

# 딥러닝팀

## 1팀

이수경  
이승우  
이은서  
주혜인  
홍현경

# INDEX

---

1. 자연어의 데이터화

2. seq2seq

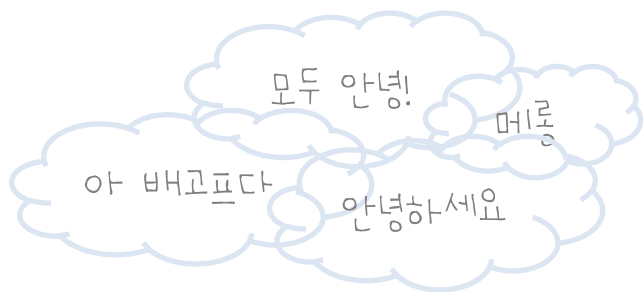
3. Attention

# 1

## 자연어의 데이터화

# 1 자연어의 데이터화

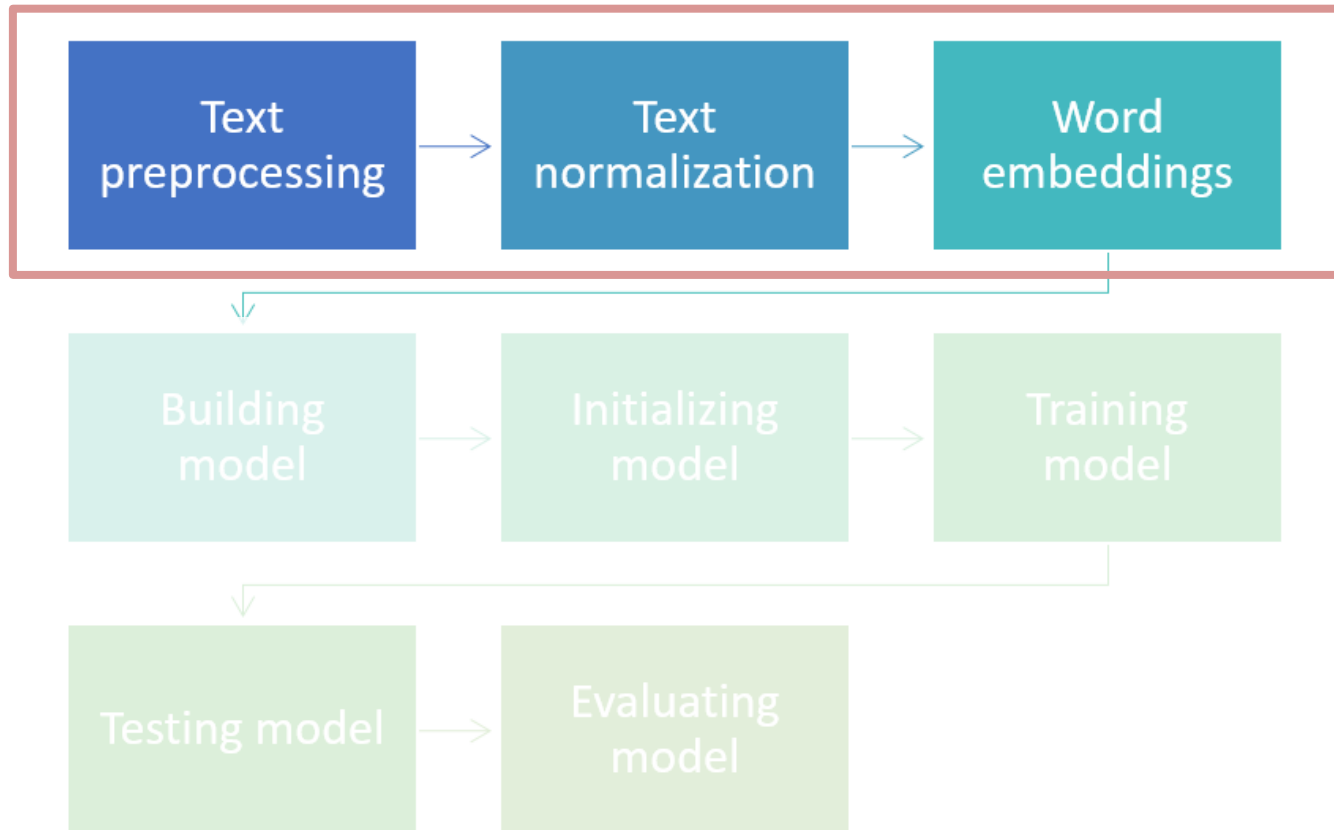
- Data preprocessing



컴퓨터는 자연어를 그대로 인식하지 못한다

# 1 자연어의 데이터화

- Data preprocessing



모델이 사용할 수 있는 형태로 바꿔주는 작업 필요

# 1 자연어의 데이터화

- Tokenization

## Tokenization이란?



↖ 문장/문서 단위의 자연어 데이터 집합

**코퍼스(Corpus)** 데이터를 **토큰(token)** 단위로 나누는 전처리 작업

ex) 구, 문장, 문단, 단어 등

# 1 자연어의 데이터화

## ● Tokenization

### Tokenization-Code

```
In [1]: from nltk.tokenize import wordpunct_tokenize
text = "The cat's sitting on the mat."
wordpunct_tokenize(text)
```

```
Out[1]: ['The', 'cat', "'", 's', 'sitting', 'on', 'the', 'mat', '.']
```

```
In [2]: from nltk.tokenize import word_tokenize
word_tokenize(text)
```

```
Out[2]: ['The', 'cat', 's', 'sitting', 'on', 'the', 'mat', '.']
```

```
In [3]: from nltk.tokenize import sent_tokenize
text="His barber kept his word. But keeping such a huge secret to himself was driving him crazy. Finally, the barber went up a mountain and almost to the edge of a cliff. He dug a hole in the midst of some reeds. He looked about, to make sure no one was near."
t=sent_tokenize(text)
print(t, f'##numbers of sentence: {len(t)}')
```

```
['His barber kept his word.', 'But keeping such a huge secret to himself was driving him crazy.', 'Finally, the barber went up a mountain and almost to the edge of a cliff.', 'He dug a hole in the midst of some reeds.', 'He looked about, to make sure no one was near.']
```

```
numbers of sentence: 5
```

토큰화를 할 때 세부 처리 규칙에 따라 다양한 토큰화 방법들이 있음

# 1 자연어의 데이터화

- 한글 tokenizing

## 한글 tokenization 1) 교착어

### 한국어의 특징

잡았다 = 잡- + -았- + -다

잡히다 = 잡- + -히- + -다

잡히셨다 = 잡- + -히- + -으시- + -었- + -다

먹이다 = 먹 + -이- + -다

먹었다 = 먹 + -었- + -다

어간/어근에 어미/접사가 붙어  
의미적/문법적 변화가 생긴다



# 1 자연어의 데이터화

- 한글 tokenizing

## 한글 tokenization 1) 교착어

### 한국어의 특징

잡았다 = 잡- + -았- + -다

잡히다 = 잡- + -히- + -다

잡히셨다 = 잡- + -히- + -으시- + -었- + -다

먹이다 = 먹 + -이- + -다

먹었다 = 먹 + -었- + -다

어간/어근에 어미/접사가 붙어  
의미적/문법적 변화가 생긴다



띄어쓰기 단위가 아닌 **형태소** 단위로 토큰화  
: 의미를 가진 가장 작은 단위

# 1 자연어의 데이터화

- 한글 tokenizing

## 한글 tokenization 1) 교착어

한국어의 특징

```
In [2]: import konlpy
from konlpy.tag import Okt
text = "나는 머리가 아파서 수업에 결석했어"
okt=Okt()
print(okt.morphs(text))
```

['나', '는', '머리', '가', '아파서', '수업', '에', '결석', '했어']

잡히셨다 = 잡- + -히- + -으시- + -었- + -다

먹이다 = 먹 + -이- + -다

먹었다 = 먹 + -었- + -다

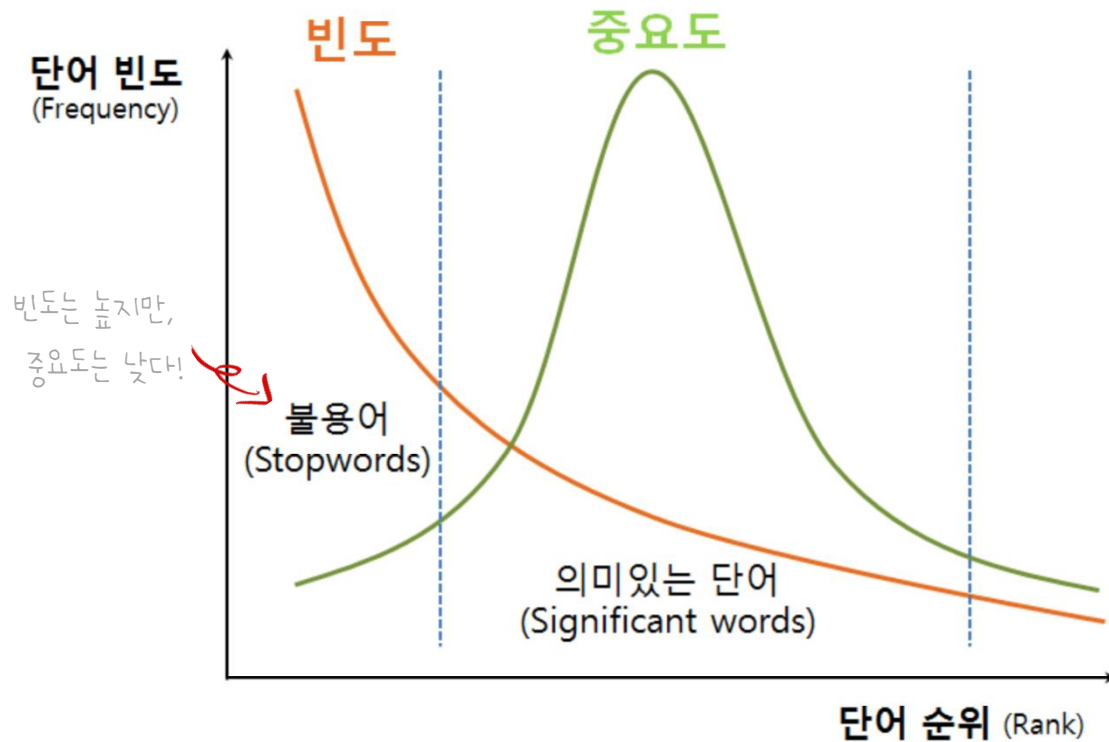
띄어쓰기 단위가 아닌 **형태소** 단위로 토큰화

의미를 가진 가장 작은 단위

# 1 자연어의 데이터화

- 한글 tokenizing

## 한글 tokenizing 2) 불용어



자주 쓰이지만 의미에 큰 기여를 하지 못하는 불용어를 제거

# 1 자연어의 데이터화

- 한글 tokenizing

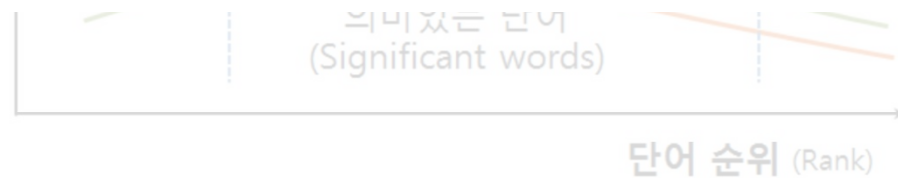
## 한글 tokenizing 2) 불용어



```
In [4]: print(oks.pos(text))
print()
word = [i for i in oks.pos(text) if i[1] != 'Josa']
print(word)
```

[('나', 'Noun'), ('는', 'Josa'), ('머리', 'Noun'), ('가', 'Josa'), ('마파서', 'Adjective'), ('수업', 'Noun'), ('에', 'Josa'), ('결석', 'Noun'), ('했어', 'Verb')]

[('나', 'Noun'), ('머리', 'Noun'), ('마파서', 'Adjective'), ('수업', 'Noun'), ('결석', 'Noun'), ('했어', 'Verb')]

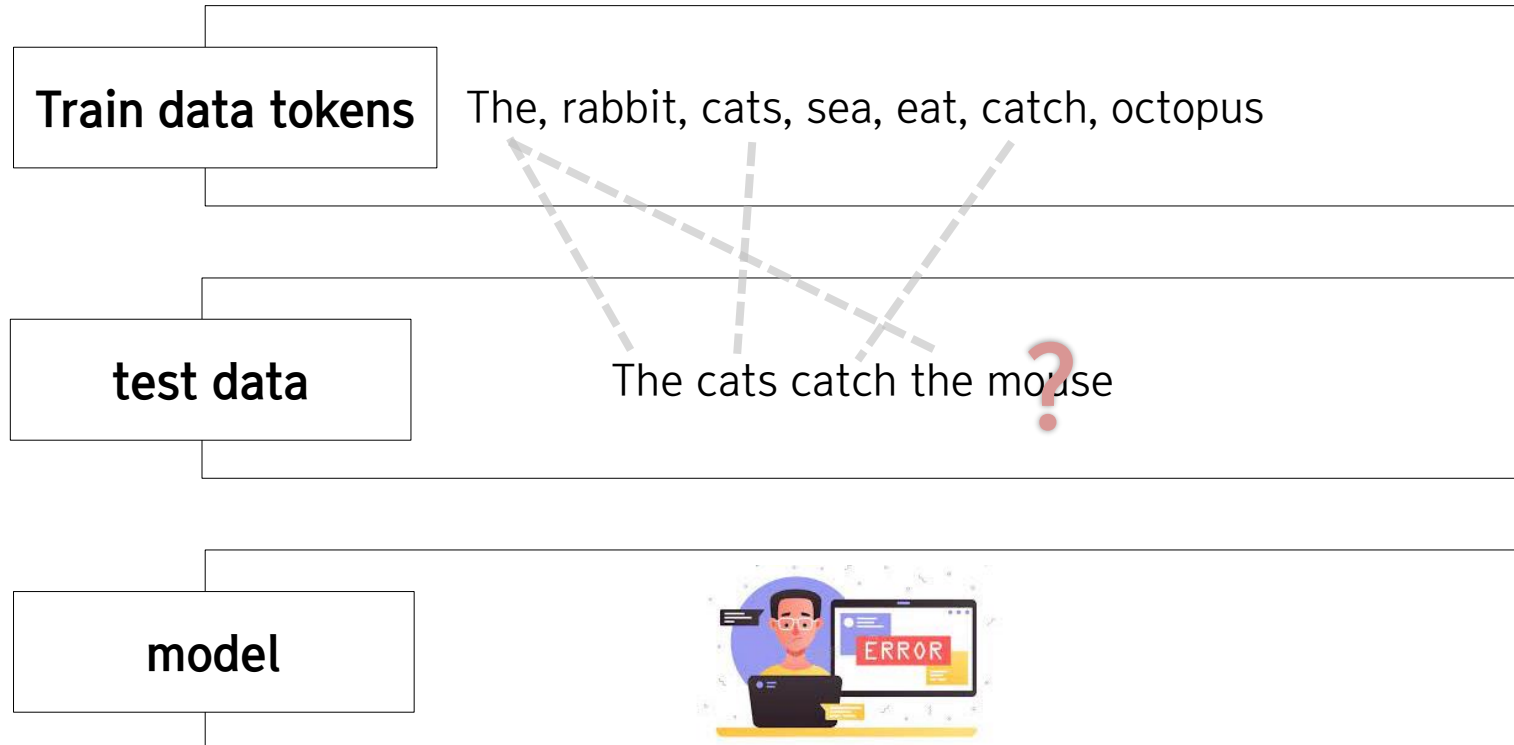


자주 쓰이지만 의미에 큰 기여를 하지 못하는 불용어를 제거

# 1 자연어의 데이터화

- Tokenization

## Out-of-Vocabulary

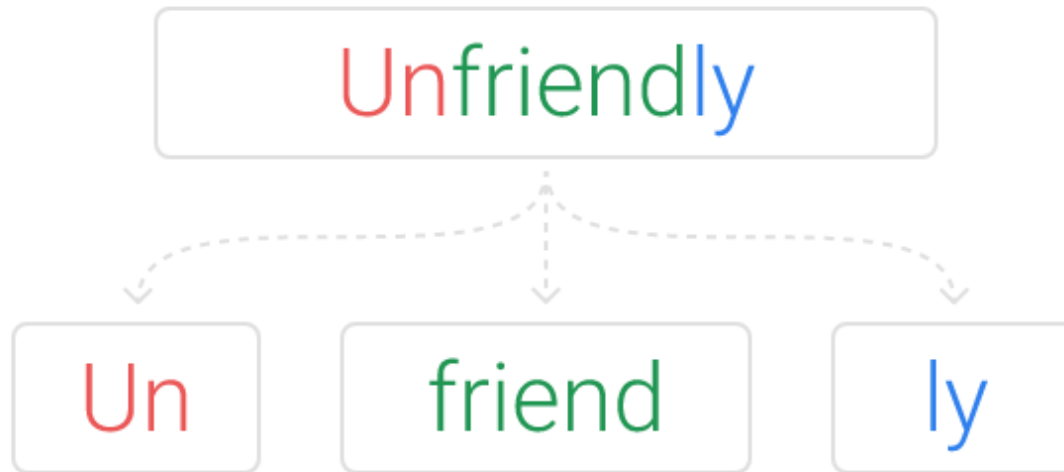


학습 데이터에 없는 단어는 처리하지 못하는 문제

# 1 자연어의 데이터화

- Subword Segmentation

## Subword Segmentation이란?

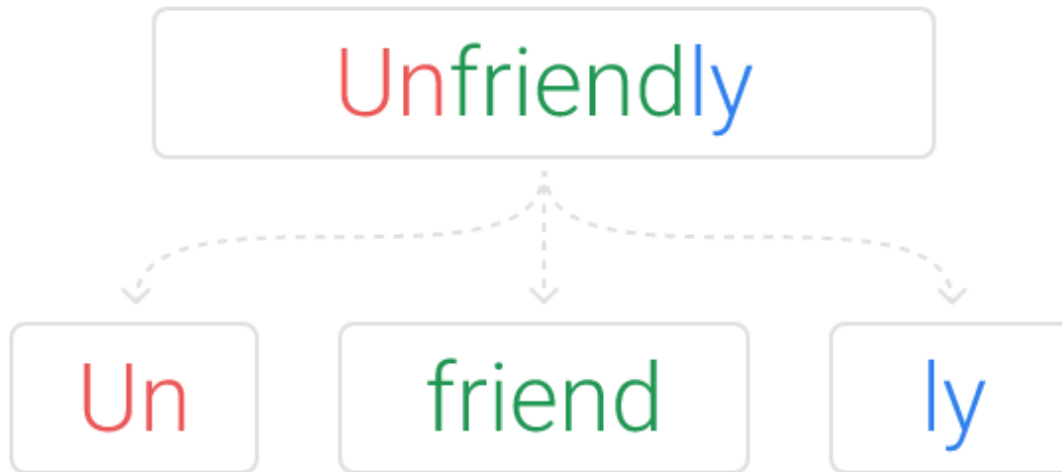


“단어는 의미를 가진 더 작은 **서브워드**들의 조합으로 이루어진다”

# 1 자연어의 데이터화

- Subword Segmentation

## Subword Segmentation이란?



“단어는 의미를 가진 더 작은 **서브워드**들의 조합으로 이루어진다”

➡ 모르는 단어를 쪼개서 의미를 추론

# 1 자연어의 데이터화

## ● Embedding

### Embedding이란?

Vocabulary:  
Man, woman, boy,  
girl, prince,  
princess, queen,  
king, monarch



	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

사람이 쓰는 자연어를 컴퓨터가 이해할 수 있는 **벡터 형태**로 바꾸는 것



# 1 자연어의 데이터화

- Embedding

## One-hot Encoding

Vocabulary:  
Man, woman, boy,  
girl, prince,  
princess, queen,  
king, monarch



	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

우리가 잘 아는 one-hot Encoding도 embedding의 한 종류!

단점: 공간적인 낭비가 일어나서 연산에 부적합하고, 단어의 의미를 담지 못함

# 1 자연어의 데이터화

## ● Embedding

### Embedding 모델

:자연어의 통계적 패턴 정보를 통해 벡터를 만드는 모델

유사한 벡터

ex) 고양이 = [0.9, -0.32, 0.27, 0.43]

'고양이'와의 상대적인 관계 학습



[Bag-of-words 가정]

유사한 등장빈도

고양이랑 유사한 빈도로 등장해서~



[언어 모델 Language Model]

유사한 등장순서

고양이랑 유사한 등장순서를 가져서~



[분포 가설 Distribution Hypothesis]

유사한 의미, 맥락

고양이랑 유사한 맥락에서 사용되어서~

# 1 자연어의 데이터화

- Embedding

## Embedding 모델

:자연어의 통계적 패턴 정보를 통해 벡터를 만드는 모델

유사한 벡터

ex) 호양이 = [0.9, -0.32, 0.27, 0.43]

'고양이'와의 상대적인 관계 학습



결합되어 응용될 수 있다

[Bag-of-words 가정]

유사한 등장빈도

고양이랑 유사한 빈도로 등장해서~

[언어 모델 Language Model]

유사한 등장순서

고양이랑 유사한 등장순서를 가져서~

[분포 가설 Distribution Hypothesis]

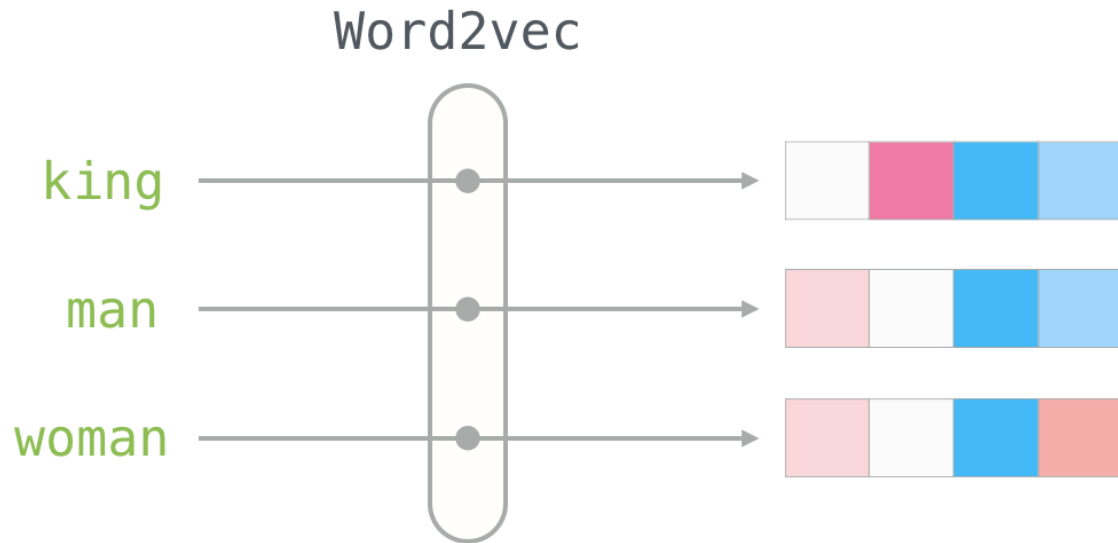
유사한 의미, 맥락

고양이랑 유사한 맥락에서 사용되어서~

# 1 자연어의 데이터화

- Embedding

## Word2Vec



Neural Network 기반으로  
단어들의 유사도를 반영하고 등장 순서를 고려

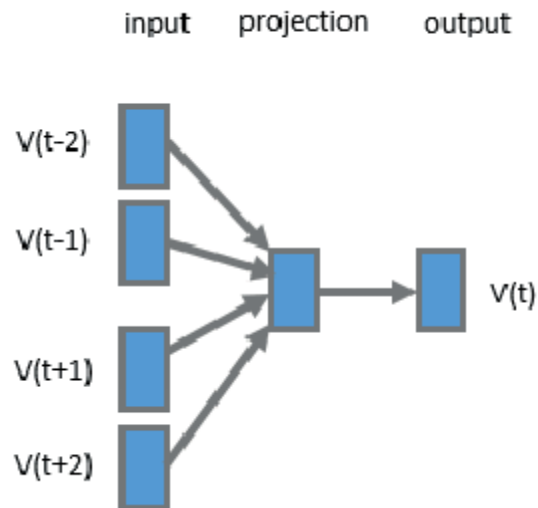
분포가설 + 언어모델 기반

# 1 자연어의 데이터화

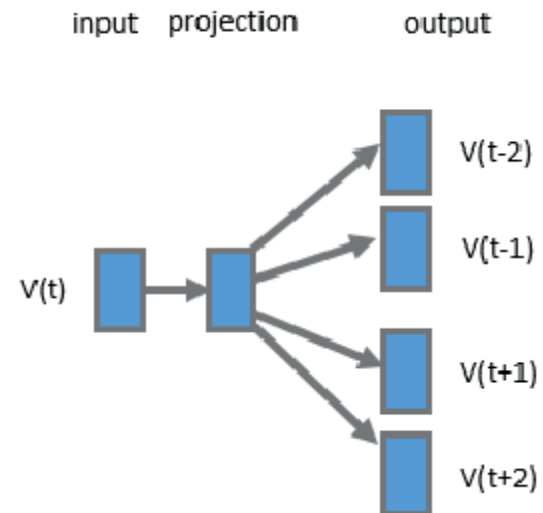
- Embedding

## Word2Vec

CBOW



Skip-gram

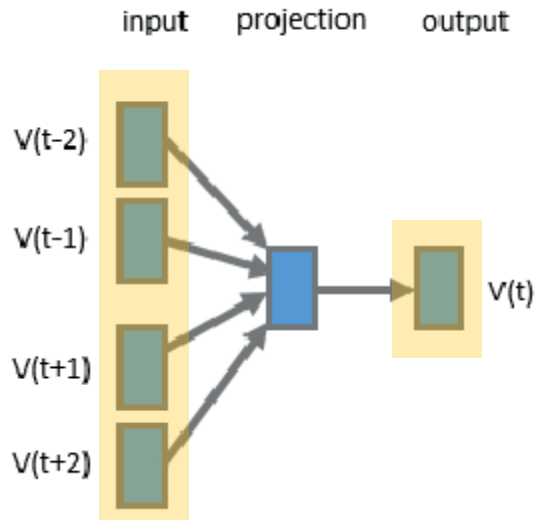


# 1 자연어의 데이터화

- Embedding

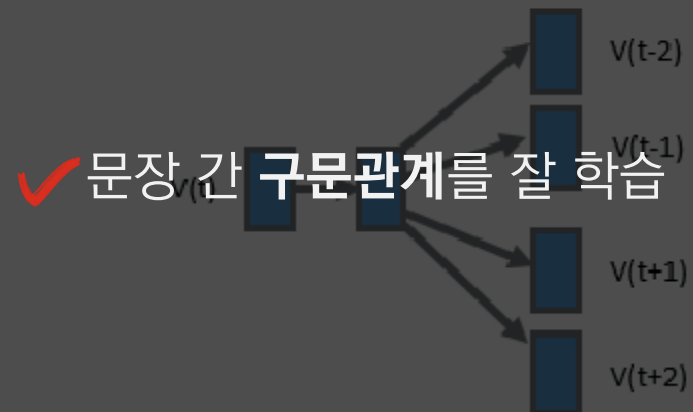
## Word2Vec

CBOW



Skip-gram

target의 주변 단어들을 기반으로  
target 예측



✓ 문장 간 구문관계를 잘 학습

# 1 자연어의 데이터화

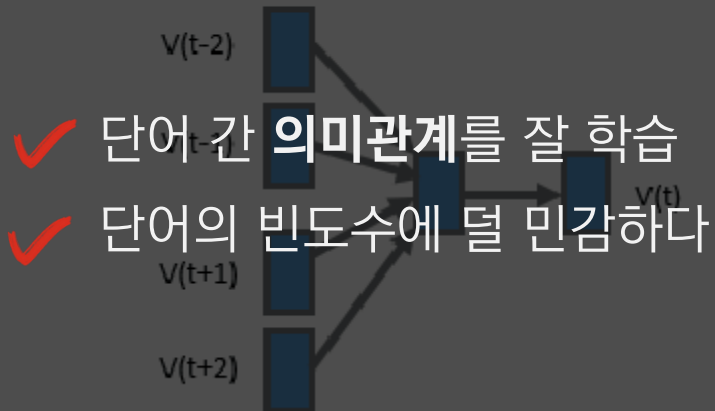
- Embedding

## Word2Vec

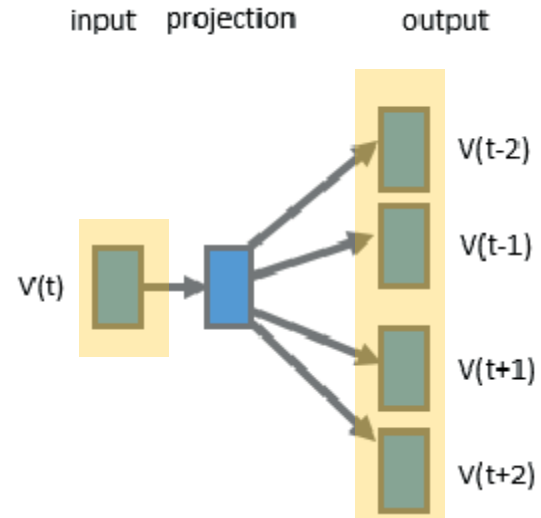
CBOW

target을 기반으로

주변 단어들을 예측



Skip-gram



# 1 자연어의 데이터화

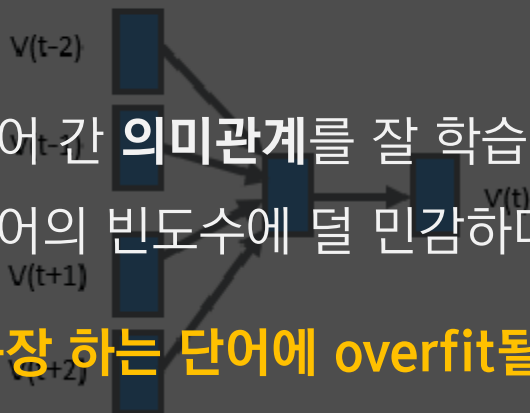
- Embedding

## Word2Vec

### CBOW

target을 기반으로

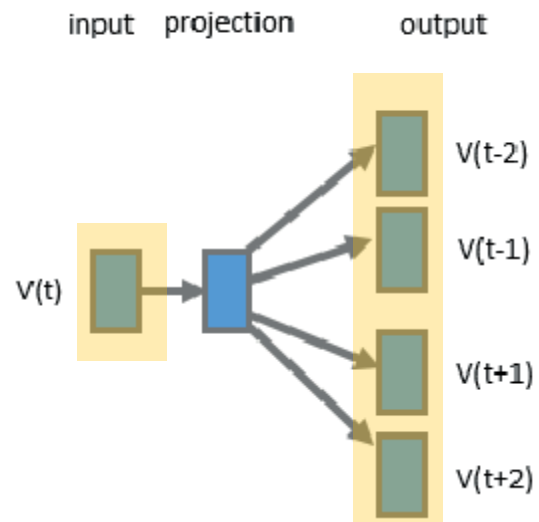
주변 단어들을 예측



- ✓ 단어 간 의미관계를 잘 학습
- ✓ 단어의 빈도수에 덜 민감하다

자주 등장 하는 단어에 overfit될 위험 ↓

### Skip-gram

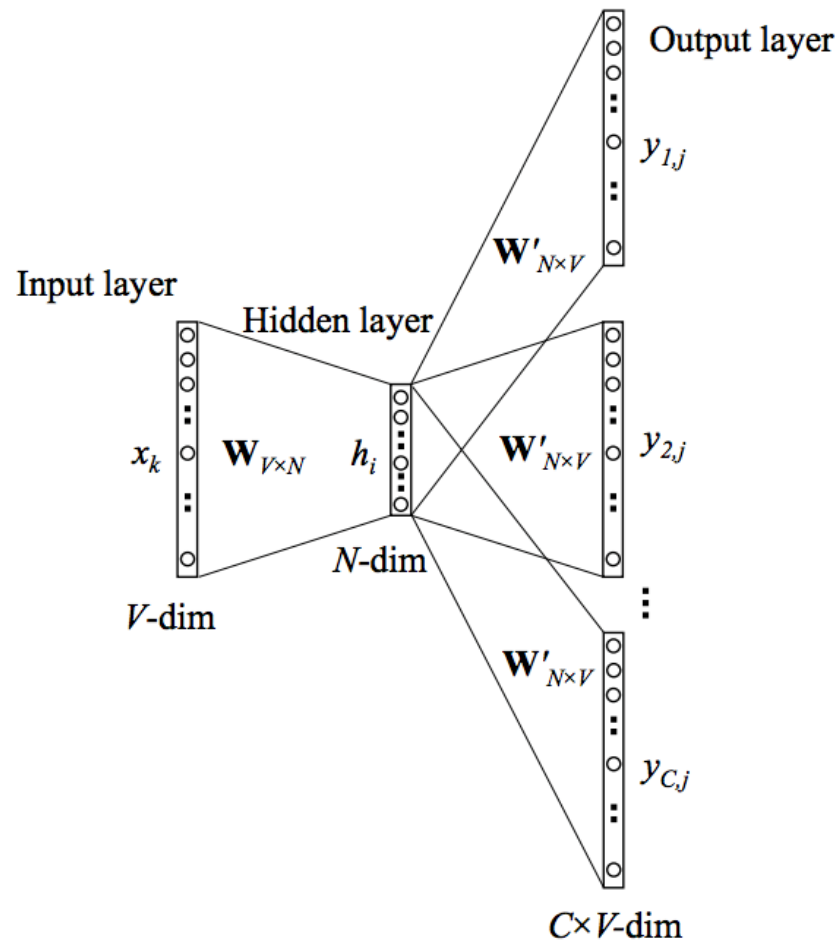




# 1 자연어의 데이터화

- Embedding

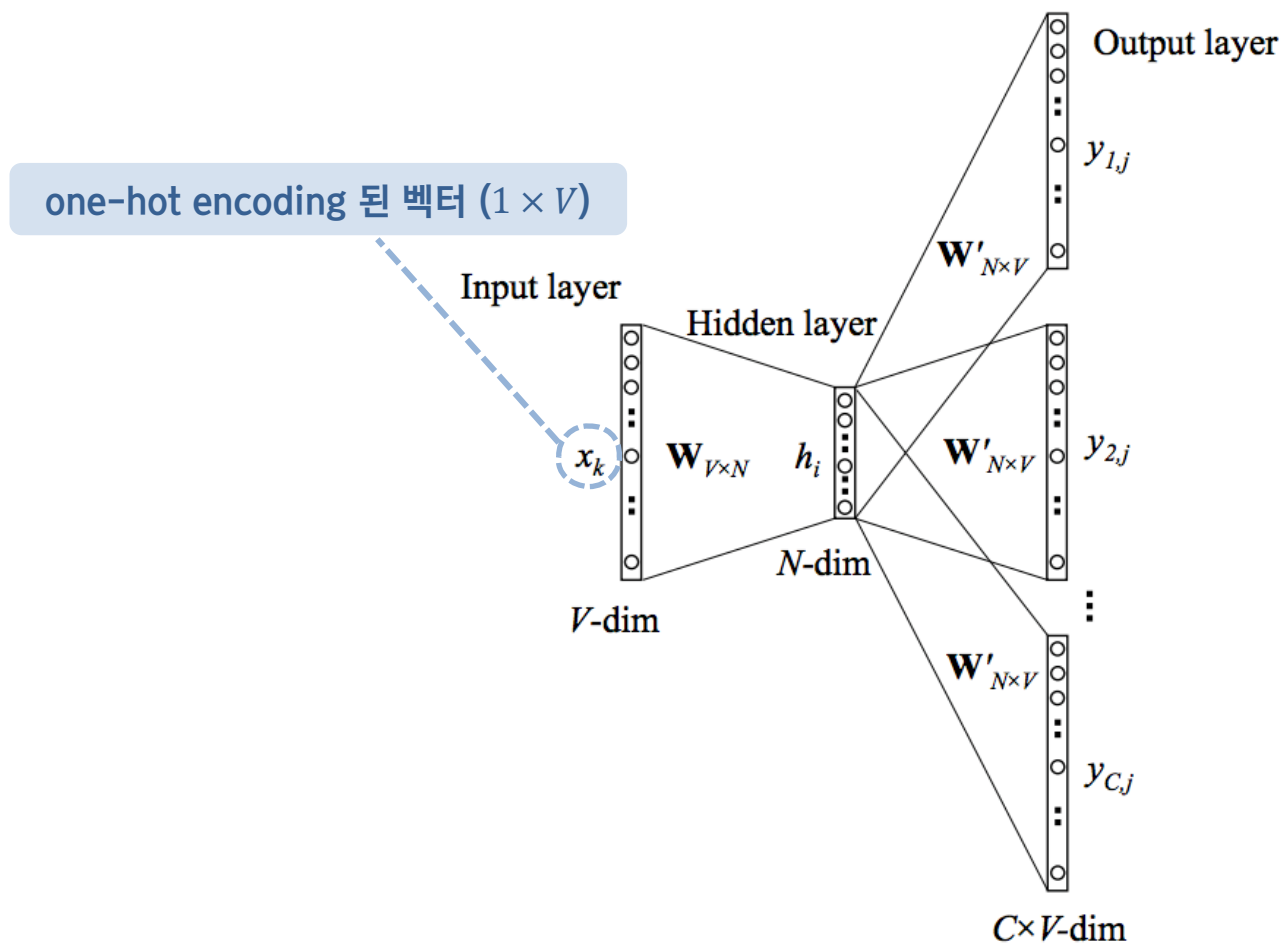
## Skip-gram의 요소



# 1 자연어의 데이터화

- Embedding

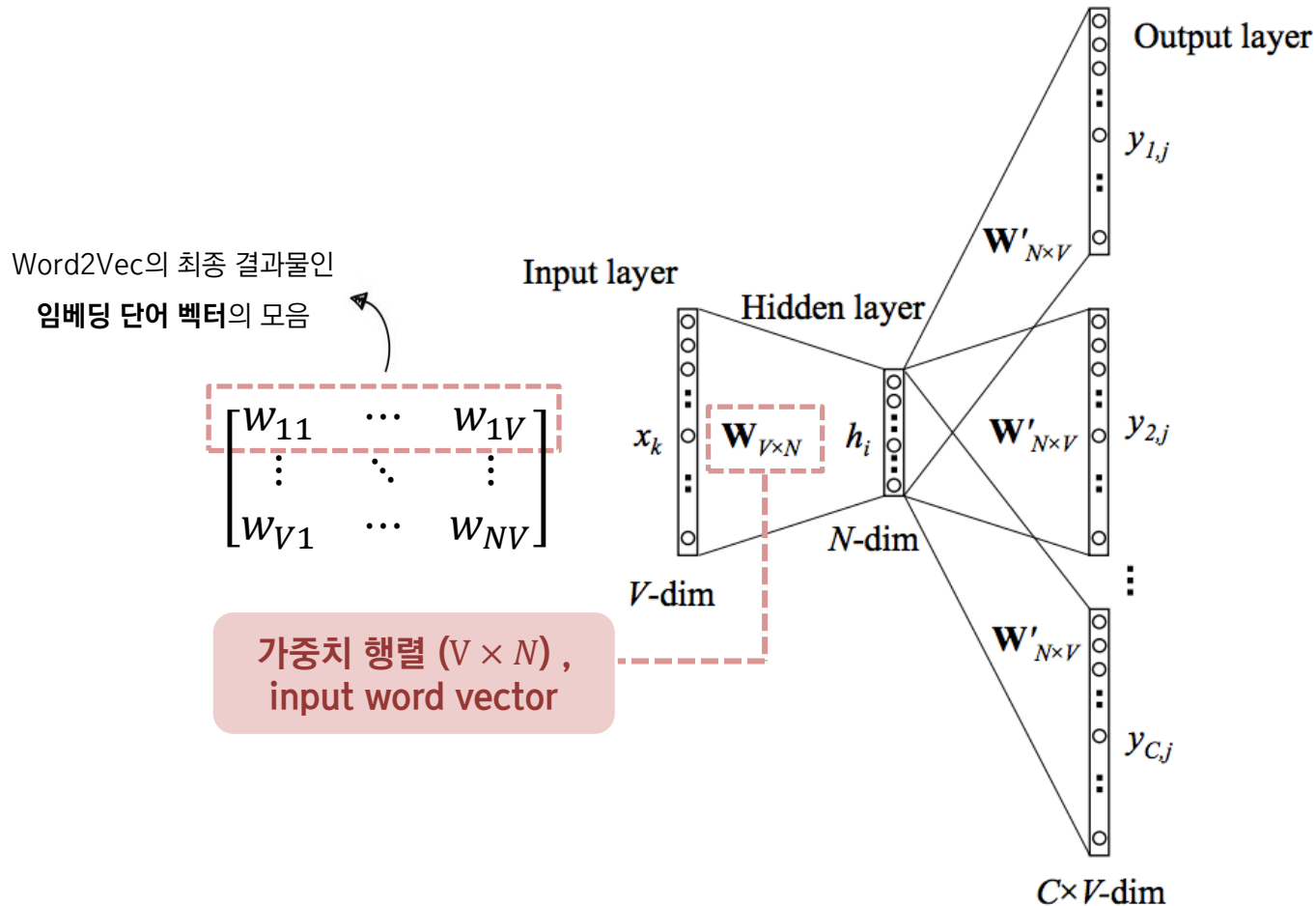
## Skip-gram의 요소



# 1 자연어의 데이터화

- Embedding

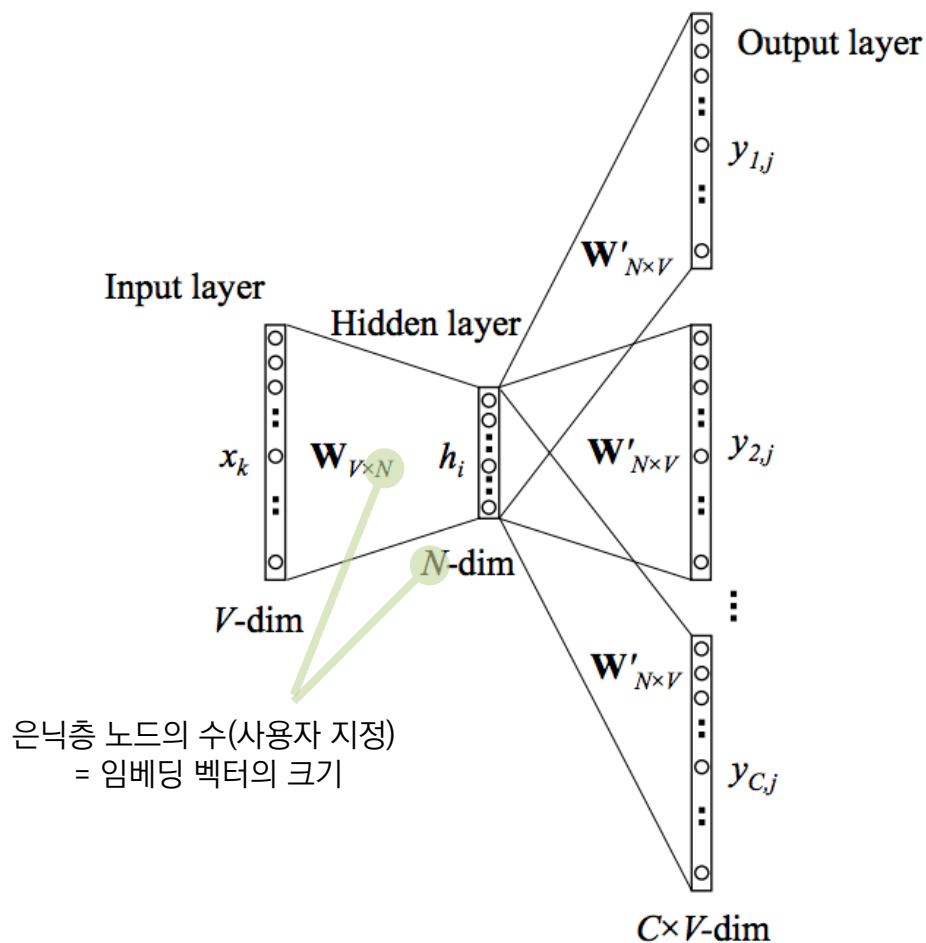
## Skip-gram의 요소



# 1 자연어의 데이터화

- Embedding

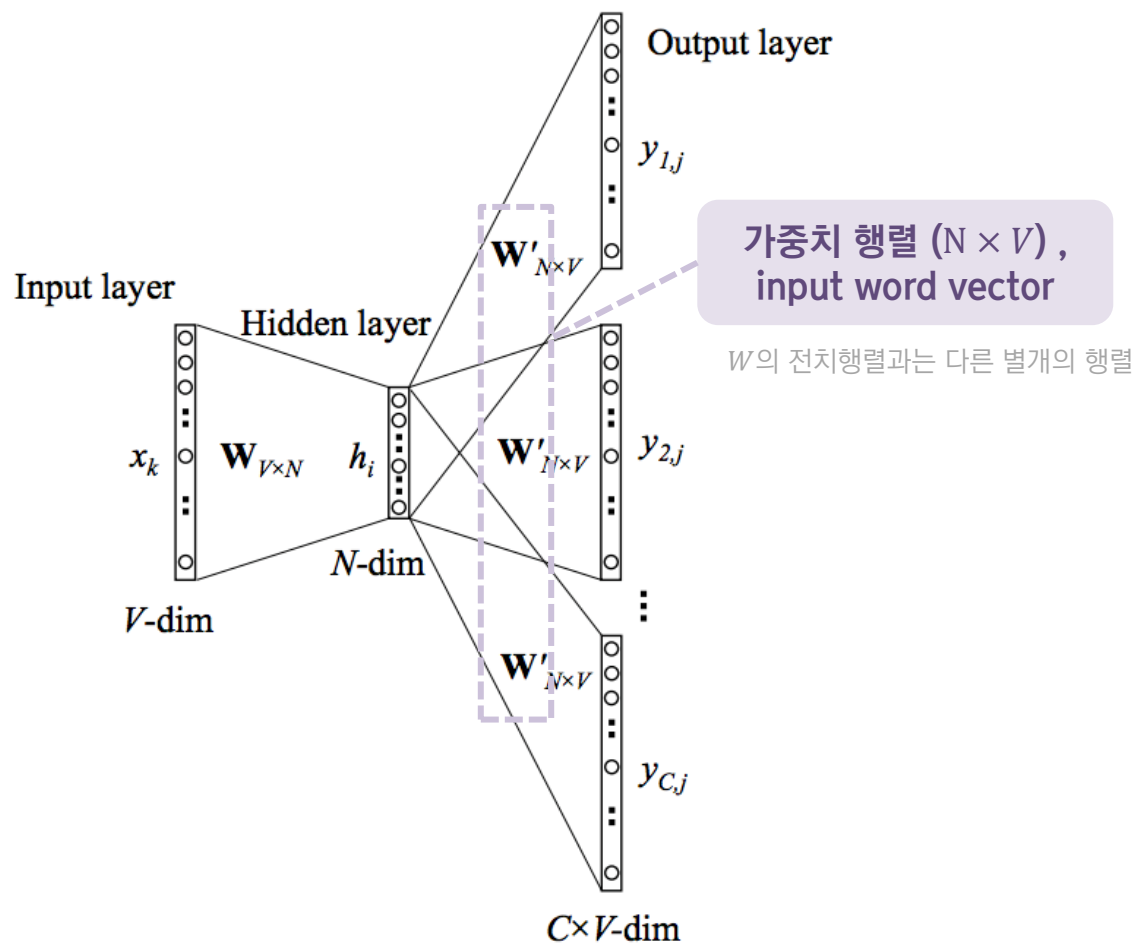
## Skip-gram의 요소



# 1 자연어의 데이터화

- Embedding

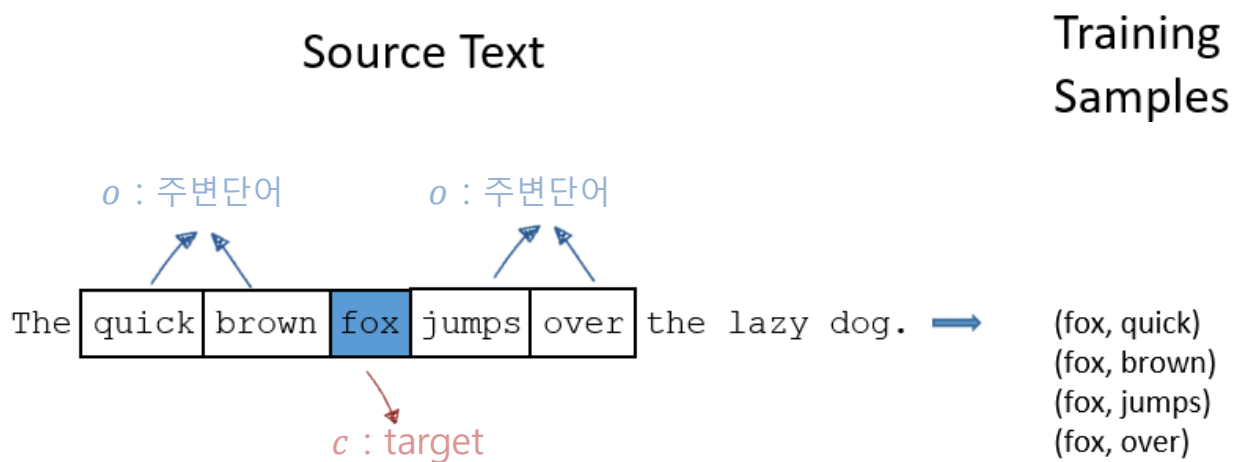
## Skip-gram의 요소



# 1 자연어의 데이터화

- Embedding

## Skip-gram의 목표



한 쌍의 단어가 문맥적으로 연계된 것인지 예측

# 1 자연어의 데이터화

- Embedding

## Skip-gram의 학습

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)}$$

$o$ : 주변 단어

$c$ : 중심 단어

$v$ :  $W$ 의 행벡터

$u$ :  $W'$ 의 열벡터

위 식을 최대화하는 방향으로 학습한다  
분모를 줄이고, 분자를 키운다

# 1 자연어의 데이터화

- Embedding

## Skip-gram의 학습

$$\uparrow P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)} \downarrow$$

$o$ : 주변 단어  
 $c$ : 중심 단어  
 $v$ :  $W$ 의 행벡터  
 $u$ :  $W'$ 의 열벡터

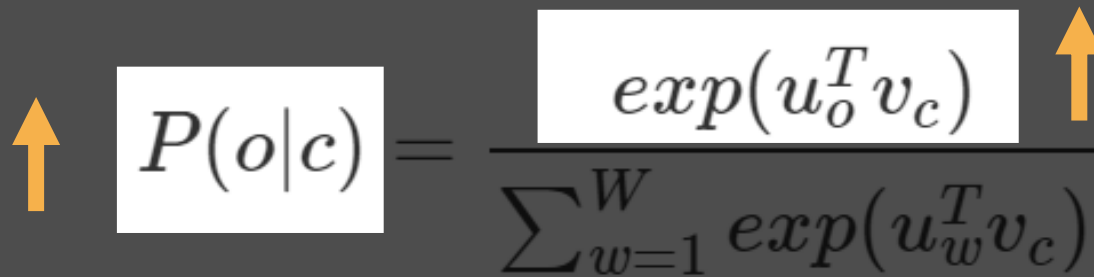
위 식을 최대화하는 방향으로 학습한다.  
원도우 크기 내에 등장하지 않는 단어와  
중심단어의 유사도를 감소시킨다.  
분모를 줄이고, 분자를 키운다.



# 1 자연어의 데이터화

- Embedding

## Skip-gram의 학습


$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)}$$

The diagram shows the Skip-gram probability formula. An orange arrow points up to the numerator  $\exp(u_o^T v_c)$ , and another orange arrow points up to the denominator  $\sum_{w=1}^W \exp(u_w^T v_c)$ . The variable  $o$  is defined as the surrounding word,  $c$  as the center word,  $v$  as the row vector of  $W$ , and  $u$  as the column vector of  $W'$ .

$o$ : 주변 단어  
 $c$ : 중심 단어  
 $v$ :  $W$ 의 행벡터  
 $u$ :  $W'$ 의 열벡터

중심 단어와 주변 단어 벡터간의 유사도를 높인다  
위 식을 최대화하는 방향으로 학습한다  
분모를 줄이고, 분자를 키운다

# 1 자연어의 데이터화

- Embedding

## Embedding의 효과

1. 단어/문장 데이터의 유사도를 구할 수 있다
2. 의미적/문법적 정보를 압축할 수 있다
3. 전이 학습이 가능하다



사전에 임베딩된 벡터들을 사용할 수 있다

2

seq2seq

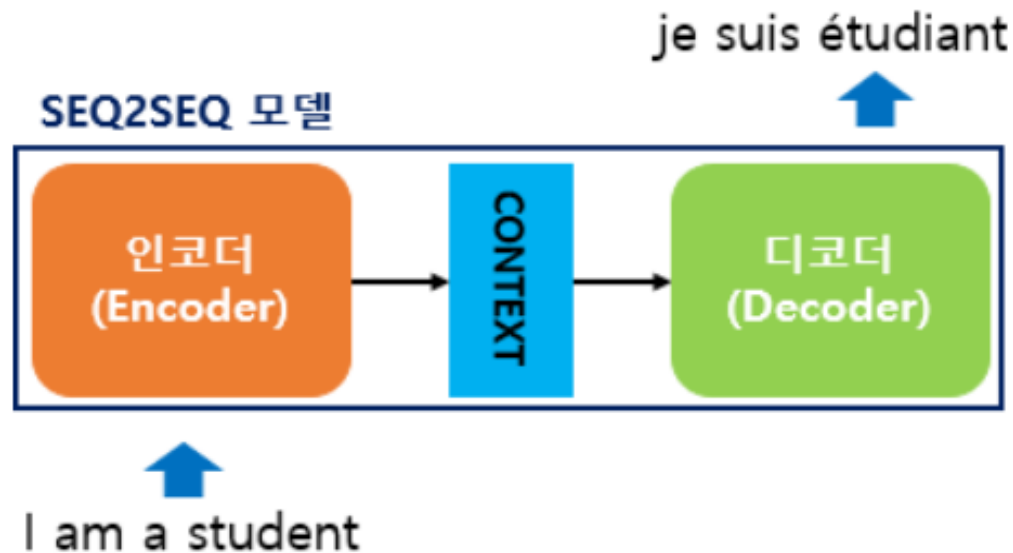
## 2 seq2seq

- seq2seq 이란

# Sequence-to-sequence

: 시퀀스 입력 - 시퀀스 출력

Ex) 기계 번역, 챗봇



인코더-디코더 2개의 RNN 아키텍처를 이어 붙인 모델

## 2 seq2seq

- seq2seq 이란

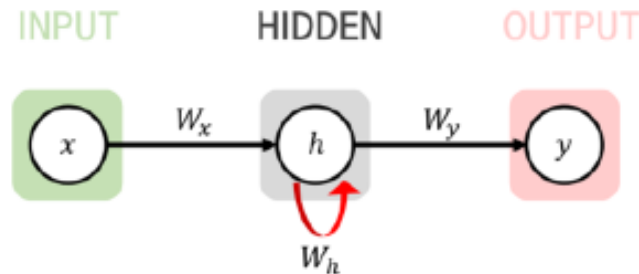
# Sequence-to-sequence Review!



시퀀스 입력 -> 시퀀스 출력  
Ex) 기계 번역, 챗봇

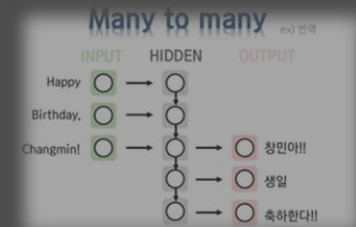
### 순환신경망(Recurrent Neural Network)

: sequential data에 적합  
ex) 자연어, 음성신호, 주식



전시점의 은닉층 출력값이 다시 입력값으로 작용하는 모델

인코더-디코더 2개의 RNN 아키텍처를 이어 붙인 모델

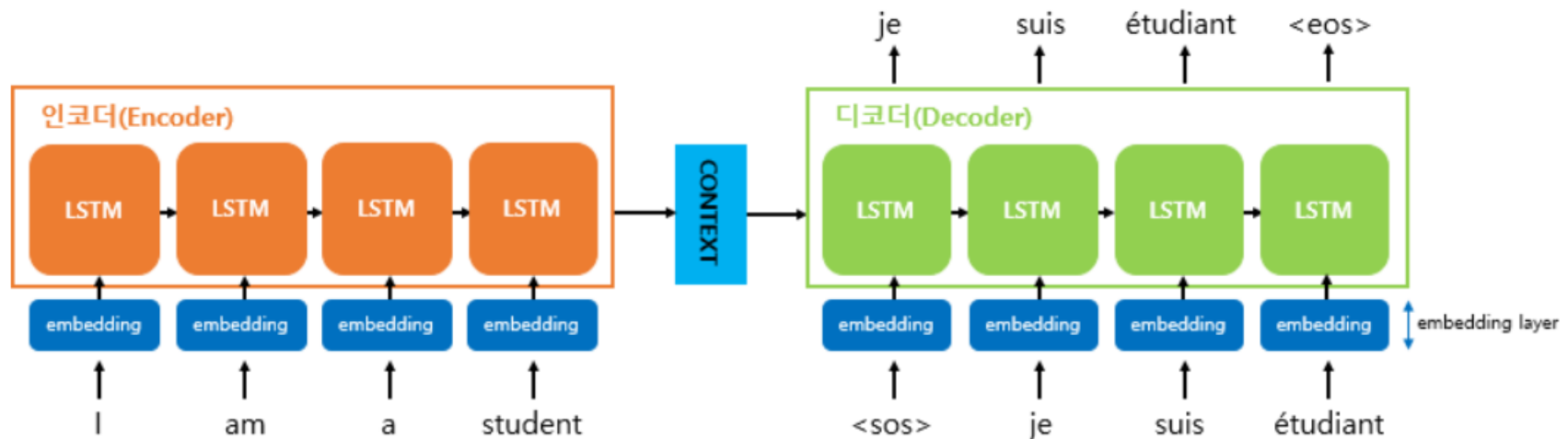


지난주에 배웠던 창민아 생일 축하한다 seq2seq 맞습니다,,

## ● seq2seq 이란

# Seq2seq의 구조

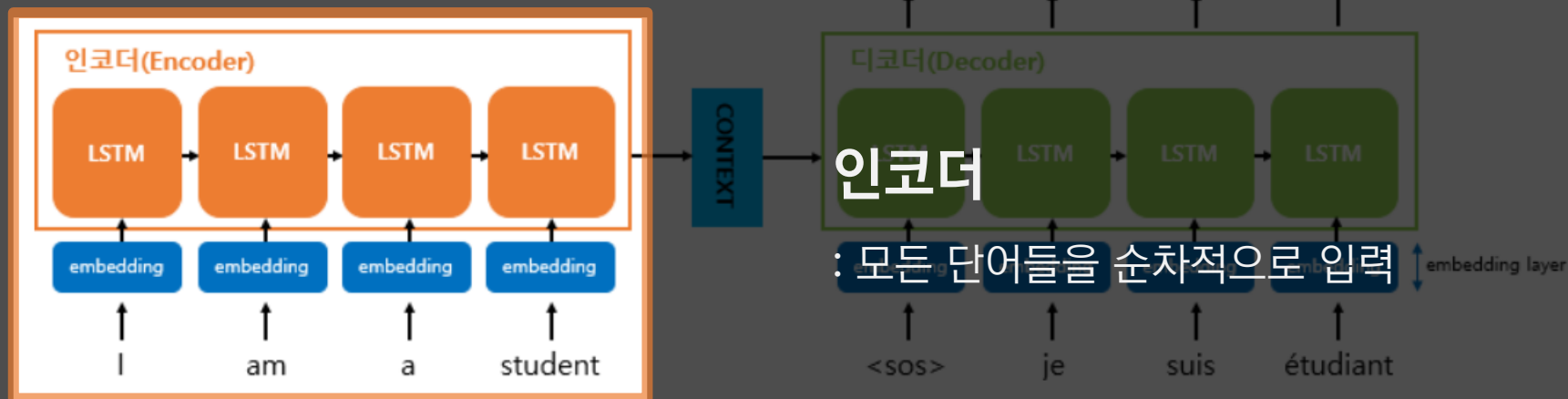
인코더- 컨텍스트벡터- 디코더



● seq2seq 이란

## Seq2seq의 구조

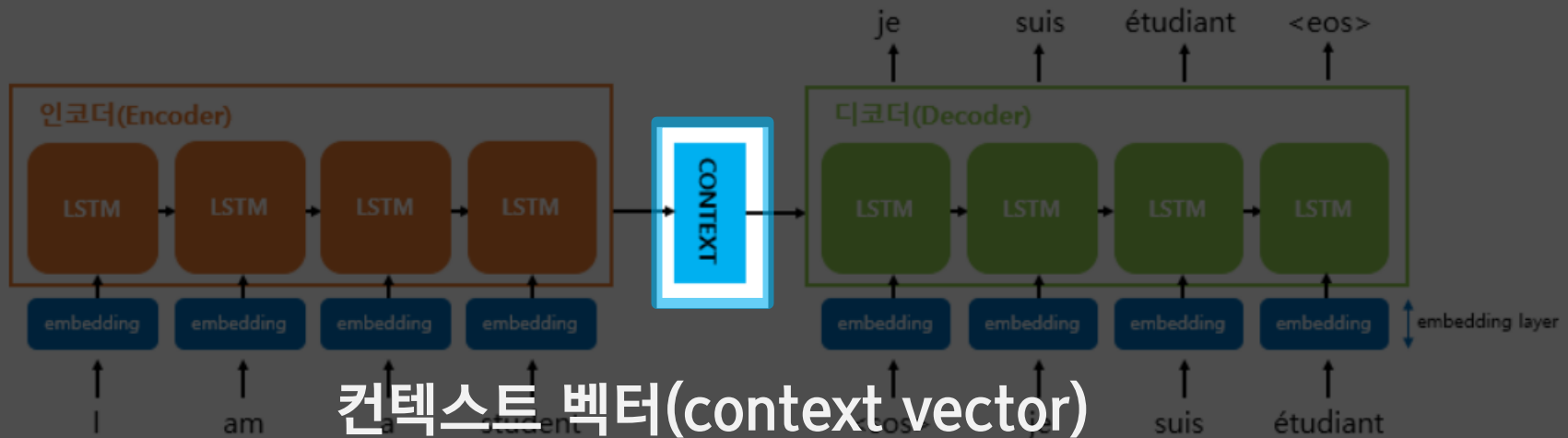
인코더- 컨텍스트 벡터- 디코더



- seq2seq 이란

# Seq2seq의 구조

인코더- 컨텍스트 벡터- 디코더



**컨텍스트 벡터(context vector)**

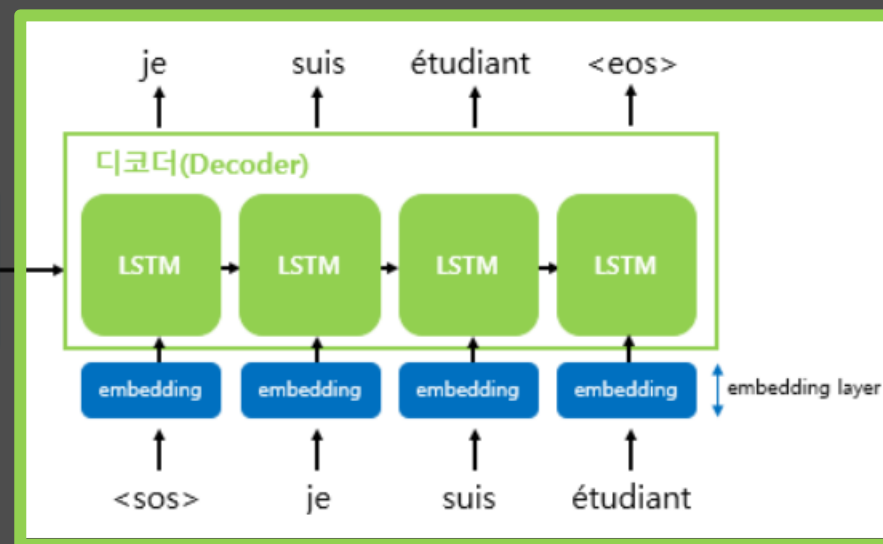
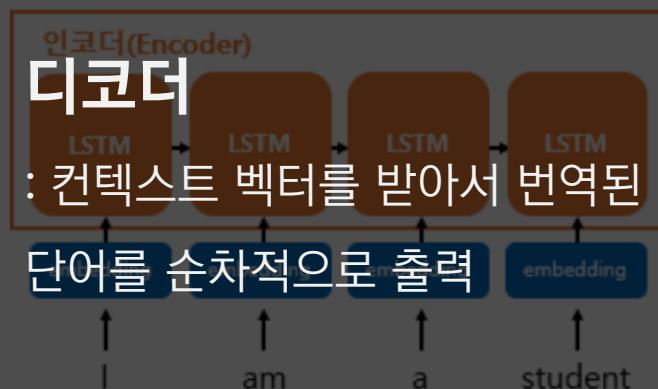
: 모든 입력 문장의 정보를 하나의 컨텍스트 벡터로 압축



## ● seq2seq 이란

# Seq2seq의 구조

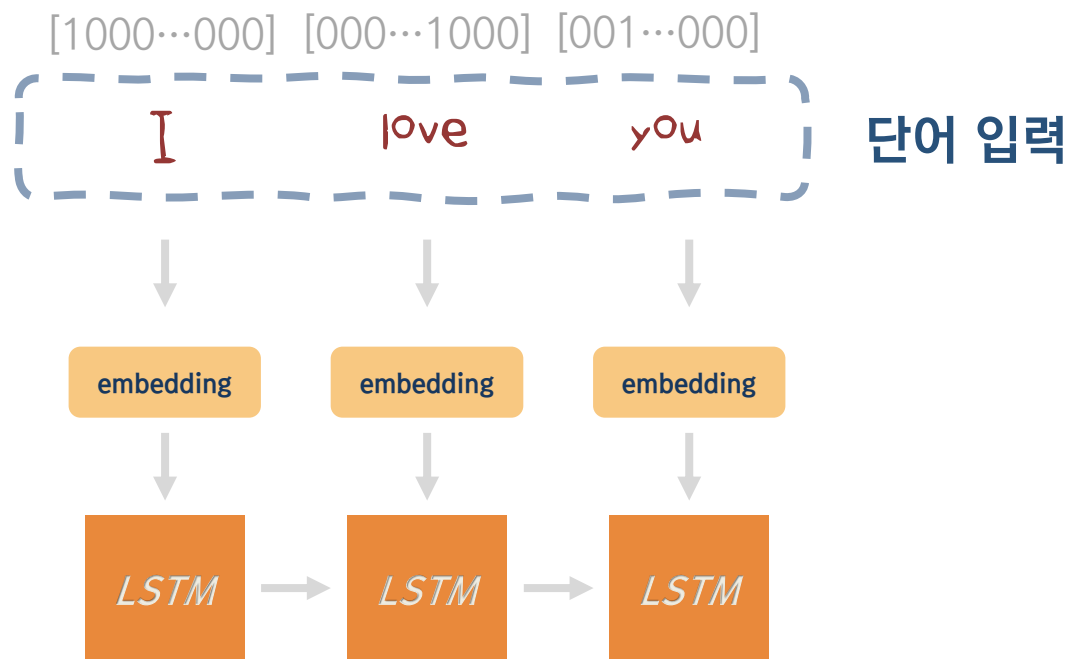
인코더- 컨텍스트 벡터- 디코더



- seq2seq의 구조

## 1) Encoder

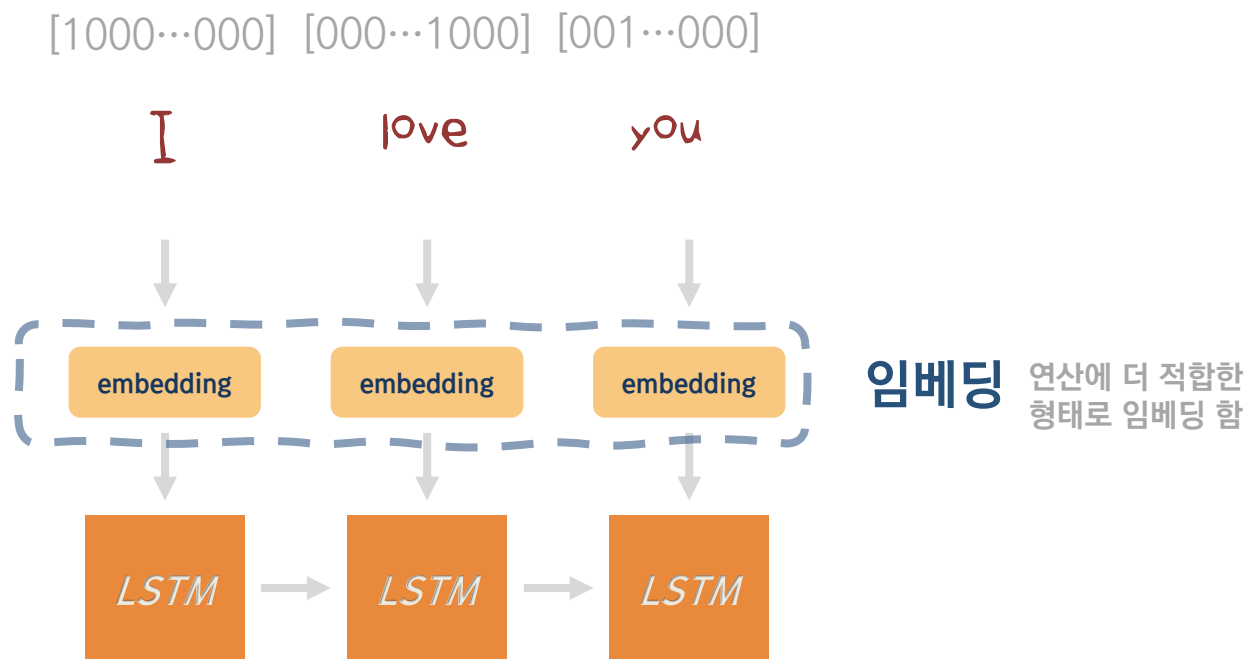
: 순차적으로 단어 입력



## ● seq2seq의 구조

### 1) Encoder

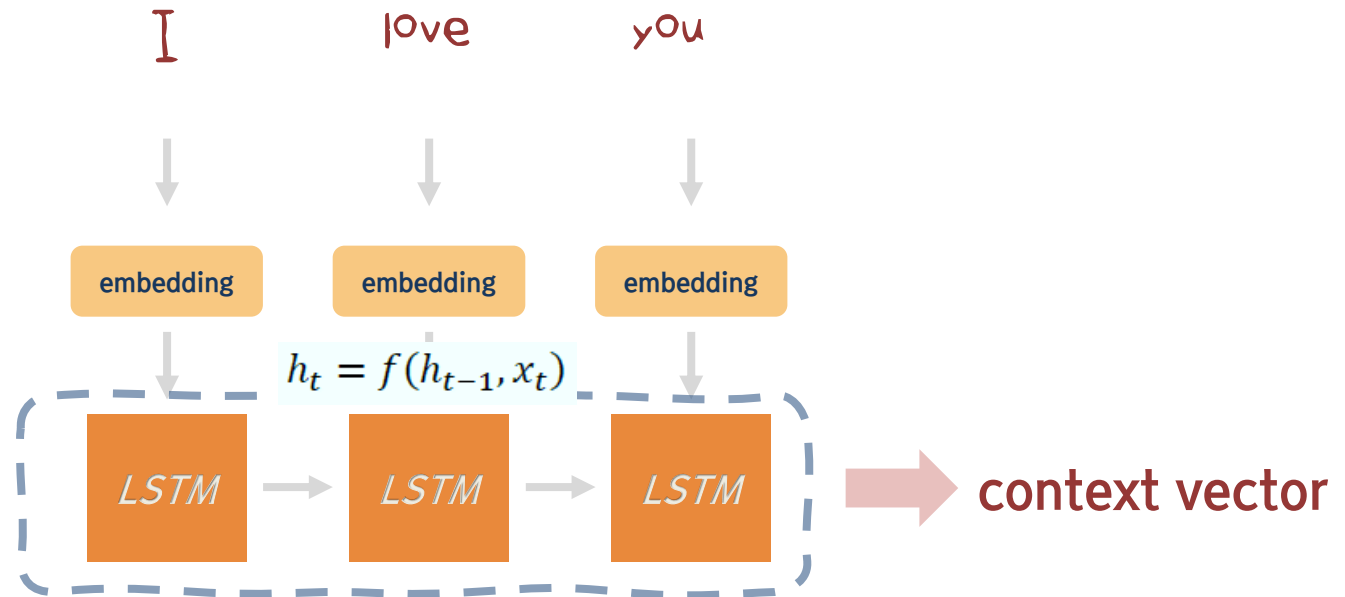
: 순차적으로 단어 입력



- seq2seq의 구조

### 1) Encoder

: 순차적으로 단어 입력



- seq2seq의 구조

## 2) Context vector



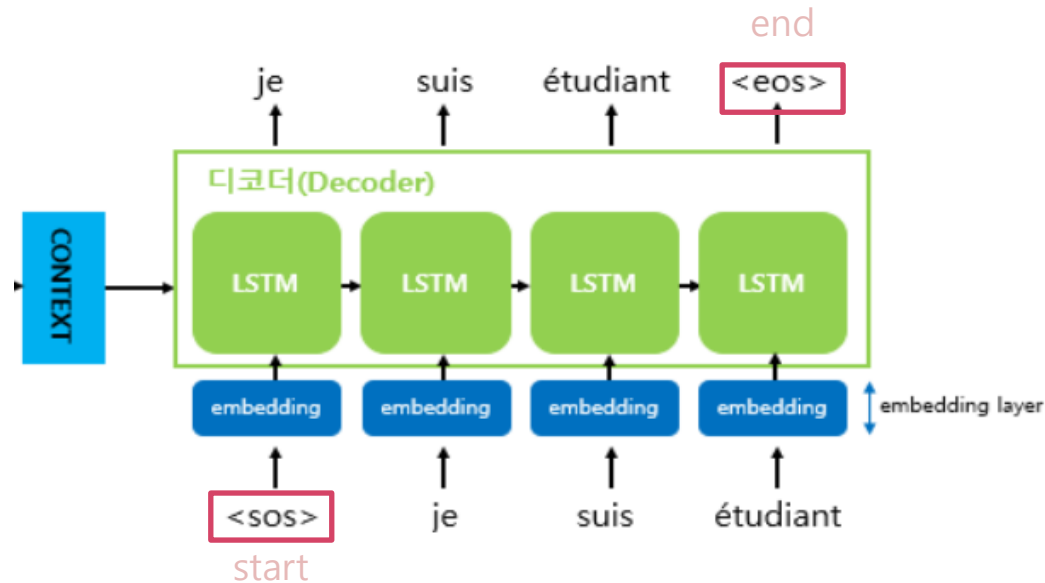
encoder에 들어온 모든 정보들을 압축해서 담은 벡터

**고정된 길이의 벡터**

- seq2seq의 구조

### 3) Decoder

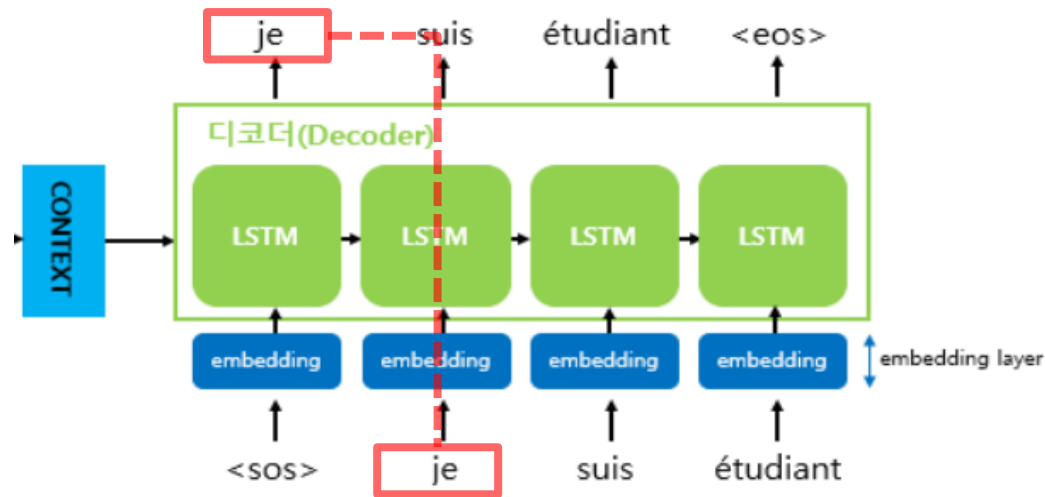
:번역된 단어를 순차적으로 출력



Context vector를 바탕으로 item by item으로 출력값 생성

- seq2seq의 구조

### 3) Decoder



$$P(y_t | y_{t-1}, y_{t-2}, \dots, y_1, c) = g(h_t, y_{t-1}, c)$$

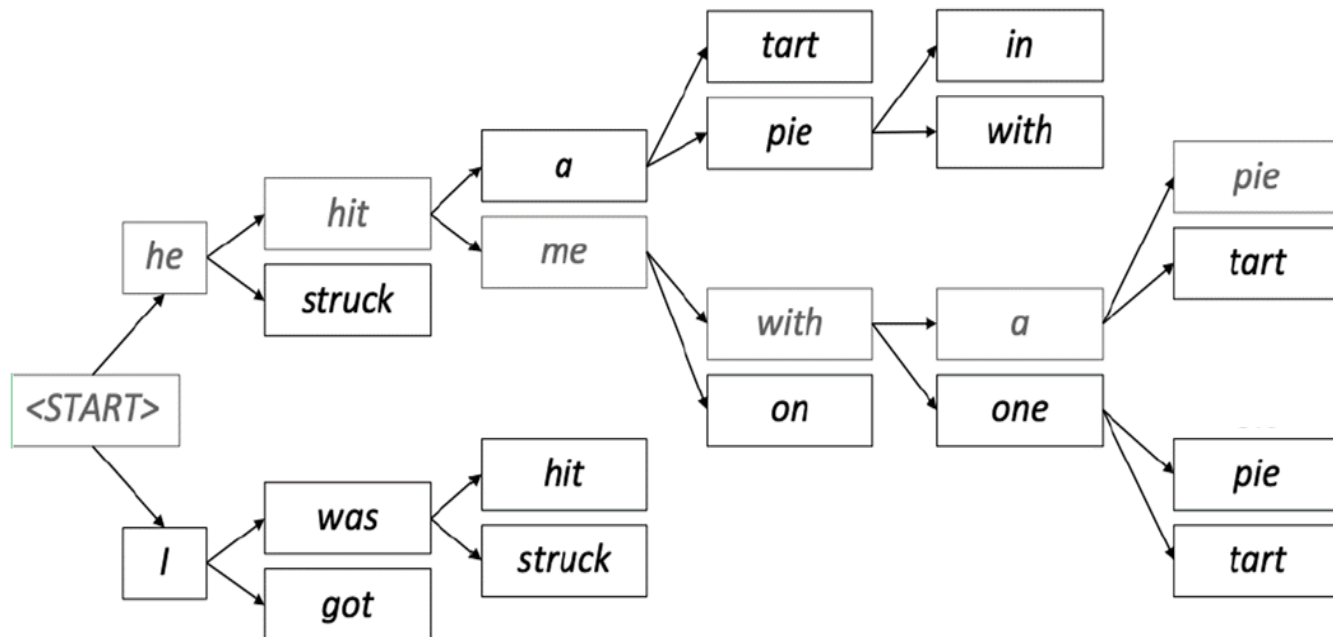
이전 시점의 출력값 = 다음 시점의 입력값

- seq2seq 디코딩

# Beam Search Decoding



:매 스텝마다 가장 확률이 높은 k개의 다음 단어 선택, 확률 예측



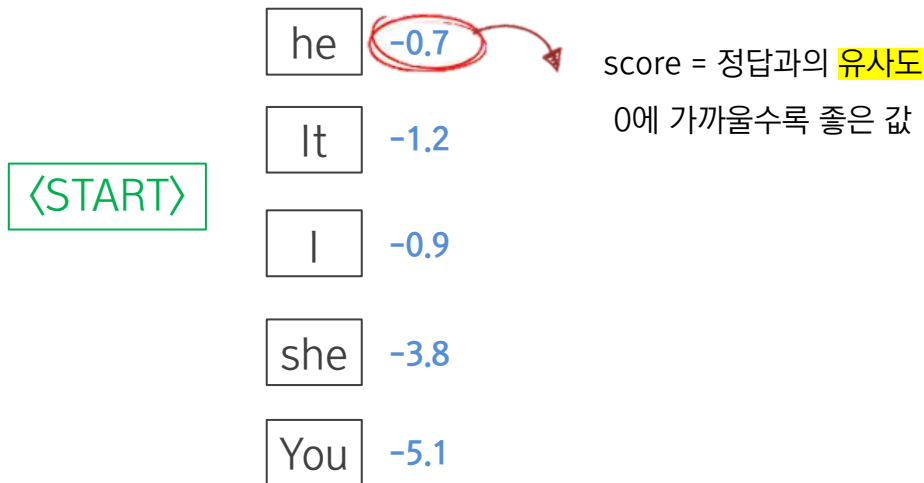
Beam size =  $k = 2$



- seq2seq 디코딩

# Beam Search Decoding

1. 현재 스텝에서 모든 단어들에 대해서 score 값을 계산

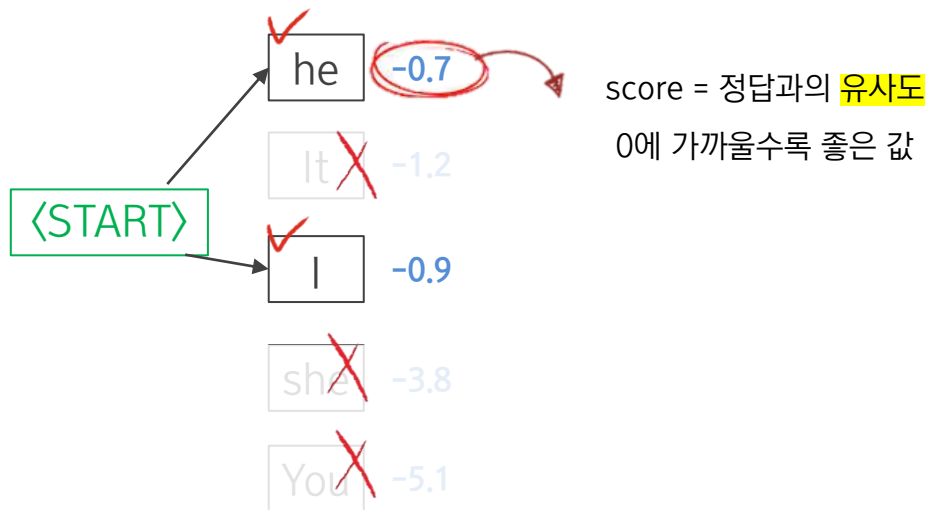


Beam size =  $k = 2$

## ● seq2seq 디코딩

# Beam Search Decoding

2. score 값이 높은 k 개의 단어를 선택 후 나머지 단어는 제거



Beam size = k = 2

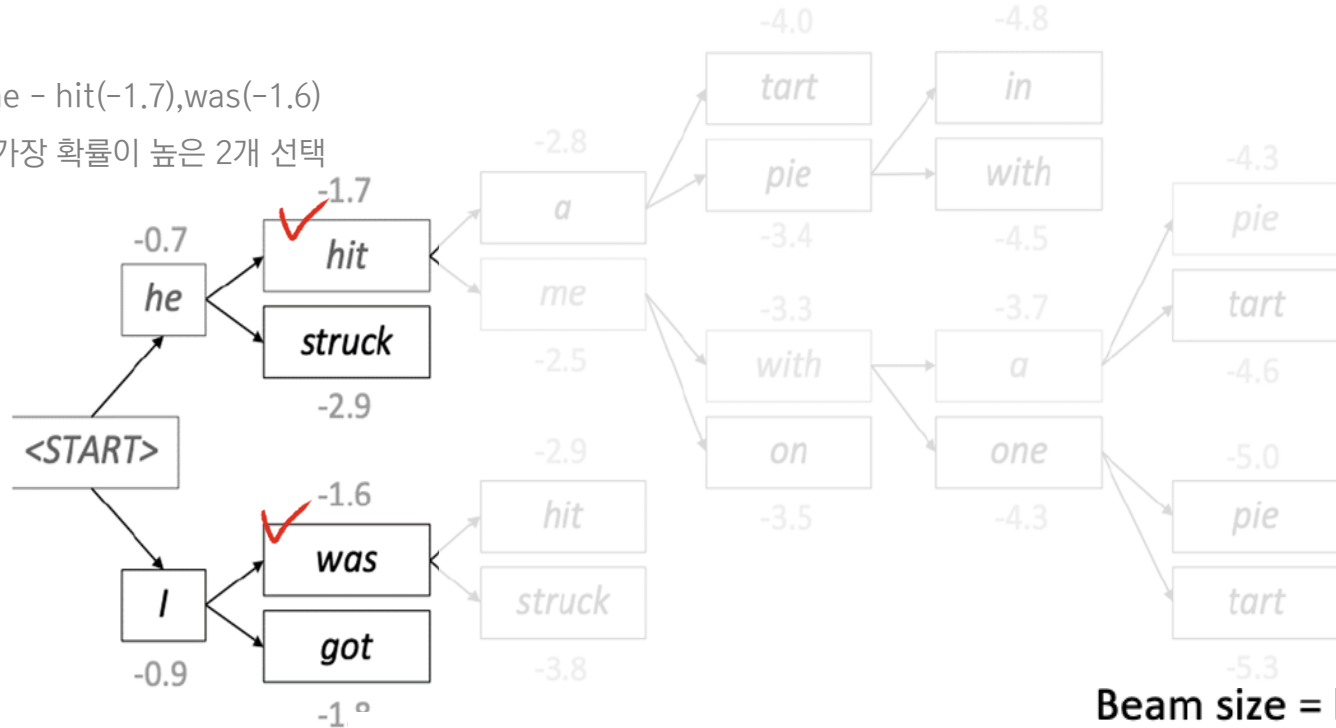
- seq2seq 디코딩

## Beam Search Decoding

3. 다음 스텝의 모든 단어들에 대해서 score 값을 계산 하고 score가 높은 k 개를 선택

ex) he - hit(-1.7), was(-1.6)

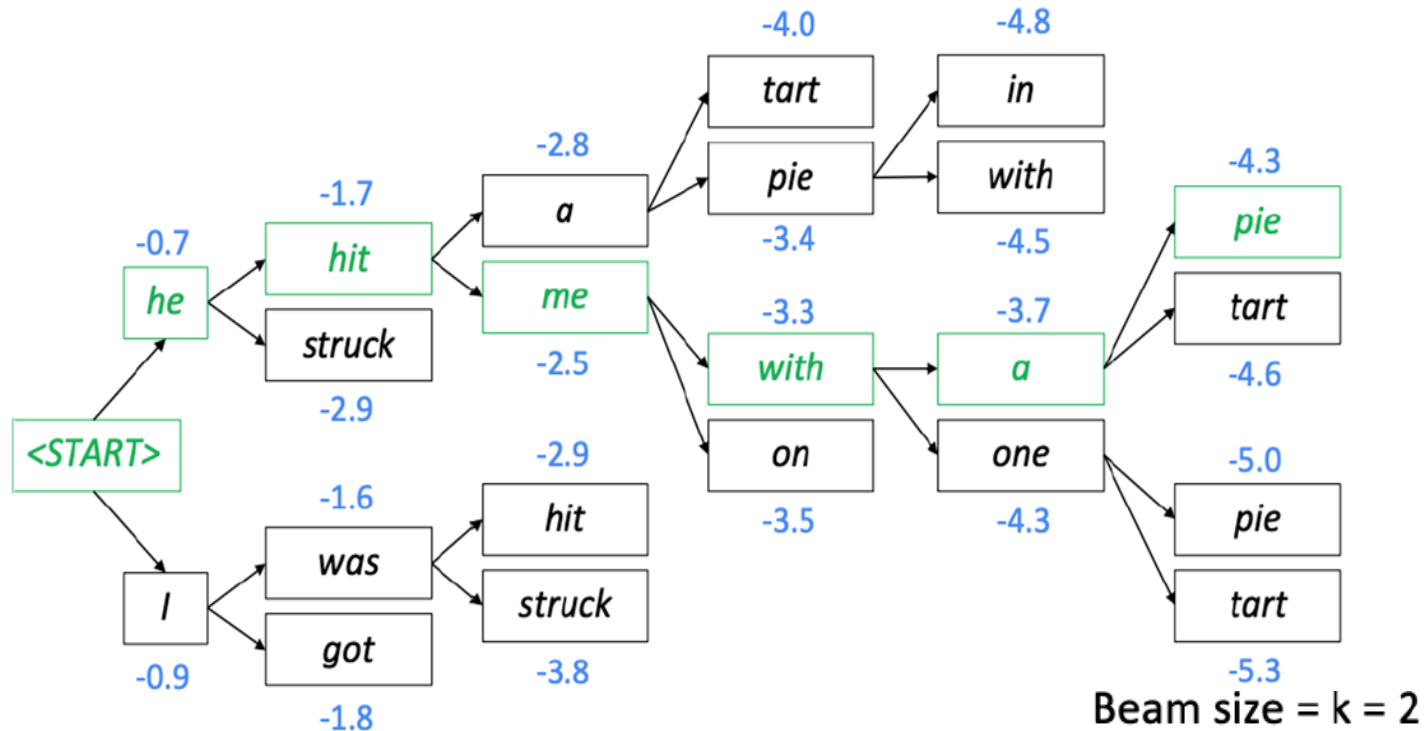
→ 가장 확률이 높은 2개 선택



- seq2seq 디코딩

# Beam Search Decoding

4. 매 스텝마다 k 개의 후보군을 유지하며 <END> 토큰이 선택될 때 까지 반복

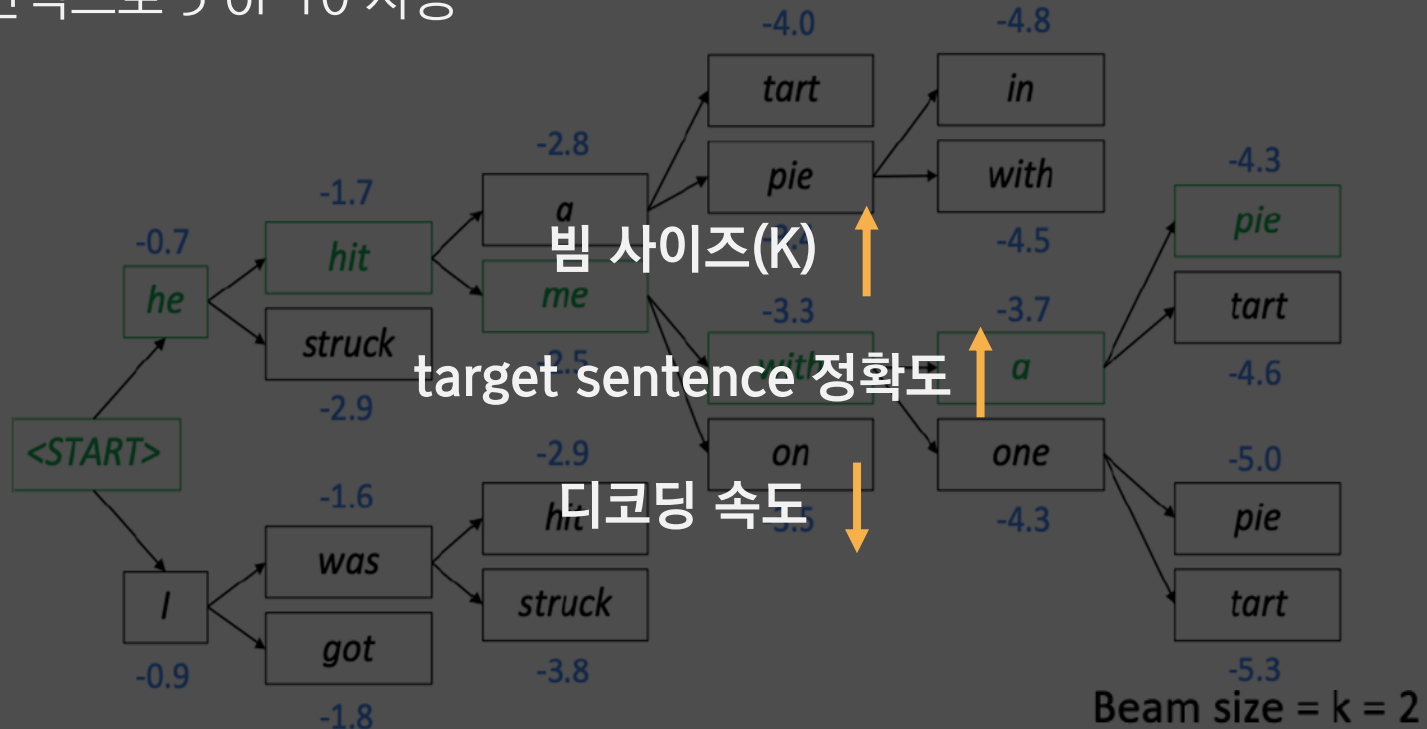


## ● seq2seq 디코딩

# Beam Search Decoding

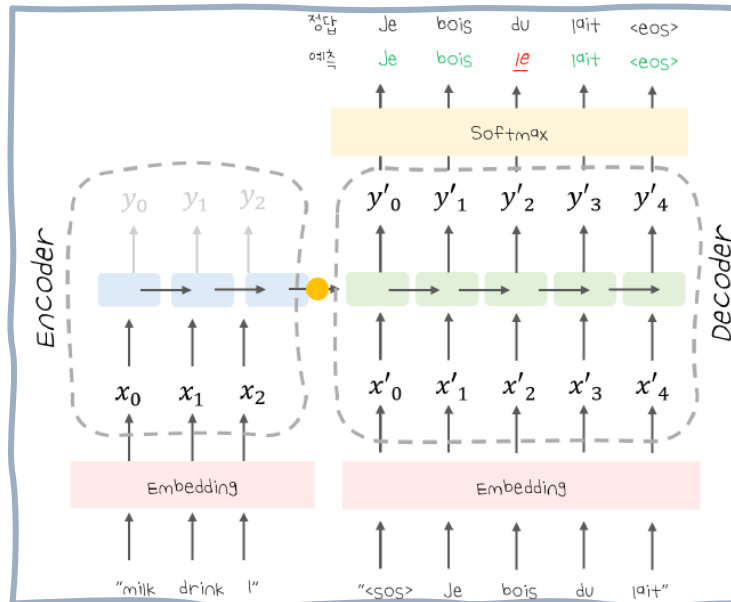
4. 매 스텝마다 k 개의 후보군을 유지하며 <END> 토큰이 선택될 때 까지 반복

K: 일반적으로 5 or 10 사용



## ● seq2seq 프로세스

### How to train?



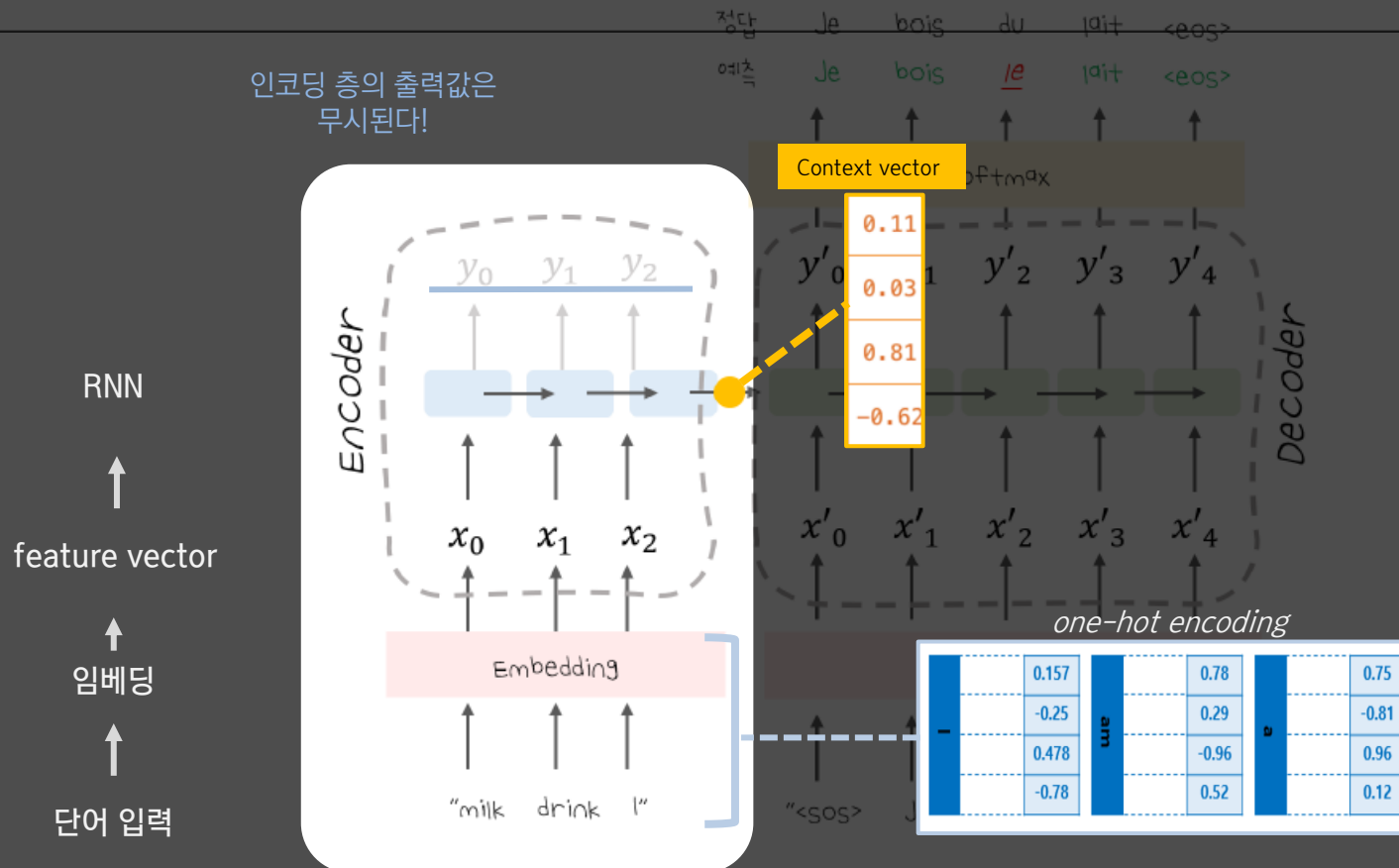
seq2seq

하나의 시스템으로  
최적화 되어있는 모델

END-TO-END

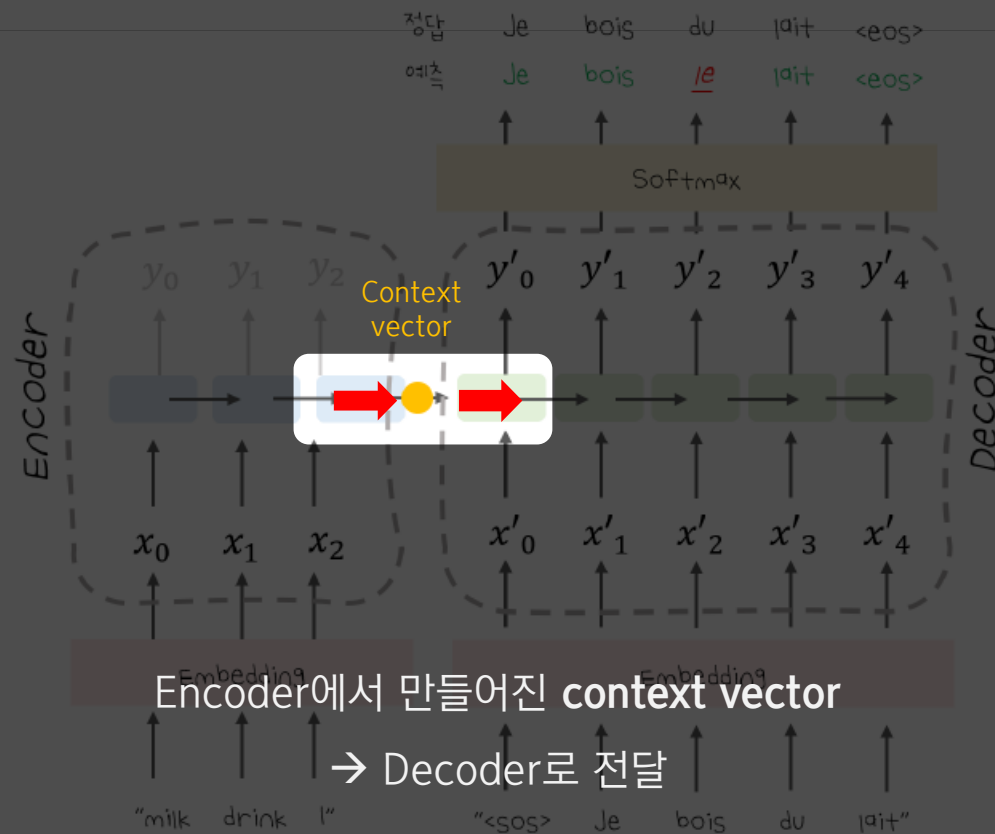
- seq2seq 프로세스

## How to train?



- seq2seq 프로세스

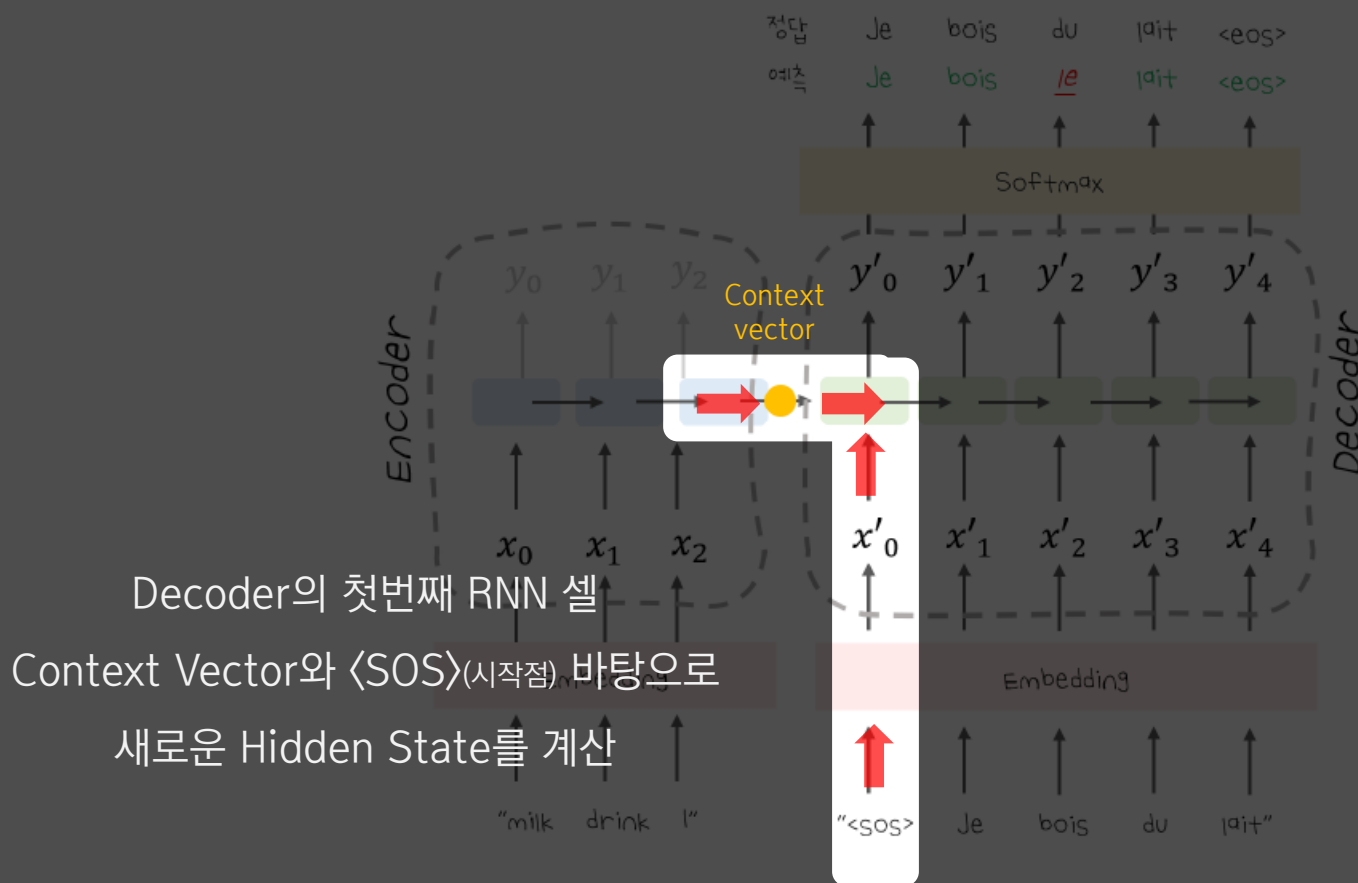
## How to train?





- seq2seq 프로세스

## How to train?



- seq2seq 프로세스

## How to train?



Affine 계층과 Softmax 계층을 거쳐 출력값 생성

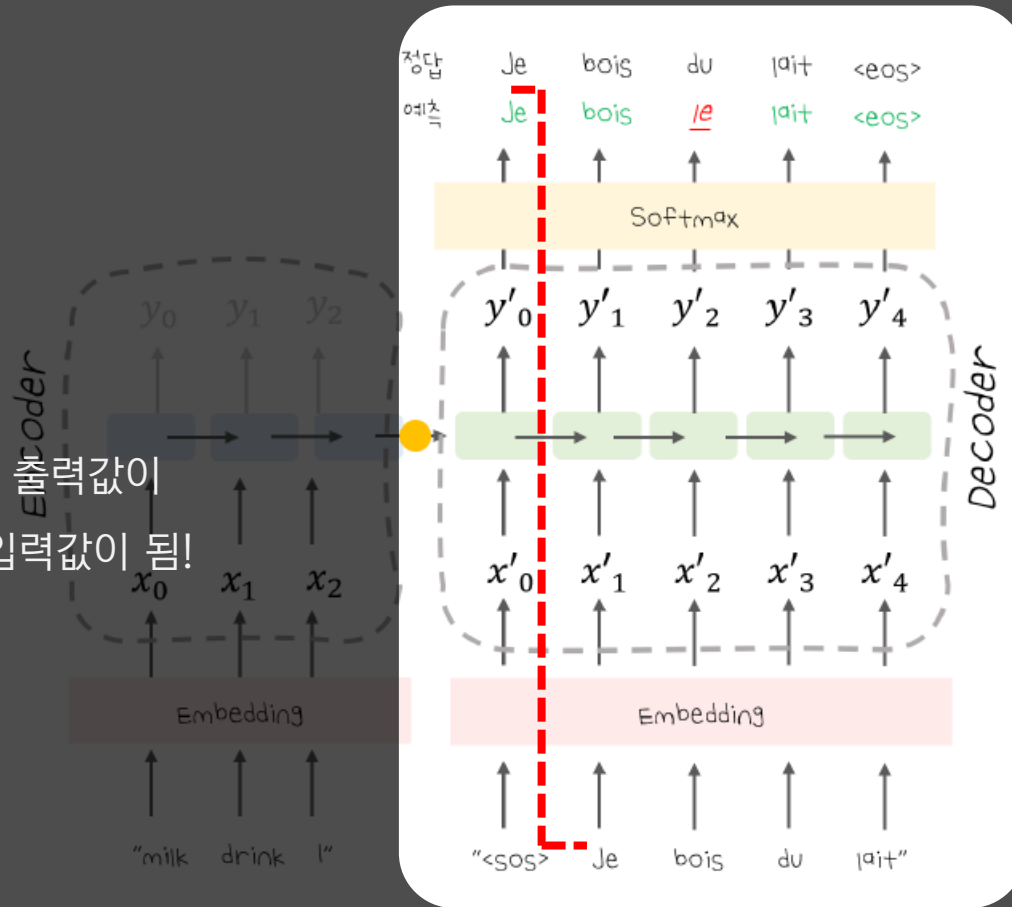
"milk drink I"

"<eos> Je bois du lait"

- seq2seq 프로세스

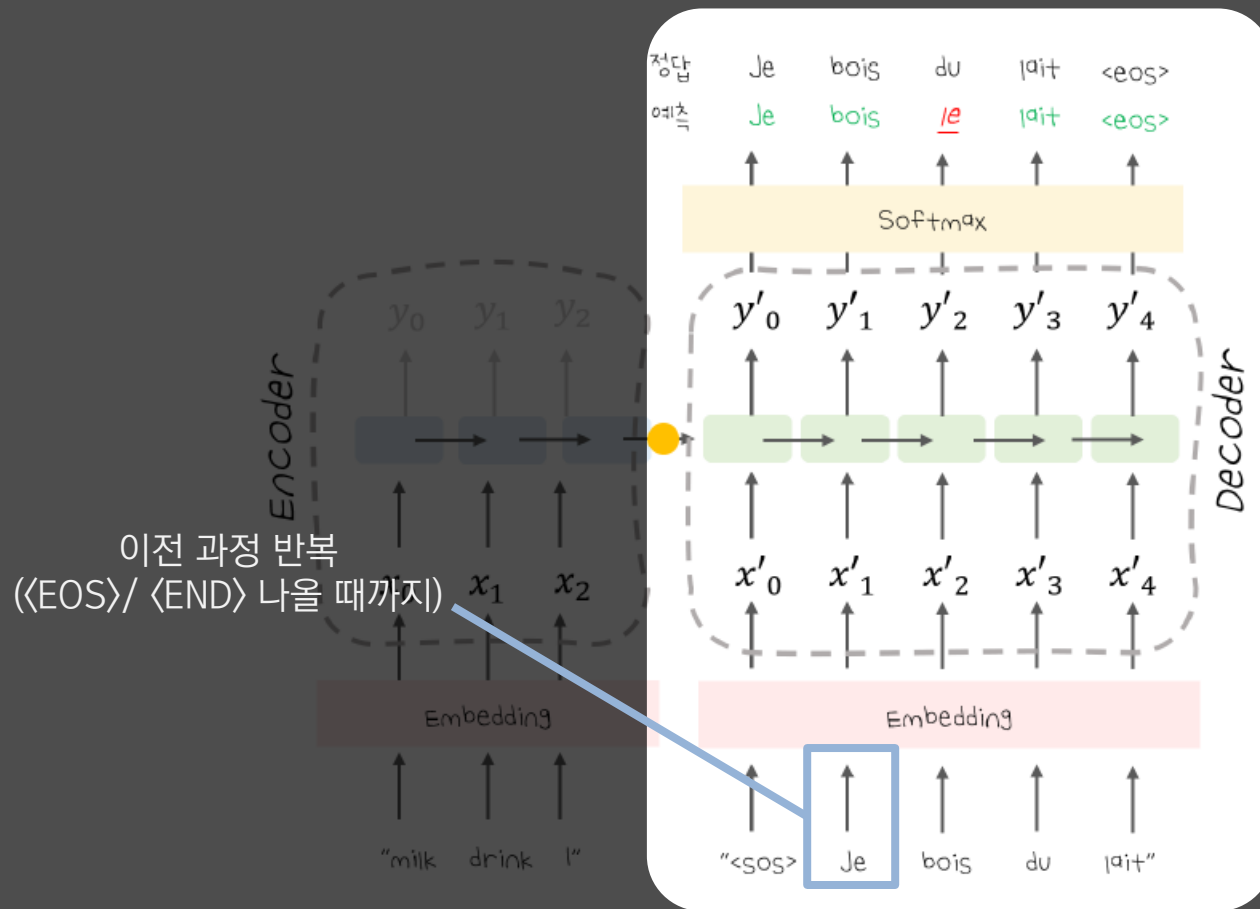
## How to train?

이전 시점의 출력값이  
다음 시점의 입력값이 됨!



## ● seq2seq 프로세스

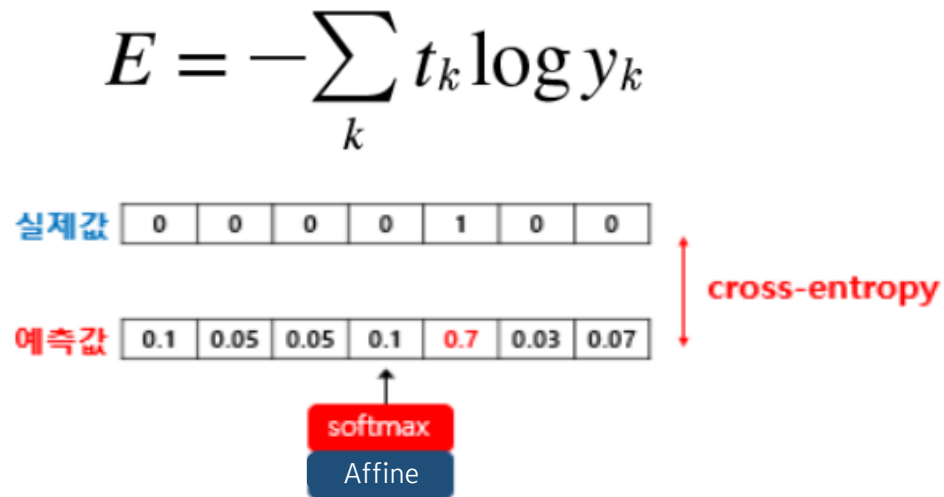
### How to train?



## ● seq2seq 프로세스

# How to train?

:오차



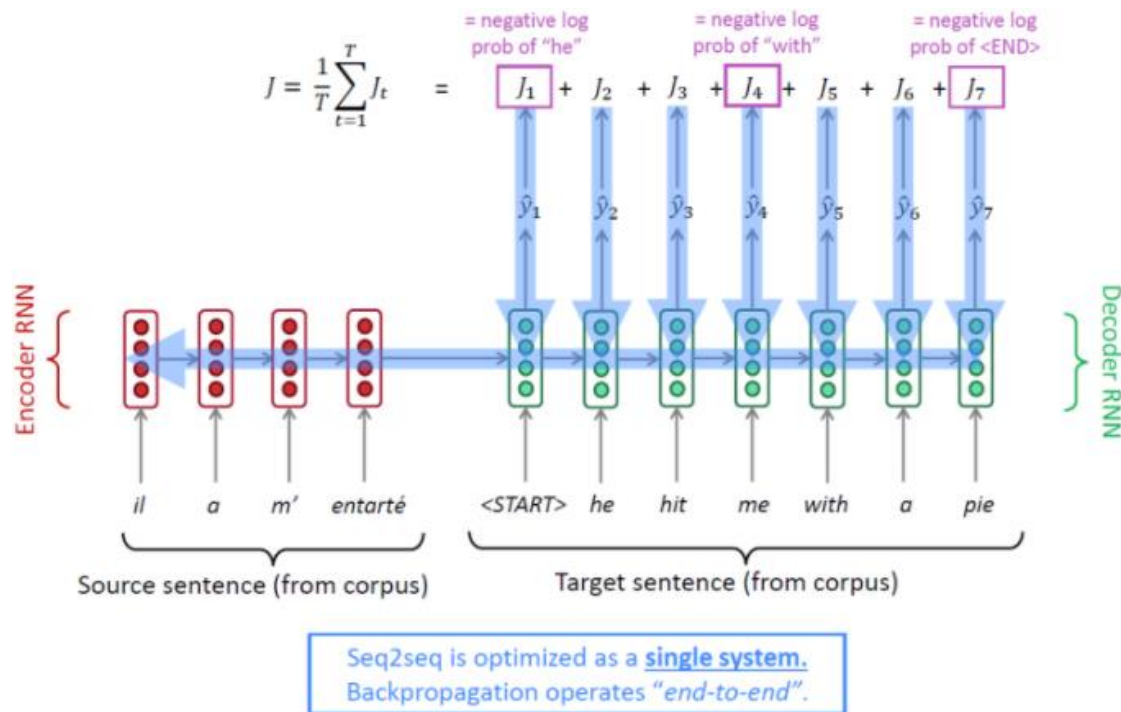
예측한 문장과 정답 문장 사이의 오차

→ 교차 엔트로피로 계산

- seq2seq 프로세스

# How to train?

:역전파



RNN, LSTM처럼 전체 시스템에 걸쳐서 역전파

→ end-to-end

3

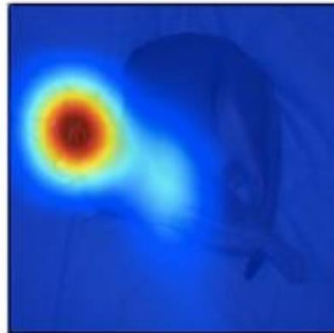
**Attention**

- Attention Mechanism

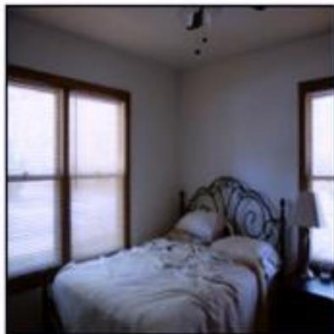
## Attention Mechanism



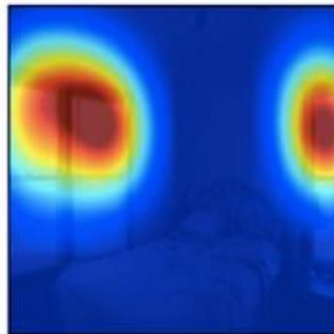
What color are the animal's eyes? green



Human Attention



What is covering the windows? blinds



Human Attention

### Attention의 아이디어

사람의 시각적인 능력에서 아이디어 얻음

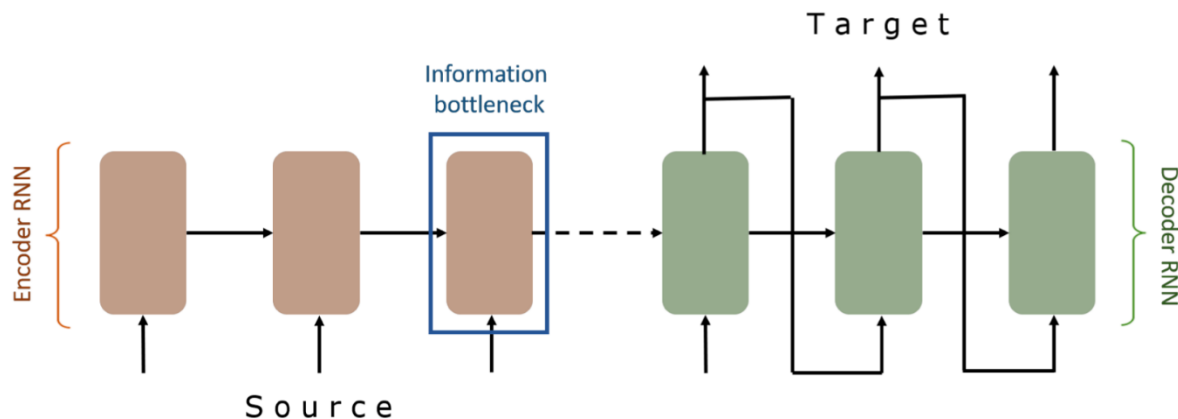


많은 정보 중 **중요한 부분에 Attention**



- Attention Mechanism

## Seq2Seq의 문제점



### Bottleneck Problem (병목 현상)

Seq2seq에서는 마지막 스텝의 hidden state인 context vector를 **decoder**에 넘겨줌

단일 벡터가 **모든 source sentence의 정보**를 담고 있어야 함



**마지막 스텝에서 모든 정보를 인코딩하여 정보가 쏠리는 현상 발생**

## ● Attention Mechanism

# Attention의 등장

Seq2Seq

1. 장기 의존성 문제

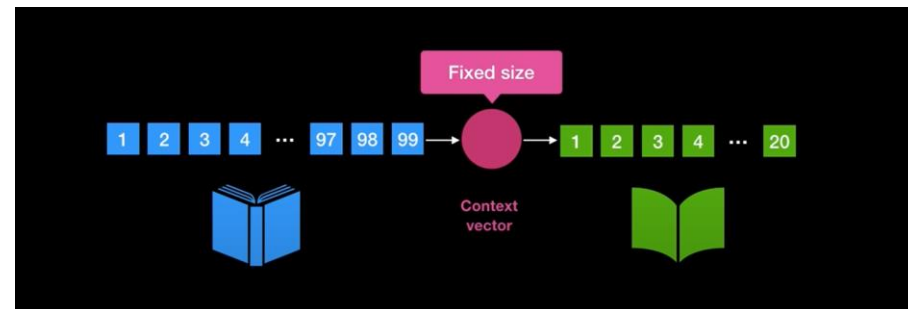
2. Bottleneck Problem

3. 매 시점에 동일한 길이의 context vector 사용



하나의 고정된 길이의 벡터로 전체 맥락 나타내는 것은 불충분

입력 문장 길이 ↑ ~ 번역 품질 ↓



## ● Attention Mechanism

### Attention의 등장

#### Seq2Seq

1. 장기 의존성 문제
2. Bottleneck Problem
3. 매 시점에 동일한 길이의 context vector 사용

하나의 고정된 길이의 벡터로 전체 맥락 나타내는 것은 불충분

입력 문장 길이 ↑ ~ 번역 품질 ↓

#### Attention

각 타임 스텝마다 생성된 context vector  
모두를 decoder에 넘겨줌

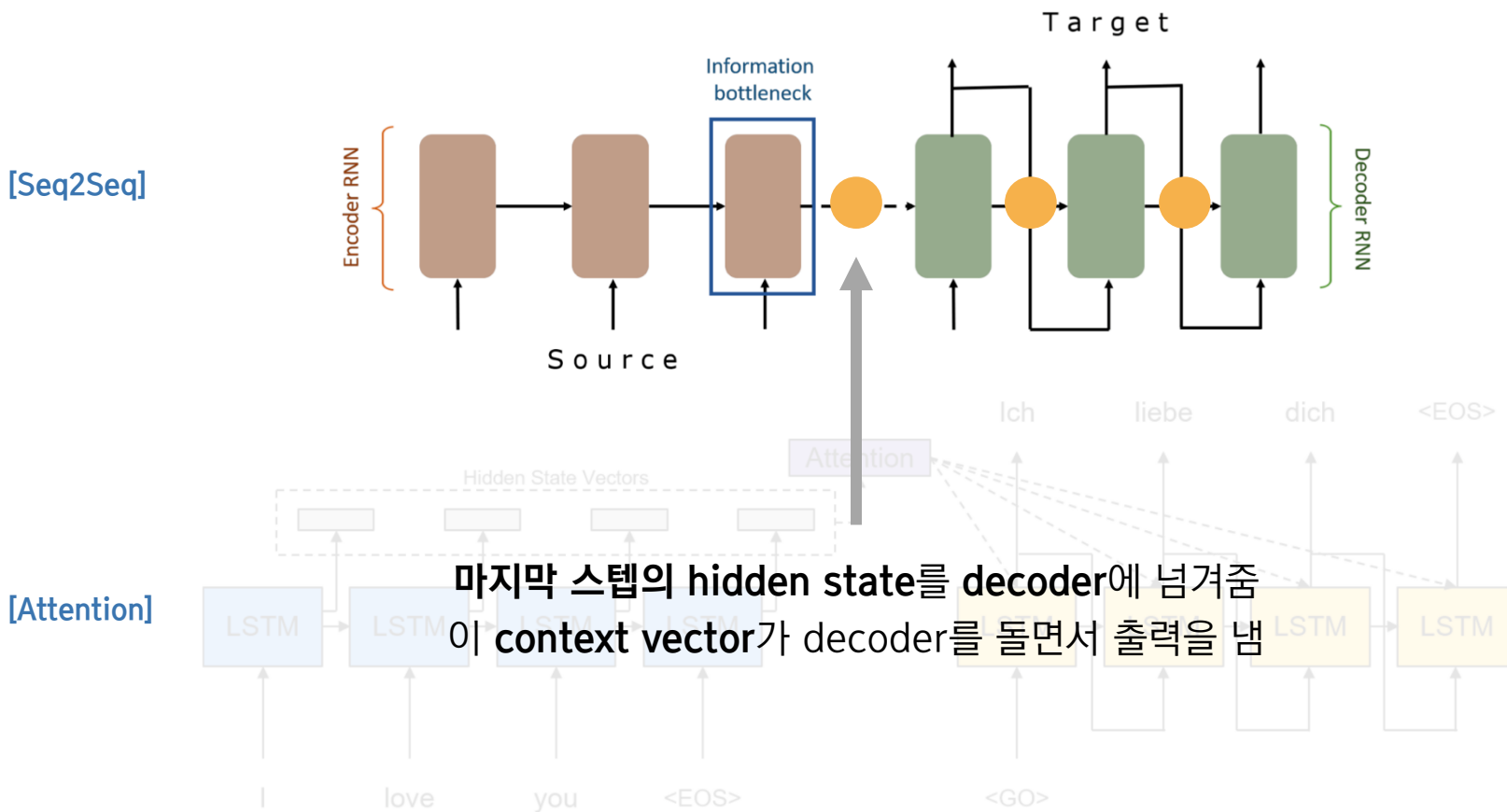


Decoding 단계에서  
입력 문장 중 어떤 부분에 **집중**해야 하는지  
타임 스텝 별로 알 수 있음

# 3 Attention

- Structure of Attention

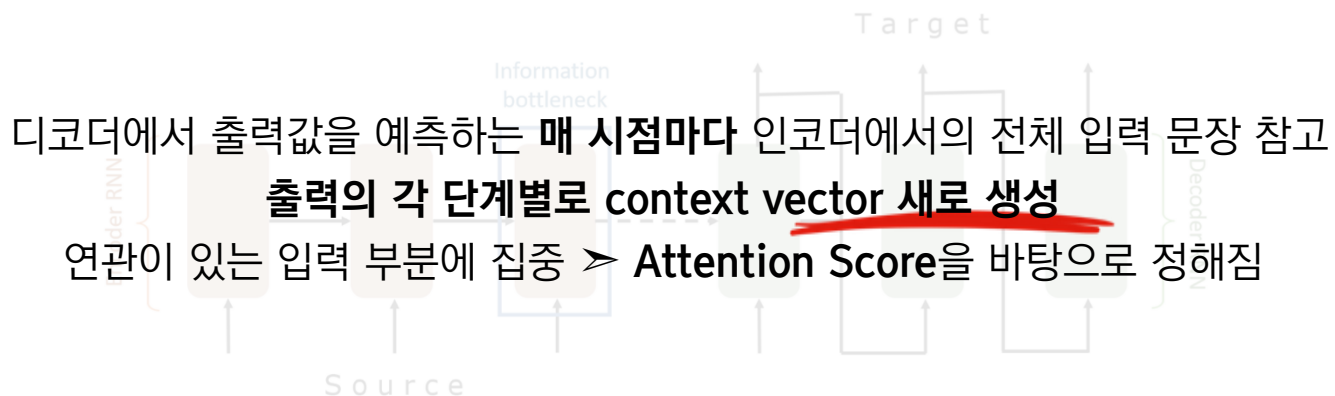
## Seq2Seq vs Attention



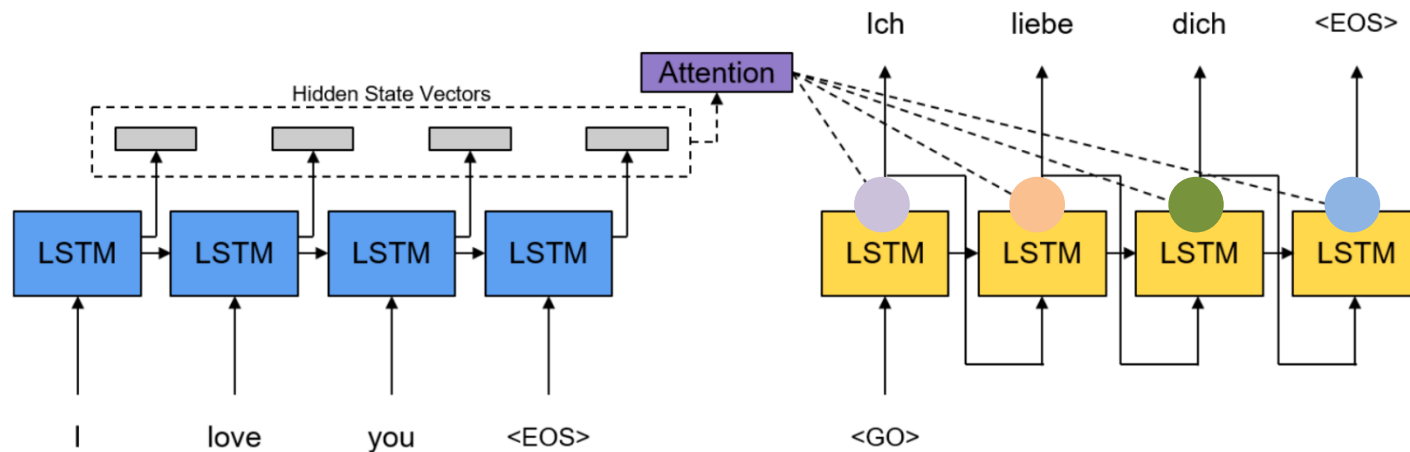
# 3 Attention

## ● Structure of Attention

## Seq2Seq vs Attention



디코더에서 출력값을 예측하는 **매 시점마다** 인코더에서의 전체 입력 문장 참고  
**출력의 각 단계별로 context vector 새로 생성**  
연관이 있는 입력 부분에 집중 > **Attention Score**을 바탕으로 정해짐



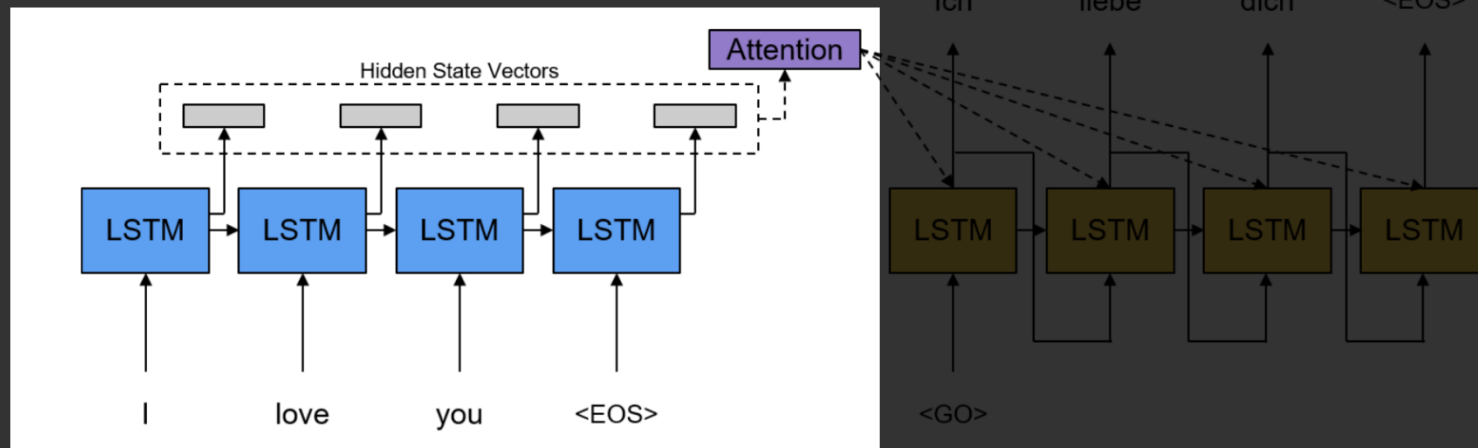
# 3 Attention

- Structure of Attention

## Seq2Seq vs Attention

[ Encoder ]

Hidden state 매 시점마다 출력



# 3 Attention

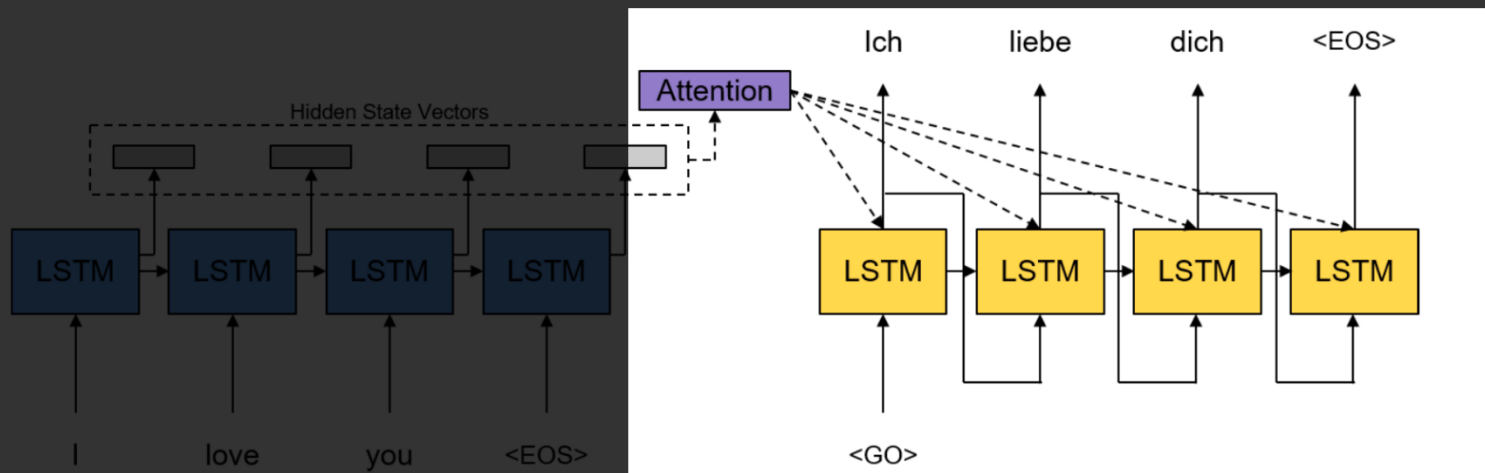
- Structure of Attention

## Seq2Seq vs Attention [ Decoder ]

Encoder **각 타임 스텝**의 hidden state들이  
Attention mechanism을 거친 후 전달됨



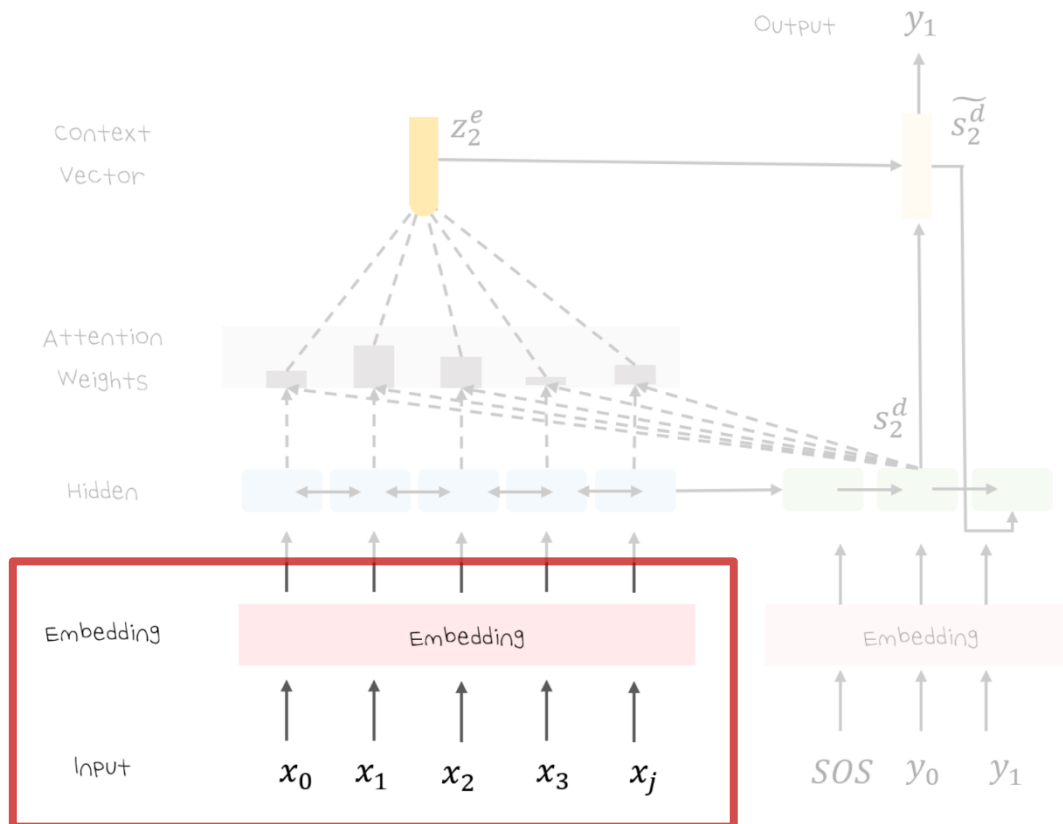
Encoder의 **모든 타임 스텝**에 영향을 받음



# 3 Attention

- Structure of Attention

## Encoder



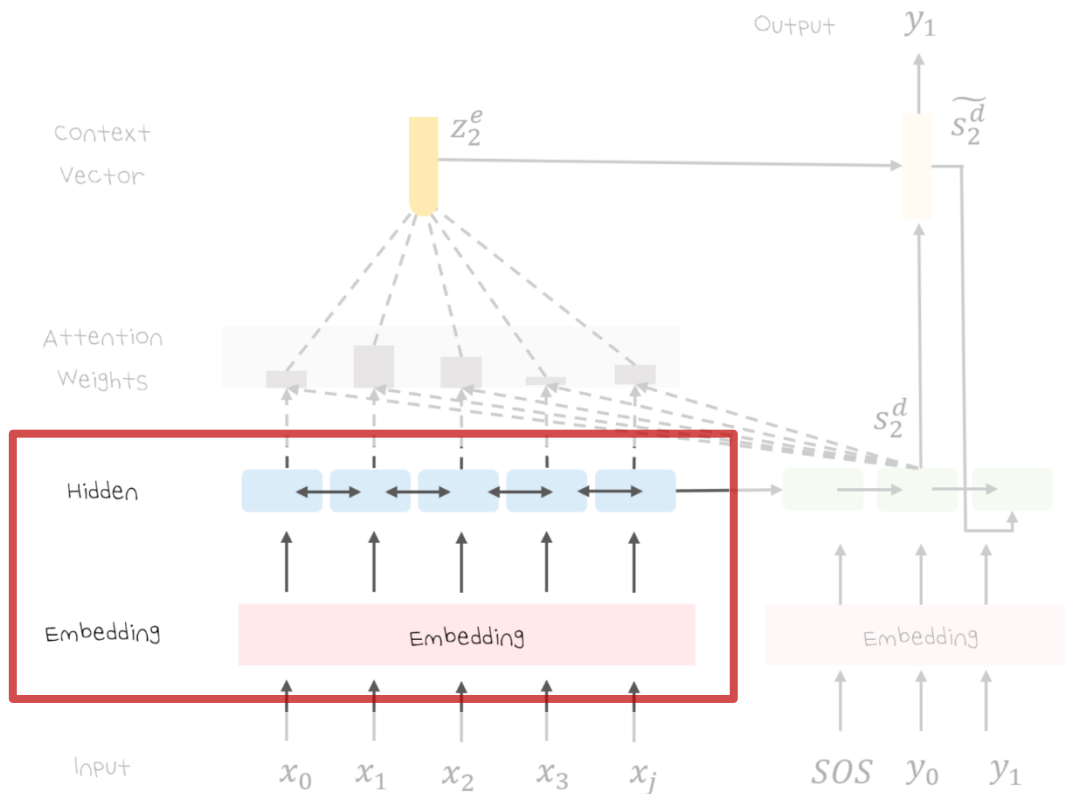
Source data가  
embedding layer로 들어감



# 3 Attention

- Structure of Attention

## Encoder



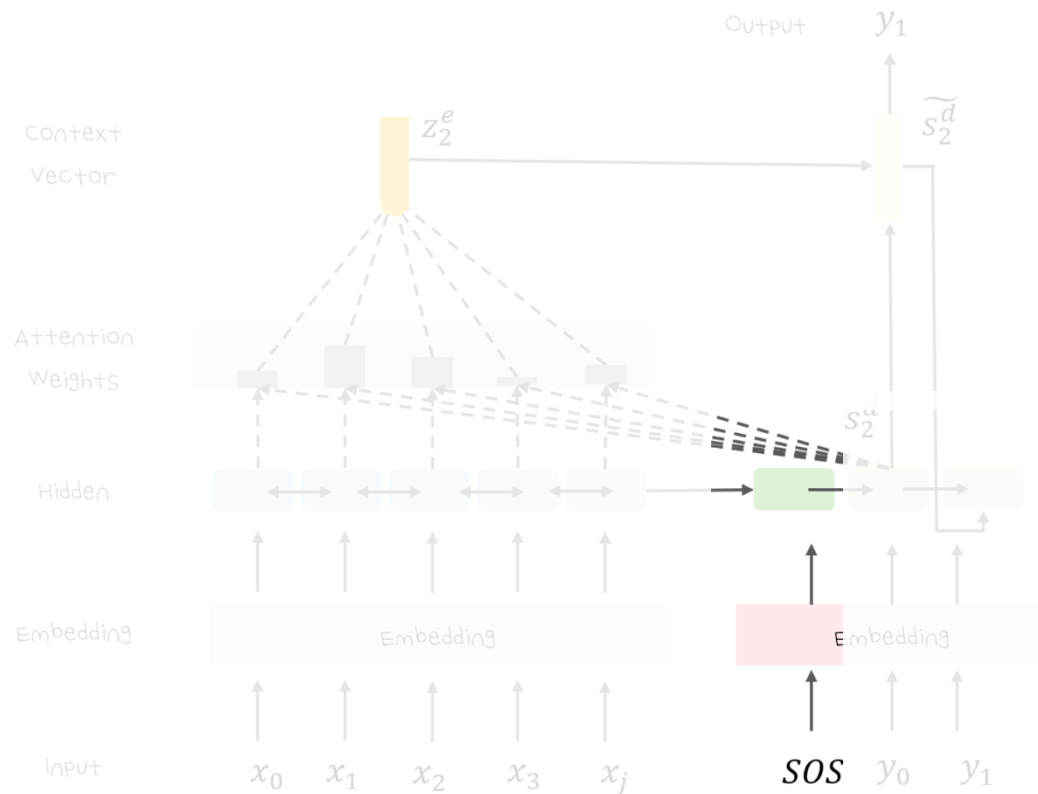
여기까지 인코더!  
Embedding layer 거친 값들이  
RNN(LSTM, GRU) 모델로 들어가서  
Hidden state 생성

Source data가  
embedding layer로 들어감

# 3 Attention

- Structure of Attention

## Decoder

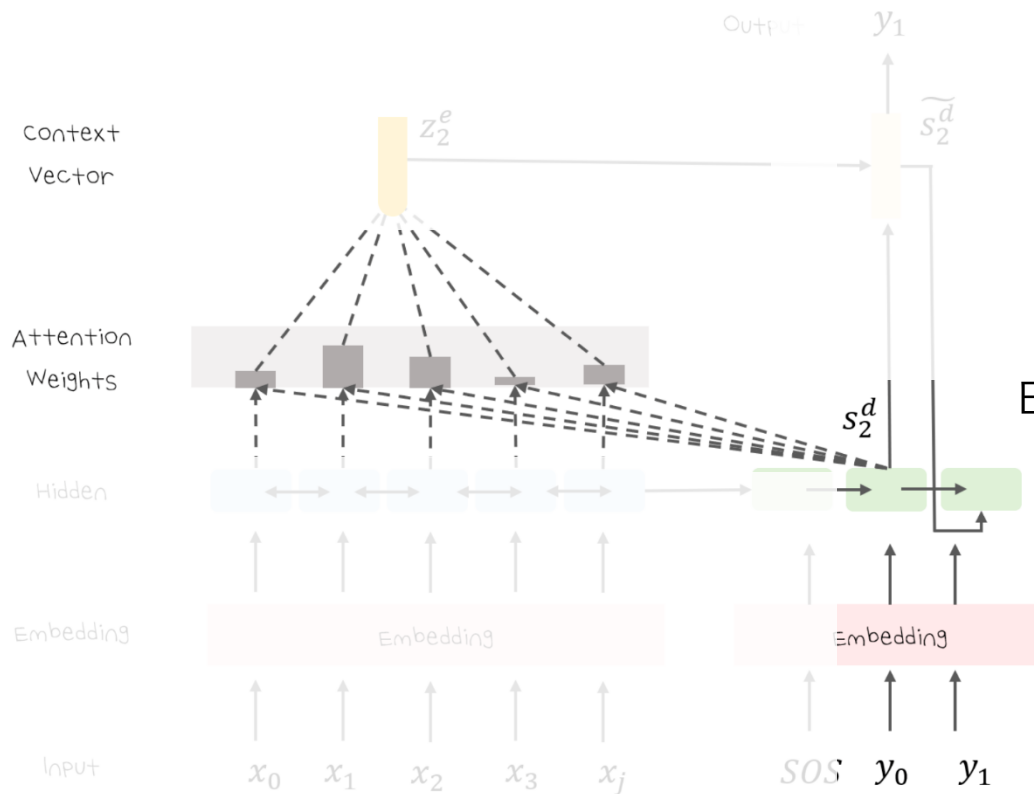


Encoder의 입력이 끝나면,  
Decoder의 입력 시작!

# 3 Attention

## ● Structure of Attention

### Decoder



Decoder의 이전 시점 hidden state와  
Encoder에서 구했던 hidden state들을 비  
교하여 attention score 생성!



Encoder의 입력이 끝나면,  
Decoder의 입력 시작!

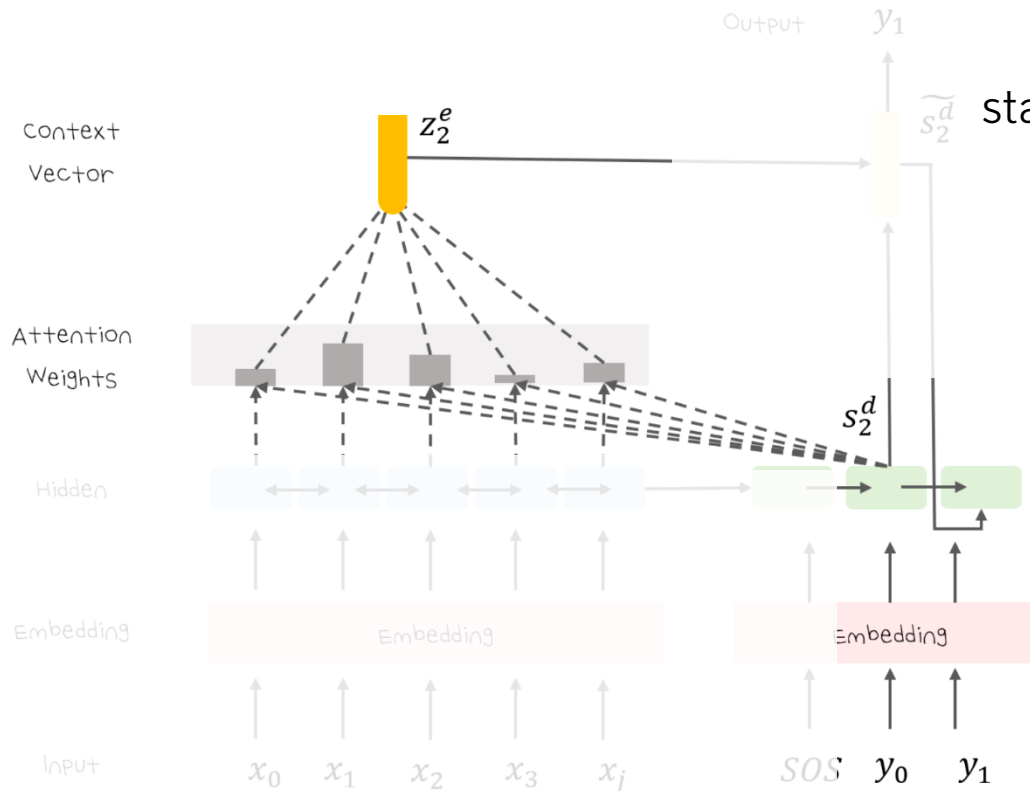
### 3 Attention

## ● Structure of Attention

# Decoder

이때 이번 시점의 decoder 출력 값과 인코더 hidden state들이 유사한 만큼 Attention! 되는 것임

Attention score와 Encoder hidden state들을 곱하여 Attention context 생성!



Decoder의 이전 시점 hidden state와 Encoder에서 구했던 hidden state들을 비교하여 attention score 생성!

Encoder의 입력이 끝나면,  
Decoder의 입력 시작!

- Structure of Attention

## Decoder – (1) Attention Score

Decoder의 hidden state

$$s_j = f(y_{j-1}, s_{j-1}, c_j)$$

$y_{j-1}$       지난 시점의 decoder 출력 값

$s_{j-1}$       지난 시점의 decoder hidden state 값

$c_j$       Attention context 값



$$c_j = \sum_{i=0}^N \overset{\text{Encoder의 hidden state}}{h_i} b_{ij}$$

$$b_{ij} = \text{softmax}(e_{ij})$$

$$e_{ij} = \text{align}(h_i, s_{j-1})$$

$h_i, s_{j-1}$ 의 유사도를 구한다

## Decoder – (1) Attention Score



VS

디코더의 이전 시점까지의 은닉 상태  $s_{j-1}$

## 유사도를 판단

- Structure of Attention

## Decoder – (1) Attention Score

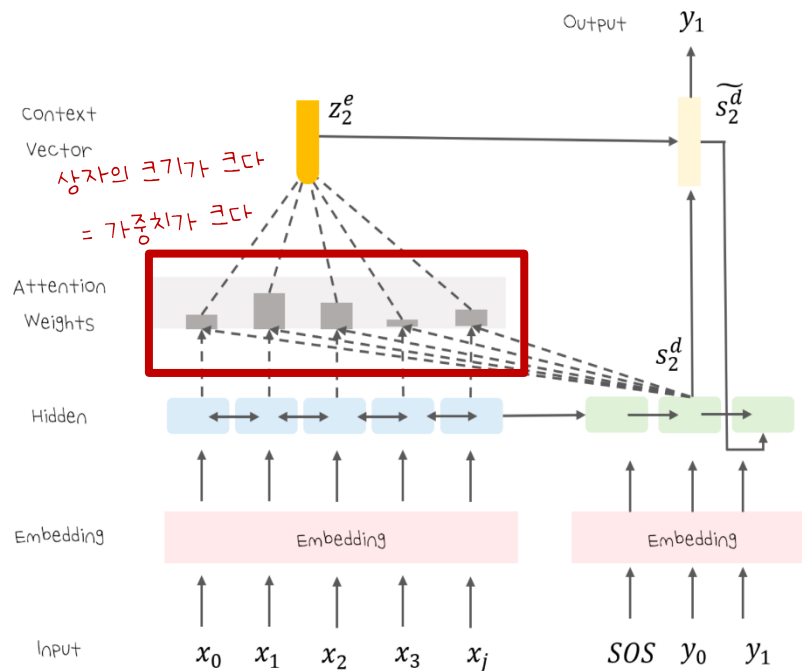
$e_{ij}$ 의 계산 방법

$$e = \begin{cases} \cos(h, s) & \text{Bahdanau attention} \\ s_j^T h_i & \text{Lounng attention} \\ \frac{h^T s}{\sqrt{N}} & \text{Scaled dot-product attention} \end{cases}$$

계산 방법에 따라 어텐션 메커니즘 종류가 변한다

## ● Structure of Attention

### Decoder – (2) Attention Distribution



### Attention Distribution

$$b_{ij} = \text{softmax}(e_{ij})$$

$e_j$ 에 **softmax** 함수를 적용

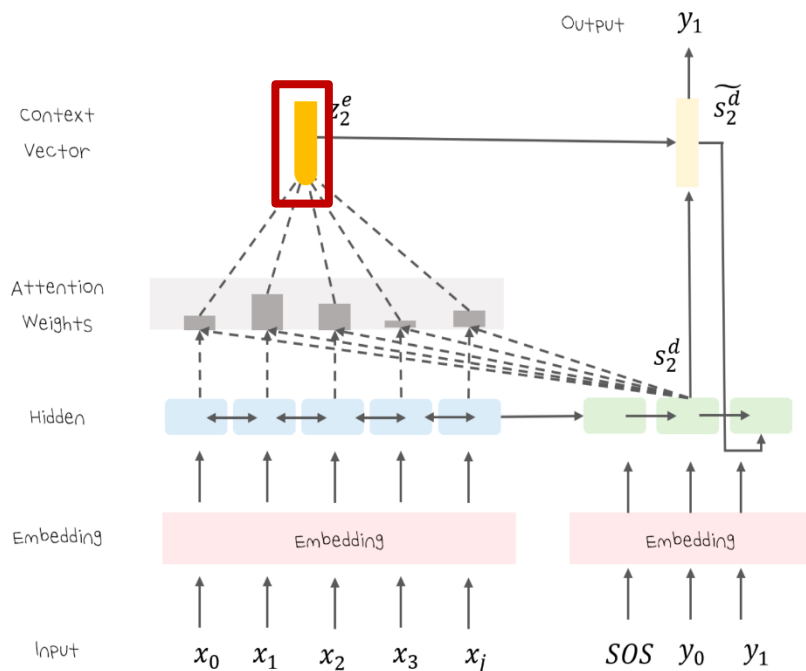
모든 값을 합하면 1이 되는 **분포** 만들어 냄

가중치  $\uparrow$  ~ 유사  $\uparrow$ , 반영  $\uparrow$



## Structure of Attention

### Decoder – (3) Attention Context



### Attention Context

$$c_j = \sum_{i=0}^N h_i b_{ij}$$

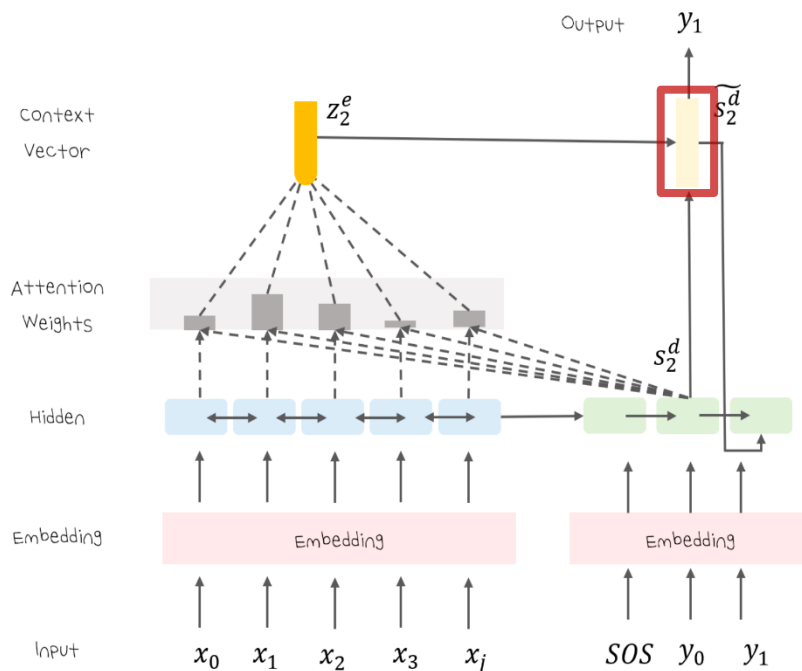
Attention context, context vector, attention value 등으로 불림

가중합을 하는 단계

각 인코더의 은닉 상태와 어텐션 가중치 값을 곱한 후  
이후 모두 더함

- Structure of Attention

## Decoder – (4) Concatenate



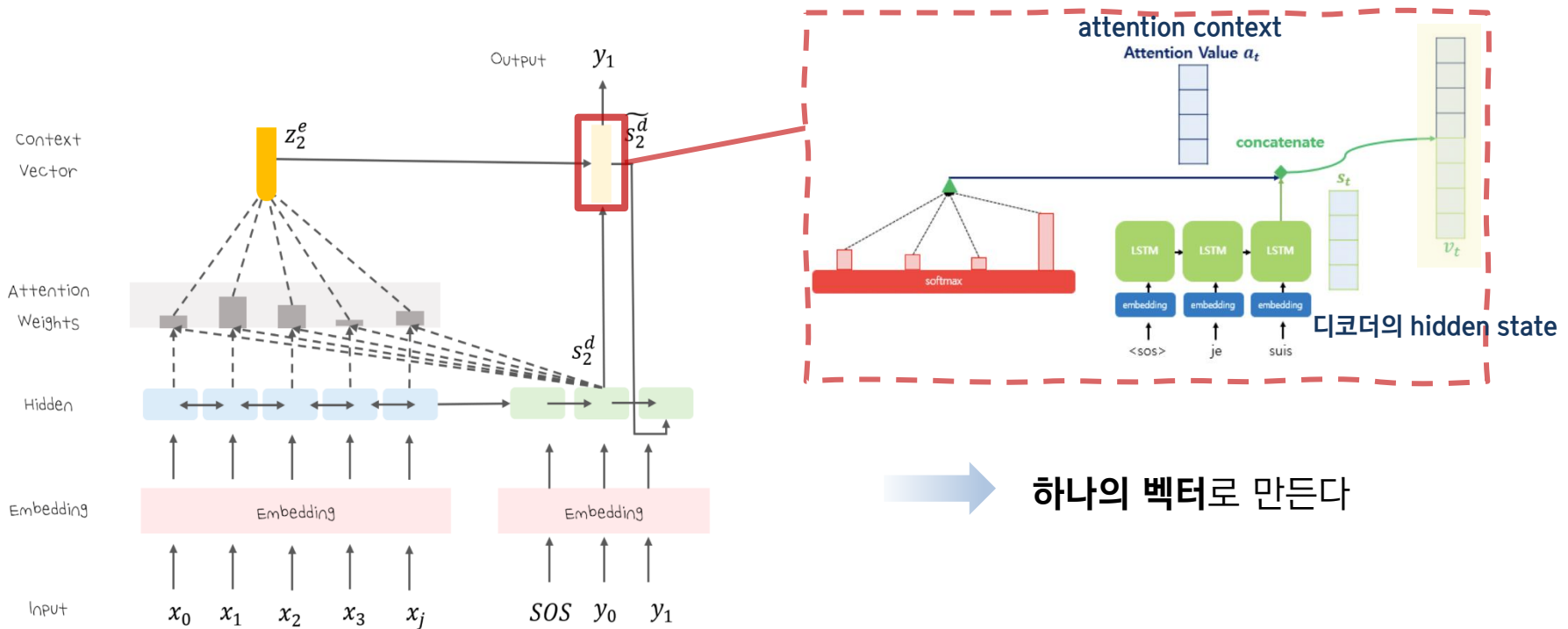
$c_j$ (attention context)를  
 $s_j$ (디코더의 hidden state)와 결합

→ 하나의 벡터로 만든다

# 3 Attention

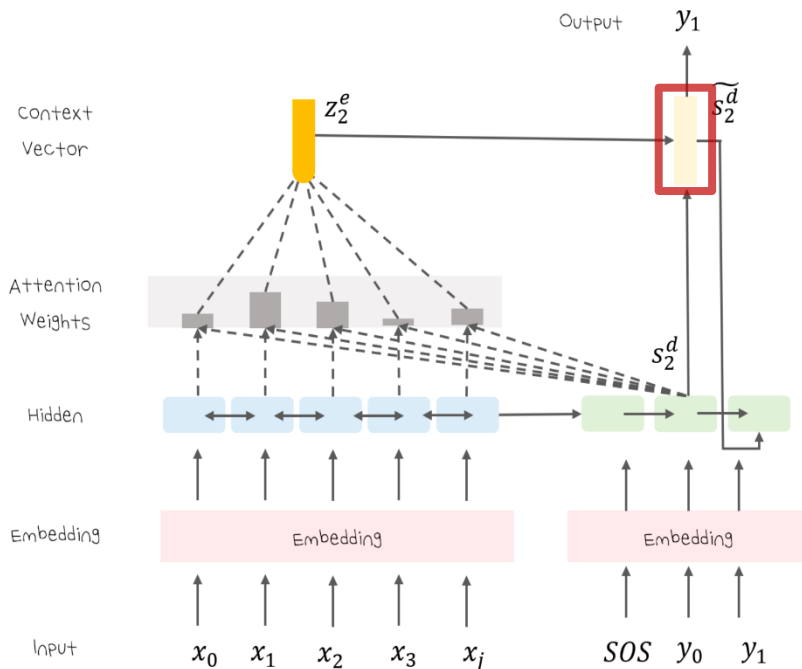
## ● Structure of Attention

### Decoder – (4) Concatenate



## Structure of Attention

### Decoder – (5) 출력층으로의 입력



$$\tilde{s}^t = \tanh(W_c[c_j; s_j] + b_c)$$

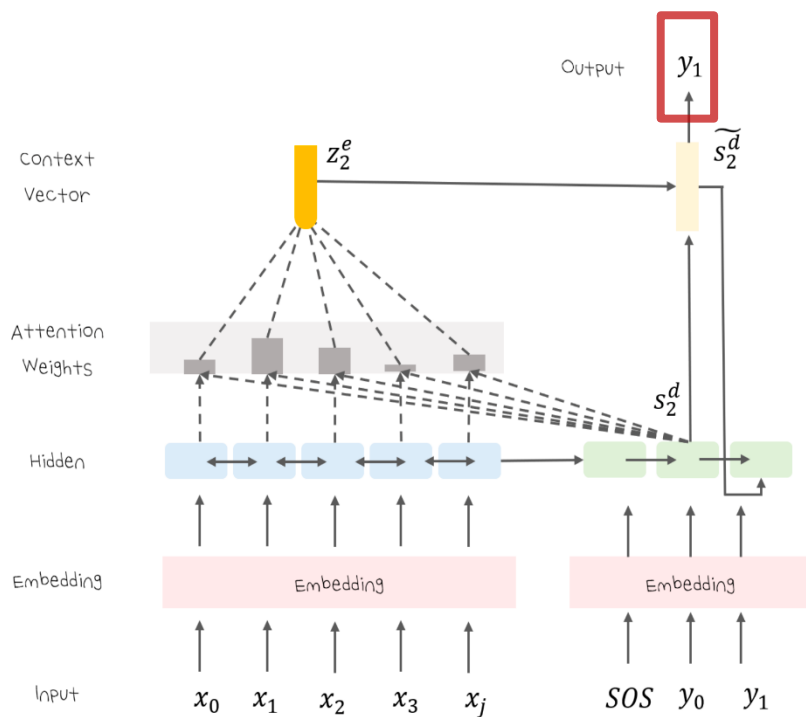
Concatenate한 값을 **가중치**와 곱하고, **편향**과 더함



**출력층의 입력**

- Structure of Attention

## Decoder – (5) 출력층으로의 입력



최종적인 예측값 얻음  
→ 다음 타임스텝 디코더의 입력



**THANK YOU**

