제 안내, Week5 진도 안내



코드과제의 파일형식은 ipynb로, KUBIG 24-1 Github repo에 업로드 될 예정입니다!
Colab 환경에서 제작된 과제들이므로 [google colab](#)에서 실행하시는 것을 권장드립니다.

WEEK4 복습과제1

Transformer
chatbot 구현

WEEK4 예습과제1

BERT Pytorch

WEEK5 진도

- BERT

WEEK5 진도 해당 범위(읽어오시길 권장 드립니다!)

- [딥러닝을 이용한 자연어 처리 입문]
- Ch17. BERT

E.O.D
수고하셨습니다!