



LangCon 2019



의존성 분석과 파싱 알고리즘

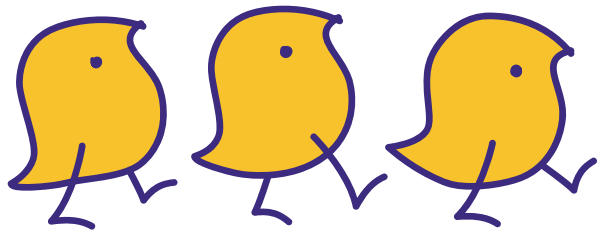
복잡복작스핀 심상진

1. 의존성 분석이란?

1. 의존성 분석의 정의
2. 자연어 처리에서 의존성 분석의 필요성

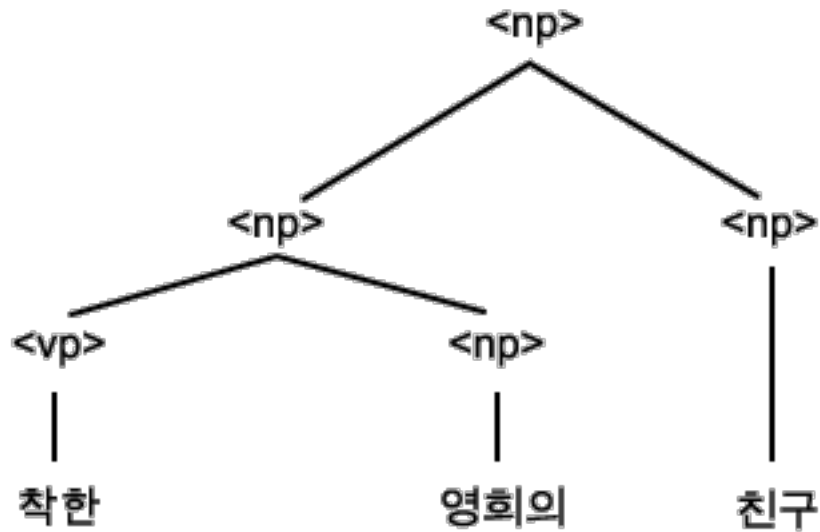
2. 의존성 분석을 위한 알고리즘

1. Transition-based Dependency Parsing Algorithm
2. Graph-based Dependency Parsing Algorithm
3. 한국어에서 의존성 분석 상황

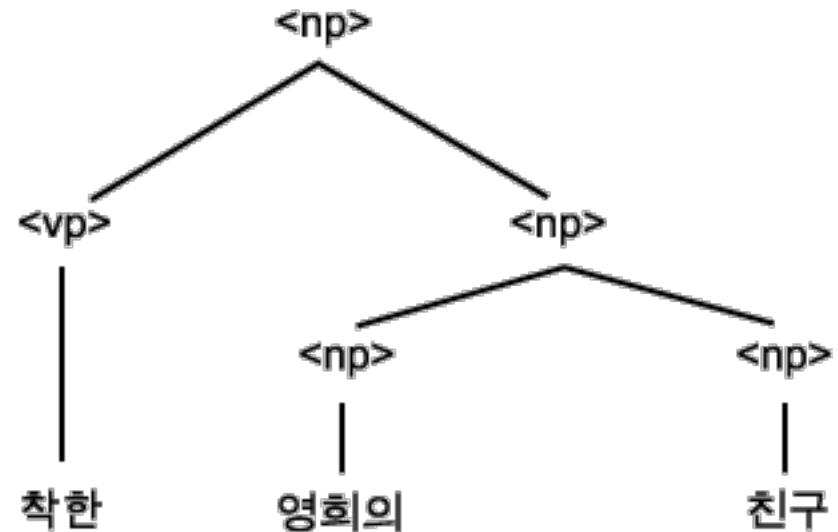


의존성 분석이란?

- 문장의 문법적 구조를 파악하여 각 단어별 관계성을 찾는 방법



(a)



(b)

- 의존성 분석을 하는 이유 : 문장의 구조적 모호성을 해결하기 위해서 하고, 이를 통해서 다음의 것들을 하기 위함

Scientists study whales from space



A diagram illustrating word dependencies for the sentence "Scientists study whales from space". The words are arranged horizontally. A red arc connects the word "study" to the word "space". Three black arcs connect "Scientists" to "study", "study" to "whales", and "whales" to "from".

Scientists study whales from space



A diagram illustrating word dependencies for the sentence "Scientists study whales from space". The words are arranged horizontally. A red arc connects the word "from" to the word "space". Three black arcs connect "Scientists" to "study", "study" to "whales", and "whales" to "from".

Reference: https://github.com/YBIGTA/Deep_learning/blob/master/RNN/cs224n/11%7C/cs224n_Lecture6-1.md

- 프로그래밍 언어(자연어보다 훨씬 쉬운 문법 규칙을 가짐)에서도 의존성 분석을 함.
 1. 사용되지 않은 코드(또는 변수)를 찾기
 2. 속도 최적화를 위한 정보 얻기(멀티 코어에서 명령어 실행 순서 배치)
 3. 코드 정적 분석을 위한 정보 얻기
 4. 등등...
- 하지만 여기서는 자연어 처리를 대상으로 하니까...

- Named-entity recognition (NER) is **the process of locating and classifying named entities** in a textual data into **predefined categories** such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

Ex) *Donald Trump will be visiting New Delhi next summer for a conference at Google.*

Ex) New York is one of the largest cities.

- **Coreference Resolution or Anaphora Resolution** is the task of finding all expressions that refer to the same entity in a text.

Ex) “*I* voted for *Nader* because *he* was most aligned with *my* values,”
she said.

I, *my* -> *she*
he -> *Nader*

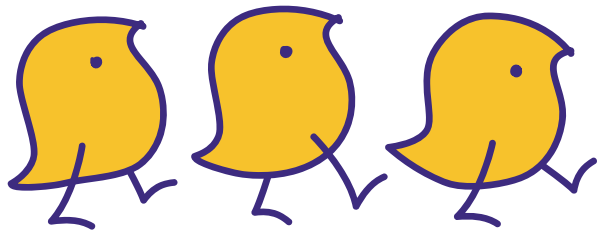
- **Question Answering** systems based on computational linguistics uses the syntactic structures of the query questions and matches them with the responses having similar syntactic structures. The similar syntactic structures contribute the answer set to a particular question.

Ex) Q: What is the capital of India?

A: New Delhi is the capital of India

- Both the question and answer dependency trees have similar patterns and can be used to generate the answer responses to specific queries.

Reference : <https://www.analyticsvidhya.com/blog/2017/12/introduction-computational-linguistics-dependency-trees/>



의존성 분석 알고리즘들

- **비결정적 의존 구문 분석**(non-deterministic dependency parsing)은 문장이 가질 수 있는 모든 의존트리(dependency tree)중에서 가장 높은 점수의 의존트리를 선택하는 방법 : **그래프 기반 의존 구문 분석**(graph-based dependency parsing)
- **결정적 의존 구문 분석**(deterministic dependency parsing)은 **일종의 탐욕적 알고리즘**(greedy algorithm)에 기반한 방법으로 지역적 학습 모델(locally training model)을 사용 : **전이기반 의존 구문 분석**(transition-based dependency parsing)
- 결정적 의존 구문 분석은 비결정적 의존구문 분석보다 더 많은 문맥자질을 사용할 수 있고, 근거리 의존관계를 찾는 데 강하며, 속도가 빠른 장점을 가지고 있다. 하지만 잘못 결정할 경우 오류의 전파가 발생하는 단점을 가짐

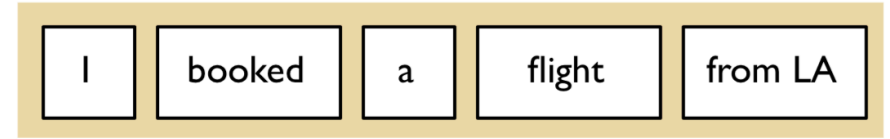
- Compiler의 LR방식과 유사
 - Shift : buffer에 있는 제일 앞에 있는 word 하나를 stack으로 옮기는 행동(action)
 - Reduce : stack에 있는 단어에 대해서 문법 규칙을 적용하고, 하나의 word만 남기는 행동(action)
 - Transition-based Dependency Parsing에서 Reduce는 Left Reduce(left-arc)와 Right Reduce(right-arc)의 두 종류가 있음
- Buffer : 토큰나이징한 단어들 중에 파싱을 기다리는 단어들을 담고 있는 공간
- Stack : 토큰나이징한 단어들 중에 파싱 작업을 진행 중인 단어들을 담고 있는 공간
- Parsing은 Buffer에 남아있는게 없으며, Stack에 하나의 Word만 남아있는 경우 끝남

Transition-based Dependency Parsing

Stack



Buffer



I

booked

a

flight

from LA

- Shift

Stack



Buffer



I

booked

a

flight

from LA

- Shift

Stack



Buffer



I

booked

a

flight

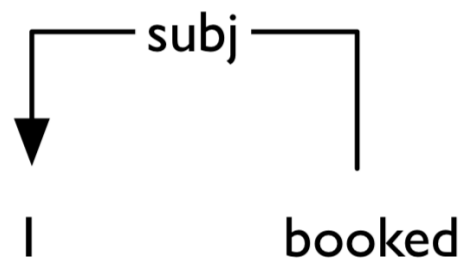
from LA

- Left-Reduce

Stack



Buffer



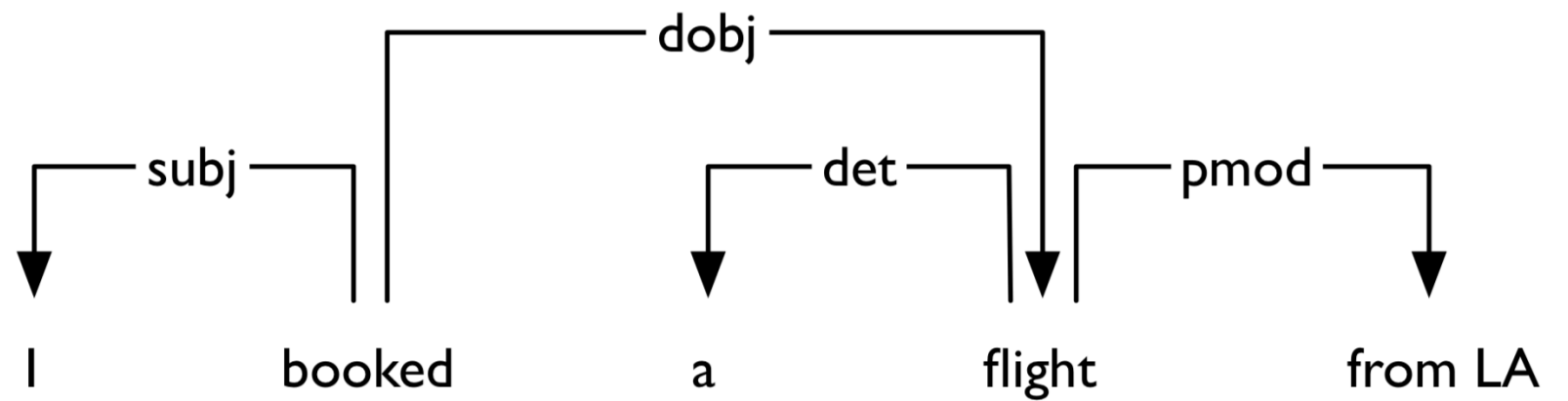
a flight from LA

- Final State

Stack



Buffer

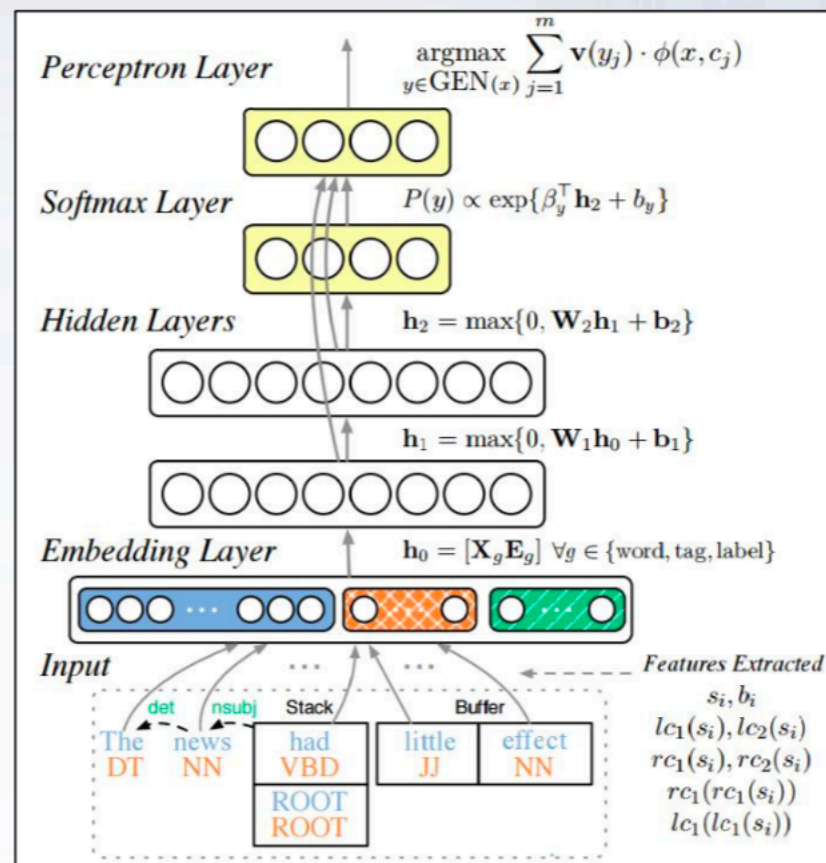


- Time complexity is linear, $O(n)$, since we only have to treat each word once
- This can be achieved since the algorithm is greedy, and only builds one tree, in contrast to Eisner's algorithm, where all trees are explored
- There is no guarantee that we will even find the best tree given the model, with the arc-standard model
- There is a risk of error propagation
- An advantage is that we can use very informative features, for the ML algorithm

- Transition based parsing
 - [Google] SyntaxNet model

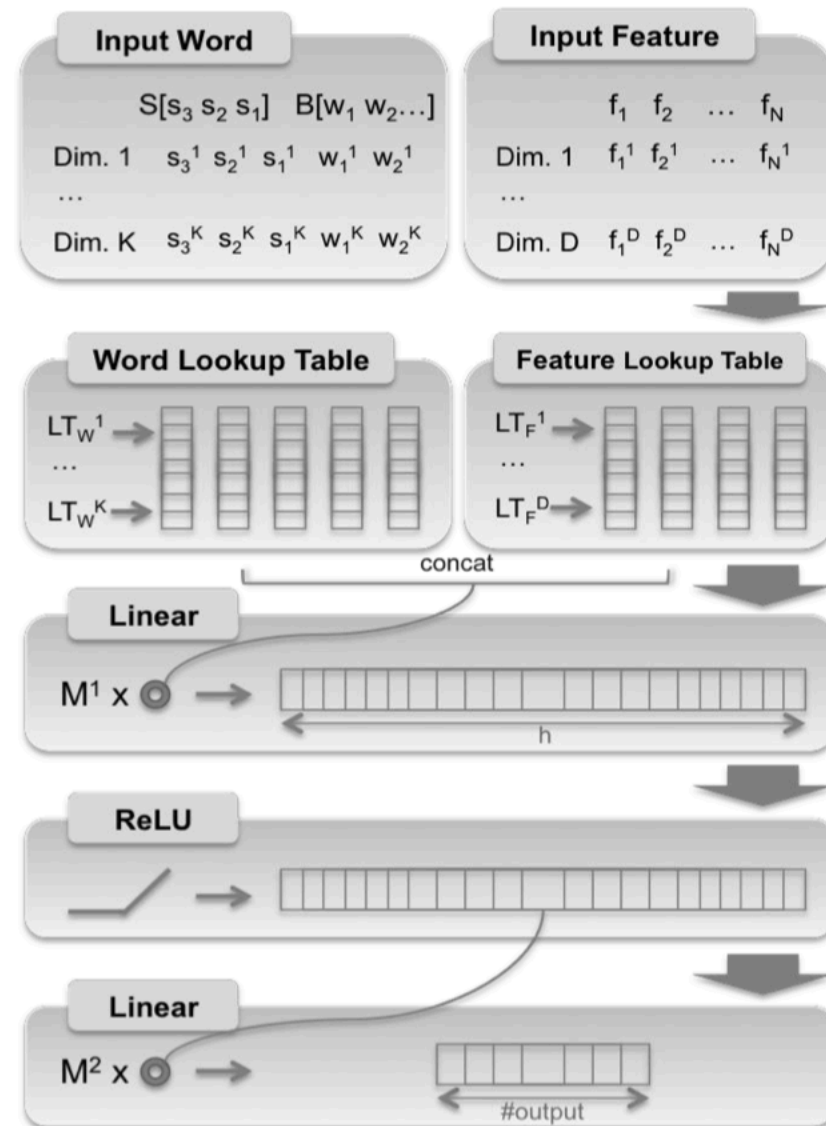
<i>Start state</i>	$([ROOT], [1, \dots, n], \{ \})$
<i>Final state</i>	$(S, [], A)$
Transitions	
Left – Arc_l	$:= (\sigma i, j \beta, A) \Rightarrow (\sigma, j \beta, A \cup \{(j, l, i)\})$
Right – Arc_r^s	$:= (\sigma i, j \beta, A) \Rightarrow (\sigma, i \beta, A \cup \{(i, l, j)\})$
Shift	$:= (\sigma, i \beta, A) \Rightarrow (\sigma i, \beta, A)$
Preconditions	
Left – Arc_l	$\neg [i = ROOT]$ $\neg \exists k \exists l' [(k, l', i) \in A]$
Right – Arc_r^s	$\neg \exists k \exists l' [(k, l', j) \in A]$

Arc-standard Transition



딥러닝 기반 한국어 의존구문분석

- Transition-based + Backward
 - $O(N)$
 - 세종코퍼스 → 의존 구문 변환
 - 보조용언/의사보조용언 후처리
- Deep Learning 기반
 - ReLU(> Sigmoid) + Dropout
 - Korean Word Embedding
 - NNLM, Ranking(hinge, logit)
 - Word2Vec
 - Feature Embedding
 - POS (stack + buffer)
 - 자동 분석(오류 포함)
 - Dependency Label (stack)
 - Distance information
 - Valency information
 - Mutual Information
 - 대용량 코퍼스 → 자동 구문 분석

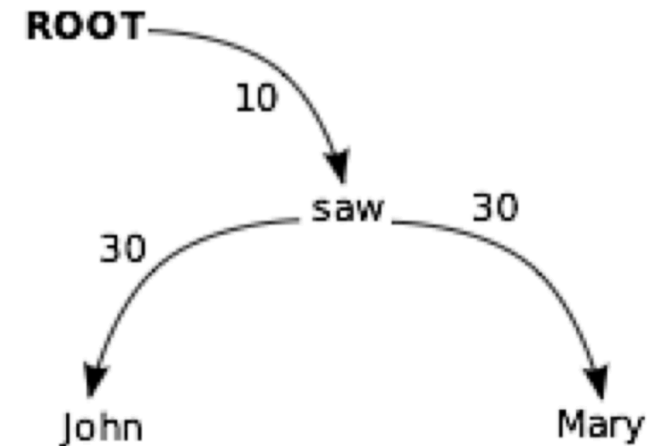
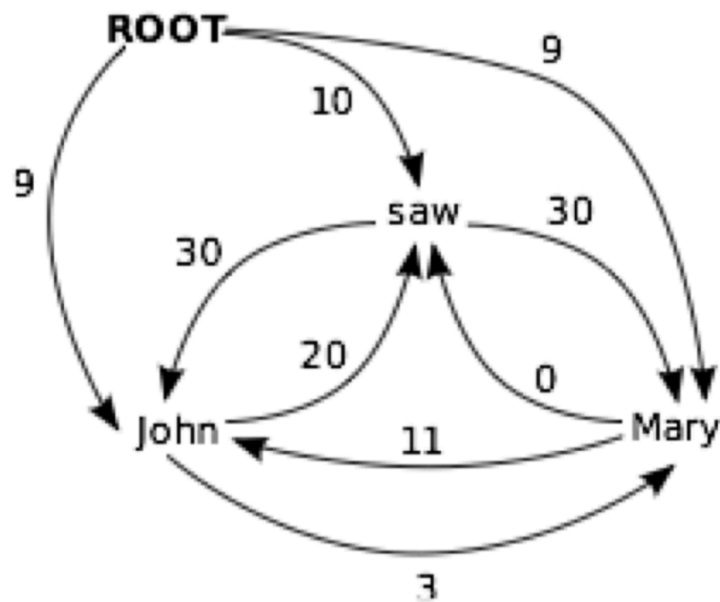


- Transition-based Dependency Parsing에서 전이방식을 결정(어떻게 묶는게 좋은지)하는 부분을 Deep Learning 기반 기계학습 기법을 이용하는 경우가 연구되고 있는 것 같음

- 오른쪽 테이블은 한국어에 그것들을 적용한 결과들

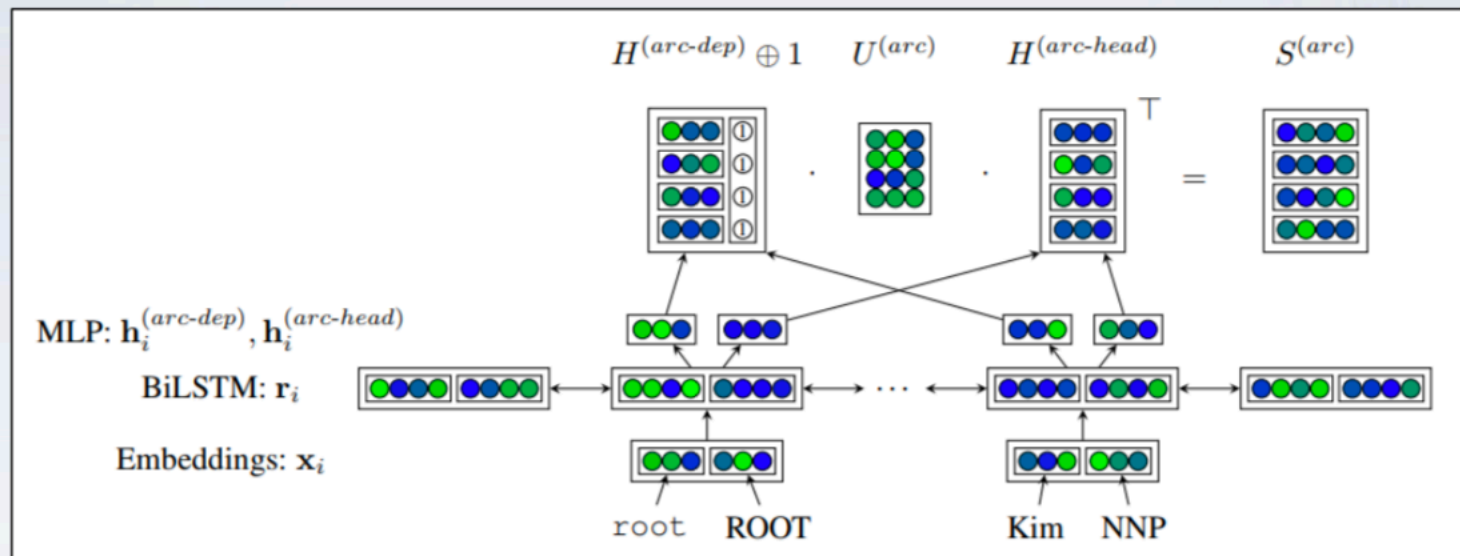
기존 의존 구문 분석	UAS	LAS
이용훈[15]: 국어정보베이스	88.42	-
오진영[16]: 세종코퍼스	87.03	-
임수종[17]: 세종코퍼스	88.15	-
J.D. Choi[18]: 세종코퍼스	85.47	83.47
박정열[19]: 세종코퍼스	86.43	-
안광모[20]: 세종코퍼스	87.52	-
딥 러닝+전이 기반 의존 구문 분석 (세종코퍼스, 자동 분석 형태소 이용)	UAS	LAS
ReLU+dropout	89.56	87.35
NNLM+ReLU+dropout	90.05	87.87
NNLM+ReLU+MI feat.	89.91	87.58
NNLM+ReLU+dropout+MI feat.	90.37	88.17
NNLM+sigmoid+MI feat.	89.94	87.64
NNLM+sigmoid+dropout+MI feat.	90.27	88.03
Ranking(hinge loss)+ReLU+dropout +MI feat.	90.19	88.01
Ranking(logit loss)+ReLU+dropout +MI feat.	90.31	88.11
Word2vec+ReLU+dropout+MI feat.	90.27	87.97

- Dependency structure를 Graph(directed Tree)로 표현
 - Vertex: nodes (w_i , i 번째 단어) 집합
 - Edge: arcs (w_i, w_j, l), l 은 label: $w_i \rightarrow w_j$



- Maximum Spanning Tree - 전형적인 Graph문제
 - Kruskal's algorithm
 - Prim's algorithm
 - 등등
- 근데 Edge의 Weight는?
 - Training Set을 이용하여 통계적인 방법 또는 Machine Learning 방법(SVM, DNN등)을 이용하여 계산할 수 있음
- Transition-Based Dependency Parsing에서와는 다르게 전역적 검색방법
- 그러므로 속도가 훨씬 느림($O(n^2)$ 또는 $O(n^3)$)

- Graph based parsing
 - [Stanford] Bi-affine attention model for dependency parsing



Model	Recurrent Cell		
	UAS	LAS	Sents/sec
LSTM	95.75	94.22	410.91
GRU	93.18*	91.08*	435.32
Cif-LSTM	95.67	94.06*	463.25

Table 2: Test accuracy and speed on PTB-SD 3.5.0. Statistically significant differences are marked with an asterisk.