exam 개인학습

Lec 4

AI summary Try on this page

Type something...

01) Syntactic Strcture and Dependency Parsing

- 이번 시간에는 문장 단위에서 단어간 상관관계가 어떻게 이뤄지는지 알아볼 것이다.

1. constituent

1. The linguistic structure of sentences – two views: Constituency = phrase structure grammar = context-free grammars (CFGs)

Phrase structure organizes words into nested constituents

Starting unit: words

the, cat, cuddly, by, door

Words combine into phrases

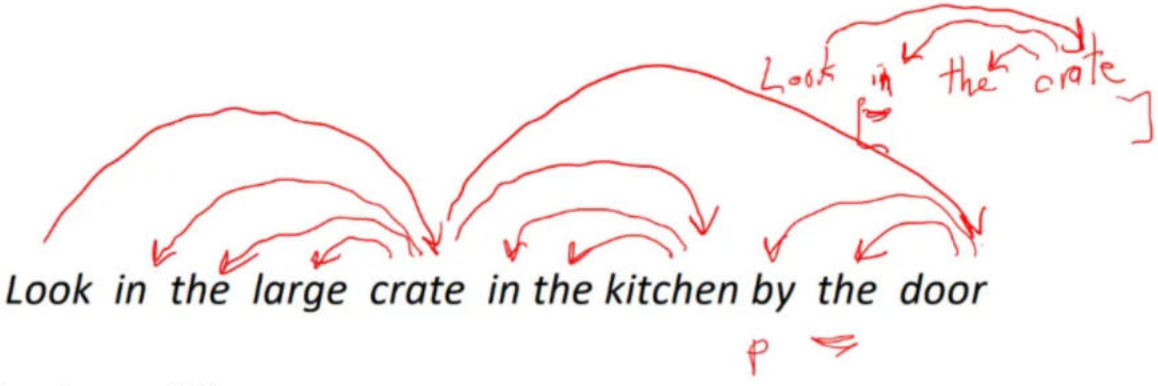
the cuddly cat, by the door

Phrases can combine into bigger phrases

the cuddly cat by the door

- 구문 구조를 단어를 중첩된 **constituent(성분)**들로 구성한다고 보는 관점이 하나 있다.

2. Dependency structure



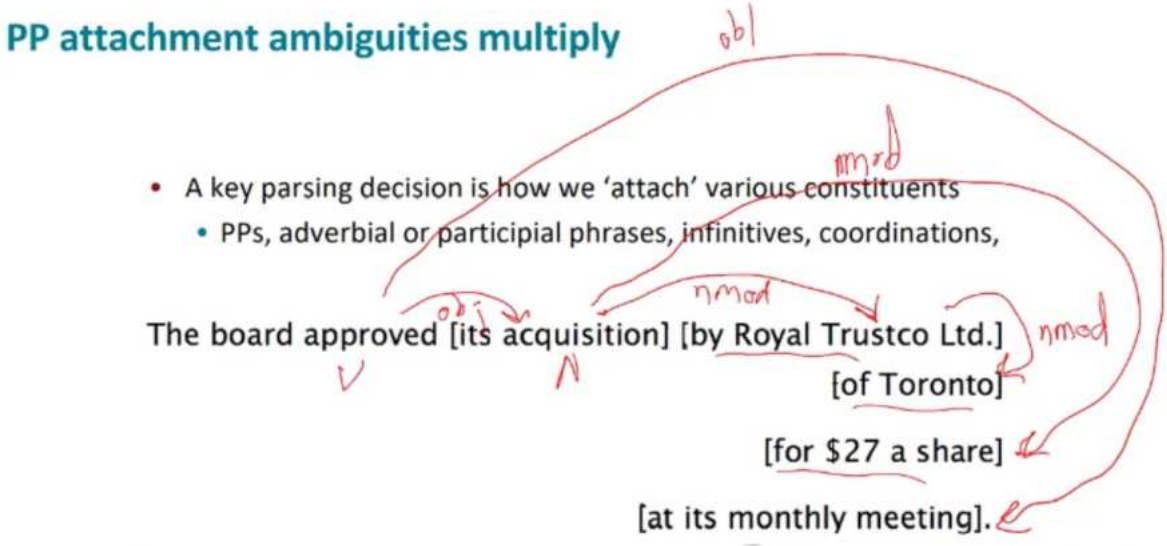
- 또 다른 방식은 **Dependency structure가 있다.**
- dependency 구조는 한 단어가 다른 단어에 어떻게 의존하는지를 보는 방식이다.

- 위의 예시에서 Look은 create를 수식하고 create는 in과 the를 수식한다.
- 사람이 문장구조를 사용하는 이유는 한 단어로는 복잡한 뜻을 전달할 수 없기 때문이다.

3. Prepositional phrase attachment ambiguity

뉴스 기사 제목, with knife가 수식하는 대상이 man인지 cops인지 명확하지 않기에 중의적으로 해석이 가능하다.

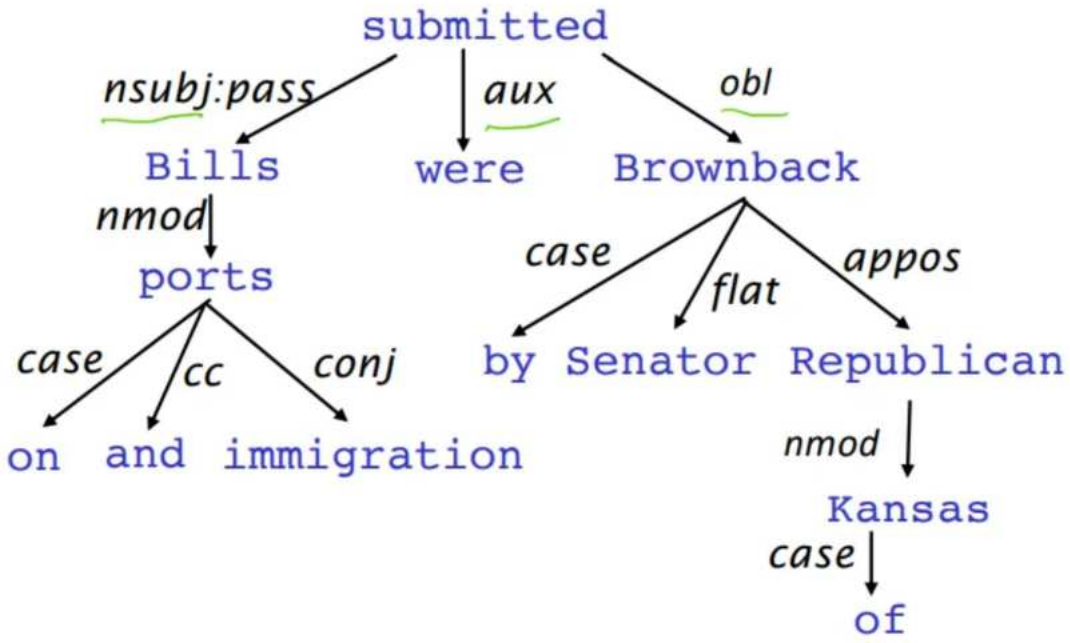
- 뉴스 제목을 보면 with knife가 수식하는 대상이 man인지 cops인지 명확하지 않기에 중의적인 해석이 가능하다.



전치사구가 4개인 문장. 각 구간 수식 관계의 경우의 수는 전치사구 수에 따라 Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$으로 생각할 수 있다.

- 또한 수식 범위에 따라 해석이 달라지는 경우도 있다.

4. Dependency 문법과 구조



- dependency구조는 어휘(lexical) 항목들 간의 관계와 dependencies로 불리는 비대칭 이진 관계(화살표)로 이루어진다.
- 일반적으로 문법적 관계로 typed진다.
- 이 때 화살표의 시작 부분은 head라고 하며 가리키는 방향은 dependent라고 한다.
- 맨 꼭대기가 root가 되며 일반적으로 root는 하나이다.

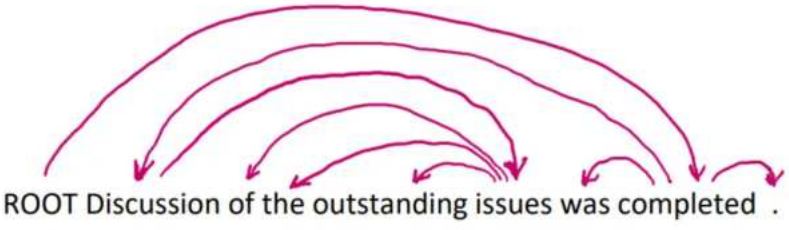
- Dependency 구조를 만들 때 fake Root를 추가하여 모든 단어가 dependent하도록 한다.
- 화살표의 방향은 head to dependent로 그린다.

5. Dependency Conditioning Preferences

Dependency Conditioning Preferences

What are the straightforward sources of information for dependency parsing?

1. Bilexical affinities — The dependency [discussion → issues] is plausible
2. Dependency distance — Most (but not all) dependencies are between nearby words
3. Intervening material — Dependencies rarely span intervening verbs or punctuation
4. Valency of heads — How many dependents on which side are usual for a head?

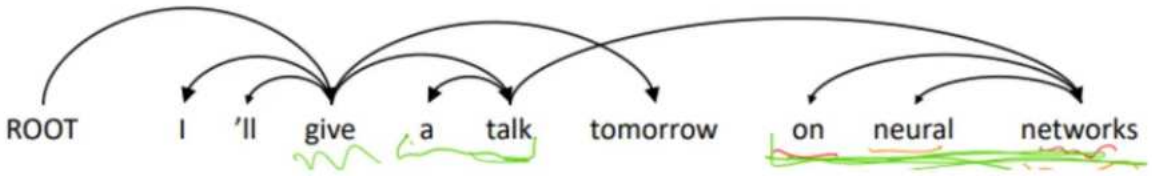


- Dependency parsing에 사용되는 정보는 어디서 오는가?

1. Bilexical affinities(단어 간 친화력): 단어 간 관계로 dependency는 그럴 듯 하다.
2. Dependecy distance: dependency와 연관된 단어의 거리는 일반적으로 인접해 있다.
3. intervening material: 동사나 구두점을 기준으로 dependencies가 확장되지 않는다.
4. valency of head: 어느 방향(왼쪽, 오른쪽)이 head가 dependents를 많이 갖는가?

6. Dependency Parsing 규칙

- Root에는 단어가 하나 뿐이다.
- 순환하지 않는다. (A → B, B → A)
- projectivity: 화살표가 겹치지 않는다.



- give → tomorrow와 talk → networks 화살표가 교차하므로 non-projective하다.
- on neural networks 전치사구를 tomorrow 앞으로 당겨옴으로써 같은 의미의 문장을 projective하게 만들 수 있다.

7. Dependency Parsing의 방법

3. Methods of Dependency Parsing

1. Dynamic programming
   Eisner (1996) gives a clever algorithm with complexity $O(n^3)$, by producing parse items with heads at the ends rather than in the middle
2. Graph algorithms
   You create a Minimum Spanning Tree for a sentence
   McDonald et al.'s (2005) $O(n^2)$ MSTParser scores dependencies independently using an ML classifier (he uses MIRA, for online learning, but it can be something else)
   Neural graph-based parser: Dozat and Manning (2017) et seq. – very successful!
3. Constraint Satisfaction
   Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.
4. "Transition-based parsing" or "deterministic dependency parsing"
   Greedy choice of attachments guided by good machine learning classifiers
   E.g., MaltParser (Nivre et al. 2008). Has proven highly effective. And fast.

---

1. Dynamic programming: O(n^3) 차원으로 비효율적
2. Graph algorithms: 매우 성공적
3. contraint satifcation
4. Transition based parsing

8. Greedy transition-based parsing

Greedy transition-based parsing [Nivre 2003]

- A simple form of a greedy discriminative dependency parser
- The parser does a sequence of bottom-up actions
  - Roughly like "shift" or "reduce" in a shift-reduce parser – CS143, anyone?? – but the "reduce" actions are specialized to create dependencies with head on left or right
- The parser has:

- a stack σ, written with top to the right
  - which starts with the ROOT symbol
- a buffer β, written with top to the left
  - which starts with the input sentence
- a set of dependency arcs A
  - which starts off empty
- a set of actions

- 가장 간단한 형태의 greedy discriminative dependency parser이다.
- Parser는 bottom-up 순서로 구축해 나아간다.

Start: $\sigma = [ROOT], \beta = w_1, \ldots, w_n, A = \emptyset$

1. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A \cup \{r(w_j, w_i)\}$
3. Right-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A \cup \{r(w_i, w_j)\}$

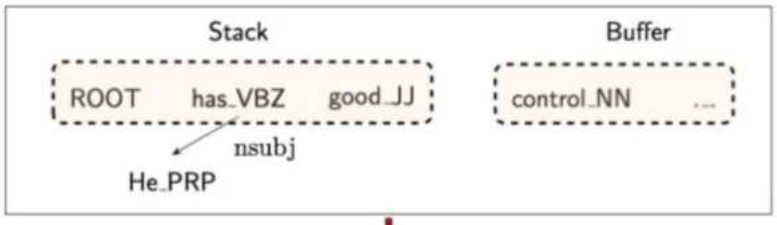
Finish: $\sigma = [w], \beta = \emptyset$

기본 transition-based dependency parser

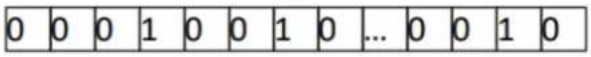
$Stack(\sigma)$: $root\ symbol$시작
$Buffer(\beta)$: 입력 문장으로 시작
$dependency\ arc$ 집합(A) : 비어있는 상태로 시작

- buffer에서 stack으로 단어를 가져오는 shift
- 오른쪽 단어에서 왼쪽 단어의 dependency를 나타내는 left-Arc
- 왼쪽 단어에서 오른쪽 단어의 dependency를 나타내는 right-Arc
- 예시를 통해서 어떻게 작동하는지 알아보았다.

- 그렇다면 여기서 어떤 기준으로 다음 액션을 정할까?



binary, sparse
dim $=10^6-10^7$



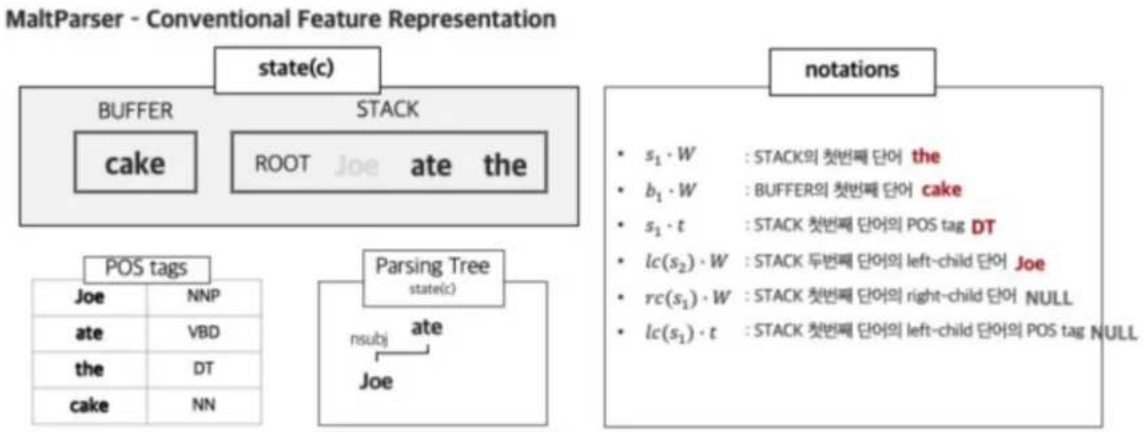
Feature templates: usually a combination of 1–3 elements from the configuration

Indicator features

$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$
$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

- 고전적인 방식에서는 모든 단어들 간의 관계와 각 단어의 품사를 반영한 indicator features를 구성해서 one-hot encoding으로 binary, sparse한 feature representation을 생성한다. 하지만 굉장히 비효율적이다.

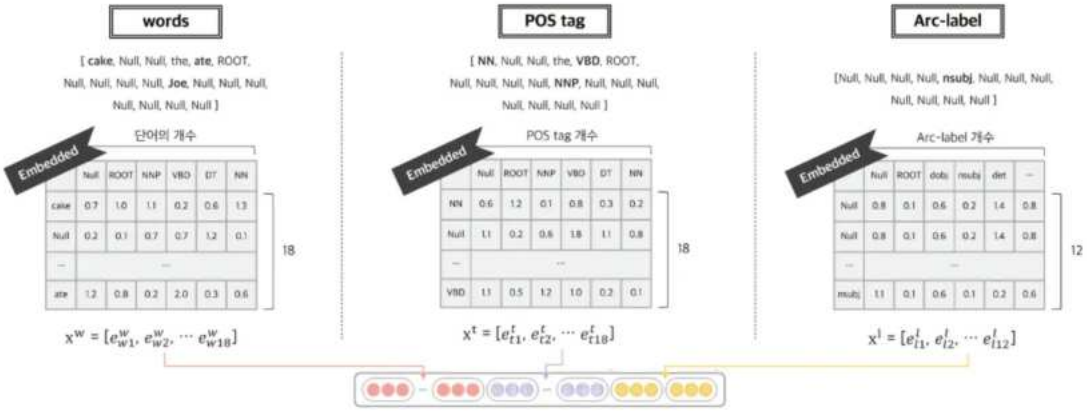
9. MaltParser



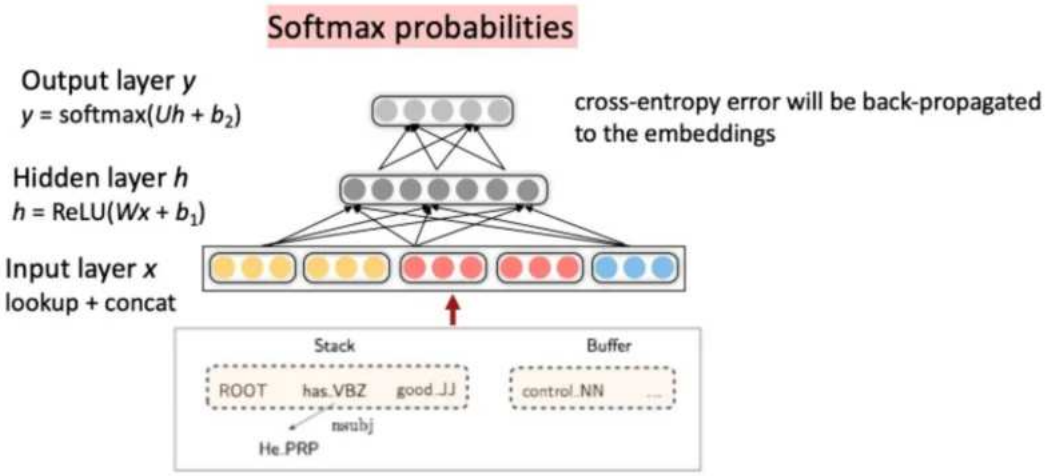
- Postagging과 notation을 기반으로 indicator features를 설정함으로 state를 embedding할 수 있습니다.
- s1 * W: Stack의 첫 번째 단어 the
- b1 * W: buffer의 첫 번째 단어 cake
- s1 * t: Stack의 첫 번째 단어의 POS tag : DT
- 이렇게 만든 state를 one-hot 벡터로 표현한다.
- 너무 sparse하고 계산이 비싸다.

10. Neural dependency parser

- Input: words, POS tag, arc labels 3가지
- words feature 18개
  1. STACK과 BUFFER의 TOP 3 words (6개)
  2. STACK TOP 1, 2 words의 첫 번째, 두 번째 left & right child word (8개)
  3. STACK TOP 1, 2 words의 left of left & right of right child word (4개)
- POS tag 18개 = word feature
- Arc label: 12개의 label의 dependent 관계(TOP 3 words 제외)



- 다음으로 word embedding matrix를 참고하여 해당 토큰의 벡터를 가져올 수 있게 됩니다.
- 각 토큰별로 벡터가 있는 상태에서 concatenate로 결합하여 input layer에 넣어준다.



- Input layer: lookup + concat
- hidden layer: ReLu
- Output layerr: softmax

Input layer → Linear → ReLU → Linear → Softmax

- 최종으로 나온 Shift, Left-Arc, Right-Arc 중 확률값이 가장 높은 decision이 최종 output이 된다.