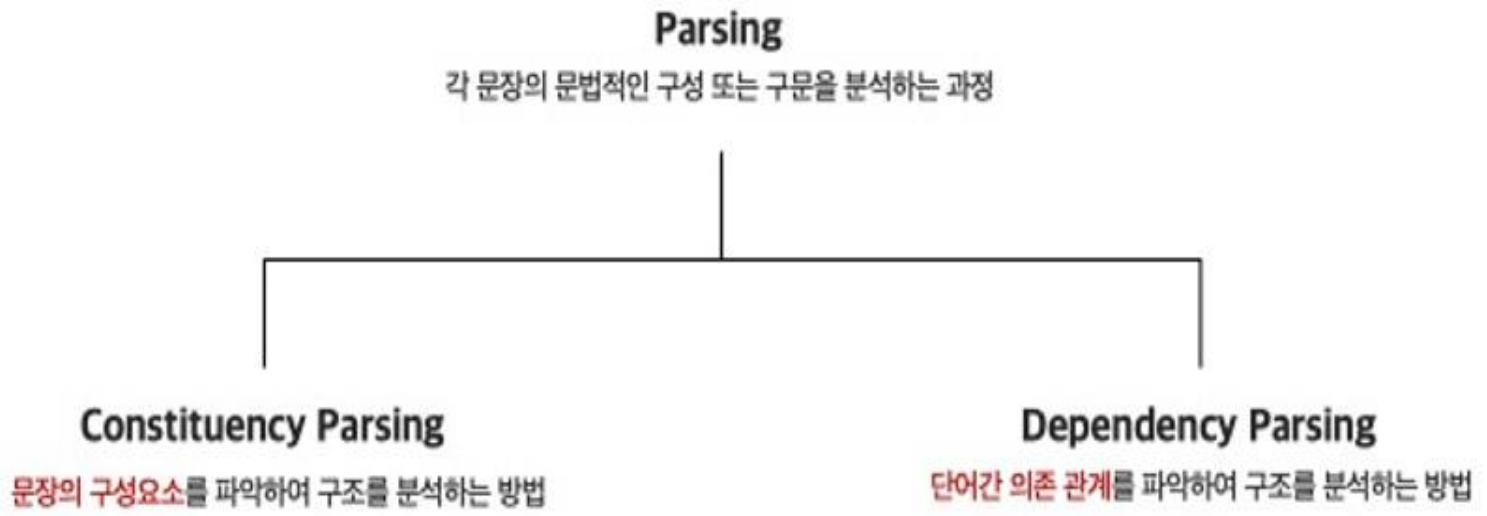
A Fast and Accurate Dependency Parser using Neural Networks

Danqi Chen, Christopher D. Manning

목차

01 용어 정리


Parsing
각 문장의 문법적인 구성 또는 구문을 분석하는 과정
Constituency Parsing
문장의 구성요소를 파악하여 구조를 분석하는 방법
Dependency Parsing
단어간 의존 관계를 파악하여 구조를 분석하는 방법

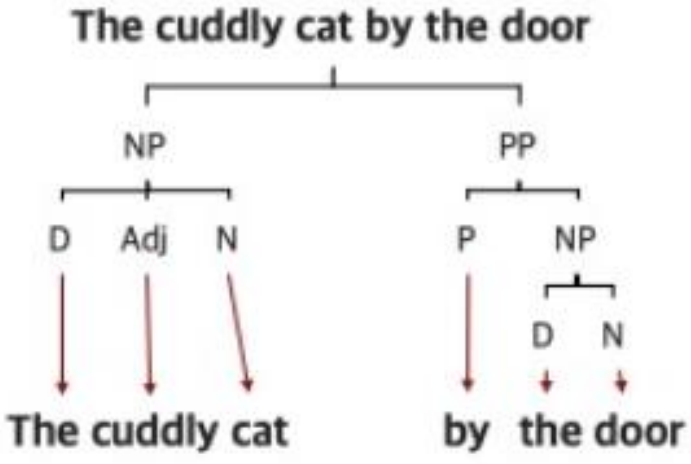
Parsing이 중요한 이유: 문장을 더 잘 이해할 수 있게 된다

Scientist observed whale from space

1. 과학자들이 위에서 고래를 관측했다.

2. 과학자들은 우주에서 온 고래를 관측했다

Constituency Parsing



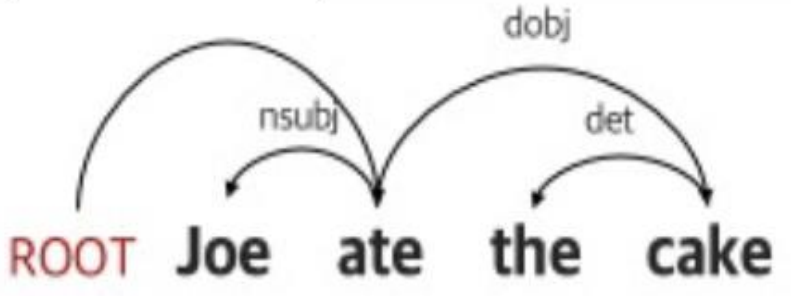
- 문장을 구성하고 있는 구를 파악하여 문장 구조를 분석하는 것

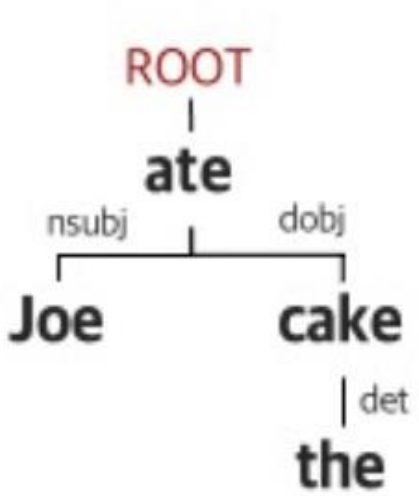
→ 각 단어의 문법적인 의미를 얻을 수 있게 된다.

- 단어 조합/순서가 고정적인 언어에서 사용된다.

Dependency Parsing

문장의 구성 요소 간의 의존 관계(Dependency)를 통해
구문 구조 (grammatical structure)를 파악하는 것



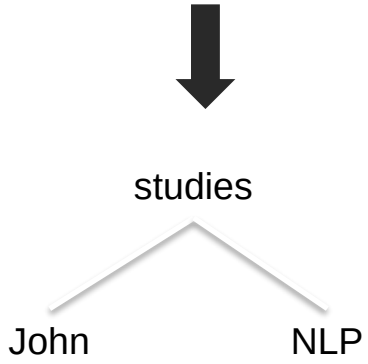


화살표의 의미 = 의존 관계를 시각적으로 나타내는 것
A → B : B는 A에 의존한다는 의미 == 간단하게 A를 꾸미는 것은 B다.

Dependency Parsing 왜 중요한가

Information Extraction (정보 추출, 요약)에 사용

Text: John studies NLP



Q: What did John study?

Studies의 목적어를 찾아서

A: NLP

Transition based Dependency Parsing

입력문장: He has good control.

Transition	Stack	Buffer		A
	[ROOT]	[He has good control .]		$\emptyset$
SHIFT	[ROOT He]	[has good control .]		
SHIFT	[ROOT He has]	[good control .]		
LEFT-ARC(nsubj)	[ROOT has]	[good control .]	주어	$A \cup$ nsubj(has,He)
SHIFT	[ROOT has good]	[control .]		
SHIFT	[ROOT has good control]	[.]		
LEFT-ARC(amod)	[ROOT has control]	[.]	형용사	$A \cup$amod(control,good)
RIGHT-ARC(dobj)	[ROOT has]	[.]	목적어	$A \cup$ dobj(has,control)
...	...	...		...
RIGHT-ARC(root)	[ROOT]	[]		$A \cup$ root(ROOT,has)

의존관계 집합

Transition based Dependency Parsing

두 단어의 의존 관계를 차례대로 결정해 나가면서 점진적으로 문장 분석을 하는 것이다.

Transition

- Left-Arc(l): (s1 → s2) arc를 A에 더한다. s1을 꾸미는 것은 s2라고 판단하고 s2를 stack에서 없앤다
- Right-Arc(l): (s2 → s1) arc를 A에 더한다. s2을 꾸미는 것은 s1라고 판단하고 s1을 stack에서 없앤다
- Shift: buffer 안의 맨 앞 단어 (입력 문장의 다음 단어)를 stack에 넣는다.

Transition based Dependency Parsing

$$configuration\ c\ =\ (s, b, A)$$

1. Stack (s): 의존 관계가 분석되길 기다리는 곳

$s_i\ (i = 1, 2, \ldots)$ as the i^{th} top element on the stack

2. Buffer (b): 입력 문장 $[w_1, \ldots, w_n]$

3. Set of dependency arcs (A) = 단어 간의 의존 관계 집합

02 연구 배경

기존 Dependency Parser 문제점

1. 의존관계를 파악하기 위해 단어에서 어휘적 특징과 상호작용 관계를 추출하는 과정이 어렵다.

2. 정보를 추출하기 위해 feature template이라는 것을 사용하는데 one hot vector를 사용하기 때문에 sparsity 문제가 발생한다.

3. Feature template은 고정된 것이기에 모든 단어들을 고려할 수 없다.

4. 모두 sparse하고 차원이 큰 matrix들을 연산하기 때문에 계산이 expensive하다.

Feature Template 예시

"A **cute** cat"

↓

Word feature template

Word	Cute
Previous word	A
Next word	Cat
Prefix	
suffix	

→

A	0
…	0
Cute	1
…	0
zebra	0

Single-word features (9)

$s_1.w$; $s_1.t$; $s_1.wt$; $s_2.w$; $s_2.t$;
$s_2.wt$; $b_1.w$; $b_1.t$; $b_1.wt$

Word-pair features (8)

$s_1.wt \circ s_2.wt$; $s_1.wt \circ s_2.w$; $s_1.wts_2.t$;
$s_1.w \circ s_2.wt$; $s_1.t \circ s_2.wt$; $s_1.w \circ s_2.w$
$s_1.t \circ s_2.t$; $s_1.t \circ b_1.t$

Three-word feaures (8)

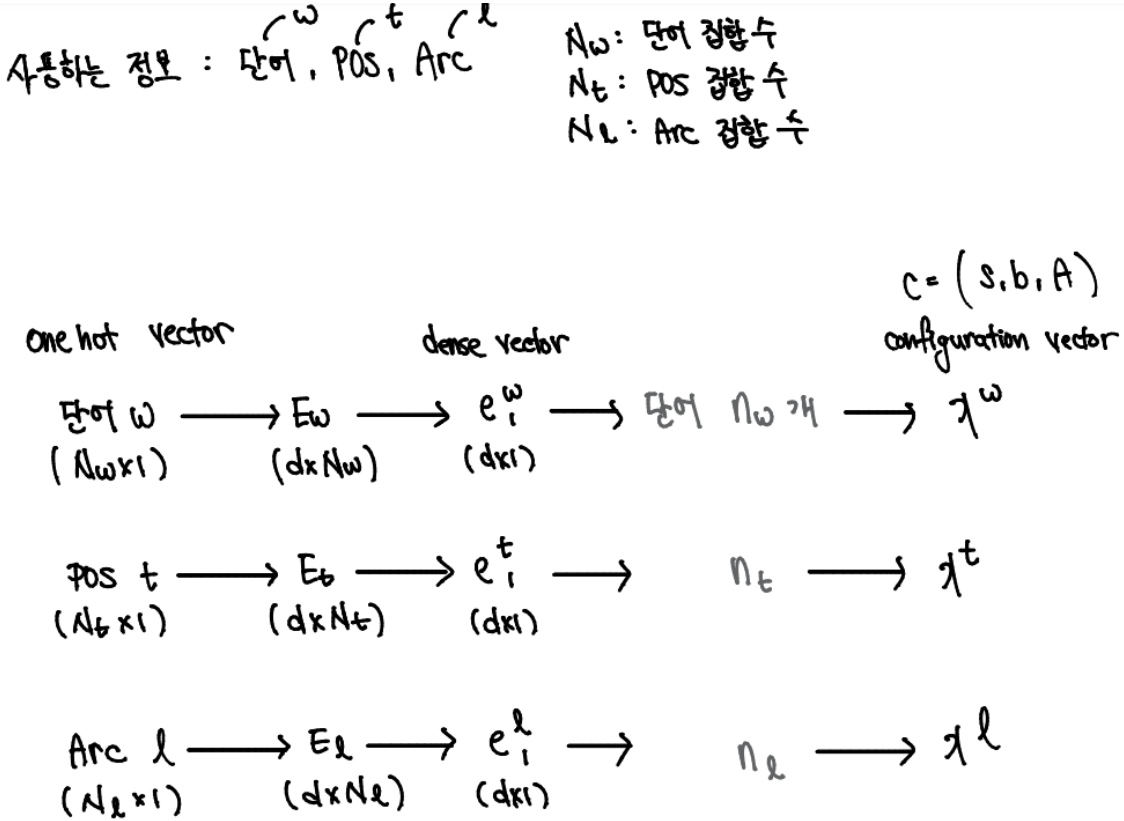
$s_2.t \circ s_1.t \circ b_1.t$; $s_2.t \circ s_1.t \circ lc_1(s_1).t$;
$s_2.t \circ s_1.t \circ rc_1(s_1).t$; $s_2.t \circ s_1.t \circ lc_1(s_2).t$;
$s_2.t \circ s_1.t \circ rc_1(s_2).t$; $s_2.t \circ s_1.w \circ rc_1(s_2).t$;
$s_2.t \circ s_1.w \circ lc_1(s_1).t$; $s_2.t \circ s_1.w \circ b_1.t$

02 연구 가정

Distributed word representation 효과: syntactic 정보뿐만 아니라 semantic 정보, semantic relationship도 담겨서 analogy 문제 (man : king :: woman : ?) 해결할 수 있었다.

→ **sparse**함을 해결하기 위해서 단어, 그리고 dependency parsing을 하기 위해 필요한 POS, Arc label들을 **dense**하게 distributed word representation으로 표현하여 Neural network으로 학습하면 문장의 구문 구조도 추론할 수 있을 것이다.

03 모델 설명: Input layer 구성



모델 설명

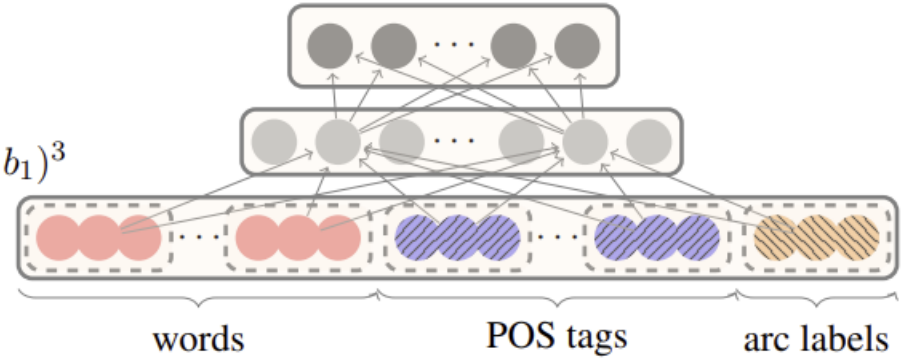
Softmax layer:

$$p = \texttt{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

Input layer: $[x^w, x^t, x^l]$



Softmax 결과: transition 각각의 확률

모델 특징

POS and arc label embedding

Word embedding처럼
POS, arc의 Semantic 정보가
들어갈 것이라고 판단

Cube Activation function

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

$(a+b+c)^3=$
$a^3+b^3+c^3+6abc+3ab(a+b)+3$
$ac(a+c)+3bc(b+c)$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Word choice

The top 3 words on the stack and buffer

The first and second leftmost / rightmost children of the top two words on the stack

The leftmost of leftmost / rightmost of rightmost children of the top two words on the stack

모델 특징: Training

Cost Function

$$L(\theta) = -\sum_i \log p_{t_i} + \frac{\lambda}{2}\|\theta\|^2$$

Initalization of Embedding matrix

POS, Arc label Embedding matrix
(−0.01, 0.01) random uniform intialization
Word Embedding matrix
Pretrained word embedding

Optimization

Adagrad
dropout

Dataset

Dataset	#Train	#Dev	#Test	#words (N_w)	#POS (N_t)	#labels (N_l)	projective (%)
PTB: CD	39,832	1,700	2,416	44,352	45	17	99.4
PTB: SD	39,832	1,700	2,416	44,389	45	45	99.9
CTB	16,091	803	1,910	34,577	35	12	100.0

04 실험 방법

Embedding size d =50
Hidden layer size = 200
Regularization λ = 10^−8
Learning rate = α = 0.01.

비교 대상

1. Feature template을 이용한 Standard parser, Eager parser
2. Malt Parser: Transition based dependency parser (stackproj, nivreeager)
3. MST Parser: Graph based dependency parser

05 실험 결과

UAS: Dependency tree에서 각 단어들의 부모를 맞춘 정도
LAS: Dependency tree에서 각 단어들의 부모 + 의존 관계까지 맞춘 정도

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	89.9	88.7	89.7	88.3	51
eager	90.3	89.2	89.9	88.6	63
Malt:sp	90.0	88.8	89.9	88.5	560
Malt:eager	90.1	88.9	90.1	88.7	535
MSTParser	92.1	90.8	**92.0**	90.5	12
Our parser	**92.2**	**91.0**	**92.0**	**90.7**	**1013**

Table 4: Accuracy and parsing speed on PTB + CoNLL dependencies.

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	90.2	87.8	89.4	87.3	26
eager	89.8	87.4	89.6	87.4	34
Malt:sp	89.8	87.2	89.3	86.9	469
Malt:eager	89.6	86.9	89.4	86.8	448
MSTParser	91.4	88.1	90.7	87.6	10
Our parser	**92.0**	**89.7**	**91.8**	**89.6**	**654**

Table 5: Accuracy and parsing speed on PTB + Stanford dependencies.

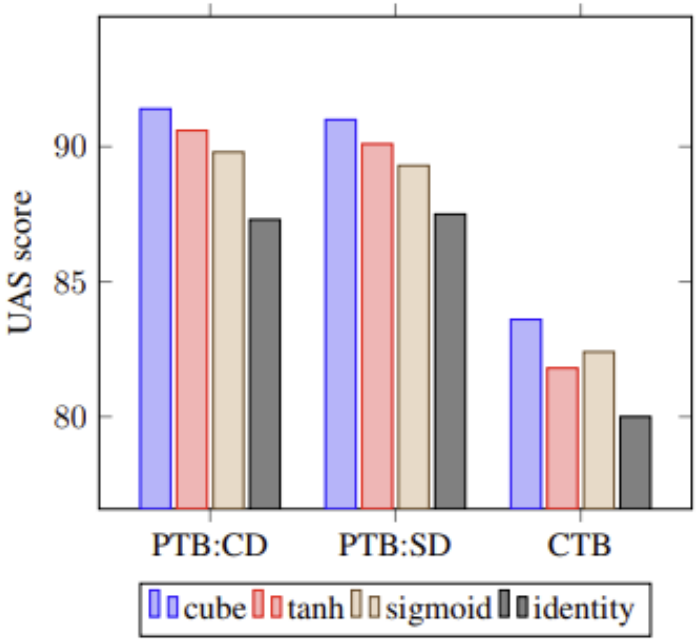
05 실험 결과

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	82.4	80.9	82.7	81.2	72
eager	81.1	79.7	80.3	78.7	80
Malt:sp	82.4	80.5	82.4	80.6	420
Malt:eager	81.2	79.3	80.2	78.4	393
MSTParser	**84.0**	82.1	83.0	81.2	6
Our parser	84.0	**82.4**	**83.9**	**82.4**	**936**

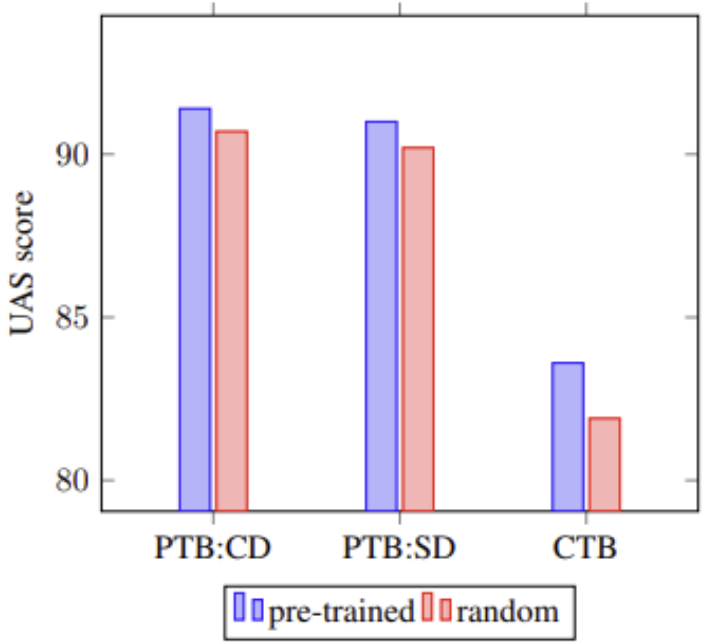
Table 6: Accuracy and parsing speed on CTB.

06 결과 분석

Activation function 비교

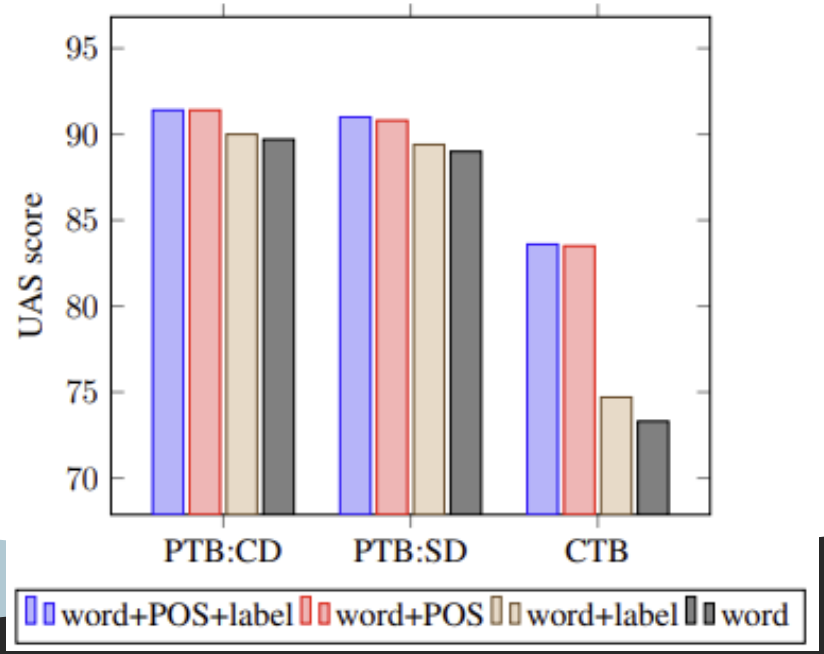


Pretrained word embedding 효과



06 결과 분석

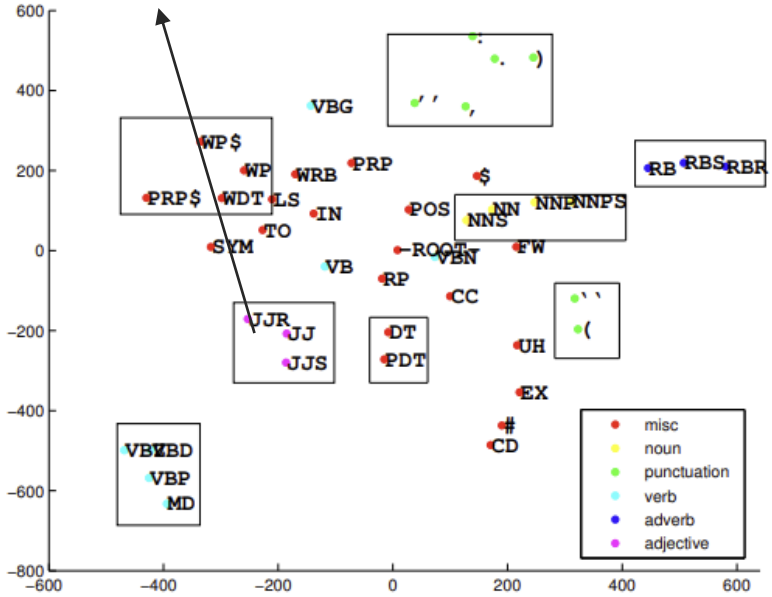
Dependency parsing에 필요한 정보



JJ: big
JJR: bigger
JJS: biggest

06 결과 분석

POS tag, Arc label embedding 결과



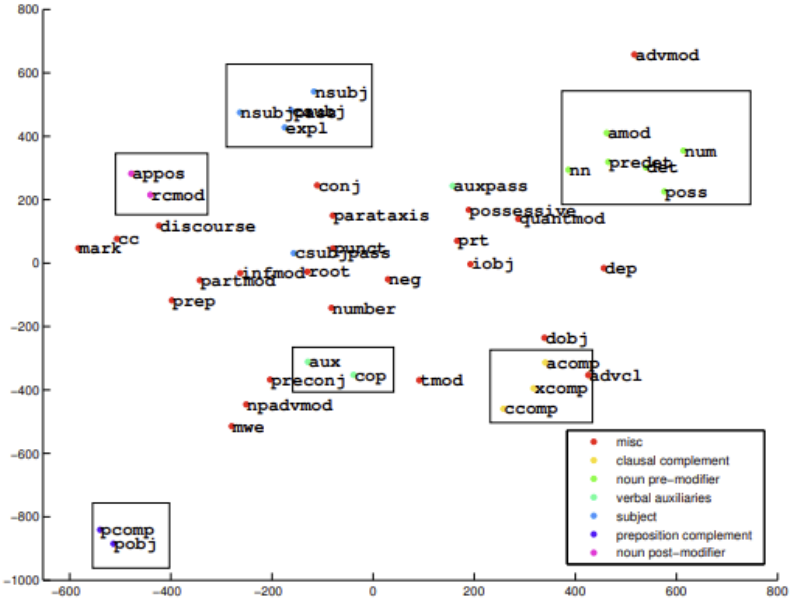
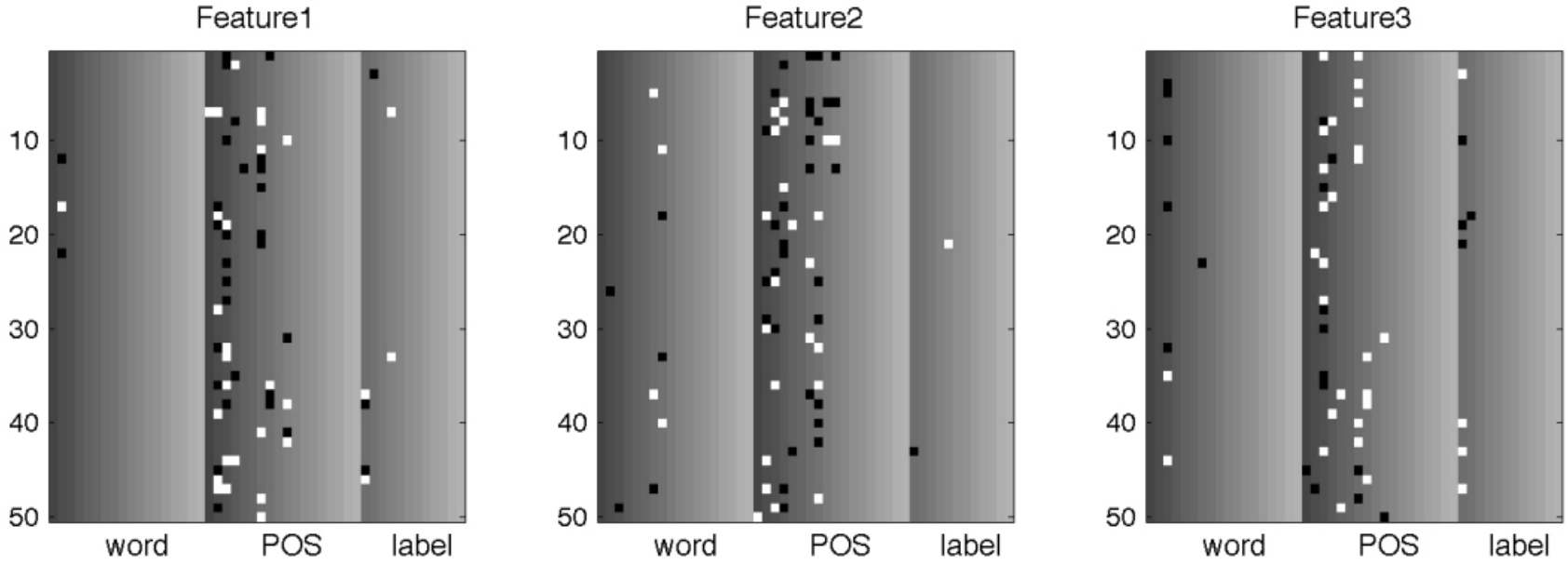


Figure 5: t-SNE visualization of POS and label embeddings

06 결과 분석

Neural Network Learning 효과: Hidden layer 분석 W_1^w, W_1^t, W_1^l



06 결과 분석

Neural Network Learning 효과

영향력 제일 큰 features 뽑아낼 수 있다

전혀 다른 시각으로 데이터 학습

- Feature 1: $s_1.t, s_2.t, lc(s_1).t.$
- Feautre 2: $rc(s_1).t, s_1.t, b_1.t.$
- Feature 3: $s_1.t, s_1.w, lc(s_1).t, lc(s_1).l.$

↓

기존의 feature template에 없는 조합

Single-word features (9)
$s_1.w$; $s_1.t$; $s_1.wt$; $s_2.w$; $s_2.t$;
$s_2.wt$; $b_1.w$; $b_1.t$; $b_1.wt$
Word-pair features (8)
$s_1.wt \circ s_2.wt$; $s_1.wt \circ s_2.w$; $s_1.wts_2.t$;
$s_1.w \circ s_2.wt$; $s_1.t \circ s_2.wt$; $s_1.w \circ s_2.w$
$s_1.t \circ s_2.t$; $s_1.t \circ b_1.t$
Three-word feaures (8)
$s_2.t \circ s_1.t \circ b_1.t$; $s_2.t \circ s_1.t \circ lc_1(s_1).t$;
$s_2.t \circ s_1.t \circ rc_1(s_1).t$; $s_2.t \circ s_1.t \circ lc_1(s_2).t$;
$s_2.t \circ s_1.t \circ rc_1(s_2).t$; $s_2.t \circ s_1.w \circ rc_1(s_2).t$;
$s_2.t \circ s_1.w \circ lc_1(s_1).t$; $s_2.t \circ s_1.w \circ b_1.t$

배운 점

단어 뿐만 아니라 text로 된 어떤 정보든 다 embedding 한다면 유의미한 정보를 얻을 수 있다.

Activation function을 tanh, sigmoid, ReLU 등으로만 생각할 것이 아니라 목적에 따라 직접 정의하면 좋은 결과를 얻을 수 있다.