

Attention Mechanism

Contents

1. Seq2Seq의 한계
2. Attention의 아이디어
3. Attention Function
4. Dot-Product Attention
5. 다양한 종류의 Attention
6. Bahdanau Attention

1. Seq2Seq의 한계

Seq2Seq 모델은

Encoder에서 입력 시퀀스를 Context Vector라는 하나의 고정된 크기의 벡터 표현으로 압축하고,
Decoder에서 이 컨텍스트 벡터를 통해 출력 시퀀스를 만들어냈다.

다만, **RNN에 기반한** seq2seq 모델에는 크게 2가지 문제가 존재한다.

1. 하나의 **고정된 크기의 벡터(Context Vector)**에 모든 정보를 압축하려고 하니까 정보 손실이 발생함
2. RNN의 고질적인 문제인 **vanishing gradient 문제**가 존재함

=> 입력 시퀀스가 길어지면 출력 시퀀스의 정확도가 떨어지는 것을 보정해주기 위해 Attention이 등장함.

2. Attention의 아이디어

Attention의 기본 아이디어

: **Decoder**에서 출력 단어를 예측하는 매 time step마다, **Encoder**에서의 전체 입력 문장을 다시 한번 참고한다.

단, 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라,

해당 time step에서 예측해야할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중(attention)해서 본다.

3-1. Attention Function: Key-Value 자료형

Attention Function에 대해 이해하기 위해서는 Key-Value 자료형에 대한 이해가 있어야함.

Example

파이썬의 딕셔너리(Dict) 자료형은 key와 value라는 2개의 쌍으로 구성되는데, key를 통해서 -> 맵핑된 value를 찾아낼 수 있음

```
# 파이썬의 딕셔너리 자료형을 선언
# 키(Key) : 값(value)의 형식으로 키와 값의 쌍(Pair)을 선언한다.
dict = {"2017" : "Transformer", "2018" : "BERT"}
```

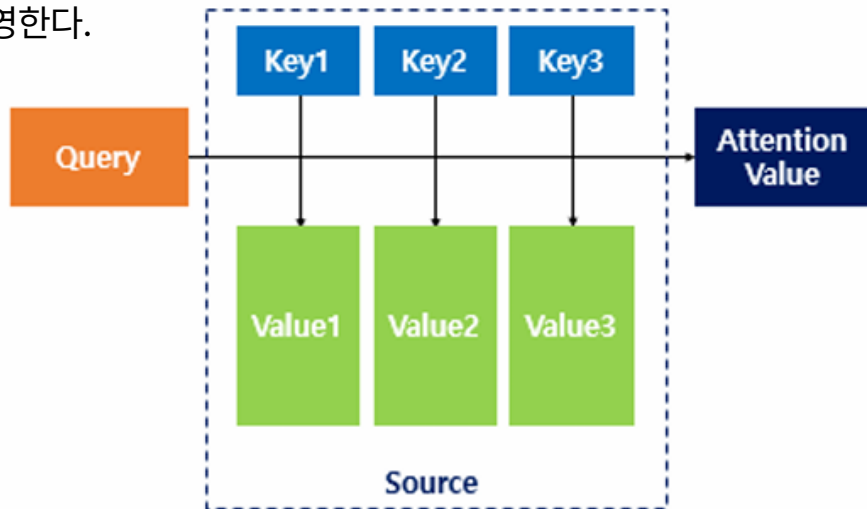
위의 사진에서 2017, 2018은 key에 해당되며, Transformer와 BERT는 각각의 key에 맵핑되는 값에 해당함

3-2. Attention Function

Attention을 함수로 표현하면 주로 아래와 같이 표현됨

Attention(Q, K, V) = Attention Value

1. Attention 함수는 주어진 **Query**(쿼리)에 대해서 모든 **key**와의 유사도를 각각 구한다.
2. 구해낸 유사도를 **key**와 매핑되어 있는 각각의 **value**에 반영한다.
3. 유사도가 반영된 **value**를 모두 더해서 리턴한다.
-> 이 리턴된 **value**를 **Attention Value**라고 한다.



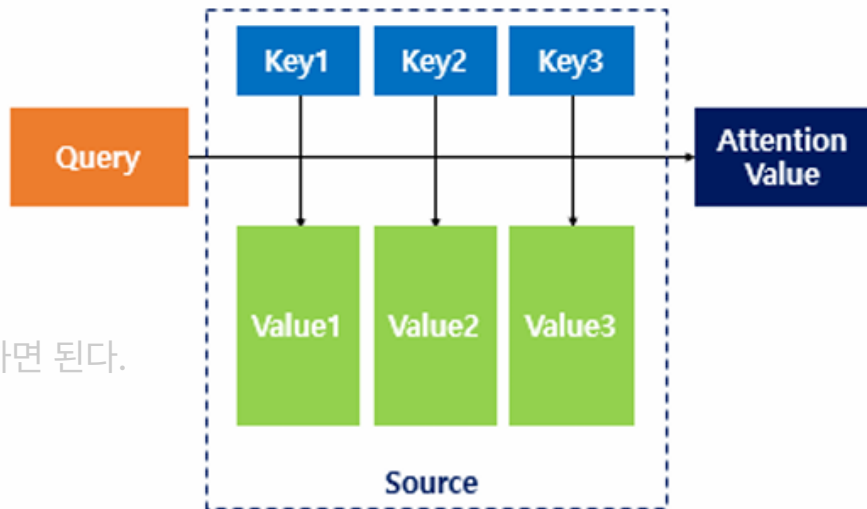
3-2. Attention Function

seq2seq + Attention 모델에서 Q, K, V에 해당하는 각각의 Query, Keys, Values는 다음과 같다.

Q(Query): t 시점의 decoder 셀에서의 hidden state

K(Keys): 모든 시점의 Encoder 셀에서의 hidden state들

V(Values): 모든 시점의 Encoder 셀의 hidden state들



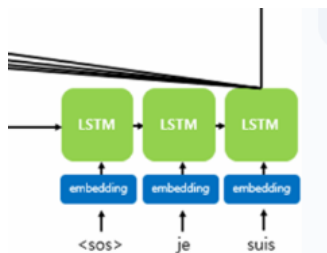
K와 V가 동일한 것을 확인할 수 있다.

동일한 정보(hidden state)를 다른 역할로 사용한다고 생각하면 된다.

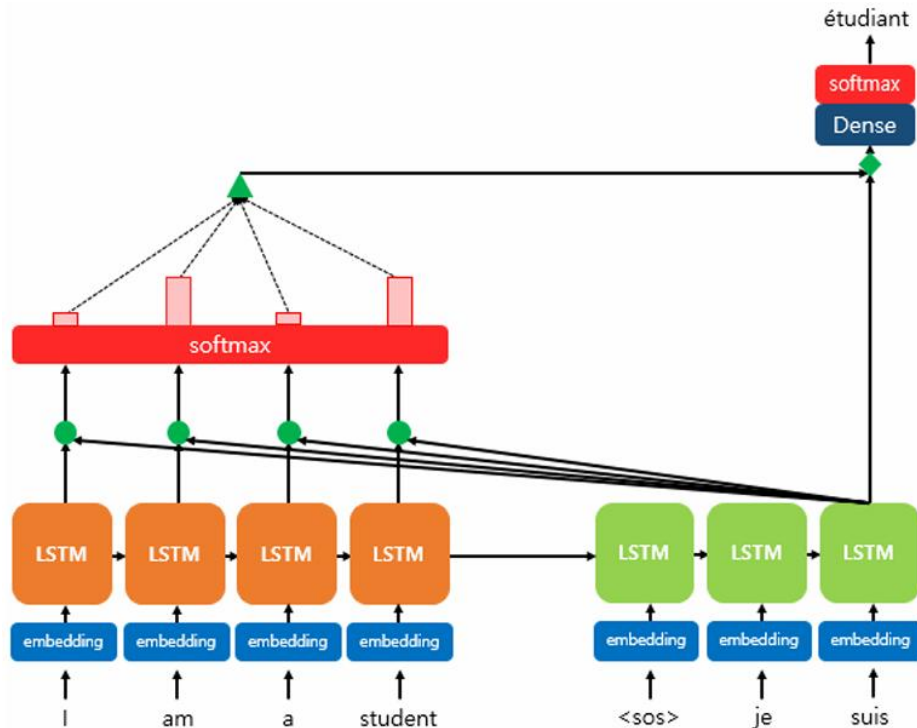
4. Dot-Product Attention: Overview

Attention 중 가장 수식적으로 이해하기 쉽게 수식을 적용한 Dot-Product Attention을 이해해보자.

오른쪽 그림은 **Decoder**의 3번째 LSTM 셀에서 출력 단어를 예측할 때, Attention 메커니즘을 사용하는 모습을 보여줌.



(**Decoder**의 1번째, 2번째 셀은 이미 Attention 메커니즘을 통해 je와 suis를 예측하는 과정을 거쳤다고 가정한다.)



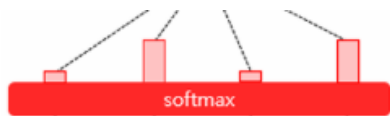
4. Dot-Product Attention: Overview

Decoder의 3번째 LSTM 셀은 출력 단어를 예측하기
위해서 **Encoder**의 **모든 입력 단어들의 정보**를 다시
참고하고자 한다.

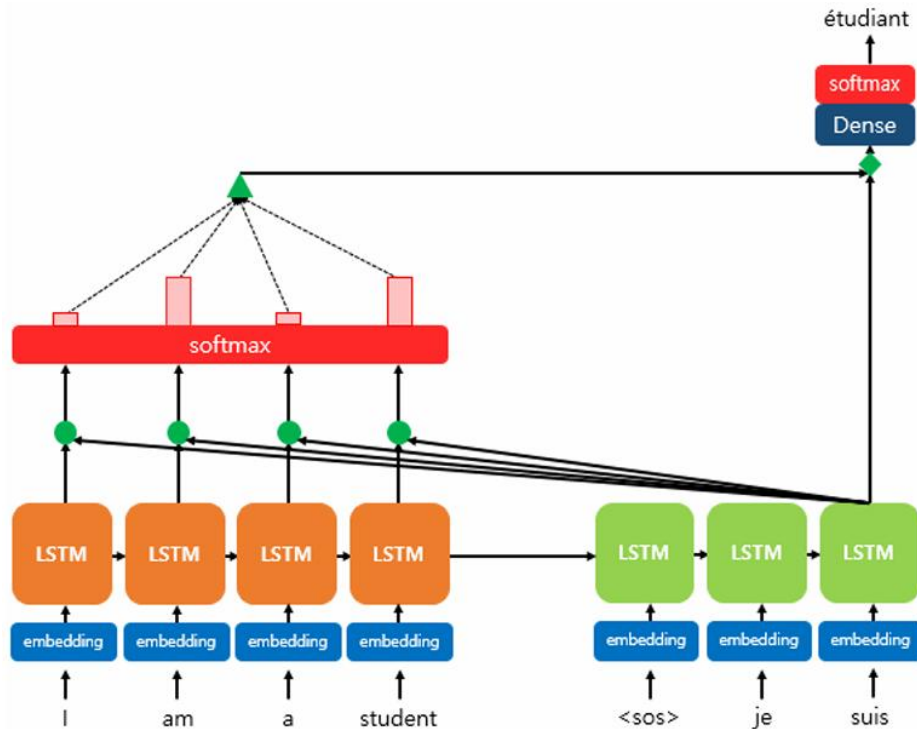
-> 주목할 것은 **Encoder의 softmax 함수**

softmax 함수의 결과값

: 출력 단어를 예측할 때 I, am, a, student 단어 각각
이 얼마나 도움이 되는지의 정도를 수치화한 값

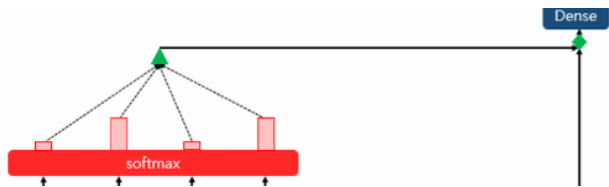


(**빨간 직사각형의 크기**로 softmax함수의 결과값의 크
기를 표현함. 직사각형의 크기가 클수록 도움이 되는
정도의 크기가 큼.)

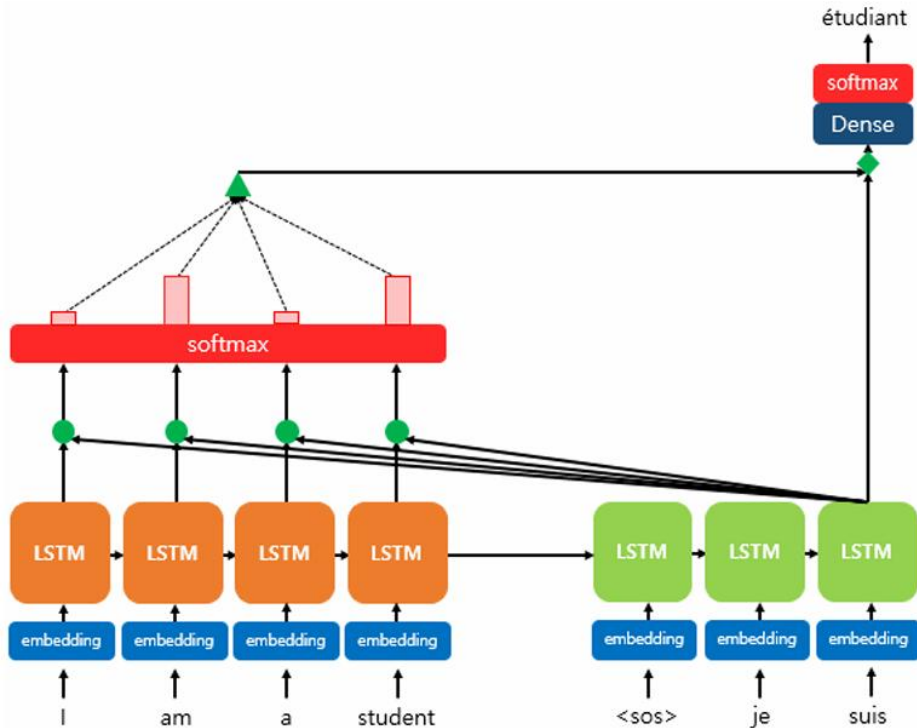


4. Dot-Product Attention: Overview

각 입력 단어가 Decoder의 예측에 도움이 되는 정보가 수치화하여 측정되면, 이를 하나의 정보 (초록색 삼각형)로 담아서 Decoder로 전송함.



=> 결과적으로, Decoder가 출력 단어를 더 정확하게 예측할 확률이 높아짐.



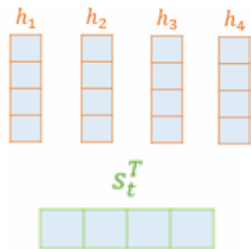
4. Dot-Product Attention: 용어 정의

용어정의

Encoder의 time step을 각각 $1, 2, \dots, N$ 이라고 하자.

Encoder의 hidden state: h_1, h_2, \dots, h_N

현재시점 t 에서의 Decoder의 hidden state: s_t



해당 Example의 가정 : Encoder의 hidden state와 Decoder의 hidden state의 차원이 같다.

Remind

Decoder의 현재 시점 t 의 셀에서 필요한 입력값: 이전 시점 $t-1$ 의 hidden state 및 이전 시점 $t-1$ 에서 나온 출력 단어

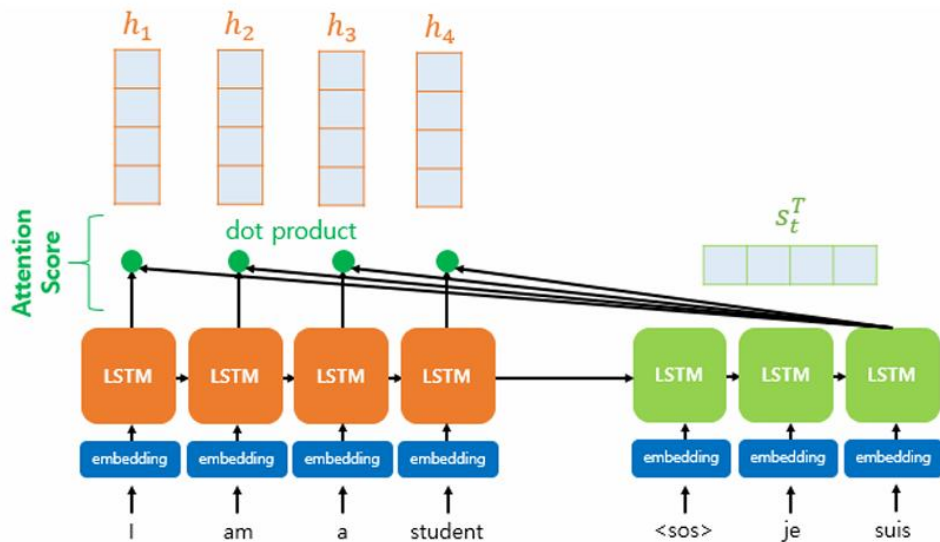
+ Attention 메커니즘에서는 Attention Value라는 새로운 값을 필요로 함.

=> t 번째 단어를 예측하기위한 Attention Value를 a_t 라고 정의할 것 <- **Goal**: Attention Value를 구하는 것

4. Dot-Product Attention: 1) Attention Score를 구한다.

Attention Value를 구하기 위해서는 Attention Score를 구해야함.

Attention Score: 현재 Decoder의 시점 t 에서 단어를 예측하기 위해, **Encoder**의 모든 hidden state 각각이 **Decoder**의 현시점 hidden state인 s_t 와 얼마나 유사한지를 판단하는 score 값

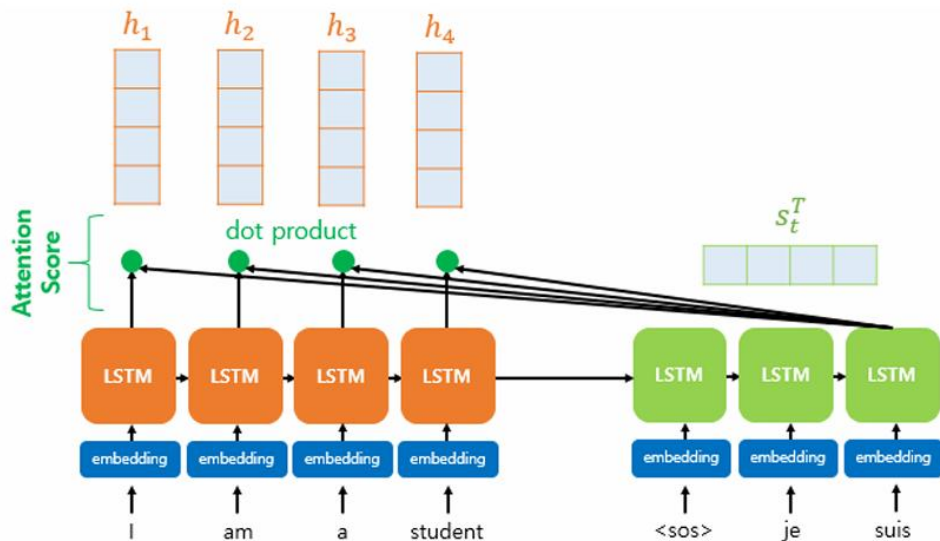


4. Dot-Product Attention: 1) Attention Score를 구한다.

dot-product attention에서는 이 score 값을 구하기 위해서

s_t 를 transpose하고 각 hidden state와 dot product(내적)을 수행한다.

=> 따라서 모든 attention score값은 scalar이다.



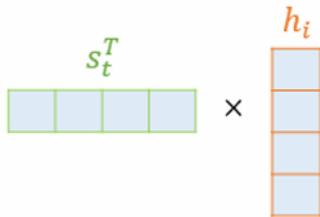
$$s_t^T \times h_i$$

즉, s_t 와 Encoder의 i 번째 hidden state의 attention score 계산 방법은 위와 같다.
(각 단어별로 모두 score값을 가지게 됨)

4. Dot-Product Attention: 1) Attention Score를 구한다.

Attention Score 함수를 정의해보면 아래와 같다.

$$\Rightarrow \text{score}(s_t, h_i) = s_t^T h_i$$



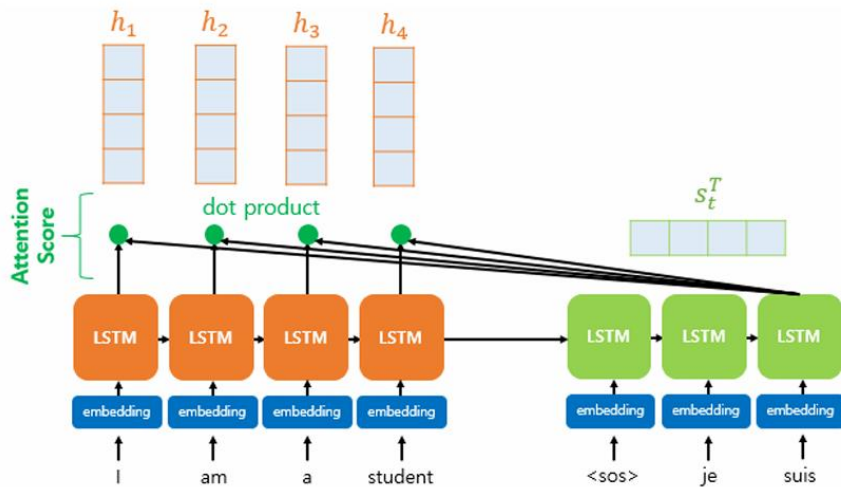
s_t 와 Encoder의 모든 hidden state의 **attention score의 모음값**

$\Rightarrow e^t$ 라고 정의하고 사용할 것이다.

$$\Rightarrow e^t = [s_t^T h_1, \dots, s_t^T h_N]$$



초록색 원 하나하나가 $s_t^T h_i$ 를 나타냄

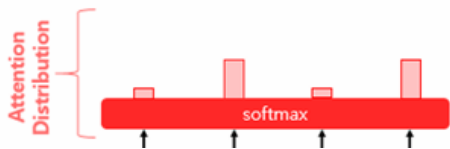


4. Dot-Product Attention

: 2) Softmax 함수를 통해 Attention Distribution을 구한다.

e^t 에 softmax 함수를 적용하면, 모든 값을 더했을 때 1이 되는 확률 분포를 얻을 수 있다.

=> 이 확률 분포를 **Attention Distribution**이라고 하며, 각각의 값(= **attention score**에 softmax를 취한 값)은 **Attention weight**라고 한다.(Attention weight가 클수록 직사각형이 크다.)

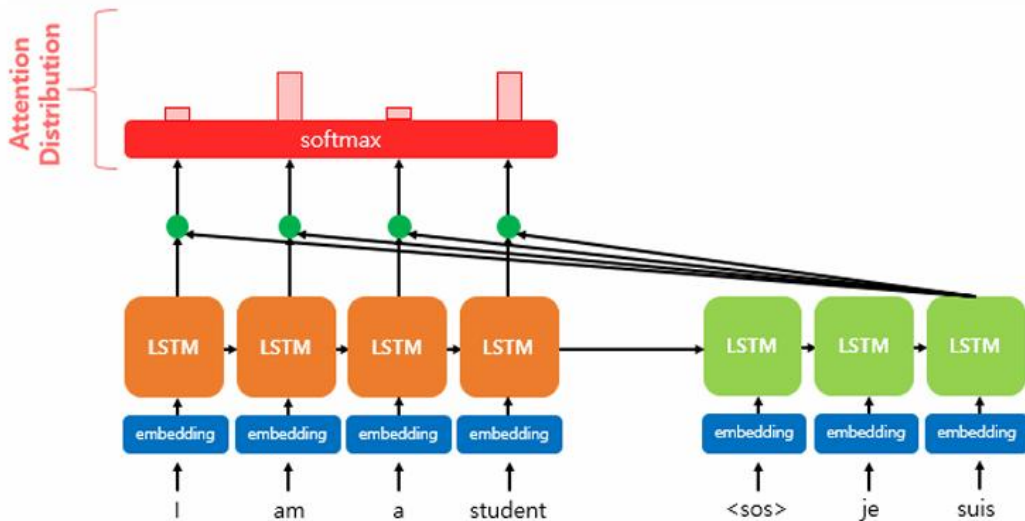


Decoder의 시점 t 에서의 Attention weight의

모음값인 **Attention Distribution**을 α^t 라고 할 때

α^t 를 식으로 정의하면 아래와 같다..

$$\rightarrow \alpha^t = \text{softmax}(e^t)$$

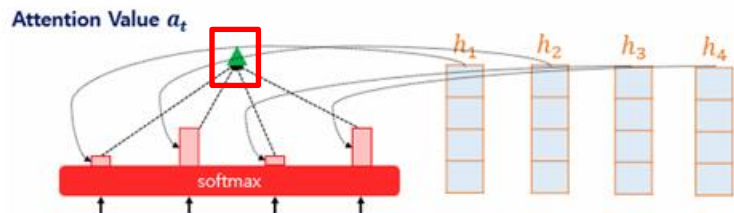


4. Dot-Product Attention

: 3) 각 Encoder의 Attention Weight와 hidden state를
가중합하여 Attention value를 구한다.

Attention의 최종 결과값(=Attention Value)을 얻기 위해서

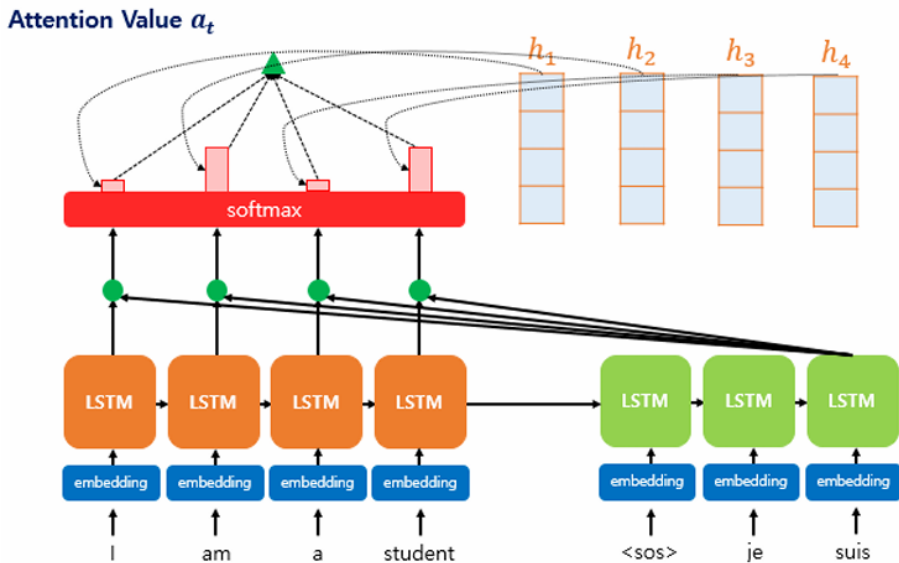
각 Encoder의 hidden state와 attention weight값들을 곱하고, 모두 더한다.(=가중합을 진행함)



Attention의 최종 결과(Attention Function의 출력값)인

Attention Value a_t (초록색 삼각형)는 아래와 같이 나타남

$$\rightarrow a_t = \sum_{i=1}^N \alpha_i^t h_i$$

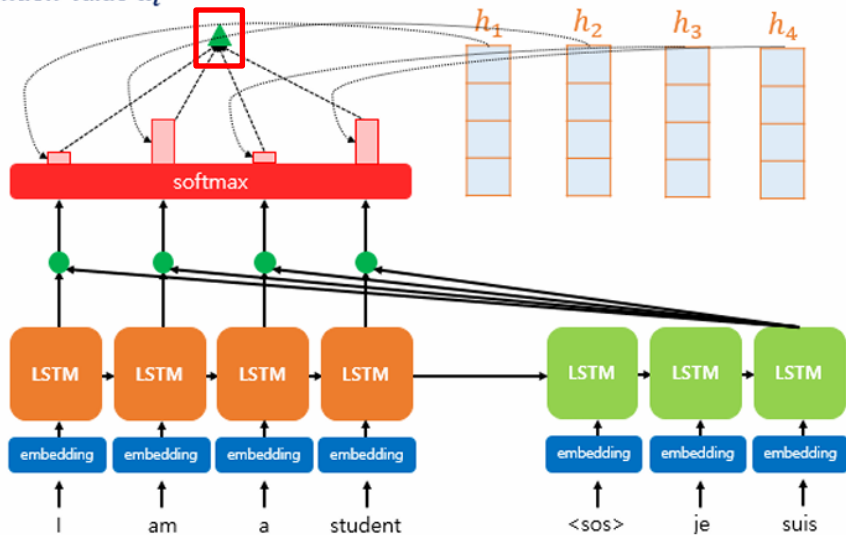


4. Dot-Product Attention

: 3) 각 Encoder의 Attention Weight와 hidden state를
가중합하여 Attention value를 구한다.

이런 Attention Value a_t 는 Encoder의 문맥을 포함하고 있다고 하여, **Context Vector**라고도 불림.
(seq2seq에서 Encoder의 마지막 hidden state를 Context Vector라고 부르는 것과 차이가 있음.)

Attention Value a_t



4. Dot-Product Attention

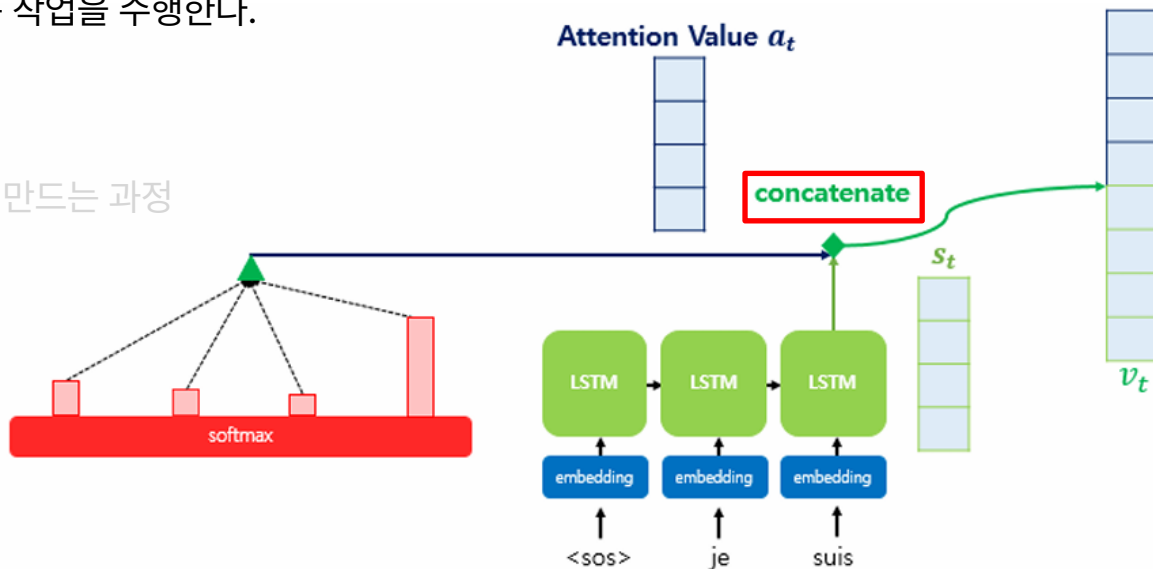
: 4) Attention Value와 Decode의 t 시점의 hidden state를 Concatenate한다.

Attention Value가 구해지면, Attention 메커니즘은

Attention Value a_t 와 현재시점 t 에서의 Decoder의 hidden state s_t 를 Concatenate하여 **1개의 벡터**로 만드는 작업을 수행한다.

-> 이를 v_t 라고 정의하자.

Concatenate: 두 개 이상의 벡터를
순서대로 이어 붙여 하나의 더 긴 벡터로 만드는 과정



4. Dot-Product Attention

: 5) 출력층 연산의 입력이 되는 \tilde{s}_t 를 계산한다.

논문에서는 v_t 를 바로 출력층으로 보내기 전, 신경망 연산을 1번 더 추가했다.

-> v_t 를 가중치행렬(W_c)과 곱한 후, 하이퍼볼릭탄젠트함수(tanh)를 지나도록 해서, 출력층의 입력이 되는 Vector \tilde{s}_t 를 구한다.

$$\tanh \left(W_c \times v_t \right) = \tilde{s}_t$$

4. Dot-Product Attention

: 5) 출력층 연산의 입력이 되는 \tilde{s}_t 를 계산한다.

Recap

즉, 정리하자면 아래와 같이 나타낼 수 있다.

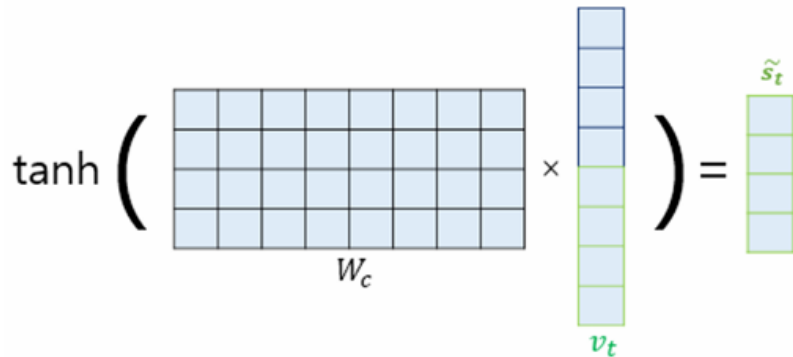
(Attention 메커니즘을 사용하지 않는) Seq2seq의 출력층의 입력: t 시점의 은닉상태인 s_t

Attention 메커니즘의 출력층의 입력: \tilde{s}_t

이를 식으로 나타내면 아래와 같다.

$$\tilde{s}_t = \tanh(W_c[a_t; s_t] + b_c)$$

- W_c : 학습 가능한 가중치 행렬
- b_c : 편향(우측 그림에선 생략됨)
- $[a_t; s_t]$: a_t 와 s_t 간 vector concatenation을 나타냄



5. 다양한 종류의 Attention

Seq2seq + attention 모델에 쓰일 수 있는 다양한 Attention 종류가 있다.

dot-product attention과 다른 attention의 차이는 **Attention Score 함수**(Attention Score를 구하는 방법; Attention Function과 다름)의 차이로, 메커니즘 자체는 거의 유사하다.

-> dot-product attention은 **attention socre**를 구하는 방법이 **dot-product**(내적)이었기 때문에 이렇게 이름지어졌다.

5. 다양한 종류의 Attention

현재 제시된 여러 종류의 Attention score 함수는 위와 같다.

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

Dot-product attention은 Luong(루옹) attention이라고도 한다.

Concat이라는 이름의 attention은 Bahdanau(바다나우) attention이라고 부른다.

s_t : Query(t 시점의 decoder 셀에서의 hidden state)

h_i : Keys(모든 시점의 Encoder 셀에서의 hidden state들)

W_a, W_b : 학습가능한 가중치 행렬

6-1. Bahdanau(바다나우) Attention Function

Dot-product attention보다 조금 더 복잡하게 설계된 Bahdanau Attention 메커니즘을 이해해보자.

Attention 메커니즘을 함수 **Attention()**으로 정의하였을 때,
Bahdanau Attention Function의 입, 출력은 다음과 같이 정의할 수 있다.

Attention(Q, K, V) = Attention Value

Q(Query): **t-1**시점의 decoder 셀에서의 hidden state

K(Keys): 모든 시점의 Encoder 셀에서의 hidden state들

V(Values): 모든 시점의 Encoder 셀의 hidden state들

Attention Function의 Query가 Decoder 셀의 t 시점의 hidden state가 아니라
t-1 시점의 hidden state임에 주목

6-2. Bahdanau Attention(바다나우 어텐션) : 1) Attention Score를 구한다.

Bahdanau attention의 연산 순서를 살펴보자.

용어정의

Encoder의 time step을 각각 $1, 2, \dots, N$ 이라고 하자.

Encoder의 hidden state: h_1, h_2, \dots, h_N

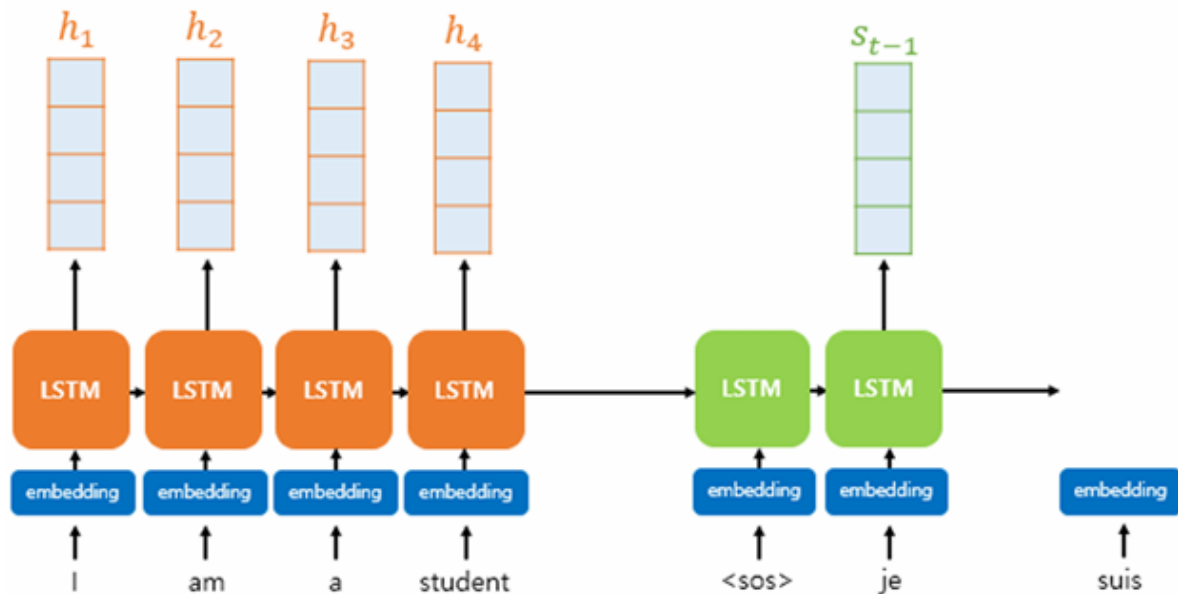
현재시점 t 에서의 Decoder의 hidden state: s_t

해당 Example의 가정 : Encoder의 hidden state와 Decoder의 hidden state의 차원이 같다.

6-2. Bahdanau Attention(바다나우 어텐션)

: 1) Attention Score를 구한다.

Dot-product attention에서는 Query로 Decoder의 t 시점의 hidden state를 사용한 것과 달리, Bahdanau Attention에서는 Query로 Decoder의 $t-1$ 시점의 hidden state인 s_{t-1} 를 이용한다.



6-2. Bahdanau Attention(바다나우 어텐션)

: 1) Attention Score를 구한다.

s_{t-1} 과 encoder의 i 번째 hidden state의 attention score 계산 방법은 아래와 같다.

$$score(s_{t-1}, h_i) = W_a^T \tanh(W_b s_{t-1} + W_c h_i)$$

- W_a, W_b, W_c : 학습가능한 가중치 행렬

s_{t-1} 과 h_1, h_2, h_3, h_4 의 attention score를 각각 구해야하므로 병렬 연산을 위해 h_1, h_2, h_3, h_4 를 -> 하나의 행렬 H 로 두면 수식은 다음과 같이 변경된다.

$$score(s_{t-1}, h_i) = W_a^T \tanh(W_b s_{t-1} + W_c H)$$

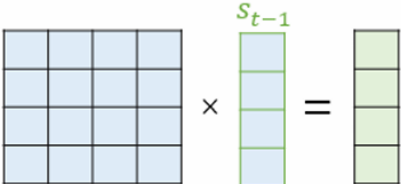
6-2. Bahdanau Attention(바다나우 어텐션)

: 1) Attention Score를 구한다.

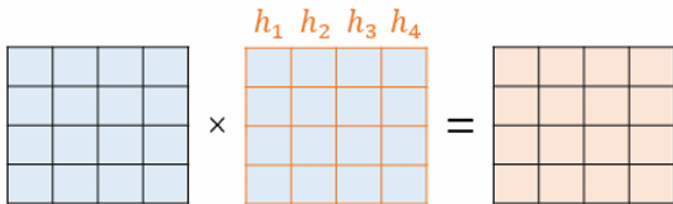
그림을 통해 이해해보자.

$W_b s_{t-1}$, $W_c H$ 를 각각 구하면 아래와 같다.

$W_b s_{t-1}$:



$W_c H$:



6-2. Bahdanau Attention(바다나우 어텐션)

: 1) Attention Score를 구한다.

$$\tanh(W_b s_{t-1} + W_c H) : \tanh \left(\begin{matrix} s_{t-1} \\ \vdots \end{matrix} + \begin{matrix} h_1 & h_2 & h_3 & h_4 \\ \vdots & \vdots & \vdots & \vdots \end{matrix} \right) = \begin{matrix} \vdots & \vdots & \vdots & \vdots \end{matrix}$$

$$score(s_{t-1}, h_i) = W_a^T \tanh(W_b s_{t-1} + W_c H) : \begin{matrix} \vdots & \vdots & \vdots & \vdots \end{matrix} \times \begin{matrix} \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{matrix} = \begin{matrix} \text{Attention Score} \\ \vdots & \vdots & \vdots & \vdots \end{matrix}$$

$h_1 \quad h_2 \quad h_3 \quad h_4$

이렇게 s_{t-1} 과 h_1, h_2, h_3, h_4 의 유사도가 기록된 attention score 벡터 e^t 를 구할 수 있다.

$$e^t = score(s_{t-1}, h_i) = W_a^T \tanh(W_b s_{t-1} + W_c H)$$

6-2. Bahdanau Attention(바다나우 어텐션)

: 2) softmax 함수를 통해 Attention Distribution을 구한다.

Dot-product Attention과 마찬가지로,

e^t 에 softmax 함수를 적용하면 모든 값을 합하여 1이 되는 확률분포(=Attention Distribution)을 얻을 수 있다.

Attention Distribution의 각각의 값(Attention Score에 softmax함수를 적용한 값)은 **Attention Weight**라고 한다.

$$\text{softmax} \left(\begin{array}{c} \text{Attention Score} \\ \begin{array}{|c|c|c|c|} \hline \text{ } & \text{ } & \text{ } & \text{ } \\ \hline \end{array} \\ h_1 \quad h_2 \quad h_3 \quad h_4 \end{array} \right) = \begin{array}{c} \text{Attention} \\ \text{Distribution} \\ \begin{array}{|c|c|c|c|} \hline \text{ } & \text{ } & \text{ } & \text{ } \\ \hline \end{array} \end{array}$$

6-2. Bahdanau Attention(바다나우 어텐션)

: 3) 각 Encoder의 Attention 가중치와 hidden state를 가중합하여 Attention Value를 구한다.

이런 Attention Value a_t 는 Encoder의 문맥을 포함하고 있다고 하여, **Context Vector**라고도 불림.
(seq2seq에서 Encoder의 마지막 hidden state를 Context Vector라고 부르는 것과 차이가 있음.)



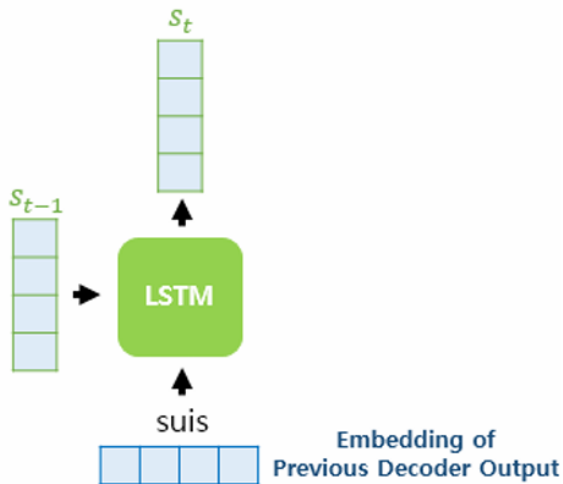
6-2. Bahdanau Attention(바다나우 어텐션)

: 4) Context Vector로부터 s_t 를 구한다.

Recap

기존의 LSTM이 s_t 를 구할 때는

이전 시점 $t-1$ 의 셀로부터 전달받은 hidden state s_{t-1} 과 현재시점 입력 x_t 를 가지고 연산하였다.



위의 LSTM은 seq2seq의 Decoder이며 현재 시점의 입력 x_t (suis)는 임베딩된 단어 벡터이다.

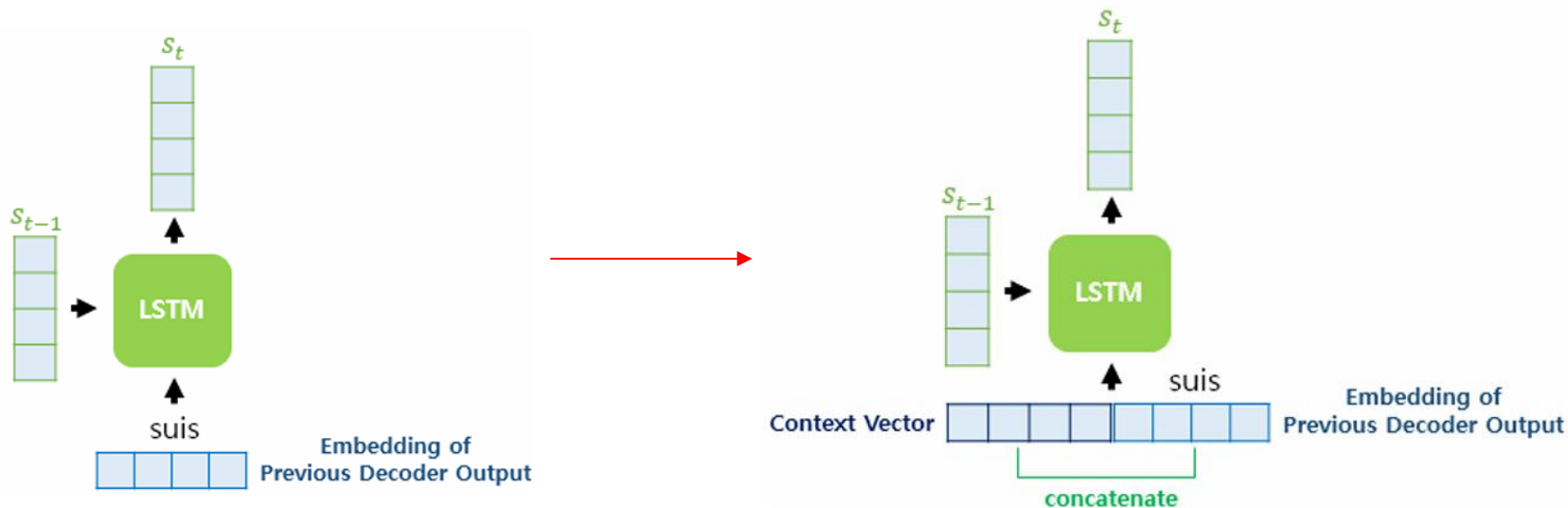
6-2. Bahdanau Attention(바다나우 어텐션)

: 4) Context Vector로부터 s_t 를 구한다.

Bahdanau attention 메커니즘에서는

Context Vector와 현재 시점 t 의 입력인 단어의 Embedding Vector를 concatenate하고,
현재 시점 t 의 새로운 입력으로 사용한다.

이후, 이전 시점의 셀로부터 전달받은 hidden state s_{t-1} 과 현재 시점의 새로운 입력으로부터 s_t 를 구한다.



6-2. Bahdanau Attention(바다나우 어텐션) : 4) Context Vector로부터 s_t 를 구한다.

이후에는 Attention 메커니즘을 사용하지 않는 경우와 동일하게 s_t 가 출력층으로 전달되어 현재 시점 t의 예측값을 구하게 된다.



Ref

모든 Summary 자료는 직접 작성하였으며,
사용된 설명 및 사진의 원 출처는 [딤러닝 파이토치 교과서](#)의 공개 내용임을 밝힙니다.