Attention

명지대학교
MYONGJI UNIVERSITY

Seq2Seq의 한계

Seq2Seq의 문제점

인코더가 입력 시퀀스를 하나의 벡터로 압축하는 과정에서 입력 시퀀스의 정보가 일부 손실

RNN 구조의 근본적인 문제점

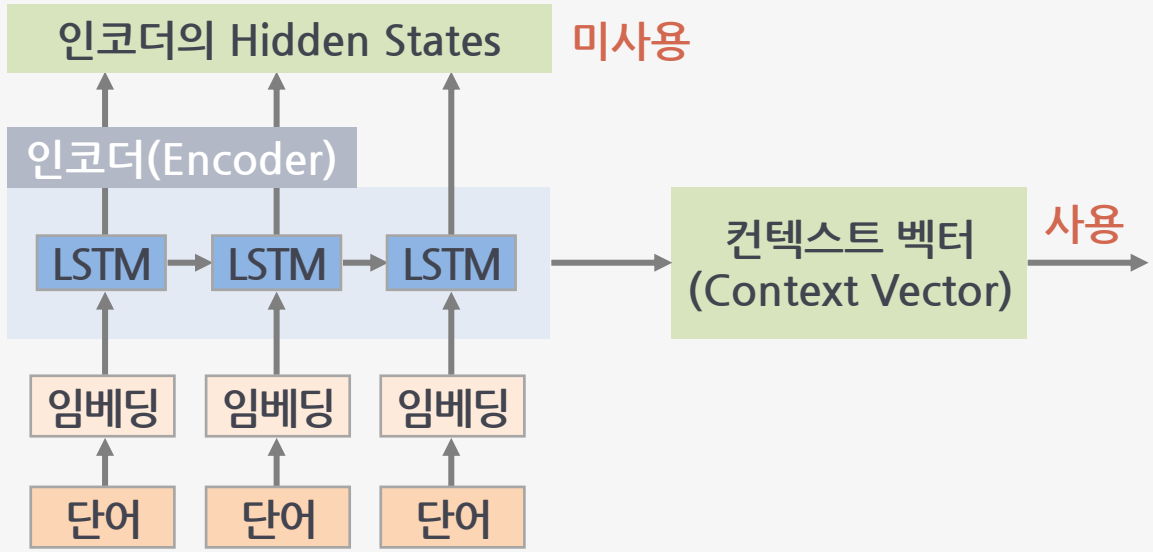
- Vanishing Gradient 발생 가능
- 입력 데이터의 길이가 길어지면 성능이 저하되는 현상 발생 가능

입력 데이터의 길이가 길어지더라도 성능이 저하되는 것을 방지하기 위해 어텐션(Attention) 메커니즘의 등장

Seq2Seq의 한계

Seq2Seq의 문제점

- 인코더의 Time step마다 출력하는 Hidden States는 미사용
- 인코더의 마지막 Hidden State인 컨텍스트 벡터만 디코더에서 사용,
  어텐션 메커니즘(Attention Mechanism)은 미사용 Hidden State를 활용



**어텐션의 기본 아이디어는 디코더의 출력 결과를 예측하는
매 시점(Time step)마다, 인코더의 Hidden State를 입력으로 전달하여 참고**

어텐션(Attention) 메커니즘

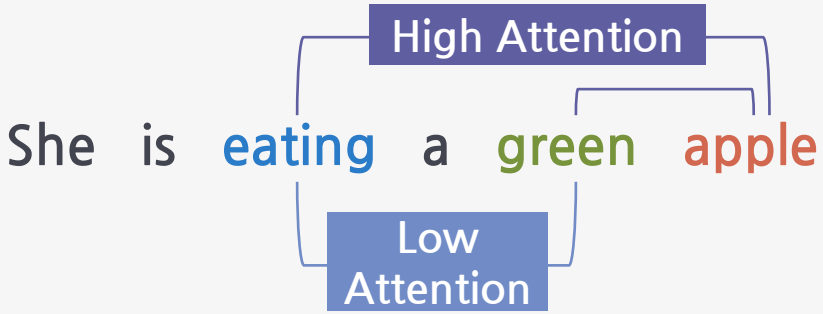
어텐션의 기본 아이디어는 디코더의 출력 결과를 예측하는
매 시점(Time step)마다 인코더의 Hidden State를 입력으로 전달하여 참고

전체 입력 문장 모두를 동일한 비율로 참고하지 않음

해당 Time step에서 예측해야 할 단어와 연관 있는
입력 단어에 집중(Attention)

어텐션(Attention) 메커니즘

같은 문장 내 모든 단어 쌍 사이의 의미적 문법적 관계를 파악하여
밀접한 관계를 가지는 단어에 High Attention 부여



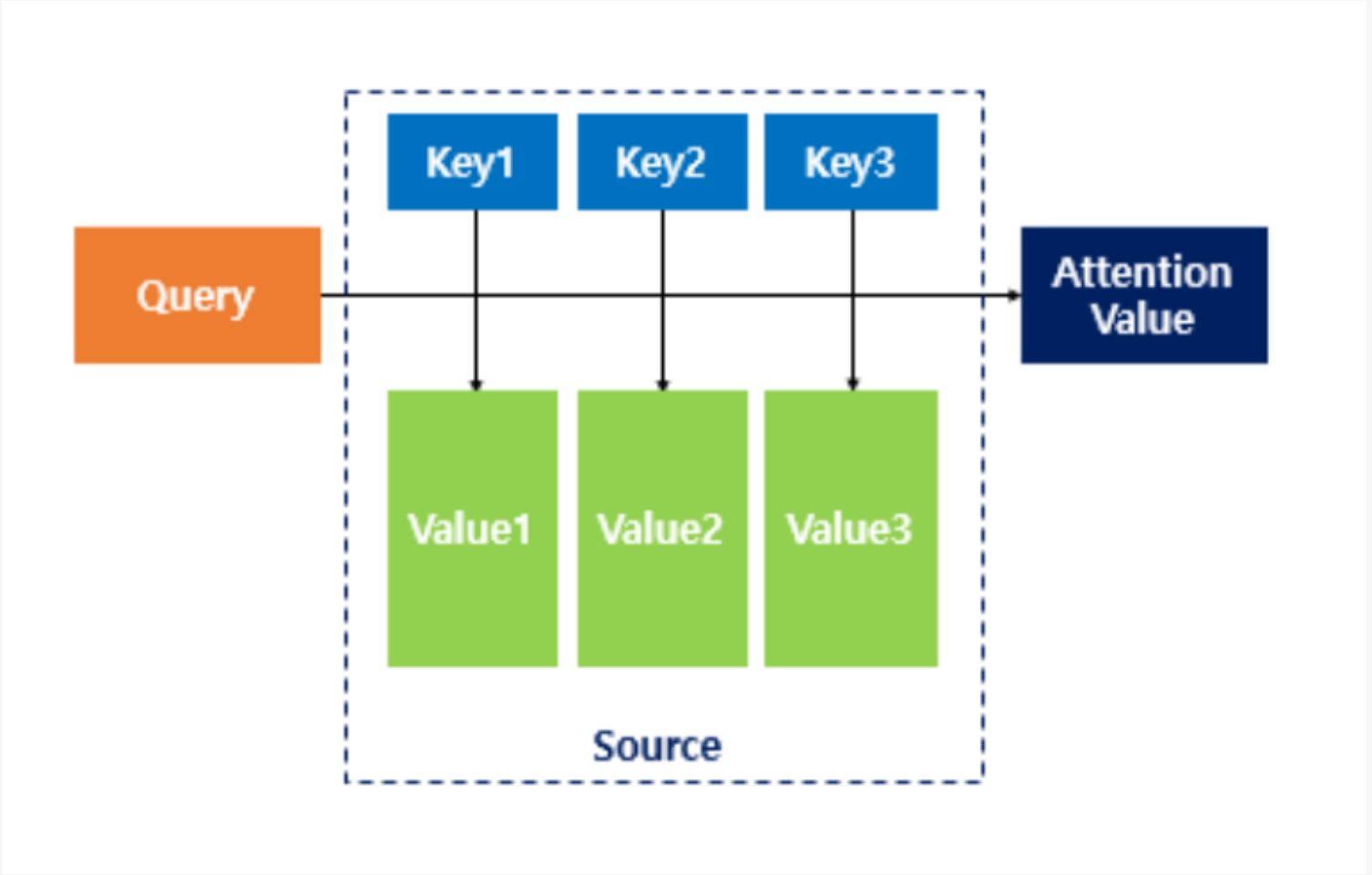
- eating은 apple과 밀접한 연관관계를 가지는 High Attention
- green은 apple과 밀접한 연관관계를 가지는 High Attention

- eating은 green과 밀접한 연관관계를 가지지 않는 Low Attention

어텐션(Attention)

어텐션(Attention) 함수

Query / Key / Value



어텐션(Attention) 함수

Query / Key / Value

Q (Query)	특정 Time step에서의 Decoder Cell의 Hidden State
K (Keys)	모든 Times step에서의 Encoder Cell의 Query 반영 전 Hidden State
V (Values)	모든 Times step에서의 Encoder Cell의 Query 반영 후 Hidden State

어텐션(Attention) 함수

주어진 '쿼리(Query)'에 대해서 모든 '키(Key)'와 각각의 유사도 계산
(유사도는 내적, 외적 등 다양한 방법으로 계산)

유사도를 정규화하여 각 단어의 가중치를 계산

가중치와 Value 벡터와 곱셈연산 후 최종 출력 벡터가 생성

Attention Value = Attention(Q, K, V)

어텐션(Attention) 함수

Attention Function은 다양한 방법으로 구현 가능

- Scaled Dot-Product Attention : 내적 연산을 사용하여 유사도를 계산
- Multi-Head Attention : 다수의 Query, Key, Value 벡터를 사용하여 각각의 가중치를 계산한 후 이를 결합

어텐션(Attention) 적용 과정

Scaled Dot-Product Attention

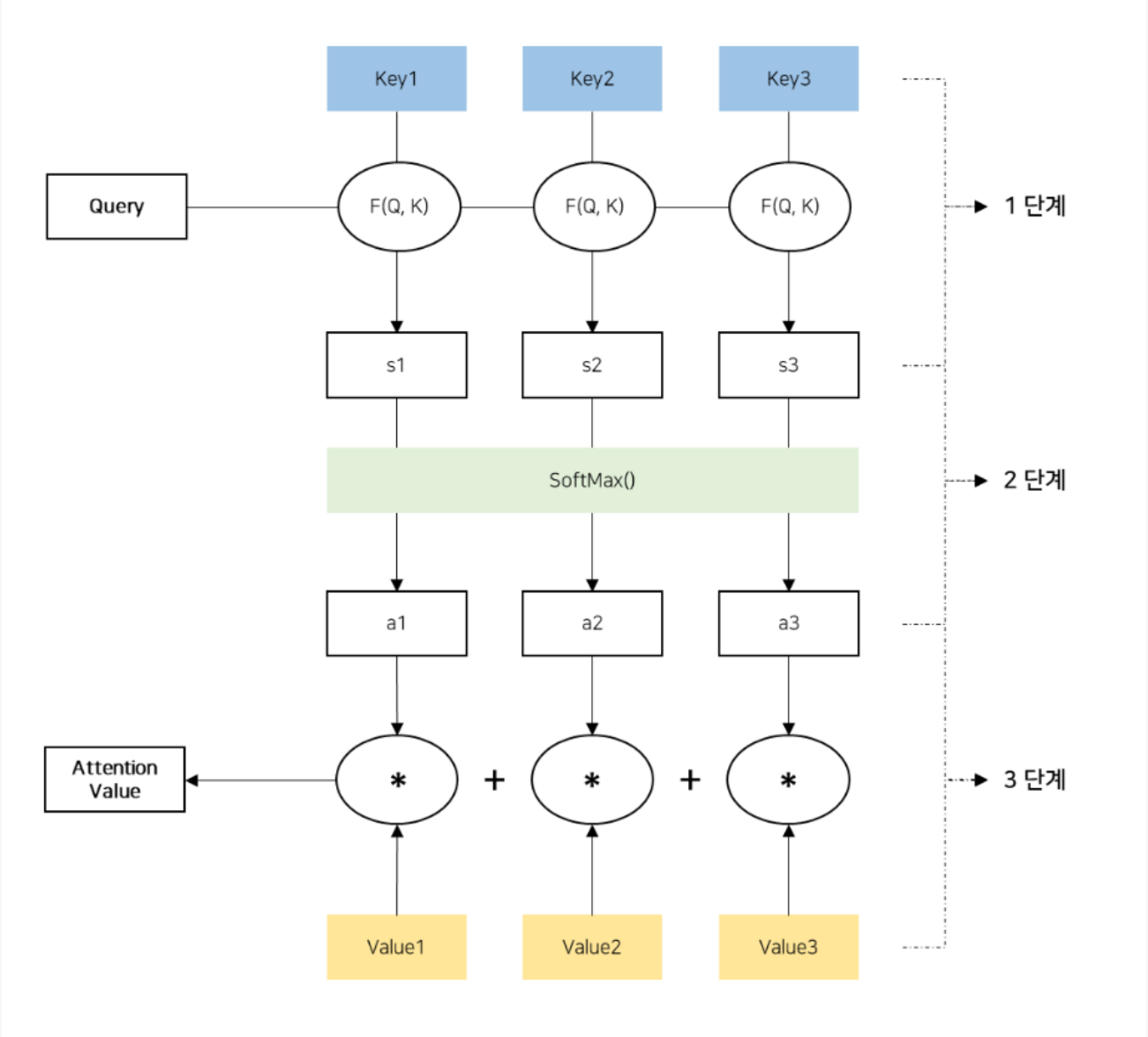
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

d_k는 Key 벡터의 차원 수

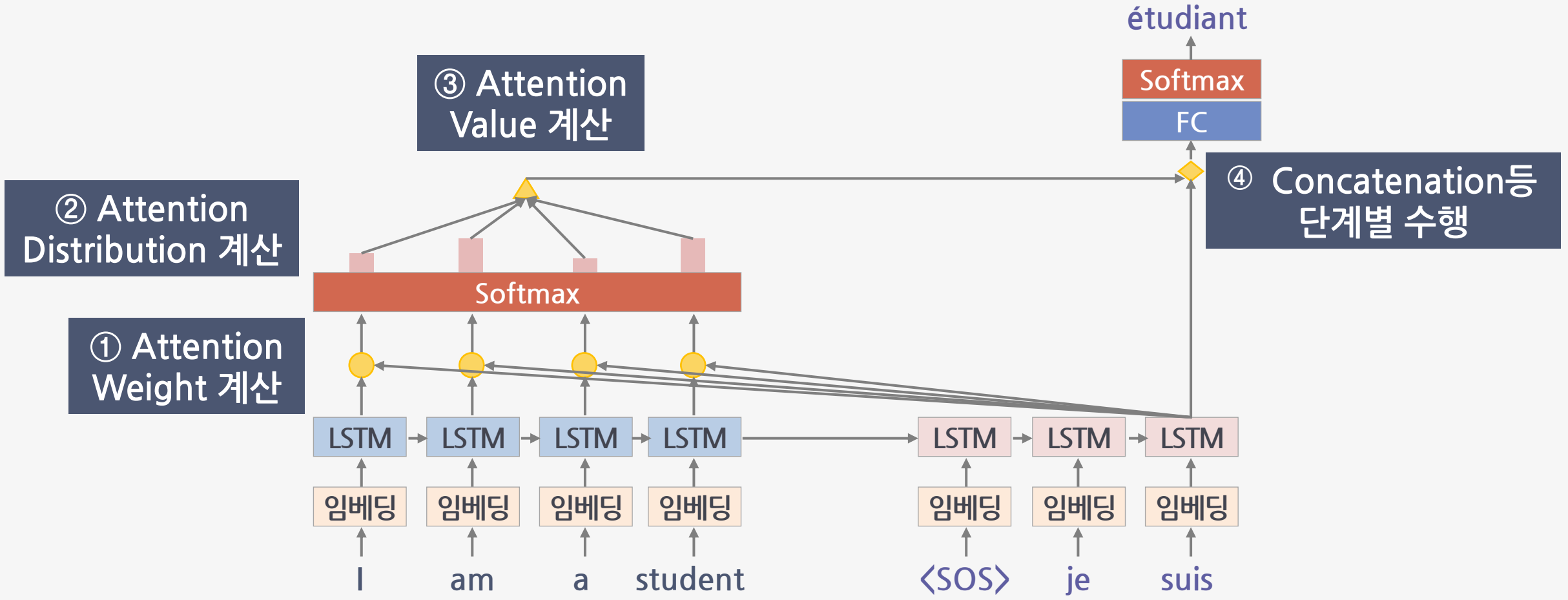
$\frac{QK^T}{\sqrt{d_k}}$는 query 벡터와 key벡터 간의 유사도를 계산

어텐션(Attention)

어텐션(Attention) 적용 과정



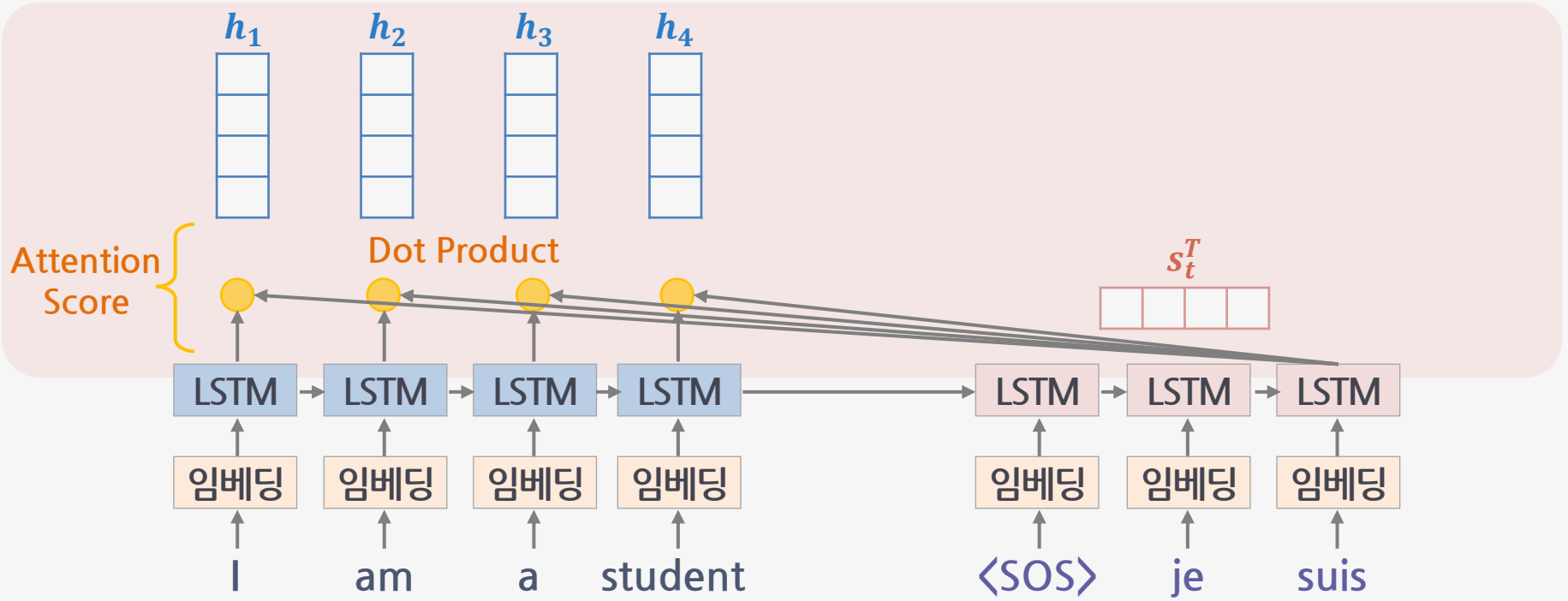
어텐션(Attention) 적용 과정



어텐션(Attention) 적용 과정

Attention Weights 계산

- 디코더의 t시점의 Time step의 Hidden State와 인코더의 모든 Time step의 Hidden State와 행렬 곱셈 연산(Dot Product, 내적) 수행
- Dot Product 연산의 결과로 Attention Weights 계산



어텐션(Attention) 적용 과정

Attention Distribution 계산

- 인코더의 모든 Hidden State의 Attention Weights의 벡터에 소프트맥스(Softmax) 함수 적용

Attention Distribution이란?

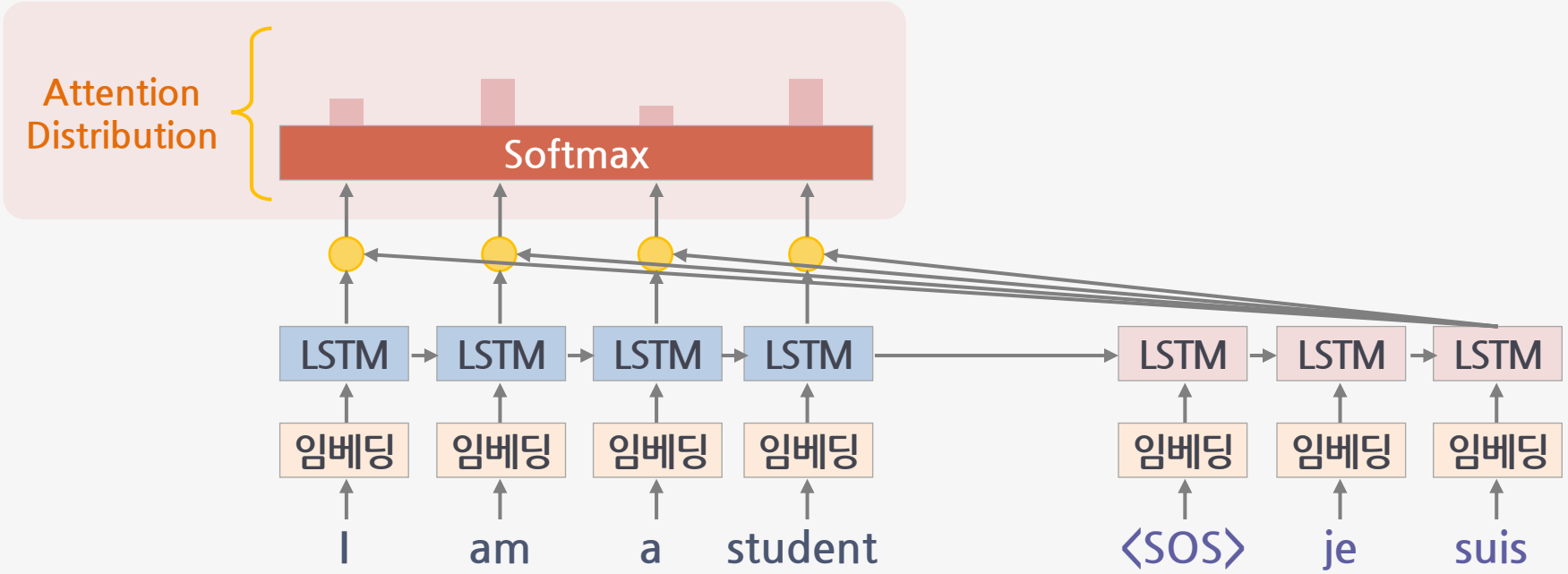
소프트맥스 함수를 적용한 후 각 Attention Weights의
벡터별 합이 1이 되는 확률 분포

각각의 값은 어텐션의 가중치에 해당

어텐션(Attention) 적용 과정

Attention Distribution 계산

- 인코더의 모든 Hidden State의 Attention Weights의 벡터에 소프트맥스(Softmax) 함수 적용



어텐션(Attention) 적용 과정

Attention Value 계산

Attention Value란?

각 인코더의 Hidden State와
Attention Weight의 가중합(Weighted Sum)

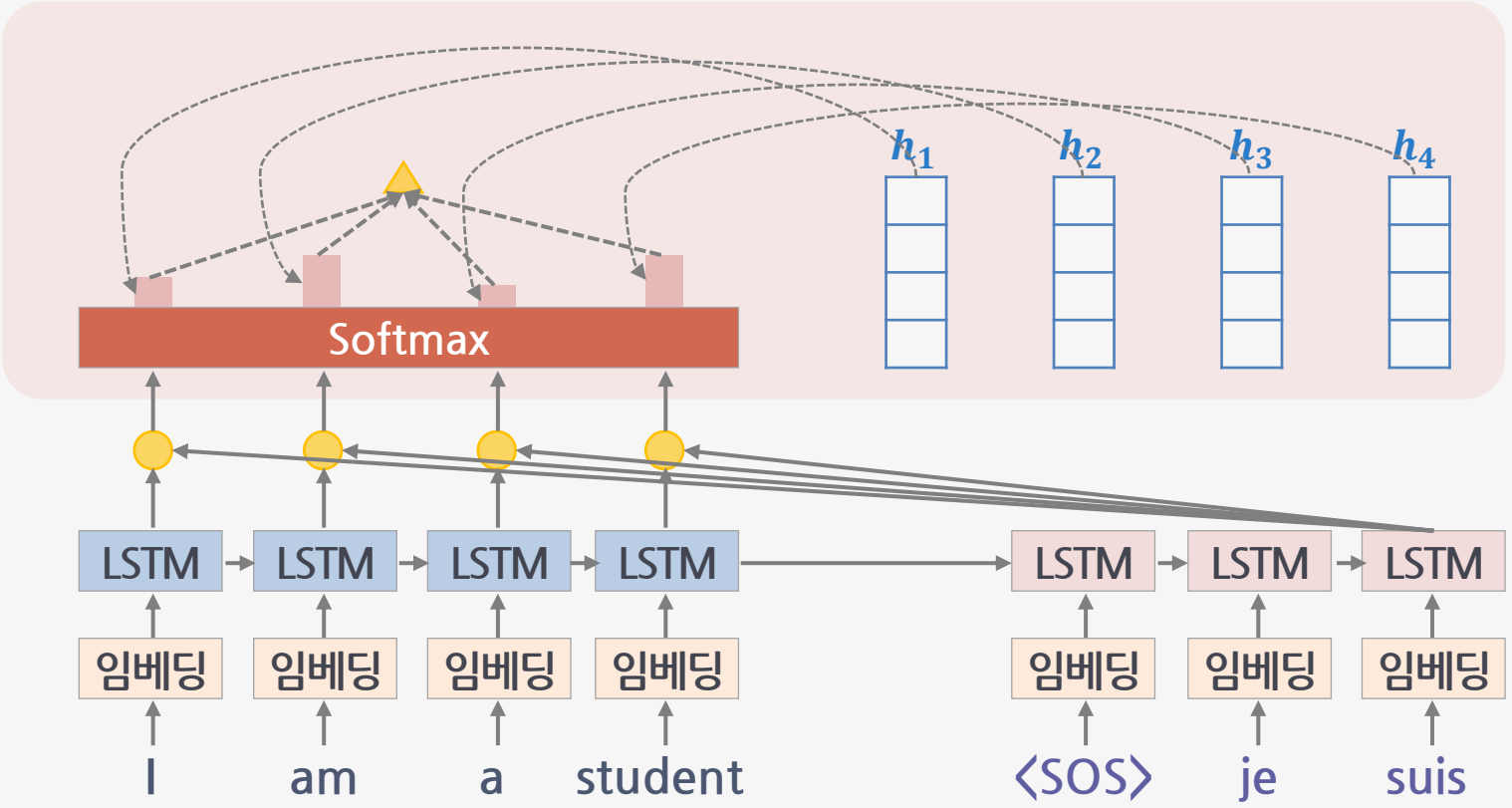
인코더의 Hidden State와 Attention Weight를 곱한 후 모두 합하여 Attention Value 계산

Attention Value는 인코더의 Content를 포함하는 컨텍스트 벡터(Context Vector)

- Seq2Seq에서 인코더의 마지막 Hidden State인 컨텍스트 벡터에 해당

어텐션(Attention) 적용 과정

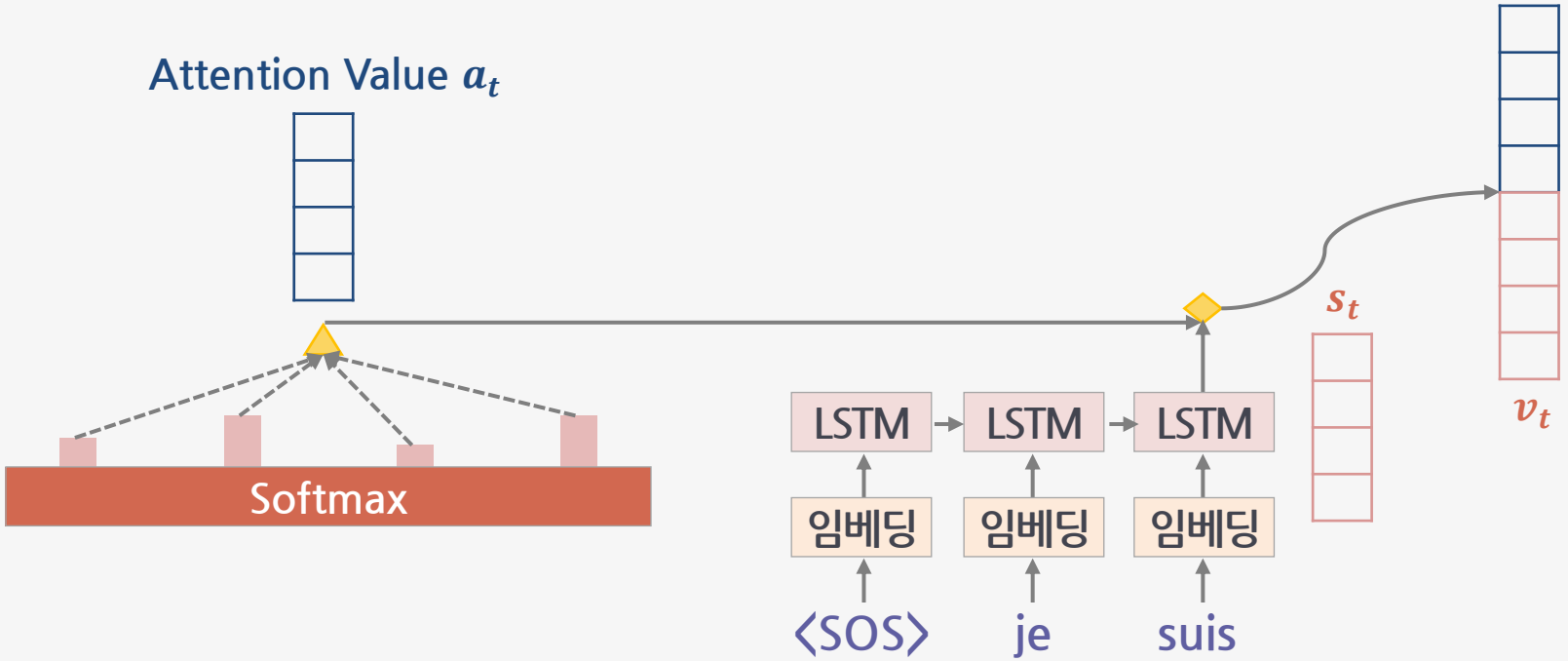
Attention Value 계산



어텐션(Attention) 적용 과정

Concatenation

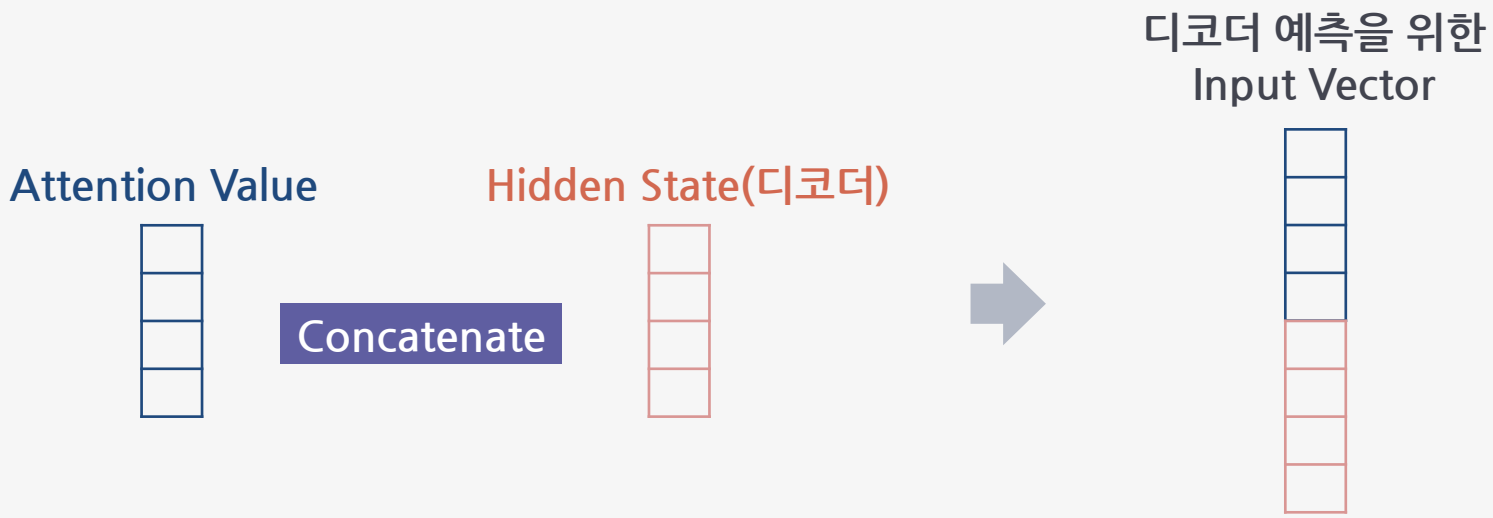
- Attention Value와 디코더의 t시점 Time step의 Hidden State를 연결 (Concatenation)



어텐션(Attention) 적용 과정

Concatenation

- Attention Value와 디코더의 t시점 Time step의 Hidden State를 연결한 벡터는 **디코더에서 예측을 수행**하기 위해 fully Conected Layer의 입력 후 다음 Time step의 입력으로 사용



어텐션(Attention) 적용 과정

Seq2Seq에 어텐션 적용

- Seq2Seq에 어텐션을 적용하기 위해 아래의 단계로 수행

① Attention Weights 계산

» ② Attention Distribution 계산

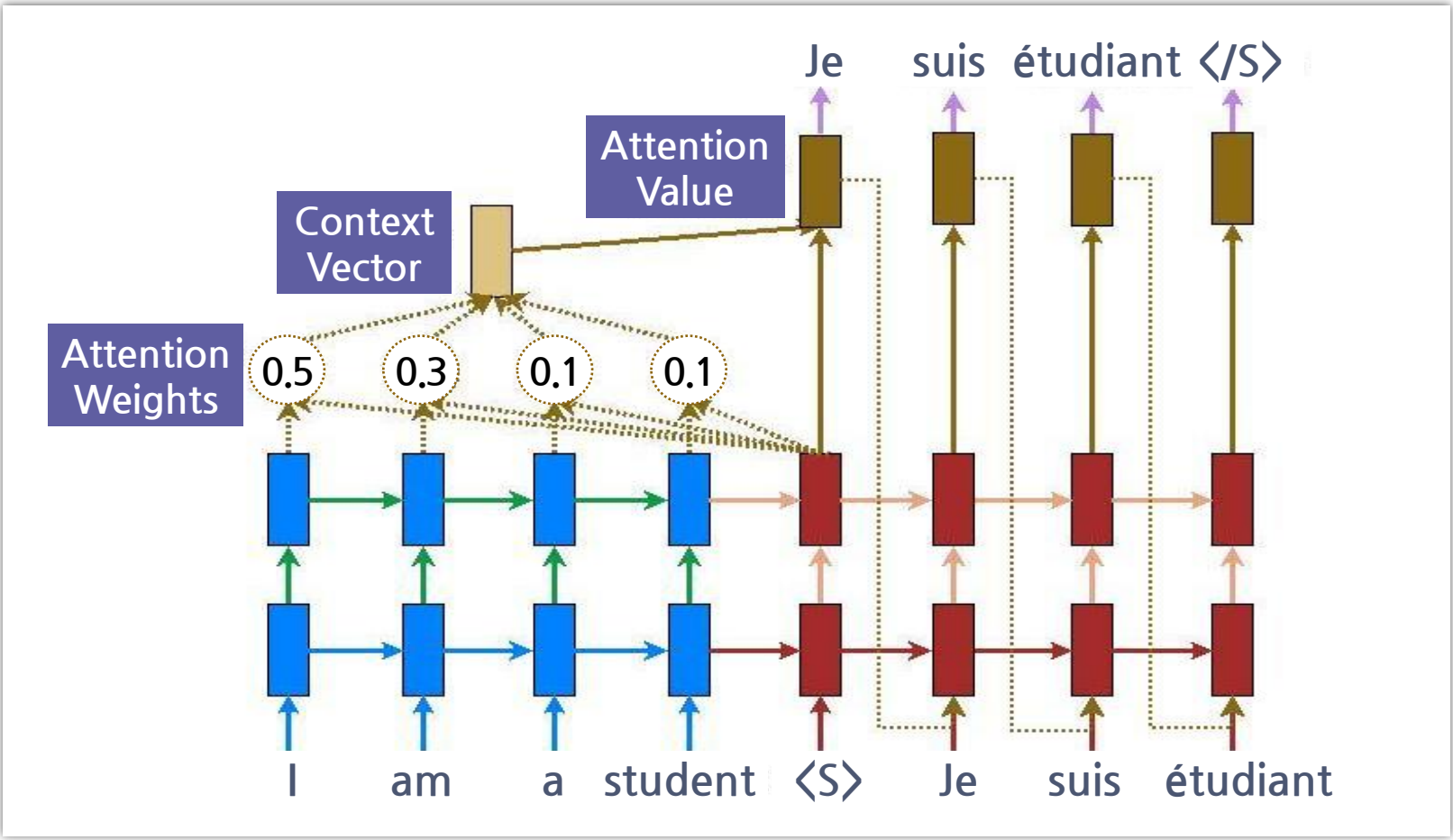
» ③ Attention Value 계산

» ④ Concatenation

어텐션(Attention)

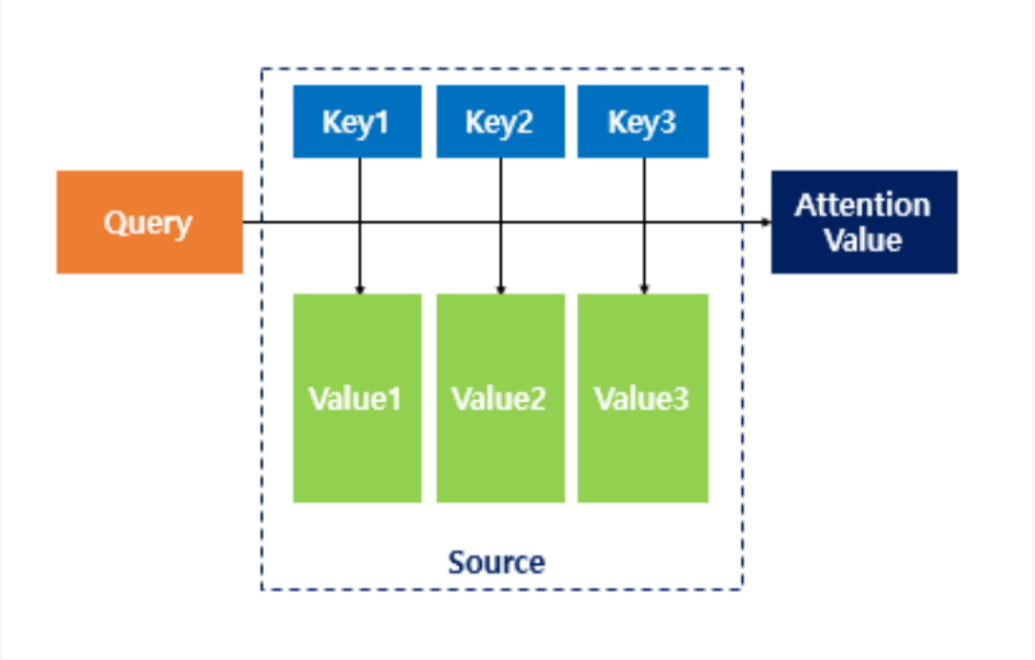
어텐션(Attention) 적용 과정

Seq2Seq에 어텐션 적용



어텐션(Attention) 적용 과정

Seq2Seq에 어텐션 적용

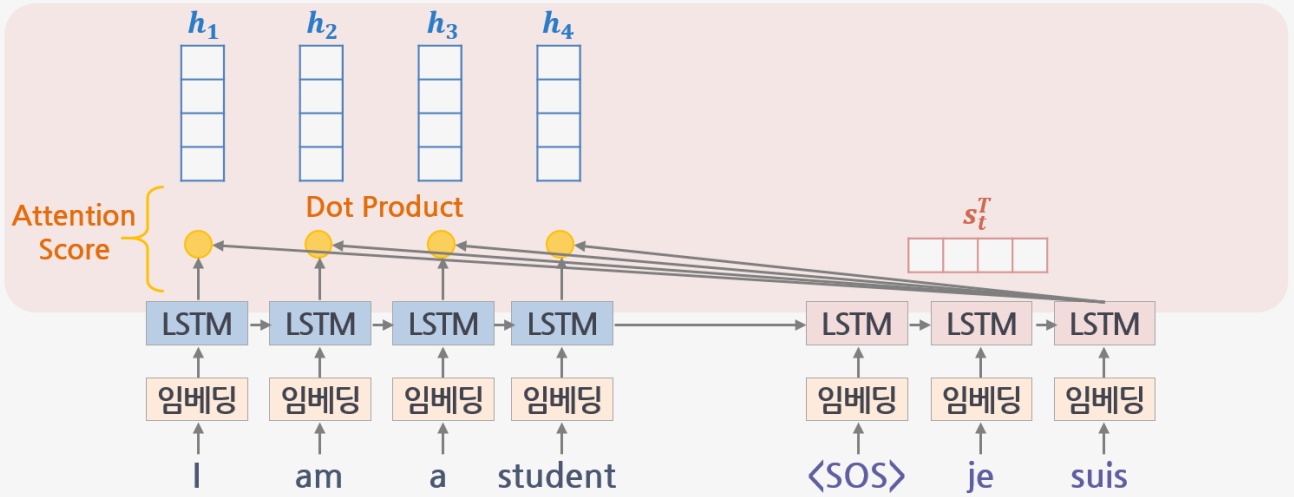



```
self.attention_Wq = nn.Linear(hidden_dim, hidden_dim)
self.attention_Wk = nn.Linear(hidden_dim, hidden_dim)
self.attention_Wv = nn.Linear(hidden_dim, hidden_dim)
self.scale_factor = torch.sqrt(torch.tensor(hidden_dim, dtype=torch.float32))
```

어텐션(Attention)

어텐션(Attention) 적용 과정

Seq2Seq에 어텐션 적용



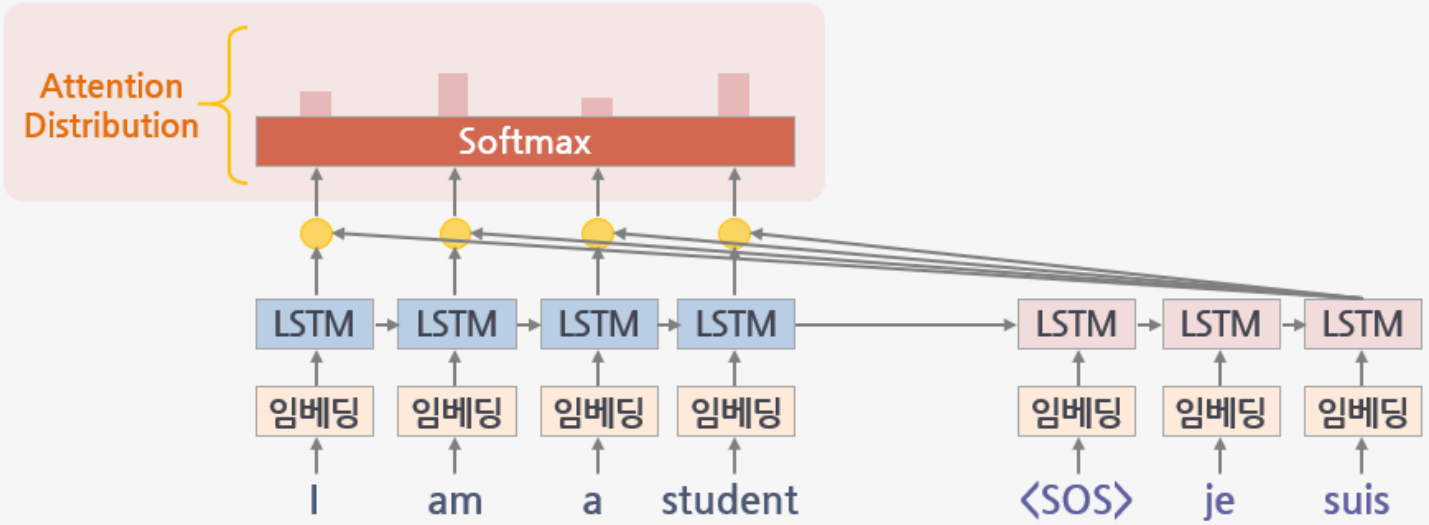
$$\frac{QK^T}{\sqrt{d_k}}$$

```
attention_scores = torch.bmm(Q, K.transpose(1, 2)) / self.scale_factor  # [batch_size, 1, src_len]
```

어텐션(Attention)

어텐션(Attention) 적용 과정

Seq2Seq에 어텐션 적용



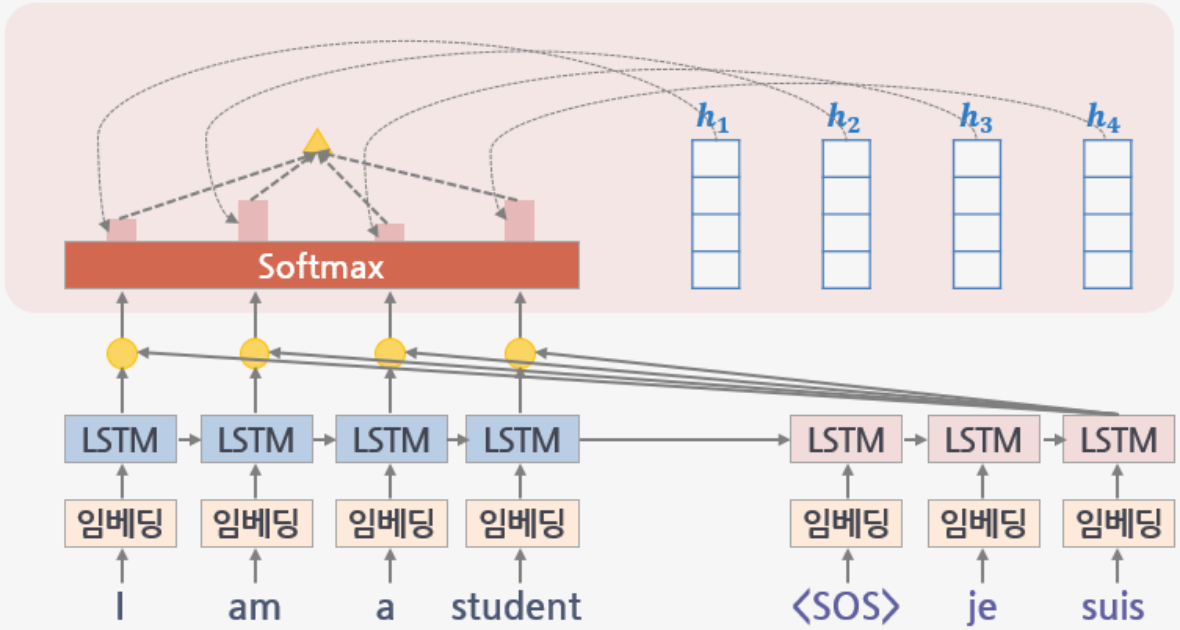
$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

```
attention_weights = torch.softmax(attention_scores, dim=-1)
```

어텐션(Attention)

어텐션(Attention) 적용 과정

Seq2Seq에 어텐션 적용



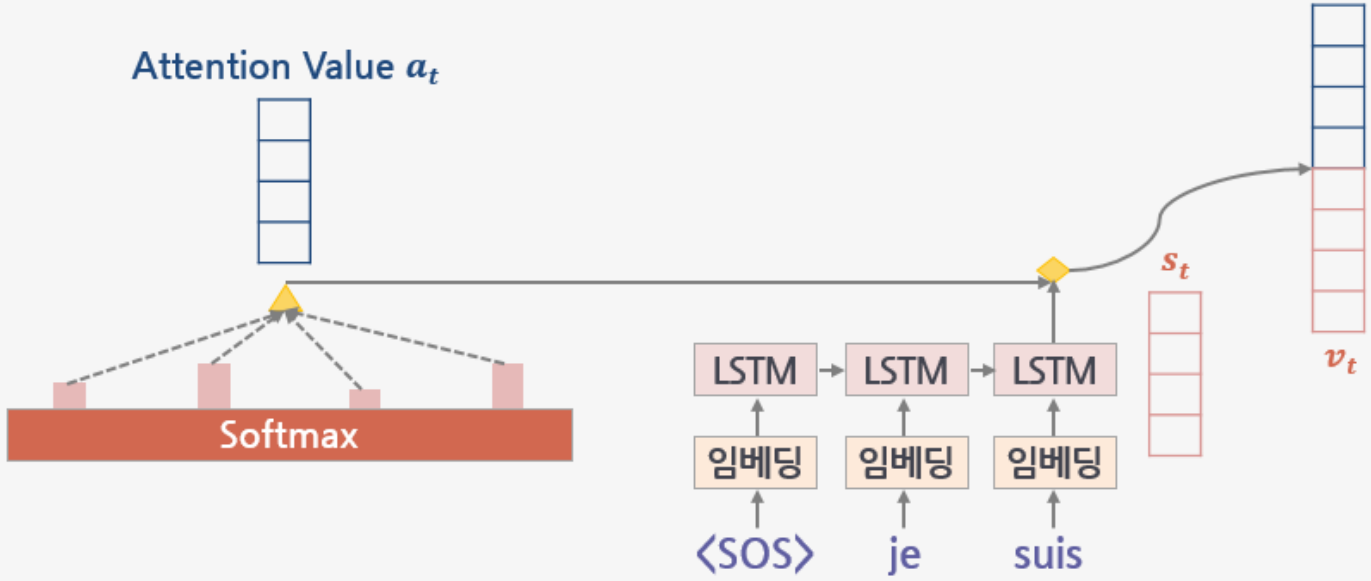
$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

```
context = torch.bmm(attention_weights, V)
```

어텐션(Attention)

어텐션(Attention) 적용 과정

Seq2Seq에 어텐션 적용




```
decoder_input_combined = torch.cat((decoder_embedded, context), dim=2)
```