Dependency Parsing

Wongyu Kim (김원규)

Internet Computing Laboratory
Department of Computer Science
Yonsei University

2020.08.20

Parsing

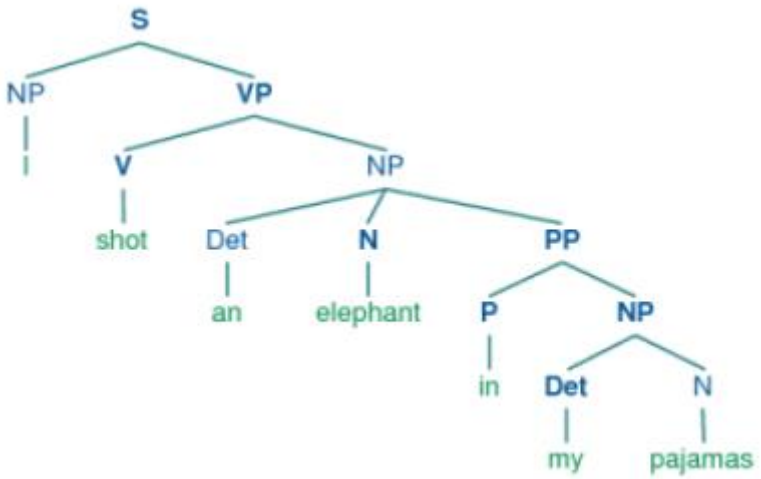
- 단어들이 base가 돼 문장을 이룸.
- 따라서 여러 형태의 문장이 나올 수 있음.
- 결국, 문장의 구조를 분석하는 것은 문장을 정확히 이해하는데 필수적임.(= 문장이 무엇을 의미하는지 알 수 있다.)

- Parsing은 구문 분석, 구조 분석을 뜻함.(쪼개고, 나누고, 분석하고 ~) 크게 두 가지의 구조 분석 방법이 있다.

1. Phrase Structure(= Context-Free Grammars(CFGs))
2. Dependency Structure

Phrase Structure

- 단어들이 모여 phrase를 이루고, phrase가 합쳐서 더 큰 phrase를 산출.
- 문장의 구조를 단어들의 품사(카테고리, part of speech, POS)로 파악



- 하지만 CFGs는 영어의 문법에 특화
- 언어에 상관없이 적용되는 Parsing이 필요함.
- 또한 Context based로 다른 단어와의 관계를 설명하는 Parsing이 필요함.

Dependency Structure

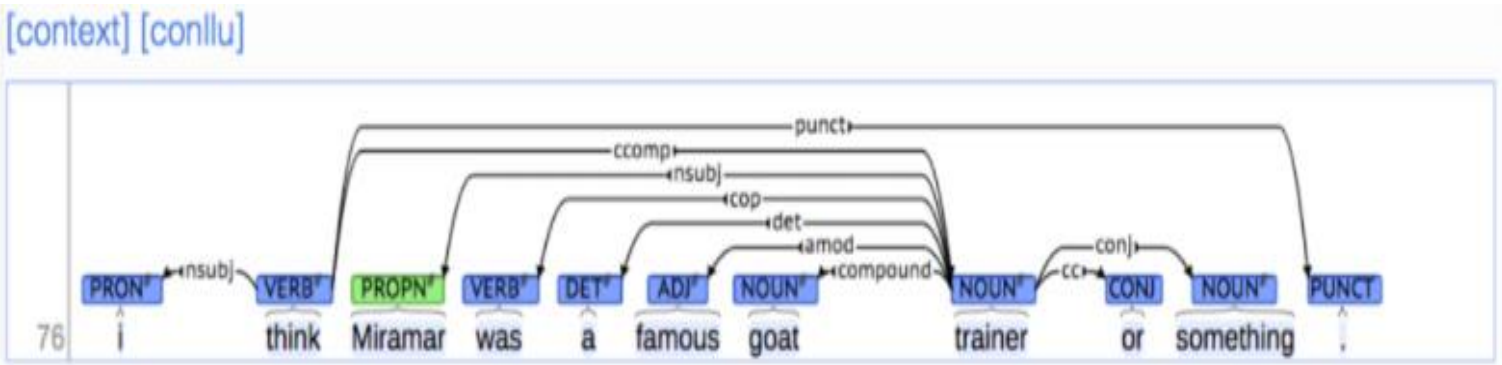
- 따라서 이에 맞는 Parsing이 Dependency Structure임.
- Ex) “Look in the large crate in the kitchen by the door”
  <Look(superior, head) -> crate(dependent, modifier)>

- 즉, 단어간의 의존 관계를 문장에서 파악하는 것이 핵심!

- 하지만 데이터들 중 중의적인 의미를 가진 문장이 있을 수 있다.
- Ex) “San Jose cops kill man with knife”
  (kill -> man), (kill -> knife) 가 될 수도 있고
  (kill -> man), (man -> knife) 가 될 수도 있다.

Dependency Structure

- Tree bank 데이터셋: 단어간의 dependency를 나타내고 각 단어의 품사 정보를 가진 데이터셋이다. 이를 통해 학습하면 parser, POS tagger로 사용할 수 있다.



- Datset의 특징
  * Binary Asymmetric Arrow
  * Arrow 에는 Type(문장 성분)이 적힘.
  * 보통 Fake Root 를 추가함. (가상의 단어)
  * Reusable

Dependency Structure

- 또한 Tree Bank와 같은 Dataset을 만들기 위해 Dependency Parsing 을 처음에는 인간이 직접해야 한다. 그때 고려해야 할 점이 있다.

* Bilexical Affinities: 두 단어 사이의 실제 의미에 드러나는 관계를 의미
ex) [discussions -> issues]와 같은 단어의 의미 상으로 dependency가 있을 법한 관계에 주목

* Dependency Distance: dependency는 가까운 위치에서 나타난다 라는 사실에 주목

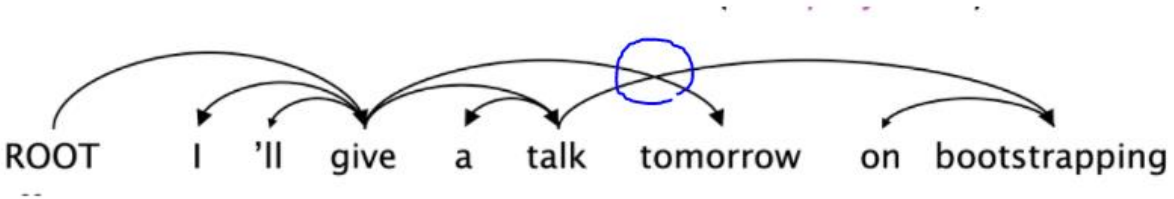
* Intervening Material: dependent 단어들 사이에 세미콜론이나 구둣점 뛰어 넘지 않음

* Valency of Head: 어느 정도 기본 rule에 따름 / 특정 단어 앞뒤로 가질 수 있는 dependency 개수를 고려

* ROOT에 dependent 단어는 1개 (대개 동사)

* A -> B, B -> A 의 cycle dependency는 허용하지 않음

* 또한 아래 처럼 cross(non-projective)가 발생해도 무관하다. (CFGs 규칙은 아래와 같은 cross 상황이 발생할 수 없음.)



Dependency Structure

최종 목표는 이러한 데이터셋을 이용한 모델이나 알고리즘인

Parser를 만드는 것

Arc-Standard Transition-Based Parser(2003)

- Action을 Greedy 하게 선택 (Action을 선택하는 자세한 Logic은 생략)

  1. Stack에는 [ROOT], 입력 문장은 Buffer에 담는다.
  2. 3개의 Actions 중 1개를 선택해 실행한다.
  3. 1~2를 반복하되 Logic이 종료될 때는 Stack에 [ROOT], Buffer는 Empty여야 한다.

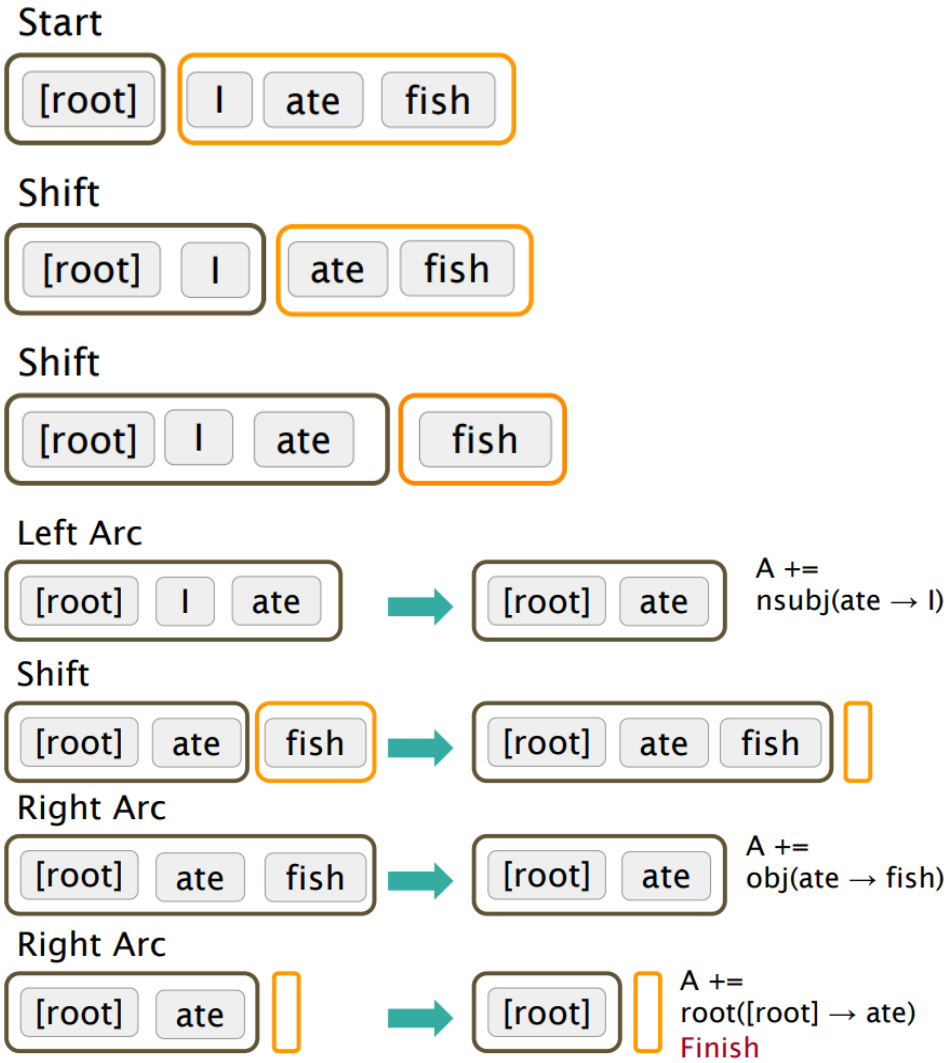
Start: $\sigma = [ROOT], \beta = w_1, \ldots, w_n, A = \emptyset$

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

Finish: $\beta = \emptyset$

- Shift: Buffer Top Word -> Stack Top Word
- Left-Arc: Stack Top Word 가 바로 왼쪽 단어와 dependency 맺음
- Right-Arc: Stack Top Word 를 바로 왼쪽 단어가 dependency 맺음

Arc-Standard Transition-Based Parser(2003)



문제점:

Greedy 한 Action의 선택은 적절한 선택을 못할 가능성이 크다.

-> 이렇게 지속된다면 Dependency Parsing이 엉망이 될 것이다.

MaltParser(2005)

- **Action 선택을 ML Classify를 도입했다. (여러 Action classes 중 가장 확률이 높은 Action 선택 = output은 Actions의 확률이라 생각)**

  1. Stack에는 [ROOT], 입력 문장은 Buffer에 담는다.
  2. 3개의 Actions 중 1개를 선택해 실행한다.
  3. 1~2를 반복하되 Logic이 종료될 때는 Stack에 [ROOT], Buffer는 Empty여야 한다.

  이 룰은 기본적으로 똑같고 ML로 Classification 하는 것만 다른 것이다.

  - Word Embedding: BoW (Vocab V dimension)

  - Features: [top of stack word(V), POS of that word(M), top of buffer word(V), POS of that word(M), …]
  (각 Feature를 Concatenate 하여 ML input으로 사용하게 된다.)
  (word를 제외한 feature는 one-hot vector로 나타낸다.)

  * SOTA는 아니지만 Linear Time + 괜찮은 성능

- **문제점**

  - Feature 크기가 너무 크다. (Feature를 만드는데 95% 이상의 시간이 쓰임.)

  - 보지 못한 단어는 vector화 할 수 없다. = incomplete
  (word2vec은 학습 데이터에 없었을 지라도 모델이 판단하여 변환)

C & M Parser(2014)

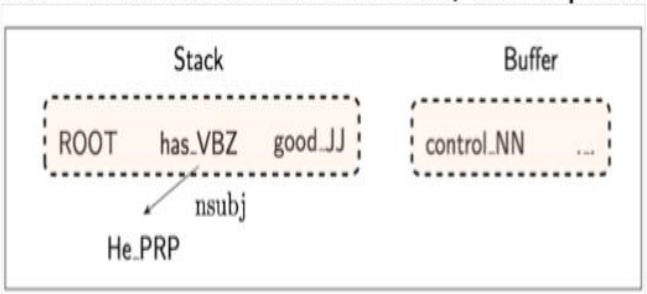
- Action 선택을 NN Classify를 도입했다.

  * Concatenated Features 의 크기가 너무 큰 문제를 해결하고, Network를 좀 더 깊이 만들었다.

  * 각 word는 word2vec 과 같은 저차원 임베딩 사용

  * POS와 Dependency Labels 도 저차원으로 vector화 해준다.

- We extract a set of tokens based on the stack / buffer positions:

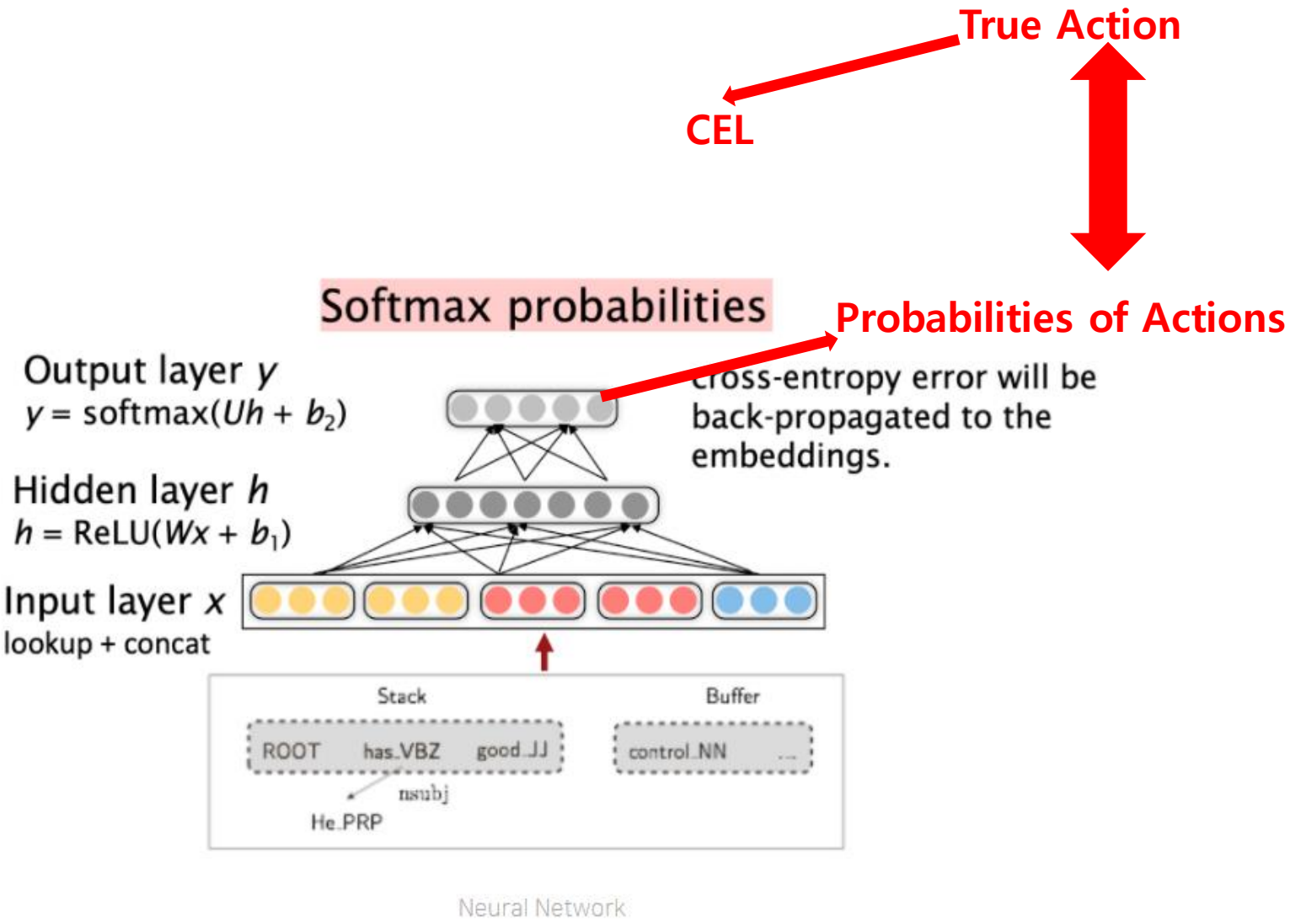


	word	POS	dep.
s_1	good	JJ	ø
s_2	has	VBZ	ø
b_1	control	NN	ø
$lc(s_1)$	ø	ø	ø
$rc(s_1)$	ø	ø	ø
$lc(s_2)$	He	PRP	nsubj
$rc(s_2)$	ø	ø	ø

- We convert them to vector embeddings and concatenate them

C & M Parser(2014)

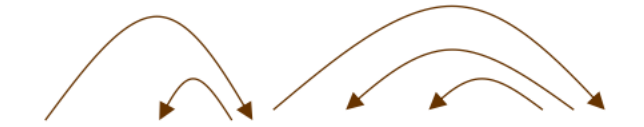
- Action 선택을 NN Classify를 도입했다.



Evaluation(Metric)

- UAS (Unlabeled Attachment Score) : dependency 관계가 제대로 됐는지만 고려
- LAS(Labeled Attachment Score): dependency label까지 제대로 됐는지 고려

Evaluation of Dependency Parsing:
(labeled) dependency accuracy



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

UAS = 4 / 5 = 80%
LAS = 2 / 5 = 40%

Gold

MF	HEAD	MF WORD	Dependency Label
1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

Parsed

MF	HEAD	MF WORD	Dependency Label
1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

Evaluation(Metric)

- English parsing to Stanford Dependencies:
  - Unlabeled attachment score (UAS) = head
  - Labeled attachment score (LAS) = head and label

Parser	UAS	LAS	sent. / s
MaltParser	89.8	87.2	469
MSTParser	91.4	88.1	10
TurboParser	**92.3**	89.6	8
C & M 2014	92.0	**89.7**	**654**

참고 문헌

한글 블로그 자료

- https://pakalguksu.github.io/2020/02/28/CS224n-5%EA%B0%95-Dependency-Parsing/
- https://gnoej671.tistory.com/5
- https://jeongukjae.github.io/posts/cs224n-lecture-5-dependency-parsing/
- https://lee6boy.wordpress.com/2013/06/28/parsing-dependency-parsing-graph-based-parsing%EC%9D%B4-%EB%AD%94%EA%B0%80/

Cs224n-winter 2019-lecture 5

https://www.youtube.com/watch?v=nC9_RfjYwqA&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=5

CS224n-winter 2019-syllabus-lecture 5

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/

Questions & Answers

Thank you