

Dependency Parsing

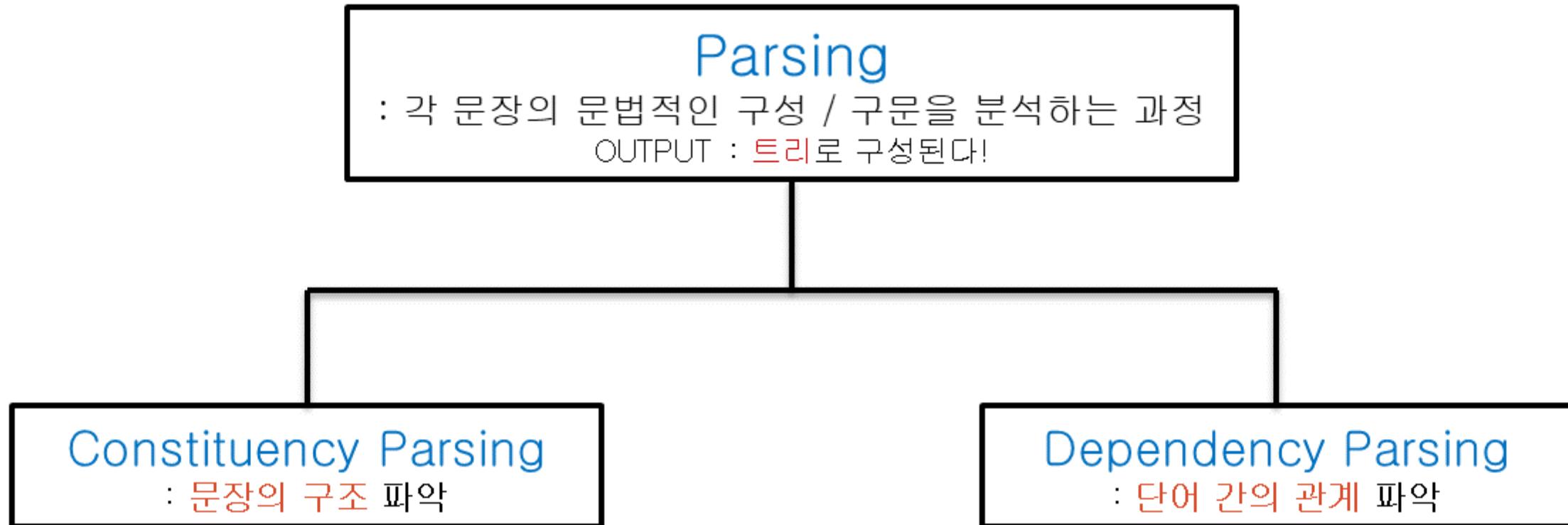
STUDY OBJECTIVES

1. Dependency Parsing?

3. Methods

2. HOW?

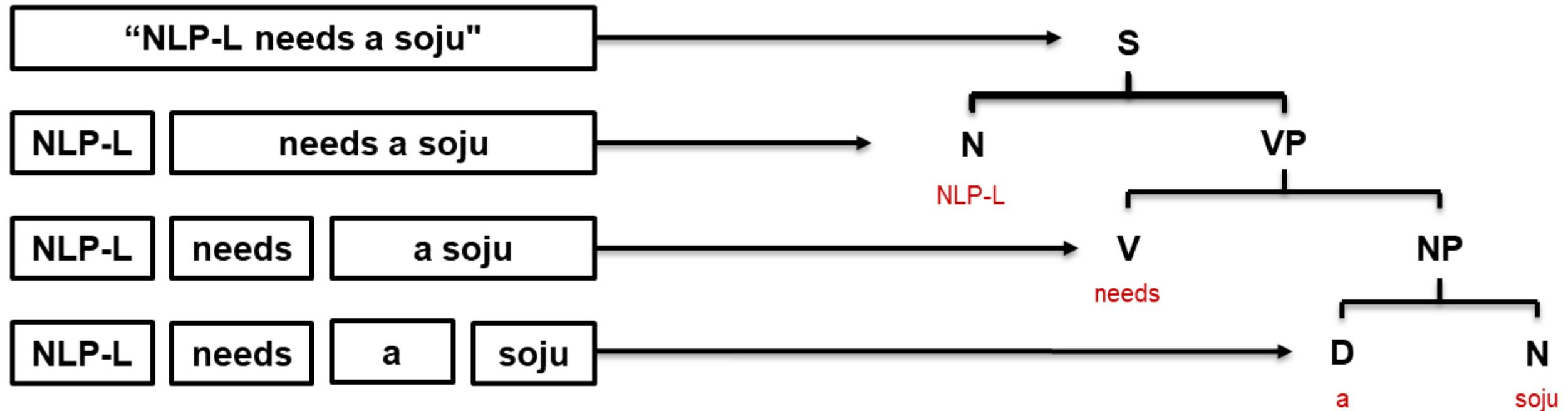
Dependency Parsing?



Dependency Parsing?

[Constituency Parsing]

문장을 성분으로 나누어 문장의 구조를 파악하는 방법



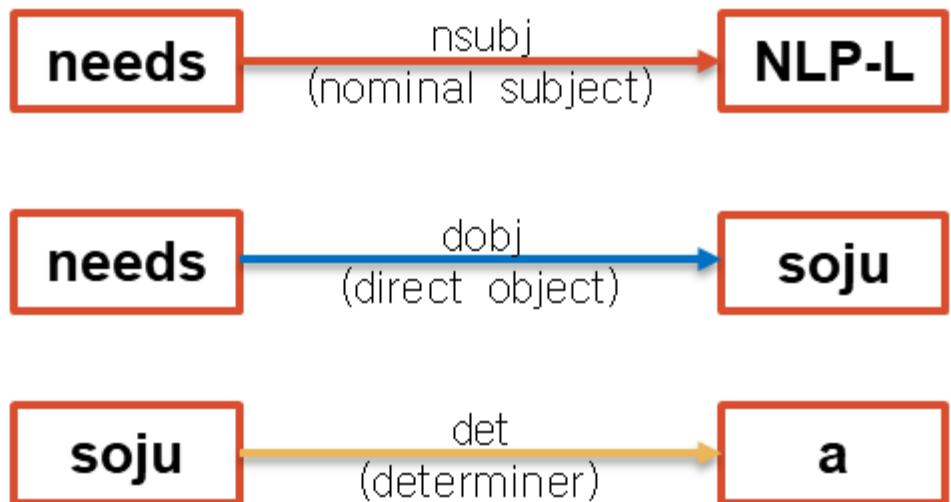
단어 간의 관계와 의미를 파악할 수 없다!!

Dependency Parsing?

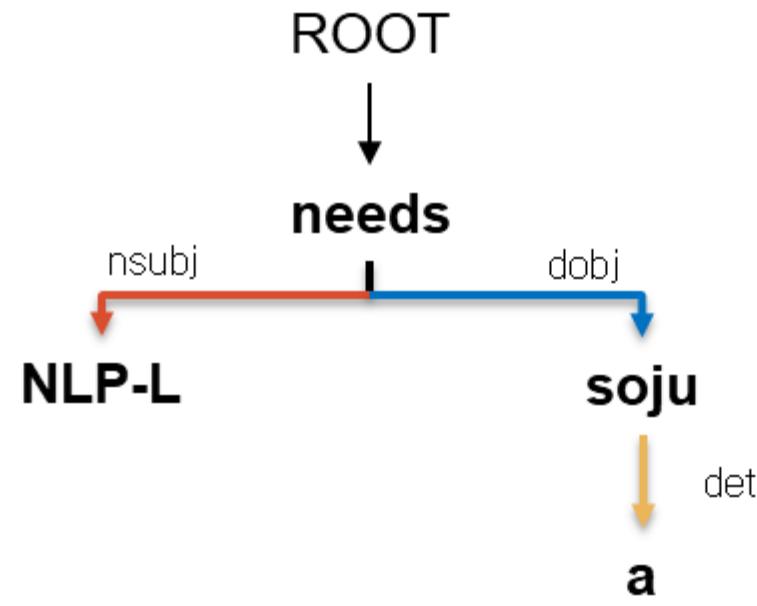
[Dependency Parsing]

단어간 의존 / 수식 관계를 파악하는 방법

“NLP-L needs a soju”



ROOT



Why Dependency Parsing?

[Coordination Scope Ambiguity]

특정 단어가 수식하는 대상의 범위가 달라져 중의적으로 해석되는 모호성

ex) 게으른 토끼와 거북이가 경주를 한다

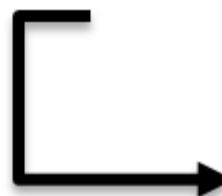
1. 게으른 (토끼와 거북이가) 경주를 한다
2. (게으른 토끼)와 거북이가 경주를 한다

[Phrase Attachment Ambiguity]

형용사구, 동사구, 전치사구 등이 어떤 단어를 수식하는지에
따라 의미가 달라지는 모호성

ex) “I saw a girl with a telescope”

1. 나는 망원경을 통해 소녀를 보았다.
2. 나는 망원경을 가지고 있는 소녀를 보았다.



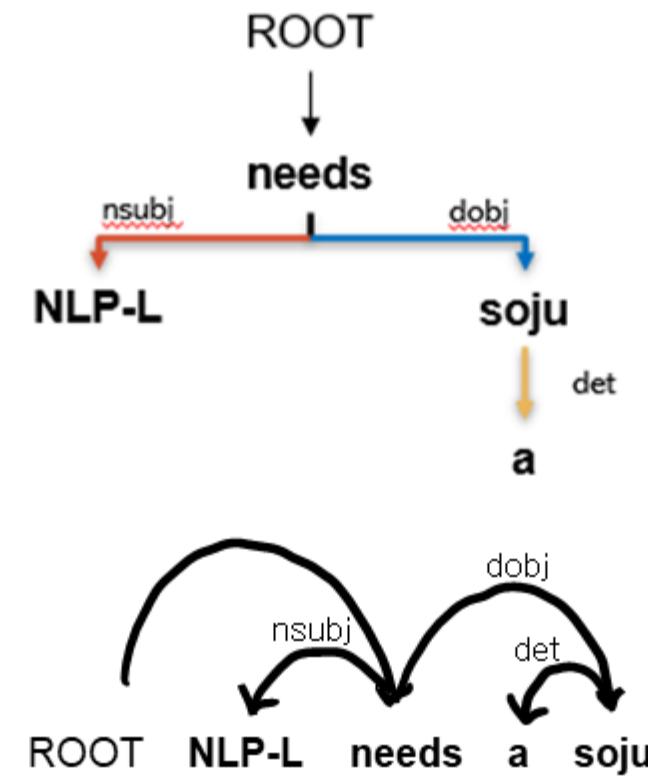
단어 간의 수식관계를 이해해야 할 필요성이 생긴다!

그 필요성을 반영한 방식이 **Dependency Parsing!**

HOW?

[Grammar and Structure]

- 아웃풋 구조는 두 가지 형태로 표현 가능
- Head -> Dependent
 - * Head : “수식을 하는” “
 - * Dependent : “수식을 받는” “
 - > Dependent는 Head의 ~ 다
- 화살표 위 label은 단어간 문법적 관계(의존성) 의미
- **화살표는 순환하지 않는다** : 결과물을 트리 형태로 표현 가능!



- **ROOT**: 어떤 단어의 수식도 받지 않는 (화살표를 받지 않는) 단어를 수식하는 가상의 node
-> 모든 단어가 최소한 1개의 dependent가 되도록 설정!

HOW?

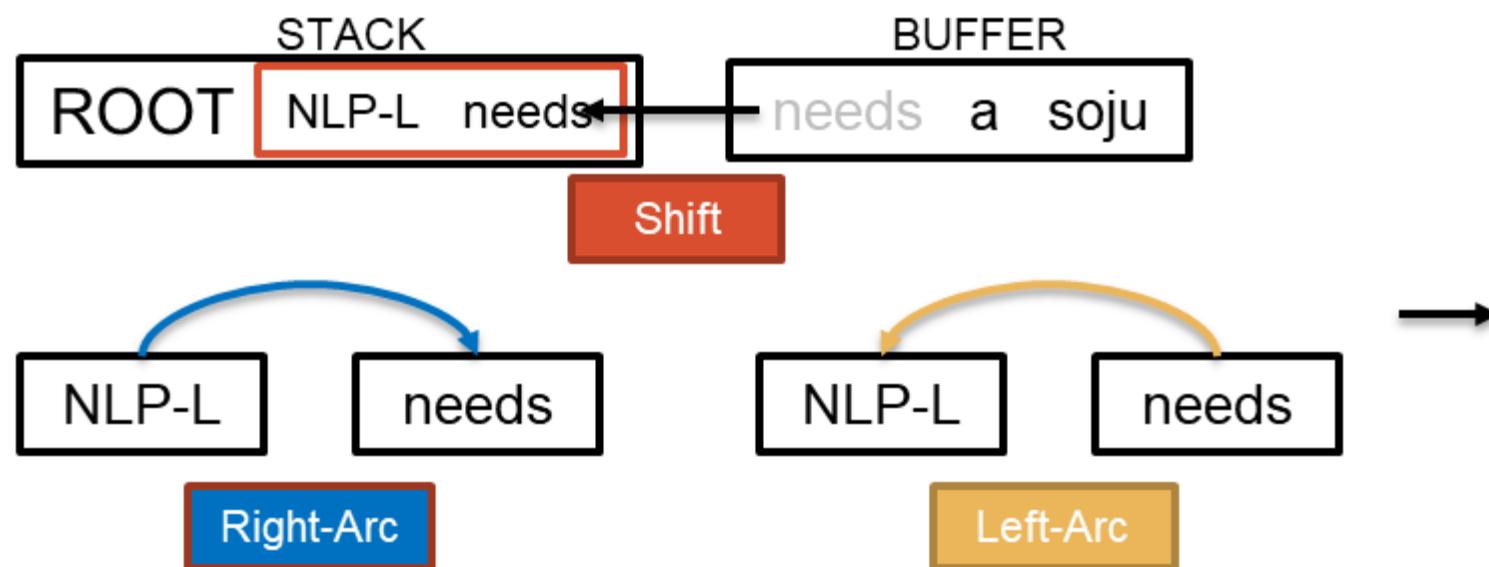
Transition-Based Parsing

Transition-Based

- 두 단어의 의존여부를 **순서대로 결정**
 > 점진적으로 트리모형을 생성
 - * 점진적 : 두 단어씩 수식관계를 파악하고, 그 다음 단계로 넘어감

Graph-Based

- 가능한 의존관계를 **모두 고려**
 > 가장 확률이 높은 트리모형을 선택

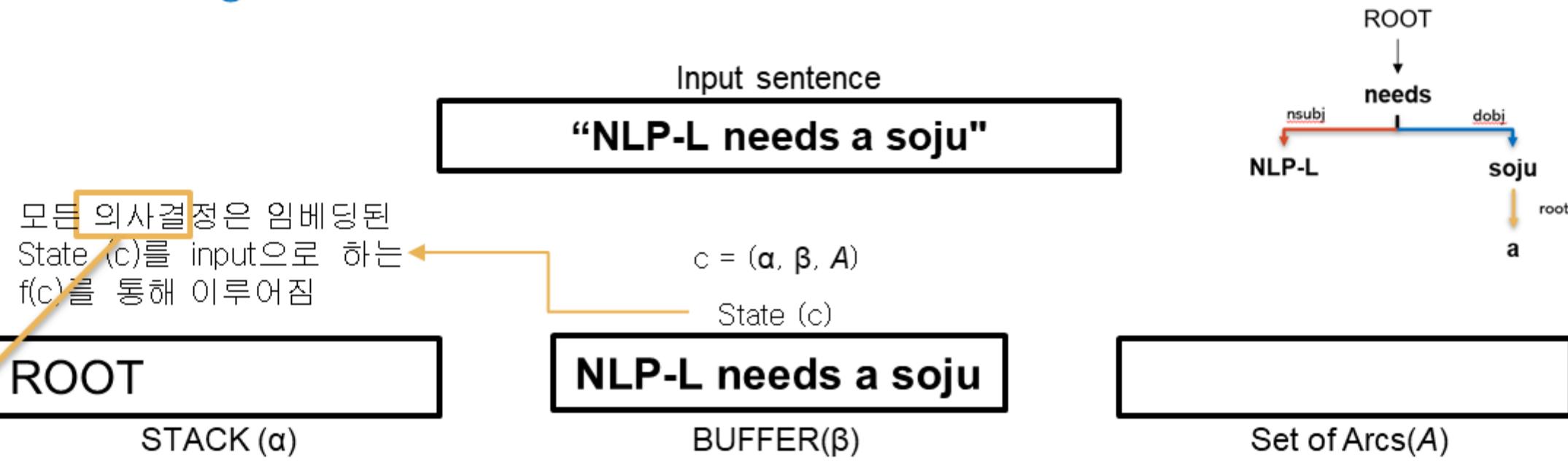


Graph-based 방식보다 빠르지만 낮은 성능

2014년 개선되어 속도와 성능 모두 업!!

Transition-Based Parsing

Parsing



SHIFT

BUFFER \rightarrow STACK으로 토큰이 이동하는 것

Left-Arc

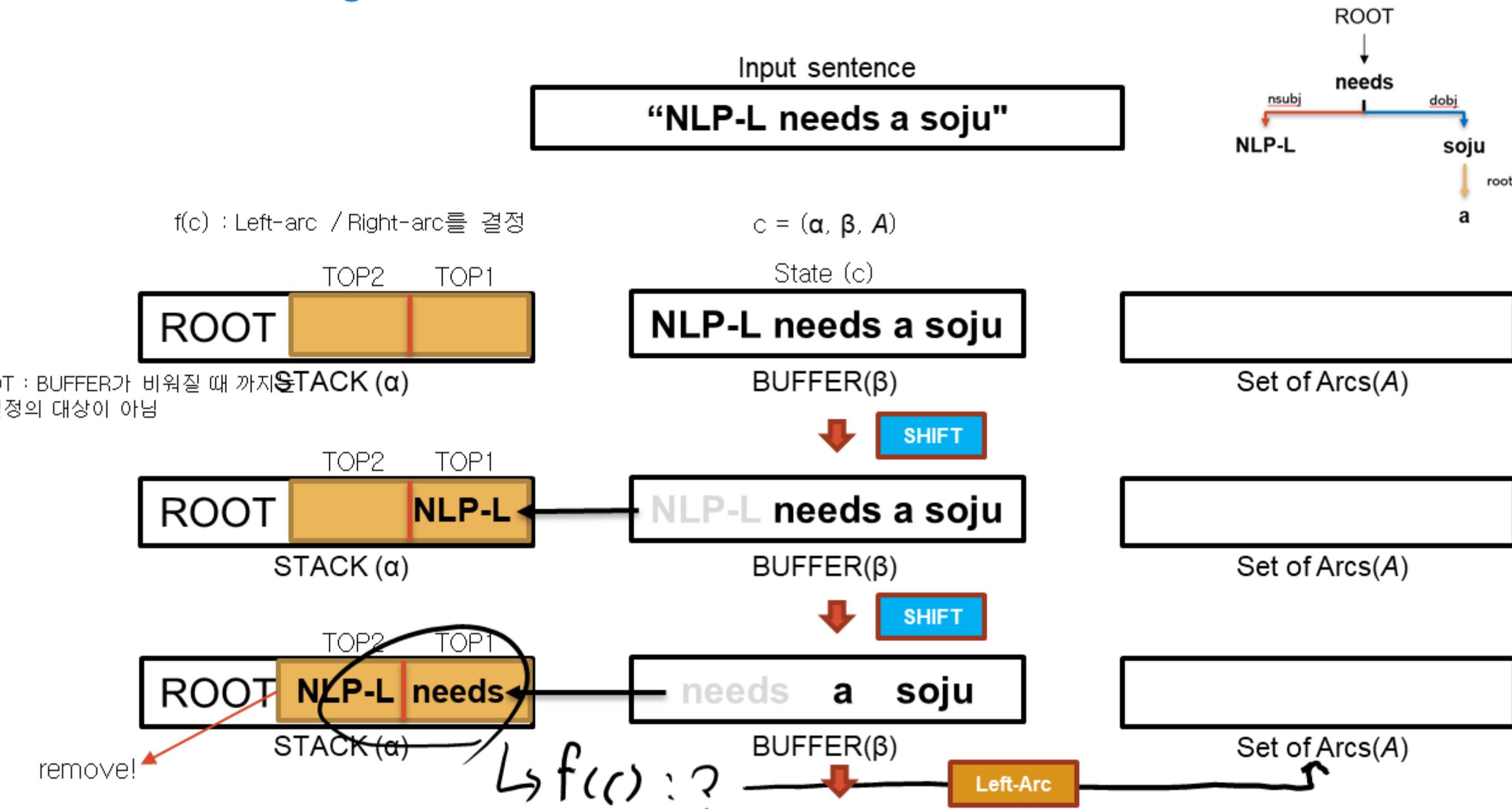
Right-Arc

화살표의 방향을 결정하는 과정.

스택(first in last out)의 TOP 1,2단어 간의 화살표 방향에 따라 결정된다

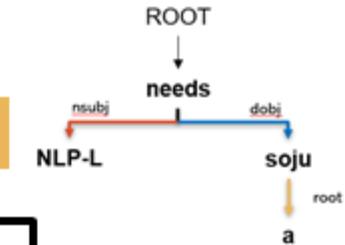
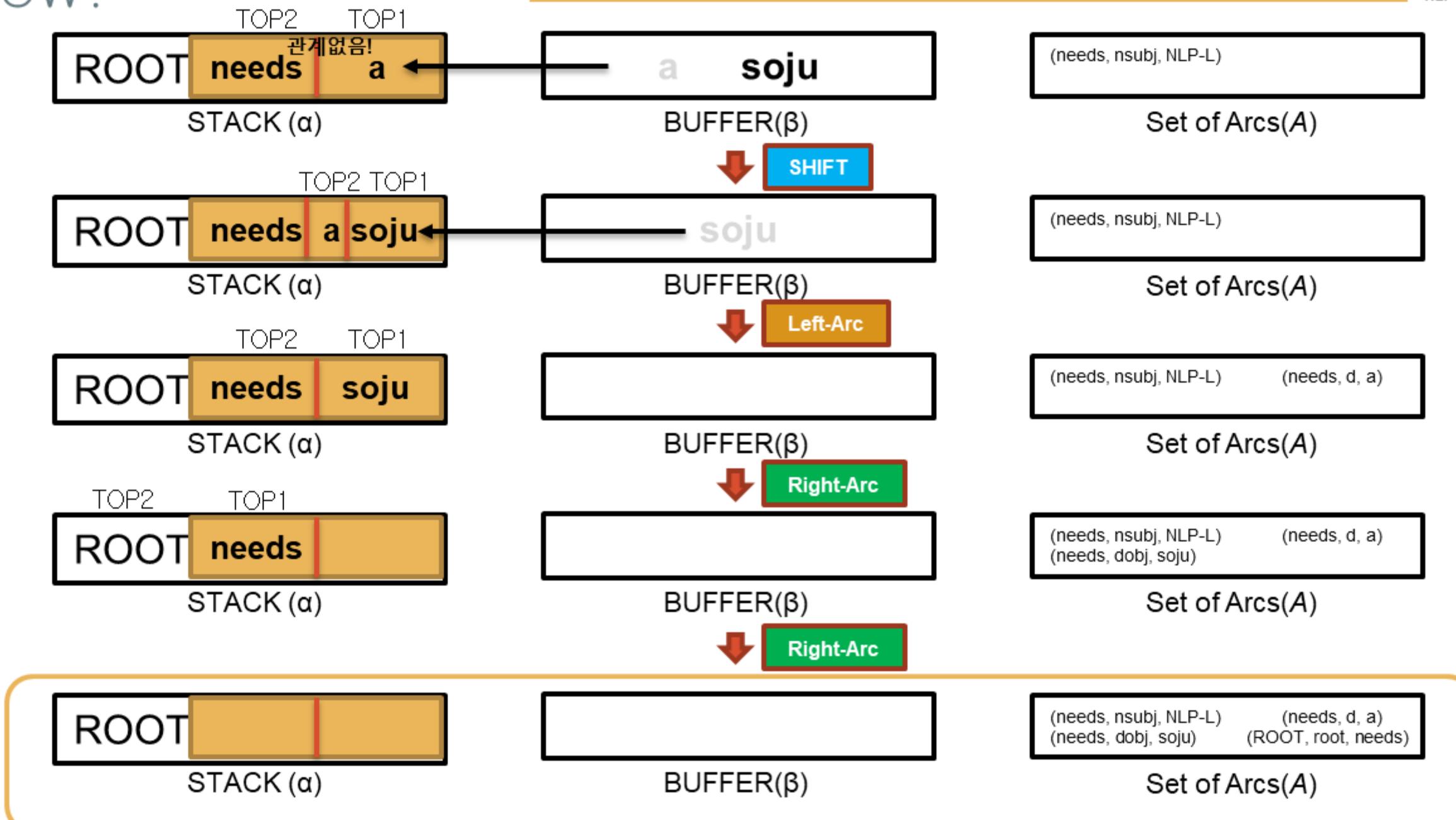
Transition-Based Parsing

Parsing



Transition-Based Parsing

How?



Transition-Based Parsing

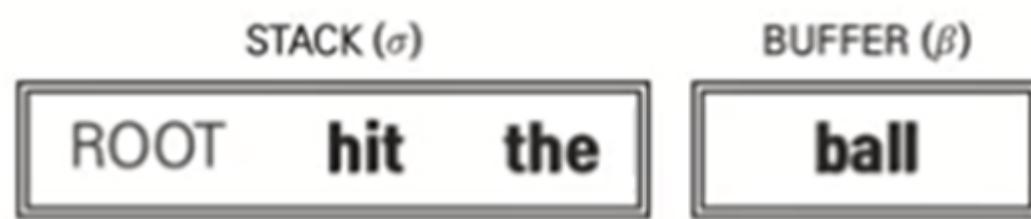
Notations

$s_1.w$	Stack의 첫번째 단어
$b_1.w$	Buffer의 첫번째 단어
$s_1.t$	Stack의 첫번째 단어의 POS Tag
$lc(s_1).w$	Stack의 첫번째 단어의 left-child 단어
$rc(s_1).w$	Stack의 첫번째 단어의 right-child 단어
$lc(s_1).t$	Stack의 첫번째 단어의 left-child 단어의 POS Tag

Example

$s_1.w$	$b_1.t$	$lc(s_2).w$	$rc(s_2).t$
the	NN	John	NULL

State (c)



HOW?

주어진 조건 만족하면 1, else 0

Indicator Features

1	$s_1.w = \text{the}$	$s_1.t = \text{DT}$	$\rightarrow 1$
0	$s_2.w = \text{hit}$	$s_2.t = \text{VBD}$	$b_1.t = \text{NN} \rightarrow 0$
0	$lc(s_2).w = \text{John}$	$lc(s_1).w = \text{hit}$	$lc(s_1).t = \text{NNP} \rightarrow 0$
1	$lc(s_2).w = \text{John}$	$lc(s_1).t = \text{NNP}$...
...			...

- Binary & **Sparse** representation
- 일반적으로 1~3개 요소가 결합된 indicator features
- Parsing 소요시간 중 95% 이상을 feature 연산이 차지함 (**계산비용 높음**)
- 단어 또는 POS Tag의 **의미를 반영하지 못함**

$f(c)$

State (c)

STACK (σ)	BUFFER (β)
ROOT hit the	ball

POS Tags

John	NNP
hit	VBD
the	DT
ball	NN

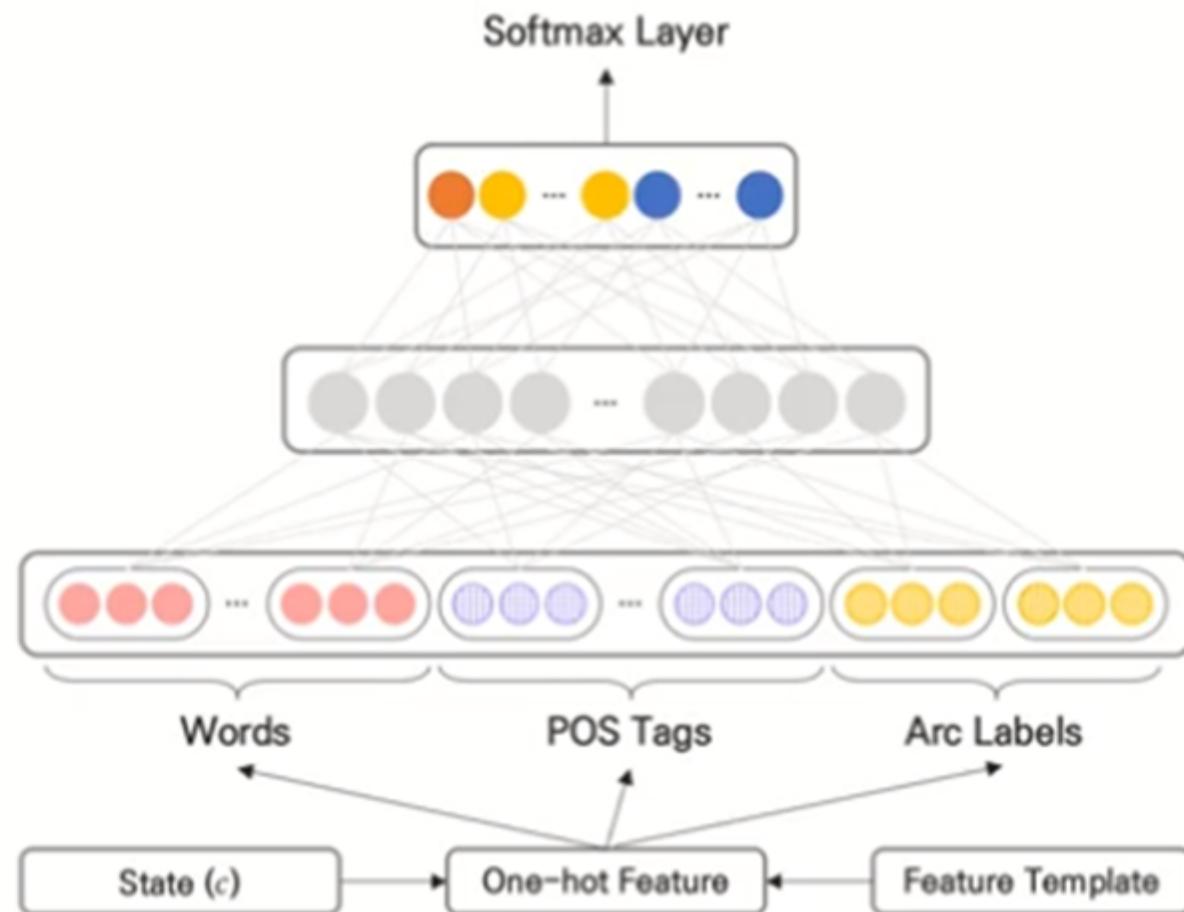
Parsing Tree



HOW?

[Neural Dependency Parser]

Chen and Manning (2014)



HOW?

*word
feature*

Feature Template

- STACK과 BUFFER의 top 3 단어 (6개) $s_1, s_2, s_3, b_1, b_2, b_3$
 $[\text{over}, \text{quick}, \text{ROOT}, \text{the}, \text{lazy}, \text{dog}]$
- STACK top 1, 2 단어의 1st and 2nd left and right child 단어 (8개)
 $lc_1(s_1), rc_1(s_1), lc_2(s_1), rc_2(s_1) \quad lc_1(s_2), rc_1(s_2), lc_2(s_2), rc_2(s_2)$
 $[\text{Null}, \text{Null}, \text{Null}, \text{Null}] \quad [\text{fox}, \text{jumping}, \text{is}, \text{and}]$
- STACK top 1, 2 단어의 (left of left) and (right of right) child 단어 (4개)
 $lc_1(lc_1(s_1)), rc_1(rc_1(s_1)), lc_1(lc_1(s_2)), rc_1(rc_1(s_2))$
 $[\text{Null}, \text{Null}] \quad [\text{the}, \text{Null}]$
- 선택된 word feature에 해당하는 POS Tag (18개)
 $[\text{IN(전치사)}, \text{JJ(형용사)}, \text{ROOT}, \text{DT(한정사)}, \dots, \text{Null}, \text{DT(한정사)}, \text{Null}]$
- STACK과 BUFFER의 6개 단어를 제외하고 선택된 word에 달린 arc-label (12개)
 $[\text{Null}, \text{Null}, \dots, \text{nsubj(주어)}, \text{conj(접속사)}, \text{cop(연결사)}, \text{cc(동위)}, \dots, \text{Null}]$

*pos Tag
feature*

*arc-label
feature*

Example State (c)

Input Sentence

**"The brown fox is quick and
he is jumping over the lazy dog"**

STACK (σ)

ROOT **quick over**

BUFFER (β)

the lazy dog



HOW?

One-hot Representation

Word Features

$$S^w = [\text{over, quick, ROOT, the, \dots, and, Null, Null, the, Null}]$$

	Null	ROOT	and	the	\dots	over	quick
over	0	0	0	0	0	1	0
quick	0	0	0	0	0	0	1
ROOT	0	1	0	0	0	0	0
\dots				\dots			
Null	1	0	0	0	0	0	0
The	0	0	0	1	0	0	0
Null	1	0	0	0	0	0	0

$$S'^w \in \mathbb{R}^{18 \times N_w}$$

POS Tag Features

$$S^t = [\text{IN, JJ, ROOT, DT, JJ, \dots, CC, Null, Null, DT, Null}]$$

	Null	ROOT	DT	NN	\dots	JJ	VBD
IN	0	0	0	0	0	0	0
JJ	0	0	0	0	0	1	0
ROOT	0	1	0	0	0	0	0
\dots					\dots		
Null	1	0	0	0	0	0	0
DT	0	0	1	0	0	0	0
Null	1	0	0	0	0	0	0

$$S'^t \in \mathbb{R}^{18 \times N_t}$$

Arc-label Features

$$S^l = [\text{Null, Null, Null, \dots, nsubj, conj, cop, cc, Null, \dots}]$$

	Null	ROOT	nsubj	cc	\dots	cop	conj
Null	1	0	0	0	0	0	0
Null	1	0	0	0	0	0	0
Null	1	0	0	0	0	0	0
\dots					\dots		
nsubj	0	0	1	0	0	0	0
conj	0	0	0	0	0	0	1
cop	0	0	0	0	0	1	0

$$S'^l \in \mathbb{R}^{12 \times N_l}$$

HOW?

[Neural Dependency Parser]

Chen and Manning (2014)



HOW?

	Null	ROOT	and	the	...	over	quick
over	0	0	0	0	0	1	0
quick	0	0	0	0	0	0	1
ROOT	0	1	0	0	0	0	0
...			...				
Null	1	0	0	0	0	0	0
The	0	0	0	1	0	0	0
Null	1	0	0	0	0	0	0

×

Feature Embedding

Word Embedding Matrix

Null	0.2	0.1	0.7	0.7	1.2	0.1
ROOT	0.3	0.8	2.3	1.2	0.1	1.3
and	0.7	1.0	1.1	0.2	0.6	0.1
the	0.7	0.4	0.3	2.1	0.3	1.0
...			...			
over	0.3	0.2	0.5	1.0	0.2	0.7
quick	1.2	0.8	0.2	2.0	0.3	0.6

Embedded Matrix

$e_{w_1}^w$	0.3	0.2	0.5	1.0	0.2	0.7
$e_{w_2}^w$	1.2	0.8	0.2	2.0	0.3	0.6
$e_{w_3}^w$	0.3	0.8	2.3	1.2	0.1	1.3
...						
$e_{w_{16}}^w$	0.2	0.1	0.7	0.7	1.2	0.1
$e_{w_{17}}^w$	0.7	0.4	0.3	2.1	0.3	1.0
$e_{w_{18}}^w$	0.2	0.1	0.7	0.7	1.2	0.1

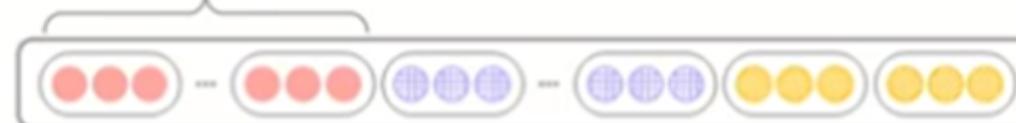
↓ Concat

$$S'^w \in \mathbb{R}^{18 \times N_w}$$

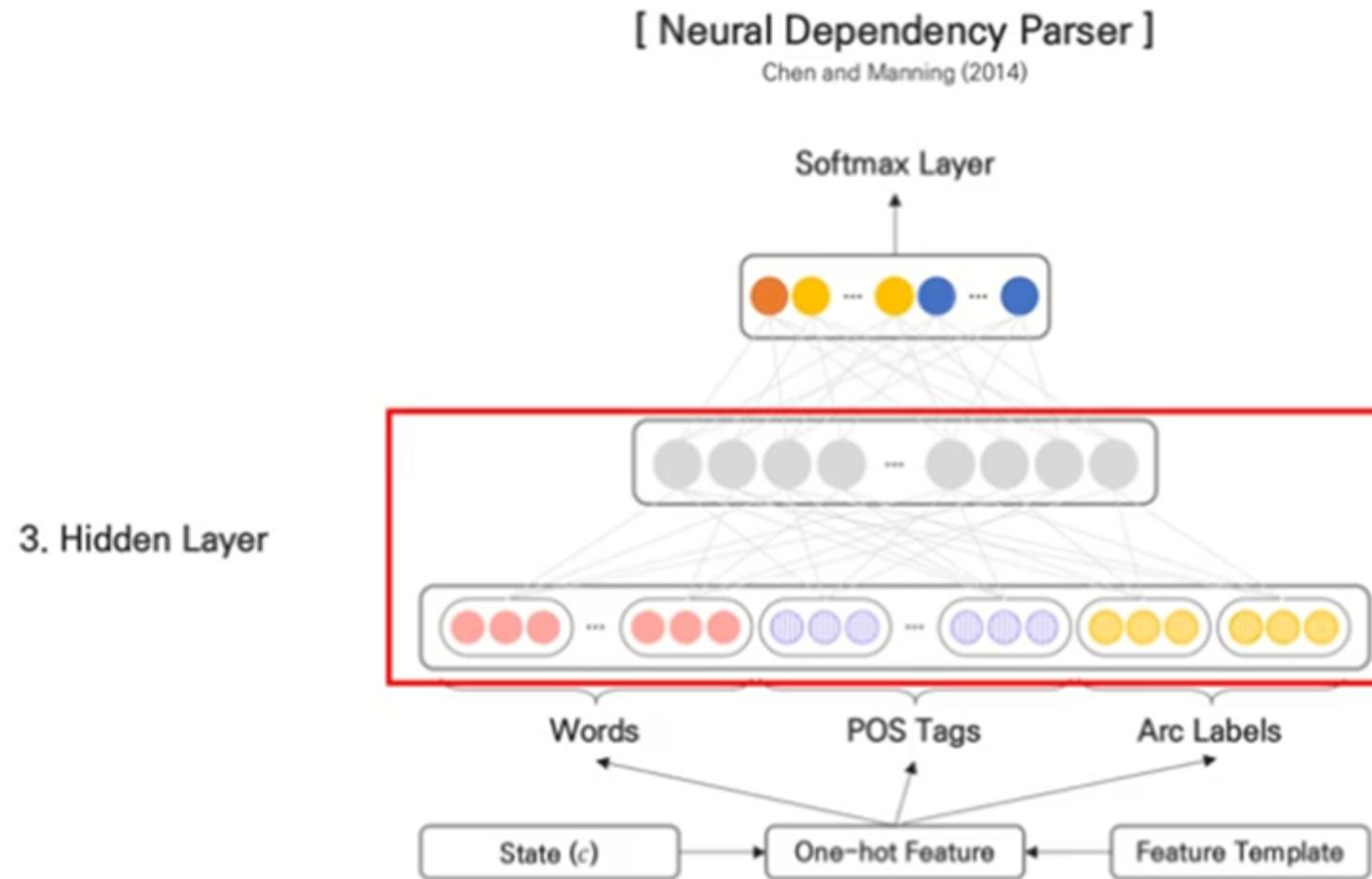
$$E^w \in \mathbb{R}^{d \times N_w}$$

$$x^w = [e_{w_1}^w; e_{w_2}^w; \dots; e_{w_{18}}^w]$$

Words



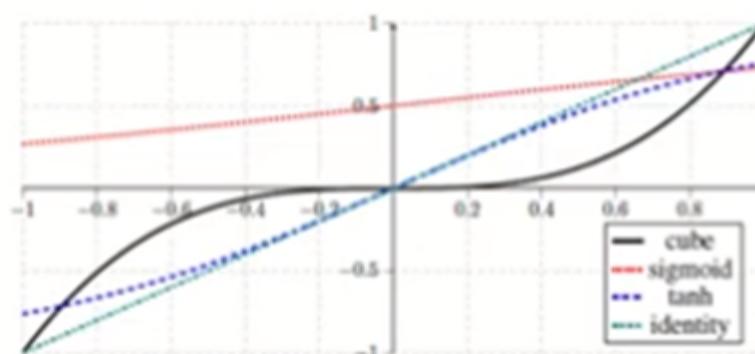
HOW?



HOW?

Hidden Layer

- Embedding vector와 weight matrix를 곱한 뒤 bias vector를 더하는 일반적인 feed forward network
- 하지만 ReLU, Sigmoid, Tanh와 같은 일반적인 activation function을 사용하지 않음
- 대신 word, POS tag, arc-label간 상호작용을 반영할 수 있는 cube function을 사용함
- 엄밀한 수학적 증명을 하지는 않았으나 실험 결과 타 non-linearity 대비 우수한 성능을 기록함



$$\begin{aligned} h &= (W_1[x^w; x^t; x^l] + b)^3 \\ &= (w_1x_1 + w_1x_2 + \dots + w_{48}x_{48} + b)^3 \\ &= \sum_{i,j,k} (w_i w_j w_k) x_i x_j x_k + \sum_{i,j} b(w_i w_j) x_i x_j + \dots \end{aligned}$$

각 word, POS tag, arc-label의 조합

HOW?

[Neural Dependency Parser]

Chen and Manning (2014)

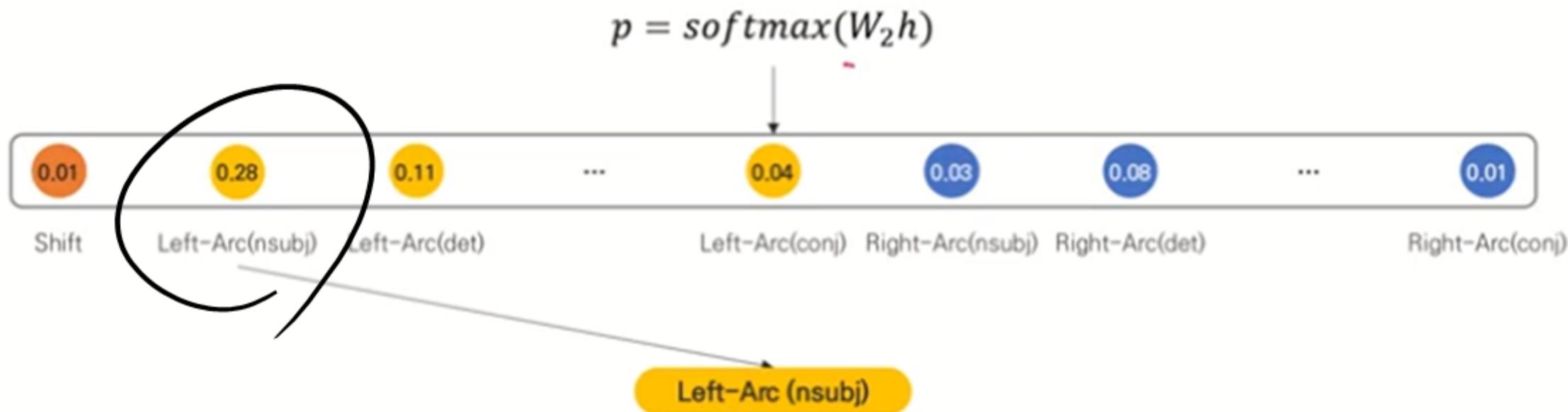
4. Softmax Layer



HOW?

Softmax Layer

- Hidden layer를 거친 feature vector를 linear projection 후 softmax function 적용
- Shift, Left-Arc, Right-Arc 중 가장 확률값이 높은 경우의 수를 output으로 산출



[Summary]

Parsing

각 문장의 문법적인 구성 또는 구문을 분석하는 과정
(구문분석 트리를 구성하는 것)

Constituency Parsing

문장의 구조를 파악하는 것이 주목적

Dependency Parsing

단어간 관계를 파악하는 것이 주목적

Transition-based

두 단어의 의존여부를 순서대로 결정

Graph-based

가능한 의존 관계를 모두 고려

Conventional

Sparse Feature 및 ML Model 사용

Neural Network

Dense Feature 및 NN Model 사용

Language Model & RNN

STUDY OBJECTIVES

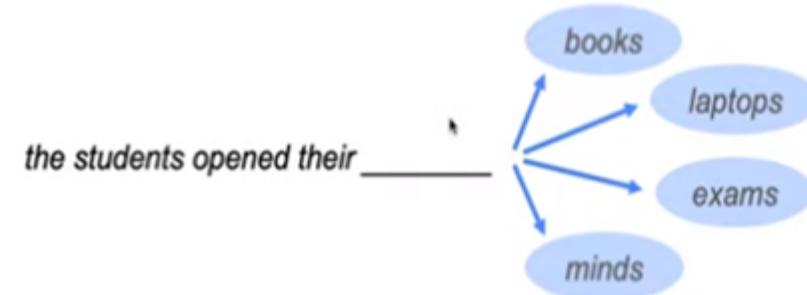
1. Language Model

3. RNN

2. N-gram Language Model

Language Model

단어의 시퀀스(=문장)에 대해 확률을 할당하는 모델.
= 가장 자연스러운 단어 시퀀스를 찾아내는 모델



Language Modeling :

주어진 단어의 시퀀스(=문장)에 대해, 다음에 나타날 단어가 어떤 것인지를 예측하는 작업

$$P(w_t | w_{t-1}, \dots, w_1)$$

where w_t can be any word in the vocabulary $V = \{w_1, \dots, w_{|V|}\}$

- 앞 단어를 가지고 뒤에 나타날 단어를 예측하는 것을 말함
- Joint probability를 활용해 각 단어마다의 확률을 나타나고, 가장 높은 확률을 선택하는 것이 일반적인 Language Model.

Language Model

하나의 단어를 w , 단어 시퀀스를 W 로 한다면,
 n 개의 단어가 등장하는 W 의 확률은 다음과 같다.

1. 기계번역

$P(\text{나는 버스를 탔다}) > P(\text{나는 버스를 태운다})$

$$P(W) = P(w_1, w_2, w_3, w_4, w_5, \dots, w_n)$$



2. 오타 교정

“나는 너에게” + $[P(\text{달려갔다}) > P(\text{잘려갔다})]$

다음 단어가 등장할 확률 :
 $n-1$ 개의 단어가 나열된 상태에서, n 번째 단어의 확률

3. 음성 인식

$P(\text{나는 메롱을 먹는다}) < P(\text{나는 메론을 먹는다})$

-> Language model은 확률을 통해 보다 적절한 문장을 판단함

$$P(w_n | w_1, \dots, w_{n-1})$$

조건부 확률



즉 전체 단어 시퀀스 W 의 확률 :

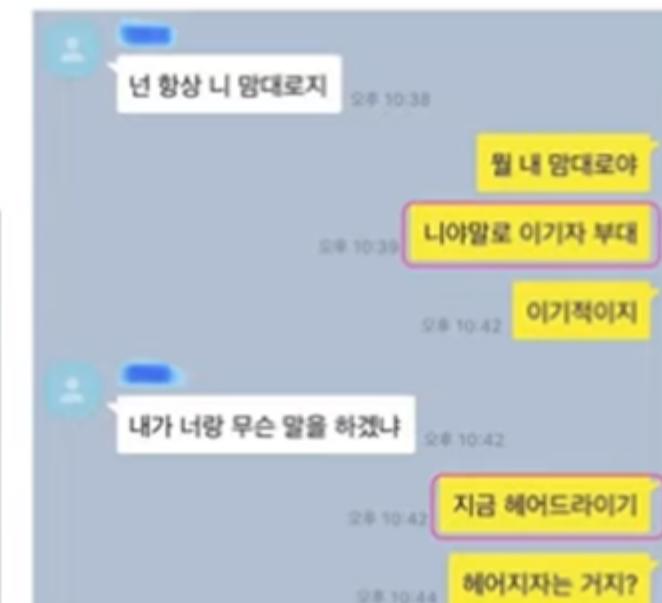
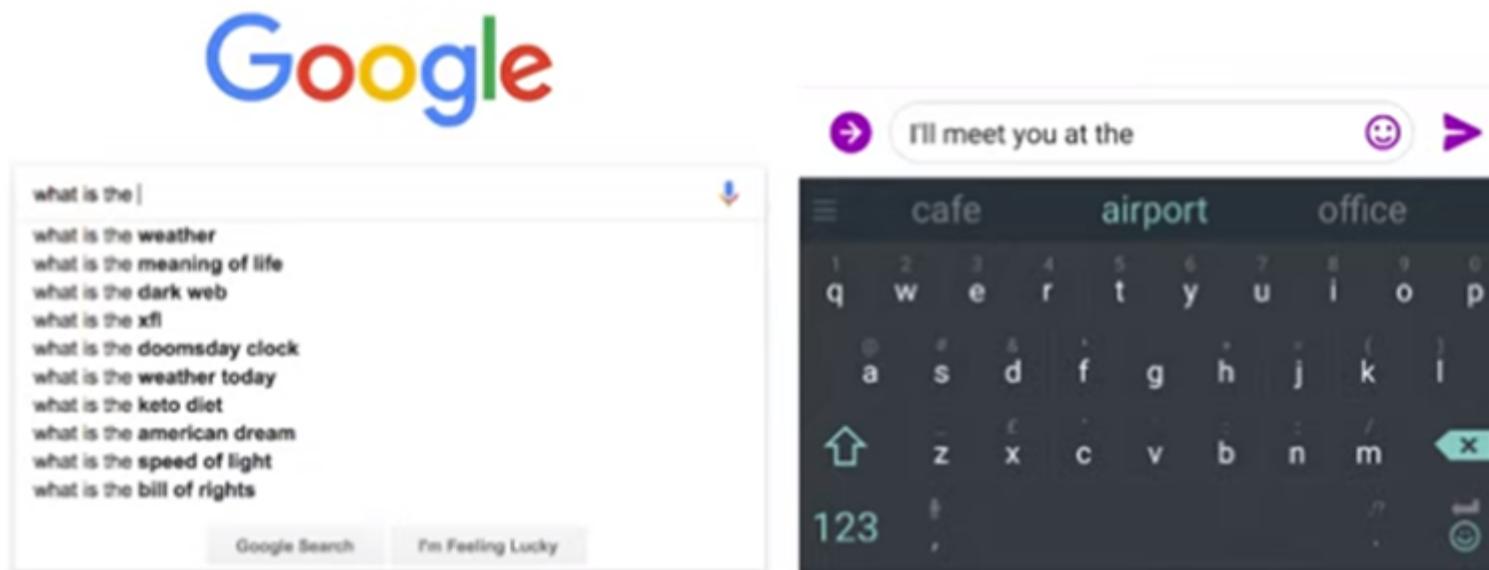
$$P(W) = P(w_1, w_2, w_3, w_4, w_5, \dots, w_n) = \prod_{i=1}^n P(w_i | w_1, \dots, w_{i-1})$$

Language Model

어디에 사용될까 :

- Language model은 문장의 확률 또는 단어의 등장 확률을 예측
- 기계번역, 음성인식, 자동완성 등에 활용

```
1 import os
2 import sys
3
4 # Count lines of code in the given directory, separated by file extension
5 def main(directory):
6     line_count = {}
7
8
9
10
11
12
13
14
15
16
17
18
19
```



n-gram Language Model

n-gram language model :

- Neural network 이전에 사용되었던 language model
- 예측에 사용할 앞 단어들의 개수를 정하여 모델링하는 방법
- n-gram : n개의 연이은 단어의 뭉치

Definition: **n-gram** 은 **n**개의 **연이은** 단어의 뭉치

$w_1, w_2, \dots, w_{t-n+1}, \dots, w_{t-1} \underbrace{w_t}_{n}, \dots, w_{T-1}, w_T$
window of $n - 1$ previous words

the students opened their



- uni-grams: "the", "students", "opened", "their"
- bi-grams: "the students", "students opened", "opened their"
- tri-grams: "the students opened", "students opened their"
- 4-grams: "the students opened their"

$$P(w_1, \dots, w_T) = P(w_1) \times P(w_2 | w_1) \times \dots \times P(w_T | w_{T-1}, \dots, w_1)$$

$$= \prod_{t=1}^T P(w_t | w_{t-1}, \dots, w_1)$$

$$\approx \prod_{t=1}^T P(w_t | w_{t-1}, \dots, w_{t-n+1})$$

n-gram Language

$w_1, w_2, \dots, w_{t-n+1}, \dots, w_{t-1}, w_t, \dots, w_{T-1}, w_T$

$$P(w_t | w_{t-1}, \dots, w_1) \approx P(w_t | w_{t-1}, \dots, w_{t-n+1}) \quad (\text{assumption})$$

$$\begin{aligned} \text{prob of a } n\text{-gram} &\rightarrow P(w_t, w_{t-1}, \dots, w_{t-n+1}) \\ \text{prob of a } (n-1)\text{-gram} &\rightarrow \frac{P(w_t, w_{t-1}, \dots, w_{t-n+1})}{P(w_{t-1}, \dots, w_{t-n+1})} \end{aligned}$$

(definition of conditional prob)

$$\approx \frac{\text{count}(w_t, w_{t-1}, \dots, w_{t-n+1})}{\text{count}(w_{t-1}, \dots, w_{t-n+1})} \quad (\text{statistical approximation})$$

Ex) as the proctor started the clock, the students opened their

(4-1)-gram 만 사용

Example: 4-grams

$$P(\omega | \text{studetns opened their}) = \frac{\text{count(students opened their } \omega)}{\text{count(students opened their)}}$$

문자열	횟수
students opened their	1,000
students opened their books	400
students opened their exams	100

$$P(\text{books} | \text{studetns opened their}) = \frac{400}{1000} = 0.4$$

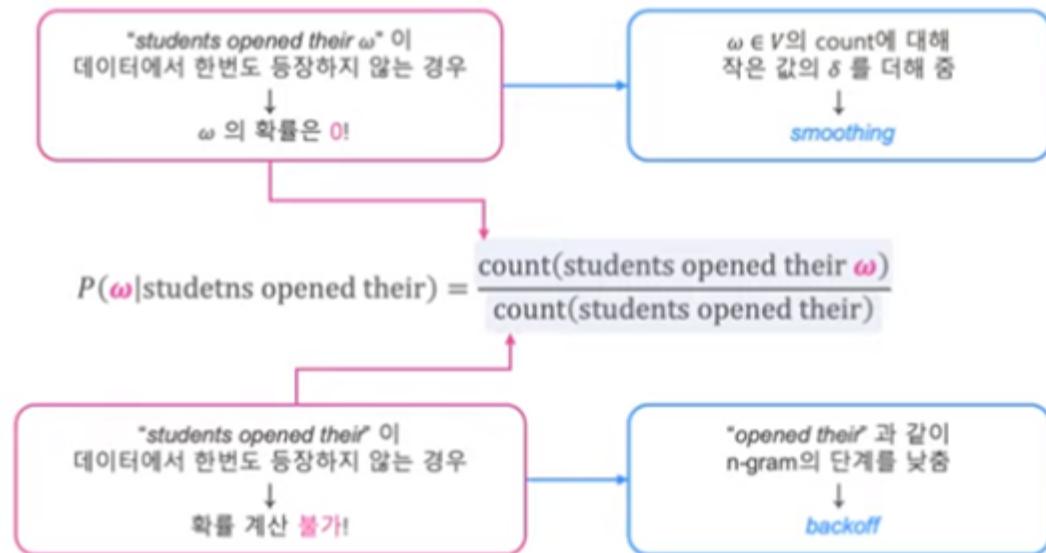
$$P(\text{exams} | \text{studetns opened their}) = \frac{100}{1000} = 0.1$$

n-gram Language Model

문제점_

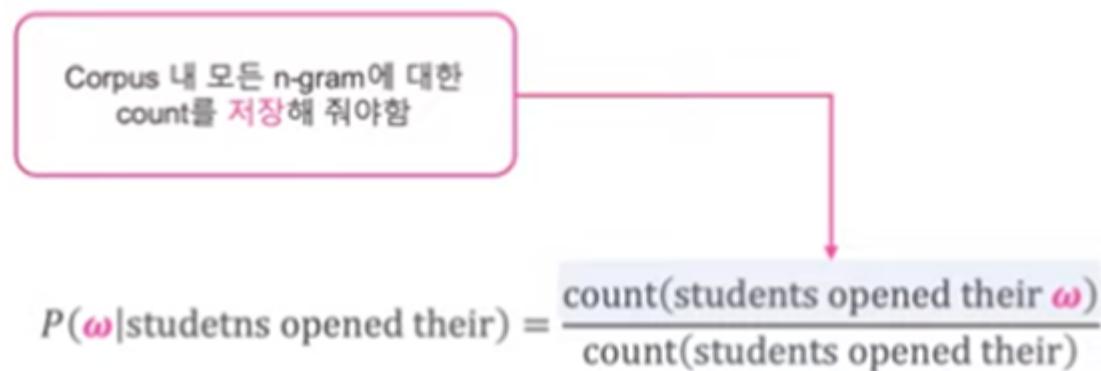
• Sparsity problem :

- 노이 커질수록 해당 n-gram을 카운트할 수 있는 확률은 적어짐
- 일반적으로 $n < 5$ 로 설정한다



• Storage problem :

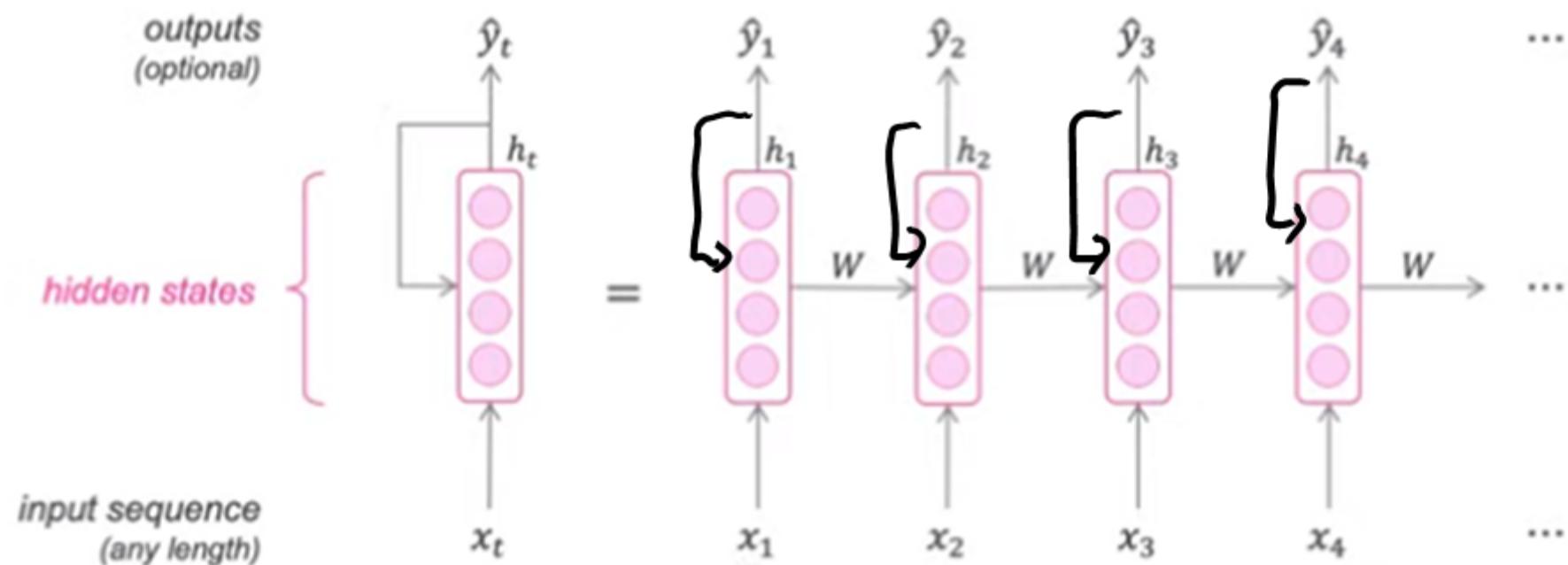
- 노이 커지거나 corpus가 증가하면 모델의 크기가 증가함



RNN

단어값이 hidden layer를 통과하고, 결과가 출력값이 되면서 그 동시에 다음 상태의 입력값으로 들어가는 원리

단어가 문장에서의 순서별로, 즉 *timestep t*를 따라 입력되고, 동일한 가중치 W 를 반복적으로 적용시킨다는 특징이 있음



RNN

✓ RNN Language Model

output distribution

$$\hat{y}_t = \text{softmax}(Uh_t + b_2) \in \mathbb{R}^{|V|}$$

hidden states

$$h_t = \sigma(W_h h_{(t-1)} + W_e e_t + b_1)$$

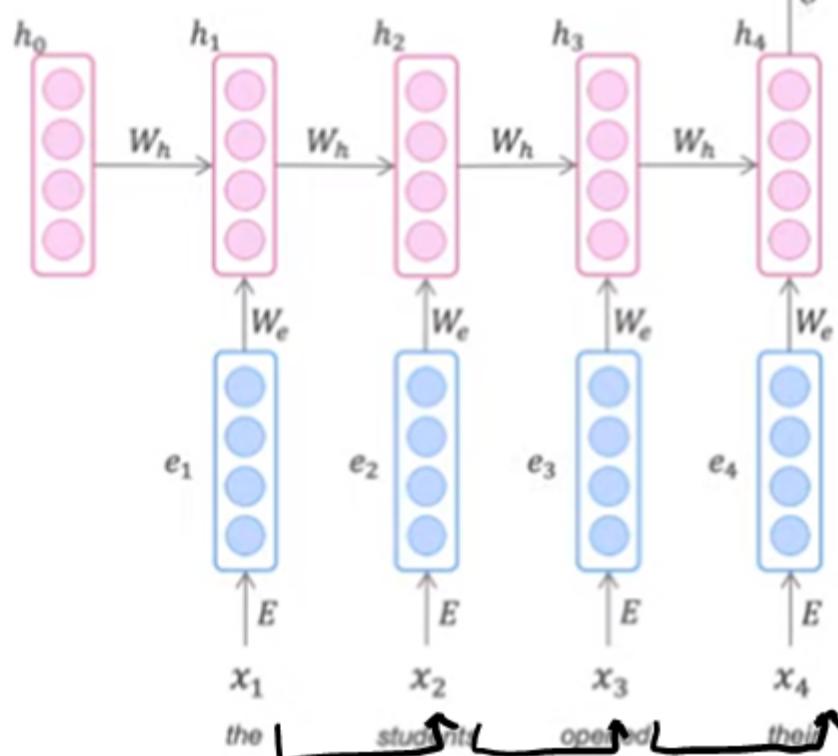
h_0 is the initial hidden state

word embeddings

$$e_t = Ex_t$$

words \rightarrow one-hot vectors

$$x_1, x_2, x_3, x_4$$



장점 :

- > 입력 길이에 제한이 없음(단어를 순서대로 하나씩 넣기 때문에)
- > 이론적으로는 길이가 긴 데이터를 처리 가능
- > (반복적으로 동일한 W 가 사용되기 때문에) 입력에 따른 모델의 크기가 증가하지 않음
- > 매 timestamp t 에 동일한 가중치를 적용하므로 symmetry 함

단점 :

- > (단어가 하나씩 순서대로 계산되기 때문에) 속도가 느림
- > 실제로는 길이가 긴 timestamp t 에 대해 데이터가 소실되는 문제도 발생하기도 함

RNN

✓ Training a RNN Language Model

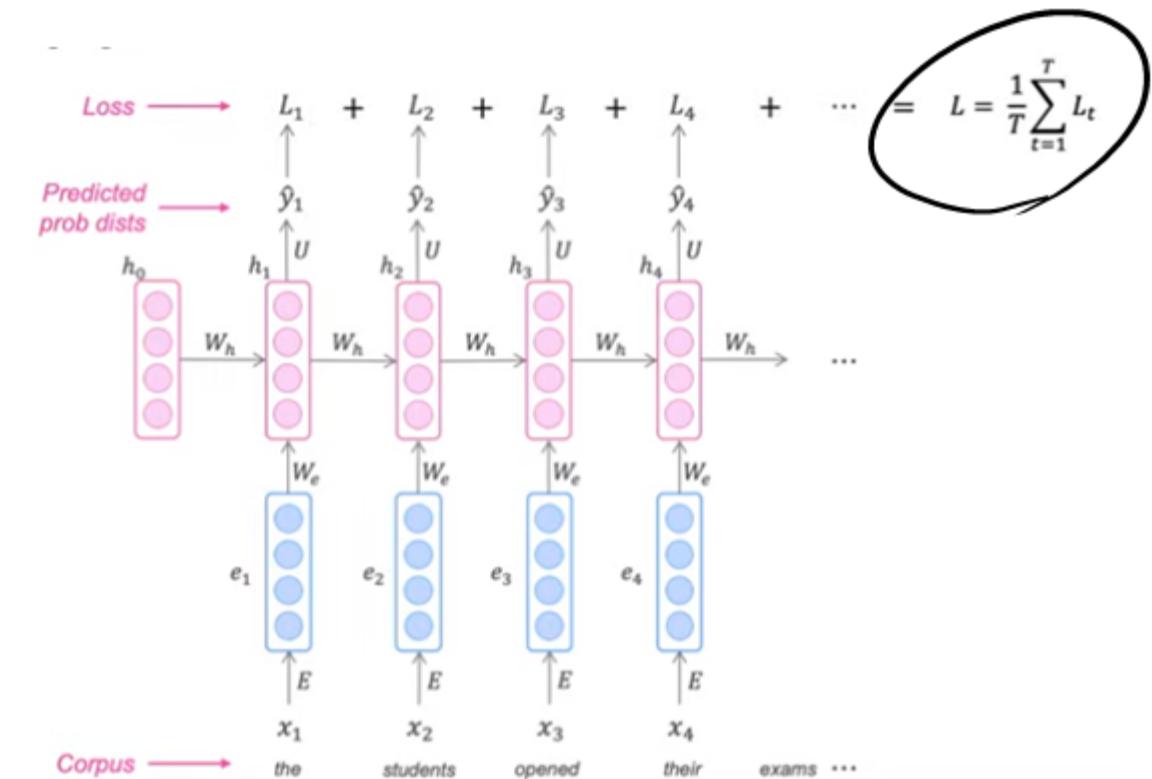
- ① x_1, \dots, x_T 의 단어들로 이루어진 시퀀스의 Corpus를 준비한다.
- ② x_1, \dots, x_T 를 차례대로 RNN-LM에 주입하고, 매 step t 에 대한 \hat{y}_t 를 계산한다.
 - 주어진 단어에서부터 시작하여 그 다음 모든 단어들에 대한 확률을 예측

- ③ Step t 에 대한 손실함수 Cross-Entropy를 계산한다. (y_t is one-hot for x_{t+1})

$$L_t = CE(y_t, \hat{y}_t) = - \sum_{w \in |V|} y_{t,w} \times \log(\hat{y}_{t,w}) = -\log(\hat{y}_{t,x_{t+1}})$$

- ④ 전제 step T 에 대해 계산한 손실함수 L_t 의 평균을 계산한다.

$$L = \frac{1}{T} \sum_{t=1}^T L_t = -\frac{1}{T} \sum_{t=1}^T \sum_{w=1}^{|V|} y_{t,w} \times \log(\hat{y}_{t,w}) = -\frac{1}{T} \sum_{t=1}^T -\log(\hat{y}_{t,x_{t+1}})$$

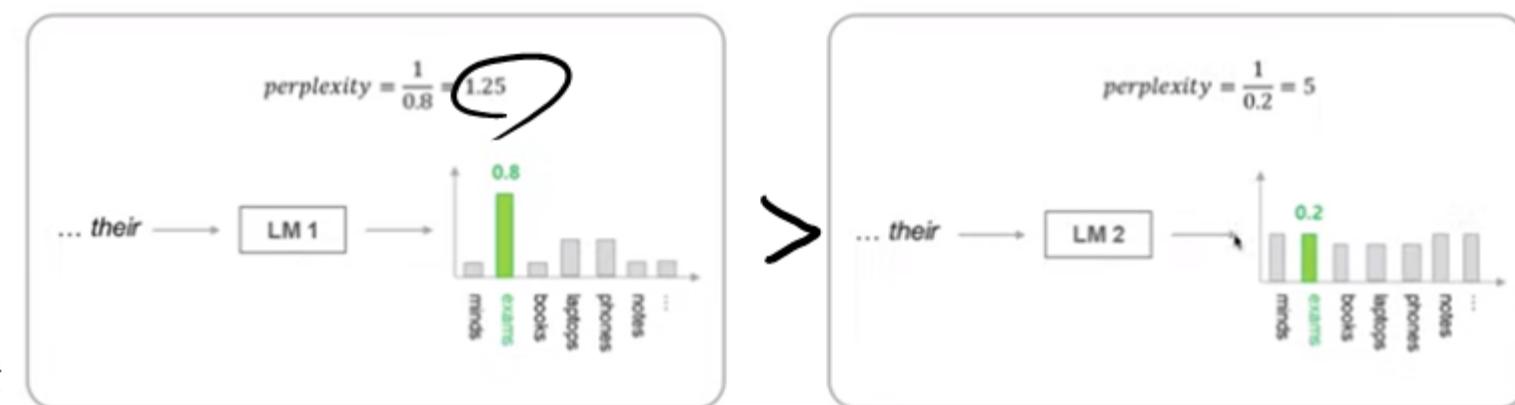


$$L = \frac{1}{T} \sum_{t=1}^T L_t$$

RNN

Evaluation

- Language model : 주어진 과거 단어로부터 다음에 출현할 단어의 확률분포를 출력하는 모델



- L.m.을 평가하는 대표적인 척도인 **Perplexity**를 사용

$$\text{perplexity} = \prod_{t=1}^T \left(\frac{1}{P_{\text{LM}}(x_{t+1} | x_t, \dots, x_1)} \right)^{1/T} = \text{normalization}$$

- Perplexity : 출현할 단어의 확률에 대한 역수

$$= \prod_{t=1}^T \left(\frac{1}{\hat{y}_{t,x_{t+1}}} \right)^{1/T} = \exp \left(\frac{1}{T} \sum_{t=1}^T -\log(\hat{y}_{t,x_{t+1}}) \right) = e^L$$

> 손실함수 L
대입

- P가 작을수록 좋은 language model

DONE!!
