

딥러닝팀

1팀

김예찬
박시언
박윤아
정승민
김민

CONTENTS

1. 자연어

2. RNN

3. RNN 모델의 응용

4. 마무리

1

자연어

자연어, 자연어 처리 (NLP)



자연어란

우리가 일상생활에서 사용하는 언어를
인공적으로 만들어진 **인공어**와 구분하여 부르는 개념



자연어 처리 (Natural Language Processing, NLP)

사람의 언어를 컴퓨터가 이해하고 처리할 수 있도록 자연어를 다루는 일

자연어의 특징



순차성

RNN은 순차적인 데이터의 특징을 잘 반영함

순차적 데이터이기 때문에 순서가 달라질 경우 의미가 손상될 수 있음

“마감시간이 끝나기 전에 과제를 끝냈다.”

“과제가 끝나기 전에 마감시간이 끝났다.”



불연속성

연속적이거나 형태가 유사하더라도 완전히 다른 의미를 가질 수 있음

의자, 이자 / 과자, 과제

자연어의 특징



모호성

다양한 해석이 가능하여 모호함

① 표현의 중의성

“차를 마시러 공원에 가는 차 안에서 나는 그녀에게 차였다.”

→ 3개의 ‘차’의 의미가 모두 다름

② 문장 내 정보 부족

“나는 철수를 안 때렸다.”

자연어의 전처리

텍스트 데이터를 모델의 입력으로 사용하기 위해서는 **변환 필요** !

정제



토큰화



Subword Segmentation

자연어의 전처리

정제

텍스트 데이터의 의미를 분석할 때 필요하지 않은 변수들을 제거

There 과 there 의 차이? : 무의미

US와 us의 차이? : 유의미

→ 필요한 부분에 한해 대문자를 남기고 소문자로 변환하는 과정 진행



코퍼스 (corpus)

정제의 대상이 되는 전체 문장 데이터 집합

자연어의 전처리

토큰화

토큰 = 의미를 가지는 최소 단위!

의미를 가지는 최소 단위로 코퍼스를 잘게 쪼개는 과정

Tokenize on
rules

Let	's	tokenize	!	Is	n't	this	easy	?
-----	----	----------	---	----	-----	------	------	---

Tokenize on
punctuation

Let	'	s	tokenize	!	Isn	'	t	this	easy	?
-----	---	---	----------	---	-----	---	---	------	------	---

Tokenize on
white spaces

Let's	tokenize!	Isn't	this	easy?
-------	-----------	-------	------	-------

Let's tokenize! Isn't this easy?

자연어의 전처리

토큰화

토큰 = 의미를 가지는 최소 단위!

의미를 가지는 최소 단위로 코퍼스를 잘게 쪼개는 과정
토큰은 **정의**에 따라, **목적**에 따라 달라질 수 있음!

= 같은 문장도 다양한 토큰화 가능

Tokenize on
rules

Let	's	tokenize	!	Is	n't	this	easy	?
-----	----	----------	---	----	-----	------	------	---

Tokenize on
punctuation

Let	'	s	tokenize	!	Isn	'	t	this	easy	?
-----	---	---	----------	---	-----	---	---	------	------	---

필요에 따라 토큰화 **방법**을 잘 선택하는 것이 중요!

Tokenize on
white spaces

Let's	tokenize!	Isn't	this	easy?
-------	-----------	-------	------	-------

Let's tokenize! Isn't this easy?

자연어의 전처리

토큰화



nltk – word_tokenize()

가장 기본적인 토큰화 함수, 띄어쓰기와 구두점을 기준으로 토큰화

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

```
In [3]: print(word_tokenize('''Don't be fooled by the dark sounding name, Mr. Jone's  
Orphanage is as cheery goes for a pastry shop.'''))
```

```
['Do', "n't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr.',  
'Jone', "'s", 'Orphanage', 'is', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'sh  
op', '.']
```

자연어의 전처리

토큰화



Keras – Text to word sequence()

대문자를 소문자로 일괄 변환, 심표와 마침표 모두 제거

```
In [9]: from tensorflow.keras.preprocessing.text import text_to_word_sequence
```

```
In [10]: print(text_to_word_sequence('''Don't be fooled by the dark sounding name, Mr. Jones'  
Orphanage is as cheery goes for a pastry shop.''))
```

```
["don't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', 'mr', "jone's",  
'orphanage', 'is', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop']
```

자연어의 전처리

한국어의 토큰화



한국어는 어간에 접사가 붙어 단어를 이루고,
의미와 문법적 기능이 더해지는 **교착어**이므로 토큰화가 까다로움



일반적으로 **형태소** 단위로 토큰화 진행

자연어의 전처리

한국어의 토큰화

```
from konlpy.tag import Hannanum
```

```
nanum = Hannanum()
```

```
nanum.morphs("드디어 교안 다 썼다!")
```

```
['드디어', '교안', '다', '쓰', '었다', '!']
```

```
from konlpy.tag import Kkma
```

```
kkma = Kkma()
```

```
kkma.morphs("드디어 교안 다 썼다!")
```

```
['드디어', '교안', '다', '쓰', '었', '다', '!']
```

자연어의 전처리

Subword Segmentation



OOV (Out of Vocabulary) 문제

: 모델이 학습하지 못한 단어로 인해 발생하는 문제

→ 이를 해결하기 위해 subword segmentation이 필요 !



Subword Segmentation

의미 분석의 일반화를 위해 단어를 더 작은 단위로 분해하는 과정
학습된 단어의 일부분으로 유사한 단어의 의미를 예측하는 것이 목표

구글에서 공개한 Sentencepiece 라이브러리로 이용 가능

Word Vector

One-hot-vector



단어 집합 (Vocabulary)

서로 다른 단어들을 중복 없이 모은 집합



One-hot-vector

단어 집합에 있는 N개의 단어를 각각 N개 차원의 벡터로 표현

이 과정을 one-hot-encoding 라고 함

값이 0 또는 1로 표현되는 **희소 표현**을 사용한 **희소 벡터**!

Word Vector

One-hot-vector

Hotel = [0 0 0 1 0 0 0 0 0]

Motel = [0 0 0 0 0 0 1 0 0]



단점

① Hotel과 Motel은 서로 orthogonal 하므로 유사성 없음

→ 검색 결과에 반영되지 않는 문제

② N개 차원 표현으로 인한 차원의 저주

③ 공간적인 낭비가 심함



Word Vector

Word Embedding



밀집 표현

희소 표현과 반대되는 개념

= 벡터의 차원을 단어 집합의 크기로 상정하지 X

사용자가 설정한 값으로 **모든 단어의 벡터 표현의 차원을 맞춤**

실수 값으로 구성된 벡터로 단어를 **의미적으로 표현**

각 실수 값의 의미를 알 수는 없음



Word Embedding = 단어를 밀집 벡터로 표현하는 방법

Embedding Vector = Word Embedding의 결과 벡터

Word Vector

Word2Vec

비슷한 위치에 등장하는 단어는 비슷한 의미를 가진다는 분포 가정

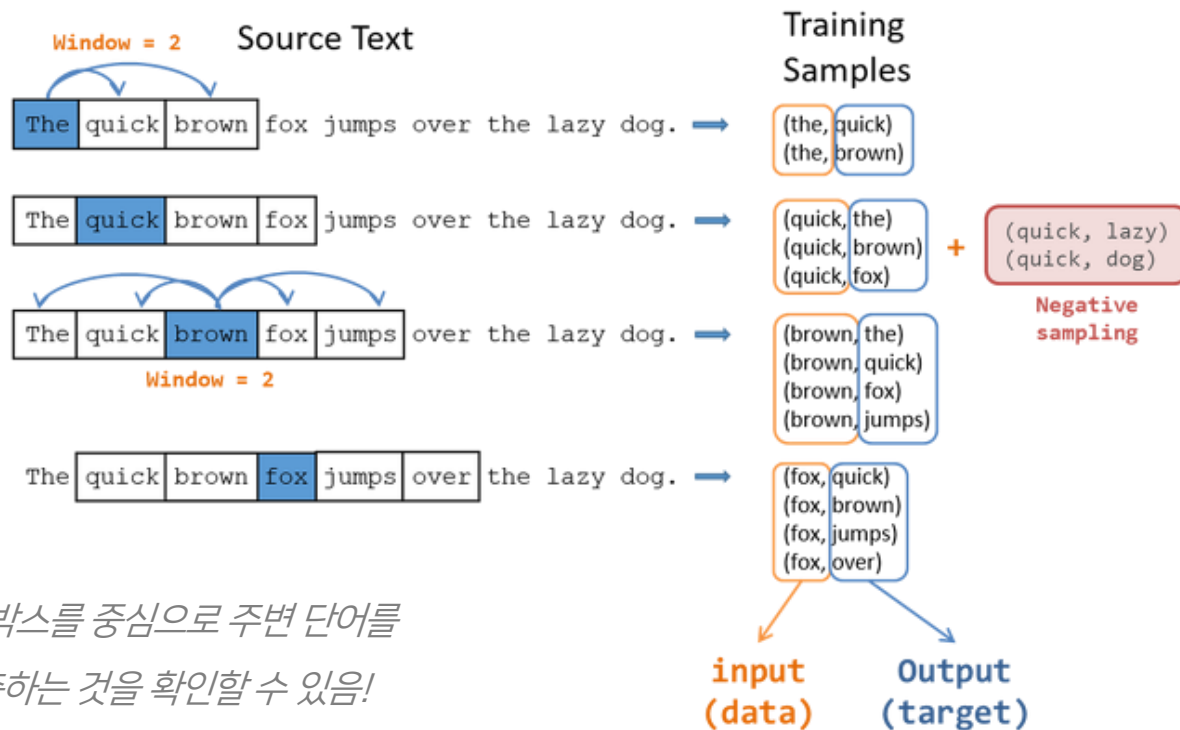
Word Embedding의 대표적인 방법

- CBOW : 주변의 단어를 입력으로中间的의 단어를 예측하는 방법
- Skip-gram : 中间的의 단어를 입력으로 주변 단어를 예측하는 방법

일반적으로 성능이 더 좋아 자주 사용

Word Vector

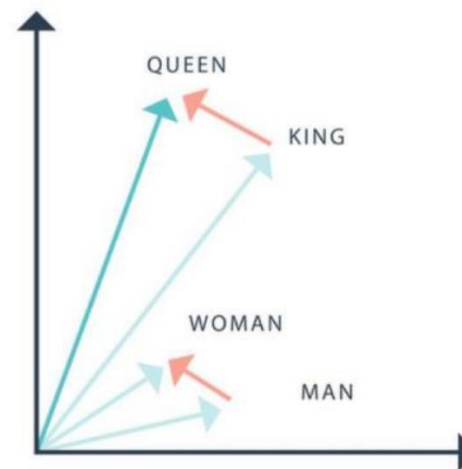
Word2Vec _ skip-gram



Word Vector

Word Embedding 장점

+ King	[0.3, 0.7, 0.1, 0.5]
- Man	[0.2, 0.2, -0.3, 0.5]
+ Woman	[0.6, 0.3, 0.2, 0.8]
<hr/>	
≈ Queen	[0.7, 0.8, 0.6, 0.8]



각 단어가 고차원 벡터 공간의 한 점으로 표현됨
따라서, 벡터간 연산이 가능



2

RNN

RNN (Recurrent Neural Network)

RNN과 CNN의 차이점



시간정보나 순서정보가
훨씬 중요한
sequential data 사용
Ex) 텍스트, 음성

V/S

CNN

입체적인 공간 정보를
다루는 데이터 사용
Ex) 이미지

RNN (Recurrent Neural Network)

RNN과 CNN의 차이점



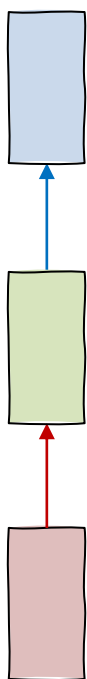
시간정보나 순서정보가
훨씬 중요한
sequential data 사용
Ex) 텍스트, 음성



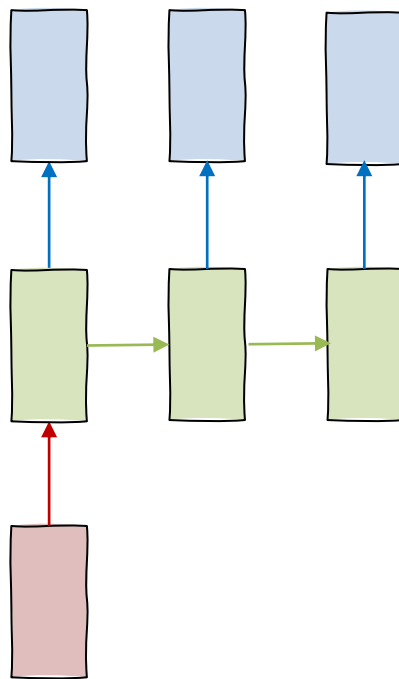
Vanilla RNN
LSTM
GRU

RNN (Recurrent Neural Network)

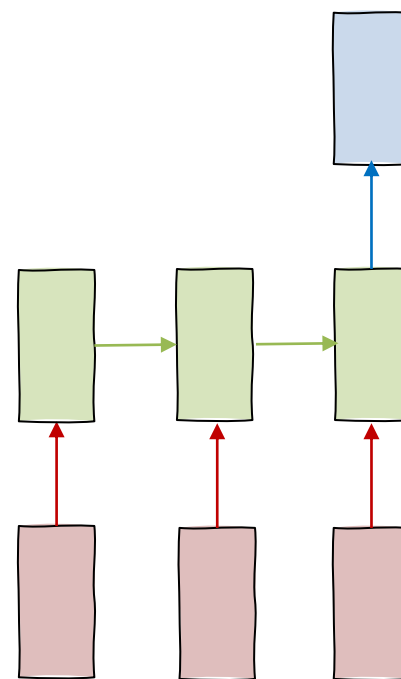
입력, 출력 개수 별 모델 분류



one to one



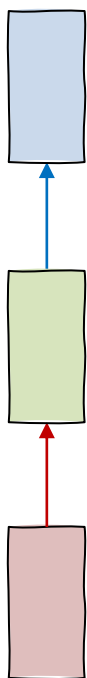
one to many



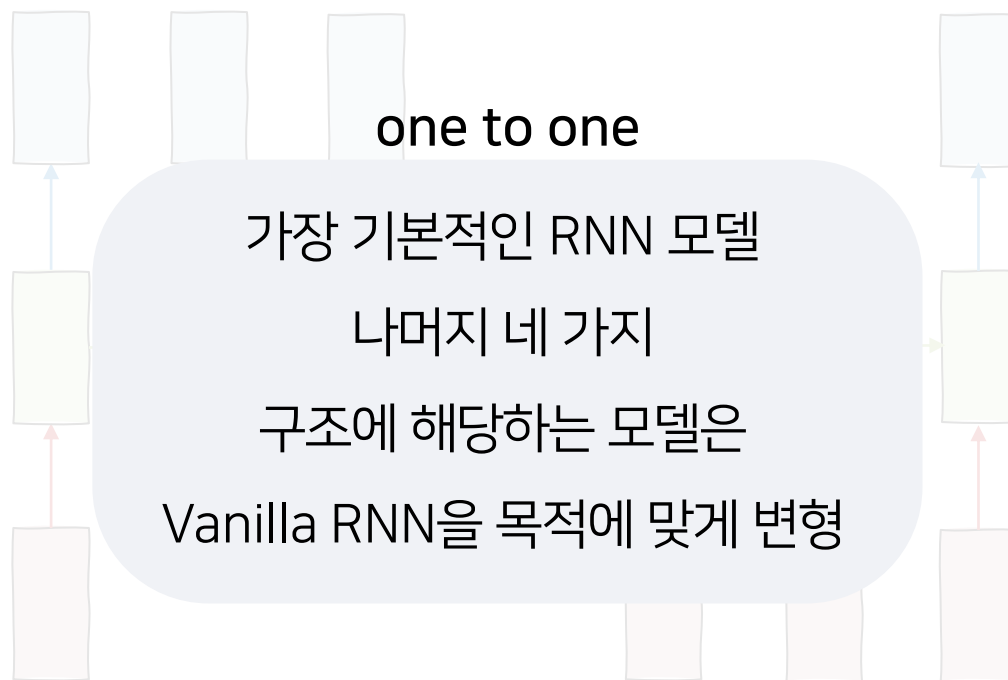
many to one

RNN (Recurrent Neural Network)

one to one



one to one



one to many

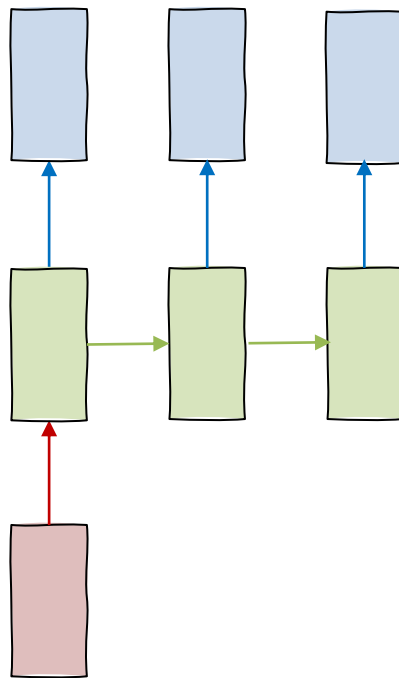
many to one

RNN (Recurrent Neural Network)

one to many



one to one



one to many

one to many

하나의 입력에 대해
여러 시점의 출력 반환
Ex) 이미지 설명

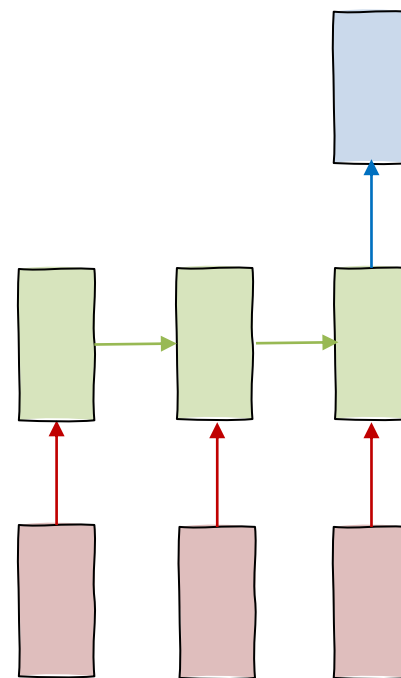
many to one

RNN (Recurrent Neural Network)

many to one

many to one

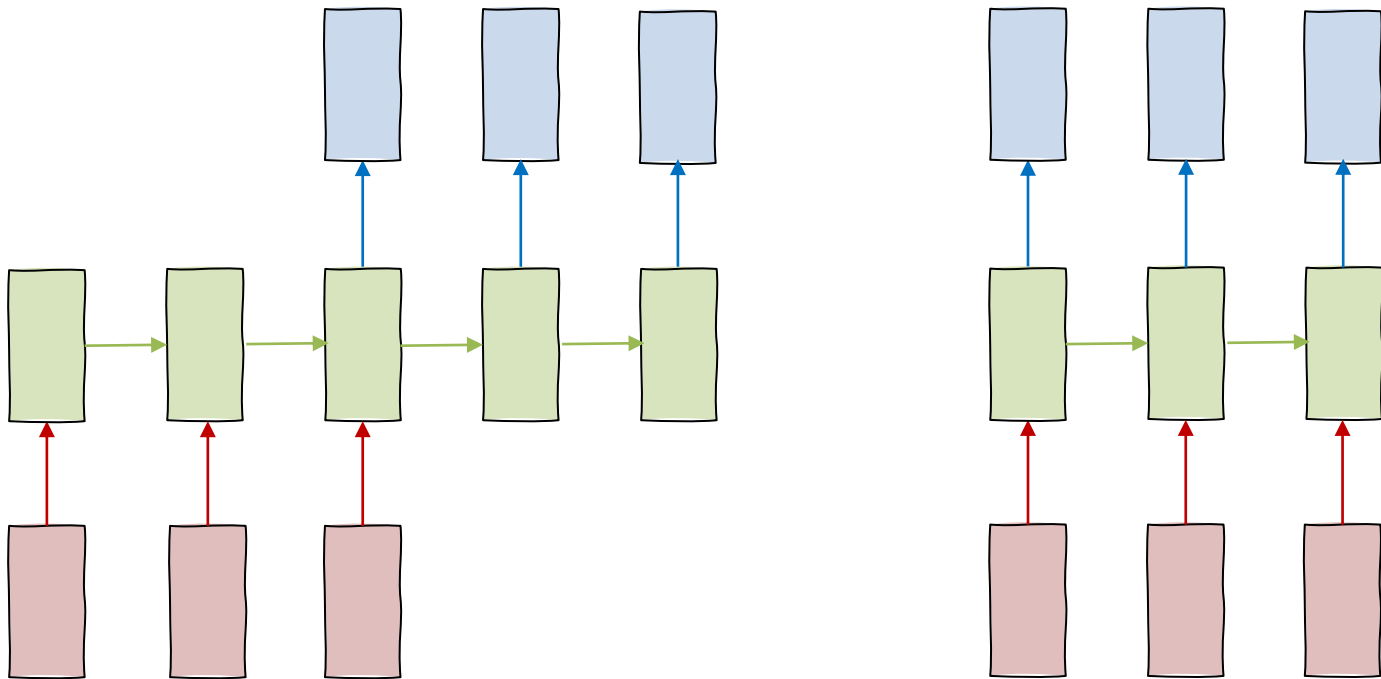
다수의 입력에 대해
하나의 출력을 반환
Ex) 감성 분석



many to one

RNN (Recurrent Neural Network)

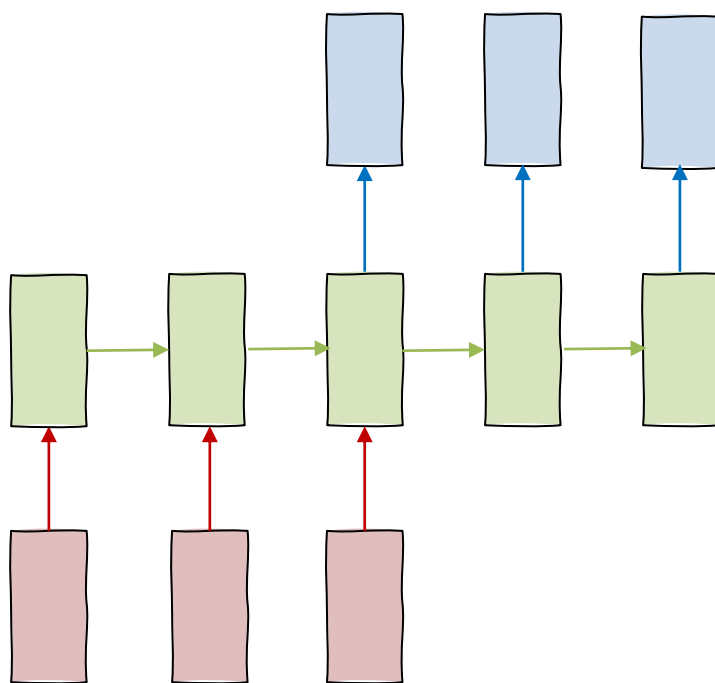
many to many



many to many

RNN (Recurrent Neural Network)

입력, 출력 개수 별 모델 분류



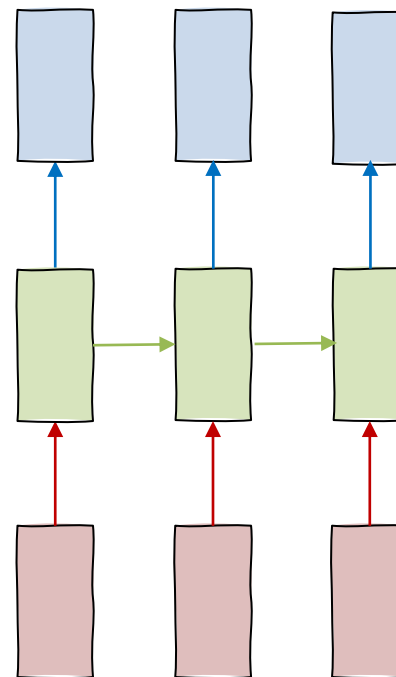
다수의 입력에 대해
다수의 출력을 반환
Ex) 기계 번역

many to many

RNN (Recurrent Neural Network)

입력, 출력 개수 별 모델 분류

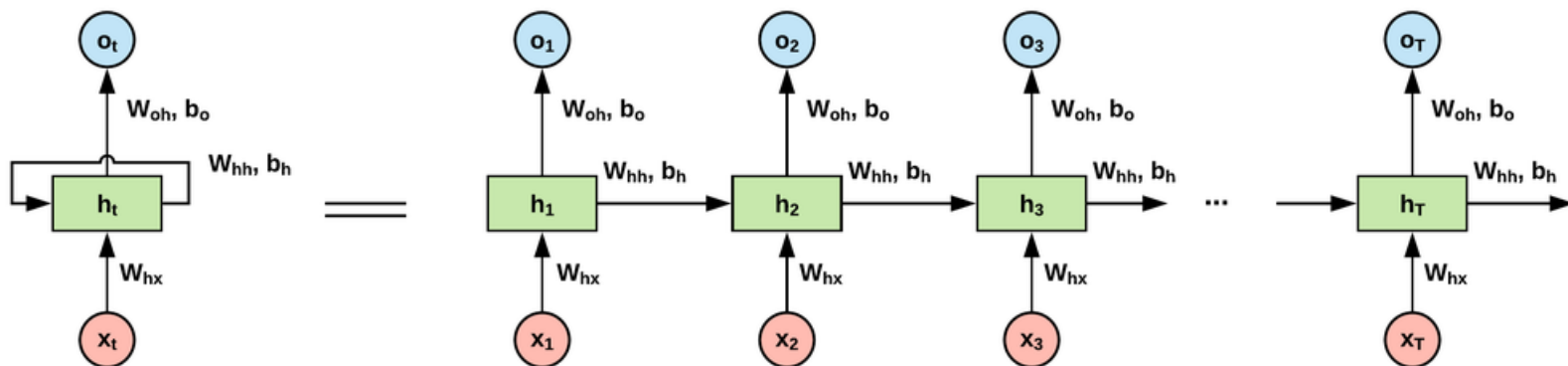
한 시점의 입력에 대해
매번 출력 반환
Ex) 영상 데이터와 관련된 과제,
실시간 오타 수정



many to many

Vanilla RNN

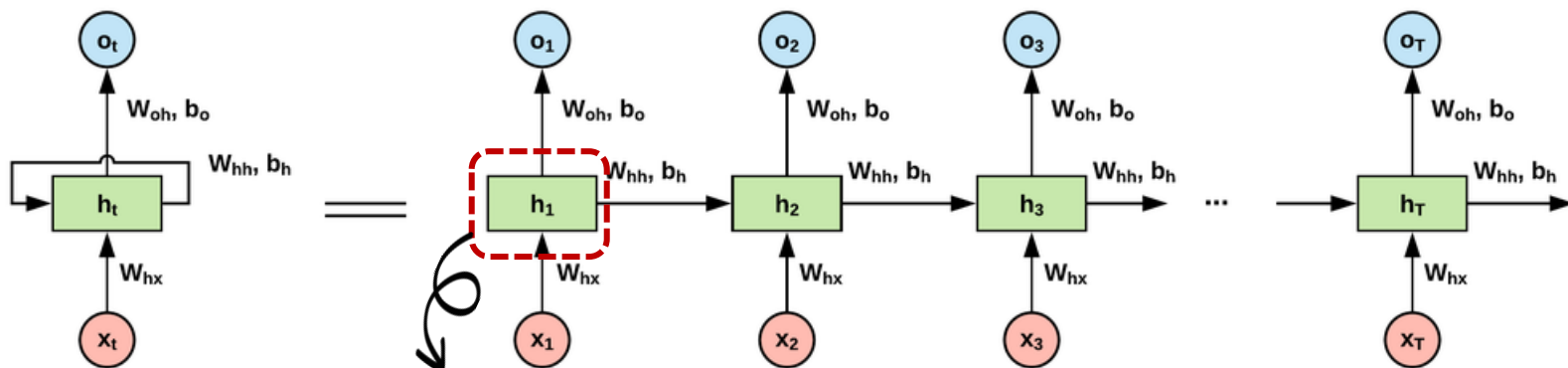
RNN의 기본적인 형태



시간 순서의 데이터를 학습함에 있어 하나의 Layer 반복 사용
즉, RNN의 Hidden Layer는 처음부터 마지막 시점 t 의 데이터까지 사용

Vanilla RNN

RNN의 기본적인 형태



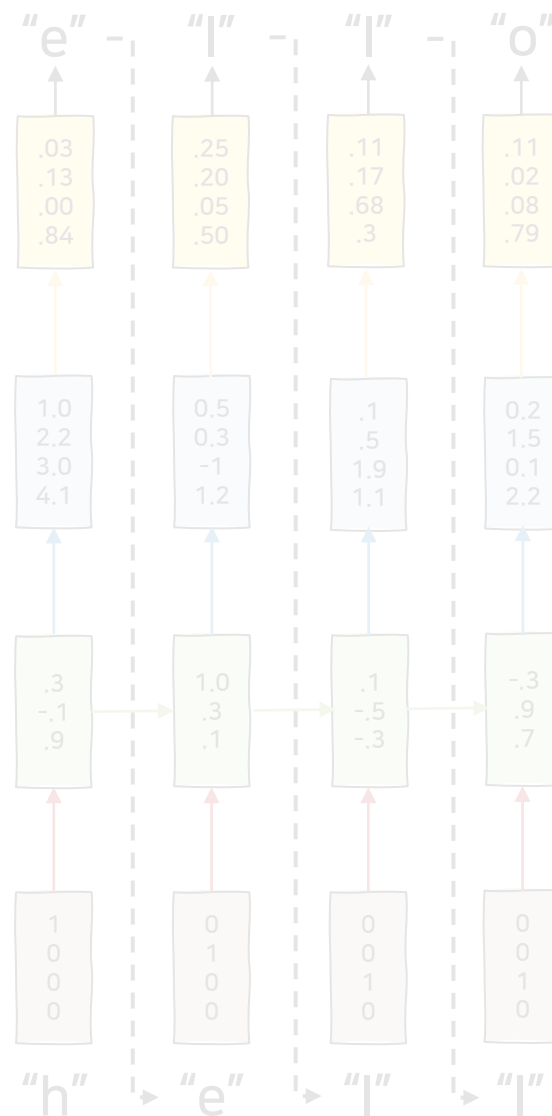
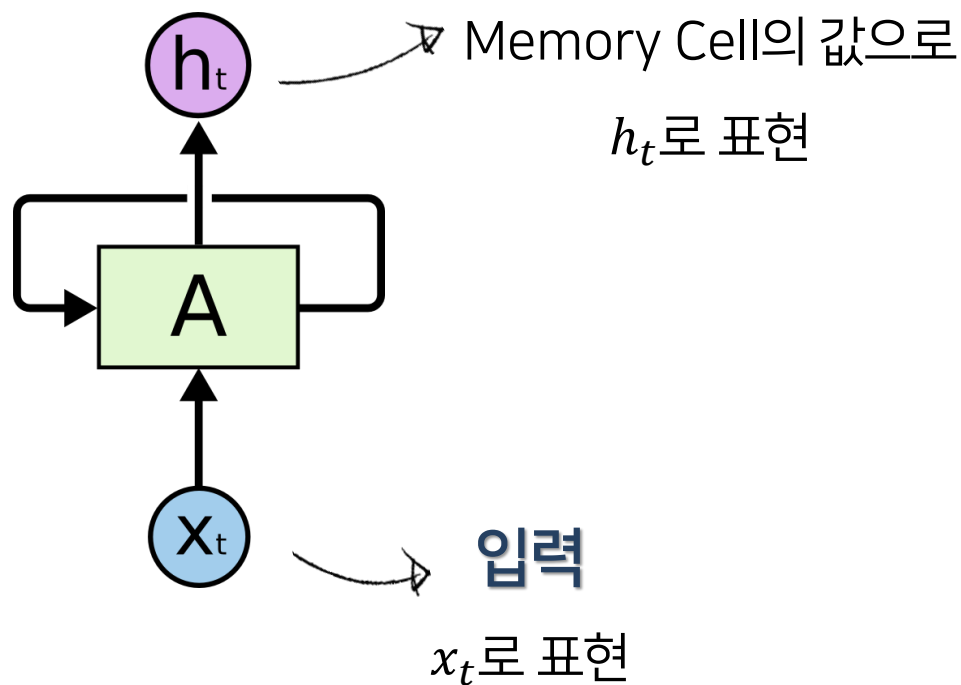
메모리셀 (Memory Cell)

이전 시점의 데이터들을 기억

시간 순서의 데이터를 학습함에 있어 **하나의 Layer** 반복 사용
즉, RNN의 Hidden Layer는 처음부터 마지막 시점 t 의 데이터까지 사용

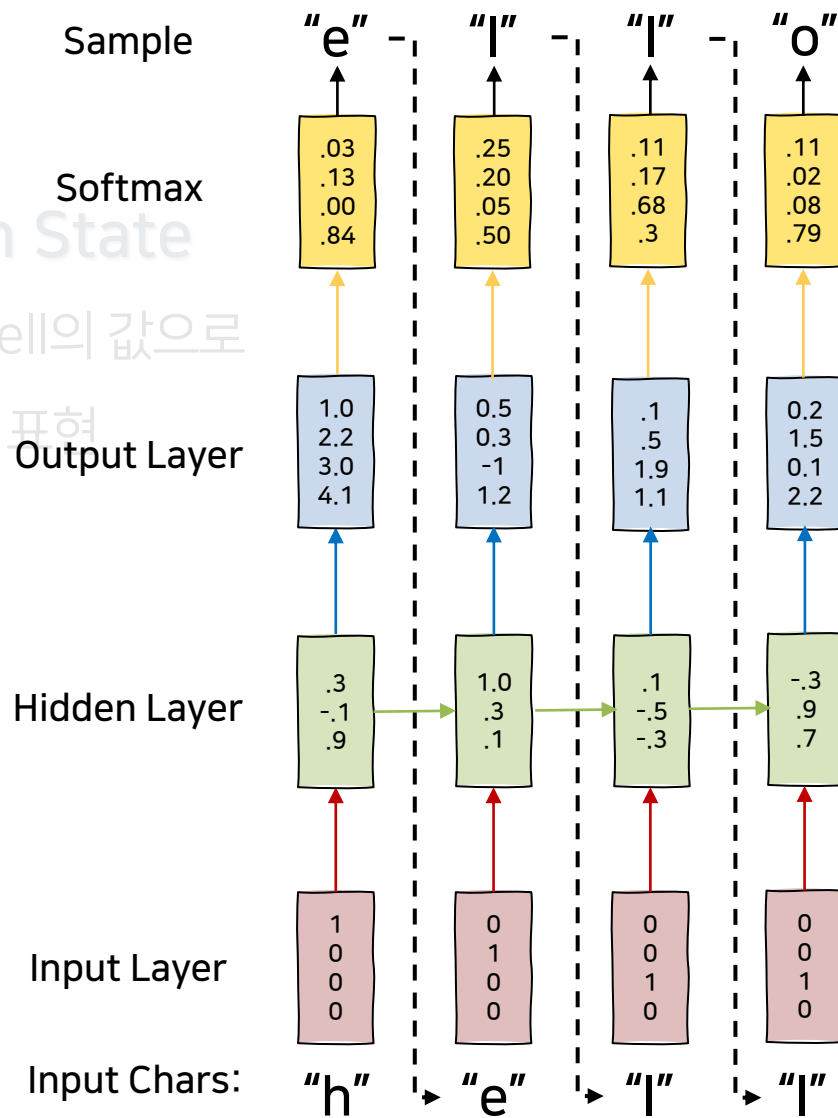
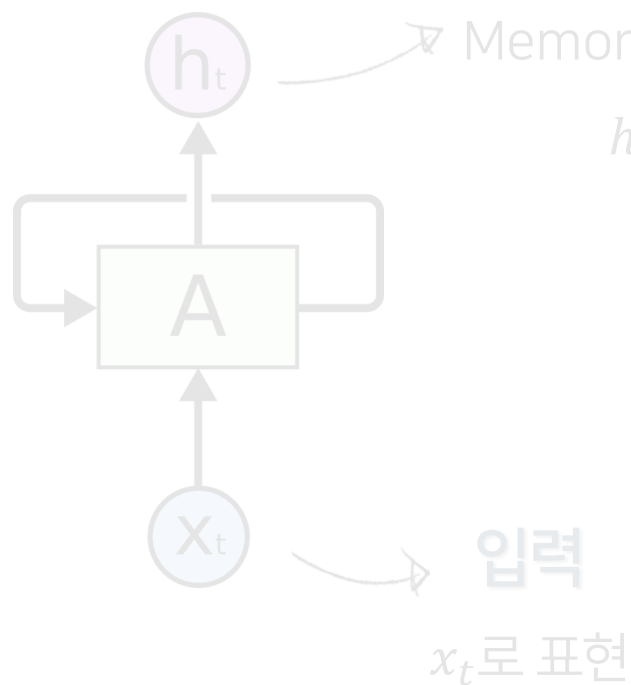
Vanilla RNN

RNN의 작동원리



Vanilla RNN

RNN의 작동원리



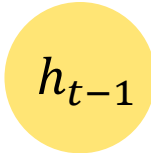
Hidden State

Memory Cell의 값

$$h_t = f_W(h_{t-1}, x_t)$$

 h_t

New state

 h_{t-1}

Old state

 f_W

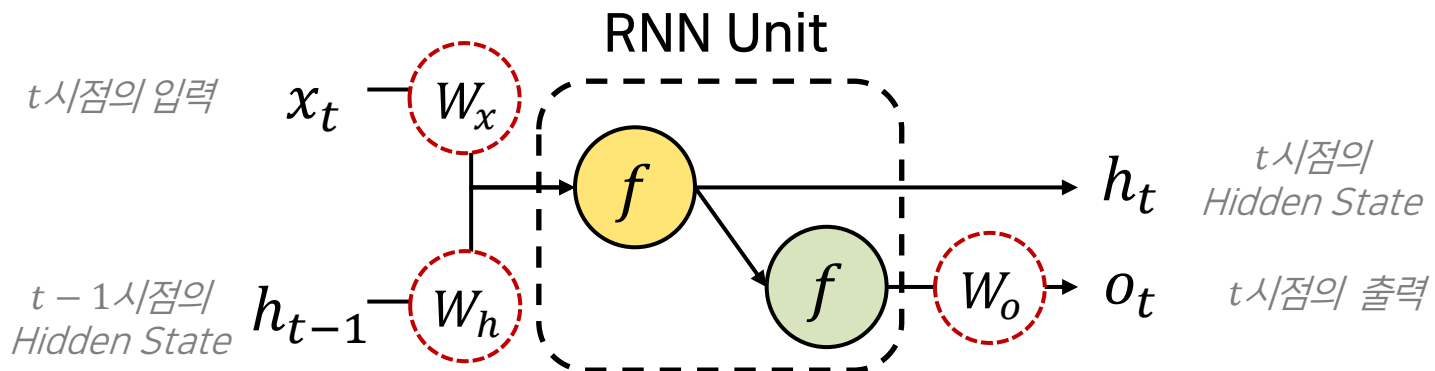
파라미터 W 를
가진 활성화함수

 x_t

Time step의 input

Hidden State

Vanilla RNN의 활성화 함수

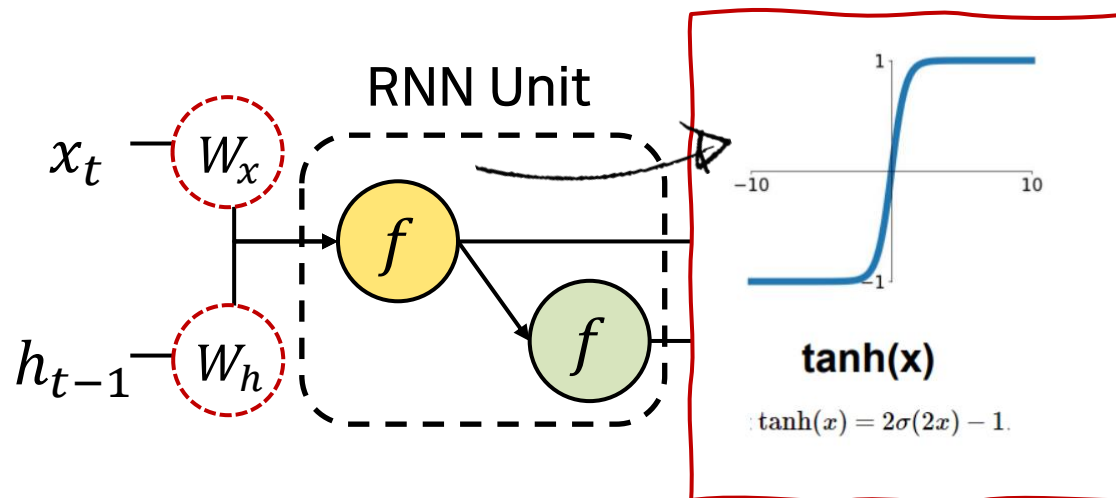


Vanilla RNN의 가중치는 크게 3가지로,

W_x, W_h, W_o 가 존재

Hidden State

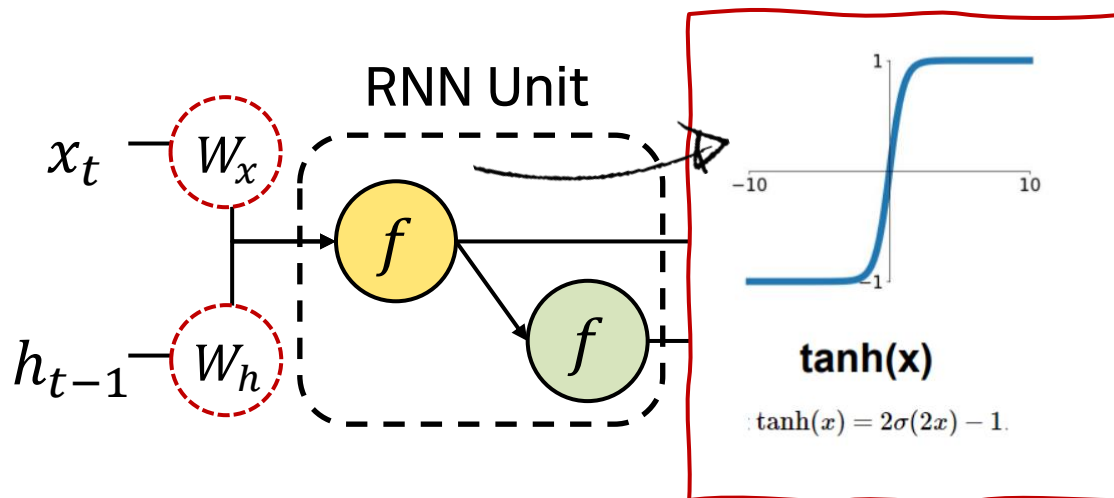
Vanilla RNN의 활성화 함수



Vanilla RNN의 활성화 함수는
 \tanh 함수를 사용

Hidden State

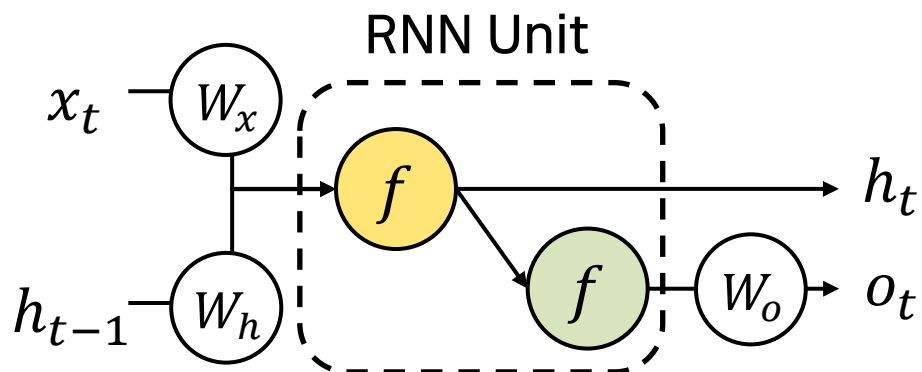
Vanilla RNN의 활성화 함수



CNN에서 자주 활용되는 ReLU 함수를 사용할 경우,
순환구조를 가지는 RNN의 특성상 값이 발산할 수 있음

Hidden State

hidden state, out put의 수식



$$h_t = \tanh(W_x \times x_t + W_h \times h_{t-1})$$

$$o_t = f(W_o h_t)$$

역전파 (Back Propagation)

BPTT, Truncated BPTT

역전파

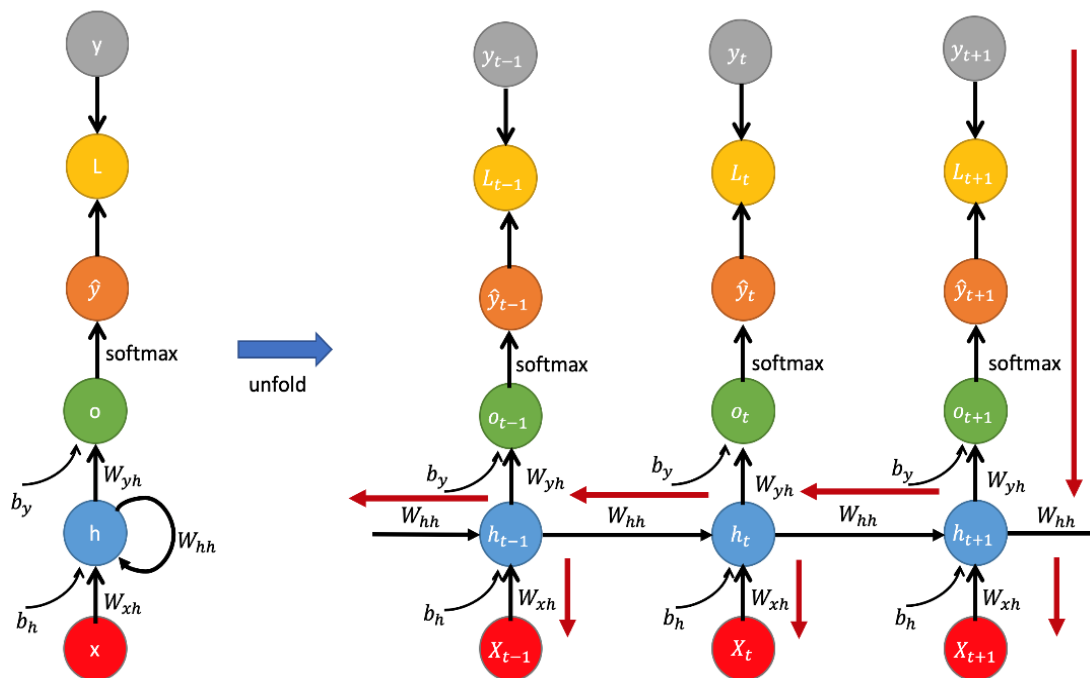
해당 모델로 해결하고자
하는 문제, Sequential
Data의 크기에 따라
다르게 정의



BPTT
Truncated BPTT

BPTT (Back Propagation Through Time)

매 시점마다 출력이 반환될 때



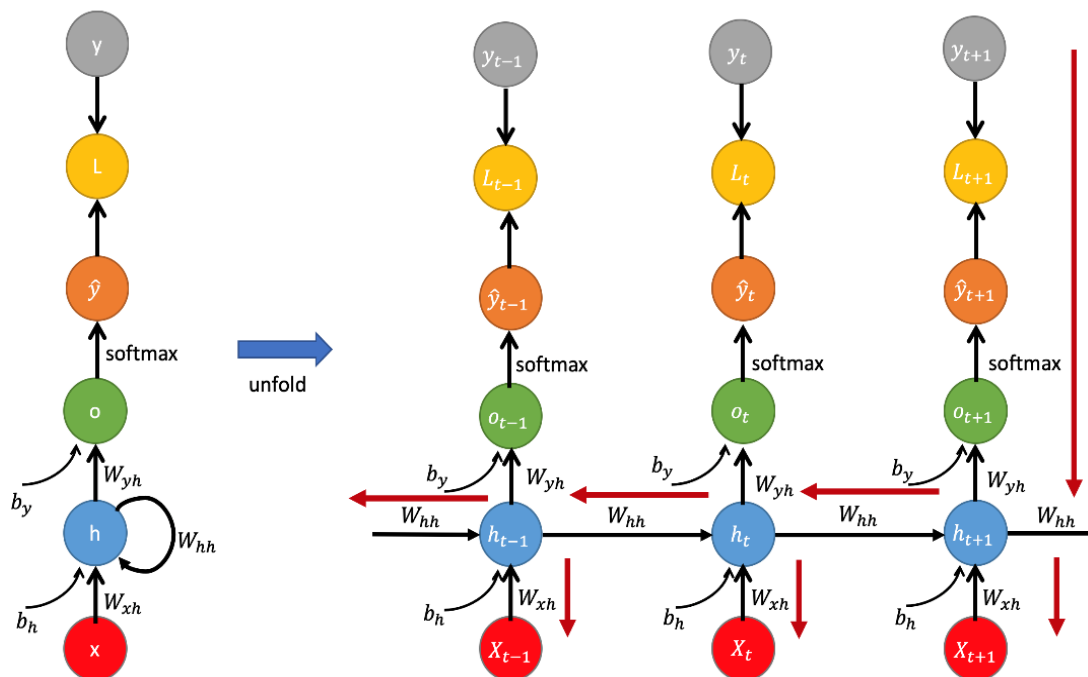
매 시점마다 출력이
반환되는 형태일 때 활용

매 시점마다
loss 역시 계산 가능

gradient를 계산하여
첫번째 시점까지 역전파 진행

BPTT (Back Propagation Through Time)

매 시점마다 출력이 반환될 때



매 시점마다 출력이
반환되는 형태일 때 활용



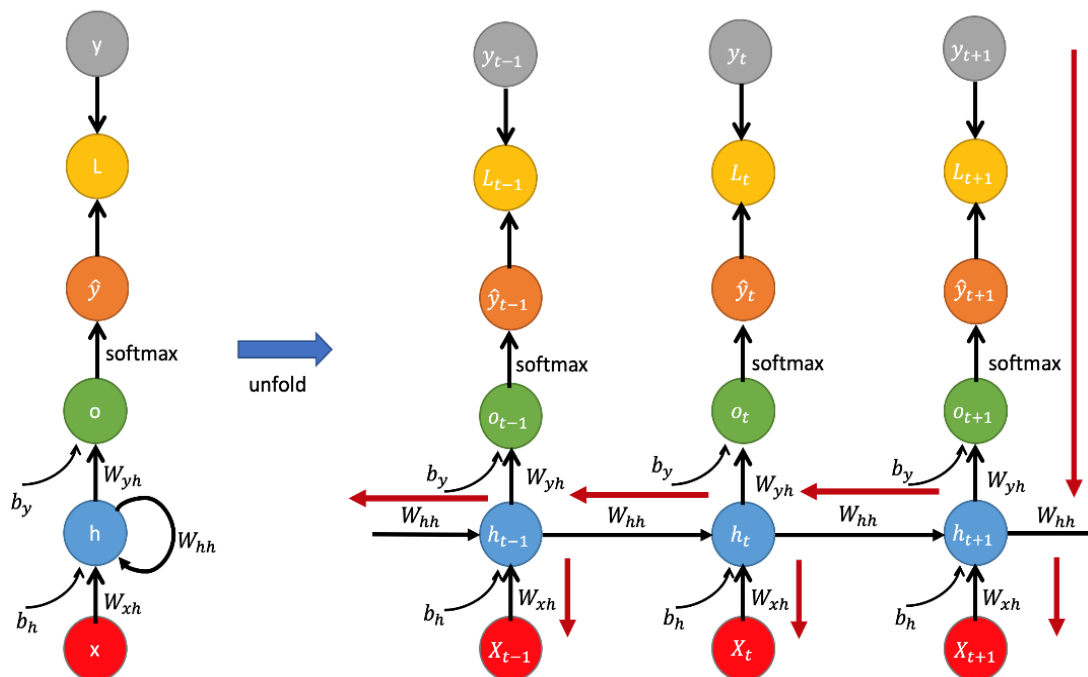
매 시점마다
loss 역시 계산 가능



gradient를 계산하여
첫번째 시점까지 역전파 진행

BPTT (Back Propagation Through Time)

매 시점마다 출력이 반환될 때



매 시점마다 출력이
반환되는 형태일 때 활용



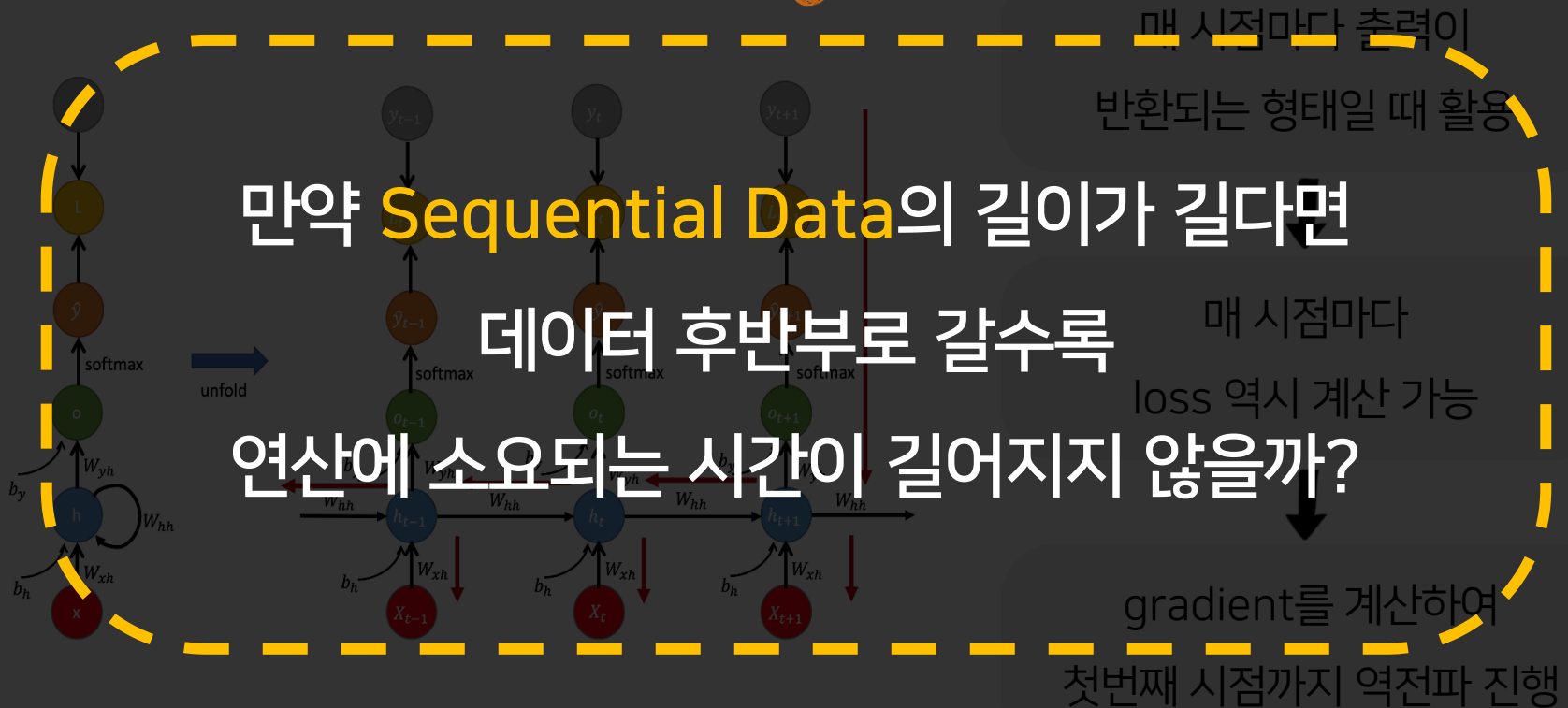
매 시점마다
loss 역시 계산 가능



gradient를 계산하여
첫번째 시점까지 역전파 진행

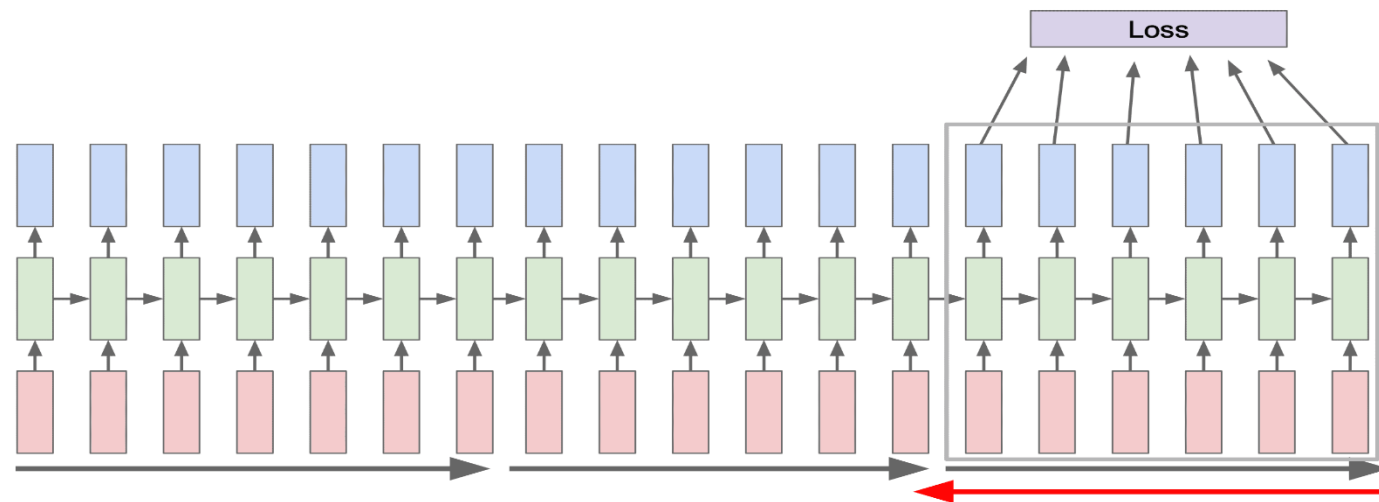
BPTT (Back Propagation Through Time)

매 시점마다 출력이 반환될 때



Truncated BPTT

BPTT의 해결책



Sequential Data를 **일정한 구간으로**
끊어 구간마다 BPTT를 진행

LSTM

배경

통계 성공관
사과 코딩 여제 위대한
예스비

Vanilla
RNN

apple statistics Sungkyunk
coding The wan
empress great University
Yeatsby

미안하다 이거 보여주려고
어그로 끌었다.. 나루토 사스케
싸움수준... 아무튼 나루토는
진짜 애니중최고명작임

Vanilla
RNN

I'm sorry. To show you
this, I've focused your
attention. Isn't it
amazing how Naruto Sa

LSTM

배경 - 장기 의존성 문제

Vanilla RNN

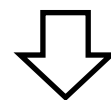
순전파 시에 활성화 함수인

\tanh 로 인해

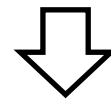
데이터의 길이가

길어질수록 취약함

미안하다 이거 보여주려고
어그로 끌었다.. 나루토 사스케
싸움수준... 아무튼 나루토는
진짜 애니중최고명작임



Vanilla
RNN



I'm sorry. To show you
this, I've focused your
attention. Isn't it
amazing how Naruto Sa

LSTM

배경 - 장기 의존성 문제

Vanilla RNN

순전파 시에 활성화 함수인

\tanh 로 인해

데이터의 길이가

길어질수록 취약함

apple statistics Sungkyunk
coding The University
empress great Yearsby

LSTM

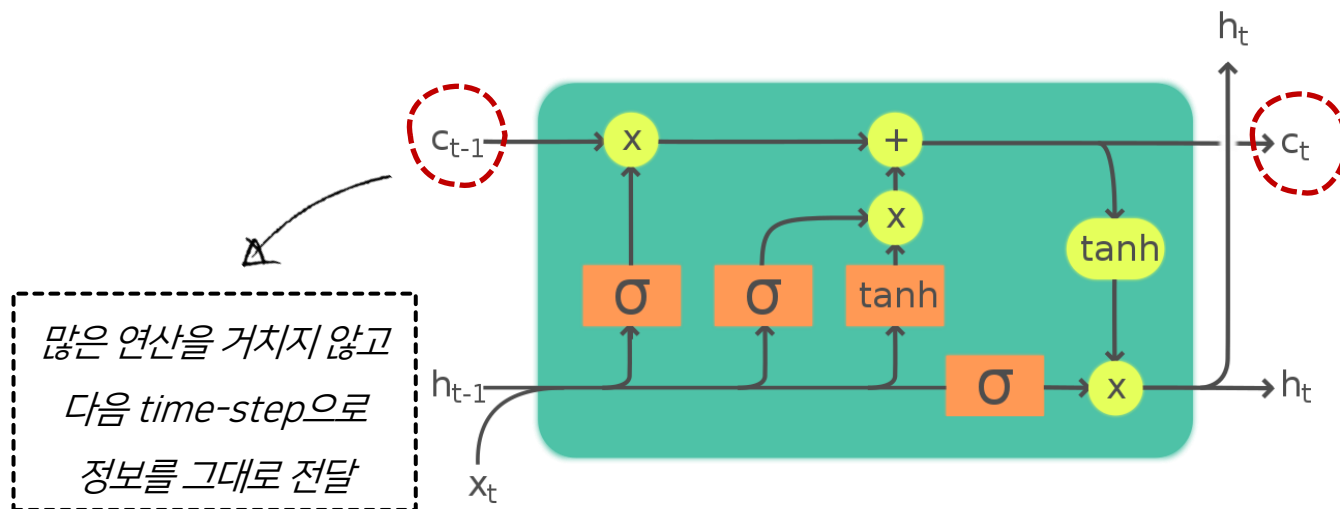
미안하다 이거 보여주려고
어그로 끌었다.. 나루토 사스
케 싸움수준... 아무튼 나루
토는 진짜 애니중최거명작임

장기기억을 담당하는
 c_t (Cell State)의 도입

I'm sorry. To show you
this, I've focused your
attention. Isn't it
amazing how Naruto Sa

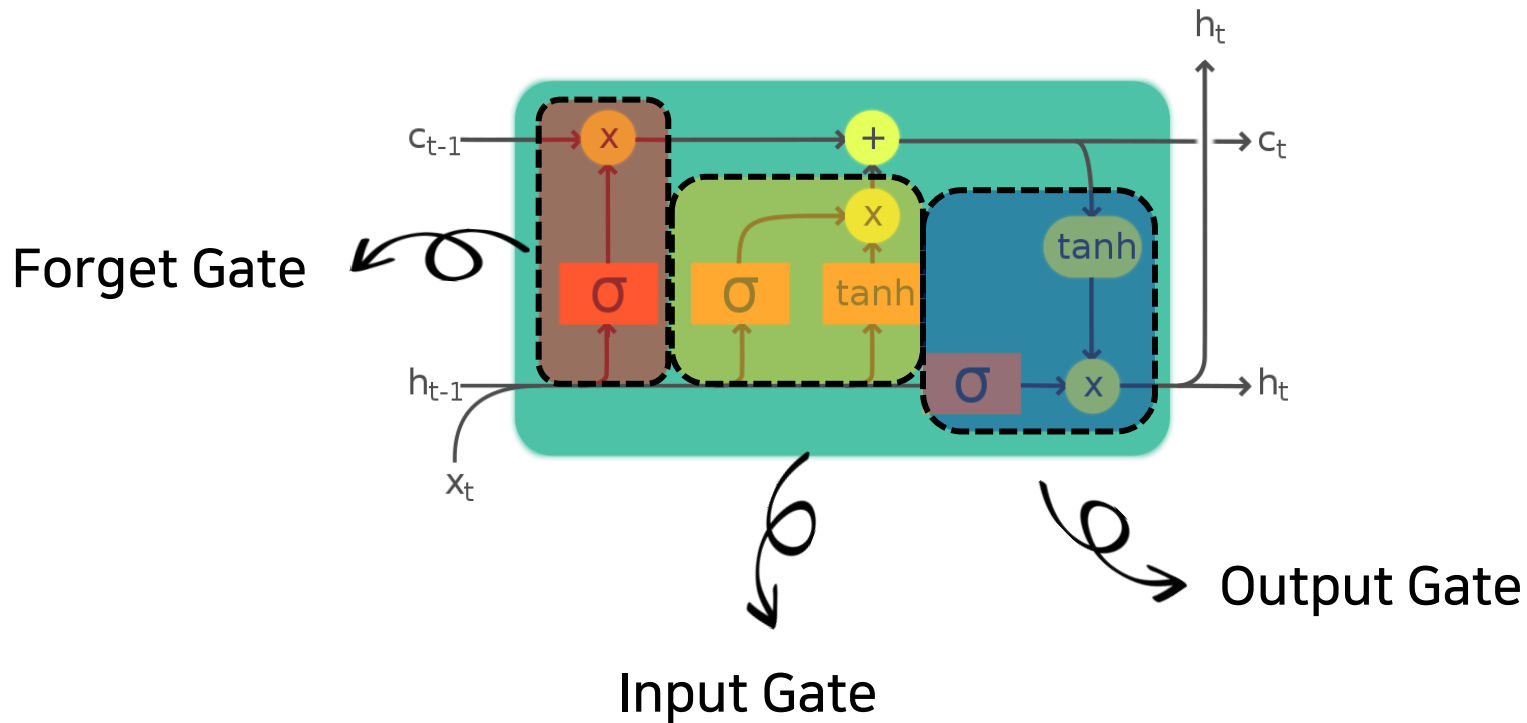
LSTM

LSTM의 구조



LSTM

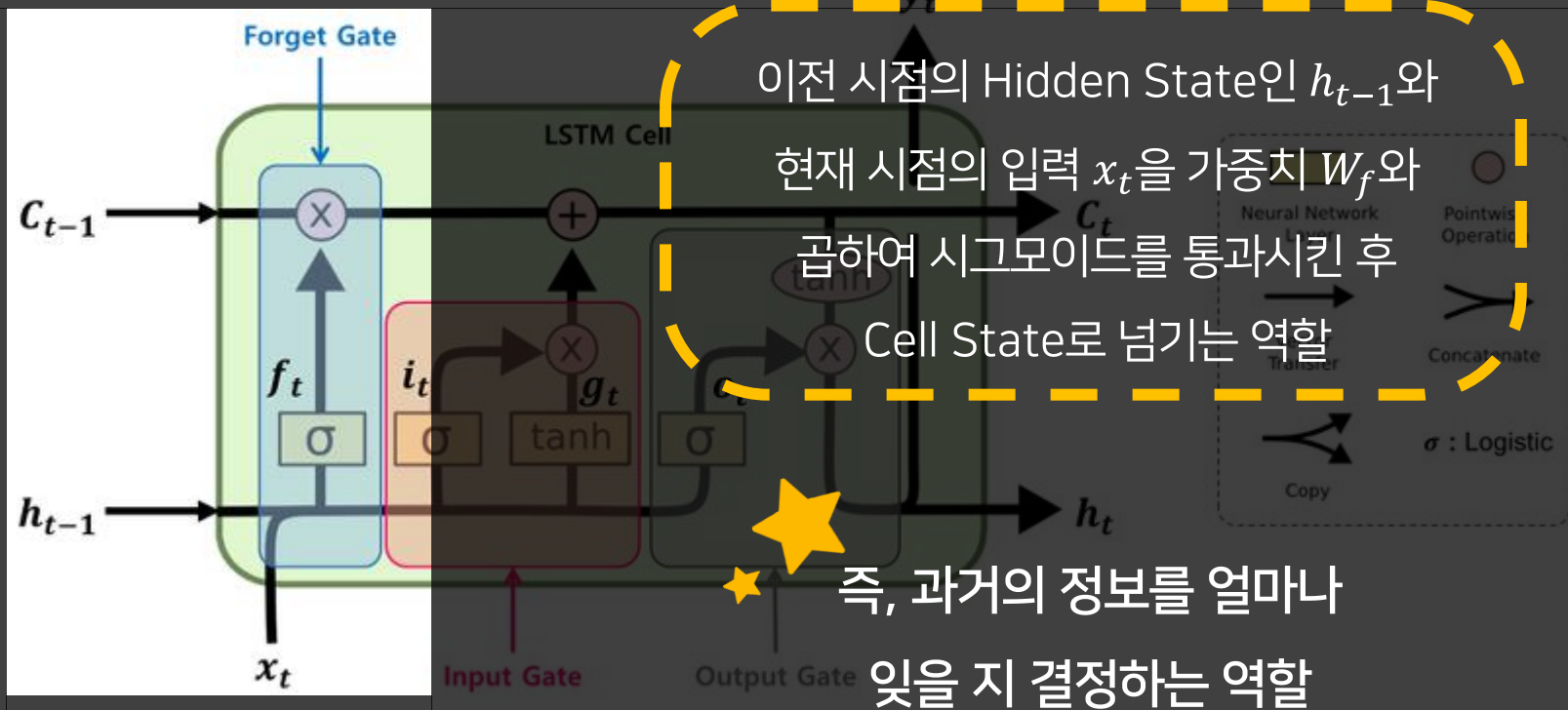
LSTM의 구조 - Gate



각각의 State에 어떤 값을 저장할지 정해주는 3가지 gate가 존재

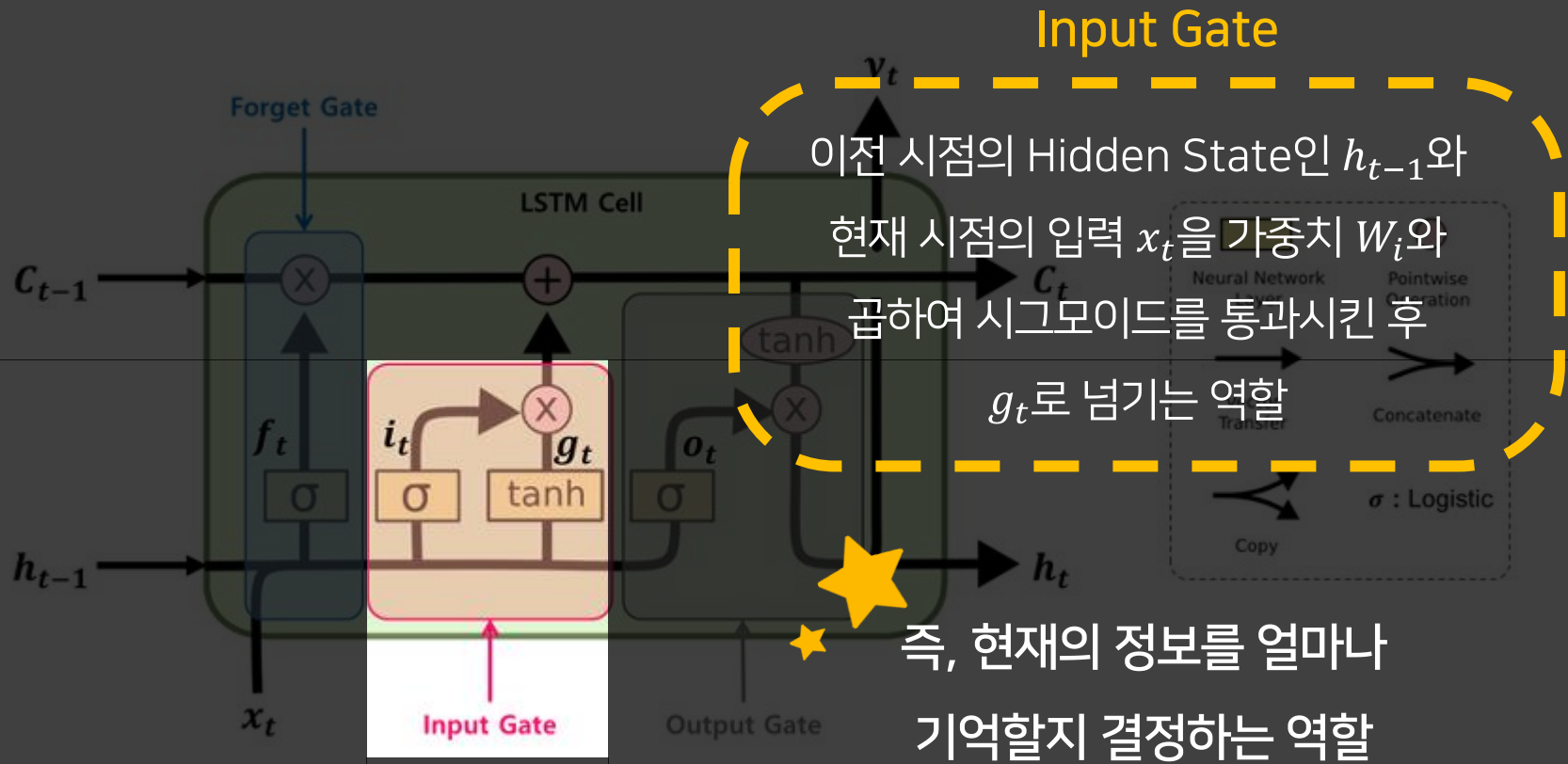
LSTM

Hidden State, Cell State



LSTM

Hidden State, Cell State



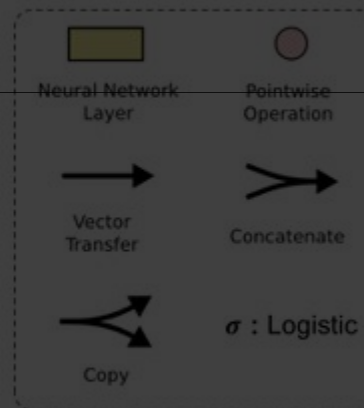
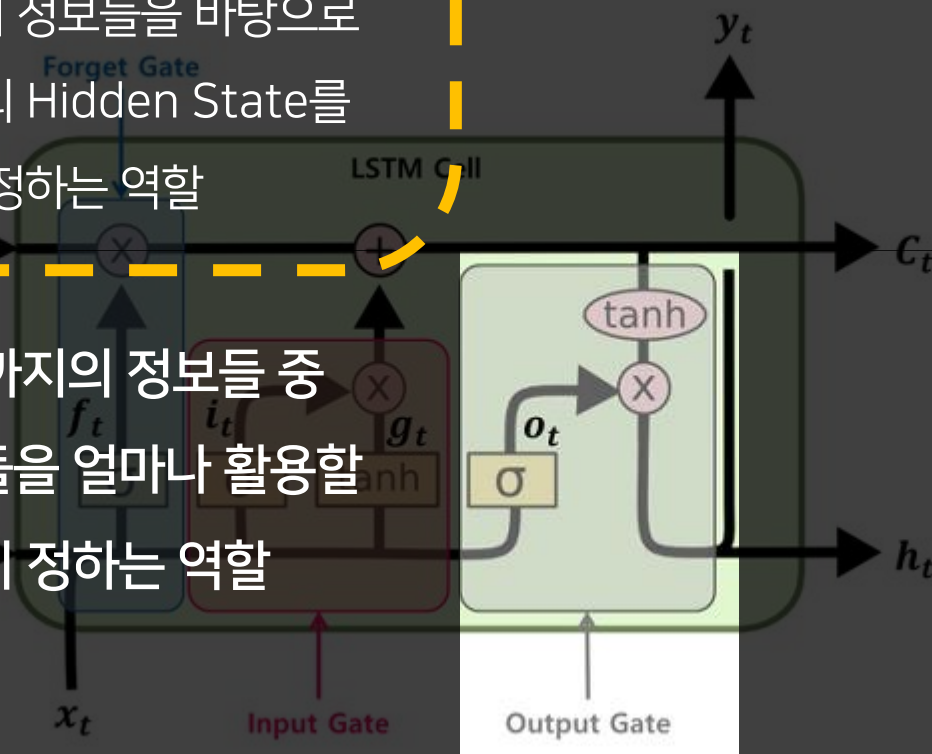
Output Gate

LSTM

Hidden State, Cell State

현재 시점의 Cell State와
이전 시점의 정보들을 바탕으로
현재 시점의 Hidden State를
결정하는 역할

즉, 현재까지의 정보들 중
어떤 정보들을 얼마나 활용할
것인지 정하는 역할



LSTM

Gate들의 수식

$$f_t = \sigma(W_f \times [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \times [h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \times [h_{t-1}, x_t] + b_o)$$

“얼마나”를 0~1사이의 출력값을 가지는
시그모이드 함수가 반영

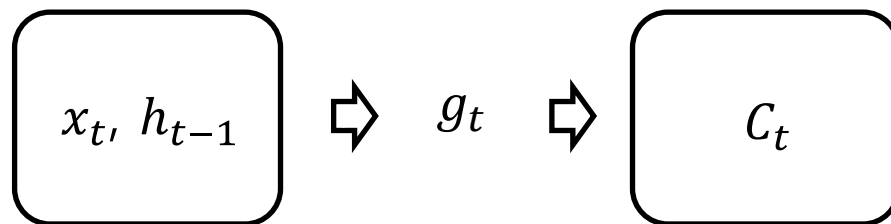
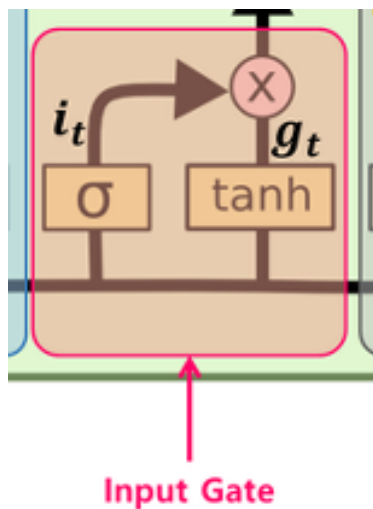
Forget Gate: 과거의 정보를 얼마나 잊을지 결정하는 역할

Input Gate: 현재의 정보를 얼마나 기억할지 결정하는 역할

Output Gate: 현재까지의 정보들 중 어떤 정보들을 얼마나
활용할 것인지 정하는 역할

LSTM

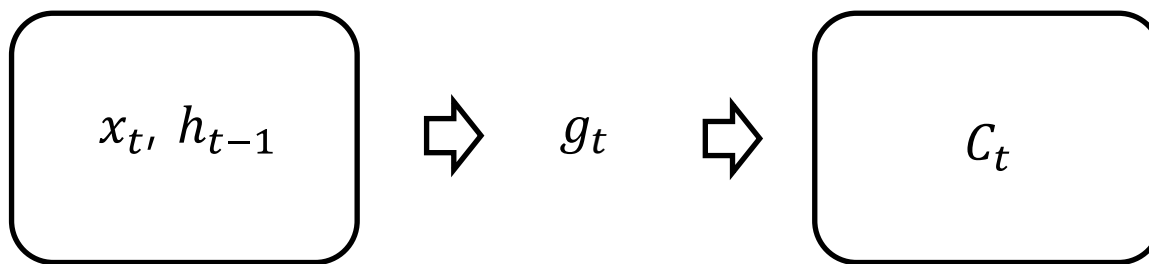
LSTM의 작동 과정



i_t 는 현재의 정보를 얼마나 기억할지 결정하는 역할이었다면,
 g_t 는 현재의 정보 중 어떤 정보를 Cell State에 전달할 것인지 결정

LSTM

LSTM의 작동 과정



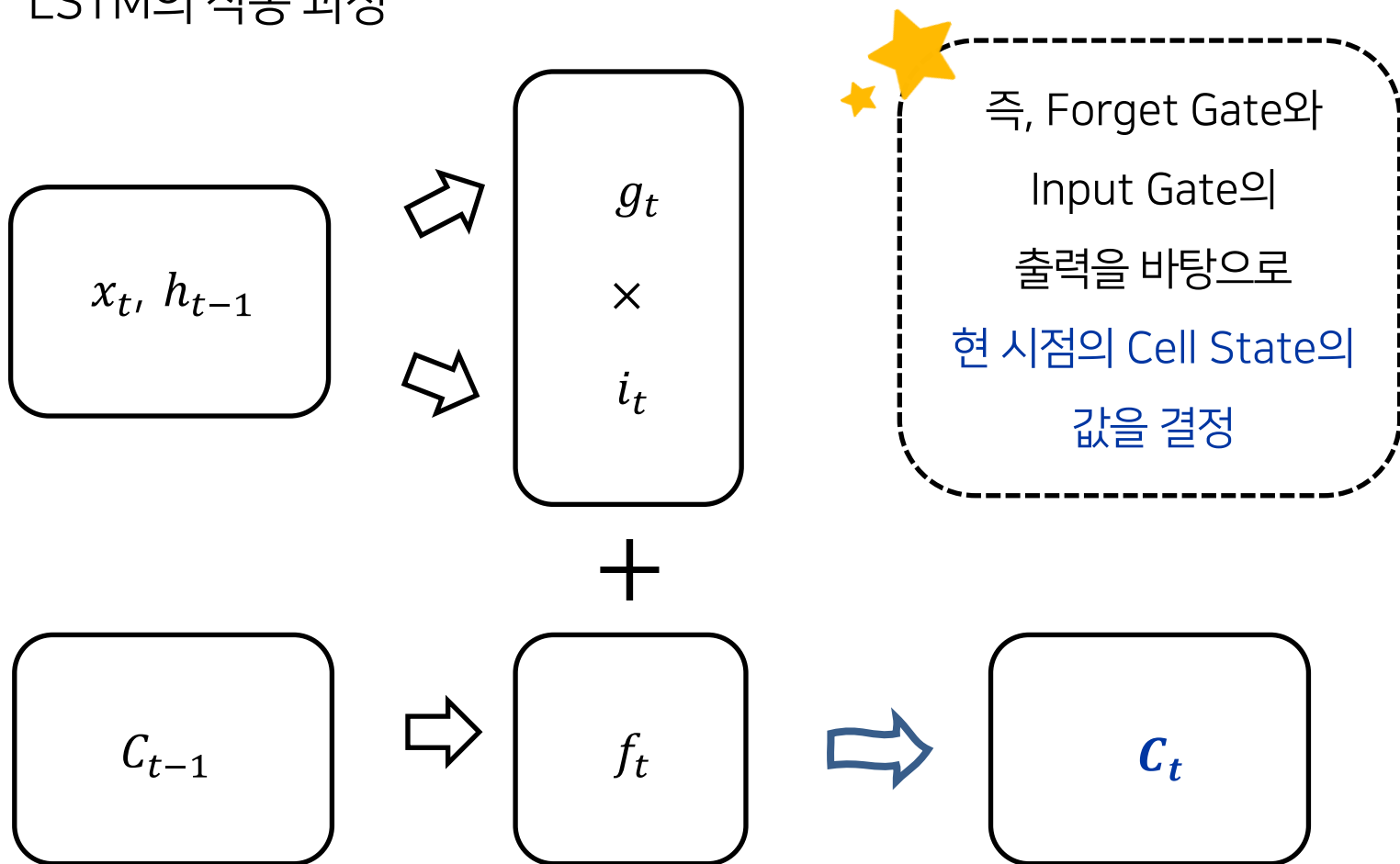
g_t 는 현재의 정보 중 **어떤** 정보를 Cell State에 전달할 것인지 결정

"어떤" 정보를 표현하기 위해서 tanh함수를 사용

$$g_t = \tanh(W_h \times [h_{t-1}, x_t] + b_i)$$

LSTM

LSTM의 작동 과정



LSTM

LSTM의 작동 과정

$$C_t = f_t \times C_{t-1} + i_t \times g_t$$

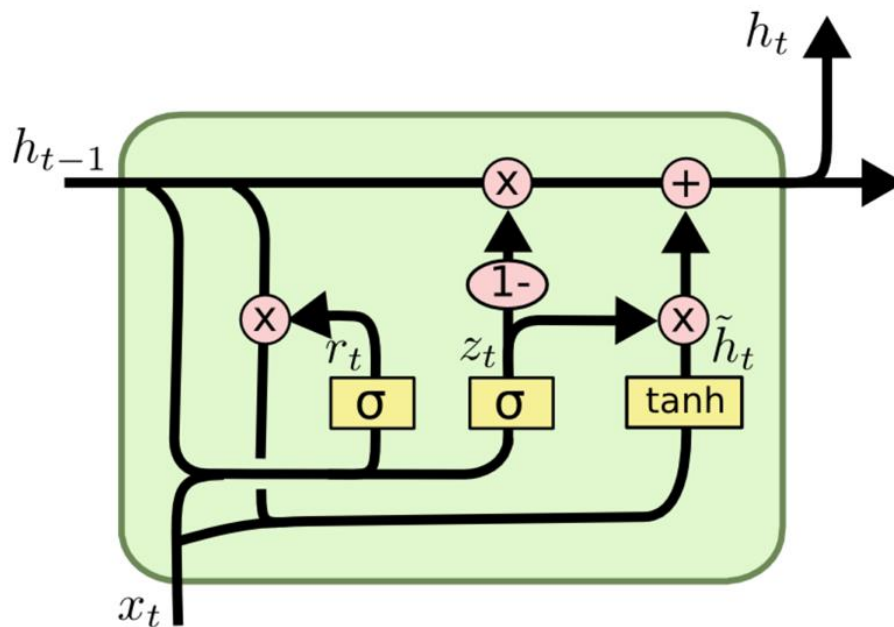


$$h_t = o_t \times \tanh(C_t)$$

최종적으로 Output Gate의 값과 Cell State의 값을 사용하여
현재시점의 Hidden State 값 결정

GRU (Gated Recurrent Unit)

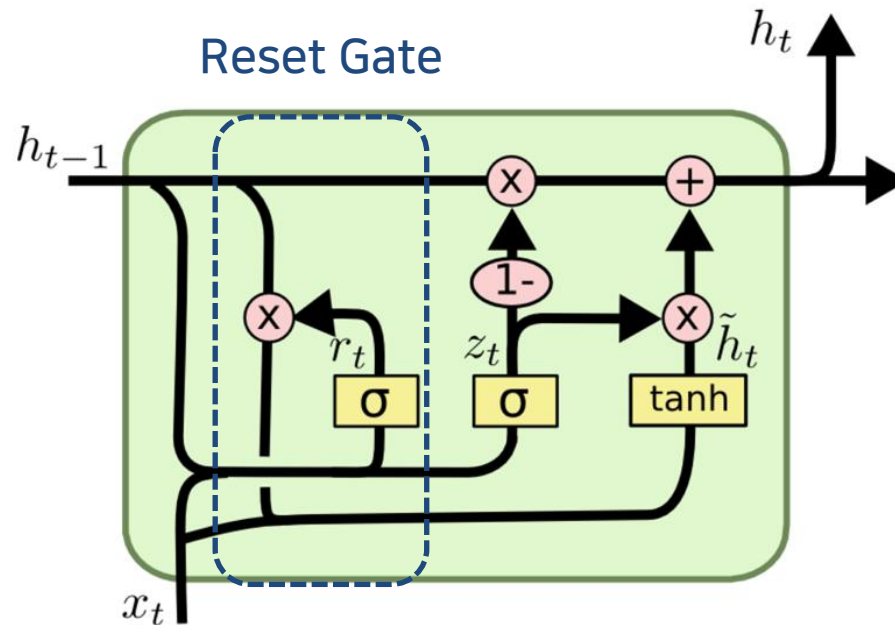
GRU 소개



LSTM의 구조를 간단하게 개선한 모델
Cell state가 없어지고 Gate의 구조가 변경됨

GRU (Gated Recurrent Unit)

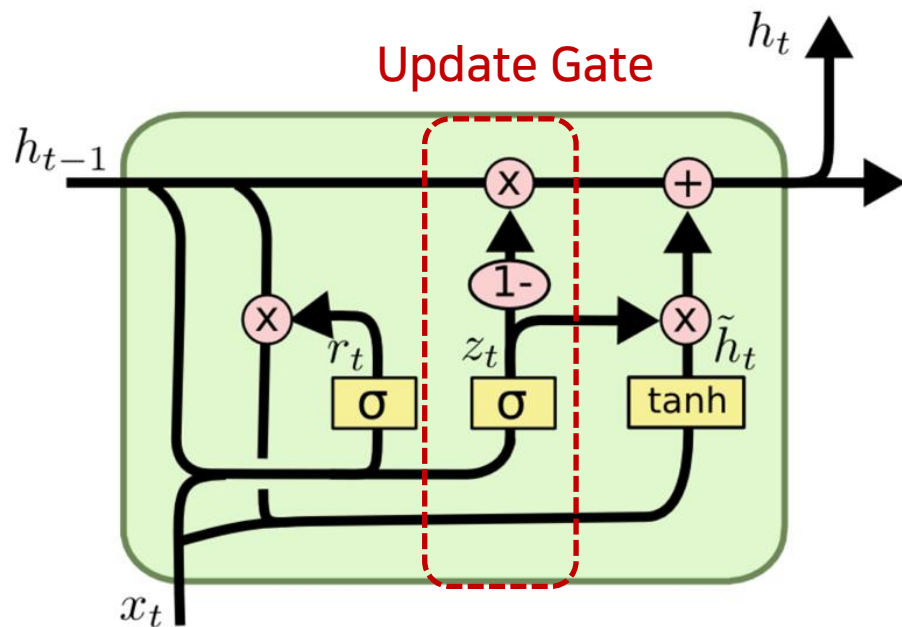
GRU 원리



r_t = 과거의 정보를 얼마나 유지할지 조정

GRU (Gated Recurrent Unit)

GRU 원리



Forget Gate와 Input Gate를 합친 개념!

z_t = 현재 정보를 얼마나 반영할지 조정

$1 - z_t$ = 과거 정보에 대해 얼마나 반영할지 조정

GRU (Gated Recurrent Unit)

GRU 원리



Update Gate

z_t 는 h_{t-1}, x_t 시그모이드 함수를 활용하여 (0, 1) 범위의 값을 반환하는
Gate Controller의 역할 수행

1에 가까우면 Forget gate 활성화
새로운 hidden state값이
이전 hidden state와 유사

0에 가까우면 Input gate 활성화
새로운 hidden state값이
현재 시점의 입력을 많이 반영

GRU (Gated Recurrent Unit)

GRU vs LSTM

LSTM의 Cell state 제거

Gate 개수 1개 감소



장점

파라미터 수 감소

연산 속도 증가

그러나 실제로 사용해보면
성능에 있어서 큰 차이가 존재하지 않음

3

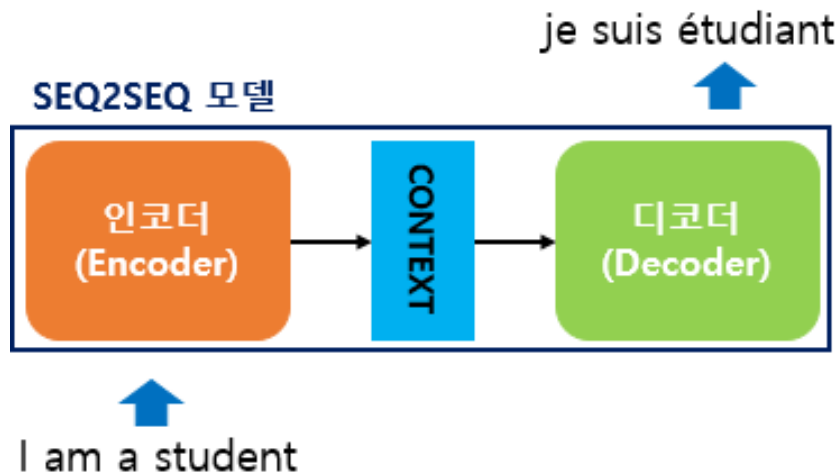
RNN 모델의 응용

Seq2Seq

Seq2Seq란?

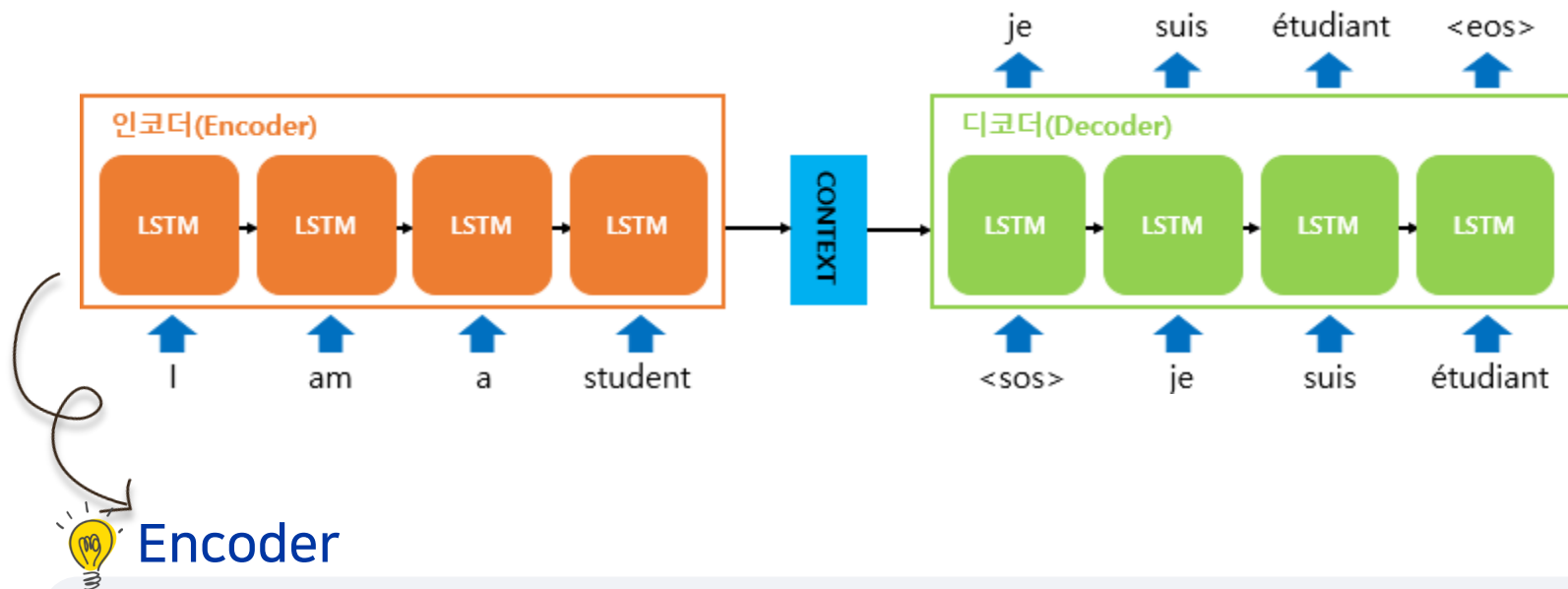
인코더-디코더 구조로 각각에 서로 다른 RNN 계열의 모델을 넣어 만든 모델

출력과 입력의 길이가 달라도 된다는 점에서
번역, 텍스트 요약 과제 등에 사용됨



Seq2Seq

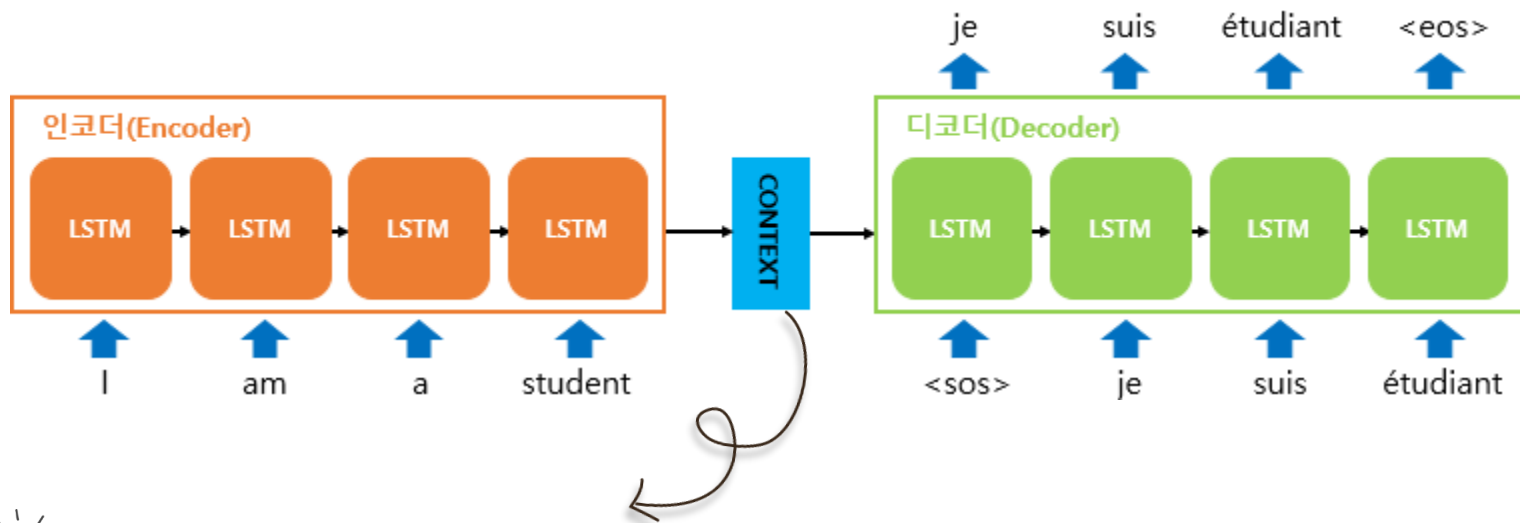
Encoder-Decoder 구조



Sequential Data를 입력 받아 압축된 하나의 벡터로 만드는 역할

Seq2Seq

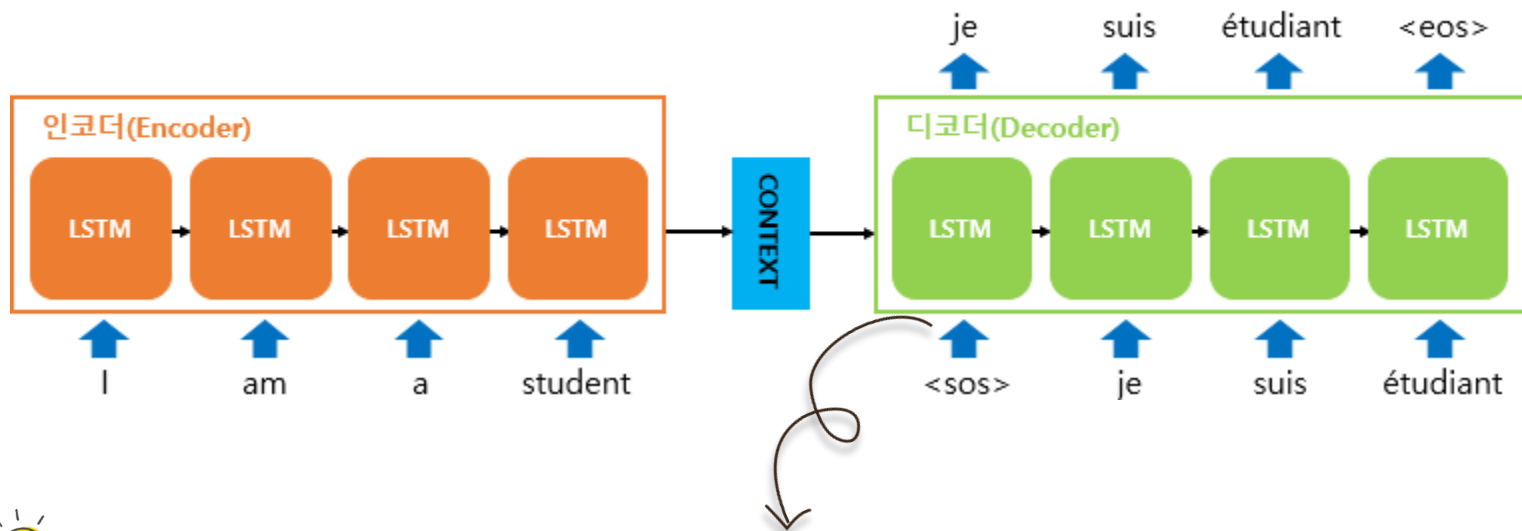
Encoder-Decoder 구조

**컨텍스트 벡터(Context Vector)**

입력 문장의 의미를 하나의 벡터로 **압축**시킨 형태로,
Encoder의 출력이자 동시에 Decoder의 첫 Hidden State으로 사용함

Seq2Seq

Encoder-Decoder 구조



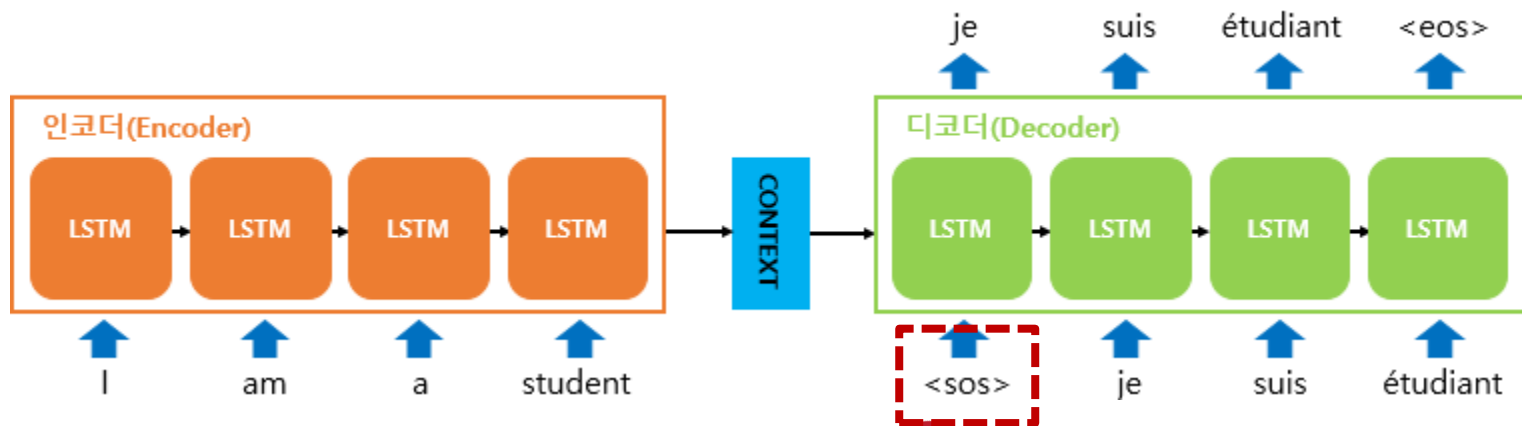
Decoder

Encoder의 마지막 Hidden State와 입력을 바탕으로
Encoder와 달리 **매 시점마다 출력을 내보냄**

이때 사용되는 매 시점의 입력은 이전 시점의 출력임

Seq2Seq

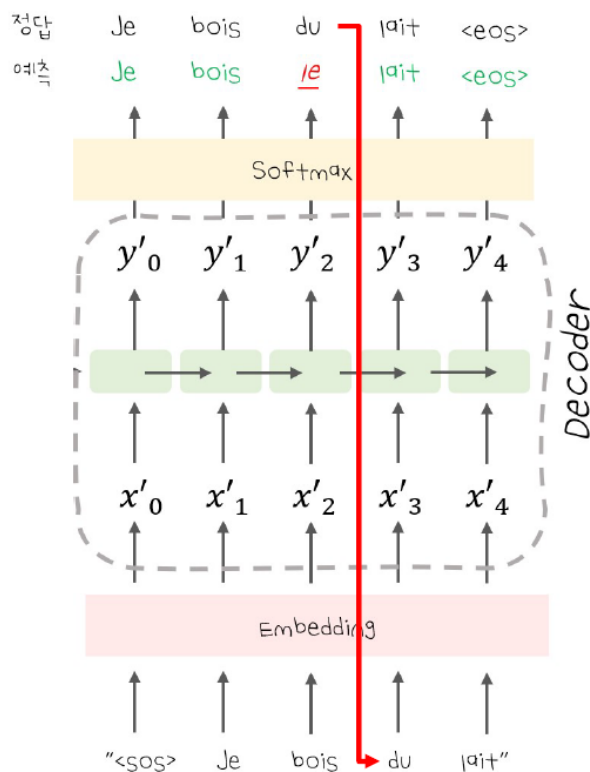
Encoder-Decoder 구조



그러나 첫번째 시점에는 이전 시점의 출력이 존재하지
않기 때문에 sos(start to sequence) 토큰을 사용

Seq2Seq

Encoder-Decoder 구조



Decoder가 초반부 예측을
잘못할 경우 전체 학습 과정을 저해

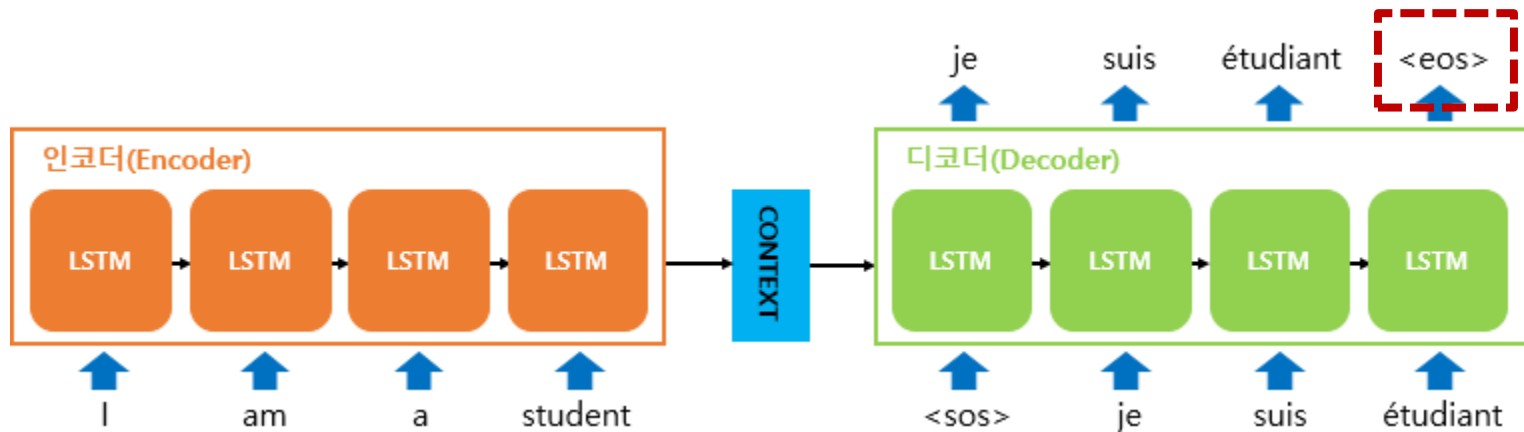


교사 강요(Teacher Forcing)

매 시점 출력과는 별개로
다음 Decoder 입력에
실제 정답을 넣음

Seq2Seq

Encoder-Decoder 구조



이 과정을 활용해 문장 끝까지 도달하면
Decoder는 eos 토큰을 반환하며 문장이 끝났음을 알림

다음 내용은 ~

NewJeans - Attention



근데 이제 딥 팀장님의 ~~팬심~~을 곁들인,,,,,
팬심은 아니고 집중 환기용이라고 써달라고 하십니다

Attention



Encoder-Decoder 구조에서 Encoder에서 문장의 의미를
하나의 압축된 벡터로 만들면서 **두가지 문제** 발생

①

병목현상(Bottleneck Problem)

Encoder에서 전체 input을
하나의 벡터 형태로 압축시키면서
정보의 손실 발생

②

Gradient Vanishing Problem

RNN에서 Cell 이 늘어날수록
Gradient Vanishing 문제 발생

Attention



Encoder-Decoder 구조에서 Encoder에서 문장의 의미를 하나의 압축된 벡터로 만들면서 **두가지 문제** 발생

①

병목현상(Bottleneck Problem)

②



Gradient Vanishing Problem



이를 해결하기 위해 Attention구조 등장

Encoder에서 문장의 의미를 하나의 벡터 형태로 압축시키면서
정보의 손실 발생

encoder가 길어질수록
gradient vanishing 문제 발생

Attention

Attention이란?

우리가 찾고자 하는 값과 유사도를 바탕으로
얼마나 그 내용에 집중해야 하는지를 정하는 것

내적을 이용한 Attention의 원리와 방법에 대해 알아볼 것



코사인 유사도란?

두 벡터 x 와 y 의 내적

$$x \cdot y = |x||y| \cos \theta$$



내적 공간의 두 벡터 간 각도의
코사인 값을 이용하여 측정된
벡터 간의 유사한 정도를 의미

Attention

Attention이란?

우리가 찾고자 하는 값과 유사도를 바탕으로
얼마나 그 내용에 집중해야 하는지를 정하는 것

내적을 이용한 Attention의 원리와 방법에 대해 알아볼 것



코사인 유사도란?

두 벡터 x 와 y 의 내적

$$x \cdot y = |x||y| \cos \theta$$



내적 공간의 두 벡터 간 각도의
코사인 값을 이용하여 측정된
벡터 간의 유사한 정도를 의미

Attention

Attention이란?

우리가 찾고자 하는 값과 유사도를 바탕으로
얼마나 그 내용에 집중해야 하는지를 정하는 것

따라서 내적을 이용하여 유사도를 구할 수 있는 것
내적을 이용한 Attention의 원리와 방법에 대해 알아볼 것



코사인 유사도란?

두 벡터 x 와 y 의 이 내적 방법을 이용해

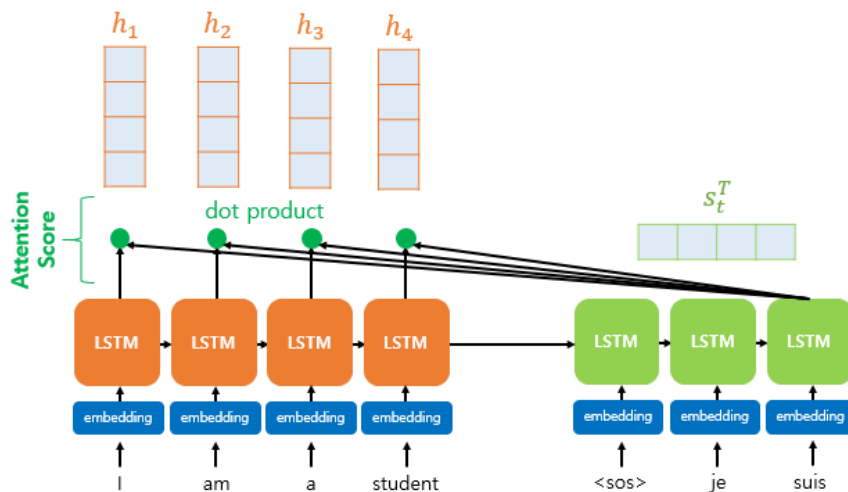
$x \cdot y = |x||y| \cos \theta$ 이 내적 방법을 이용해
Attention Score를 구해보자

원리공간의 두 벡터 간 각도의
코사인 값을 이용하여 측정된
벡터 간의 유사한 정도를 의미

Attention

Attention의 진행 과정

① Attention Score 계산



h_t : Encoder의 각 시점의 hidden state

s_t : Decoder의 각 시점의 hidden state

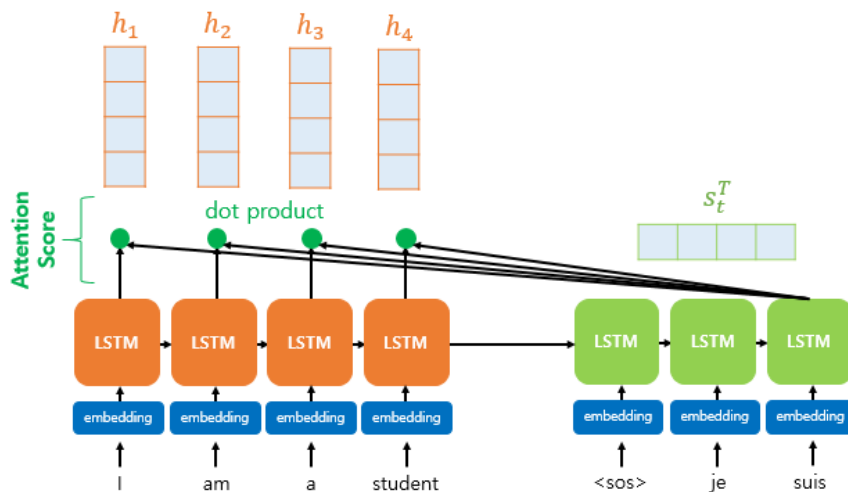
Attention Score

Decoder의 매 시점에 출력 반환 전에
해당 시점의 hidden state와
인코더 각 시점의 hidden state의
유사도를 구해 출력에 반영

Attention

Attention의 진행 과정

① Attention Score 계산



h_t : Encoder의 각 시점의 hidden state

s_t : Decoder의 각 시점의 hidden state

디코더의 매 시점에 출력을 반환하기

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

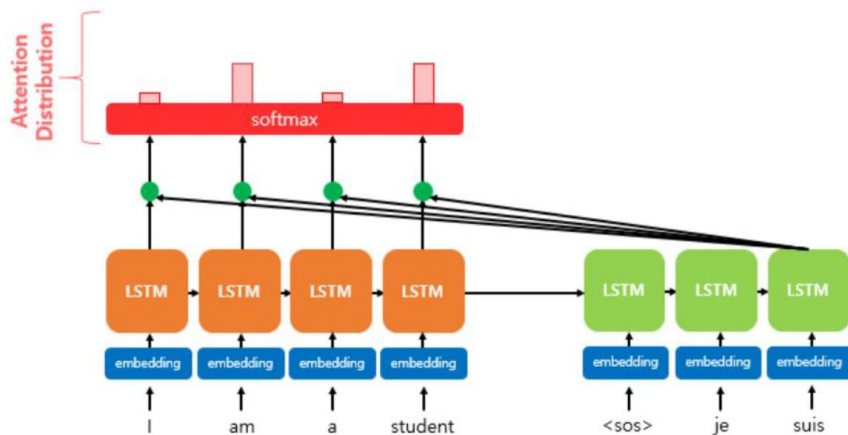
$$e^t = [s_t^T h_1, s_t^T h_2, \dots, s_t^T h_n]$$

➡ Attention Score

Attention

Attention의 진행 과정

② Attention Distribution 구하기



Attention score에 softmax 함수를
적용해 0과 1 사이의 값으로

정규화된 확률을 얻음

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

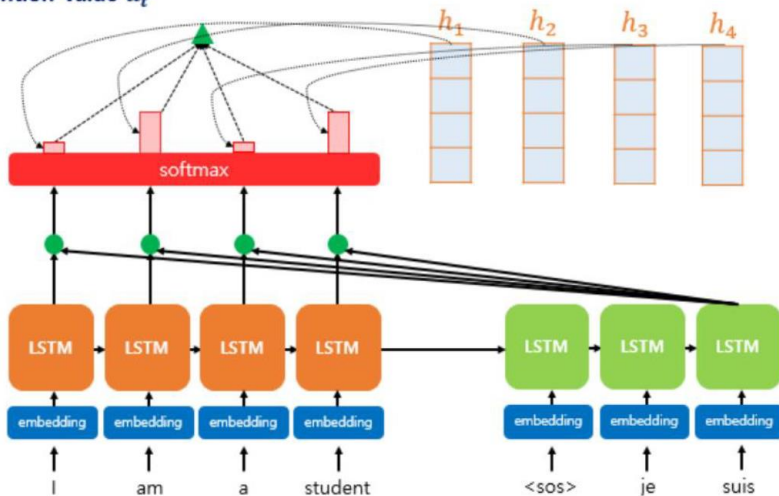
*Encoder의 각 시점의 hidden state들이
decode의 hidden state에 가지는 중요도를 의미*

Attention

Attention의 진행 과정

③ Attention Value 구하기

Attention Value α_t



$$a^t = \sum_{i=1}^n a_i^t h_i = a^t \cdot h$$

where $h = [h_1, \dots, h_n]$

앞서 구한 Attention Distribution을
Encoder의 모든 hidden state와
곱해 가중합 효과 얻음



Attention Value는 Attention모델에서의 Context Vector

Attention

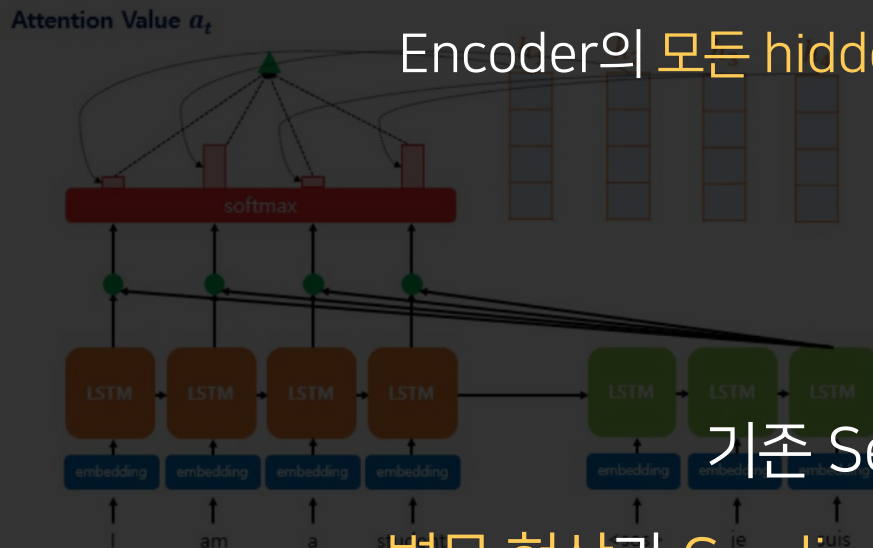
Attention의 진행 과정

③ Attention을 사용하는 Seq2Seq에서는 h_t 만을 Context Vector로 사용했지만

Attention모델에서는 a_t 를 사용하면서

앞서 구한 Attention Distribution을
Encoder의 모든 hidden state $[h_1, \dots, h_n]$ 을 활용
Encoder의 모든 hidden state를

곱해 가중합 효과 얻음



기존 Seq2Seq의

병목 현상과 Gradient Vanishing 문제 해결

$$a^t = \sum_{i=1}^n a_i^t h_i = a^t \cdot h$$

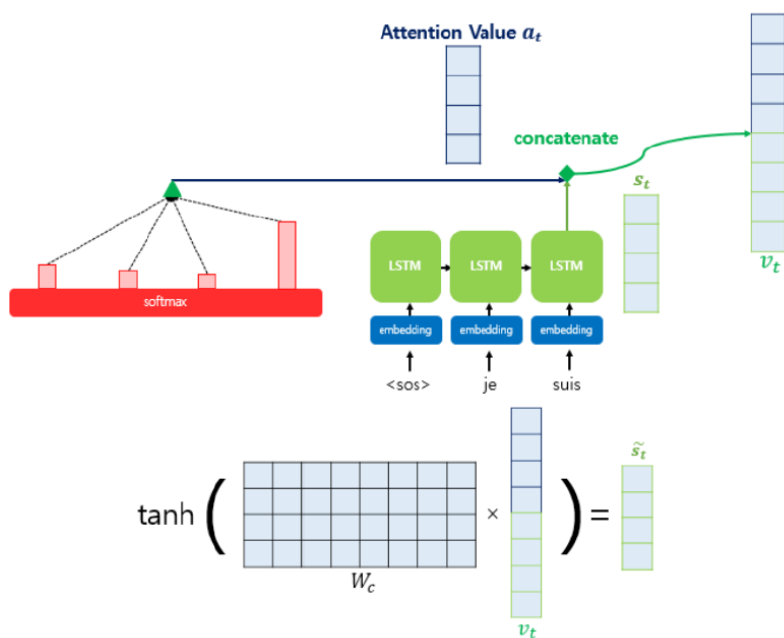
where $h = [h_1, \dots, h_n]$



Attention

Attention의 진행 과정

④ Decoder의 Hidden state와 연결하기

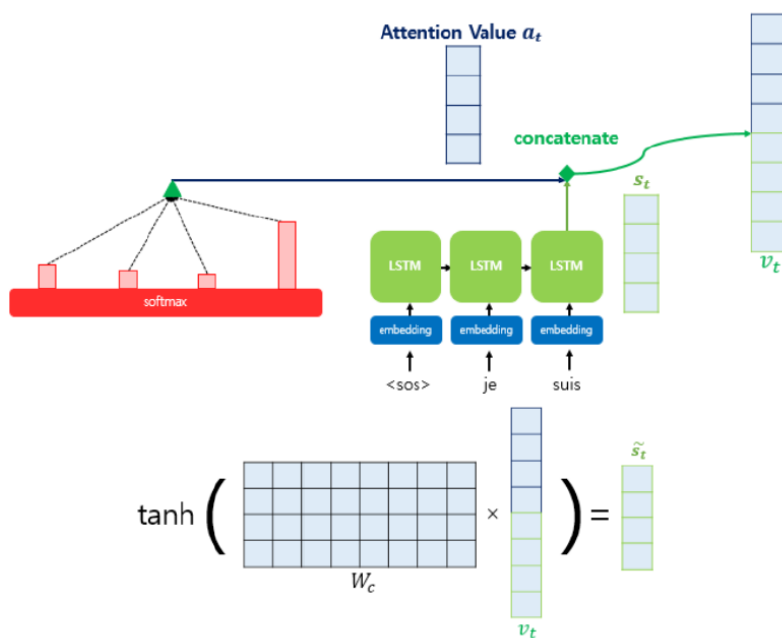


Attention Value와
해당 시점 디코더의 hidden state를
연결해 하나의 벡터 v_t 를 구함

Attention

Attention의 진행 과정

⑤ Decoder 출력 계산



벡터 v_t 를 가중치와 연산하여
출력층의 입력을 만들어 줌

$\tilde{s}_t = \tanh(W_c \cdot v_t + b_c)$ 가
출력층의 입력이 됨

기존 Encoder-decode 구조에서는
 h_t 이 바로 출력층의 입력



“그동안 딥러닝 클린업을 들어주신 여러분에게 Cheers”



THANK YOU

