Syntactic Structure and Dependency Parsing

권재선 주연우

EWHA EURON

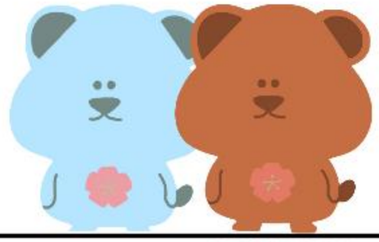
목차

#01 Syntactic Structure : Constituency and Dependency

#02 Dependency Grammar & Structure

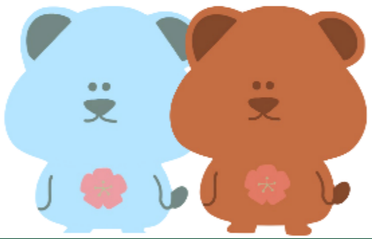
#03 Dependency Parsing 방법

- Transition-based dependency parsing
- Neural Dependency parsing



EWHA EURON

Syntactic Structure : Constituency and Dependency



EWHA EURON

#01 Syntactic Structure : Constituency and Dependency

#1 Syntactic Structure

Parsing | 구문 분석 | 문장을 이루고 있는 구성 성분으로 분해하고 그 사이의 위계 관계를 분석하여 문장의 구조를 분석하는 것

Parsing
- Constituency Parsing
  : 문장을 이루고 있는 구성요소를 파악
- Dependency Parsing
  : 문장을 이루고 있는 단어간 의존 관계를 파악

#01 Syntactic Structure : Constituency and Dependency

Tokenizing

텍스트 데이터에 대한 정보를 단위별로 나누는 것

=주어진 코퍼스(corpus)에서 토큰(token) 단위로 나누는 작업

corpus (말뭉치)
: 구조를 이루고 있는 텍스트 집합

token
: 문법적으로 더 이상 나눌 수 없는 언어요소

토큰 단위는 사용자의 목적에 맞게 설정된다
ex) 문장, 단어, 문장부호, 컴마(,)

#01 Syntactic Structure : Constituency and Dependency

Pos tagging

POS = Parts-of-speech = 품사

문장을 구성하는 단어들의 품사를 식별하여 태그를 붙이는 것

1. 품사에 따른 발음
2. 의미의 중의성 해소
3. 정보추출

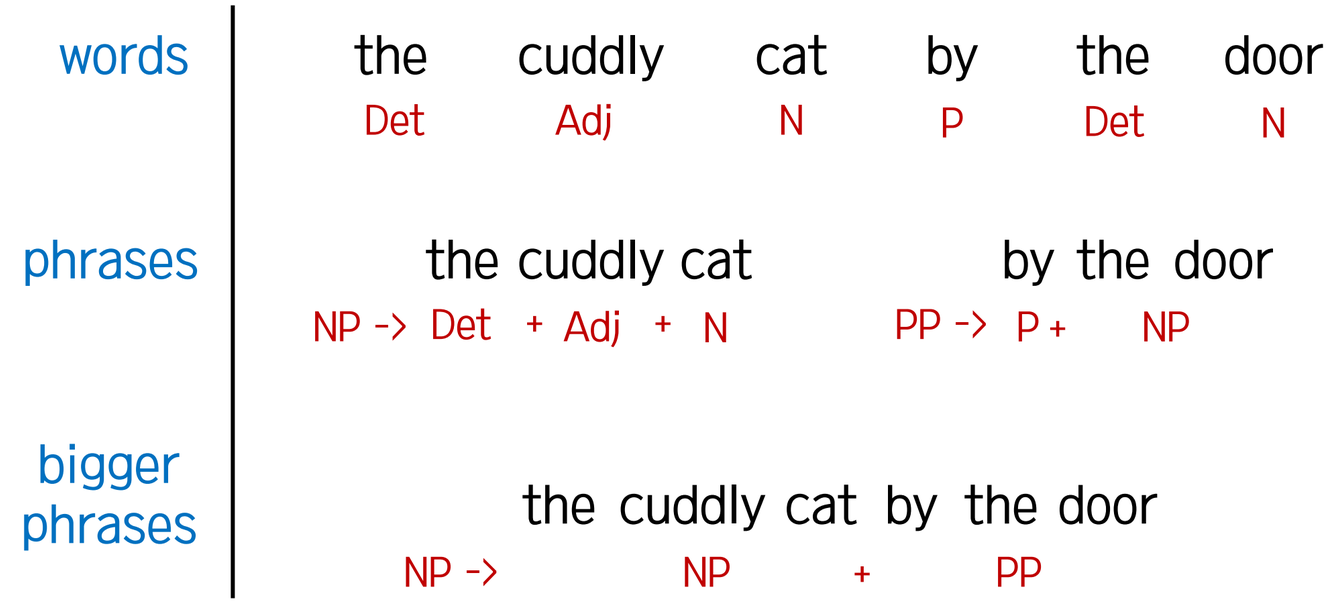
1) Noun
2) Verb
3) Pronoun
4) Preposition
5) Adverb
6) Conjunction
7) Particle
8) Article

#01 Syntactic Structure : Constituency and Dependency

#2 Constituency Parsing

=Phrase-structure grammar, context-free grammar

문장을 구성하고 있는 phrase(구)를 파악하여 문장 구조 분석

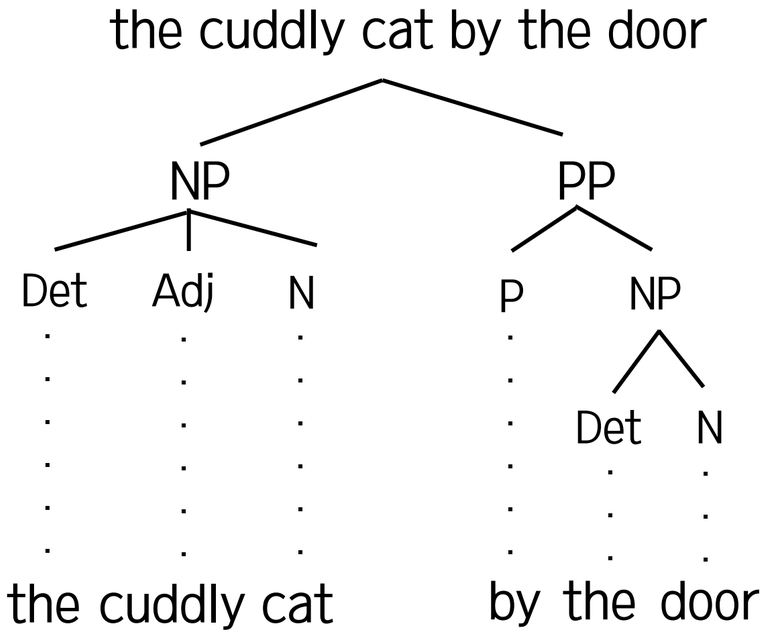


#01 Syntactic Structure : Constituency and Dependency

#2 Constituency Parsing

=Phrase-structure grammar, context-free grammar

문장을 구성하고 있는 phrase(구)를 파악하여 문장 구조 분석



#01 Syntactic Structure : Constituency and Dependency

#3 Dependency Parsing

문장을 구성하는 단어간 의존, 수식 관계를 파악하여 문장 구조 분석

#01 Syntactic Structure : Constituency and Dependency

#4 구문 분석이 필요한 이유

1. 복잡한 의미 전달을 위해 문장을 더 큰 단위로 구성하게 되고,
이를 정확히 이해하기 위해서는 연결 구조에 대한 이해가 필요하다

2. 언어를 정확히 해석하기 위해 문장 구조에 대한 파악이 중요하다

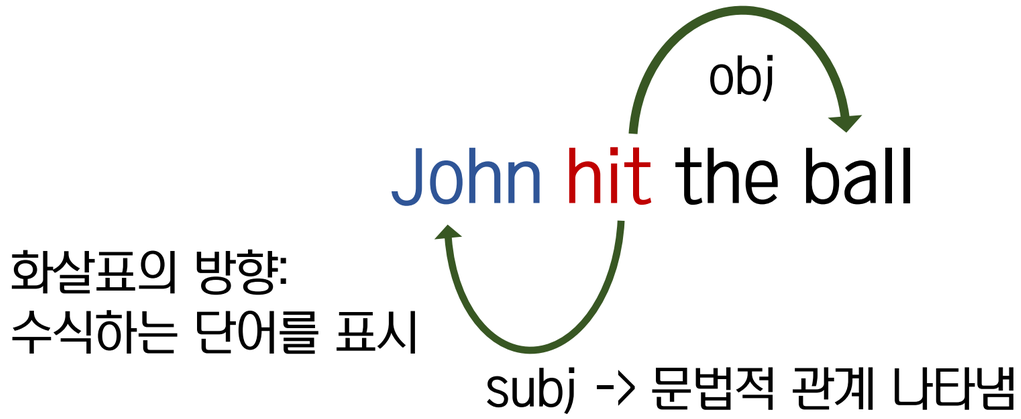
#01 Syntactic Structure : Constituency and Dependency

#3 Dependency Parsing

문장을 구성하는 단어간 의존, 수식 관계를 파악하여 문장 구조 분석

수식 받는 단어: head = governor

수식 하는 단어: dependent = modifier



#01 Syntactic Structure : Constituency and Dependency

#3 Dependency Parsing

문장을 구성하는 단어간 의존, 수식 관계를 파악하여 문장 구조 분석

수식 하는 단어: head = governor

수식 받는 단어: dependent = modifier

obj = object

subj = subject

nmod = nominal modify

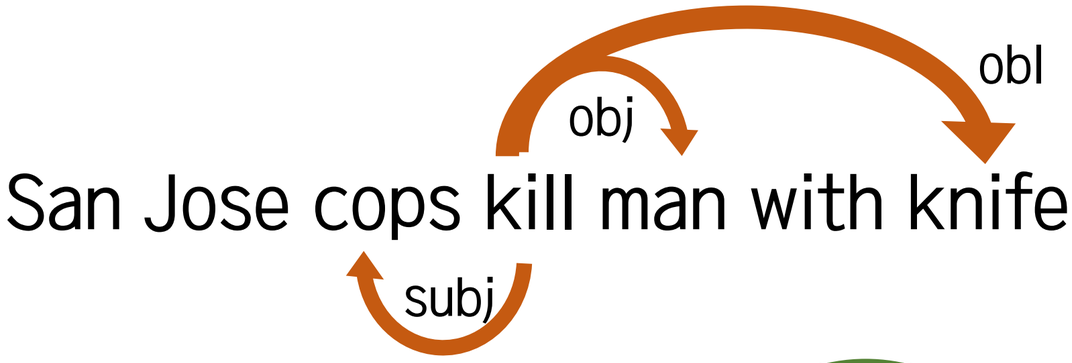
obl = oblique

#01 Syntactic Structure : Constituency and Dependency

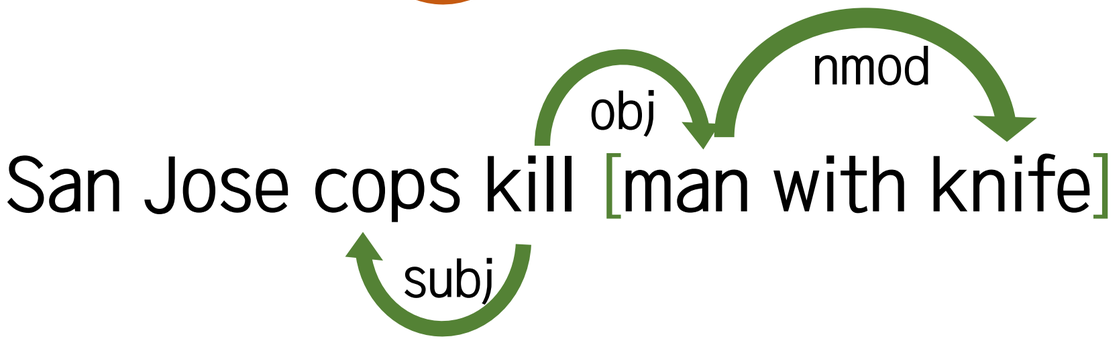
#5 Ambiguity

1) Phrase attachment ambiguity

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



San jose cops는 남자를 칼로 죽였다



San jose cops는 칼을 지닌 남자를 죽였다

#01 Syntactic Structure : Constituency and Dependency

#5 Ambiguity

2) Coordination Scope ambiguity

단어가 수식하는 대상의 범위가 달라짐에 따라 의미가 변하는 모호성

2 people [Shuttle veteran] and [longtime NASA executive Fred Gregory] appointed to board

1 people [Shuttle veteran and longtime NASA executive] Fred Gregory appointed to board

nmod

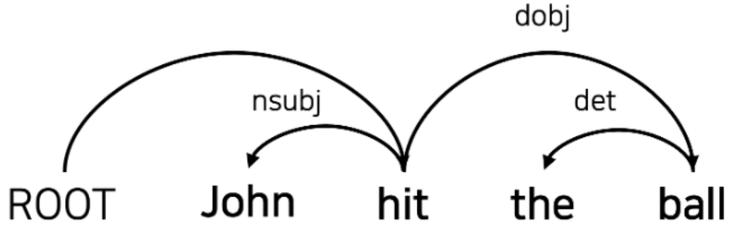
#02. Dependency Grammar & Structure

#1 Dependency parsing의 문법과 구조

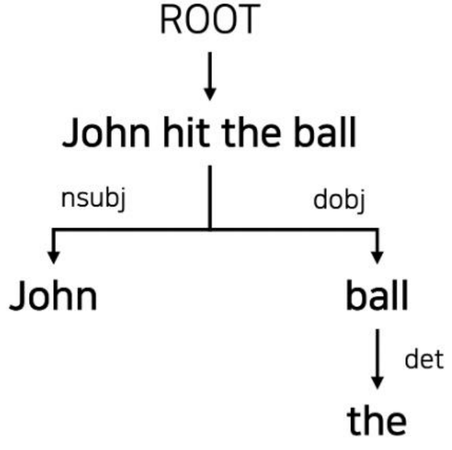
- dependency parsing의 형태 : sequence 형태 & tree 형태로 표현가능,

•가상의 노드 ROOT를 문장의 맨 앞에 추가 -> 모든 성분의 최종 head를 ROOT로 설정
→ ROOT는 모든 단어가 최소 1개 노드의 dependent가 되어 의존 관계를 가지도록 함

•화살표는 수식을 받는 단어 head에서 수식을 하는 dependent로 향함, 화살표 위 라벨 : 단어간 문법적 관계 의미,
(nsubj: 주어, dobj: 직접목적어, det: 한정사)



sequence



tree

#02. Dependency Grammar & Structure

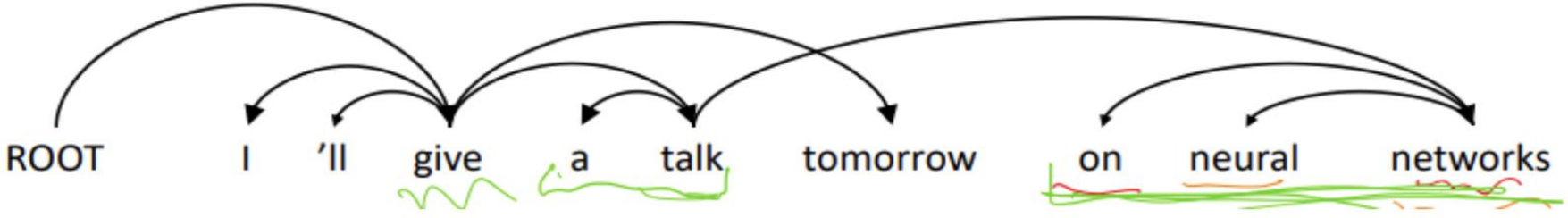
#2 Parsing 규칙 (제약)

*** dependency parsing: 어떤 단어가 어떤 단어에 의존하는지를 정의하며 진행됨***

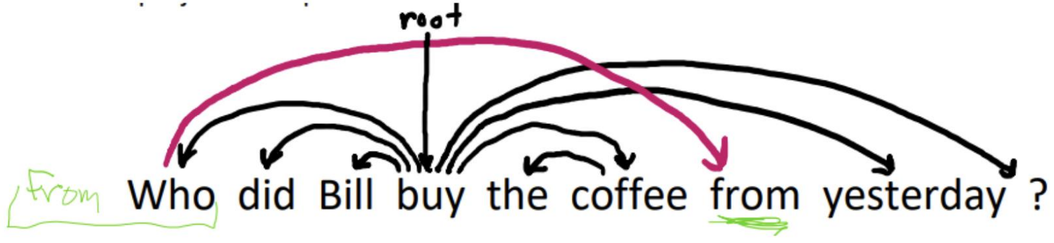
•하나의 단어만이 ROOT의 dependent임

•화살표는 순환하지 않음 (A->B, B->A), 즉 universal한 데이터 구축을 위해 projectivity하게 parsing한다
(projectivity: 화살표가 겹치지 않음)

Projectivity



Give-> tomorrow 와 talk-> networks의 화살표 교차 : non- projective
따라서, on neural networks를 tomorrow 앞으로 당겨옴으로써 projective하게 만들 수 있음



#02. Dependency Grammar & Structure

#3 Dependency parsing에서 고려되는 보편적 특징
(사용되는 정보는 어디서 오는가?)

- Bilexical affinities: 두 단어 사이의 실제 의미가 드러나는 관계
ex) discussion – issues

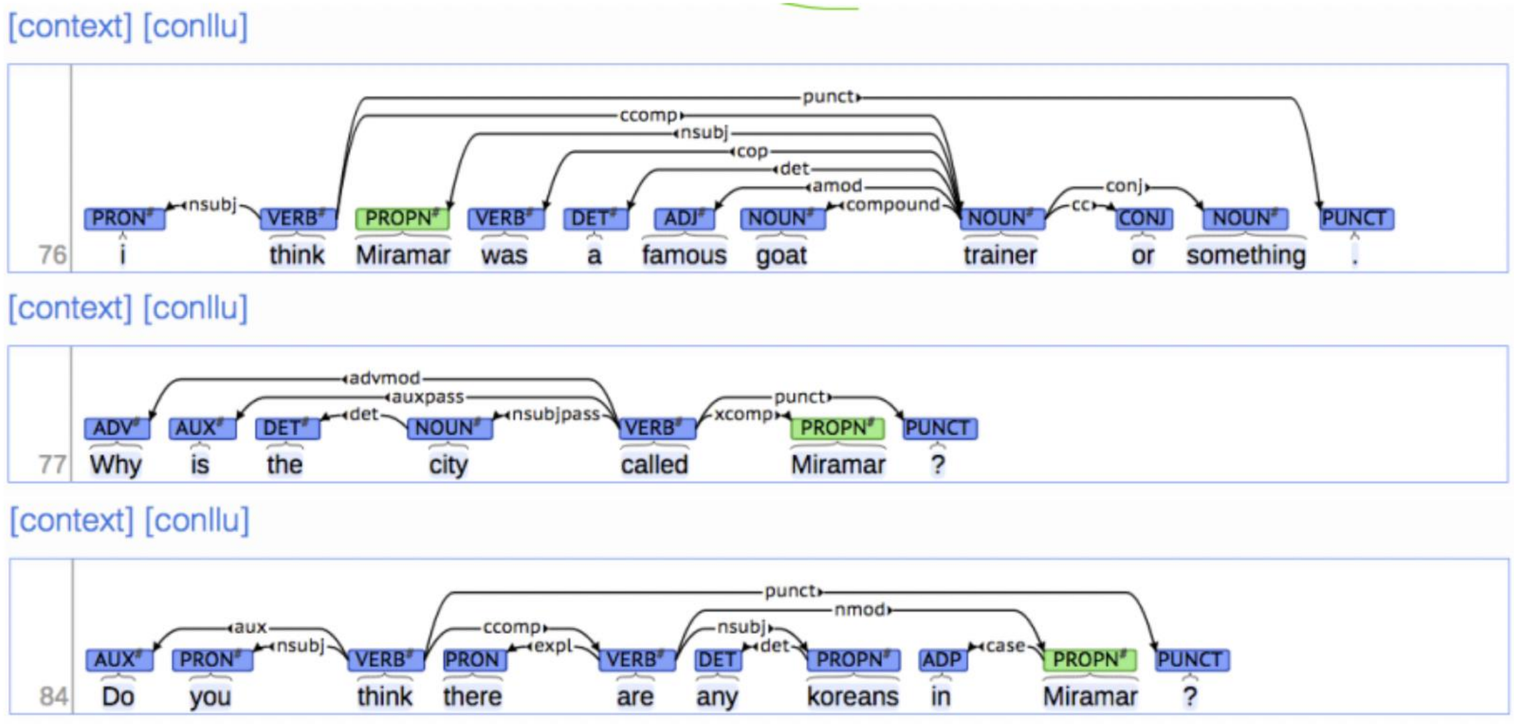
- Dependency distance: dependency의 거리는 주로 가까운 위치에서 dependent 관계 형성

- Intervening material: 동사나 구두점을 사이에 두고 dependency 관계가 잘 형성되지는 않음

- Valency of heads: head가 문장 내에서 결합하는 문장 구성 성분의 수 고려
ex) was completed는 다른 문장 성분을 꼭 필요로 함

#02. Dependency Grammar & Structure

#4 The rise of annotated data & Universal Dependencies tree bank



Treebank 장점:

1.가장 좋은 점: treebank는 재사용이 가능
2.언어의 핵심을 관통하기 때문에 커버할 수 있는 범위가 넓음
3.문장 구조를 분석하면 통계 정보를 얻을 수 있고 이를 활용하기 좋음
4. NLP 평가에 사용될 수 있음 (영어에서 단어들은 다양한 parser를 가지기 때문에 8,90 년대에 많이 사용됨)

#03. Dependency Parsing 방법

#1 Methods of Dependency parsing

1.Dynamic programming(1996): 동적 계획법을 사용하여 긴 문장을 여러 개로 나누어 하위 문자열에 대한 트리를 만들고 최종적으로 다시 합치는 방식으로 parsing 진행

2.Graph algorithms(2005): 가능한 의존 관계를 모두 고려한 뒤 가장 확률이 높은 구문분석 트리 선택

3.Constraint Satisfaction: 문법적 제한 조건을 초기에 설정하고, 조건을 만족한 경우만 남기고 나머지는 제거하여 조건을 만족하는 단어만 parsing

4.Transition-based parsing(2008): 두 단어의 의존 여부를 순서대로 결정하여 점진적으로 구문분석 트리 구성

#03. Dependency Parsing 방법

#2 Transition-based Parsing Process

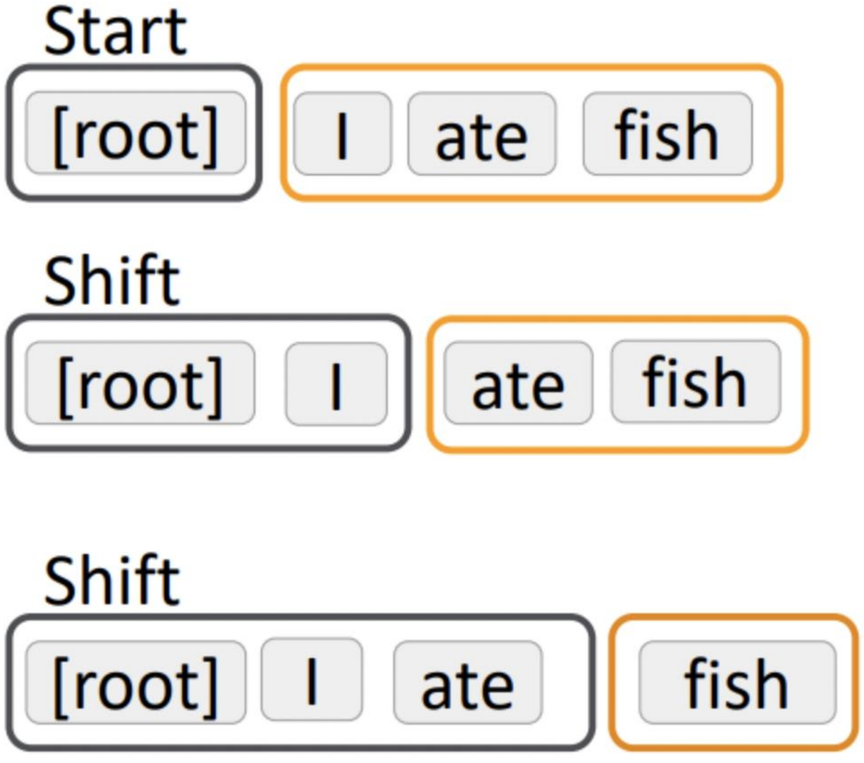
1.Greedy transition-based parsing

•Parser는 bottom-up 순서로 구축해 나아감. (leaves -> root)

•The parser has:
•Stack (starts with ROOT)
•Buffer (starts with input sentence)
•Set of dependency arcs (starts off empty)
•Set of actions (행동의 집합)

•가능한 행동 3가지:
1.buffer에서 stack으로 단어를 가져오는 Shift
2.오른쪽 단어에서 왼쪽 단어의 dependency를 나타내는 left-Arc
3.왼쪽 단어에서 오른쪽 단어의 dependency를 나타내는 right-Arc

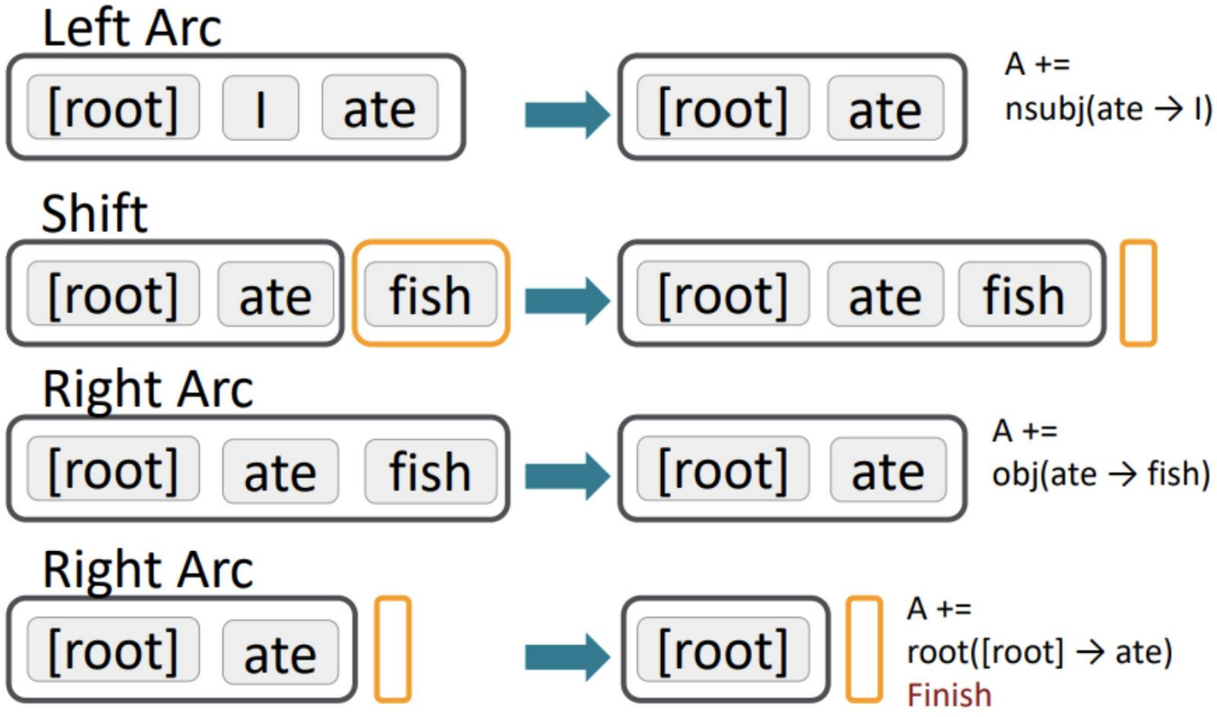
-> stack에 단어 하나만 남고, buffer가 비면 종료됨



#03. Dependency Parsing 방법

#2 Transition-based Parsing Process

1.Greedy transition-based parsing



이후 Left Arc를 통해 I와 ate의 관계를 Arc 집합(A)에 추가하고 dependent I를 제거한다.
다음 또 Shift하여 fish를 Stack으로 가져온다.
이후 Right Arc를 통해 ate에서 fish로의 관계를 Arc 집합(A)에 추가하고 dependent fish를 제거한다.
Right Arc를 한 번 더해서 ROOT와 ate의 관계를 Arc 집합(A)에 추가하고 ate를 제거 후 종료 조건을 만족하므로 종료한다.

#03. Dependency Parsing 방법

#2 Transition-based Parsing Process

2.MaltParser

여기서 어떤 기준으로 다음 액션을 정할까?

정답은 없음, 모든 경우의 수를 진행하게 되면 지수적으로 문장 크기에 따라 증가
⇒next action classifier에 machine learning을 이용
⇒각각의 action을 예측하는 classifier를 구축하고 run classifier - choose next action 을 반복하여 parsing을 진행

•class는 action이므로 최대 3개
•관계의 총 경우의 수는 최대 R x 2(단어 당 arc가 두 개) + 1(shift) 이다.
•classifier의 feature: top of stack word, POS; first in buffer word, POS

탐색을 진행하지 않고 greedy 하게 진행 (beam search를 쓰면 더 개선)
성능은 비록 최신 dependency parsing에 못미치지만, 문장길이에 대해 매우 빠른 linear time parsing이 가능

#03. Dependency Parsing 방법

#2 Transition-based Parsing Process

2.MaltParser

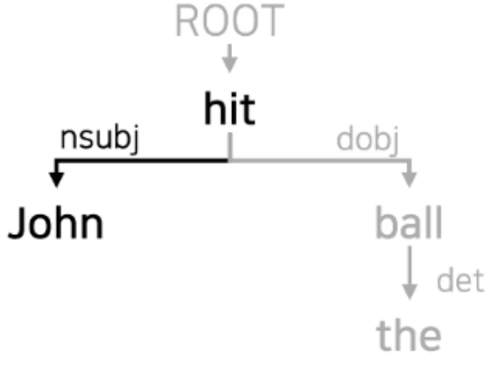
state c

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

POS Tags

John	NNP
hit	VBD
the	DT
ball	NN

Parsing Tree



notations

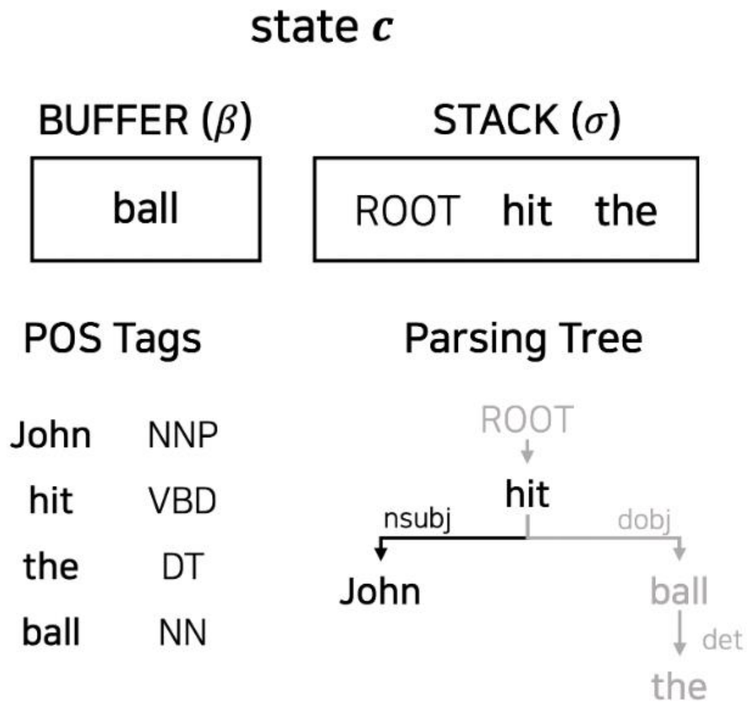
$s_1 \cdot w$	STACK의 첫번째 단어
$b_1 \cdot w$	BUFFER의 첫번째 단어
$s_1 \cdot t$	STACK의 첫번째 단어의 POS Tag
$lc(s_1) \cdot w$	STACK의 첫번째 단어의 left-child 단어
$rc(s_1) \cdot w$	STACK의 첫번째 단어의 right-child 단어
$lc(s_1) \cdot t$	STACK의 첫번째 단어의 left-child 단어의 POS Tag

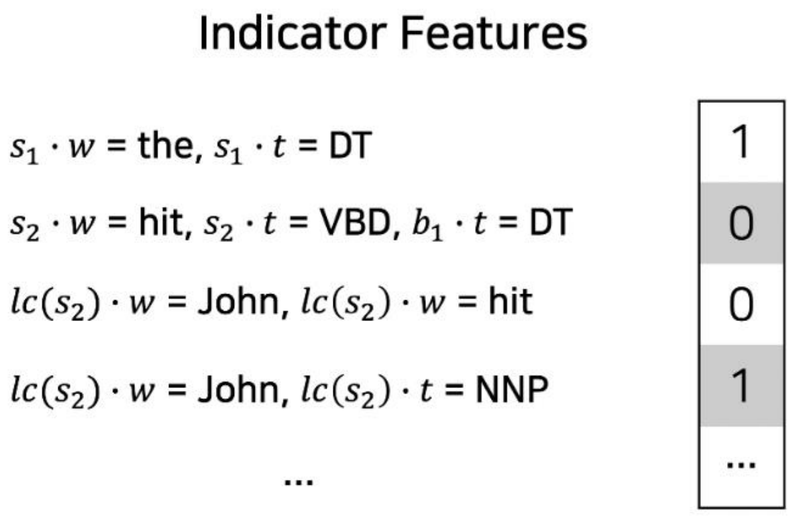
$s_1 \cdot w$	$b_1 \cdot w$	$lc(s_2) \cdot w$	$rc(s_2) \cdot t$
the	NN	John	NULL

#03. Dependency Parsing 방법

#2 Transition-based Parsing Process

3. Conventional Feature Representation



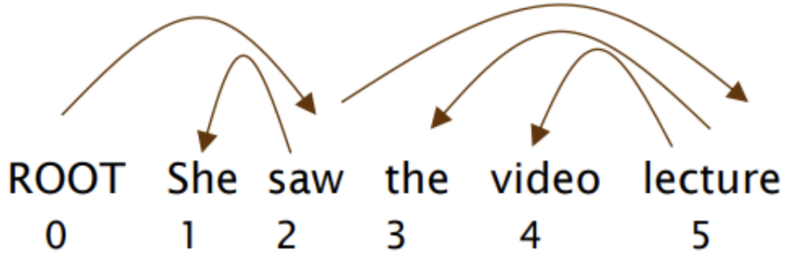


고전적인 방식에선 모든 단어들 간의 관계와 각 단어의 품사를 반영한 indicator features 구성해서 원-핫 인코딩으로 binary, sparse한 feature representation을 생성 (굉장히 비효율적)

#03. Dependency Parsing 방법

#3 Dependency Parsing 평가 방법

Dependency 정확도



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

UAS = 4 / 5 = 80%

LAS = 2 / 5 = 40%

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

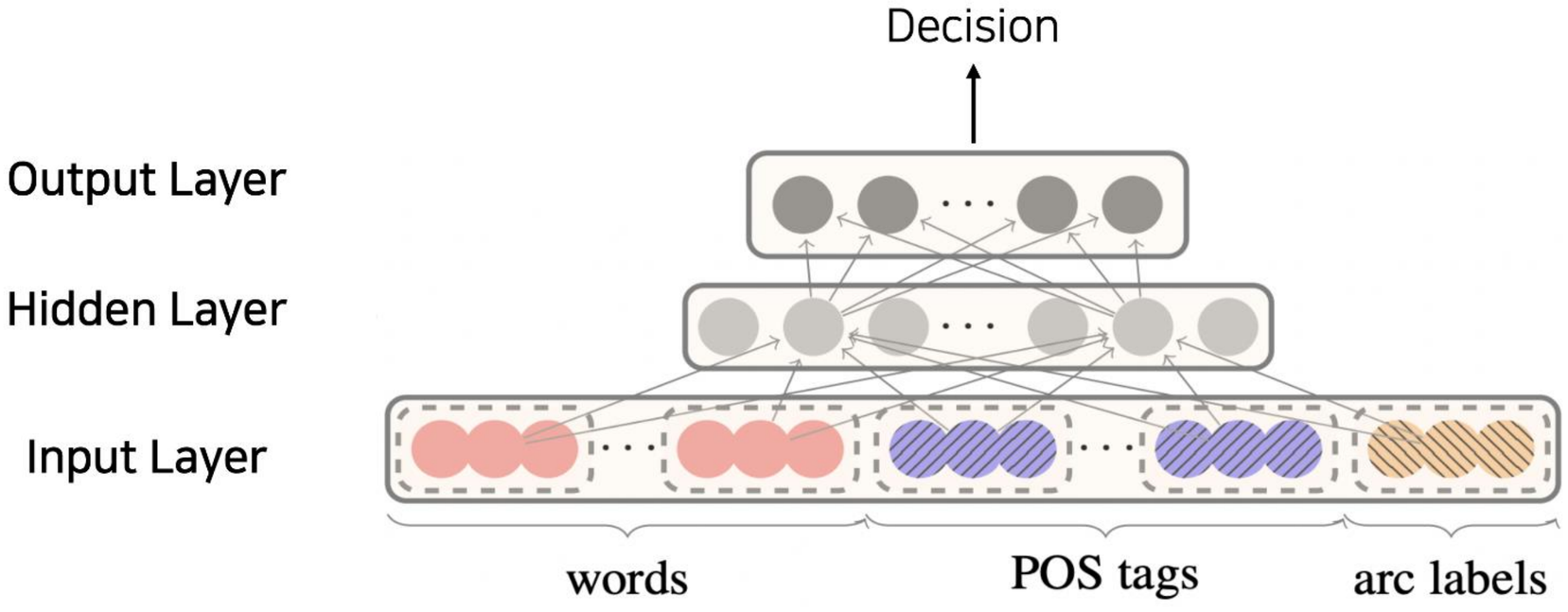
Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

unlabelled accuracy score(UAS): head와 dependent가 일치하는지 여부만 판단
labelled accuracy score(LAS): head와 dependent가 일치하는지에 더불어 관계도 올바르게 태깅됐는지 확인

#03. Dependency Parsing 방법

#4 Neural Dependency Parser

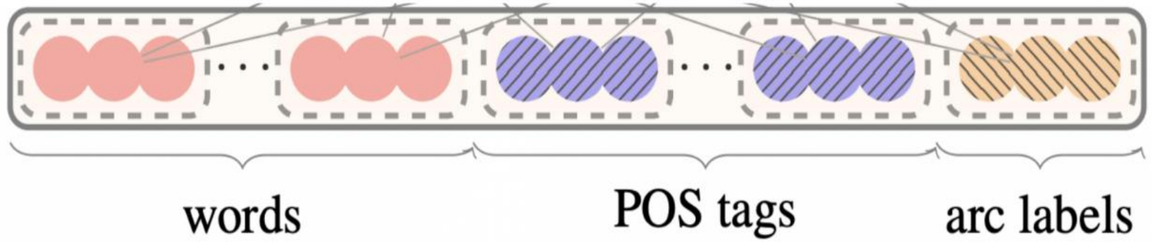


Neural dependency parser는 모델 구조: 기본적인 feed forward network와 닮았음
그러나, feature representation하는 과정과 hidden layer의 활성화 함수에서 차별점

#03. Dependency Parsing 방법

#4 Neural Dependency Parser

Feature Selection



- STACK과 BUFFER의 Top 3 단어 (6개)
  $[s_1, s_2, s_3, b_1, b_2, b_3]$
- STACK Top 1, 2 단어의 첫번째, 두번째 left & right child 단어 (8개)
  $[lc_1(s_1), rc_1(s_1), lc_2(s_1), rc_2(s_1), lc_1(s_2), rc_1(s_2), lc_2(s_2), rc_2(s_2)]$
- STACK Top 1, 2 단어의 left of left & right of right child 단어 (4개)
  $[lc_1(lc_1(s_1)), rc_1(rc_1(s_1)), lc_1(lc_1(s_2)), rc_1(rc_1(s_2))]$

word features의 해당 POS tag (18개)

word features의 해당 arc-label (12개)

Input으로 입력되는 feature: words, POS tag, arc labels

1) words feature 데이터: 총 18개로 구성
STACK과 BUFFER의 TOP 3 words (6개)

STACK TOP 1, 2 words의 첫번째,
두번째 left & right child word (8개)

STACK TOP 1,2 words의 left of left & right of right child word (4개)

2) POS tags feature데이터:
words feature에서 들어가는 데이터의 태그를 의미 (18개)

3) Arc labels 데이터:
STACK과 BUFFER의 TOP 3 words 6개를 제외한 12개의 label(dependent 관계 표시)

EWHA EURON

#03. Dependency Parsing 방법

#4 Neural Dependency Parser

One hot Representation

words

[**cake**, Null, Null, the, **ate**, ROOT,
Null, Null, Null, Null, **Joe**, Null, Null, Null,
Null, Null, Null, Null]

One-hot

단어의 개수

	Null	ROOT	NNP	VBD	DT	NN
cake	0	0	0	0	0	1
Null	1	0	0	0	0	0
…	…					
ate	0	0	0	1	0	0

18

POS tag

[**NN**, Null, Null, the, **VBD**, ROOT,
Null, Null, Null, Null, **NNP**, Null, Null, Null,
Null, Null, Null, Null]

One-hot

POS tag 개수

	Null	ROOT	NNP	VBD	DT	NN
NN	0	0	0	0	0	1
Null	1	0	0	0	0	0
…	…					
VBD	0	0	0	1	0	0

18

Arc-label

[Null, Null, Null, Null, **nsubj**, Null, Null, Null,
Null, Null, Null, Null]

One-hot

Arc-label 개수

	Null	ROOT	dobj	nsubj	det	…
Null	1	0	0	0	0	0
Null	1	0	0	0	0	0
…	…					
nsubj	0	0	0	1	0	0

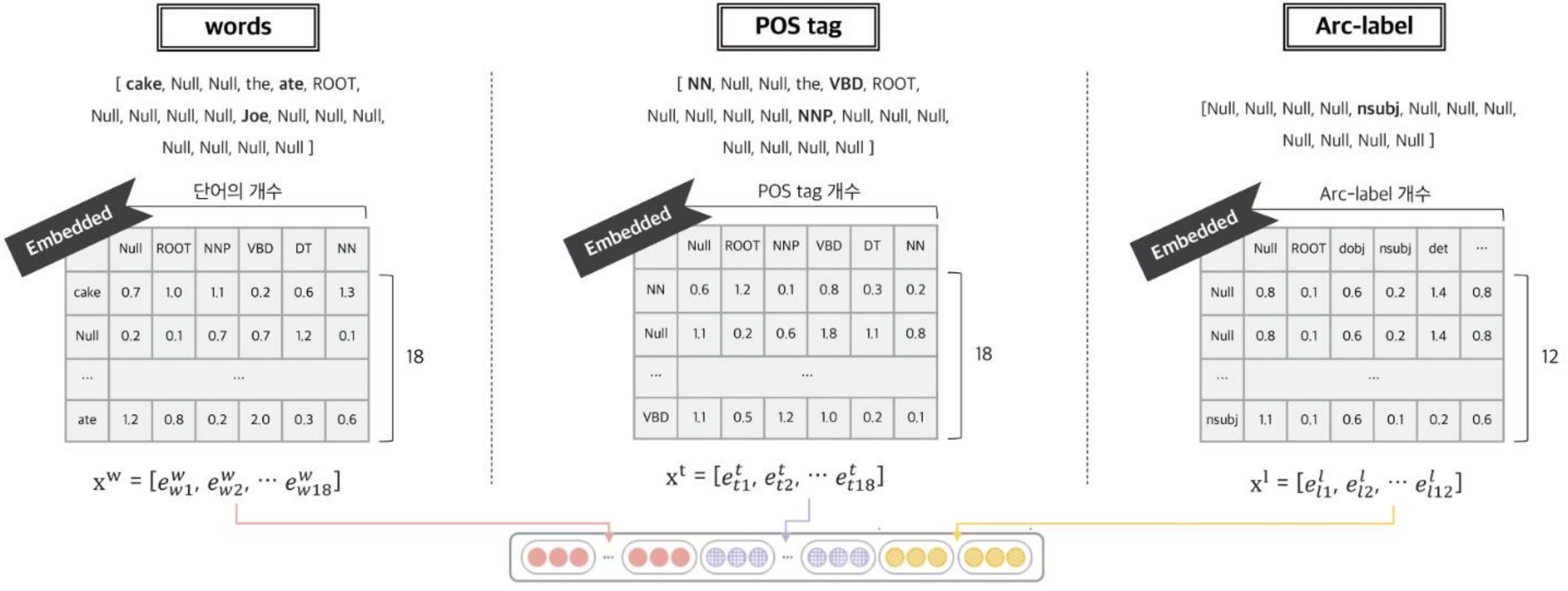
12

EWHA EURON

#03. Dependency Parsing 방법

#4 Neural Dependency Parser

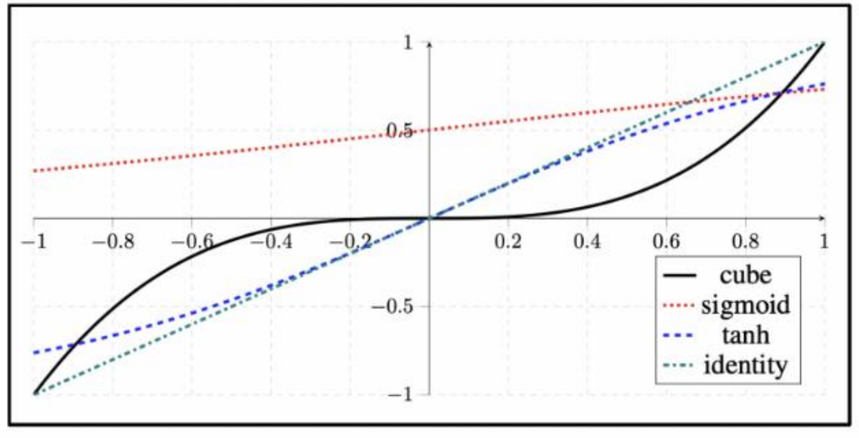
Feature Embedding



#03. Dependency Parsing 방법

#4 Neural Dependency Parser

Hidden layer



$$g(x) = x^3$$

$$g(w_1x_1 + \ldots + w_mx_m + b) =$$

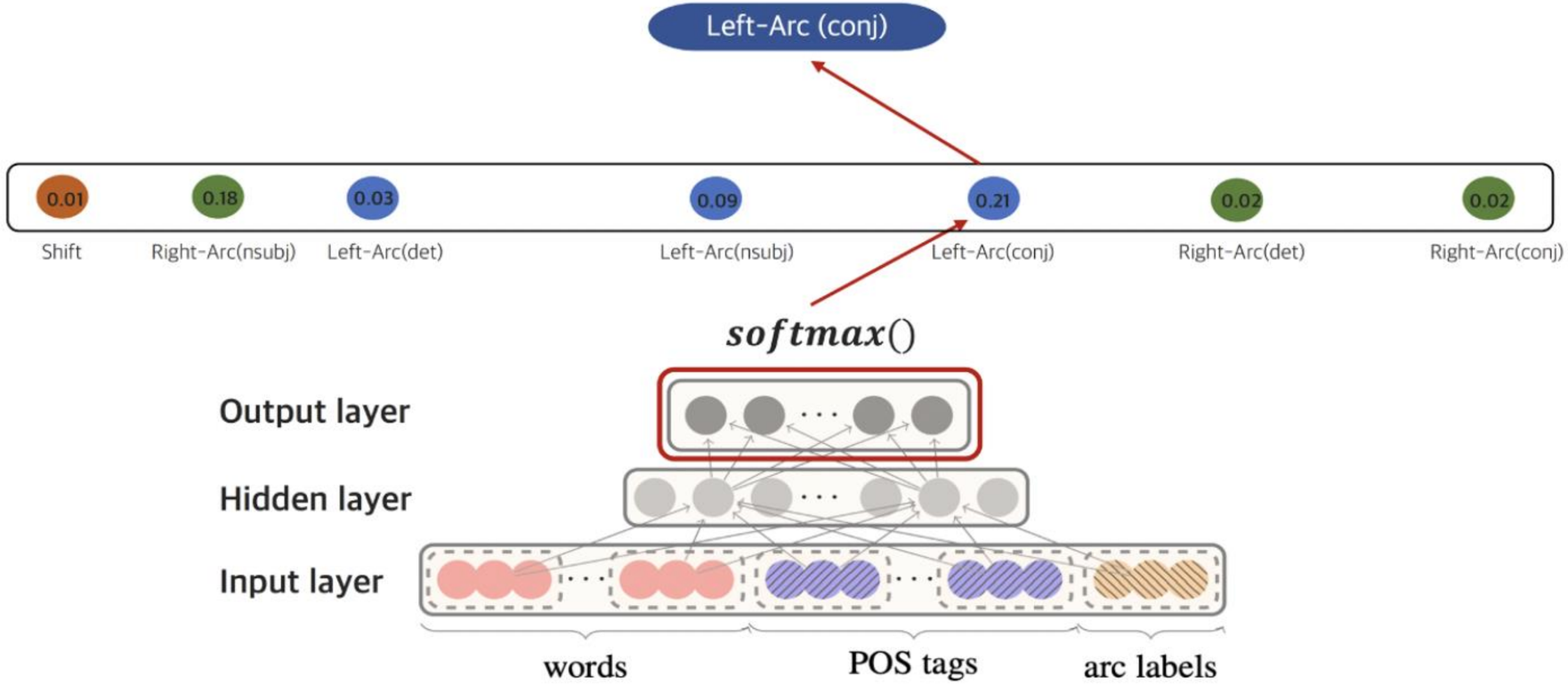
$$\sum_{i,j,k} (w_iw_jw_k)x_ix_jx_k + \sum_{i,j} b(w_iw_j)x_ix_j \ldots$$

Words, POS tag, Arc-label간
상호작용을 반영할 수 있는 조합

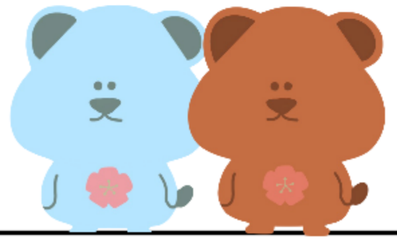
#03. Dependency Parsing 방법

#4 Neural Dependency Parser

Output layer



THANK YOU



EWHA EURON