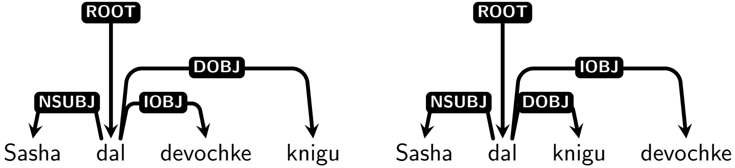
Phrase structure vs dependencies

- In languages with **free word order**, phrase structure (constituency) grammars don't make as much sense.
  - E.g., we would need both $\mathtt{S} \rightarrow \mathtt{NP\ VP}$ and $\mathtt{S} \rightarrow \mathtt{VP\ NP}$, etc. Not very informative about what's really going on.

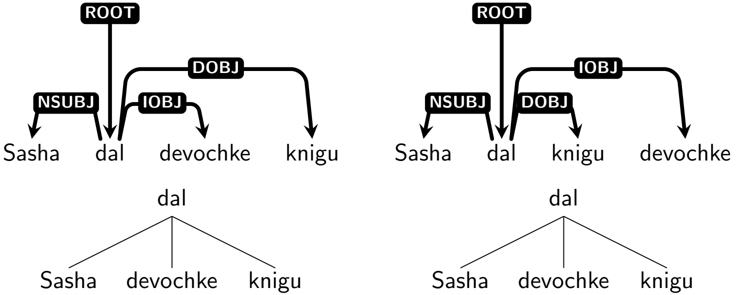
Phrase structure vs dependencies

- In languages with **free word order**, phrase structure (constituency) grammars don't make as much sense.
  - E.g., we would need both $\texttt{S} \rightarrow \texttt{NP VP}$ and $\texttt{S} \rightarrow \texttt{VP NP}$, etc. Not very informative about what's really going on.
- In contrast, the dependency relations stay constant:



Phrase structure vs dependencies

- Even more obvious if we just look at the trees without word order:



Pros and cons

- Sensible framework for free word order languages.
- Identifies syntactic relations directly. (using CFG, how would you identify the subject of a sentence?)
- Dependency pairs/chains can make good features in classifiers, for information extraction, etc.
- Parsers can be very fast (coming up...)

But

- The assumption of asymmetric binary relations isn't always right... e.g., how to parse dogs and cats?

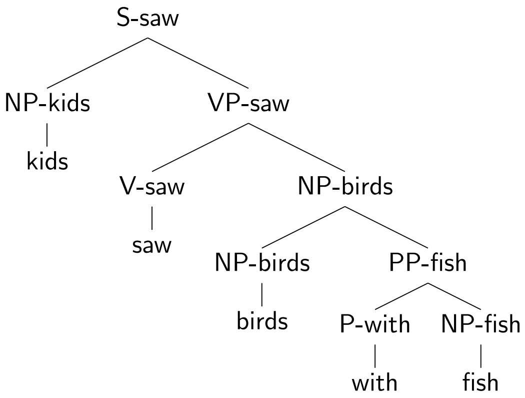
How do we annotate dependencies?

Two options:

1. Annotate dependencies directly.
2. Convert phrase structure annotations to dependencies. (Convenient if we already have a phrase structure treebank.)

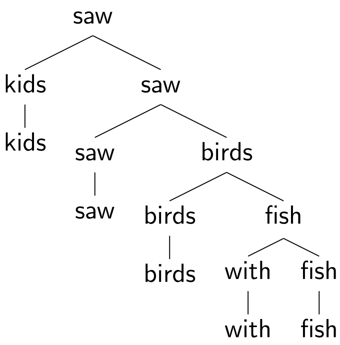
Next slides show how to convert, assuming we have head-finding rules for our phrase structure trees.

Lexicalized Constituency Parse

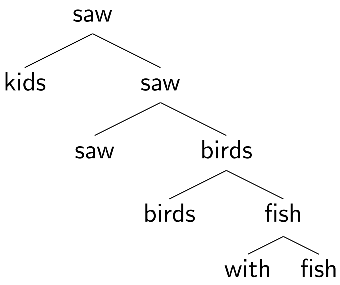


Shay Cohen, Accelerated Natural Language Processing (Slides)

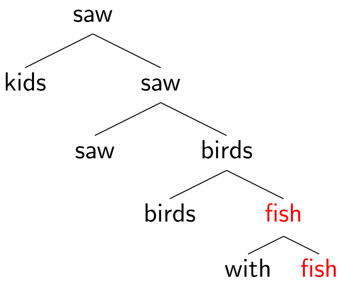
...remove the phrasal categories...



. . . remove the (duplicated) terminals. . .

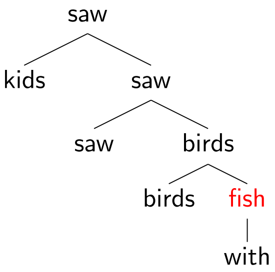


. . . and collapse chains of duplicates. . .

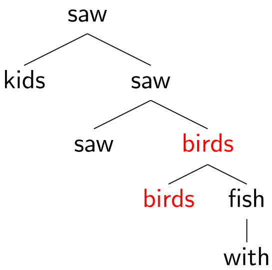


Shay Cohen, Accelerated Natural Language Processing (Slides)

...and collapse chains of duplicates...

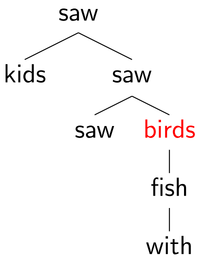


. . . and collapse chains of duplicates. . .



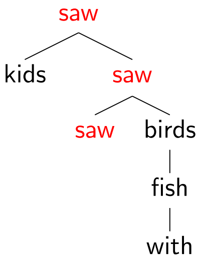
Shay Cohen, Accelerated Natural Language Processing (Slides)

. . . and collapse chains of duplicates. . .

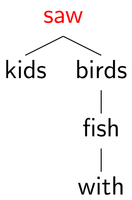


Shay Cohen, Accelerated Natural Language Processing (Slides)

. . . and collapse chains of duplicates. . .



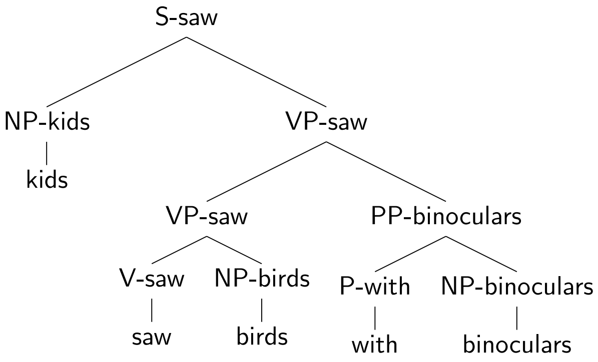
. . . done!



Shay Cohen, Accelerated Natural Language Processing (Slides)

Constituency Tree → Dependency Tree

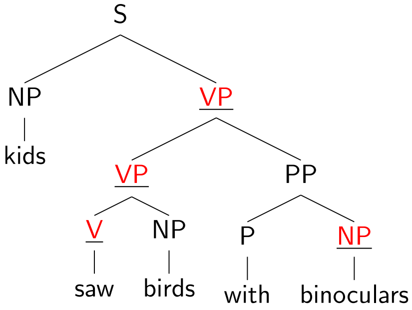
We saw how the **lexical head** of the phrase can be used to collapse down to a dependency tree:



- But how can we find each phrase's head in the first place?

Head Rules

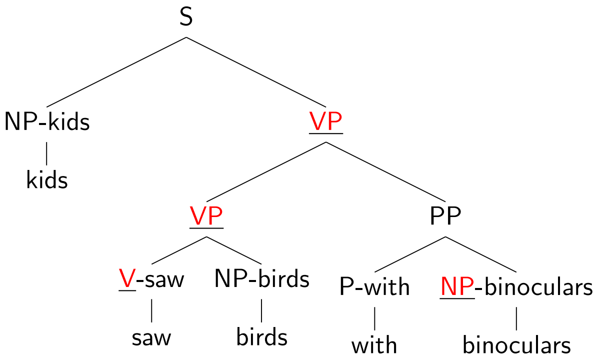
The standard solution is to use **head rules**: for every non-unary (P)CFG production, designate one RHS nonterminal as containing the head. S → NP <u>VP</u>, VP → <u>VP</u> PP, PP → P <u>NP</u> (content head), etc.



- Heuristics to scale this to large grammars: e.g., within an NP, last immediate N child is the head.

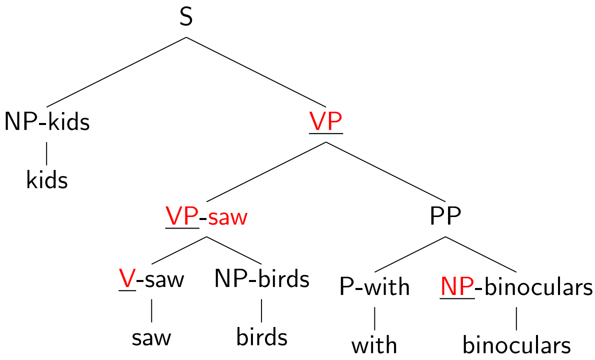
Head Rules

Then, propagate heads up the tree:



Head Rules

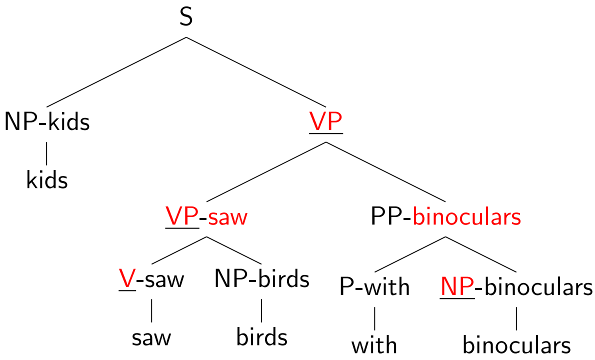
Then, propagate heads up the tree:



Shay Cohen, Accelerated Natural Language Processing (Slides)

Head Rules

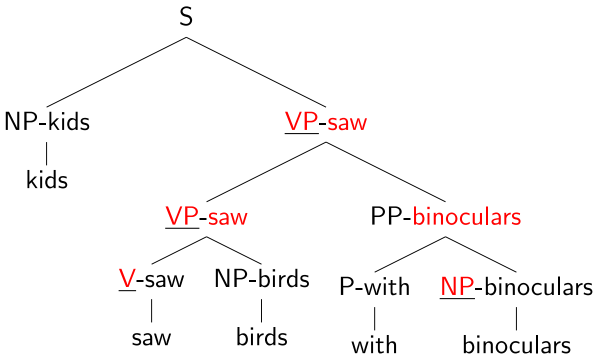
Then, propagate heads up the tree:



Shay Cohen, Accelerated Natural Language Processing (Slides)

Head Rules

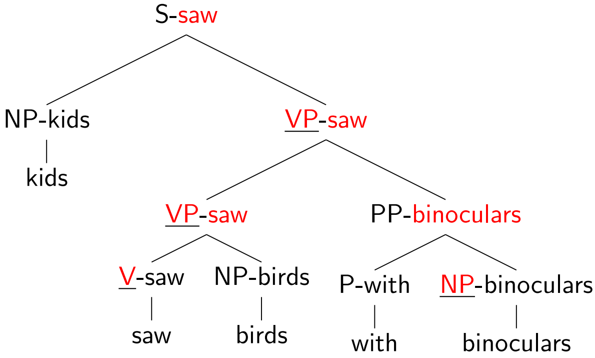
Then, propagate heads up the tree:



Shay Cohen, Accelerated Natural Language Processing (Slides)

Head Rules

Then, propagate heads up the tree:

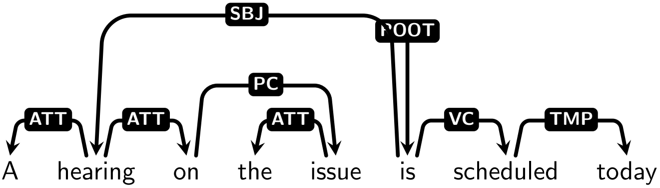


Shay Cohen, Accelerated Natural Language Processing (Slides)

Projectivity

If we convert constituency parses to dependencies, all the resulting trees will be **projective**.

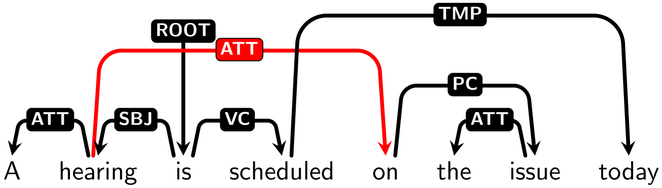
- Every subtree (node and all its descendants) occupies a *contiguous span* of the sentence.
- = the parse can be drawn over the sentence w/ no crossing edges.



Shay Cohen, Accelerated Natural Language Processing (Slides)

Nonprojectivity

But some sentences are **nonprojective**.



- We'll only get these annotations right if we directly annotate the sentences (or correct the converted parses).
- Nonprojectivity is rare in English, but common in many languages.
- Nonprojectivity presents problems for parsing algorithms.

Dependency Parsing

Some of the algorithms you have seen for PCFGs can be adapted to dependency parsing.

- **CKY** can be adapted, though efficiency is a concern: obvious approach is $O(Gn^5)$; Eisner algorithm brings it down to $O(Gn^3)$
  - N. Smith's slides explaining the Eisner algorithm: `http://courses.cs.washington.edu/courses/cse517/16wi/slides/an-dep-slides.pdf`
- **Shift-reduce**: more efficient, doesn't even require a grammar!

Recall: shift-reduce parser with CFG

- Same example grammar and sentence.
- Operations:
  - Reduce (R)
  - Shift (S)
  - Backtrack to step n (Bn)
- Note that at 9 and 11 we skipped over backtracking to 7 and 5 respectively as there were actually no choices to be made at those points.

Step	Op.	Stack	Input
0			the dog bit
1	S	the	dog bit
2	R	DT	dog bit
3	S	DT dog	bit
4	R	DT V	bit
5	R	DT VP	bit
6	S	DT VP bit	
7	R	DT VP V	
8	R	DT VP VP	
9	B6	DT VP bit	
10	R	DT VP NN	
11	B4	DT V	bit
12	S	DT V bit	
13	R	DT V V	
14	R	DT V VP	
15	B3	DT dog	bit
16	R	DT NN	bit
17	R	NP	bit
...			

Transition-based Dependency Parsing

The **arc-standard** approach parses input sentence $w_1 \ldots w_N$ using two types of **reduce** actions (three actions altogether):

- **Shift:** Read next word w_i from input and push onto the stack.
- **LeftArc:** Assign head-dependent relation $s_2 \leftarrow s_1$; pop s_2
- **RightArc:** Assign head-dependent relation $s_2 \rightarrow s_1$; pop s_1

where s_1 and s_2 are the top and second item on the stack, respectively. (So, s_2 *preceded* s_1 in the input sentence.)

Example

Parsing Kim saw Sandy:

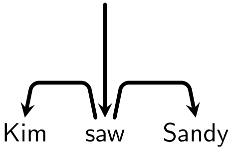
Step	←bot. Stacktop→	Word List	Action	Relations
0	[root]	[Kim,saw,Sandy]	Shift	
1	[root,Kim]	[saw,Sandy]	Shift	
2	[root,Kim,saw]	[Sandy]	LeftArc	Kim←saw
3	[root,saw]	[Sandy]	Shift	
4	[root,saw,Sandy]	[]	RightArc	saw→Sandy
5	[root,saw]	[]	RightArc	root→saw
6	[root]	[]	(done)	

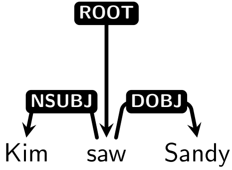
- Here, top two words on stack are also always adjacent in sentence. Not true in general! (See longer example in JM3.)

Shay Cohen, Accelerated Natural Language Processing (Slides)

Labelled dependency parsing

- These parsing actions produce **unlabelled** dependencies (left).
- For **labelled** dependencies (right), just use more actions: LeftArc(NSUBJ), RightArc(NSUBJ), LeftArc(DOBJ), ...





Shay Cohen, Accelerated Natural Language Processing (Slides)

Differences to constituency parsing

- Shift-reduce parser for CFG: not all sequences of actions lead to valid parses. Choose incorrect action $\rightarrow$ may need to backtrack.
- Here, all valid action sequences lead to valid parses.
  - Invalid actions: can't apply LeftArc with root as dependent; can't apply RightArc with root as head unless input is empty.
  - Other actions may lead to **incorrect** parses, but still **valid**.
- So, parser doesn't backtrack. Instead, tries to greedily predict the correct action at each step.
  - Therefore, dependency parsers can be very fast (linear time).
  - But need a good way to predict correct actions (next lecture).

Shay Cohen, Accelerated Natural Language Processing (Slides)

Notions of validity

- In constituency parsing, valid parse = grammatical parse.
  - That is, we first define a grammar, then use it for parsing.
- In dependency parsing, we don't normally define a grammar.
  - Valid parses are those with the properties on slide 4.

Summary: Transition-based Parsing

- **arc-standard** approach is based on simple shift-reduce idea.
- Can do labelled or unlabelled parsing, but need to train a **classifier** to predict next action, as we'll see next time.
- Greedy algorithm means time complexity is linear in sentence length.
- Only finds **projective** trees (without special extensions)
- Pioneering system: Nivre's MaltParser.

Alternative: Graph-based Parsing

- Global algorithm: From the fully connected directed graph of all possible edges, choose the best ones that form a tree.
- **Edge-factored** models: Classifier assigns a nonnegative score to each possible edge; **maximum spanning tree** algorithm finds the spanning tree with highest total score in $O(n^2)$ time.
- Pioneering work: McDonald's MSTParser
- Can be formulated as constraint-satisfaction with **integer linear programming** (Martins's TurboParser)
- Details in JM3, Ch 16.5 (optional).

Graph-based vs. Transition-based vs. Conversion-based

- TB: Features in scoring function can look at any part of the stack; no optimality guarantees for search; linear-time; (classically) projective only
- GB: Features in scoring function limited by factorization; optimal search within that model; quadratic-time; no projectivity constraint
- CB: In terms of accuracy, sometimes best to first constituency-parse, then convert to dependencies (e.g., STANFORD PARSER). Slower than direct methods.

Shay Cohen, Accelerated Natural Language Processing (Slides)

Choosing a Parser: Criteria

- Target representation: constituency or dependency?
- Efficiency? In practice, both runtime and memory use.
- Incrementality: parse the whole sentence at once, or obtain partial left-to-right analyses/expectations?
- Retrainable system?
- Accuracy?

Summary

- Constituency syntax: hierarchically nested phrases with categories like `NP`.
- Dependency syntax: trees whose edges connect words in the sentence. Edges often labeled with relations like `nsubj`.
- Can convert constituency to dependency parse using head rules.
- For projective trees, transition-based parsing is very fast and can be very accurate.
- Google "online dependency parser".
  Try out the Stanford parser and SEMAFOR!

Content

Parsing with neural networks

- Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks, EMNLP2014

  (transition-based dependency parser)
- Timothy Dozat, Peng Qi, and Chris Manning. Stanford's Graph-based Neural Dependency Parser at the CoNLL 2017 Shared Task. In CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, 2017.

  (graph-based dependency parser)

Parsing with neural networks

- Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks, EMNLP2014

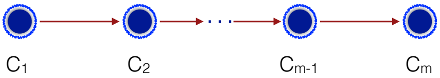
  (transition-based dependency parser)

- Timothy Dozat, Peng Qi, and Chris Manning. Stanford's Graph-based Neural Dependency Parser at the CoNLL 2017 Shared Task. In CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, 2017.

  (graph-based dependency parser)

(Nivre et al, 2004)

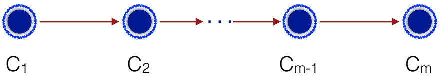
Greedy Transition-based Parsing



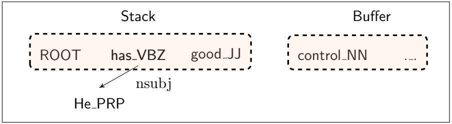
Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

(Nivre et al, 2004)

Greedy Transition-based Parsing



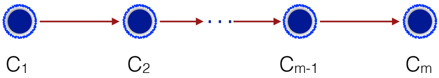
- A configuration = a stack, a buffer and some dependency arcs



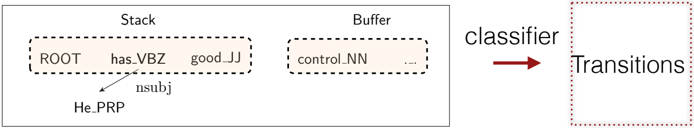
Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

(Nivre et al, 2004)

Greedy Transition-based Parsing



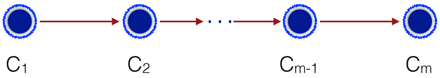
- A configuration = a stack, a buffer and some dependency arcs



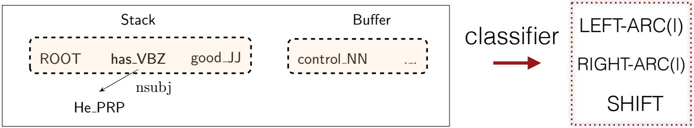
Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

(Nivre et al, 2004)

Greedy Transition-based Parsing



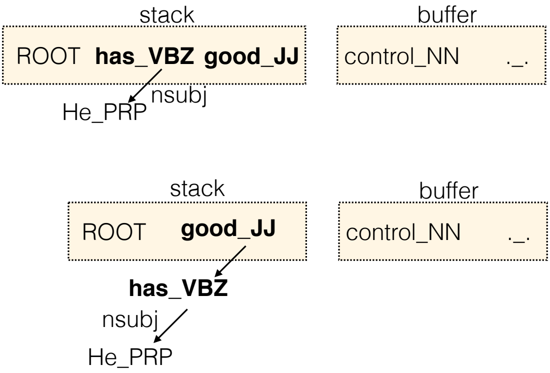
- A configuration = a stack, a buffer and some dependency arcs



- We employ the arc-standard system.

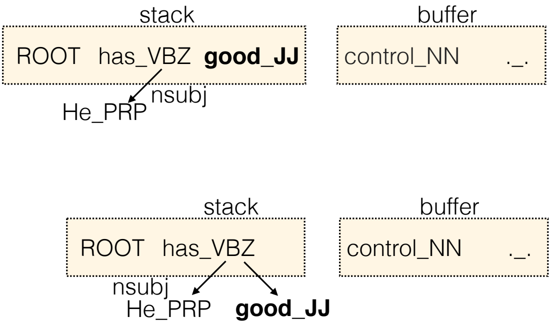
Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

LEFT-ARC (l)



Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

RIGHT-ARC (l)



Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

SHIFT

stack: ROOT has_VBZ good_JJ (He_PRP ←nsubj— has_VBZ)

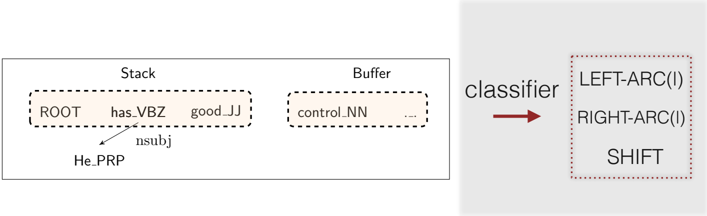
buffer: **control_NN** ._.

stack: ROOT has_VBZ good_JJ **control_NN** (He_PRP ←nsubj— has_VBZ)

buffer: ._.

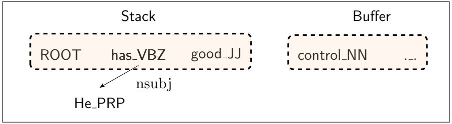
Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

Greedy Transition-based Parsing



Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

Traditional Features



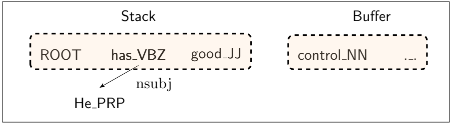
binary, sparse
dim =$10^6 \sim 10^7$

0	0	0	1	0	0	1	0	...	0	0	1	0

Feature templates: usually a combination of **1 ~ 3** elements from the configuration.

Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

Traditional Features



binary, sparse
dim $=10^6 \sim 10^7$

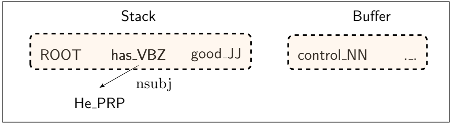
0	0	0	1	0	0	1	0	...	0	0	1	0

Indicator features

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

Traditional Features



binary, sparse
dim $=10^6 \sim 10^7$

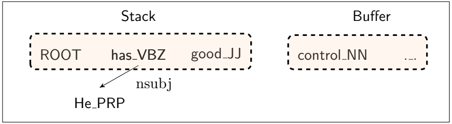
0	0	0	1	0	0	1	0	...	0	0	1	0

Indicator features

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)

Traditional Features



binary, sparse
dim $=10^6 \sim 10^7$

0	0	0	1	0	0	1	0	…	0	0	1	0

word   part-of-speech tag

Indicator features

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

dep. label

Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks (Slides)