

KUBIG 25-W
겨울방학 BASIC STUDY SESSION

NLP SESSION

WEEK4

01 Announcement, 복습과제 우수 코드 review

02 seq2seq

03 Attention

04 Transformer

05 Announcement

01 Announcement,
우수 복습과제 선정

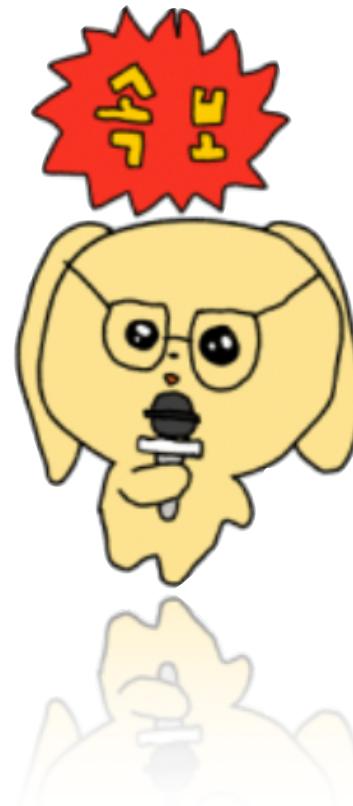


!! WEEK4 과제 *NEW*

- WEEK4 과제 (예습, 복습 과제)
- <Attention Is All You Need> paper review

복습, 예습 과제는 그대로 해주시고,

<Attention is all you need> 논문은 팀원들과 분량을 나누어 자유롭게 리뷰해주시면 됩니다(:)



💡 WEEK5 (2/6 목요일) 대면 세션 진행

- 대면 참여가 불가능한 분들에겐 줌 링크 제공 예정
 - ! 캠 켜야 출석 인정
- 세션 후 각 팀별로 중간발표 진행
 - 프로젝트 소개 및 진행과정 공유
- CV 세션과 함께 뒤풀이

20기 김유진님

20기 김재훈님

WEEK3
복습과제1

WEEK3
복습과제2

RNN, LSTM

네이버 쇼핑 리뷰 감성분석
with RNN, LSTM, GRU

02 seq2seq

특정 도메인의 문장을 다른 도메인의 문장으로

왜, Attention과 Transformer가 중요한가?

Attention Is All You Need

시간 흐름에 따른 관심도 변화 help_outline

file_download code share



2017년 논문 발표 이후에도
꾸준한 관심을 받는 Transformer

2-0. Why is it important?

디지털 이미지는 사진이나 그림을 디지털 형태로 표현한 것을 말한다. 이러한 디지털 이미지는 최소 단위의 점인 **화소**로 구성되며, 각 화소에는 밝기나 색상 등을 나타내는 값이 부여되어 있다. 일반적으로 화소의 수가 많을수록 해상도는 높아지지만 대신 저장되는 데이터의 용량은 커지게 된다. 따라서 이러한 디지털 이미지를 효율적으로 저장하고 전송하기 위해서는 데이터의 용량을 줄여 주는 디지털 이미지 압축 기술이 필요하다.

디지털 이미지 압축 기술에는 무손실 압축과 손실 압축이 있다. 무손실 압축은 압축 과정에서 데이터를 손실시키는 방법을 사용하지 않고 압축이 진행되기 때문에 압축 효율은 떨어지지만, 원본과 동일한 이미지로 복원이 가능하다. 반면 손실 압축은 중복되거나 필요치 않은 데이터를 제거하여 원본과 동일한 이미지로 복원하기는 어렵지만, 무손실 압축에 비해 수 배에서 수천 배 이상의 높은 압축 효율을 얻을 수 있어 보편적인 압축 기술로 활용되고 있다.

텍스트를 읽을 때… 🤔

전체를 하나하나 집중해서 (X)

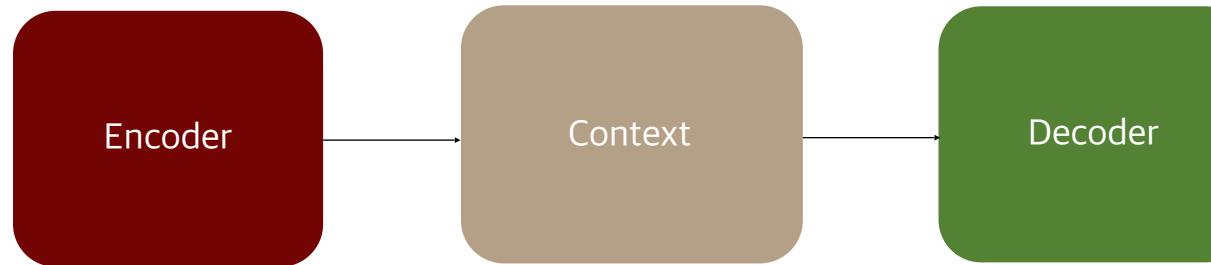
중요한 정보만 포착해가며 (O)

텍스트에서 모든 정보가 중요하지는 않음!
핵심적인 의미를 담은 부분에 특히 집중해서 읽게 됨

→ Attention의 기본 골자

seq2seq

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



입력 문장의 모든 단어를
순차적으로 입력 받음

인코더에서 입력 받은 단어 정보를
압축한 하나의 벡터

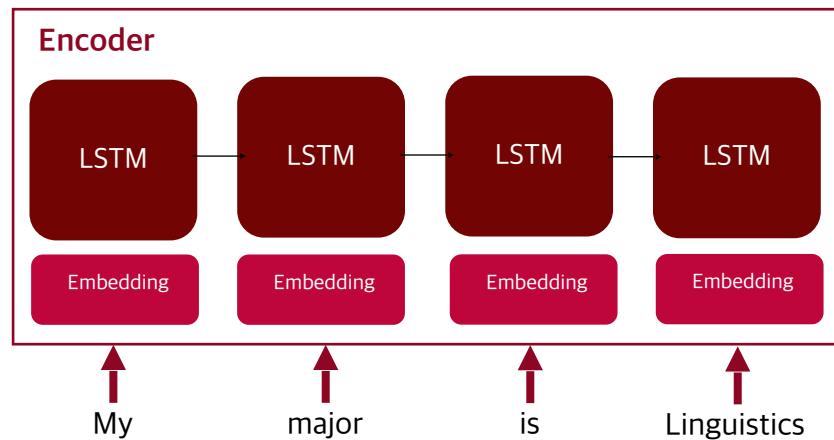
Context Vector를 받아 단어를
한 개씩 순차적으로 출력함



seq2seq

Encoder, Decoder의 내부 구조는 *RNN으로 구성

* vanilla RNN, LSTM, GRU 등

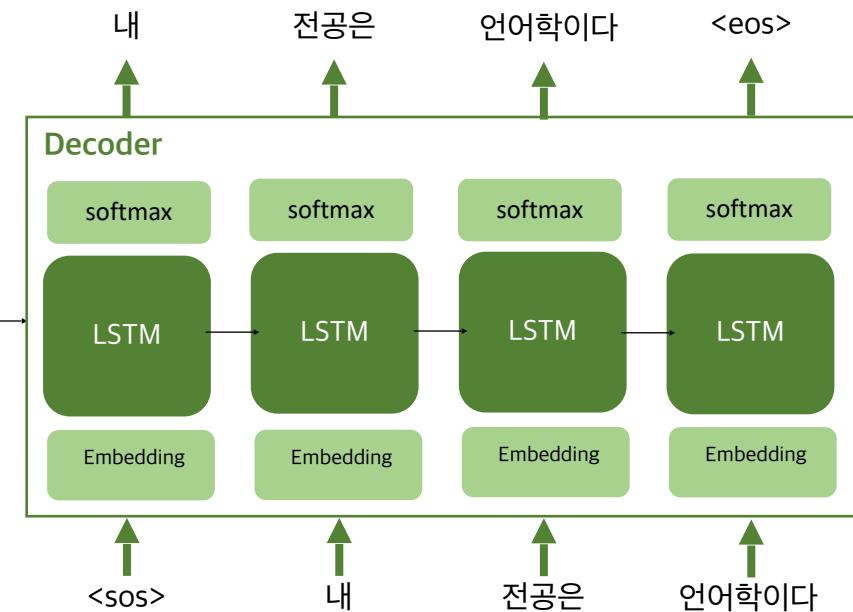


Context

Encoder 가장 마지막 시점의 **hidden state**

=
 "My major is Linguistics"
 에 대한 정보가 담긴
 하나의 벡터

=
 Decoder의 input으로!

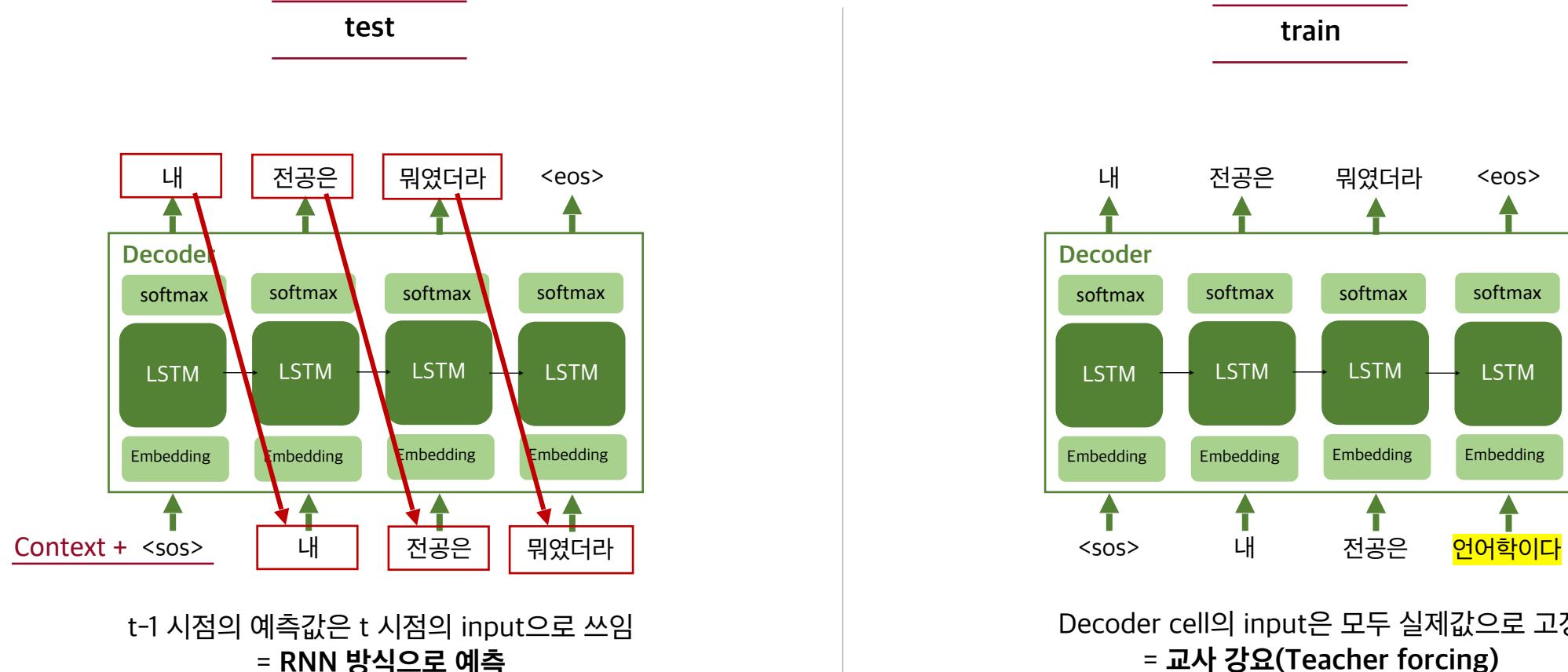


Decoder: 입력 문장을 통해 출력 문장을 예측하는 언어 모델 형식

<sos> : start of string 문장 시작을 알리는 토큰

<eos> : end of string 문장 종료를 알리는 토큰

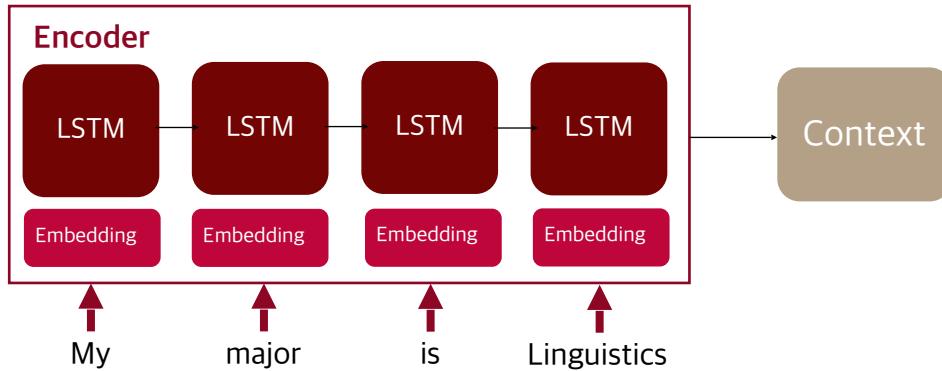
2-1. seq2seq



전 셀을 기반으로 가장 높은 확률이 나올 토큰을 예측하는 형식
→ ground truth랑 다른 결과가 나올 가능성도 높음! 잘못된 prediction :(

Decoder의 각 단계에서 이전 셀의 정답(ground truth)을 Decoder 입력으로 제공
→ Ground Truth만으로 제대로 학습하여 Decoder가 항상 올바른 컨텍스트를 유지하도록

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 비율 (overfitting 방지)
- **n_directions** = 1 (cf. bidirectional RNN의 경우 : n_directions=2)
- **src** = (input seq_len, batch_size)로 구성된 Tensor
 - **src_len** = 입력 seq의 길이 = token들의 길이
 - **batch_size** = 한 epoch 내 별별 처리하는 문장들의 수

```
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)

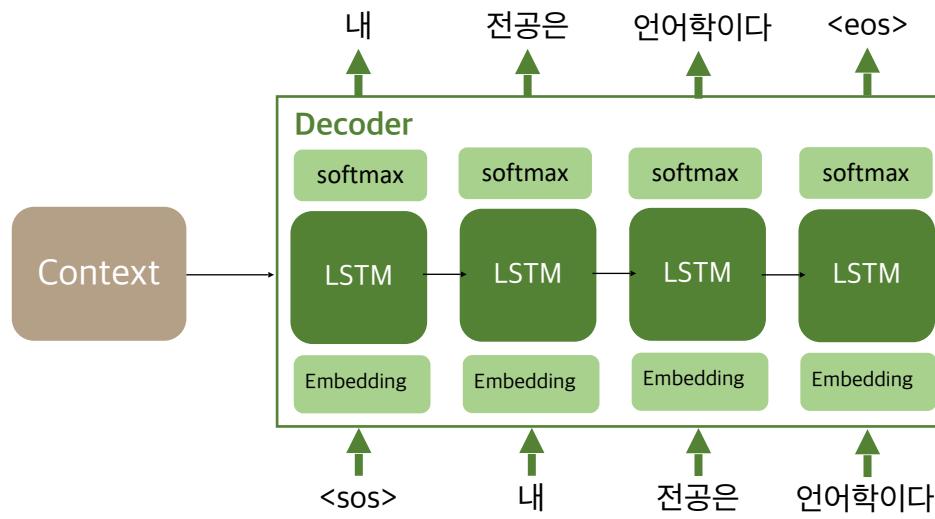
    def forward(self, src):
        # src: [src_len, batch_size]
        embedded = self.dropout(self.embedding(src))

        # 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

- **src = input seq. with size [src_len, batch_size]**

- **trg = target output size with size [target_len, batch_size]**

- **teacher_forcing_ratio**

= decoder가 예측값 대신 실제 목표값을 다음 input으로 사용할 비율

```
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()

        self.encoder = encoder
        self.decoder = decoder
        self.device = device

    def forward(self, src, trg, teacher_forcing_ratio=0.5):
        trg_len = trg.shape[0]
        batch_size = trg.shape[1]
        trg_vocab_size = self.decoder.output_dim

        # decoder 결과를 저장할 텐서
        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size)

        # Encoder의 마지막 은닉 상태가 Decoder의 초기 은닉상태로 쓰임
        hidden, cell = self.encoder(src)

        # Decoder에 들어갈 첫 input은 <sos> 토큰
        input = trg[0, :]

        # range(1,trg_len)인 이유 : 0번째 trg는 항상 <sos>라서 그에 대한 output도 항상 0
        for t in range(1, trg_len):
            output, hidden, cell = self.decoder(input, hidden, cell)
            outputs[t] = output

            # 랜덤 숫자가 teacher_forcing_ratio보다 작으면 True니까 teacher_force=1
            teacher_force = random.random() < teacher_forcing_ratio

            # 확률 가장 높게 예측한 토큰
            top1 = output.argmax(1)

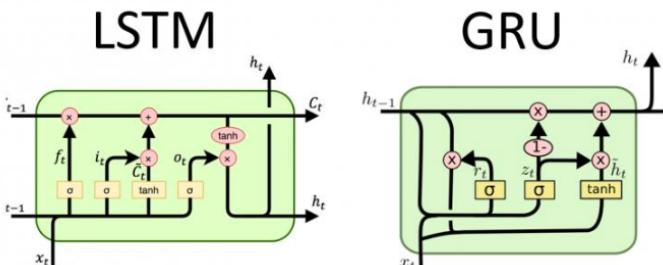
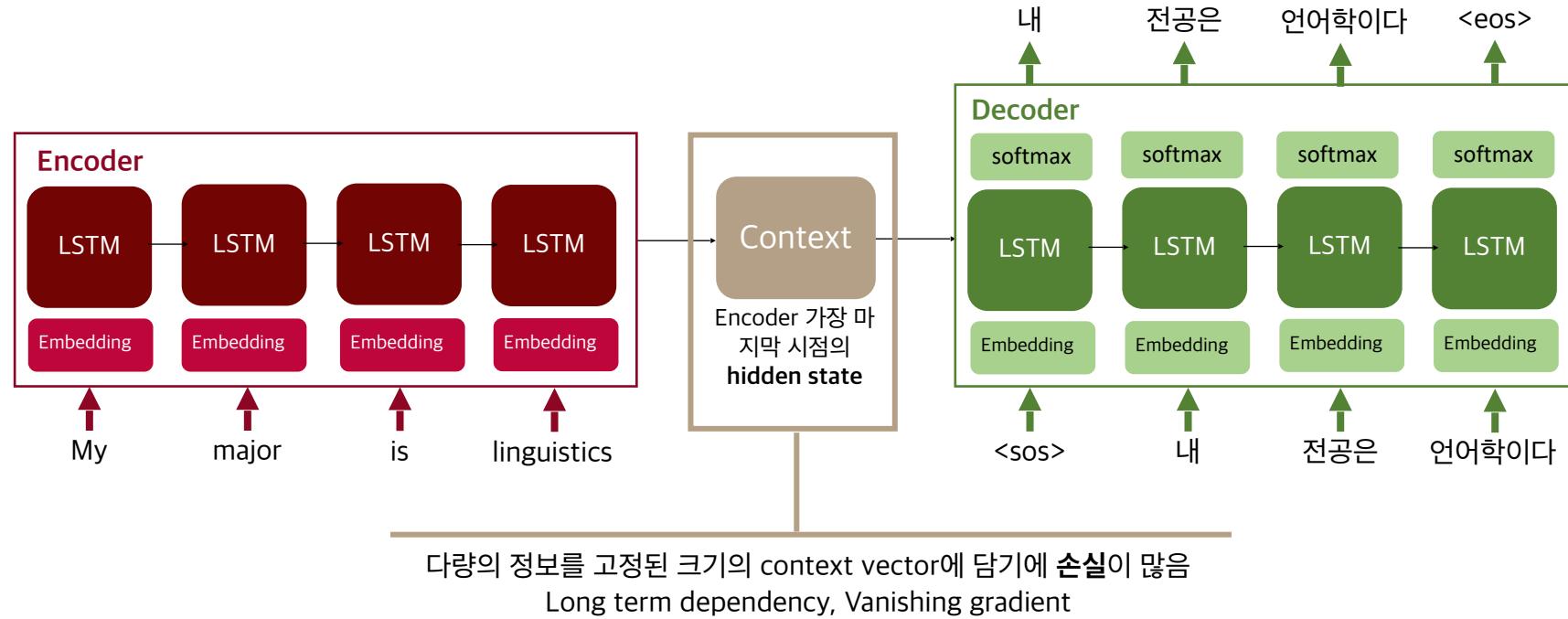
            # teacher_force = 1 = True이면 trg[t]를 아니면 top1을 input으로 사용
            input = trg[t] if teacher_force else top1

return outputs
```

03 Attention

특정 단어에 “집중” 하게 하려면?

3-1. Limitation of seq2seq



LSTM, GRU가 대안이기는 하나,
여전히 tanh, sigmoid 같은 nonlinear 함수가 지닌 계산 복잡성이 존재

3-2. Attention

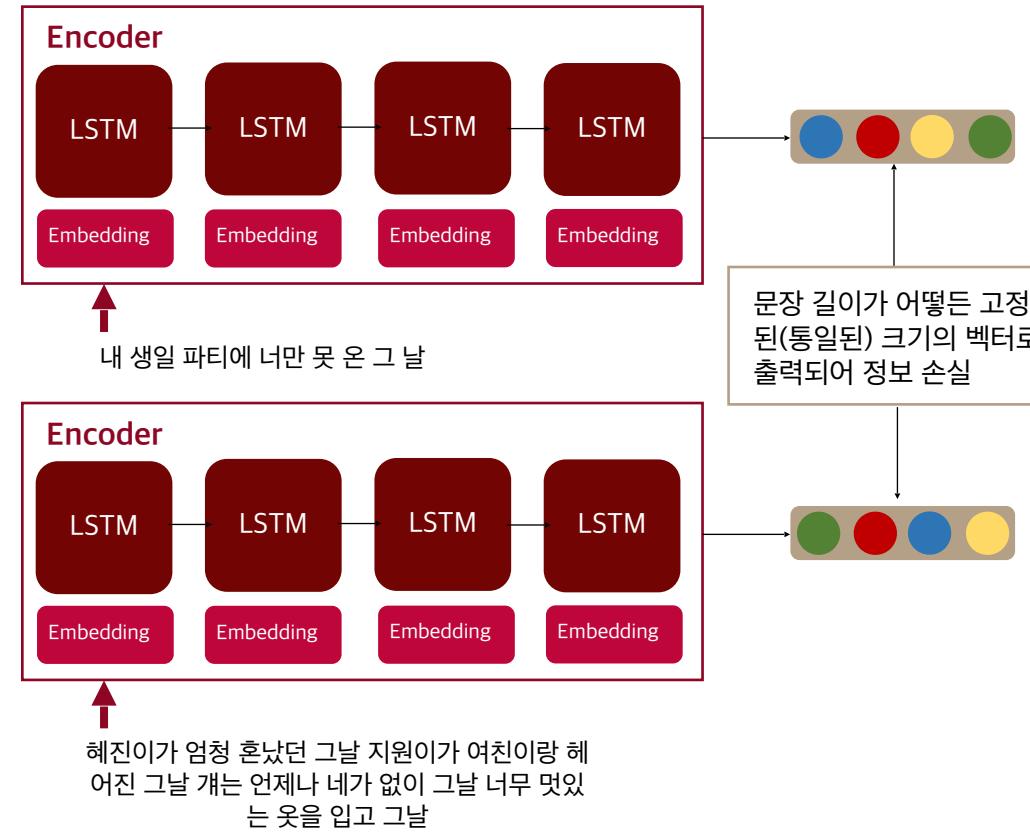
Sol) Attention:

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

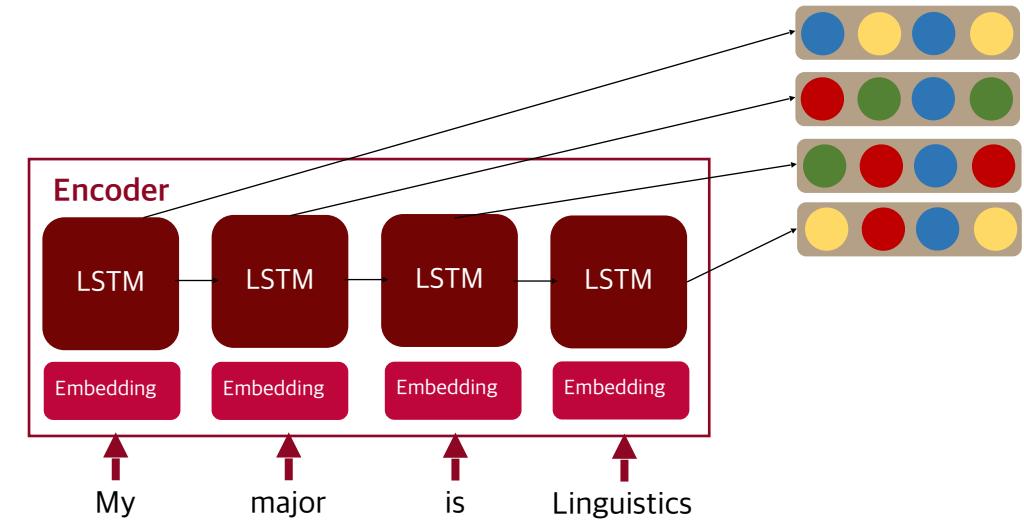
Decoder의 매 시점마다 그 시점에서 예측해야 할 단어와 연관이 있는 encoder의 hidden state만 집중(attention)해서 참고함



seq2seq



Attention



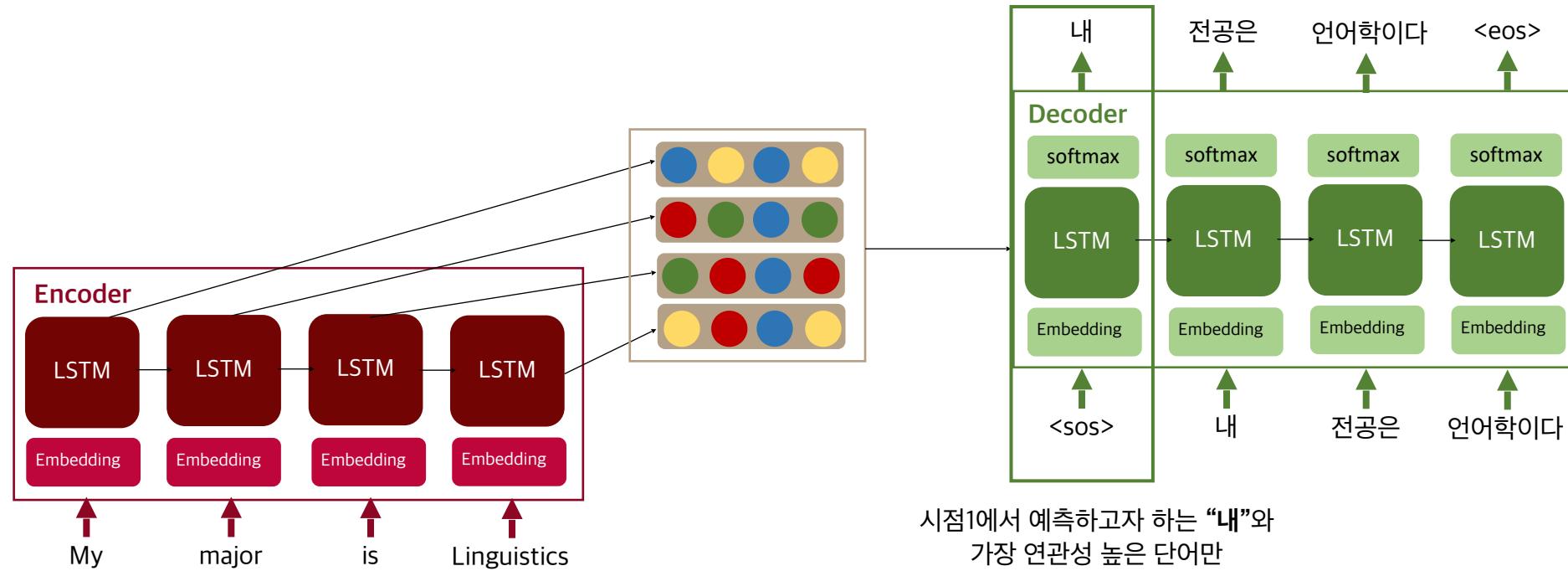
Encoder의 최종 hidden state만 참고하는 것이 아니라
매 time step의 hidden state를 참고함
참고) biLSTM이 더욱 효과적으로 쓰이나, 편의상 도식화에선 단방향으로



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



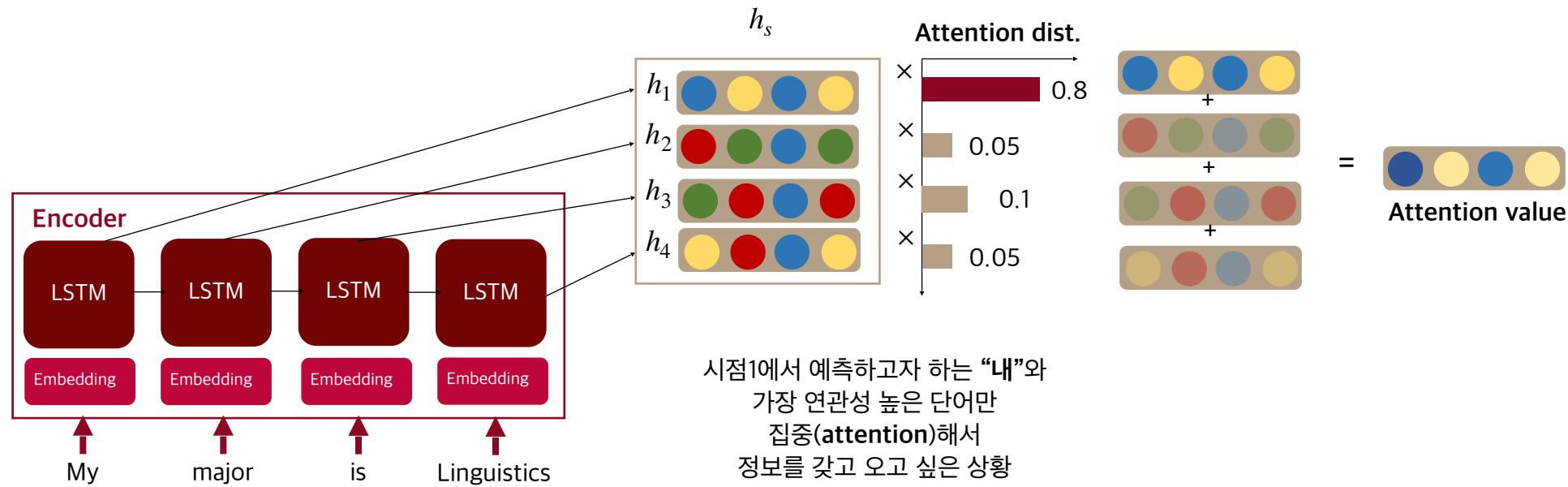
시점1에서 예측하고자 하는 “내”와
가장 연관성 높은 단어만
집중(attention)해서
정보를 갖고 오고 싶은 상황

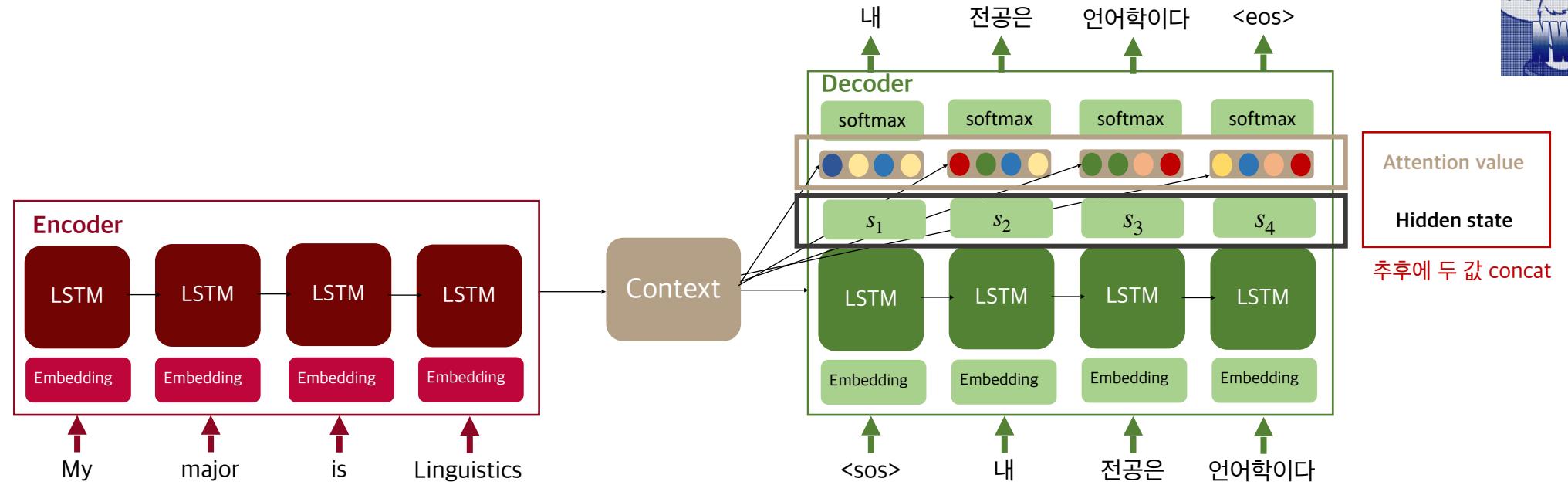
3-2. Attention

Sol) Attention:

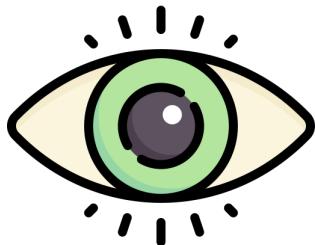
Encoder의 최종 hidden state만 참고하는 것이 아니라 매 time step의 hidden state를 참고함

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



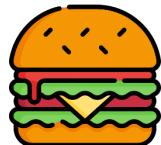


Decoder의 특정 시점에 input으로 들어가는 것들
 - hidden state
 - Attention value
 - Embedding vector



(내가 궁금해 하는 것)

q
query



(주변에 있는 모든 정보)

list of keys



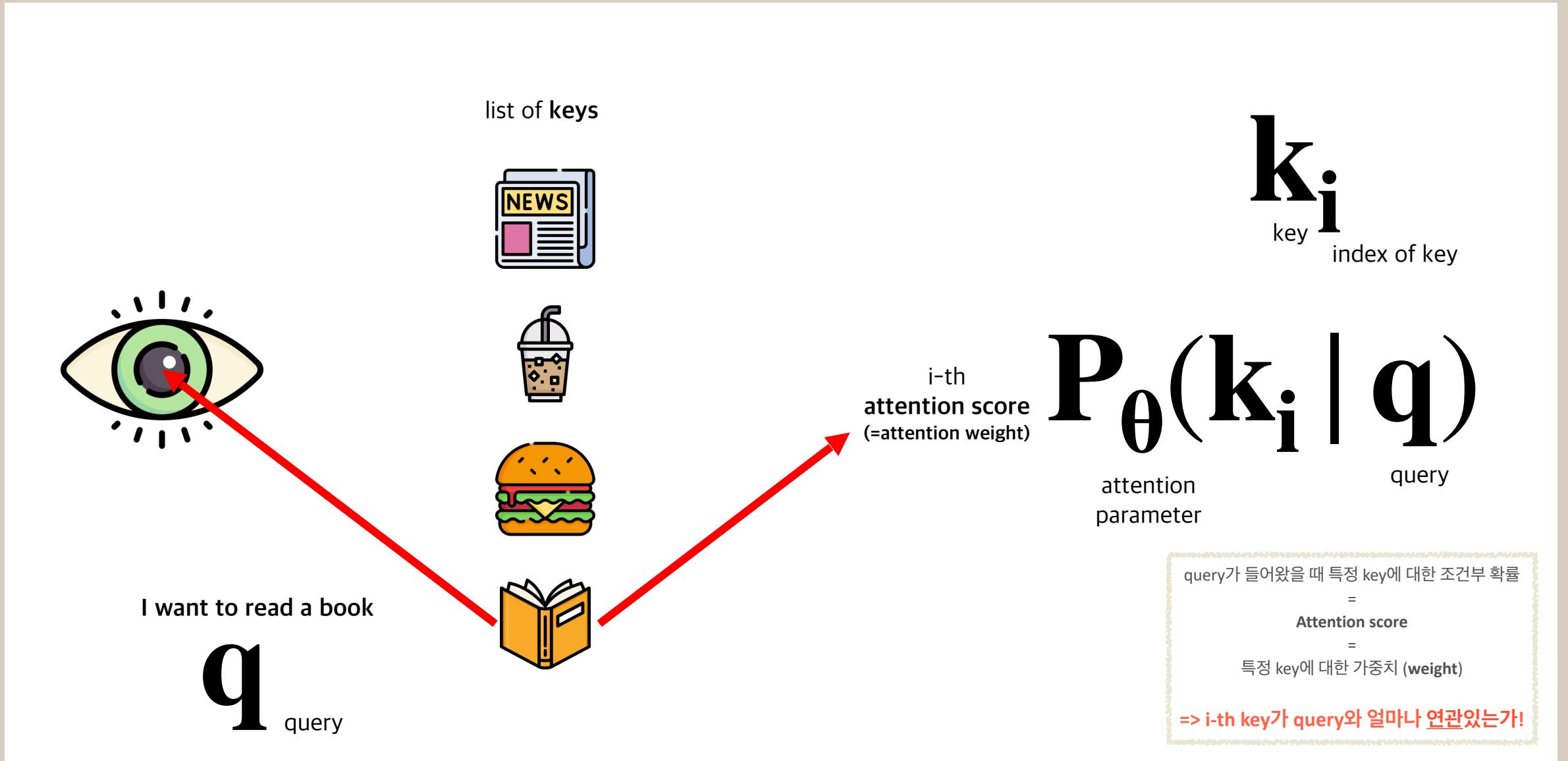
k_i

key

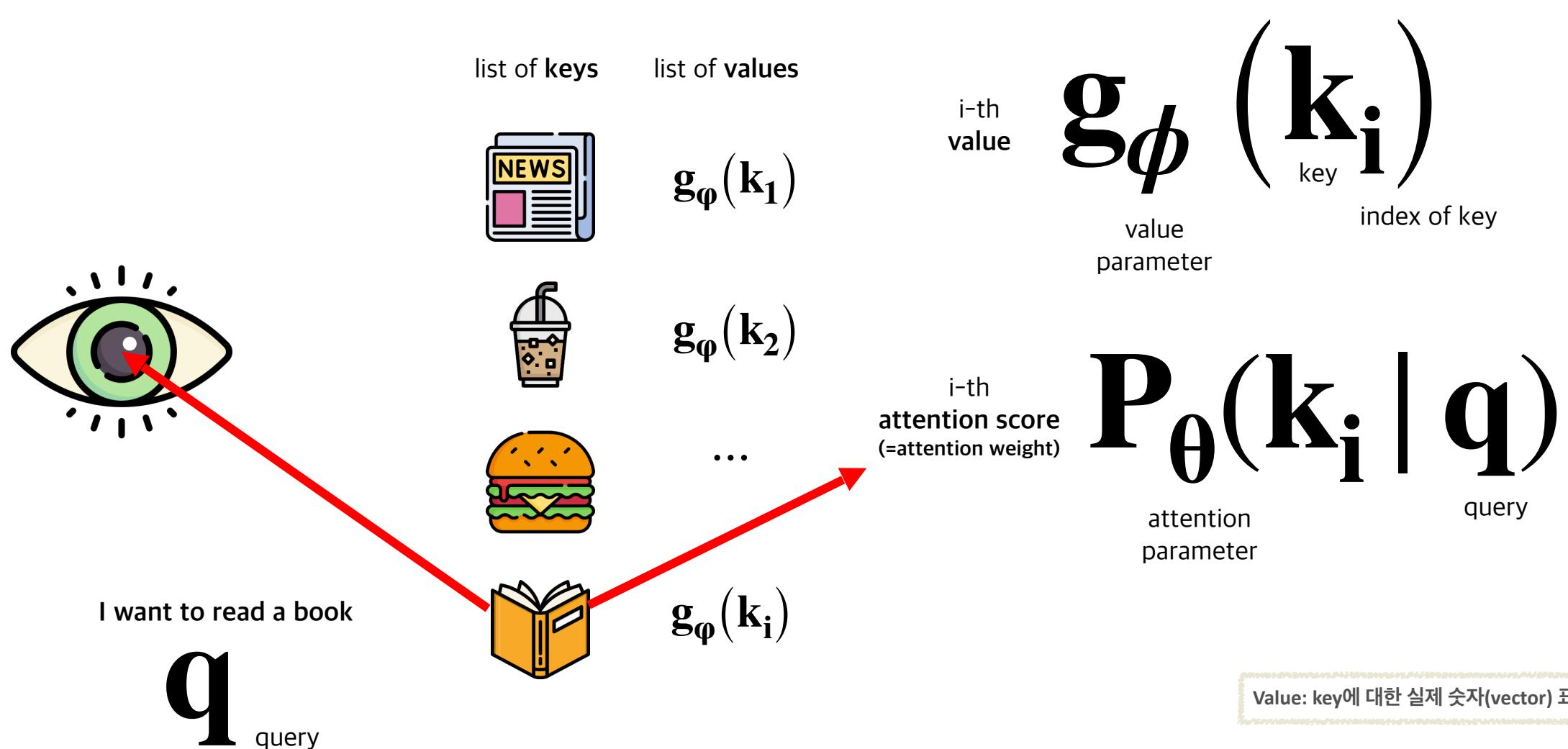
Index of key

Attention의 목표
=

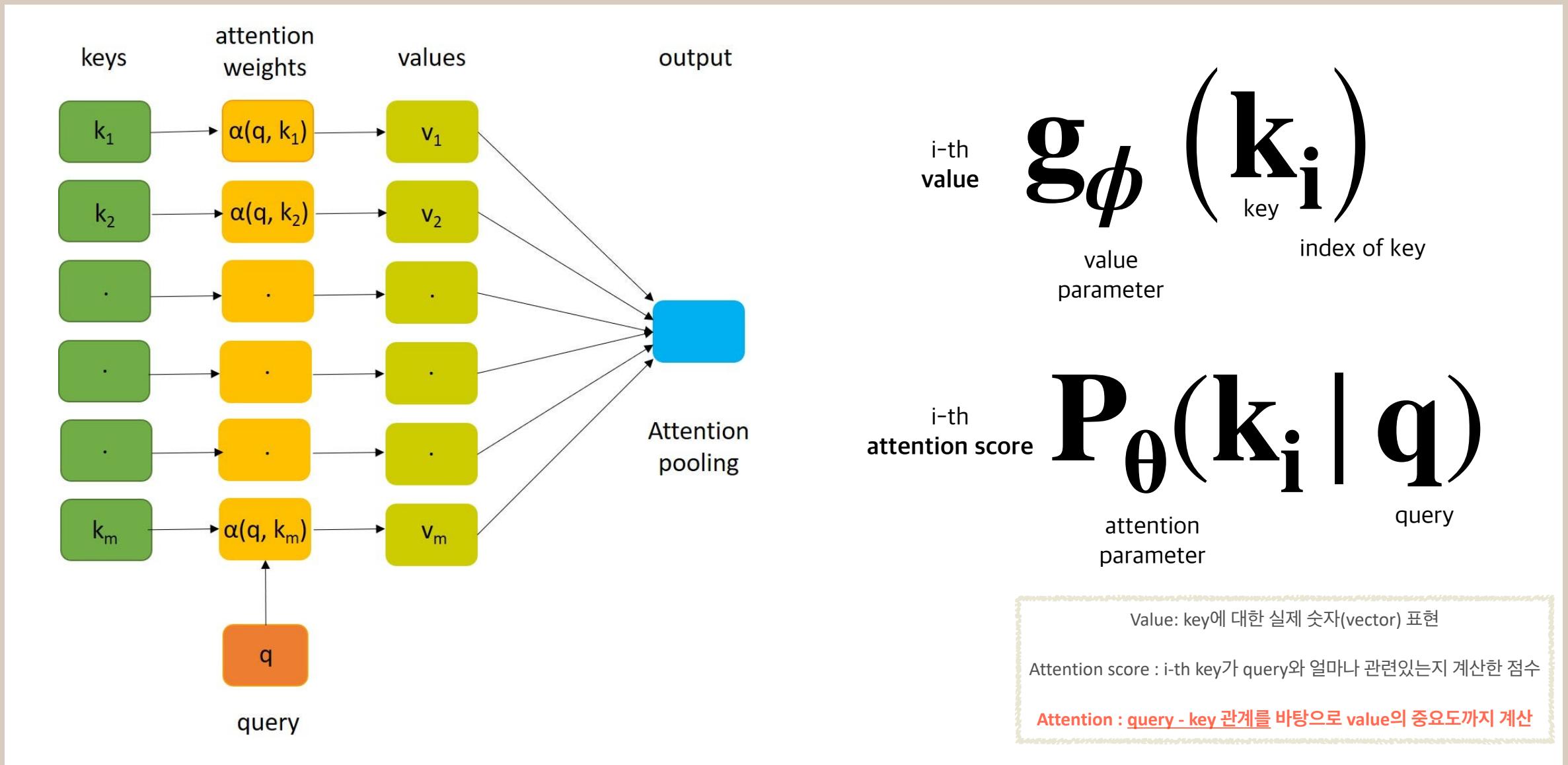
Query와 가장 관련이 있는 Key를 찾는 것!



3-3. Key, Query, Values



3-3. Key, Query, Values



3-3. Key, Query, Values

💡 그 유사도는 어떻게 구하는건데!

Attention score를 계산하는 방법

$$a_\theta(\mathbf{q}, \mathbf{k}) = \mathbf{q} \cdot \mathbf{k}$$

Dot Attention Score

$$a_\theta(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q} \cdot \mathbf{k}}{\sqrt{d_k}}$$

Scaled Dot Attention Score

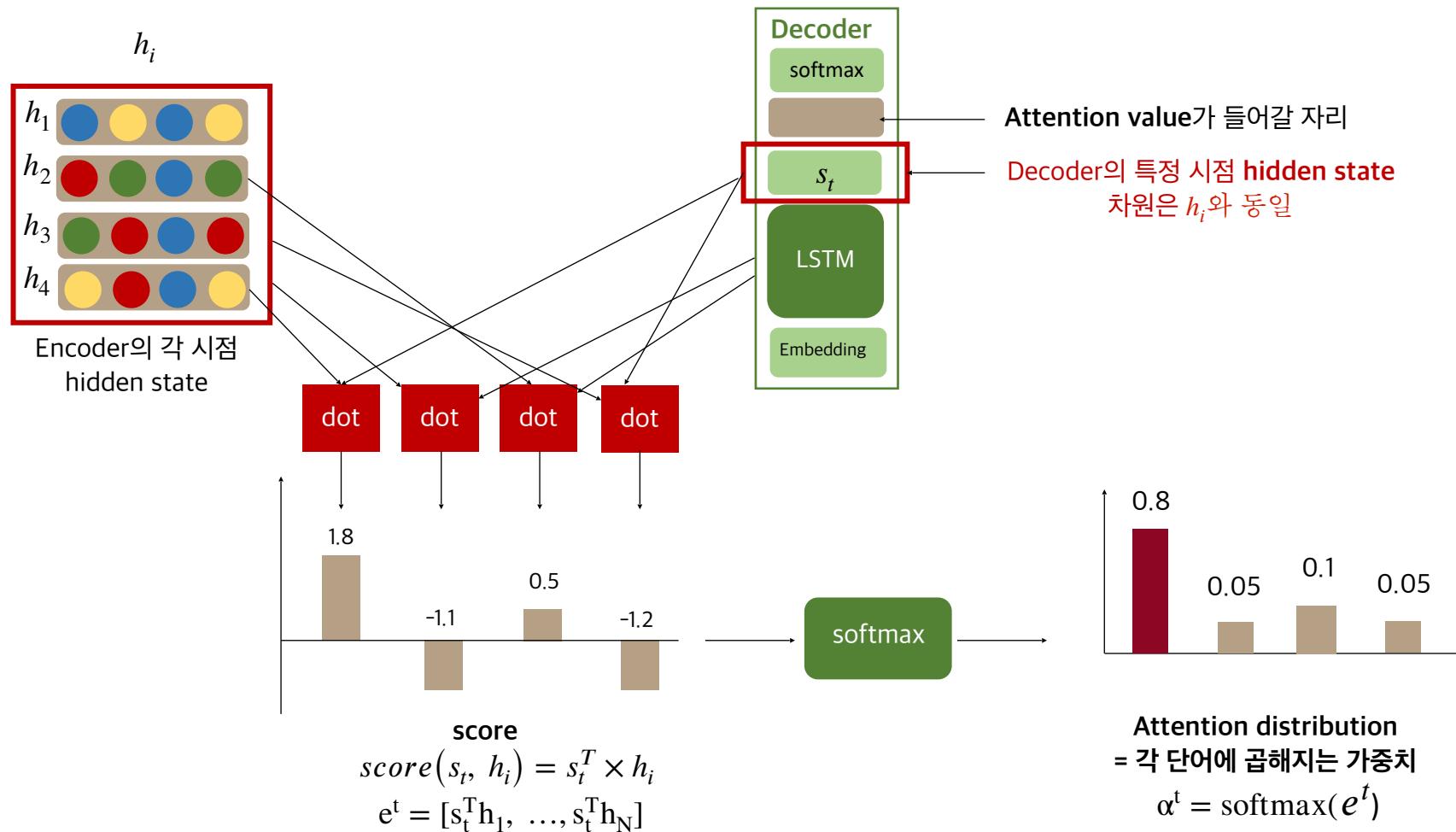
3-3. Key, Query, Values

Attention: Query와 Key의 유사도를 바탕으로 각 Value의 중요도를 계산하고 이를 종합하여 최종 결과를 도출하는 메커니즘

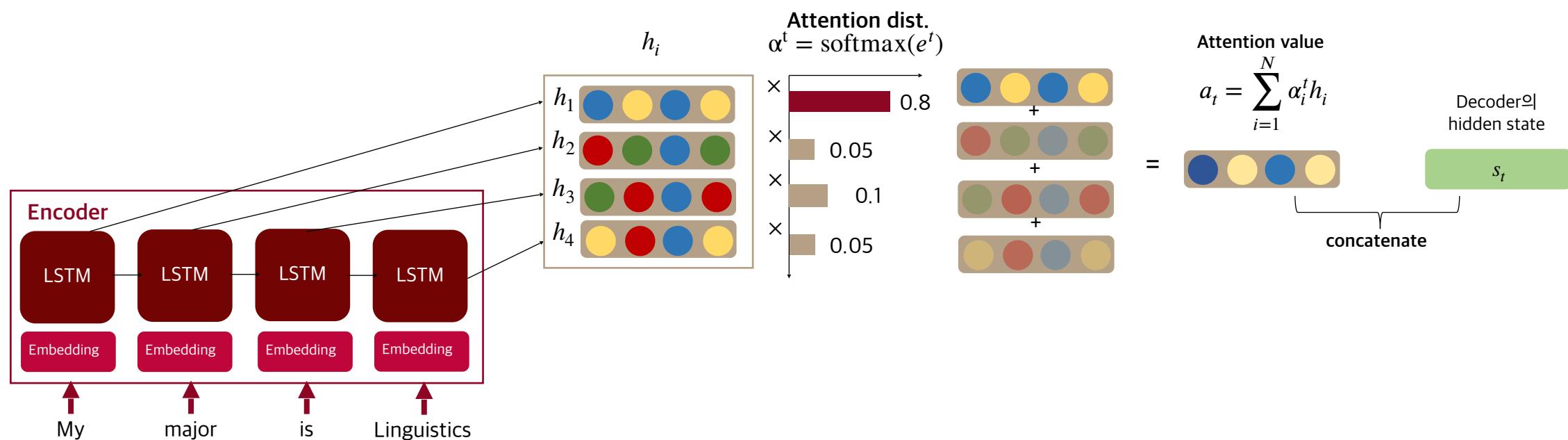
$$\begin{aligned} A_{\theta, \phi}(q, \{k_i\}) &= E_{k \sim P_\theta(\cdot | q)}[g_\phi(k)] \\ &= \sum_i P_\theta(k_i | q) g_\phi(k_i) \\ &\stackrel{\text{Softmax}}{=} \sum_i \frac{\exp(a_\theta(q, k_i))}{\sum_j \exp(a_\theta(q, k_j))} g_\phi(k_i) \end{aligned}$$

3-4. Attention score, distribution, value

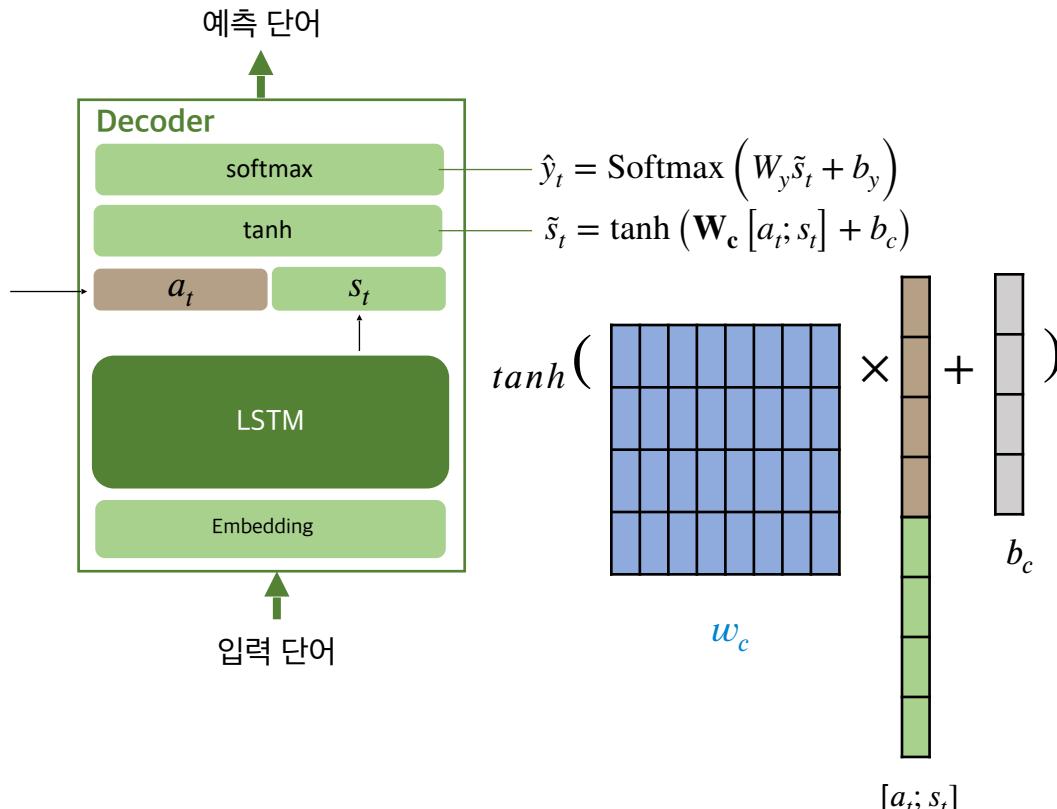
각 단어에 곱하는 가중치(attention distribution)는 어떻게 구하는가?



3-4. Attention score, distribution, value



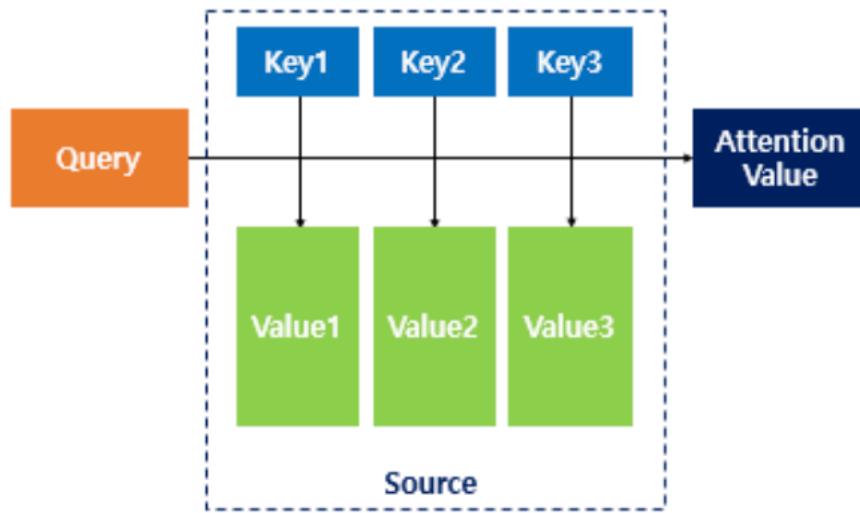
3-4. Attention score, distribution, value



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

이름	스코어 함수	Defined by
<i>dot</i>	$\text{score}(s_t, h_i) = s_t^T h_i$	<i>Luong et al. (2015)</i>
<i>scaled dot</i>	$\text{score}(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{n}}$	<i>Vaswani et al. (2017)</i>
<i>general</i>	$\text{score}(s_t, h_i) = s_t^T W_a h_i$	<i>Luong et al. (2015)</i>
<i>concat</i>	$\text{score}(s_t, h_i) = W_a^T \tanh(W_b [s_t; h_i]), \text{score}(s_t, h_i) = W_a^T \tanh(W_b s_t + W_c h_i)$	<i>Bahdanau et al. (2015)</i>
<i>location-base</i>	$\alpha_t = \text{softmax}(W_a s_t)$	<i>Luong et al. (2015)</i>

3-4. Attention score, distribution, value

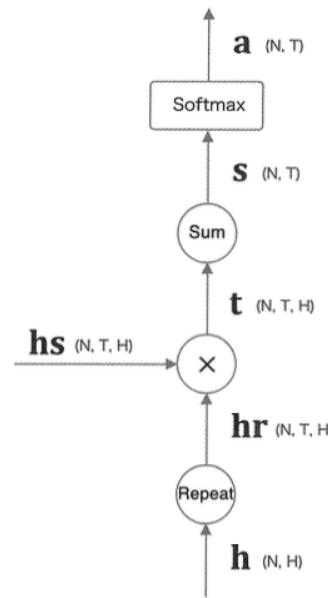


$$\text{Attention}(Q, K, V) = \text{Attention Value}$$

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

사진: 딥러닝을 이용한 자연어 처리 입문

3-5. Attention Code



a (5,)

0.8	0.1	0.03	0.05	0.02
-----	-----	------	------	------

reshape →

ar (5, 1)

0.8
0.1
0.03
0.05
0.02

hs

0.1	1.3	-0.4	1.2
-0.3	-0.4	-0.3	-0.4
-1.2	0.9	-1.7	0.2
-0.7	-0.8	0.3	-0.2
0.6	2.1	1.0	-0.4

ar

0.8
0.1
0.03
0.05
0.02

*

```

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
  
```

```

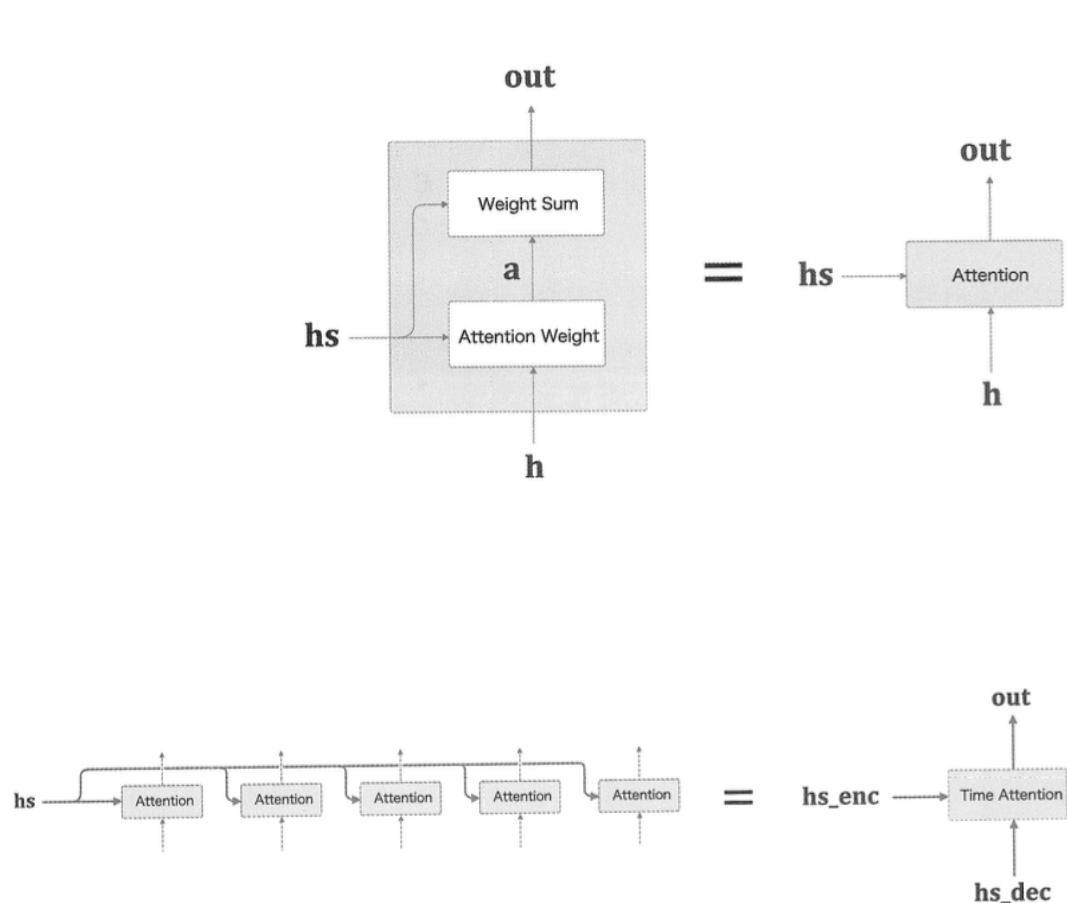
class WeightSum:
    def __init__(self):
        self.params, self.grads = [], []
        self.cache = None

    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



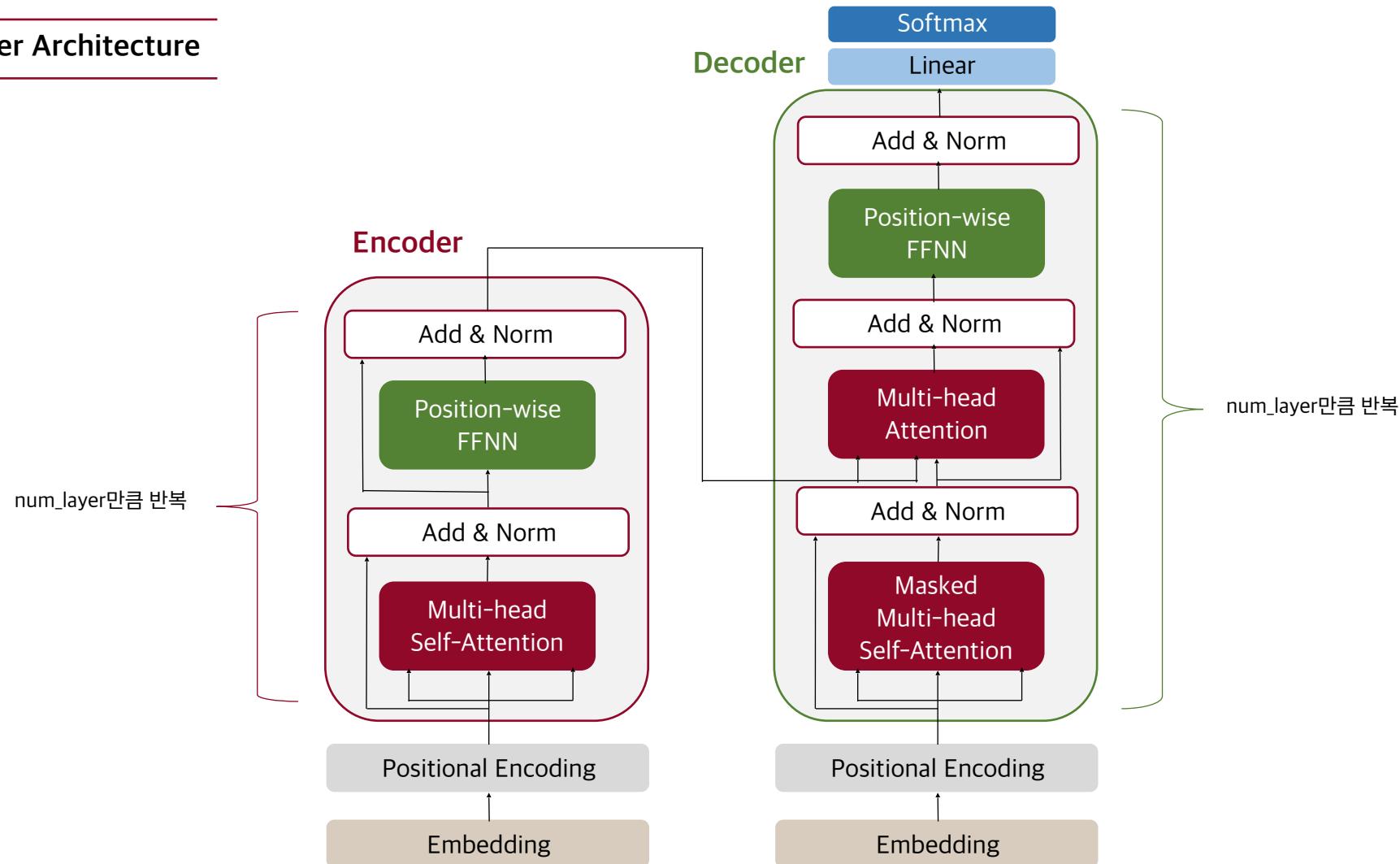
```
class Attention:  
    def __init__(self):  
        self.params, self.grads = [], []  
        self.attention_weight_layer = AttentionWeight()  
        self.weight_sum_layer = WeightSum()  
        self.attention_weight = None  
  
    def forward(self, hs, h):  
        a = self.attention_weight_layer.forward(hs, 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

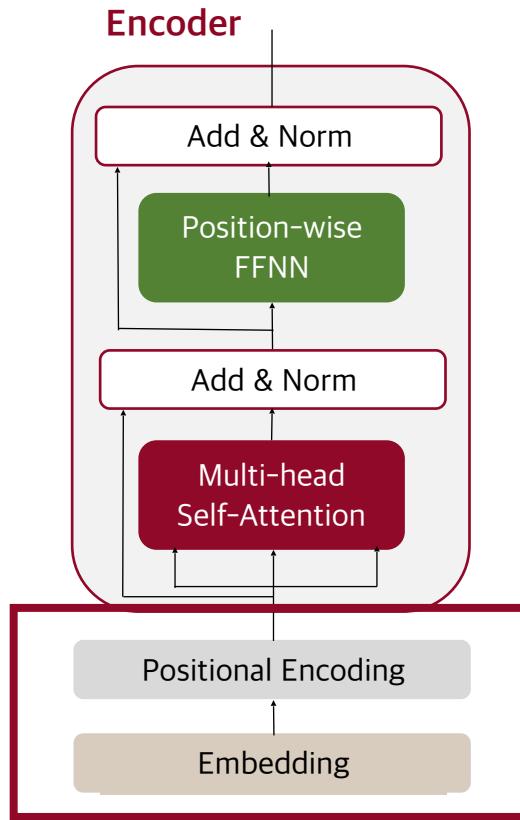
Attention만으로 encoder와 decoder 만들기

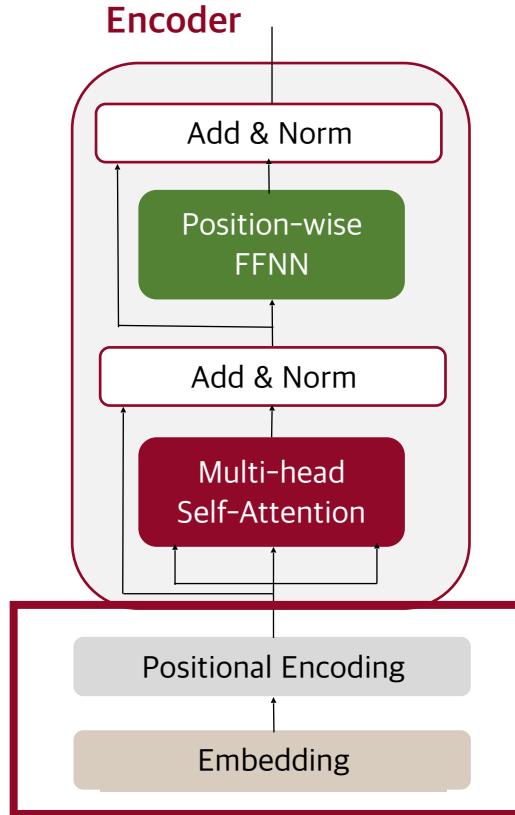
4-1. Transformer Architecture

Transformer Architecture



4-2. Embedding Layer





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



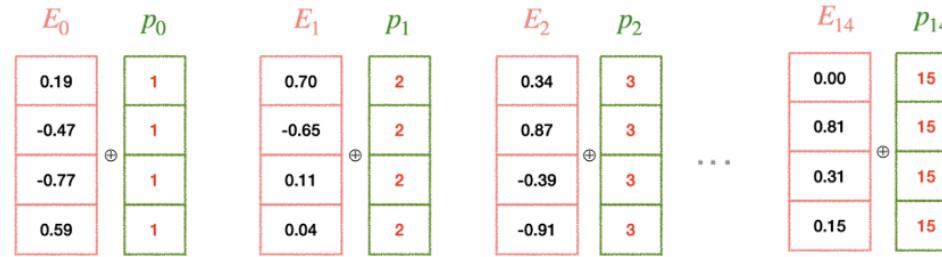
참고) Concatenate가 아닌 sum 연산을 하는 이유

- 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



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

- 같은 위치더라도 시퀀스 길이에 따라 다른 벡터값이 부여됨

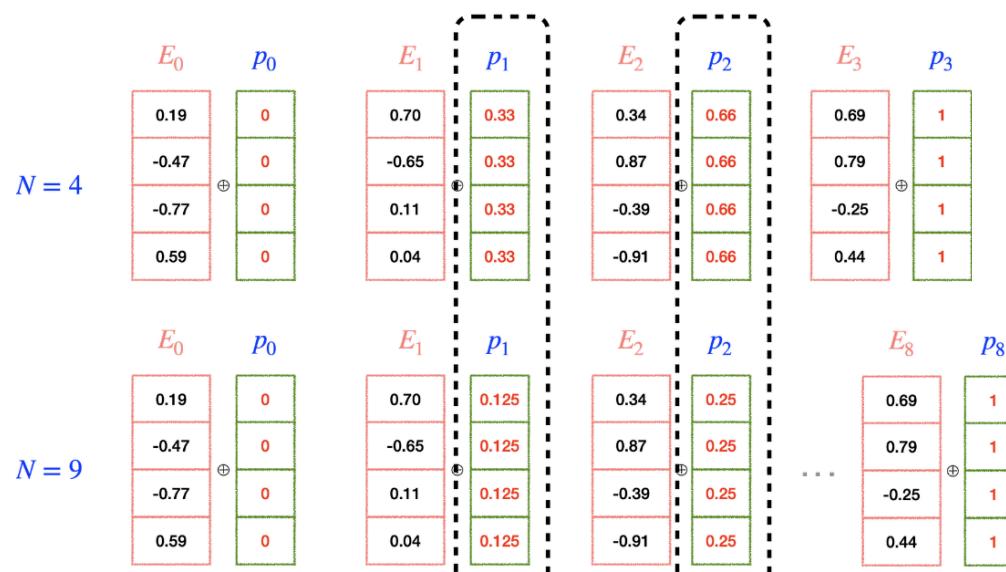
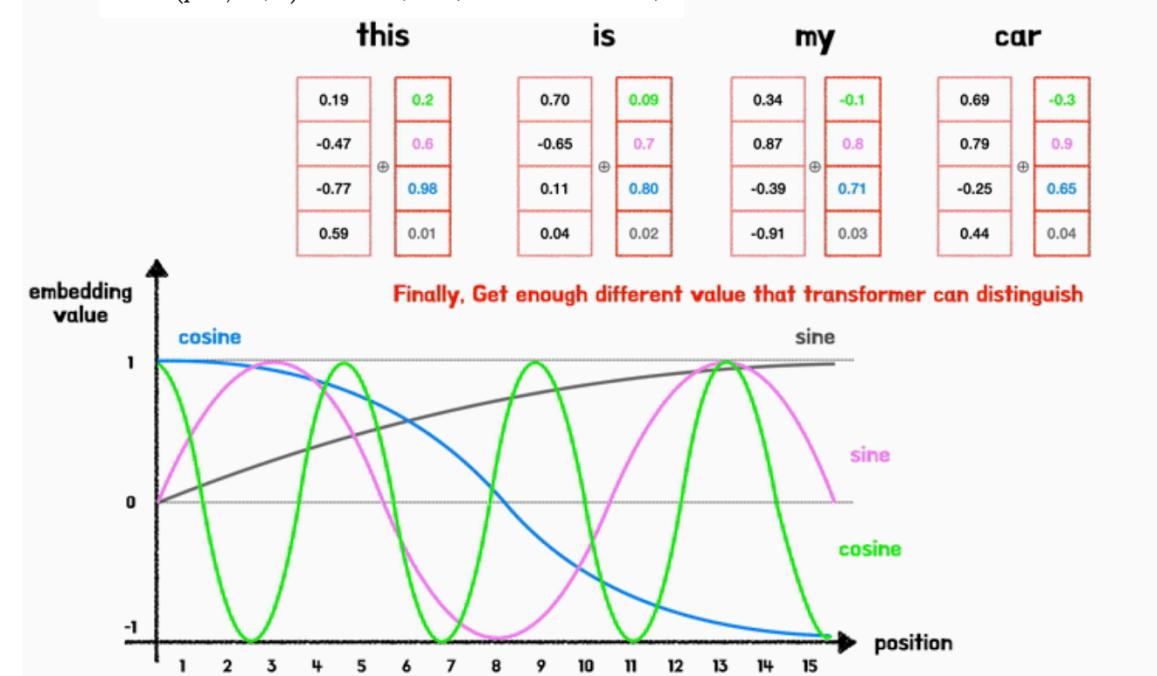


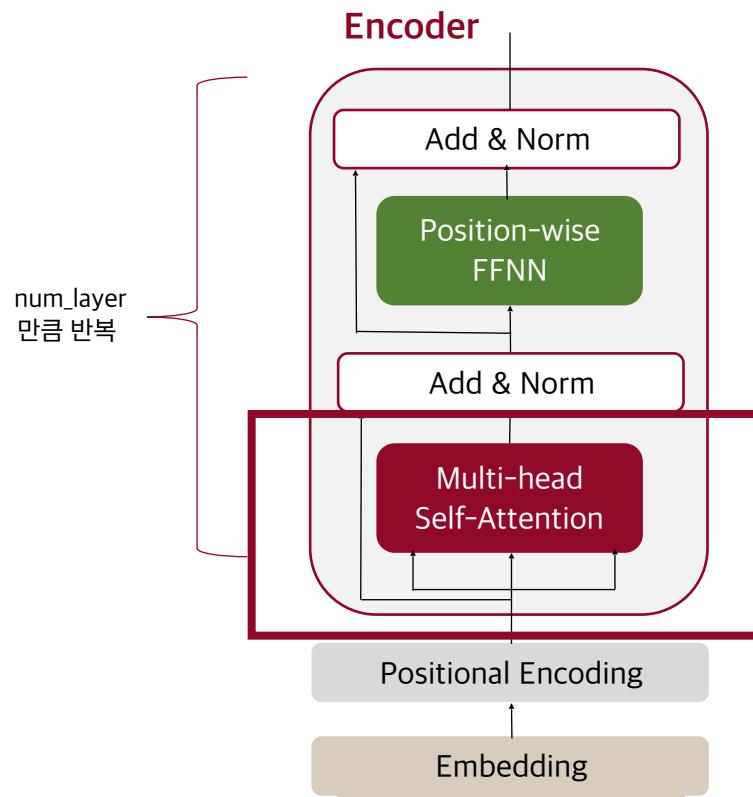
사진: <https://www.blossominkyung.com/deeplearning/transformer-positional-encoding>

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

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

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





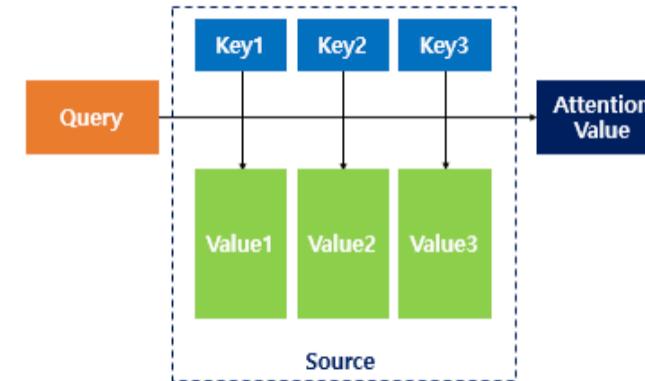
Multi-head Self-Attention

What is Self-Attention?

Encoder의 token들

Shim loves Kendrick and Drake too

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

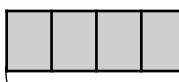


Self Attention의 경우, Q,K,V의 출처가 모두 같음!
출처가 같을 뿐, 아예 같은 값이란 의미는 아님.

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

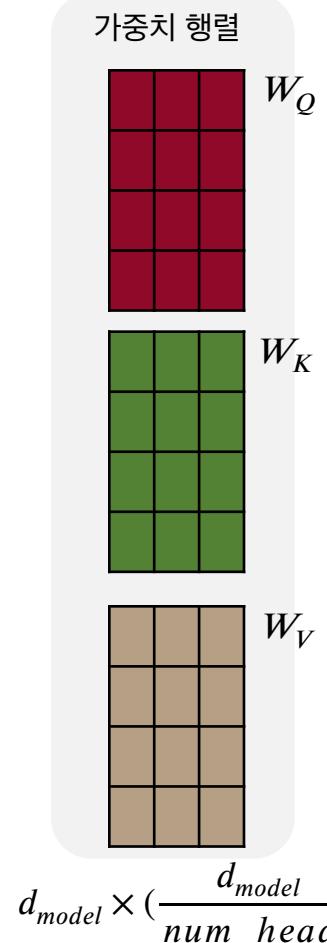
4-2. Encoder

Self-Attention

seunghyun 
loves 
newjeans 

논문에선
 $d_{model}=512$

위 예시에선
 $d_{model}=4$



Query

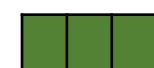
seunghyun 
 $Q_{seunghyun}$

loves 
 Q_{loves}

newjeans 
 $Q_{newjeans}$

Key

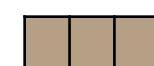
seunghyun 
 $K_{seunghyun}$

loves 
 K_{loves}

newjeans 
 $K_{newjeans}$

Value

seunghyun 
 $V_{seunghyun}$

loves 
 V_{loves}

newjeans 
 $V_{newjeans}$

$$d_q = \left(\frac{d_{model}}{num_heads} \right)$$

$$d_k = \left(\frac{d_{model}}{num_heads} \right)$$

$$d_v = \left(\frac{d_{model}}{num_heads} \right)$$

Self-Attention

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



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



$$\sqrt{d_k}$$

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



$$\begin{matrix} \text{seunghyun really loves newjeans} \\ \text{seunghyun} \\ \text{really} \\ \text{loves} \\ \text{newjeans} \end{matrix} \quad \text{Attention Score Matrix}$$



softmax

$$\text{softmax} \left(\frac{\text{---}}{\sqrt{d_k}} \right)$$

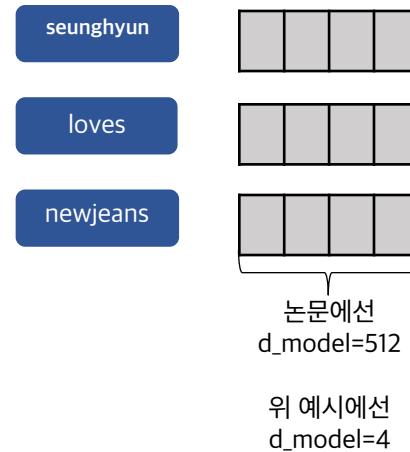
$$\begin{matrix} Q \\ \times \\ k^T \end{matrix}$$

$$\times$$

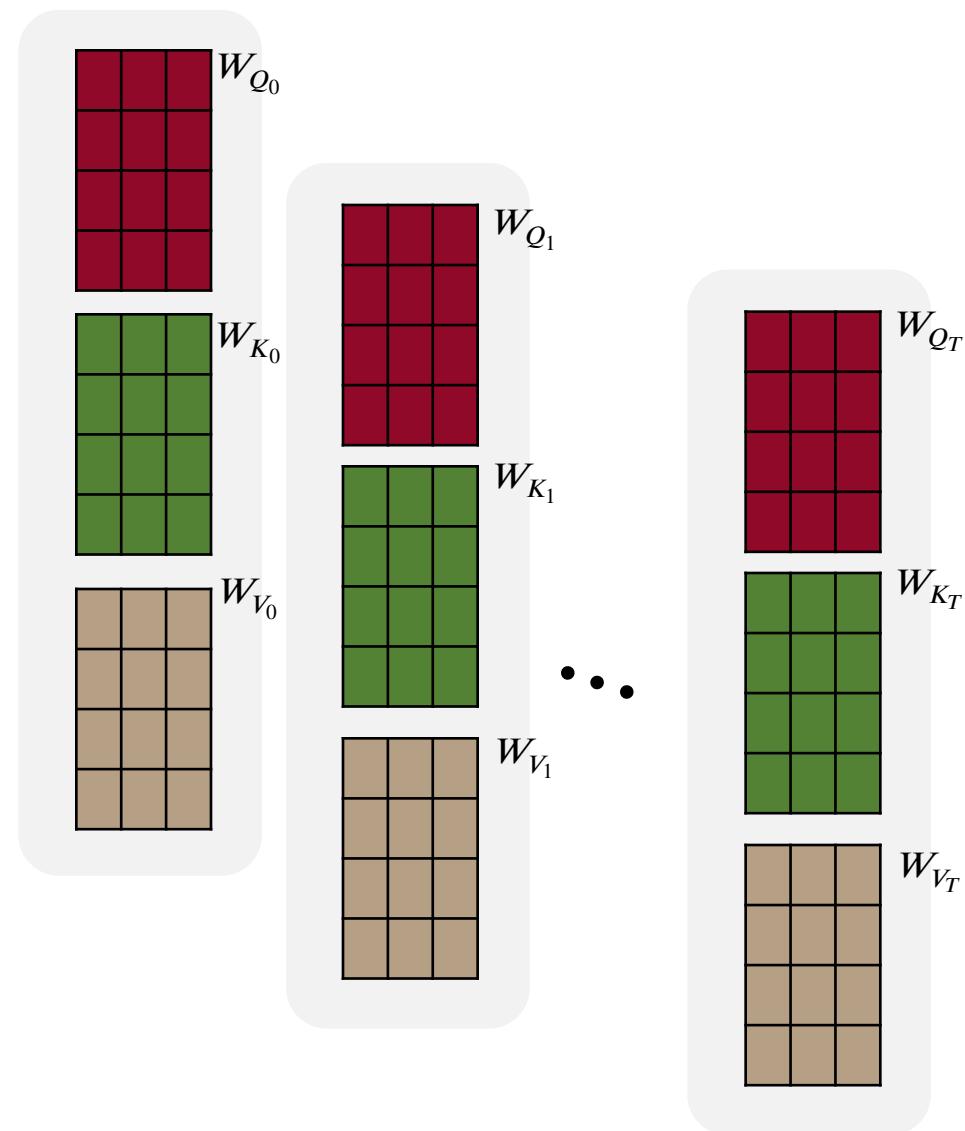
$$V$$

$$= \begin{matrix} \text{Attention Value Matrix} \\ a \\ \text{seq_len} \times d_V \end{matrix}$$

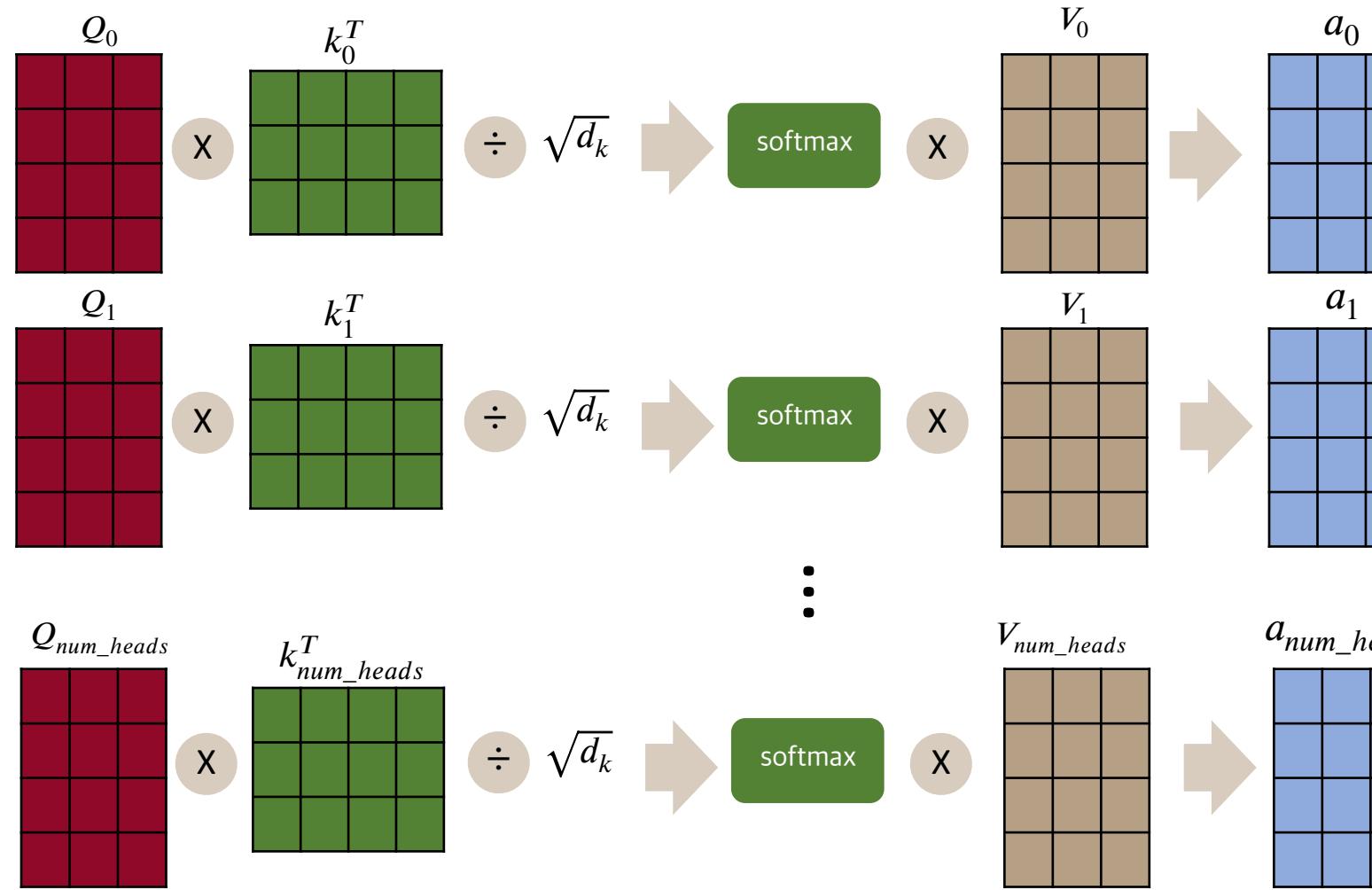
Multi-head Attention



X

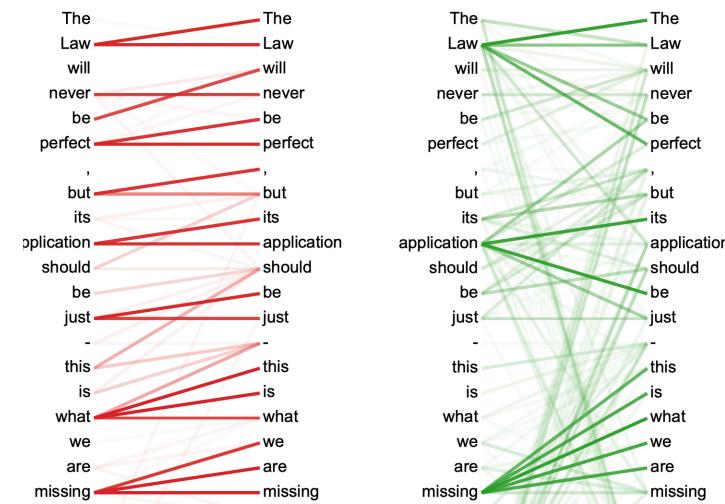
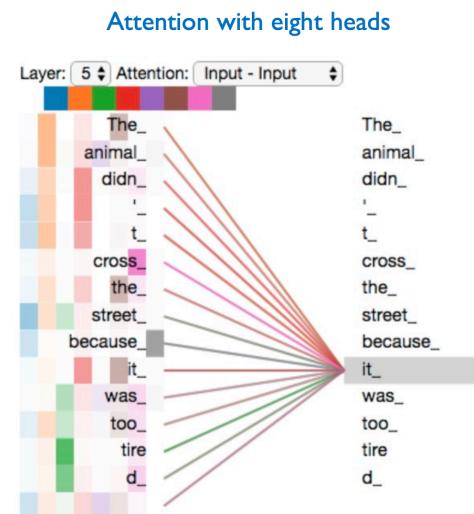
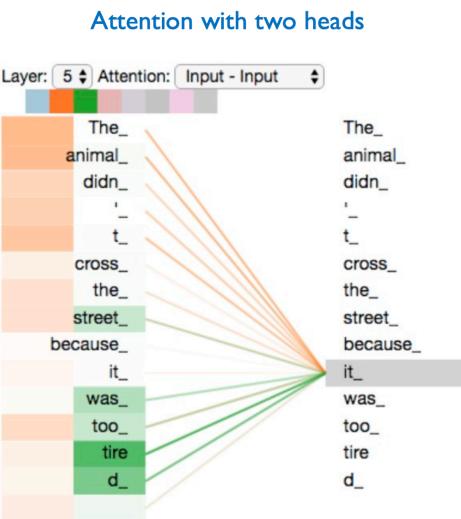
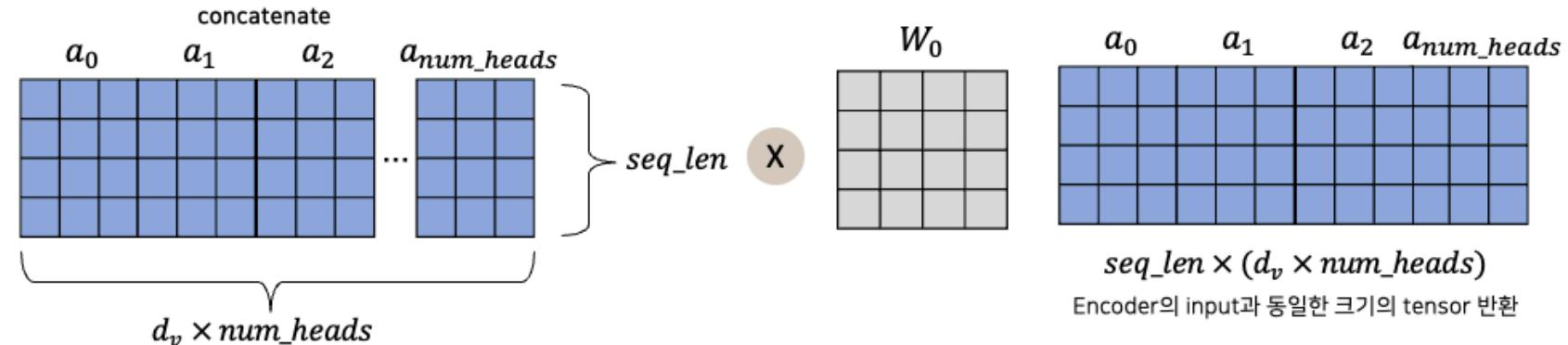


Multi-head Attention



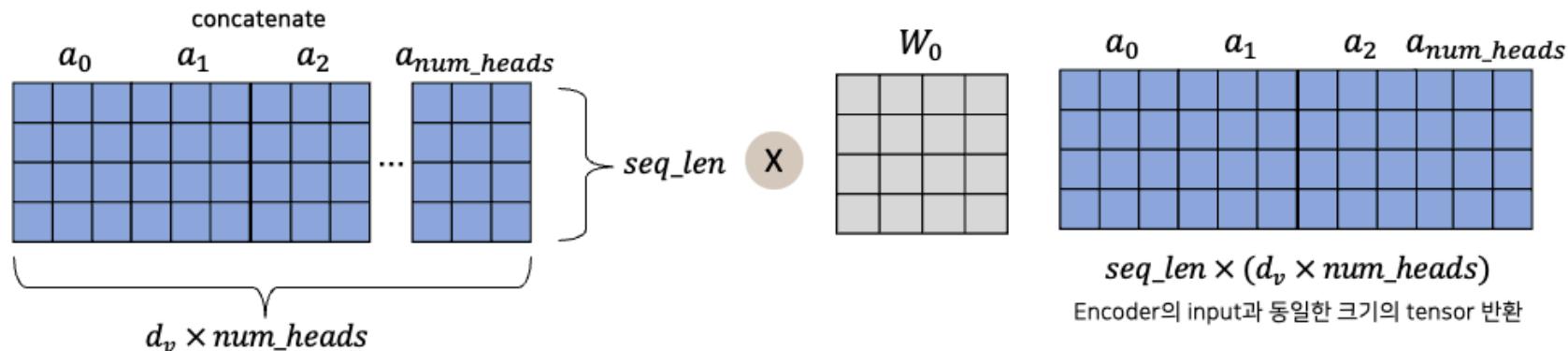
num_heads 횟수만큼
Self-attention을
병렬로 연산
num_heads개의
Attention head 생산

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

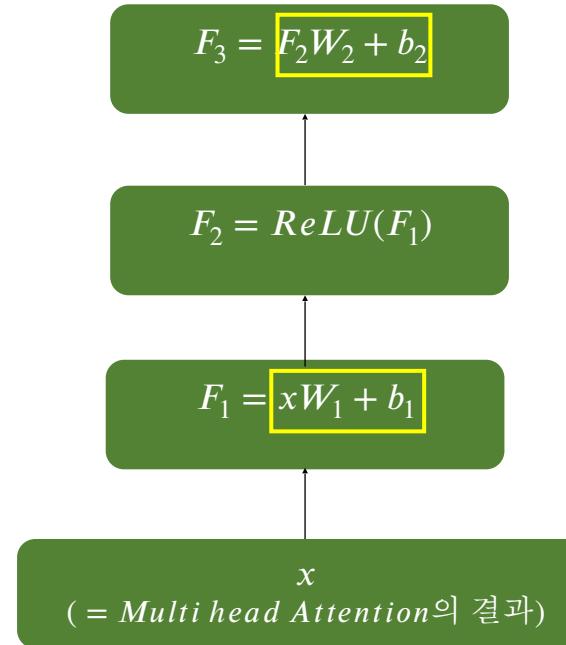
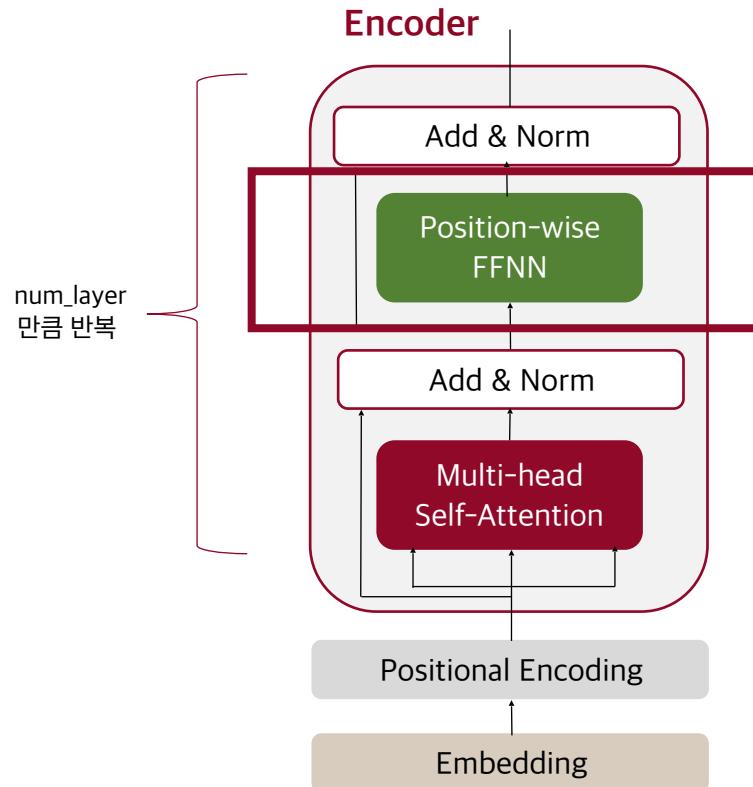
Which do you like better, coffee or tea?

- 관계에 집중하는 어텐션

Which do you like better, coffee or tea?

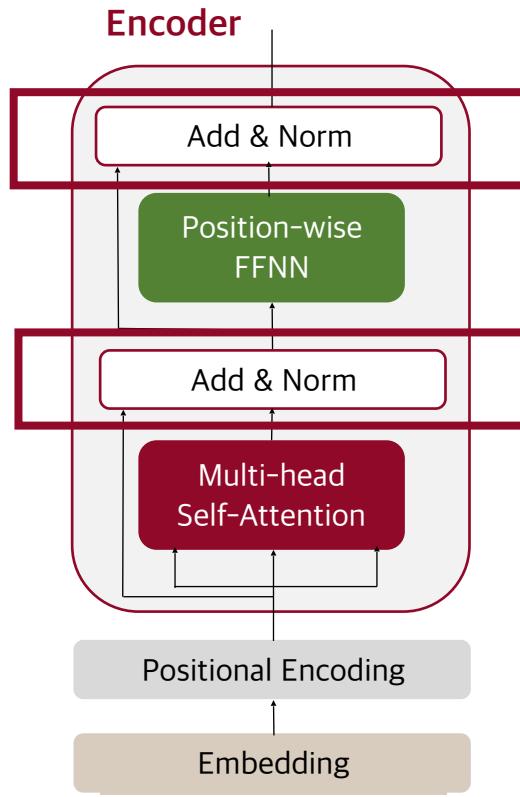
- 강조에 집중하는 어텐션

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

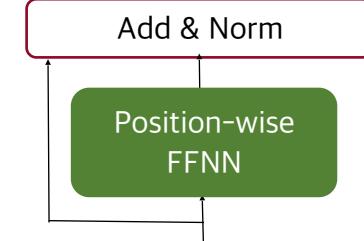
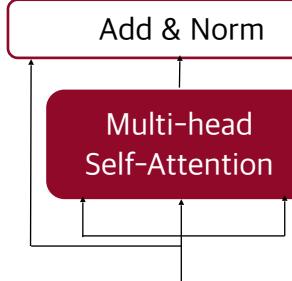


- 여러 개의 attention-head는 각각의 관점에 따라 정보가 편향되어 있음. 이것들을 균등하게 섞는 역할 수행
- 각 가중치들은 하나의 인코더 내에서는 모든 문장에서 동일하게 share
- 다른 인코더 층끼리는 다른 가중치를 share

4-2. Encoder



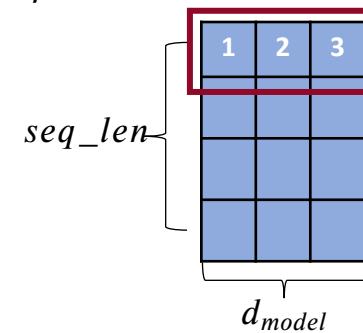
Residual Connection



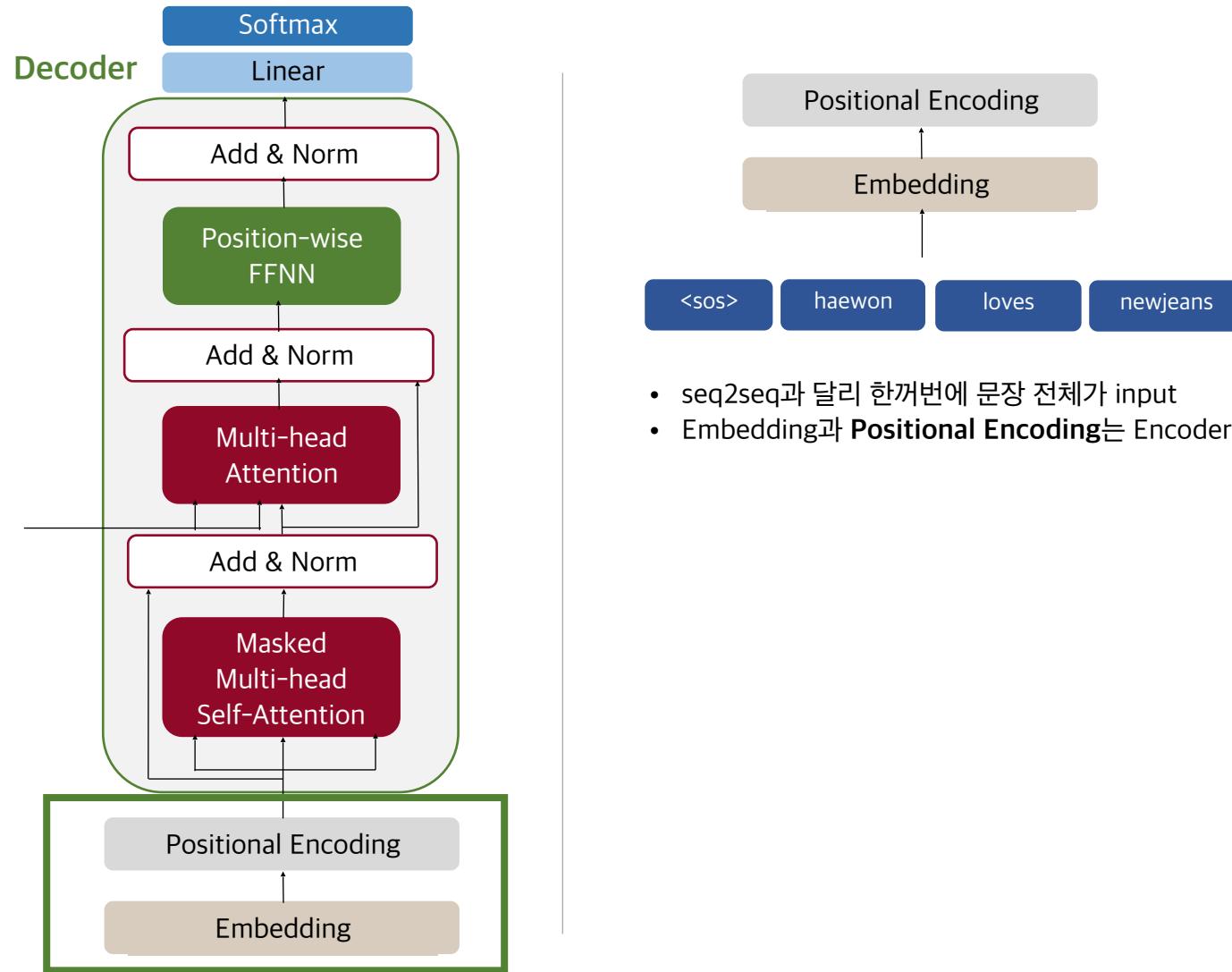
Layer Normalization

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

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

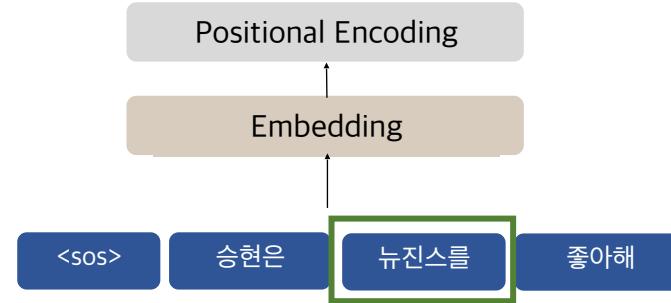
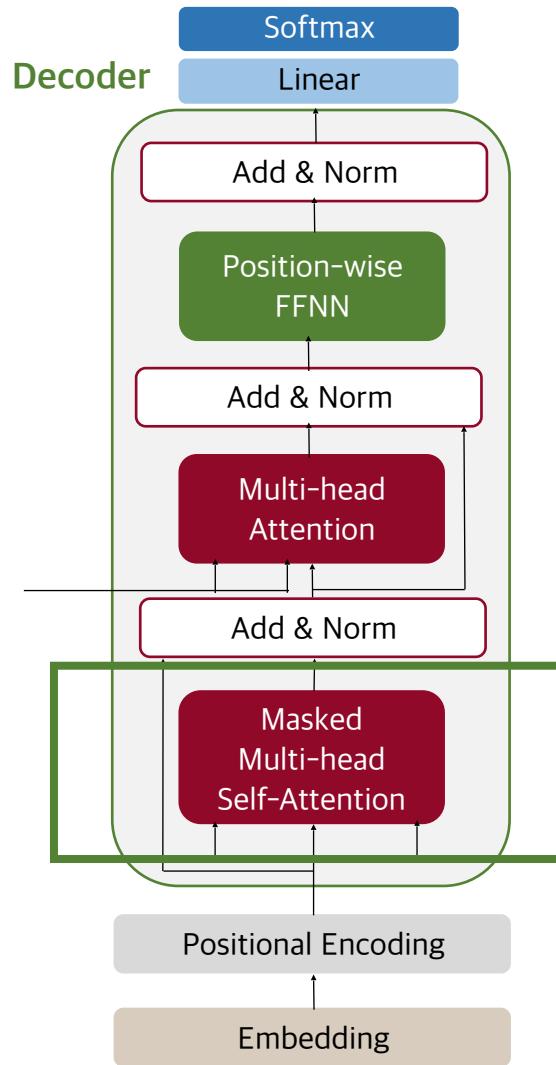


$$\gamma \left[\frac{1 - E(x_1)}{\sqrt{Var(x_i) + \epsilon}} \quad \frac{3 - E(x_1)}{\sqrt{Var(x_i) + \epsilon}} \right] + \beta$$

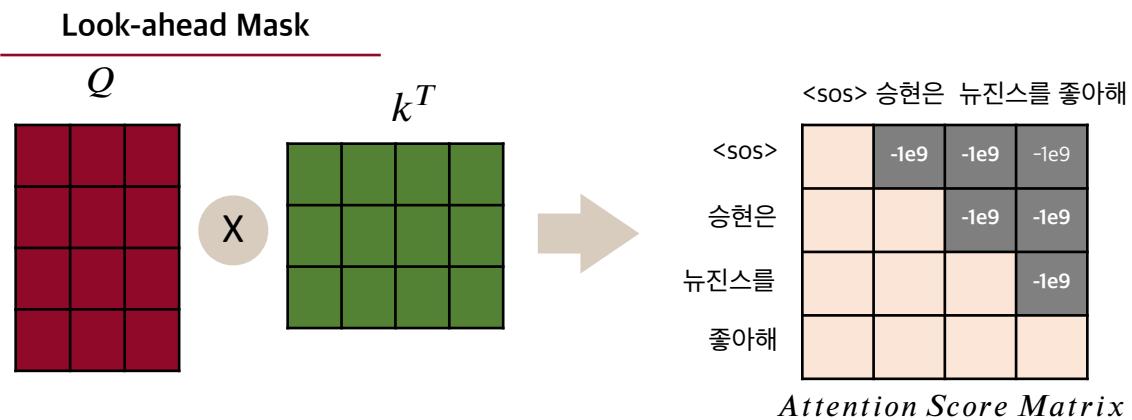


- seq2seq과 달리 한꺼번에 문장 전체가 input
- Embedding과 **Positional Encoding**은 Encoder에서와 동일함

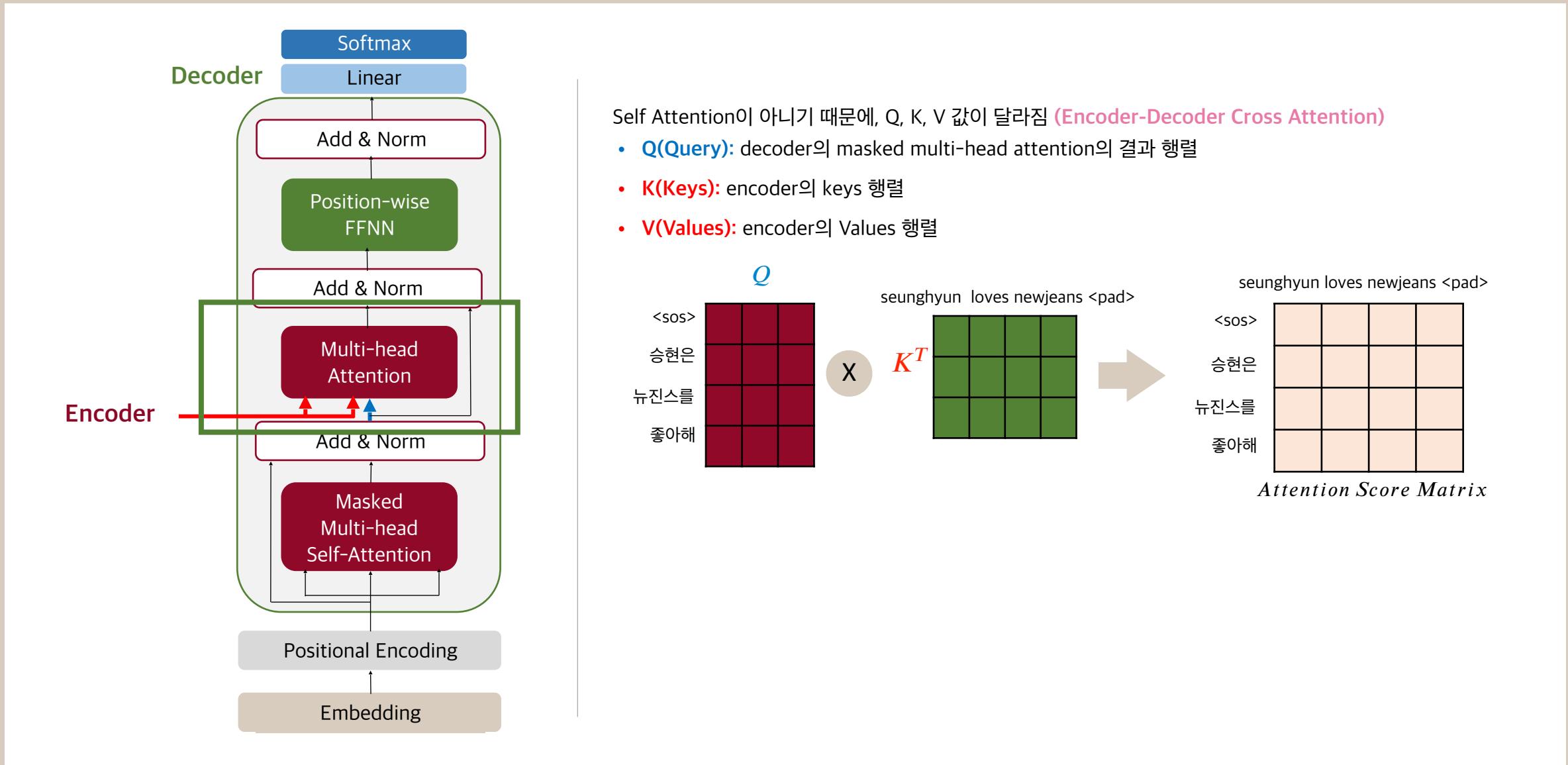
4-3. Decoder



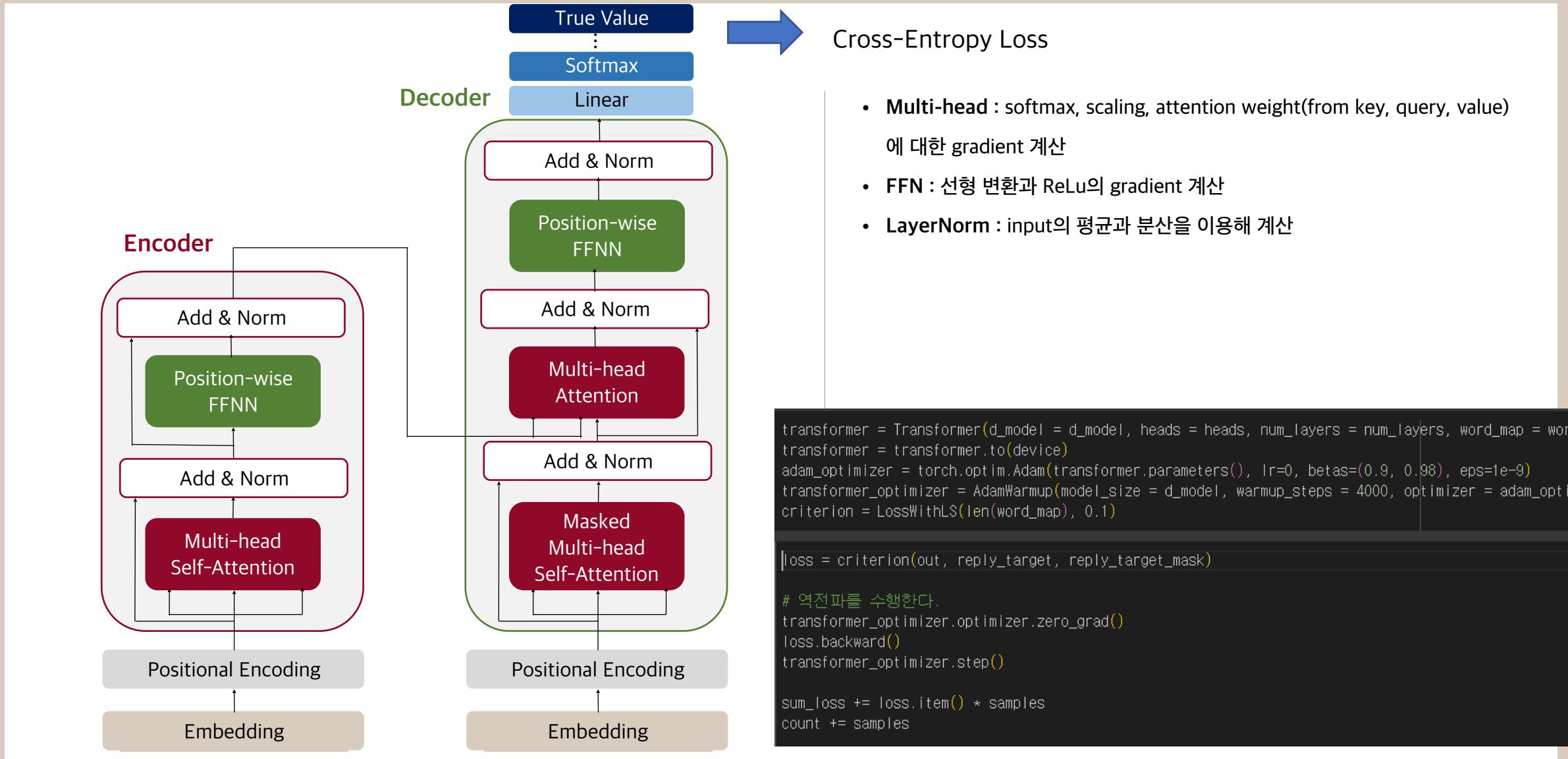
- <뉴진스를>을 예측해야 하는 시점을 가정해보자.
- Seq2seq라면 주어진 input은 <sos>, <승현은>
 - Transformer는 미래 시점인 <좋아해>까지도 이미 알고 있음
 - 미래의 단어를 미리 참고하지 못하게 Masking할 필요가 있음.



4-3. Decoder



4-4. Loss & Backpropagation



05 Announcement

Week3 예습과제 Review, week4 예복습 과제 안내

5-2. Week4 예, 복습과제 안내, Week5 진도 안내



코드과제의 파일형식은 ipynb로, KUBIG 25-1 Github repo에 업로드 될 예정입니다!
Colab 환경에서 제작된 과제들이므로 [google colab](#)에서 실행하시는 것을 권장드립니다.



WEEK4 조별 복습과제

Team summary of
'Attention is All You Need'

WEEK4 개인 복습과제

Transformer
chatbot 구현

WEEK4 예습과제1

BERT Pytorch

WEEK4 예습과제2

koalpaca 문장생성

WEEK5 진도

- BERT
- GPT

WEEK5 진도 해당 범위(읽어오시길 권장 드립니다!)

[딥러닝을 이용한 자연어 처리 입문]

- Ch17. BERT
- Ch19. GPT

수고하셨습니다!