Attention Mechanism

190408

김유리

Index

Introduction

Introduction
Idea

Attention

→ 모델이 중요한 부분에 **집중(Attention)**하게 만들자!

Introduction
Why?

Sequence data 처리 → recurrent model이 많이 쓰임

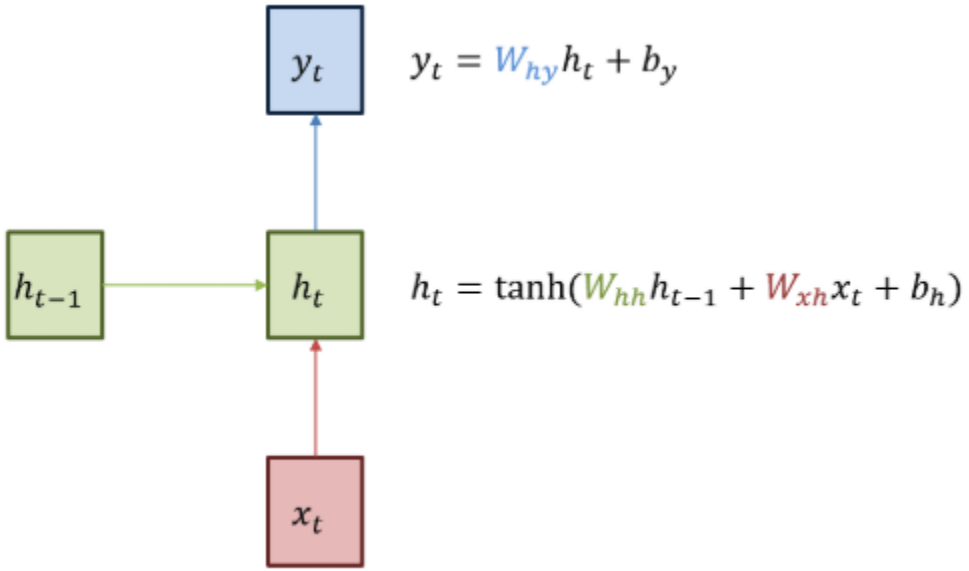
Introduction
Why?

- Long-term dependency problem
- Parallelization

Introduction
Why?

recurrent model

→ t번째에 대한 output을 만들기 위해, t번째 input과 t-1번째 hidden state 이용



(그림1) RNN 기본 구조

Introduction
Why?

∴ recurrent model은 문장의 순차적 특성 유지

하지만, 두 정보 사이의 **거리가 멀 때 해당 정보를 이용하지 못하는** 문제 발생

(Long-term dependency problem)

Introduction
Why?

recurrent model은 학습 시 t번째 hidden state를 얻기 위해 t-1번째 hidden state 필요

→ 즉, 순서대로 계산되어야 하기 때문에 **병렬 처리 불가능 & 속도가 느림**

(Parallelization)

Introduction
Why?

→ 문장의 길이가 길어 지고 층이 깊어 지면, 아래의 문제가 발생하기 때문에 Attention Mechanism 제안

- Long-term dependency problem
- Parallelization

Attention

Attention
Idea

Attention

→ 모델이 중요한 부분에 **집중(Attention)**하게 만들자!

Attention
Idea

- Decoder에서 출력 단어를 예측하는 매 시점 마다 **Encoder에서의 전체 입력 문장을 다시 참고**
- 동일한 비율로 참고하는 것이 아닌 해당 시점에서 **예측해야 할 단어와 연관 있는 입력 단어 부분**에 좀 **더 집중해서(Attention)**해서 보겠다

Attention
Idea

독일어 “Ich mochte ein bier”를 영어 “I’d like a beer”로 번역할 때,

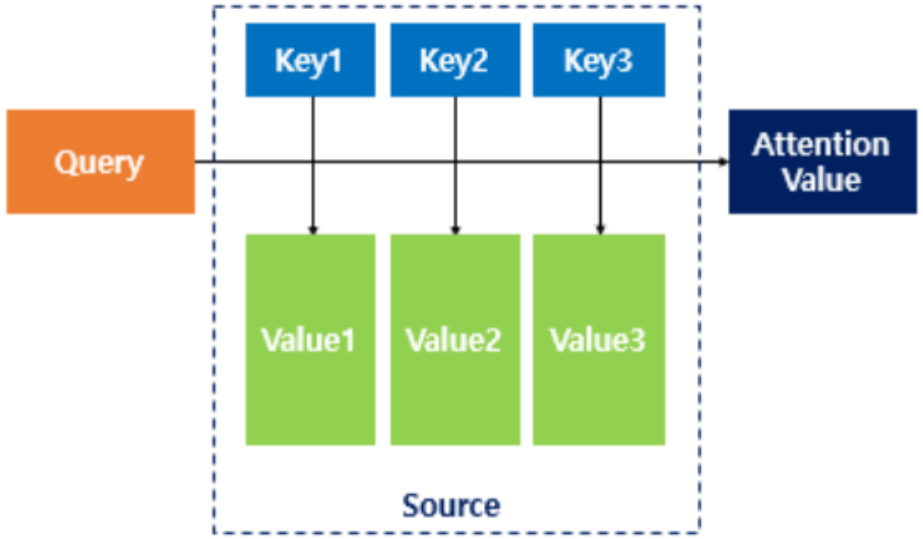
모델이 네번째 단어인 ‘beer’ 예측 시 ‘bier’에 주목하게 만드는 것

→ Encoder가 ‘bier’를 받아서 벡터로 만든 결과(Encoder 출력)은

Decoder가 ‘beer’를 예측할 때 쓰는 벡터(Decoder 입력)과 유사 할 것이다

Attention
Attention Function

Attention을 함수로 표현하면, **Attention(Q, K, V) = Attention Value**



(그림2) Attention 함수

Attention
Attention Function

Attention(Q, K, V) = Attention Value

→ Query에 대해 모든 Key와의 유사도를 각각 구함

→ 이 유사도를 키와 매핑 되어 있는 각각의 Value에 반영

→ 유사도가 반영된 Value를 모두 더해서 리턴 (이 값이 Attention Value)

Attention
Attention Function

Seq2seq+Attention 모델에서 Q, K, V ?

Attention(Q, K, V) = Attention Value

Q = Query : **Decoder의 t-1 셀에서의 은닉 상태**

K = Keys : **모든 Encoder 셀의 은닉 상태들**

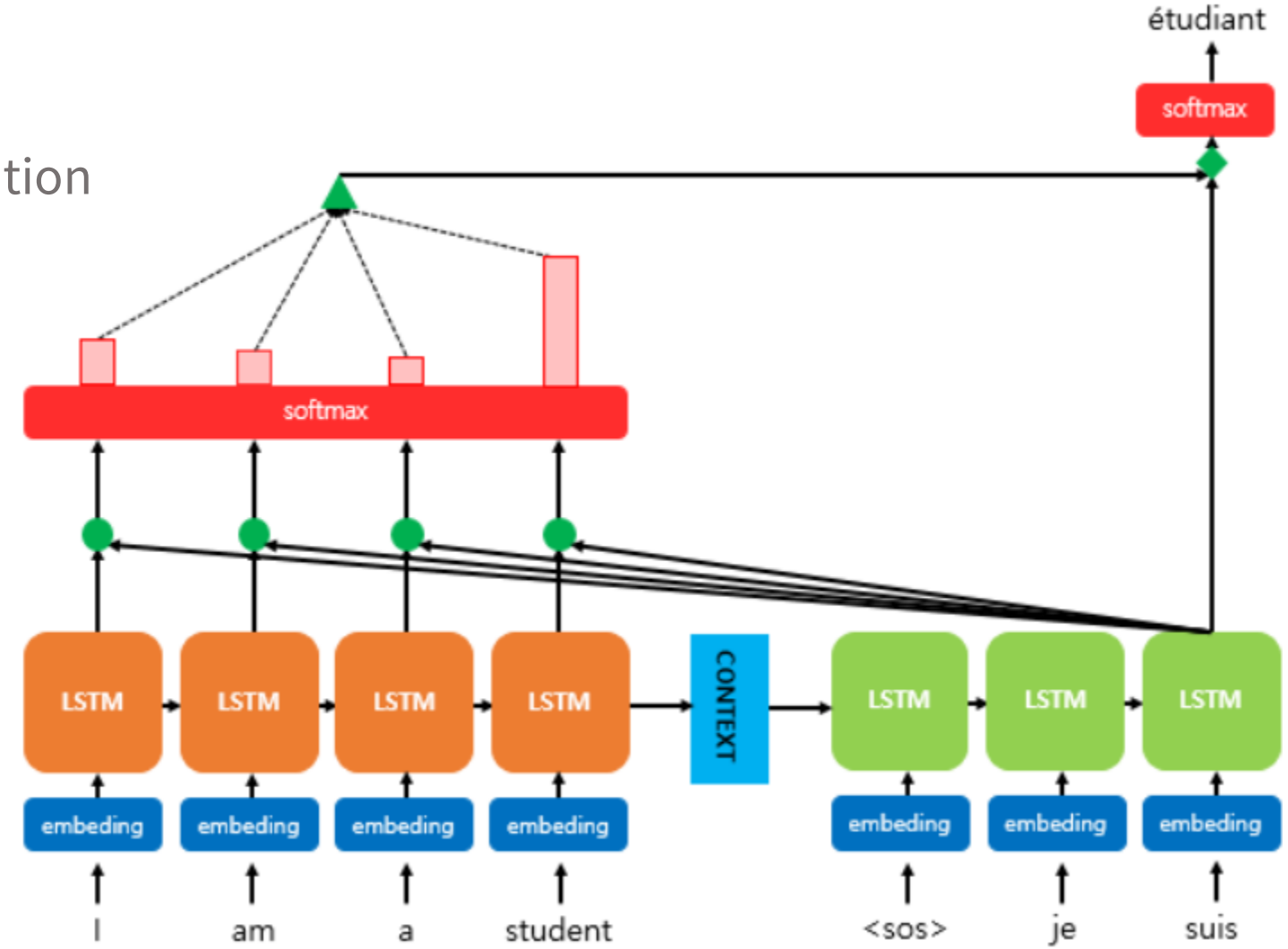
V = Values : **모든 Encoder 셀의 은닉 상태들**

Attention
Dot-Product Attention

Dot-Product Attention

(다른 Attention model과 차이는 중간 수식의 차이이며 메커니즘 자체는 동일)

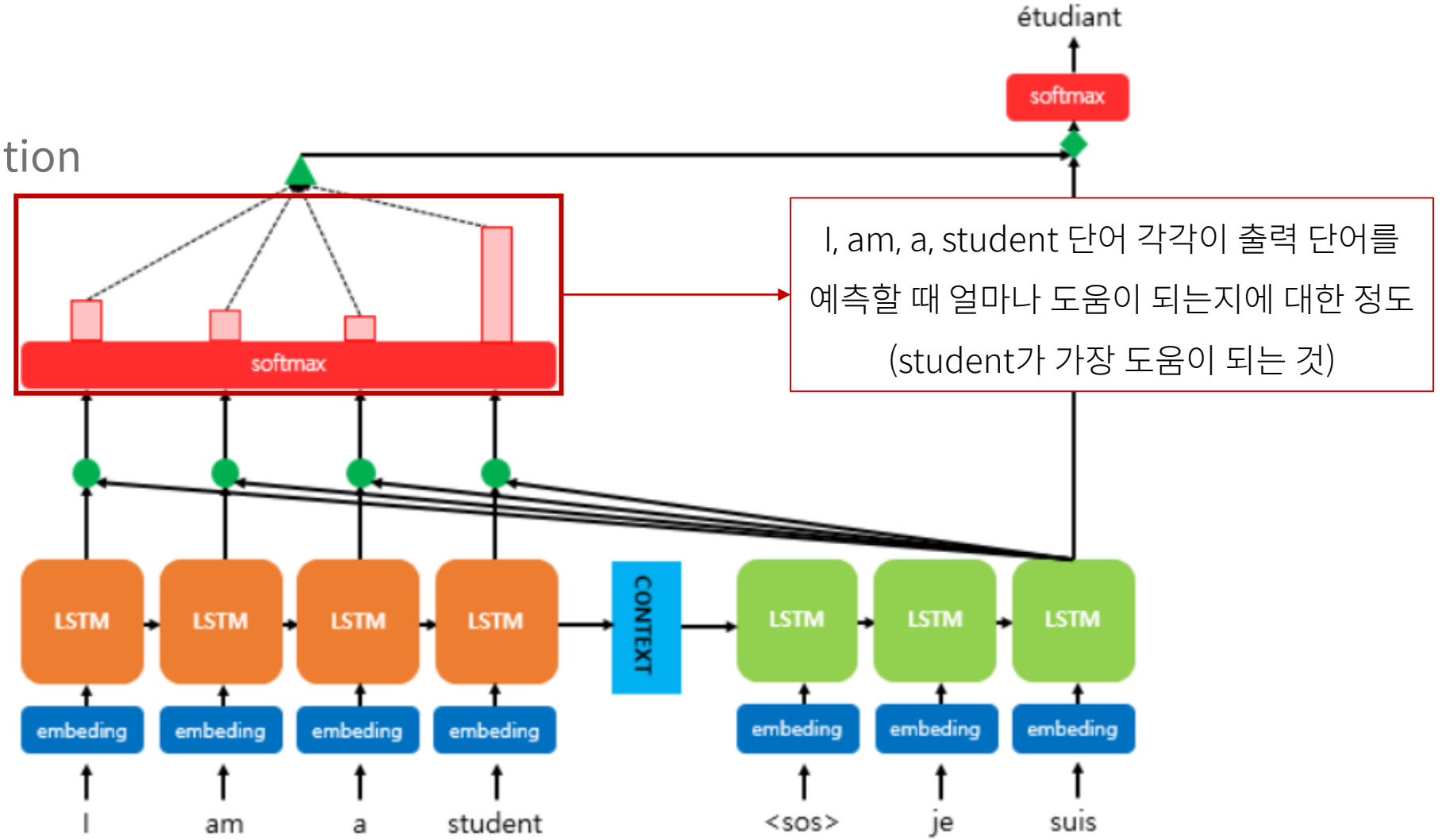
Attention
Dot-Product Attention



Decoder의 세번째 LSTM 셀에서 출력 단어를 예측할 때, Attention 메커니즘을 사용하는 모습

(1, 2번째 LSTM 셀은 이미 Attention 메커니즘을 통해 'je'와 'suis'를 예측하는 과정을 거쳤다고 가정)

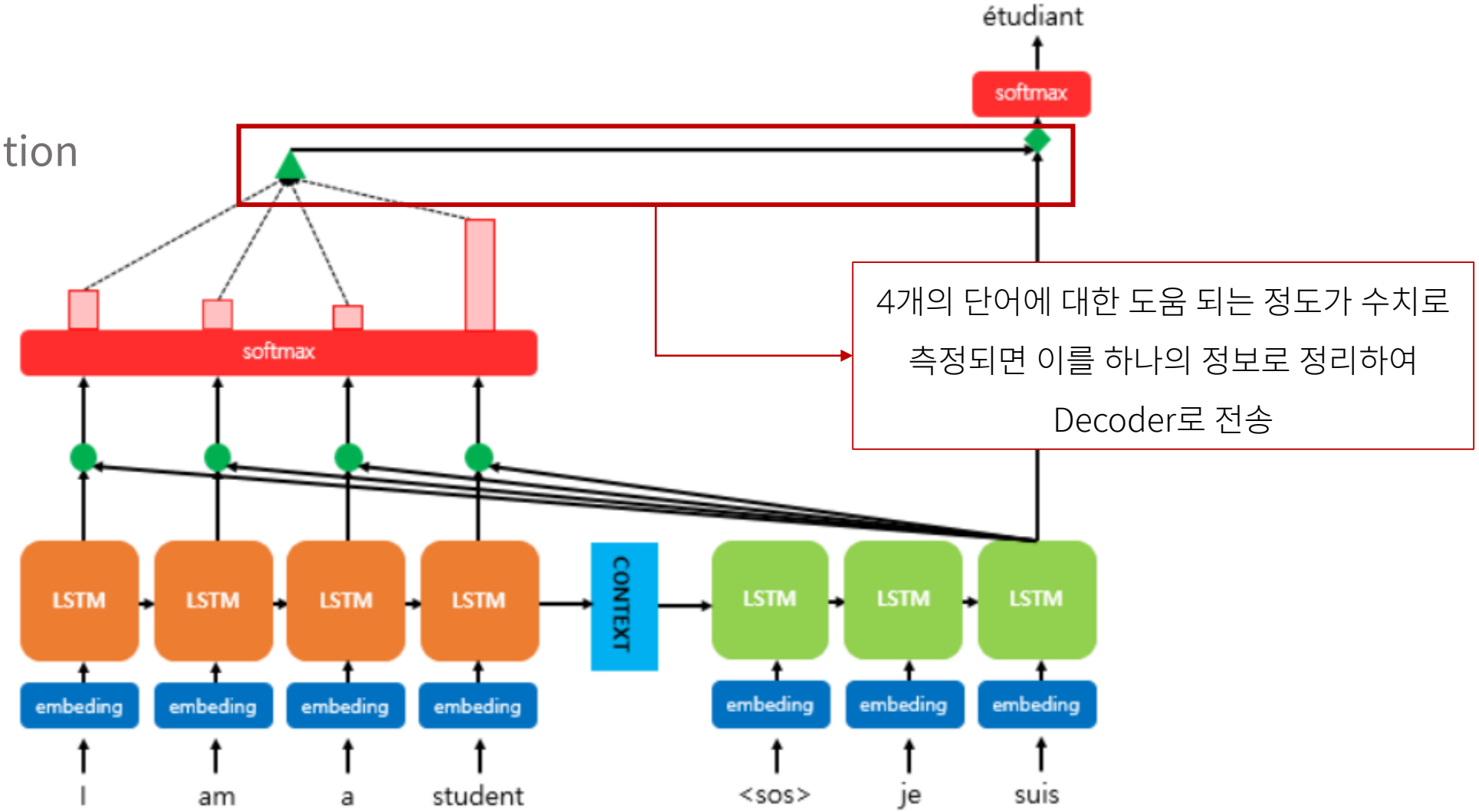
Attention
Dot-Product Attention



Decoder의 세번째 LSTM 셀에서 출력 단어를 예측할 때, Attention 메커니즘을 사용하는 모습

(1, 2번째 LSTM 셀은 이미 Attention 메커니즘을 통해 'je'와 'suis'를 예측하는 과정을 거쳤다고 가정)

Attention
Dot-Product Attention



Decoder의 세번째 LSTM 셀에서 출력 단어를 예측할 때, Attention 메커니즘을 사용하는 모습

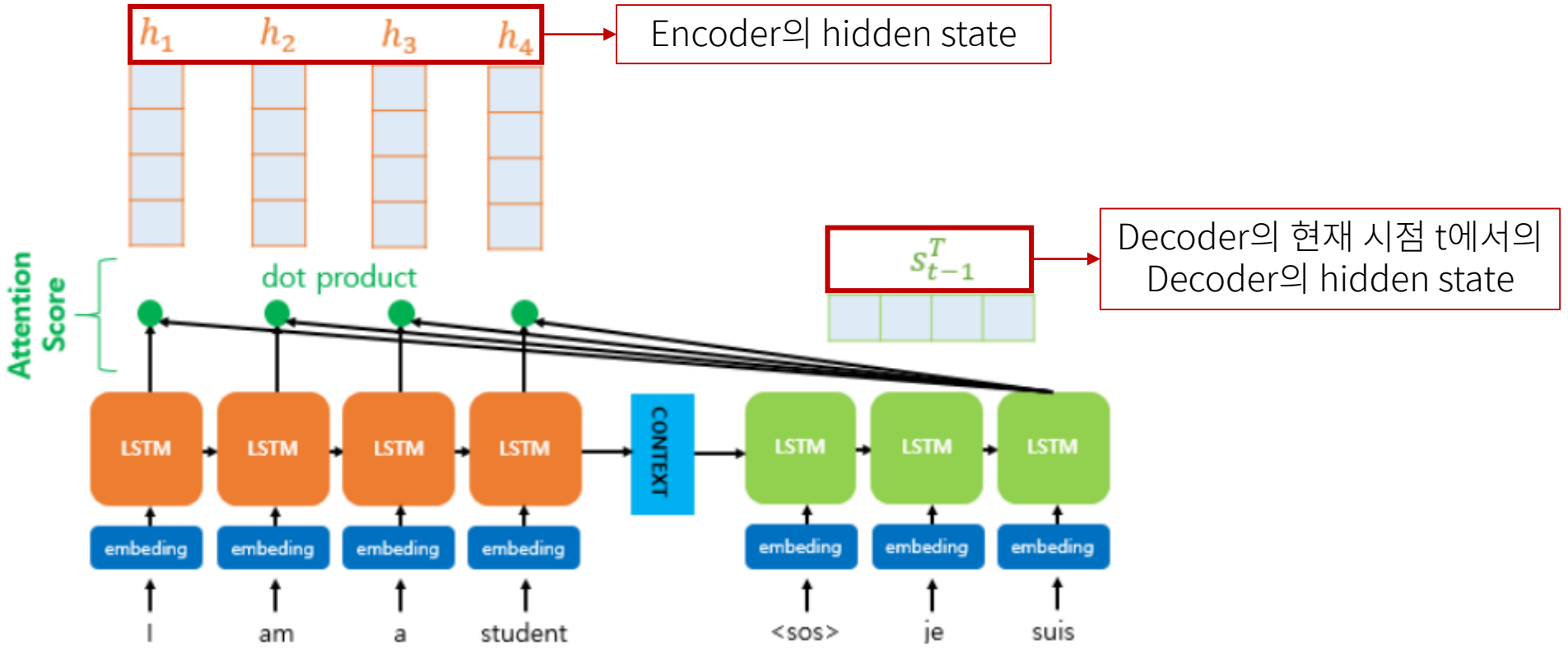
(1, 2번째 LSTM 셀은 이미 Attention 메커니즘을 통해 'je'와 'suis'를 예측하는 과정을 거쳤다고 가정)

Attention
Dot-Product Attention

더 자세하게 알아봅시다!

Attention
Dot-Product Attention

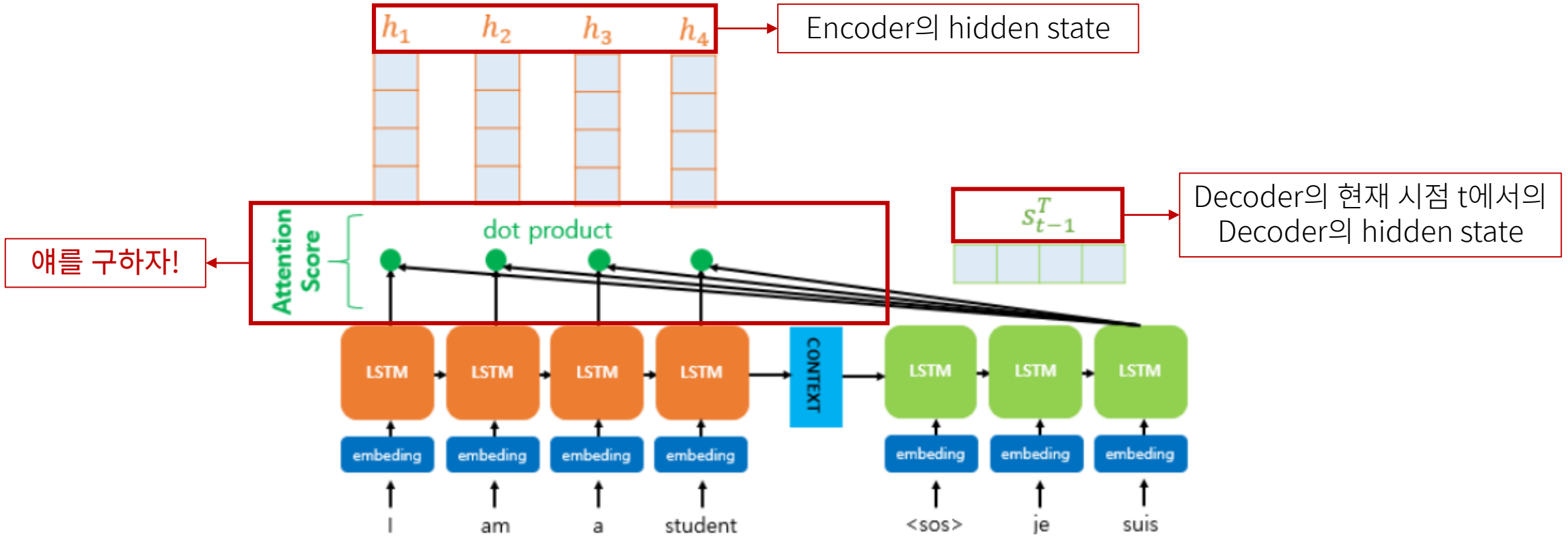
1. Attention Score를 구한다



여기서는 Encoder의 hidden state와 Decoder의 hidden state가 같다고 가정

Attention
Dot-Product Attention

1. Attention Score를 구한다



여기서는 Encoder의 hidden state와 Decoder의 hidden state가 같다고 가정

Attention
Dot-Product Attention

1. Attention Score를 구한다

Decoder의 현재 시점 t에서 필요한 입력 값 : t-1의 hidden state와 출력 단어

$$S_t = f(S_{t-1}, y_{t-1})$$

Attention Mechanism에서는 이를 예측하기 위해 Attention Value라는 새로운 값이 추가로 필요

현재 시점 t에서의 Attention Value를 a_t라고 할 때, 현재 시점 t에서의 hidden state는,

$$S_t = f(S_{t-1}, y_{t-1}, a_t)$$

S_t : Decoder의 현재 시점 t에서의 hidden state

S_{t-1} : Decoder의 이전 시점 t-1에서의 hidden state

y_{t-1} Decoder의 이전 시점 t-1에서의 출력 단어

Attention
Dot-Product Attention

1. Attention Score를 구한다

Attention Score란?

Decoder의 현재 시점 t에서 단어를 예측하기 위해 Encoder의 모든 은닉 상태 각각이

Decoder의 전 시점 은닉상태 S_{t-1} 와 얼마나 유사한지를 판단하는 Score 값

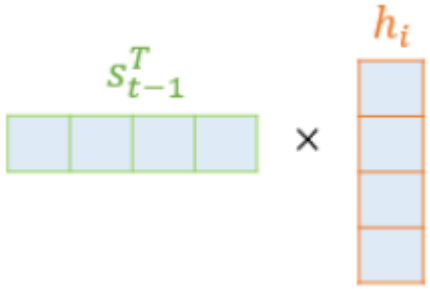
Attention
Dot-Product Attention

1. Attention Score를 구한다

Dot-Product Attention에서 Attention Score 구하기

S_{t-1} 를 전치(Transpose)하고 각각의 hidden state와 스칼라 곱(dot product) 수행

(즉, 모든 Attention Score 값은 스칼라)



Attention Score 함수 정의

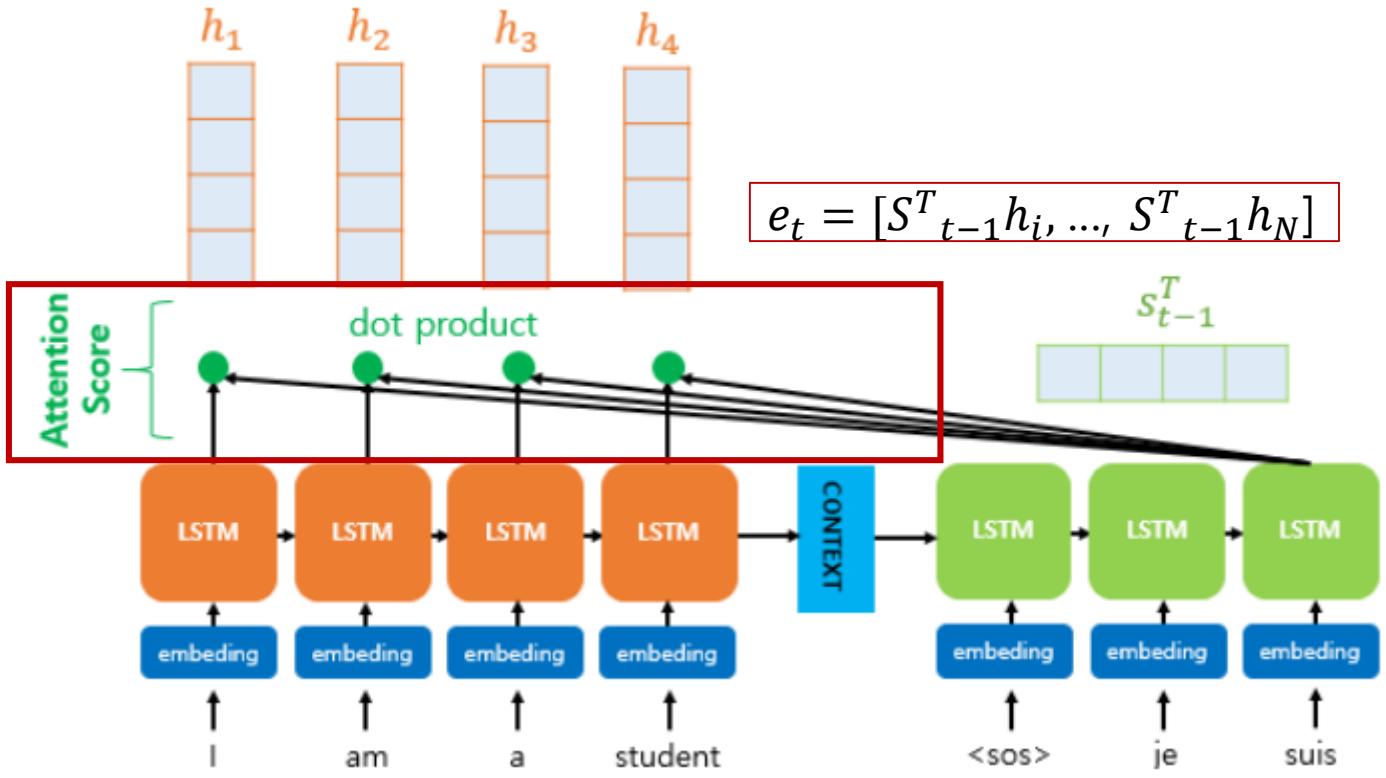
$$Score(S_{t-1}, h_i) = S^T{}_{t-1} h_i$$

S_{t-1} 와 Encoder의 모든 hidden state의 Attention Score 모음 e_t 정의

$$e_t = [S^T{}_{t-1} h_i, \ldots, S^T{}_{t-1} h_N]$$

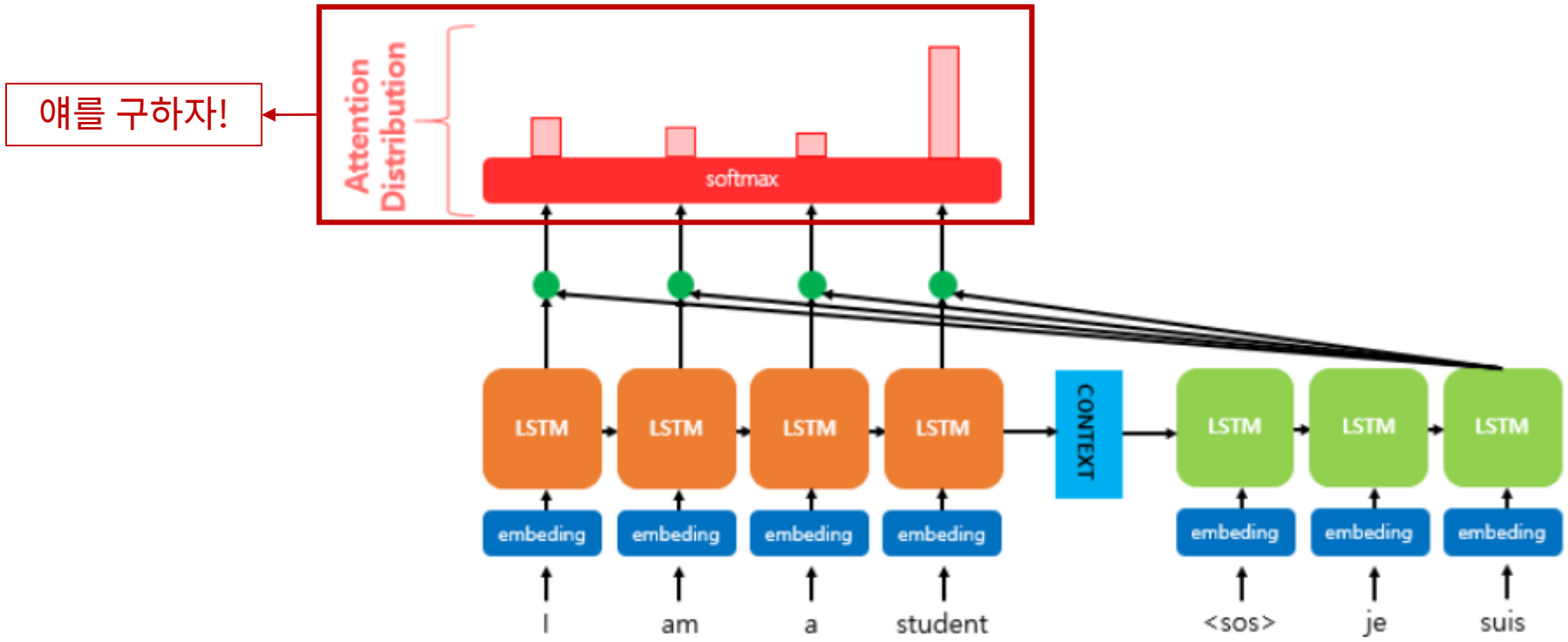
Attention
Dot-Product Attention

1. Attention Score를 구한다



Attention
Dot-Product Attention

2. softmax 함수를 통해 Attention Distribution을 구한다



Attention
Dot-Product Attention

2. softmax 함수를 통해 Attention Distribution을 구한다

e_t 에 **softmax 함수를 적용**해 모든 값의 합이 1이 되는 확률분포(Attention Distribution) 구함

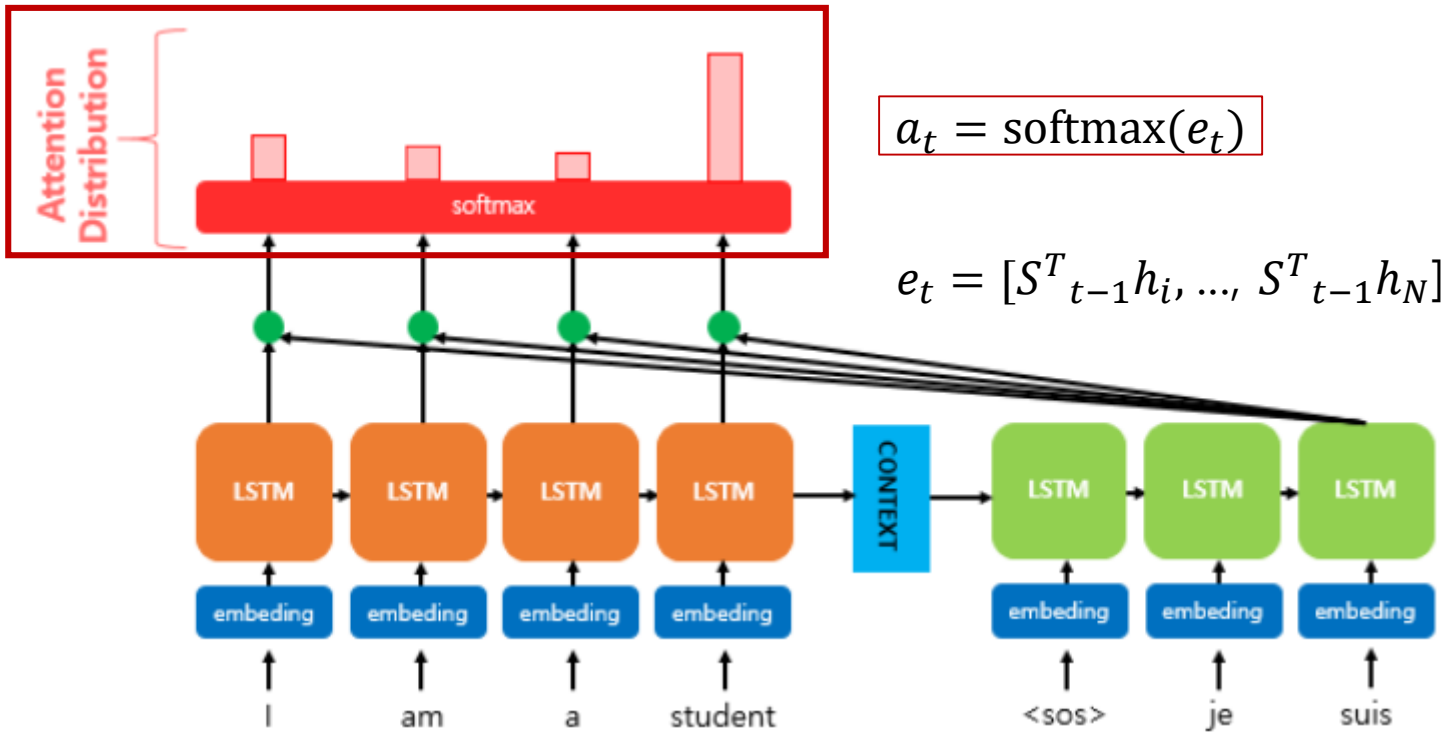
(ex. I, am, a, student의 Attention 가중치가 각각 0.18, 0.12, 0.06, 0.64라고 할 때 가중치의 합은 1이며, student의 가중치가 가장 큼)

Decoder의 시점 t에서 Attention 가중치를 a_t 라고 하면,

$$a_t = \text{softmax}(e_t)$$

Attention
Dot-Product Attention

2. softmax 함수를 통해 Attention Distribution을 구한다

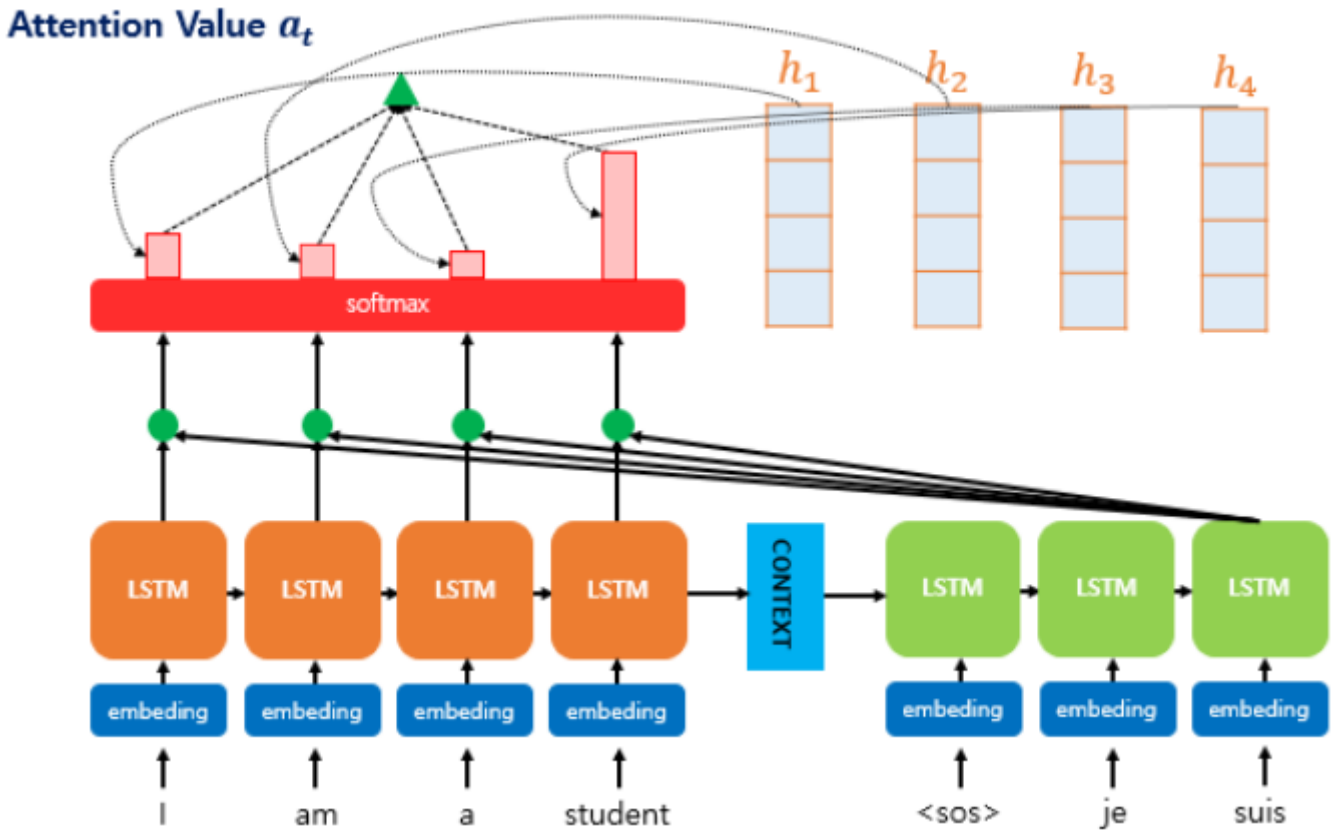


$$a_t = \text{softmax}(e_t)$$

$$e_t = [S^T{}_{t-1}h_i, \ldots, S^T{}_{t-1}h_N]$$

Attention
Dot-Product Attention

3. 각 Encoder의 Attention 가중치와 hidden state를 가중 합 하여 Attention Value를 구한다



Attention
Dot-Product Attention

3. 각 Encoder의 Attention 가중치와 hidden state를 가중 합 하여 Attention Value를 구한다

Attention의 최종 결과 값을 구하기 위해

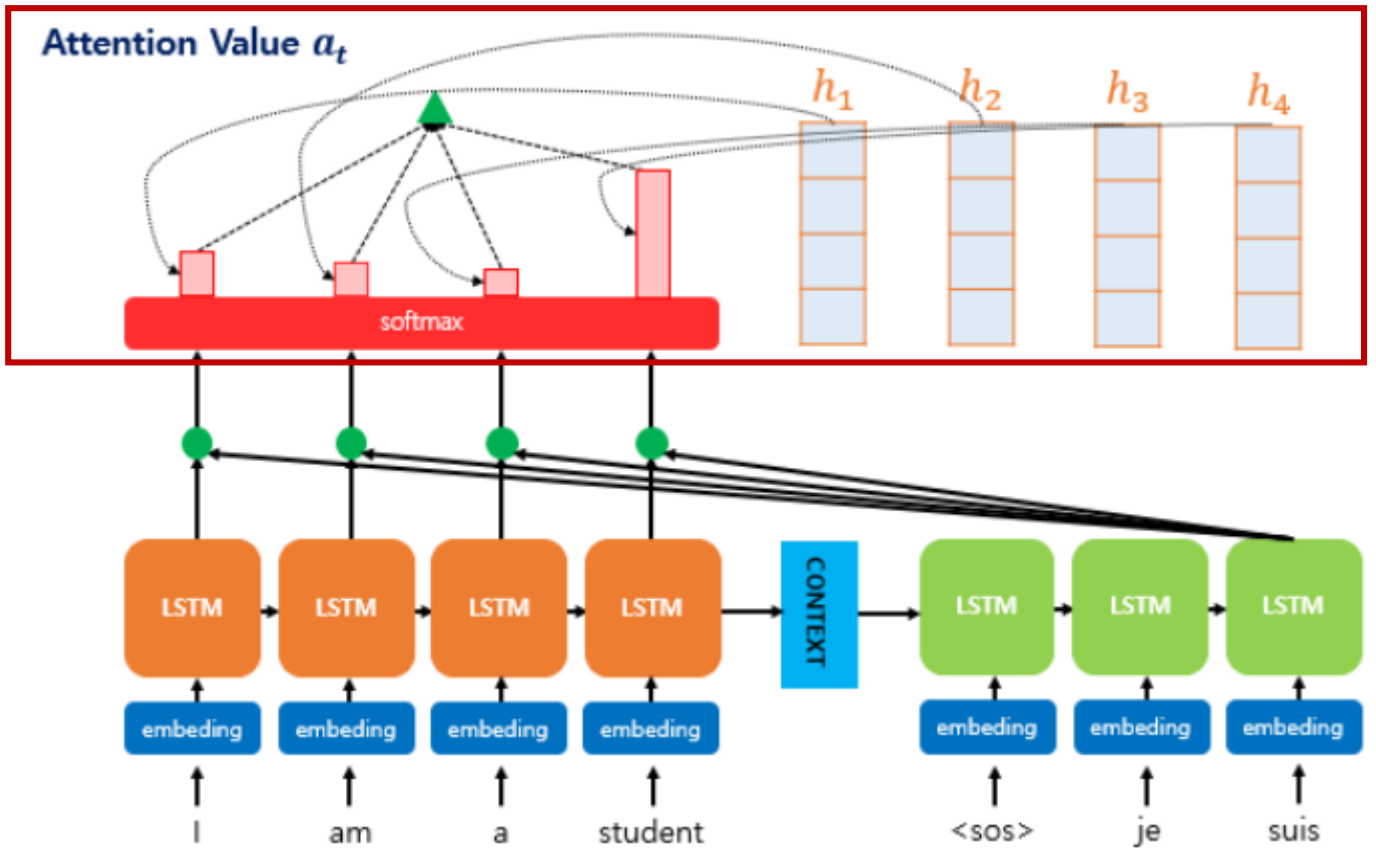
각 Encoder의 hidden state와 Attention 가중치를 곱하고 모두 더함(가중 합)

Attention 함수의 출력 값인 Attention Value a_t

$$a_t = \sum_{i=1}^{N} \alpha^t{}_i h_i$$

Attention
Dot-Product Attention

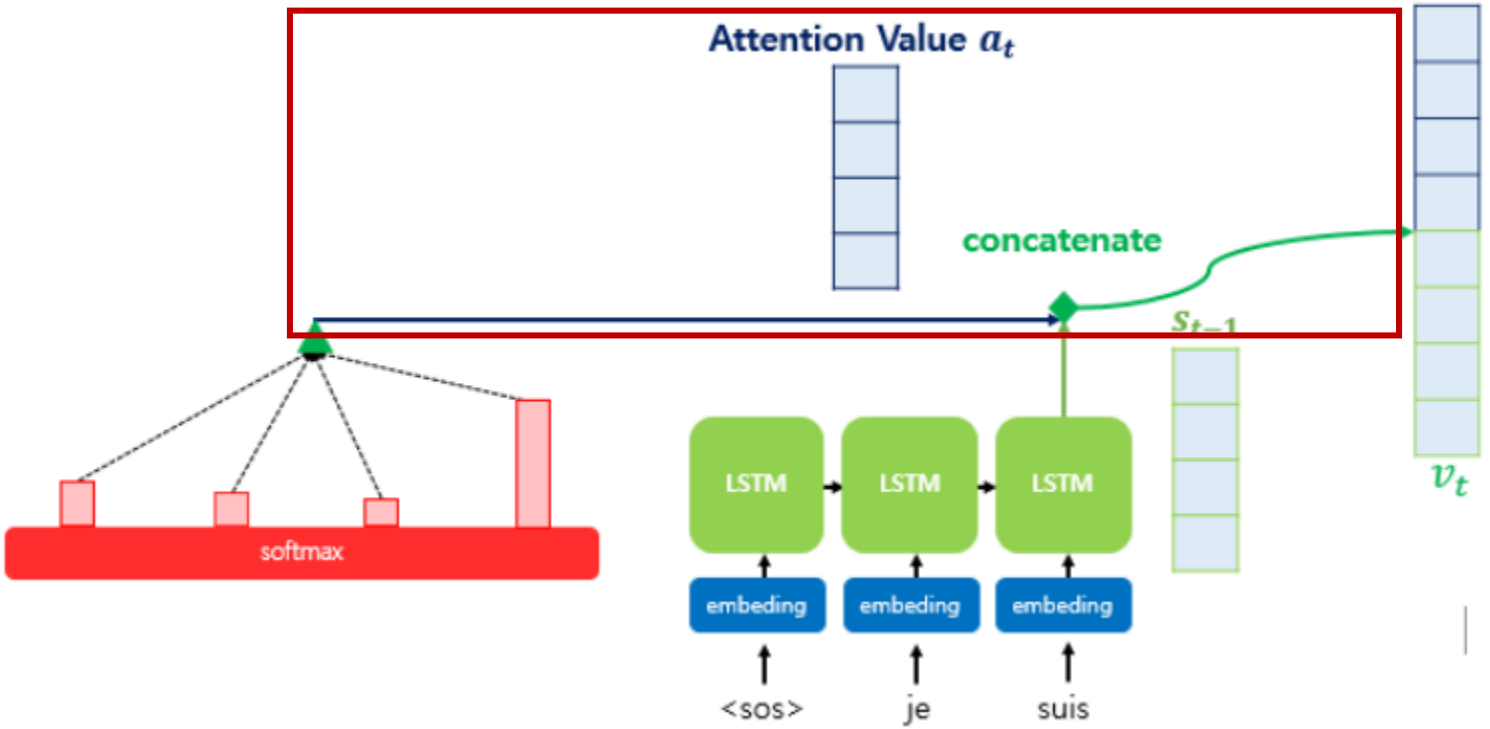
3. 각 Encoder의 Attention 가중치와 hidden state를 가중 합 하여 Attention Value를 구한다



$$a_t = \sum_{i=1}^{N} \alpha^t{}_i h_i$$

Attention
Dot-Product Attention

4. Attention Value와 Decoder의 t-1 시점의 hidden state를 결합한다 (Concatenate)



Attention
Dot-Product Attention

4. Attention Value와 Decoder의 t-1 시점의 hidden state를 결합한다 (Concatenate)

앞서 Attention Value를 a_t라고 할 때, 현재 시점 t에서의 hidden state는 아래와 같다고 했는데,

$$S_t = f(S_{t-1}, y_{t-1}, a_t)$$

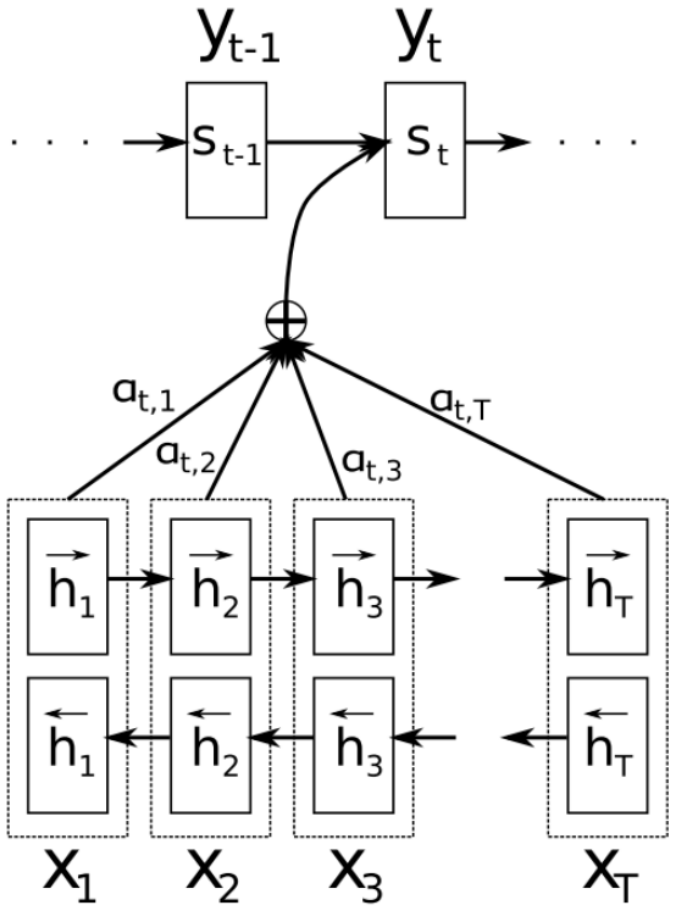
Attention Value가 구해지면 Attention Mechanism은

a_t 와 S_{t-1}을 결합하여 하나의 벡터 v_t 로 만드는 작업을 수행

즉, t 시점에서의 hidden state를 구하는 식은

$$S_t = f(v_t, y_{t-1})$$

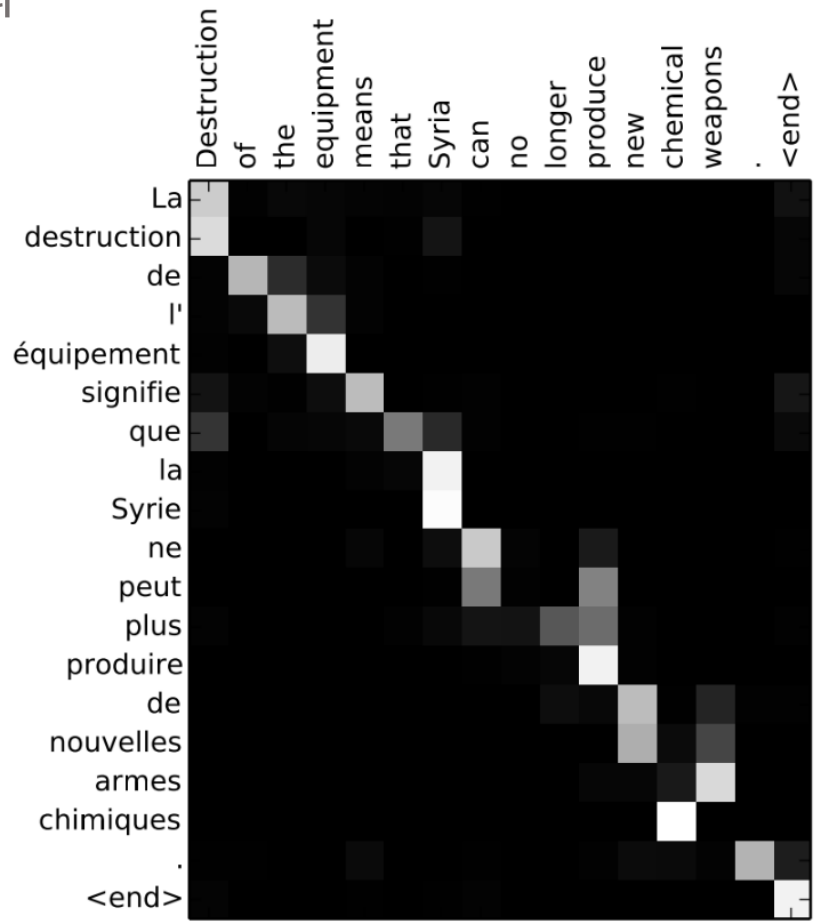
Attention
요약



Attention Model의 개념적 동작

- Input Sequence 벡터들의 중요도를 현재 상태를 고려해 계산
- 각각의 중요도를 합이 1인 상대 값으로 변환
- 상대 값 중요도를 가중치로 두고 Sequence 벡터들을 가중 합

Attention
시각화



Attention 기반의 모델을 이용하면 Attention score을 이용해 이러한 alignment table을 그려볼 수 있고, 모델이 제대로 학습하고 있는가를 살펴볼 수 있음

Improve

Improve
Global & Local

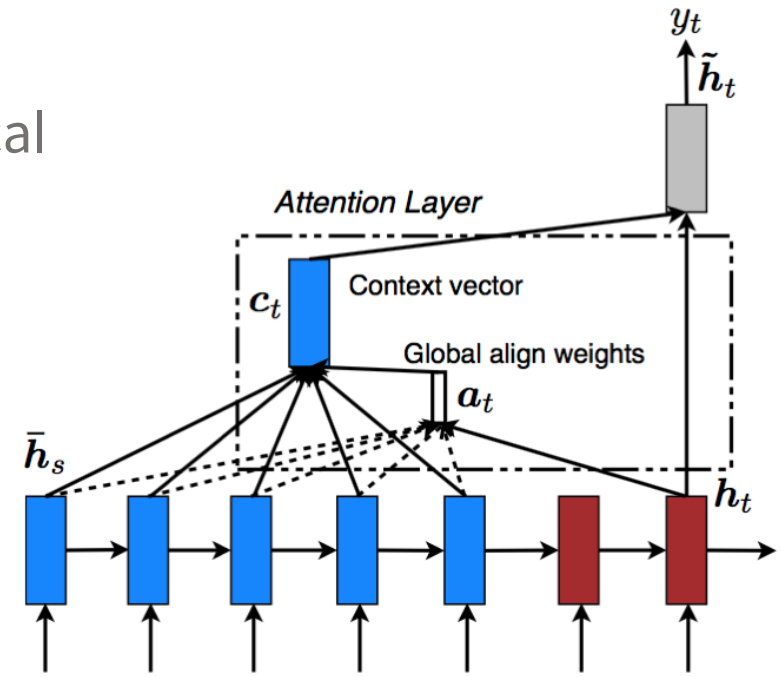


Figure 2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector $\boldsymbol{a}_t$ based on the current target state $\boldsymbol{h}_t$ and all source states $\bar{\boldsymbol{h}}_s$. A global context vector $\boldsymbol{c}_t$ is then computed as the weighted average, according to $\boldsymbol{a}_t$, over all the source states.

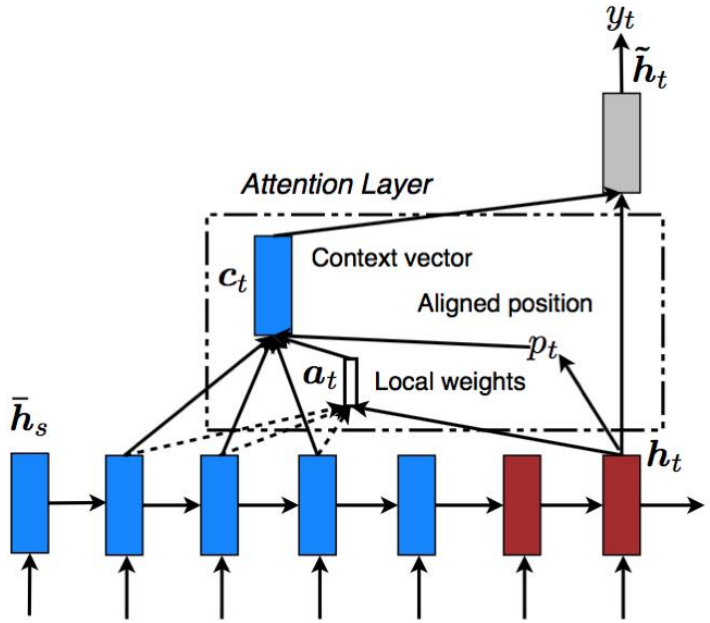
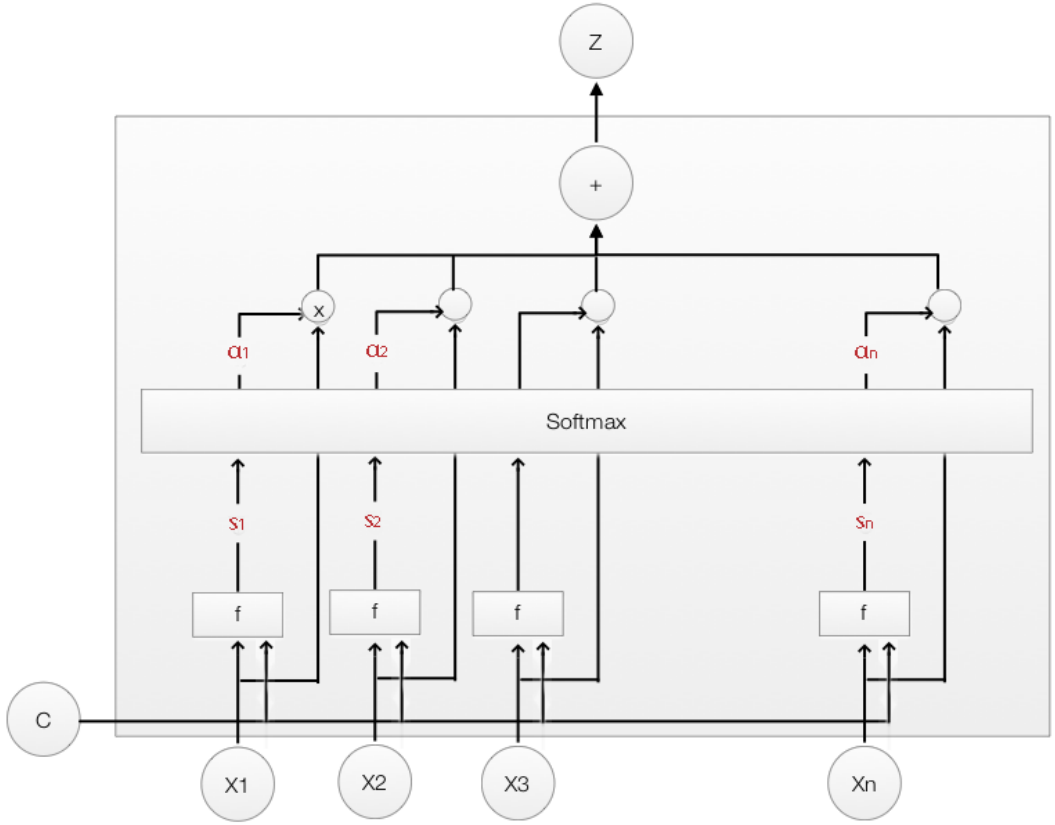


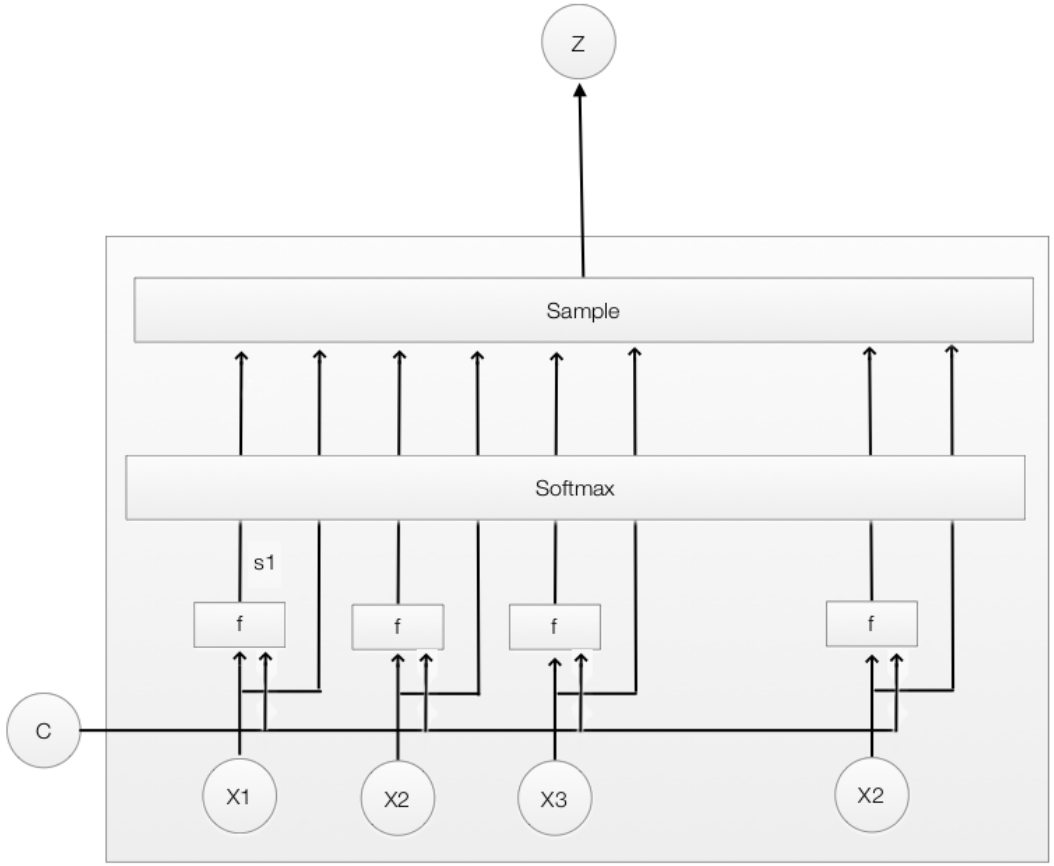
Figure 3: **Local attention model** – the model first predicts a single aligned position p_t for the current target word. A window centered around the source position p_t is then used to compute a context vector $\boldsymbol{c}_t$, a weighted average of the source hidden states in the window. The weights $\boldsymbol{a}_t$ are inferred from the current target state $\boldsymbol{h}_t$ and those source states $\bar{\boldsymbol{h}}_s$ in the window.

- global attention : 인코더의 전체 hidden state에 대해서 가중치를 계산
- local attention : 윈도우를 이용해 대략적으로 입력 문장의 단어를 추려서 계산

Improve
Soft & Hard



Soft Attention



Hard Attention

Thank you

References
paper

- Attention Is All You Need (https://arxiv.org/abs/1706.03762)
- Neural Machine Translation by Jointly Learning to Align and Translate (https://arxiv.org/abs/1409.0473)
- Effective Approaches to Attention-based Neural Machine Translation (https://aclweb.org/anthology/D15-1166)