

# Attention Model



- 딥러닝 초창기의 기계번역 기술의 주요 방식은 Sequence 방식
  - 대표 모델: seq2seq
    - 데이터를 토큰으로 나눠서 순차적으로 입력 데이터를 준 다음 순차적으로 출력을 뽑아내는 방식 → Encoder-Decoder 방식
    - 인코더에서 입력 시퀀스를 컨텍스트 벡터라는 하나의 고정된 크기의 벡터 표현으로 압축하고, 디코더는 이 컨텍스트 벡터를 통해서 출력 시퀀스를 만들어 내는 모델
    - 그러나 문장이 길어지면(30~40 단어 이상) 성능이 떨어짐
  - 인간의 경우도 기본적으로 순차 번역을 지향함
    - 인간도 문장이 길어지면 이해도가 떨어짐

- 문제점을 정리해보면
  - 하나의 고정된 크기의 벡터에 모든 정보를 압축하려고 하다 보니
    - 정보 손실 발생
  - RNN 모델을 베이스로 하기때문에
    - RNN의 고질적인 문제인 기울기 소실 문제가 여전히 존재함
  - 이런 이유로 입력 문장이 길어지면 번역 품질이 떨어지는 현상 발생
- 문제의 해결, 개선을 위한 방법으로 Attention 모델(기법) 제안

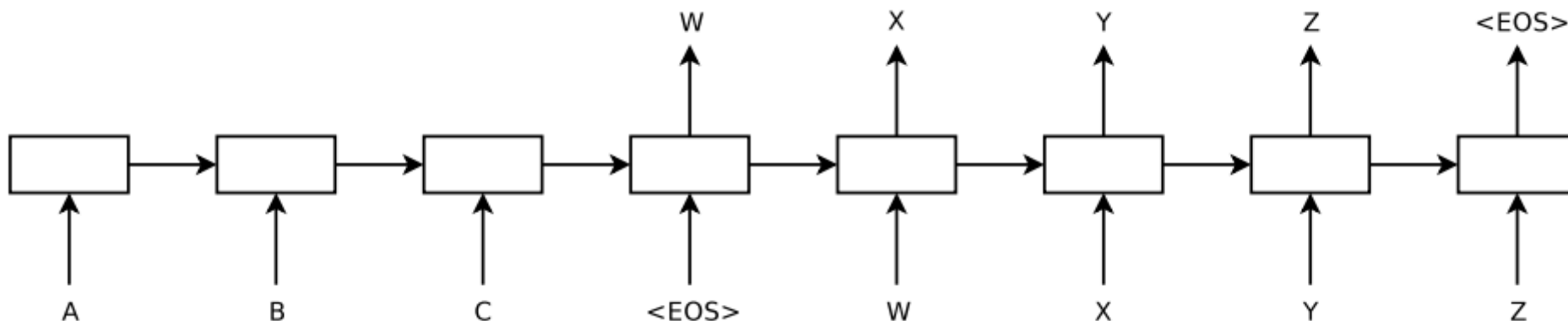
- 사람은 번역을 할 때, 모든 문장을 듣지만 각 단어를 번역할 때마다 모든 문장의 정보를 이용하지는 않는다.
- Encoder-Decoder 방식에서는
  - 다음 단어를 번역할 때마다  $C$  라는 문맥 벡터를 전달하는데,
  - 이 **고정된 길이의 벡터에 그 동안 본 모든 단어에 대한 정보가 축약되어 있음**
    - 문장이 길어지면 효율 저하
    - **이게 문제!!!**

- 그렇다면...
    - 고정된 길이의  $C$  벡터에 모든 정보를 축약하지 말고
    - 매 Step마다 필요한 정보를 새로 만들자! → 해결/개선안 제시
- Attention 모델



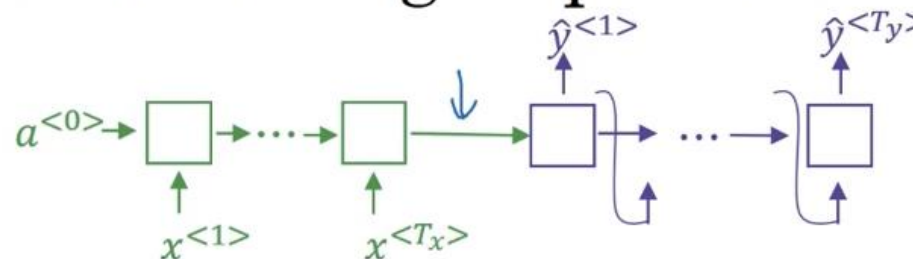
- 정리하면, 어텐션의 기본 아이디어는
  - 디코더에서 출력 단어를 예측하는 때 시점(time step)마다
  - 인코더에서의 전체 입력 문장을 다시 한 번 참고하자!!
- 단, 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라
- 해당 시점에서 예측해야 할 단어와 연관이 있는 입력 단어 부분을
- 좀 더 집중(attention)해서 참고하자!!

- RNN 모델에 대한 Attention 기법 적용
  - 기존의 encode는 word-for-word translation.
  - 즉, 단어를 하나 번역하면 그 단어를 넘기는 방식



- 번역하고자 하는 내용과 문제점

## The problem of long sequences

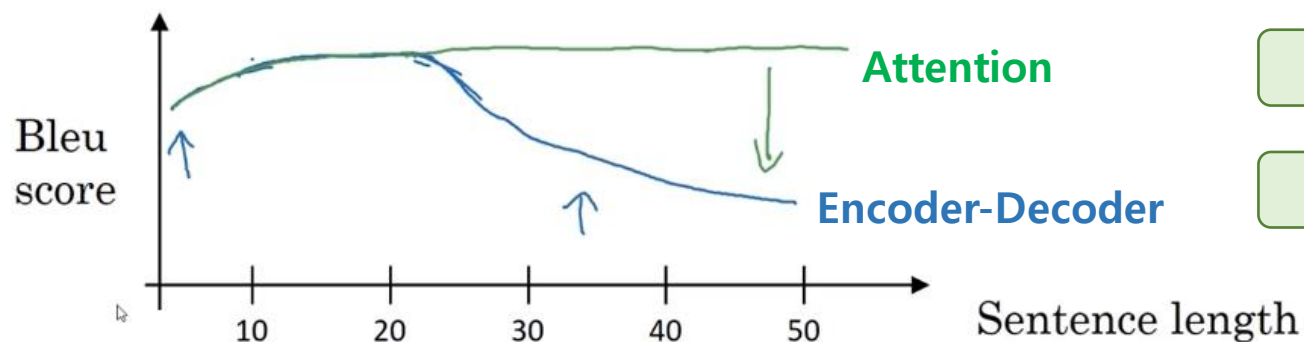


BLEU Score (?)  
( Bilingual Evaluation Understudy Score )

자동 번역 및 동일한 소스 문장에 대해  
사람이 만든 하나 이상의 참조 번역 간의 차이를 측정한 것

Jane s'est rendue en Afrique en septembre dernier, a apprécié la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y aller aussi.

Jane went to Africa last September, and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was, and is tempting me to go too.



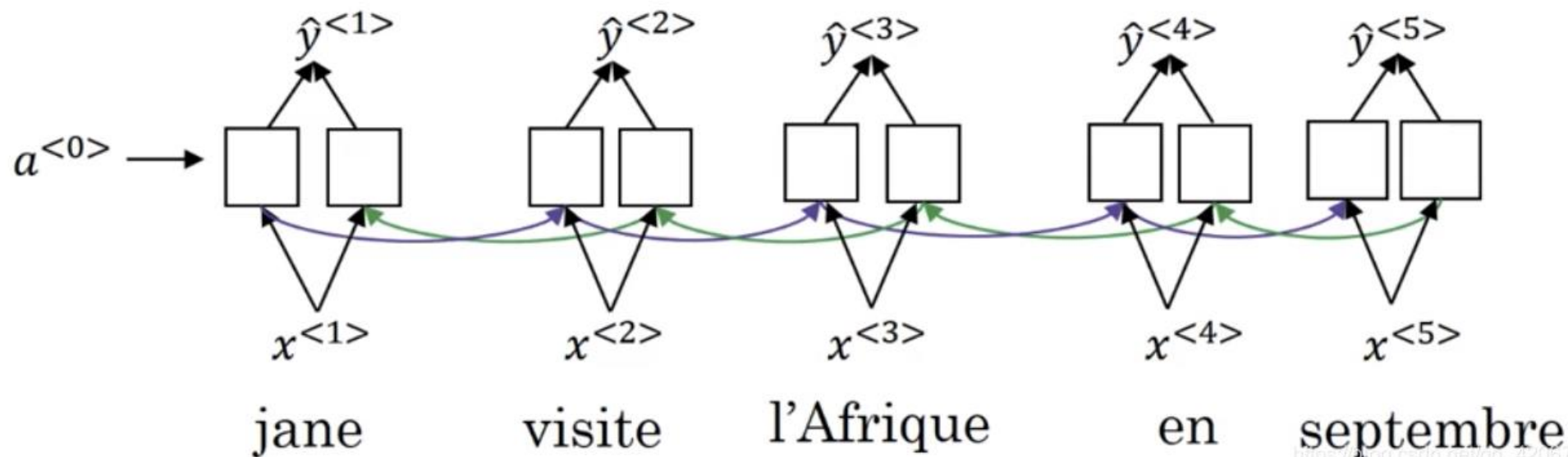
시퀀스가 너무 길어서 손대기가 어렵다

문장이 길어지면서 정확도가 급격히 떨어진다

Andrew Ng



- RNN 모델에서의 번역

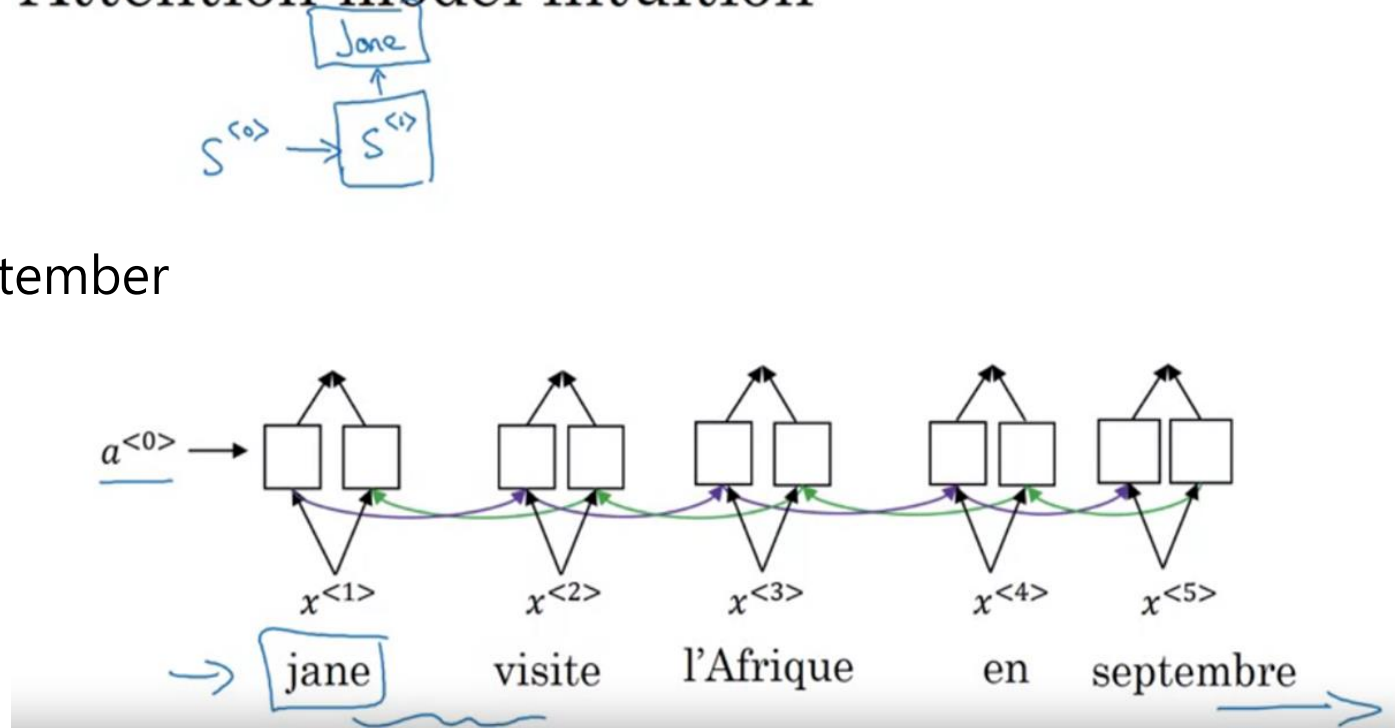


[https://blog.csdn.net/yqq\\_42087550](https://blog.csdn.net/yqq_42087550)

- RNN 모델에서의 번역

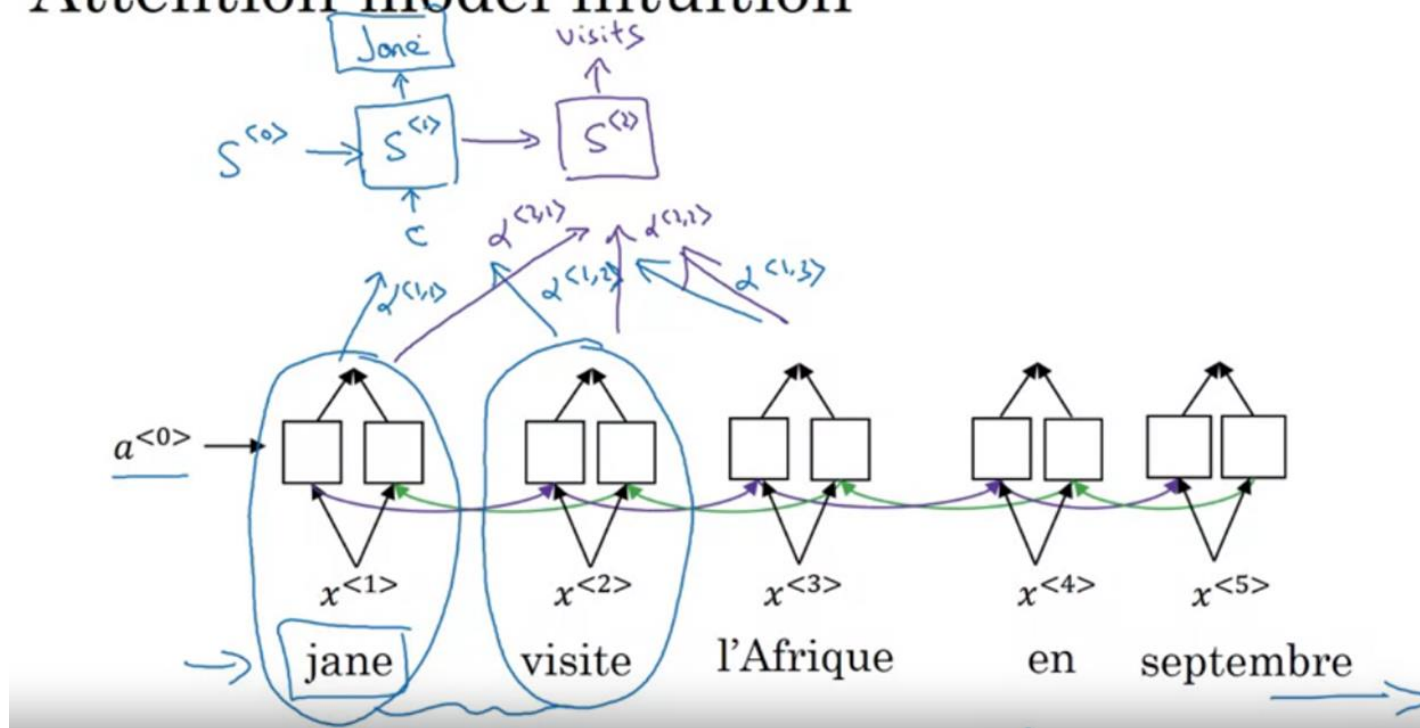
- S: Hidden State
- 우리는 첫번째 단어가 jane 이 될 것을 기대
- 목표는 Jane visits to Africa September
- 여기서 Jane이라는 이름의 결과물을 내려면 몇 개의 프랑스어 단어를 봐야하나?

Attention model intuition



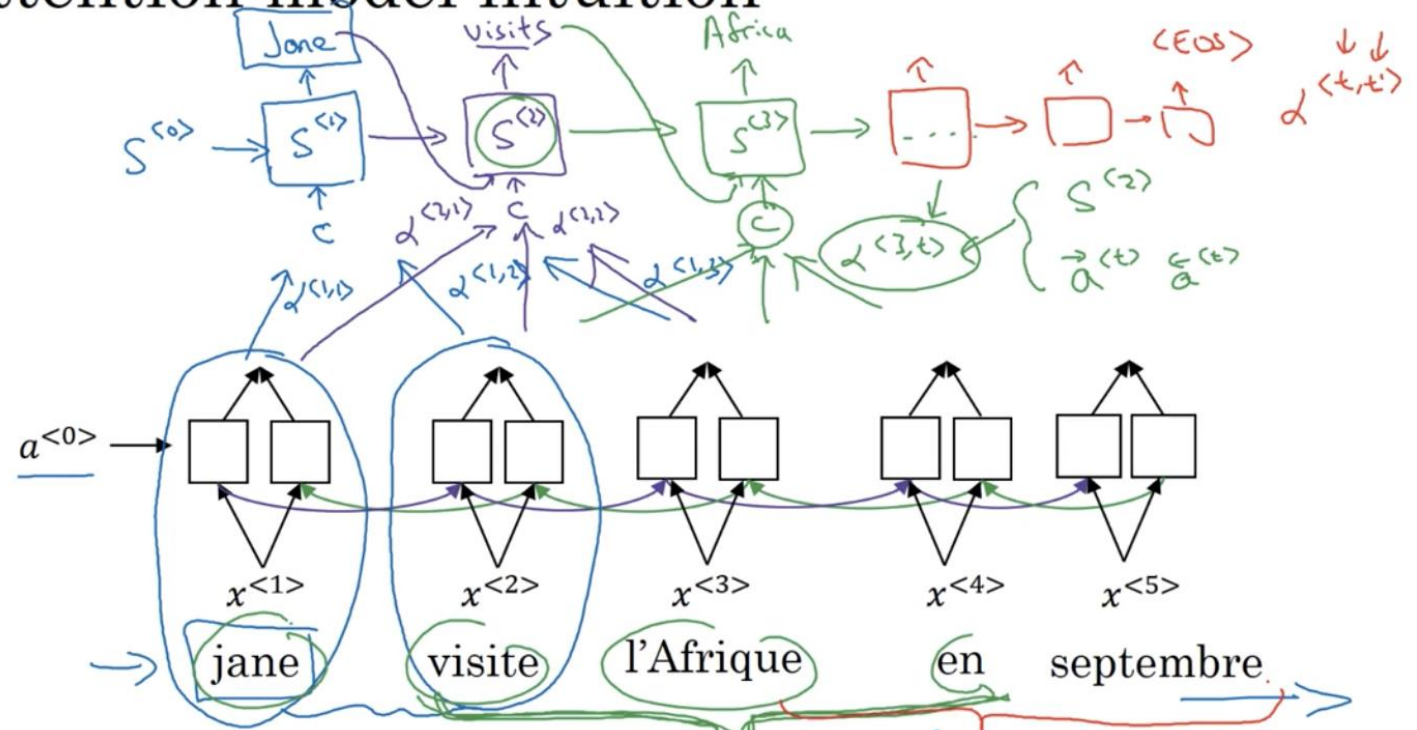
- 문장의 끝까지 볼 필요는 없음
- Attention 모델이 계산해야 할 것은 Attention Weight ( $A(1,1)$ 로 표기)
- $A(1, 1)$ 에서 첫 번째 1은 첫 번째 단어를 생성하는데 얼마나 가중치를 부여할 것인가? 두 번째 1은 첫 번째 정보를 사용하겠다는 의미
- $A(1, 2)$ : 첫번째 단어를 생성하는데 두 번째 정보를 사용하겠다. 의 의미
- 이런 정보들이 합쳐져서 맥락을 나타내는 벡터  $C$ 를 계산함

## Attention model intuition



- 이런 구성을 가짐
- $A(3, t)$ 의 경우는,  
 $S(2)$ ,  $A(2, t)$ ,  $A(4, t)$ 의  
영향을 받음

## Attention model intuition



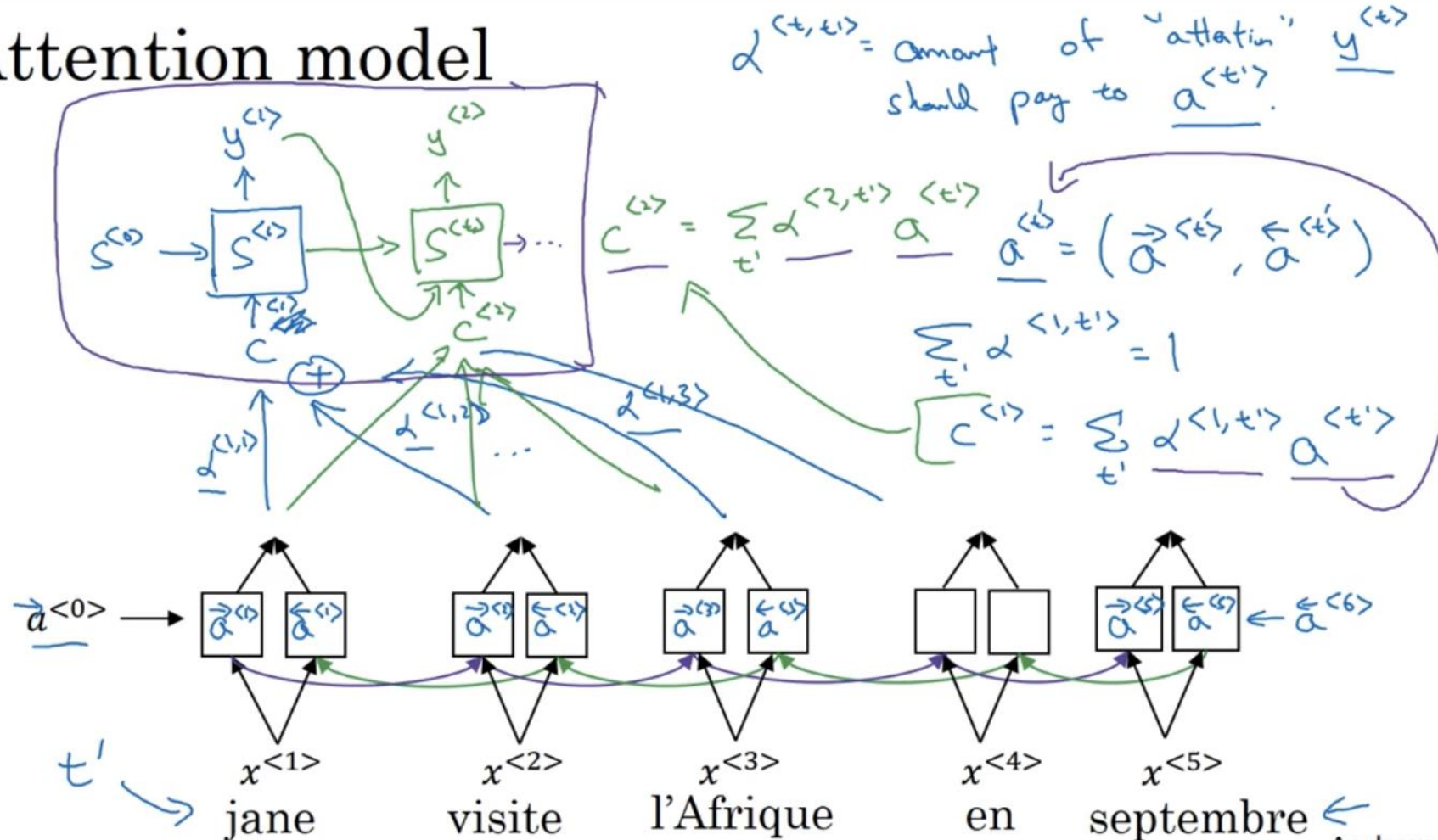
[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

Andrew Ng

- $A(t, t')$ 는  $t$ 번째 단어 번역을 할 때, 원문의  $T'$ 번째 단어에 얼마나 Attention을 줄 것이냐?  
를 의미하게 됨 → 전체 원문에서 일부, 즉 Local Window에 Attention을 주겠다는 의미

# Attention 모델 직관적으로 보기

## Attention model



$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^T \exp(e^{<t,t'>})}$$

[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

Andrew Ng

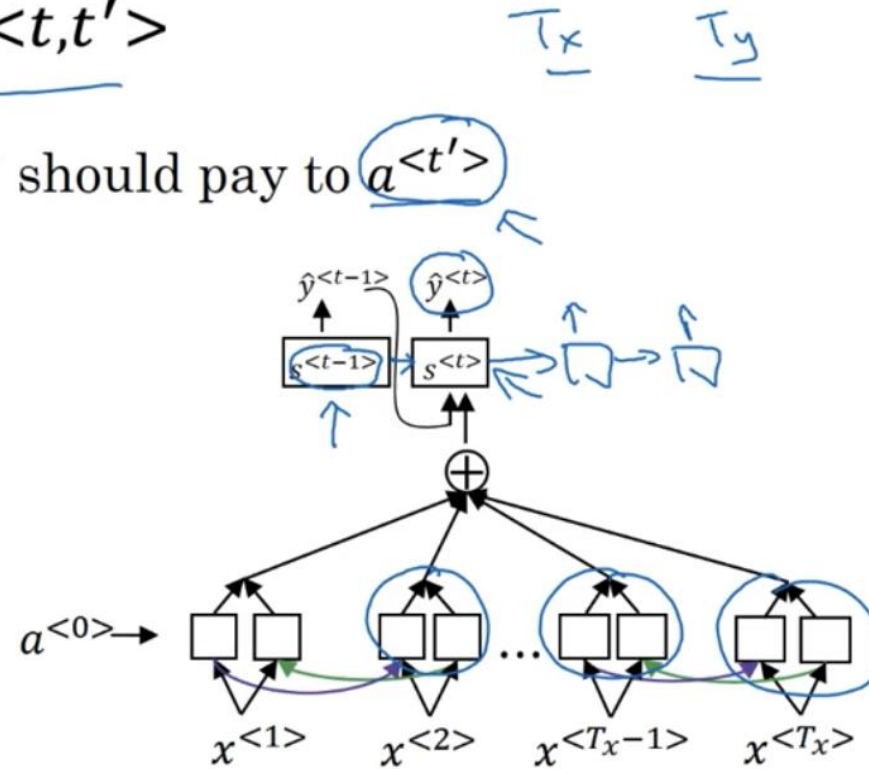


Computing attention  $\alpha^{<t,t'>}$

$\alpha^{<t,t'>}$  = amount of attention  $y^{<t>}$  should pay to  $a^{<t'>}$

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$

Diagram illustrating the computation of  $e^{<t,t'>}$ . Inputs  $s^{<t-1>}$  and  $a^{<t'>}$  are fed into a neural network (represented by a vertical stack of circles) to produce the output  $e^{<t,t'>}$ .



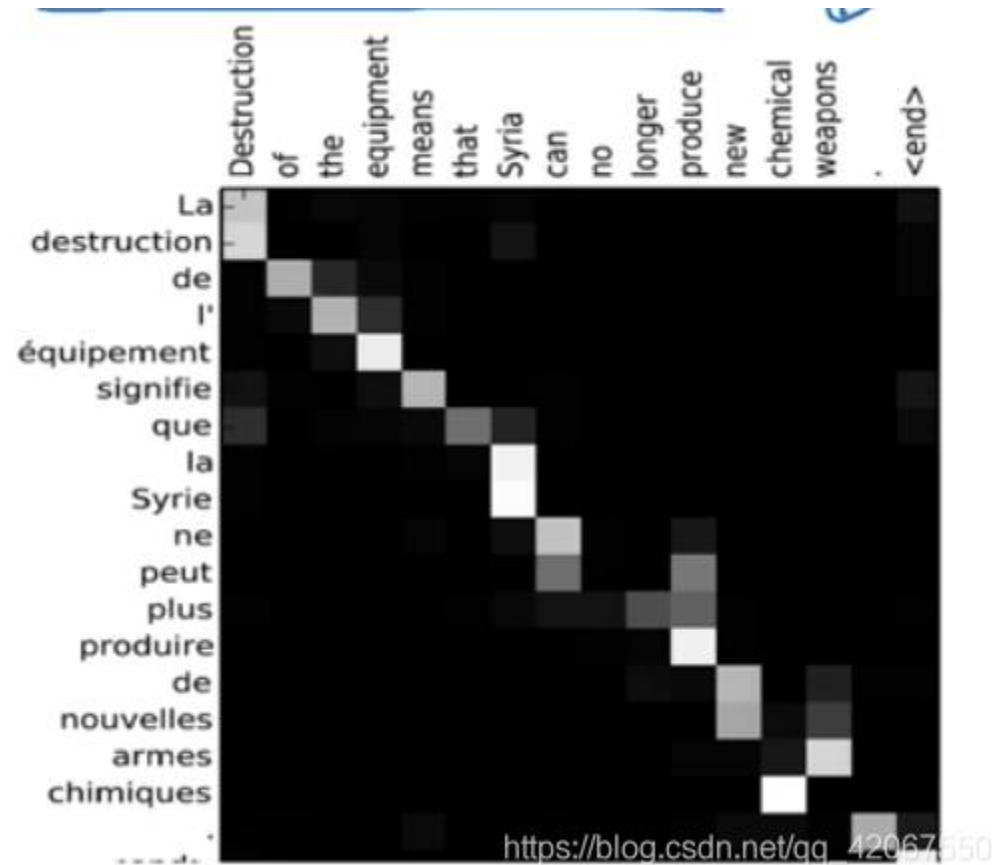
[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

[Xu et. al., 2015. Show, attend and tell: Neural image caption generation with visual attention]

Andrew Ng

# Attention 모델 직관적으로 보기

Visualization of  $\alpha^{<t,t'>}$ :



AiDA  
Lab.

- Attention은

- Query와 비슷한 값을 가진 키(Key)를 찾아서 그 값(Value)을 얻는 과정이다.  
→ 일반적인 Key-Value 방식과 비교해 볼 수 있다.

- Key-Value 자료형

- 컴퓨터 공학의 많은 분야에서 사용되고 있는 자료형
- 파이썬의 경우 Dictionary 자료형을 예로 들 수 있음





- Key-Value 자료형

- 키(Key)와 값(Value)이라는 두 개의 쌍으로 구성되며, 키를 통해서 맵핑된 값을 찾아낼 수 있다는 특징을 가짐

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

```
print(dict["2017"]) #2017이라는 키에 해당되는 값을 출력
```

Transformer

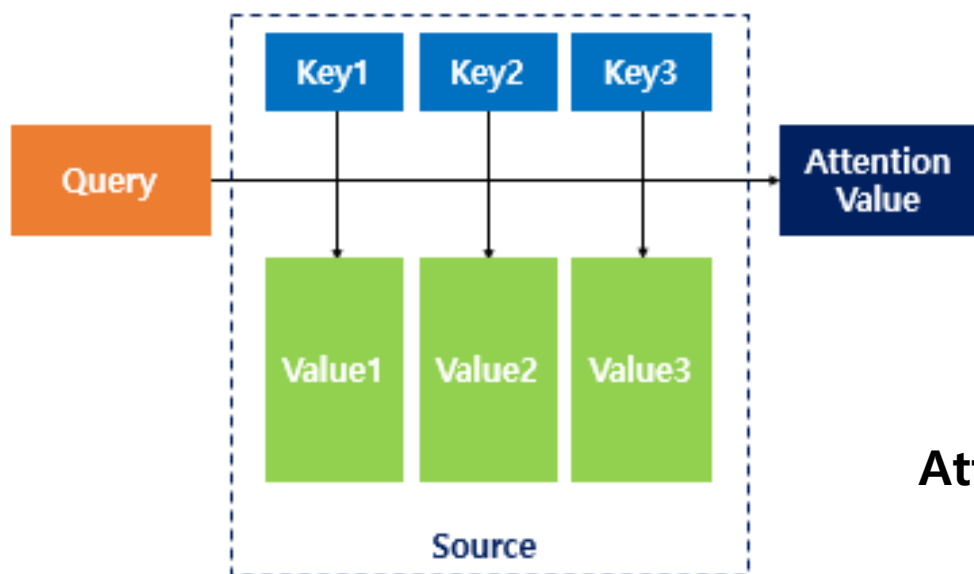
```
print(dict["2018"]) #2018이라는 키에 해당되는 값을 출력
```

BERT

"2017"는 키, "Transformer"는 "2017"의 키와 맵핑된 값

"2018"은 키, "BERT"는 "2018"과 맵핑된 값

- Key-Value 자료형을 기준으로 Attention 함수를 살펴보면



Attention을 함수로 표현하면 다음과 같다.

$$Attention(Q, K, V) = Attention\ Value$$

Q = Query : t 시점의 디코더 셀에서의 은닉 상태

K = Keys : 모든 시점의 인코더 셀의 은닉 상태들

V = Values : 모든 시점의 인코더 셀의 은닉 상태들

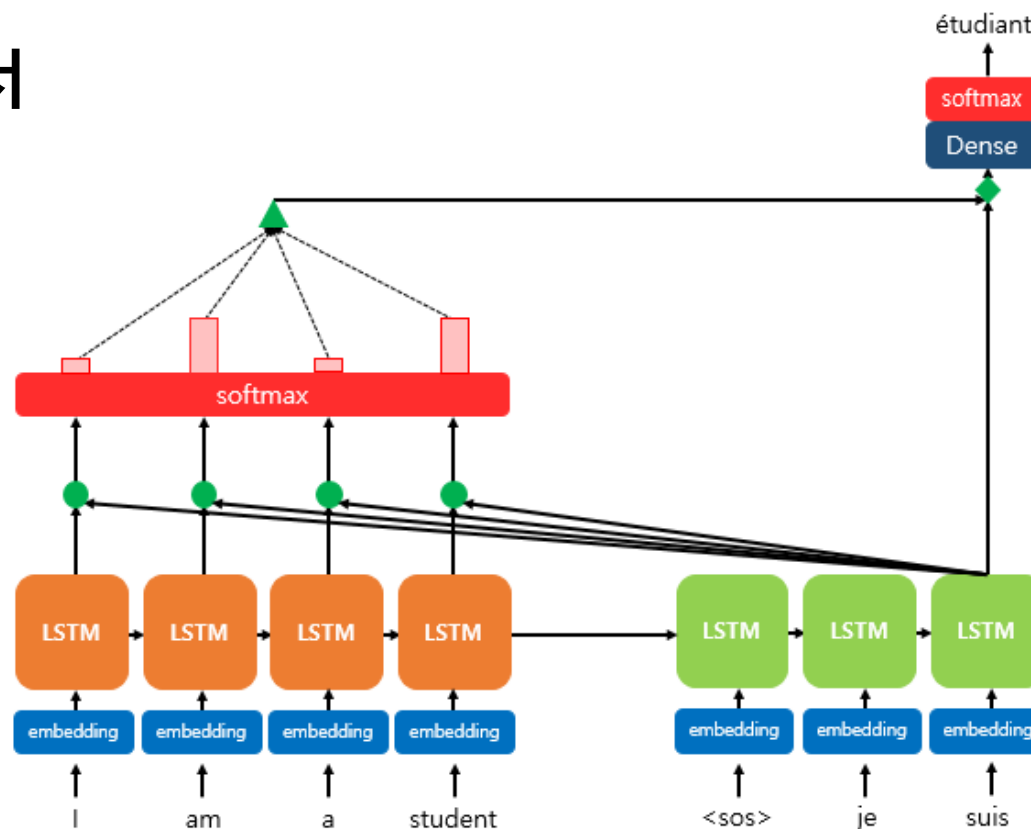
## Attention 함수는...

1. 주어진 쿼리(Query)에 대해서 모든 키(Key)와의 유사도를 각각 계산
2. 계산된 유사도를 키와 매핑되어 있는 각각의 값(Value)에 반영
3. 유사도가 반영된 값(Value)을 모두 더해서 반환
4. 이때 반환되는 값이 Attention Value

- Dot-Product Attention은

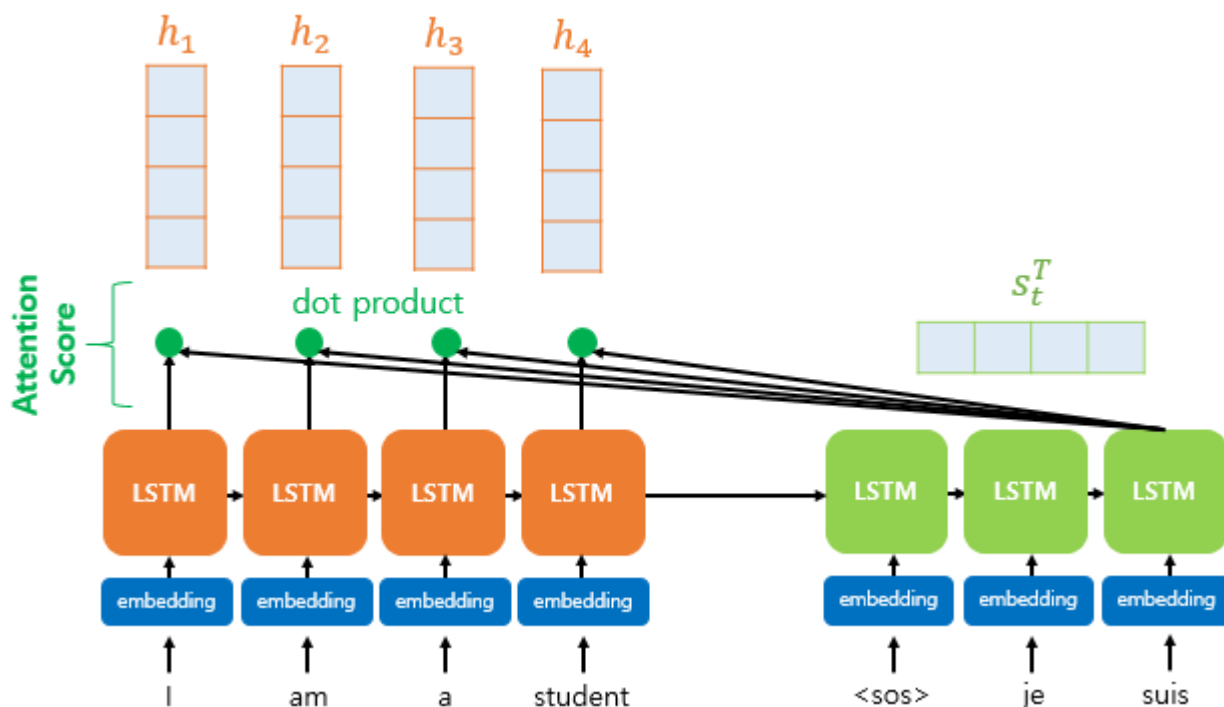
- 다양한 어텐션 종류 중에서 가장 수식적으로 이해하기 쉽게 적용된 모델
- seq2seq에서 사용되는 어텐션 중에서 Dot-Product Attention과 다른 어텐션들은 주로 중간 수식만 다를 뿐 메커니즘은 거의 유사함

그림은 디코더의 세번째 LSTM에서 출력 단어를 예측할 때, 어텐션 메커니즘을 사용하는 모습을 보여줌

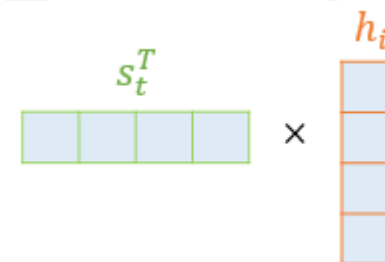


## • 처리 과정

### 1. 어텐션 스코어(Attention Score)를 구한다.



$s_t$ 와 인코더의  $i$ 번째 은닉상태의  
어텐션 스코어 계산 방법



어텐션 스코어의 정의

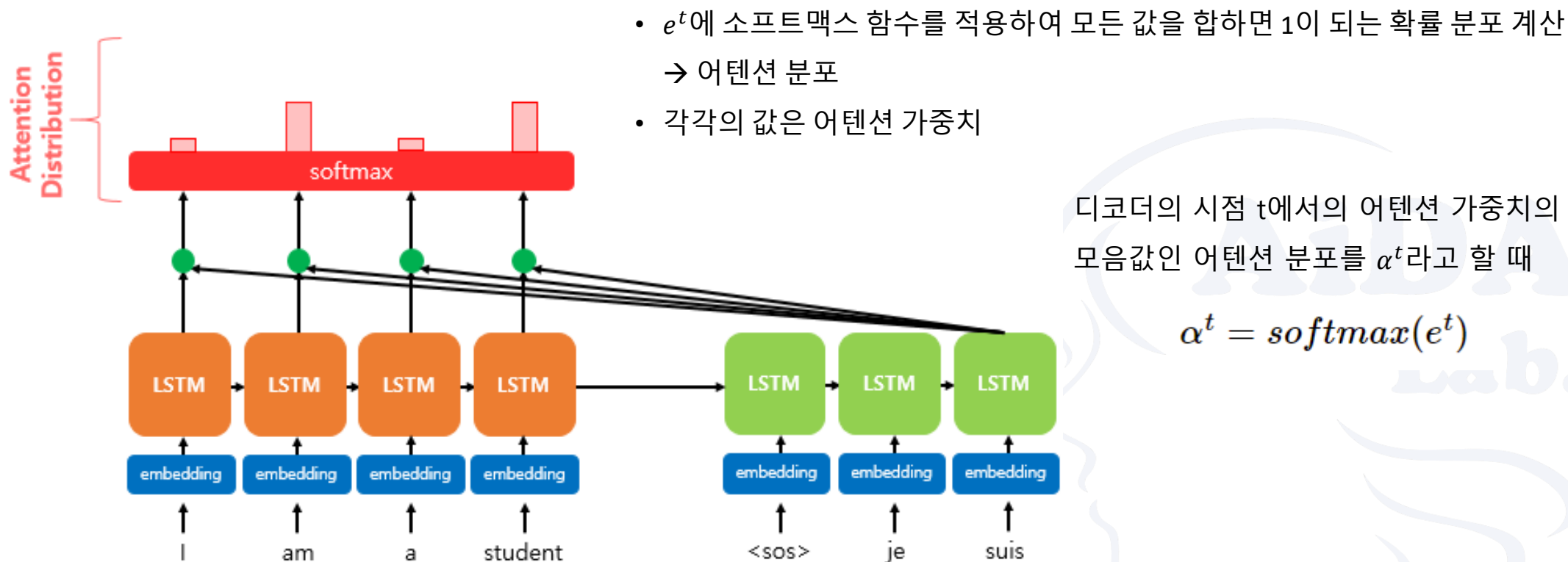
$$score(s_t, h_i) = s_t^T h_i$$

$s_t$ 와 인코더의 모든 은닉상태의  
어텐션 스코어의 모음 값을  $e^t$

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

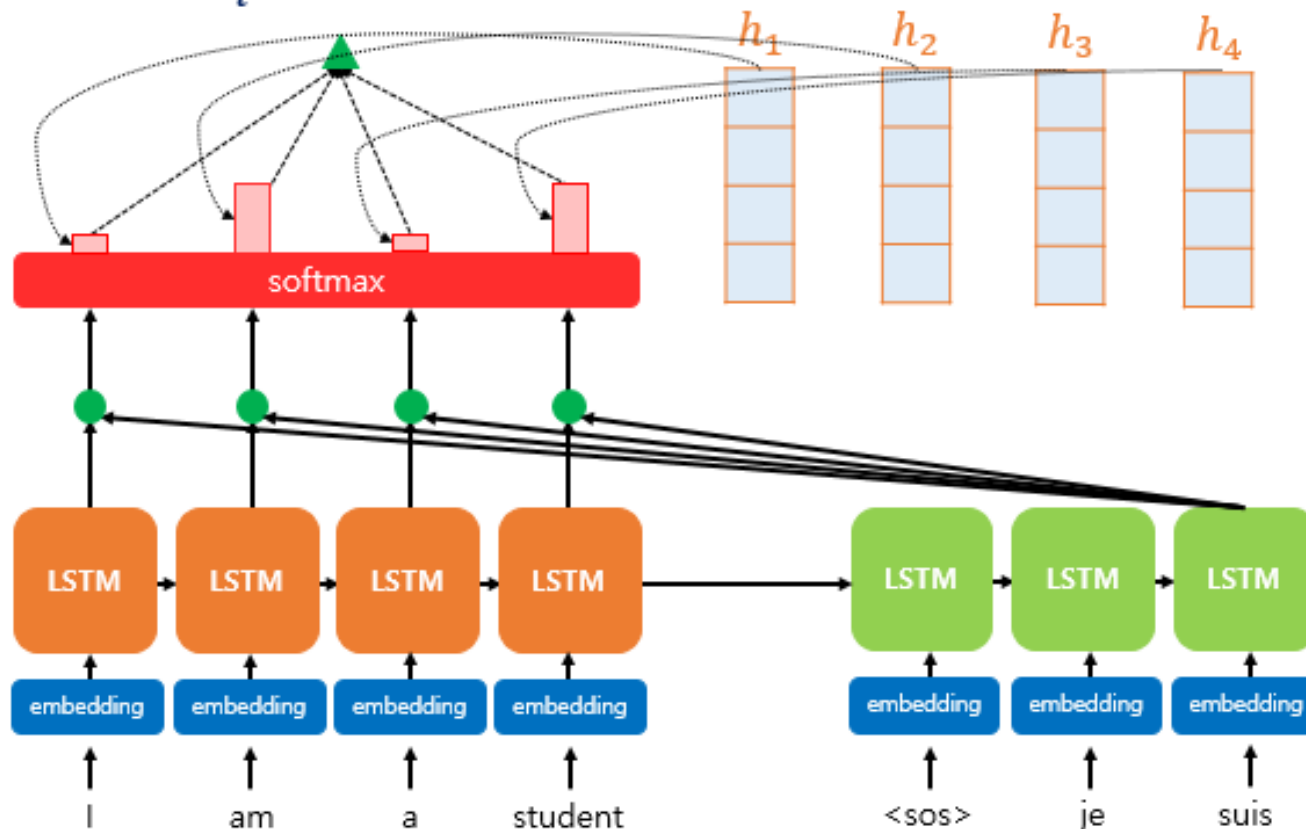
- 인코더의 시점(time step)을 각각 1, 2, ... N이라고 하였을 때, 인코더의 은닉 상태(hidden state)를 각각  $h_1, h_2, \dots, h_N$
- 디코더의 현재 시점(time step)  $t$ 에서의 디코더의 은닉 상태(hidden state)를  $s_t$
- 인코더의 은닉 상태와 디코더의 은닉 상태의 차원이 같다고 가정 (그림의 경우에는 인코더의 은닉 상태와 디코더의 은닉 상태가 동일하게 차원이 4)

## 2. 소프트맥스(softmax) 함수를 통해 어텐션 분포(Attention Distribution) 계산



## 3. 각 인코더의 어텐션 가중치와 은닉 상태를 가중합하여 어텐션 값 계산

Attention Value  $a_t$



지금까지 준비해온 정보들을 하나로 합치는 단계

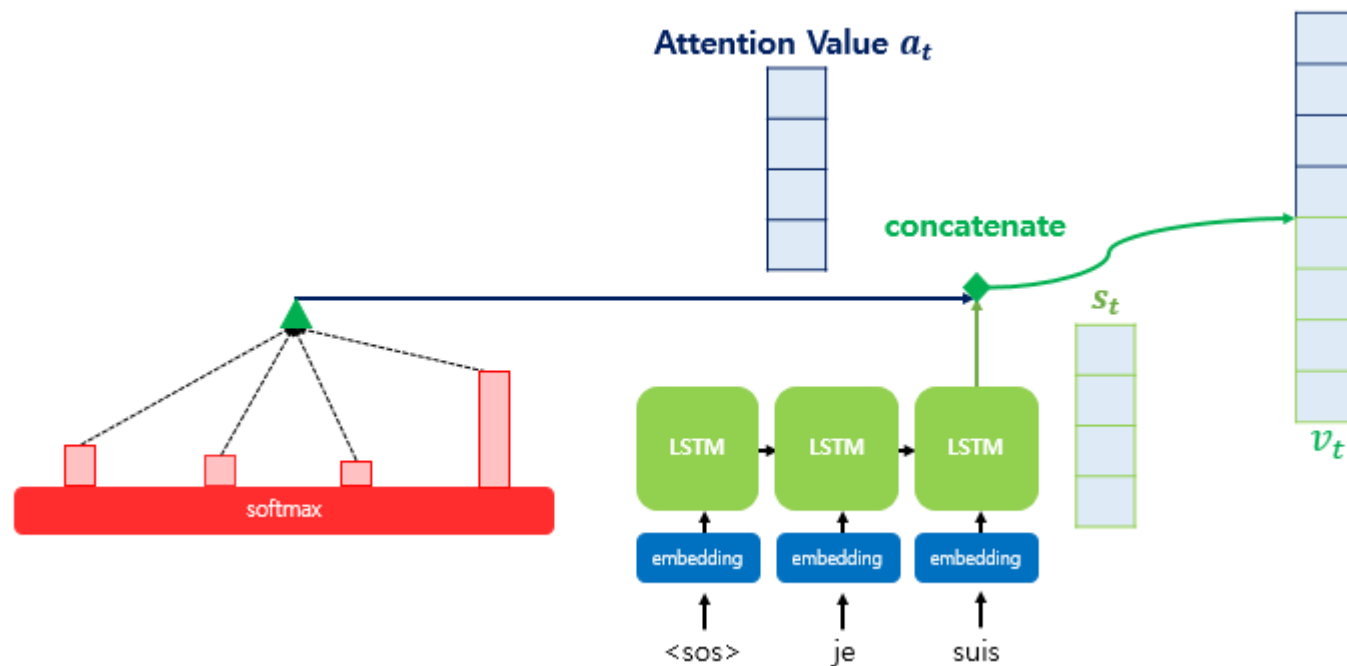
어텐션의 최종 결과값을 얻기 위해서 각 인코더의 은닉 상태와 어텐션 가중치값들을 곱하고, 최종적으로 모두 더한다 → 가중합(Weighted Sum) 계산

어텐션의 최종 결과, 즉, 어텐션 함수의 출력값인 어텐션 값(Attention Value)  $a_t$ 에 대한 식

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

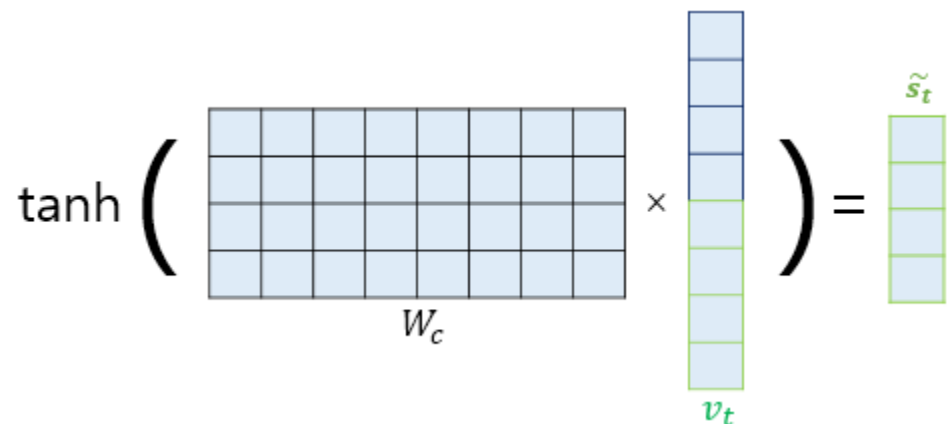
어텐션 값(Attention Value)  $a_t$  은 인코더의 문맥을 포함하고 있으므로 컨텍스트 벡터(context vector)라고도 부름

## 4. 어텐션 값과 디코더의 $t$ 시점의 은닉 상태를 연결(Concatenate)



- 어텐션 값이 구해지면  $a_t$ 를  $s_t$ 와 결합(concatenate)하여 하나의 벡터로 만드는 작업 수행  $\rightarrow v_t$
  - $v_t$ 를  $\hat{y}$  예측 연산의 입력으로 사용하고 여기에 인코더로부터 얻은 정보를 활용하여  $\hat{y}$ 를 좀 더 잘 예측할 수 있게 함
- $\rightarrow$  어텐션 메커니즘의 핵심**

## 5. 출력층 연산의 입력이 되는 $\tilde{s}_t$ 를 계산


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

- $v_t$  를 바로 출력층으로 보내기 전에 신경망 연산을 한 번 더 추가 (필수는 아님. 논문에서의 사례)
- 가중치 행렬과 곱한 후에 하이퍼볼릭탄젠트 함수를 지나도록 하여 출력층 연산을 위한 새로운 벡터인  $\tilde{s}_t$  확보
- 어텐션 메커니즘을 사용하지 않는 seq2seq에서는 출력층의 입력이 t시점의 은닉 상태인  $s_t$  였으나 어텐션 메커니즘에서는 출력층의 입력이  $\tilde{s}_t$  가 됨

## 6. $\tilde{s}_t$ 를 출력층의 입력으로 사용하여 예측 벡터 획득

$$\hat{y}_t = \text{Softmax} (W_y \tilde{s}_t + b_y)$$



## • 다양한 어텐션의 차이는 중간 수식(어텐션 스코어 함수)

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>	$\alpha_t = softmax(W_a s_t)$ // $\alpha_t$ 산출 시에 $s_t$ 만 사용하는 방법.	Luong et al. (2015)	

$s_t$ 는 Query,  $h_i$ 는 Keys,  $W_a$ 와  $W_b$ 는 학습 가능한 가중치 행렬

- 어텐션은 RNN 기반의 seq2seq 성능 보정을 목적으로 제안되었지만
- 현재에 이르러서는 어텐션 스스로가 기존의 seq2seq를 대체하는 방법이 되고 있음 → Transformer
- Attention 소개 논문
  - <https://arxiv.org/pdf/1409.0473.pdf>



- 바나다우 어텐션 함수(Bahdanau Attention Function)

**Attention(Q, K, V) = Attention Value**

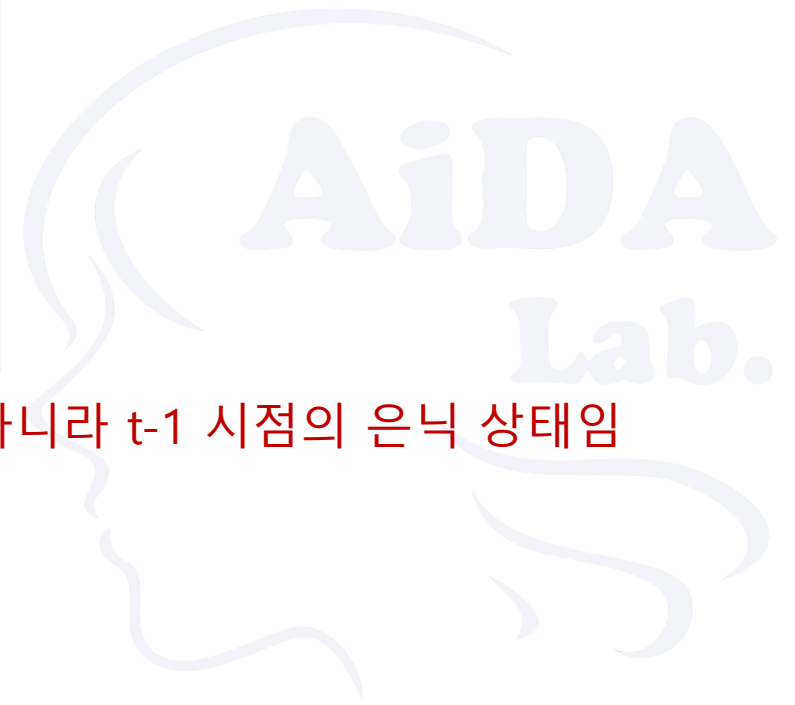
$t$  = 어텐션 메커니즘이 수행되는 디코더 셀의 현재 시점을 의미.

$Q$  = Query :  $t-1$  시점의 디코더 셀에서의 은닉 상태

$K$  = Keys : 모든 시점의 인코더 셀의 은닉 상태들

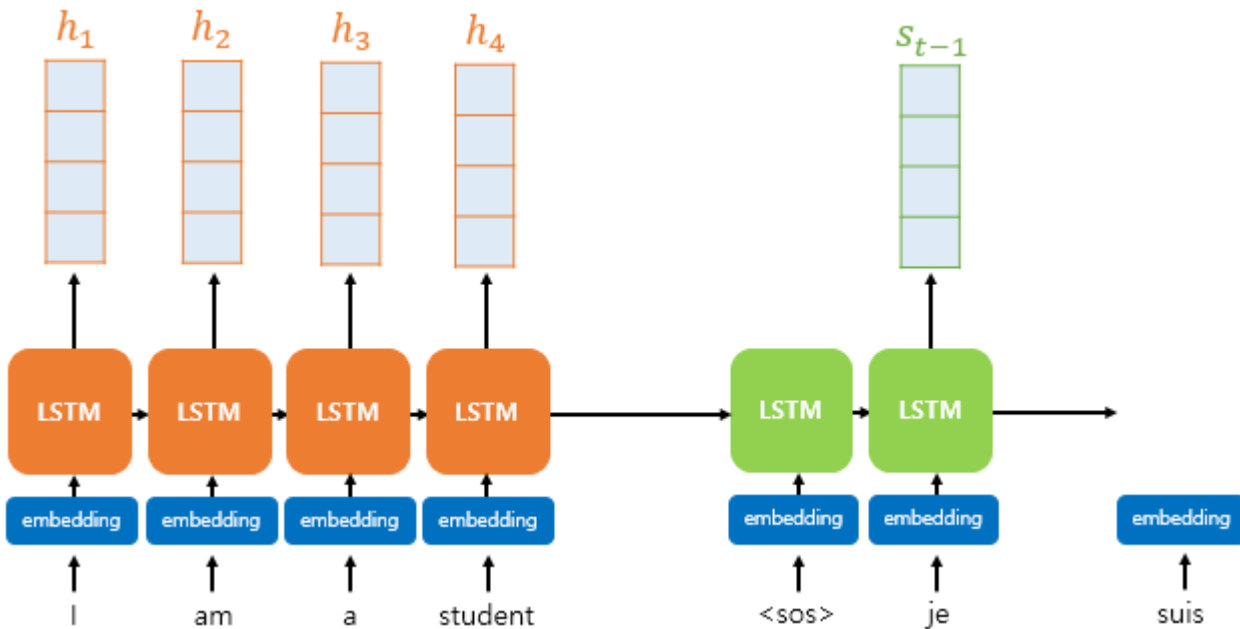
$V$  = Values : 모든 시점의 인코더 셀의 은닉 상태들

어텐션 함수의 Query가 디코더 셀의  $t$  시점의 은닉 상태가 아니라  $t-1$  시점의 은닉 상태임



## • 처리 과정

### 1. 어텐션 스코어(Attention Score) 계산



$s_{t-1}$ 과 인코더의  $i$ 번째 은닉상태의  
어텐션 스코어 계산 방법

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

$W_a, W_b, W_c$ 는 학습 가능한 가중치 행렬

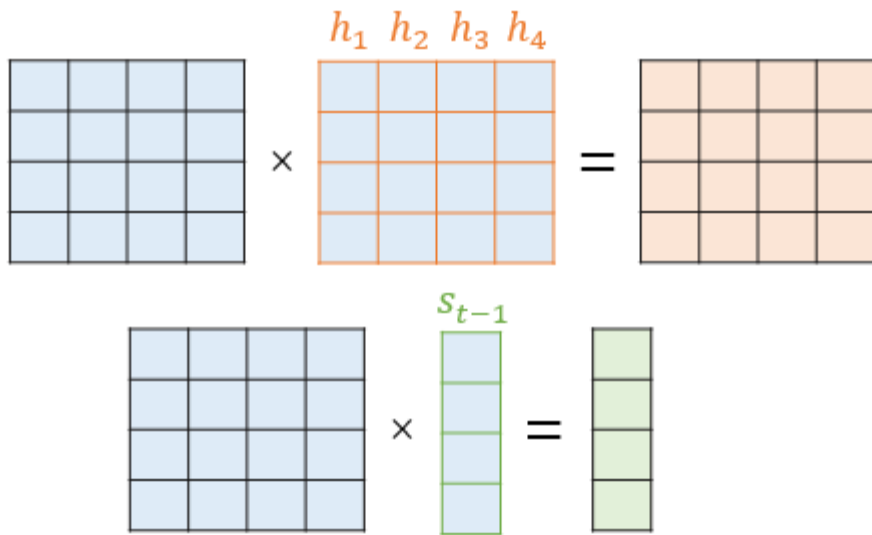
$s_{t-1}$  과 의  $h_1, h_2, h_3, h_4$  어텐션 스코어를 각각 구해야하므로  
병렬 연산을 위해 를 하나의 행렬  $H$ 로 적용

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

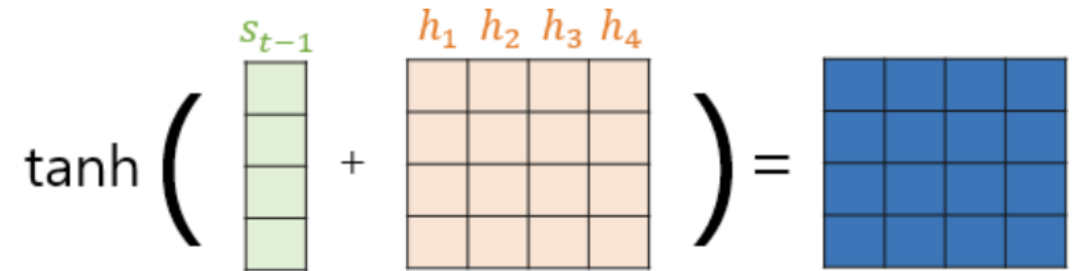
- 인코더의 시점(time step)을 각각 1, 2, ... N이라고 하였을 때, 인코더의 은닉 상태(hidden state)를 각각  $h_1, h_2, \dots, h_N$
- 디코더의 현재 시점(time step)  $t$ 에서의 디코더의 은닉 상태(hidden state)를  $s_t$
- 인코더의 은닉 상태와 디코더의 은닉 상태의 차원이 같다고 가정 (그림의 경우에는 인코더의 은닉 상태와 디코더의 은닉 상태가 동일하게 차원이 4)

# 바나다우 어텐션(Bahdanau Attention)

우선  $W_b s_{t-1}$ 와  $W_c H$ 를 각각 구하면


$$\begin{matrix} & h_1 & h_2 & h_3 & h_4 \\ \begin{matrix} \text{blue grid} \end{matrix} & \times & \begin{matrix} \text{orange grid} \end{matrix} & = & \begin{matrix} \text{orange grid} \end{matrix} \\ \\ \begin{matrix} \text{blue grid} \end{matrix} & \times & \begin{matrix} \text{green column} \end{matrix} & = & \begin{matrix} \text{green column} \end{matrix} \end{matrix}$$

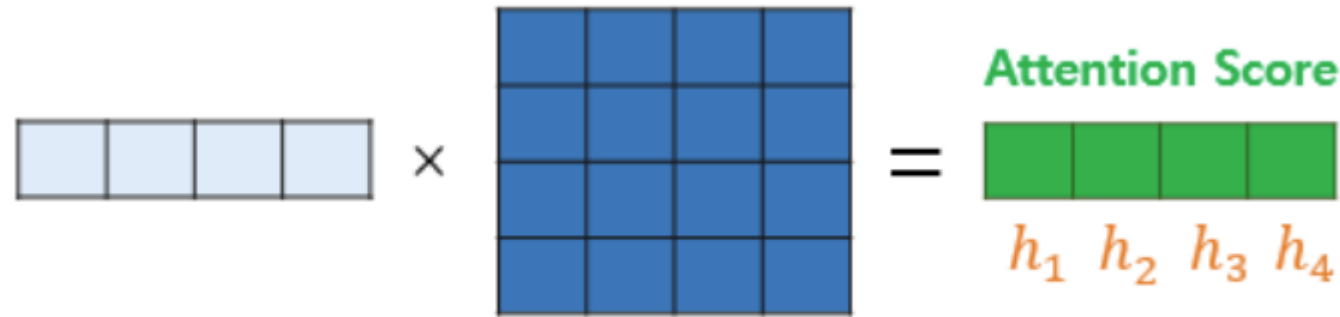
이들을 더한 후, 하이퍼볼릭탄젠트 함수를 지나도록


$$\tanh \left( \begin{matrix} \text{green column} \end{matrix} + \begin{matrix} \text{orange grid} \end{matrix} \right) = \begin{matrix} \text{blue grid} \end{matrix}$$

$$\tanh(W_b s_{t-1} + W_c H)$$

# 바나다우 어텐션(Bahdanau Attention)

이제  $W_a^T$ 와 곱하여  $s_{t-1}$ 와  $h_1, h_2, h_3, h_4$ 의 유사도가 기록된 어텐션 스코어 벡터  $e^t$  계산



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



## 2. 소프트맥스(softmax) 함수를 통해 어텐션 분포 계산

$$\text{softmax} \left( \begin{array}{c} \text{Attention Score} \\ \text{[Green Box]} \\ h_1 \ h_2 \ h_3 \ h_4 \end{array} \right) = \begin{array}{c} \text{Attention} \\ \text{Distribution} \\ \text{[Red Box]} \end{array}$$

$e^t$ 에 소프트맥스 함수 적용, 모든 값을 합하면 1이 되는 확률 분포 획득  
→ 어텐션 분포(Attention Distribution),  
각각의 값은 어텐션 가중치(Attention Weight)

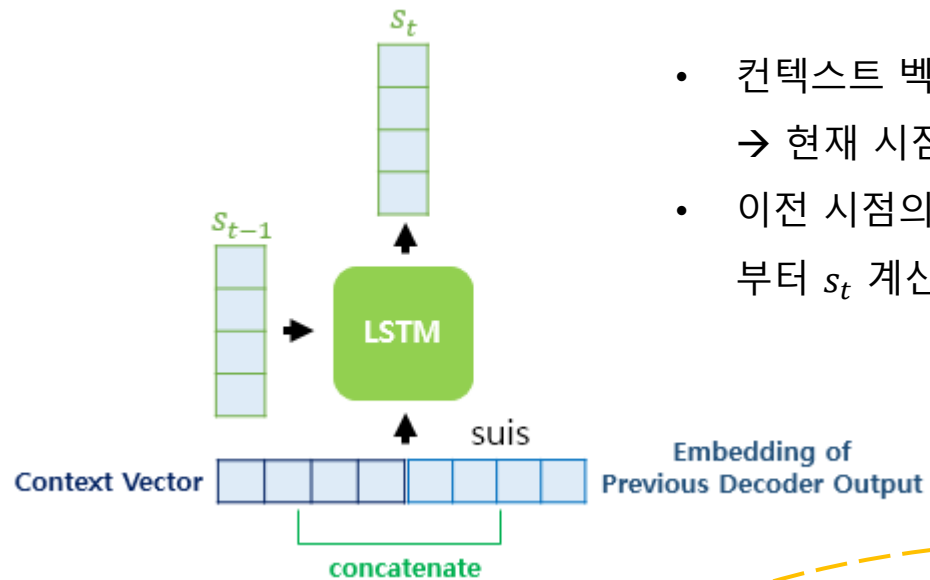
## 3. 각 인코더의 어텐션 가중치와 은닉 상태를 가중합하여 어텐션 값 계산

$$\begin{array}{c} h_1 \ h_2 \ h_3 \ h_4 \\ \begin{bmatrix} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{bmatrix} \times \begin{bmatrix} \square \\ \square \\ \square \\ \square \end{bmatrix} = \begin{array}{c} \text{Context Vector} \\ \begin{bmatrix} \square & \square & \square & \square \end{bmatrix} \end{array}$$

지금까지 준비해온 정보들을 하나로 합치는 단계

각 인코더의 은닉 상태와 어텐션 가중치값들을 곱하고,  
최종적으로 모두 더하여 가중합(Weighted Sum) 계산  
→ 컨텍스트 벡터

## 4. 컨텍스트 벡터로부터 $s_t$ 계산



- 컨텍스트 벡터와 현재 시점의 입력인 단어의 임베딩 벡터를 연결(concatenate)  
→ 현재 시점의 새로운 입력으로 사용
- 이전 시점의 셀로부터 전달받은 은닉 상태  $s_{t-1}$ 과 현재 시점의 새로운 입력으로 부터  $s_t$  계산 →  $s_t$ 는 출력층으로 전달되어 현재 시점의 예측값 계산

LSTM이 임베딩된 단어 벡터를 입력으로 하는 것과 비교할 때,  
컨텍스트 벡터와 임베딩된 단어 벡터를 연결(concatenate)하여  
입력으로 사용하는 것이 다름

[LSTM이  $s_t$ 를 구할 때]

- LSTM은 이전 시점의 셀로부터 전달받은 은닉 상태  $s_{t-1}$ 과 현재 시점의 입력  $x_t$ 를 가지고 연산
- 오른쪽의 LSTM은 seq2seq의 디코더이며 현재 시점의 입력  $x_t$ 는 임베딩된 단어 벡터

