CSC 439/539
Statistical Natural Language Processing
Lecture 8: Dependency Parsing

Mihai Surdeanu

Fall 2017

1

## Take-away

- Shift-reduce dependency parsing algorithms for projective trees
- Shift-reduce dependency parsing algorithms for directed acyclic graphs
- Tree LSTMs

2

## Readings

- Dan Jurafsky. Speech and Language Processing, **3rd edition**, chapter 14. https://web.stanford.edu/~jurafsky/slp3/14.pdf
- Danqi Chen and Christopher D. Manning. A Fast and Accurate Dependency Parser using Neural Networks. https://cs.stanford.edu/~danqi/papers/emnlp2014.pdf
- Kenji Sagae, Jun'ichi Tsuji. Shift-reduce Dependency DAG Parsing. https://aclweb.org/anthology/C/C08/C08-1095.pdf
- Mihai Surdeanu and Christopher D. Manning. Ensemble Models for Dependency Parsing: Cheap and Good? http://surdeanu.info/mihai/papers/naacl10-parsing.pdf
- Kai Sheng Tai, Richard Socher, Christopher D. Manning. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. https://arxiv.org/pdf/1503.00075.pdf

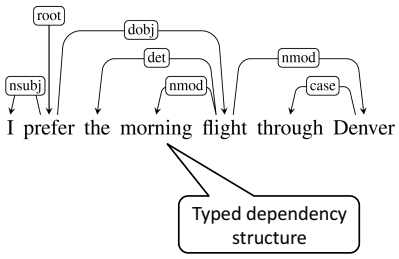
3

## Take-away

- **Shift-reduce dependency parsing algorithms for projective trees**
- Shift-reduce dependency parsing algorithms for directed acyclic graphs
- Tree LSTMs

4

## Remember this?




5

## Typed dependency structures

- A set of binary dependency relations for a sentence
  - Head: central organizing word of a larger constituent
  - Modifier: depends on the head word
  - Label: indicates the grammatical function captured

6

## Selected dependency relations from the Universal Dependency set

| **Clausal Argument Relations** | **Description** |
|---|---|
| NSUBJ | Nominal subject |
| DOBJ | Direct object |
| IOBJ | Indirect object |
| CCOMP | Clausal complement |
| XCOMP | Open clausal complement |
| **Nominal Modifier Relations** | **Description** |
| NMOD | Nominal modifier |
| AMOD | Adjectival modifier |
| NUMMOD | Numeric modifier |
| APPOS | Appositional modifier |
| DET | Determiner |
| CASE | Prepositions, postpositions and other case markers |
| **Other Notable Relations** | **Description** |
| CONJ | Conjunct |
| CC | Coordinating conjunction |

7

## Exercise: what are the dependency labels in these examples?

| Relation | Examples with *head* and **dependent** |
|---|---|
| | **United** *canceled* the flight. |
| | United *diverted* the **flight** to Reno. |
| | We *booked* her the first **flight** to Miami. |
| | We *booked* **her** the flight to Miami. |
| | We took the **morning** *flight*. |
| | Book the **cheapest** *flight*. |
| | Before the storm JetBlue canceled **1000** *flights*. |
| | *United*, a **unit** of UAL, matched the fares. |
| | **The** *flight* was canceled. |
| | **Which** *flight* was delayed? |
| | We *flew* to Denver and **drove** to Steamboat. |
| | We flew to Denver **and** *drove* to Steamboat. |
| | Book the flight **through** *Houston*. |

8

## Exercise: what are the dependency labels in these examples?

| Relation | Examples with *head* and **dependent** |
|---|---|
| NSUBJ | **United** *canceled* the flight. |
| DOBJ | United *diverted* the **flight** to Reno. |
| | We *booked* her the first **flight** to Miami. |
| IOBJ | We *booked* **her** the flight to Miami. |
| NMOD | We took the **morning** *flight*. |
| AMOD | Book the **cheapest** *flight*. |
| NUMMOD | Before the storm JetBlue canceled **1000** *flights*. |
| APPOS | *United*, a **unit** of UAL, matched the fares. |
| DET | **The** *flight* was canceled. |
| | **Which** *flight* was delayed? |
| CONJ | We *flew* to Denver and **drove** to Steamboat. |
| CC | We flew to Denver **and** *drove* to Steamboat. |
| CASE | Book the flight **through** *Houston*. |

9

## Exercise: what is the unlabeled dependency tree for this sentence?

I ate a bagel today with sesame seeds .

This example has a non-projective tree (crossing edges)!

10

## Practice Exercise

- Draw the dependency trees following universal dependencies for these sentences:
  - John Doe visited China.
  - I saw a girl with the telescope.
  - They told him a story.
  - A hearing is scheduled on the issue today.
  - JetBlue canceled our flight this morning which was already late.

The last 2 examples have non-projective trees!

11

## Properties of dependency trees

- There is a single designated root node that has no incoming arcs.
- With the exception of the root node, each word has exactly one incoming arc (one head).
- There is a unique path from the root node to each word in the sentence.

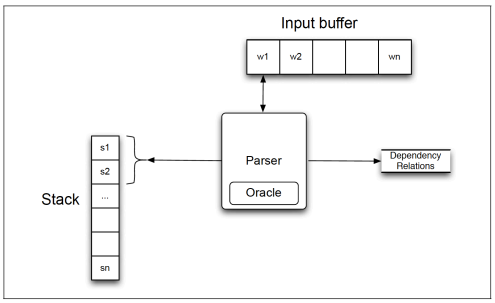
12

## Projectivity

- Not a theoretical requirement, but useful in practice because
  - Training data was generated from the original constituent-based treebanks resulting in projective trees
  - The simplest algorithms produce projective trees

13

## Shift-reduce (or transition-based) algorithms




14

## Arc-standard algorithm

- Possible actions:
  - LEFTARC: Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack.
  - RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack;
  - SHIFT: Remove the word from the front of the input buffer and push it onto the stack.

15

## Arc-standard algorithm

- Possible actions:
  - LEFTARC: Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack.
  - RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack;
  - SHIFT: Remove the word at the front of the input buffer and push it onto the stack.

- LeftArc and RightArc will be subclassed with the label of the dependency to be created, e.g., LeftArc-nsubj, LeftArc-dobj...
- How many classes will your multi-class classifier have?

16

## Generic transition-based dependency parser

Stack

Input buffer

**function** DEPENDENCY-PARSE(*words*) **returns** dependency tree

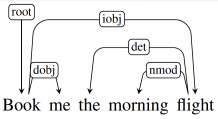
state ← {[root], [*words*], [] } ; initial configuration
**while** *state* **not final**
t ← ORACLE(*state*) ; choose a transition operator to apply
state ← APPLY(*t*, *state*) ; apply it, creating a new state
**return** *state*

Training: Correct action given gold data
Testing: Decision of a multi-class classifier

*State* is final when it contains a single tree anchored at *root*

17

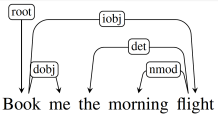
## Example: gold actions for arc-standard




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

18

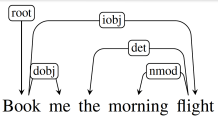
## Example: gold actions for arc-standard




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | SHIFT | |
| 4 | [root, book, the] | [morning, flight] | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

19

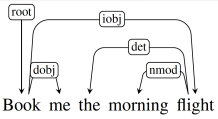
## Example: gold actions for arc-standard




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | SHIFT | |
| 4 | [root, book, the] | [morning, flight] | SHIFT | |
| 5 | [root, book, the, morning] | [flight] | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

20

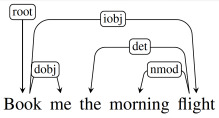
## Example: gold actions for arc-standard




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | SHIFT | |
| 4 | [root, book, the] | [morning, flight] | SHIFT | |
| 5 | [root, book, the, morning] | [flight] | SHIFT | |
| 6 | [root, book, the, morning, flight] | [] | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

21

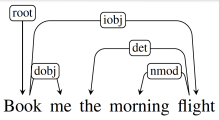
## Example: gold actions for arc-standard




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | SHIFT | |
| 4 | [root, book, the] | [morning, flight] | SHIFT | |
| 5 | [root, book, the, morning] | [flight] | SHIFT | |
| 6 | [root, book, the, morning, flight] | [] | LEFTARC | (morning ← flight) |
| 7 | [root, book, the, flight] | [] | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

22

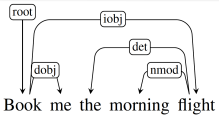
## Example: gold actions for arc-standard




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | SHIFT | |
| 4 | [root, book, the] | [morning, flight] | SHIFT | |
| 5 | [root, book, the, morning] | [flight] | SHIFT | |
| 6 | [root, book, the, morning, flight] | [] | LEFTARC | (morning ← flight) |
| 7 | [root, book, the, flight] | [] | LEFTARC | (the ← flight) |
| 8 | [root, book, flight] | [] | | |
| 9 | | | | |
| 10 | | | | |

23

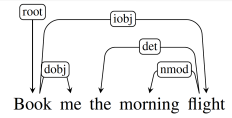
## Example: gold actions for arc-standard




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | SHIFT | |
| 4 | [root, book, the] | [morning, flight] | SHIFT | |
| 5 | [root, book, the, morning] | [flight] | SHIFT | |
| 6 | [root, book, the, morning, flight] | [] | LEFTARC | (morning ← flight) |
| 7 | [root, book, the, flight] | [] | LEFTARC | (the ← flight) |
| 8 | [root, book, flight] | [] | RIGHTARC | (book → flight) |
| 9 | [root, book] | [] | | |
| 10 | | | | |

24

## Example: gold actions for arc-standard




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | SHIFT | |
| 4 | [root, book, the] | [morning, flight] | SHIFT | |
| 5 | [root, book, the, morning] | [flight] | SHIFT | |
| 6 | [root, book, the, morning, flight] | [] | LEFTARC | (morning ← flight) |
| 7 | [root, book, the, flight] | [] | LEFTARC | (the ← flight) |
| 8 | [root, book, flight] | [] | RIGHTARC | (book → flight) |
| 9 | [root, book] | [] | RIGHTARC | (root → book) |
| 10 | [root] | [] | Done | |

How many actions for a sentence of length *N*?

25

## Practice Exercise

- Generate the program trace for the arc-standard algorithm for the sentences:
  - John Doe visited China.
  - I saw a girl with the telescope.
  - They told him a story.

26

## Generating training data: the oracle

- The training oracle generates actions for a given state as follows:
  - Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration,
  - Otherwise, choose RIGHTARC if (1) it produces a correct head-dependent relation given the reference parse and (2) all of the dependents of the word at the top of the stack have already been assigned,
  - Otherwise, choose SHIFT.

Why?

- Then train your favorite multi-class classifier on these actions!

27

## Features

- Generally from:
  - Top level(s) of the stack
  - Words near the front of the buffer
  - The dependency relations already created

28

## Features example

$$\langle s_1.w, op\rangle, \langle s_2.w, op\rangle \langle s_1.t, op\rangle, \langle s_2.t, op\rangle$$
$$\langle b_1.w, op\rangle, \langle b_1.t, op\rangle \langle s_1.wt, op\rangle$$

- *s* – stack
- *b* – input buffer
- *w* - word
- *t* – POS tag
- *op* – dependency relation

29

## Features example

| Stack | Word buffer | Relations |
|---|---|---|
| [root, canceled, flights] | [to Houston] | (canceled → United)<br>(flights → morning)<br>(flights → the) |

$$\langle s_1.w = flights, op = shift\rangle$$
$$\langle s_2.w = canceled, op = shift\rangle$$
$$\langle s_1.t = NNS, op = shift\rangle$$
$$\langle s_2.t = VBD, op = shift\rangle$$
$$\langle b_1.w = to, op = shift\rangle$$
$$\langle b_1.t = TO, op = shift\rangle$$
$$\langle s_1.wt = flightsNNS, op = shift\rangle$$

30

## Feature combinations

| Source | Feature templates | | |
|---|---|---|---|
| One word | $s_1.w$ | $s_1.t$ | $s_1.wt$ |
| | $s_2.w$ | $s_2.t$ | $s_2.wt$ |
| | $b_1.w$ | $b_1.w$ | $b_0.wt$ |
| Two word | $s_1.w \circ s_2.w$ | $s_1.t \circ s_2.t$ | $s_1.t \circ b_1.w$ |
| | $s_1.t \circ s_2.wt$ | $s_1.w \circ s_2.w \circ s_2.t$ | $s_1.w \circ s_1.t \circ s_2.t$ |
| | $s_1.w \circ s_1.t \circ s_2.t$ | $s_1.w \circ s_1.t$ | |

Concatenation operator

31

## Concrete example: features for arc-standard in maltparser

```
<?xml version="1.0" encoding="UTF-8"?>
<featuremodels>
   <featuremodel name="nivreeager0">
      <feature>InputColumn(POSTAG, Stack[0])</feature>
      <feature>InputColumn(POSTAG, Input[0])</feature>
      <feature>InputColumn(POSTAG, Input[1])</feature>
      <feature>InputColumn(POSTAG, Input[2])</feature>
      <feature>InputColumn(POSTAG, Input[3])</feature>
      <feature>InputColumn(POSTAG, Stack[1])</feature>
      <feature>Merge(InputColumn(POSTAG, Stack[0]), InputColumn(POSTAG, Input[0]))</feature>
      <feature>Merge3(InputColumn(POSTAG, Stack[1]), InputColumn(POSTAG, Stack[0]), InputColumn(POSTAG, Input[0]))</feature>
      <feature>Merge3(InputColumn(POSTAG, Stack[0]), InputColumn(POSTAG, Input[0]), InputColumn(POSTAG, Input[1]))</feature>
      <feature>Merge3(InputColumn(POSTAG, Input[0]), InputColumn(POSTAG, Input[1]), InputColumn(POSTAG, Input[2]))</feature>
      <feature>Merge3(InputColumn(POSTAG, Input[1]), InputColumn(POSTAG, Input[2]), InputColumn(POSTAG, Input[3]))</feature>
      <feature>OutputColumn(DEPREL, ldep(Stack[0]))</feature>
      <feature>OutputColumn(DEPREL, rdep(Stack[0]))</feature>
      <feature>OutputColumn(DEPREL, ldep(Input[0]))</feature>
      <feature>OutputColumn(DEPREL, rdep(Input[0]))</feature>
      <feature>Merge3(InputColumn(POSTAG, Stack[0]), OutputColumn(DEPREL, ldep(Stack[0])), OutputColumn(DEPREL, rdep(Stack[0])))</feature>
      <feature>Merge3(InputColumn(POSTAG, Input[0]), OutputColumn(DEPREL, ldep(Input[0])), OutputColumn(DEPREL, rdep(Input[0])))</feature>
      <feature>InputColumn(FORM, Stack[0])</feature>
      <feature>InputColumn(FORM, Input[0])</feature>
      <feature>InputColumn(FORM, Input[1])</feature>
      <feature>InputColumn(FORM, head(Stack[0]))</feature>
   </featuremodel>
</featuremodels>
```

DEPREL – label of dependency relation, FORM – word, POSTAG – POS tag, ldep - dependency having this word as head, rdep - dependency having this word as modifier

32

## Classifiers

- Any multi-class classifier works!
- Maltparser supports
  - Logistic regression
  - Linear SVM
  - Non-linear SVM
- We can use neural networks here!
  - See (Chen and Manning, 2014) next slides

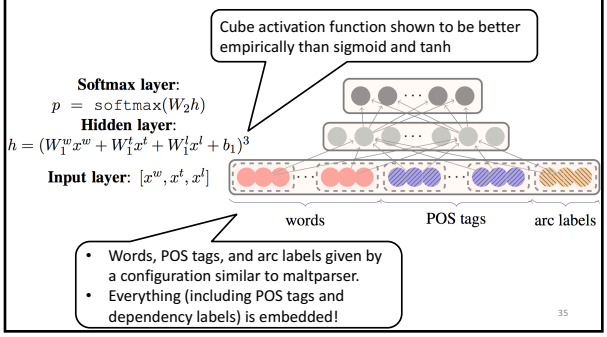
33

## Using NNs to train a shift-reduce parser: Motivation

- Explicit features are *sparse* (especially in combination)
- Explicit features *incompletely* capture the relevant phenomena
- Generating explicit features is *expensive* (especially in combination)

34

## NN for shift-reduce parsing




35

## NN results

| 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.

36

## NN analysis

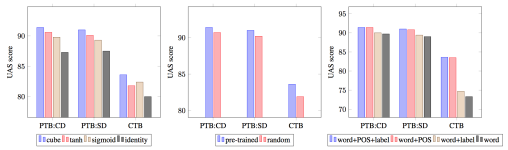
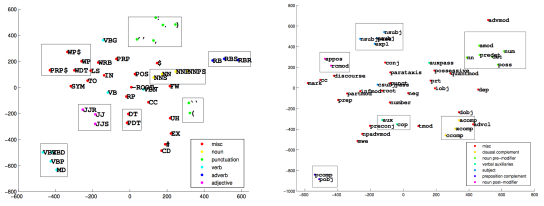



Figure 4: Effects of different parser components. Left: comparison of different activation functions. Middle: comparison of pre-trained word vectors and random initialization. Right: effects of POS and label embeddings.

37

## Visualization of POS and dependency label embeddings



38

## Practice Exercise

- How would you use NNs in a better way for this task?
  - Hint: the stack and the input buffer are sequences…

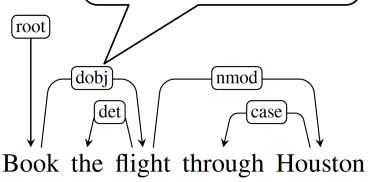
39

**ALTERNATIVE PARSING ALGORITHMS**

40

## Arc-eager algorithm: motivation

- This relation created only at step 8 by arc-standard, even though these words show up early in the input
- Too many chances to mess up!




41

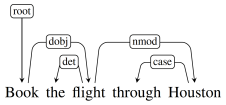
## Arc-eager: actions

- LEFTARC: Assert a head-dependent relation between the word at the front of the input buffer and the word at the top of the stack; pop the stack.
- RIGHTARC: Assert a head-dependent relation between the word on the top of the stack and the word at front of the input buffer; shift the word at the front of the input buffer to the stack.
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.
- REDUCE: Pop the stack.

These changes make arc-eager a top-down algorithm, while arc-standard is bottom-up. Why?

42

## Arc-eager example




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, the, flight, through, houston] | RIGHTARC | (root → book) |
| 1 | [root, book] | [the, flight, through, houston] | SHIFT | |
| 2 | [root, book, the] | [flight, through, houston] | LEFTARC | (the ← flight) |
| 3 | [root, book] | [flight, through, houston] | RIGHTARC | (book → flight) |
| 4 | [root, book, flight] | [through, houston] | SHIFT | |
| 5 | [root, book, flight, through] | [houston] | LEFTARC | (through ← houston) |
| 6 | [root, book, flight] | [houston] | RIGHTARC | (flight → houston) |
| 7 | [root, book, flight, houston] | [] | REDUCE | |
| 8 | [root, book, flight] | [] | REDUCE | |
| 9 | [root, book] | [] | REDUCE | |
| 10 | [root] | [] | Done | |

43

## Practice Exercise

- Generate the program trace for the arc-eager algorithm for the sentences:
  - John Doe visited China.
  - I saw a girl with the telescope.
  - They told him a story.

44

## Handling non-projective trees

- Not required
- If curious, see:
  - Joakim Nivre. Non-Projective Dependency Parsing in Expected Linear Time. https://stp.lingfil.uu.se/~nivre/docs/acl09.pdf
  - Hint: added the SWAP action

45

## Ensemble of shift-reduce parsers

- In practice, arc-eager and arc-standard perform relatively similarly
- But they are different enough to be useful in an ensemble model
  - How would you build such an ensemble?

46

## Participants in the ensemble

| | Devel | In domain | | Out of domain | |
|---|---|---|---|---|---|
| | LAS | LAS | UAS | LAS | UAS |
| MST | 85.36 | 87.07 | 89.95 | 80.48 | 86.08 |
| Malt$^{\rightarrow}_{AE}$ | 84.24 | 85.96 | 88.64 | 78.74 | 84.18 |
| Malt$^{\rightarrow}_{CN}$ | 83.75 | 85.61 | 88.14 | 78.55 | 83.68 |
| Malt$^{\rightarrow}_{AS}$ | 83.74 | 85.36 | 88.06 | 77.23 | 82.39 |
| Malt$^{\leftarrow}_{AS}$ | 82.43 | 83.90 | 86.70 | 76.69 | 82.57 |
| Malt$^{\leftarrow}_{CN}$ | 81.75 | 83.53 | 86.17 | 77.29 | 83.02 |
| Malt$^{\leftarrow}_{AE}$ | 80.76 | 82.51 | 85.35 | 76.18 | 82.02 |

Table 1: Labeled attachment scores (LAS) and unlabeled attachment scores (UAS) for the base models. The parsers are listed in descending order of LAS in the development partition.

47

## Ensemble results

Trivial to implement; nearly 2 points higher than the best individual model

| | In domain | | Out of domain | |
|---|---|---|---|---|
| | LAS | UAS | LAS | UAS |
| Word by word | 88.89 | 91.52 | 82.13* | 87.51* |
| Eisner | 88.83* | 91.47* | 81.99 | 87.32 |
| Attardi | 88.70 | 91.34 | 81.82 | 87.29 |

Table 5: Scores of different combination schemes. * indicates that a model is significantly different than the next lower ranked model.

48

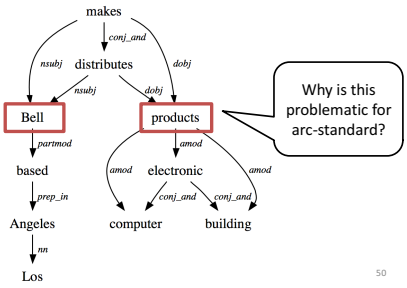
## Take-away

- Shift-reduce dependency parsing algorithms for projective trees
- Shift-reduce dependency parsing algorithms for directed acyclic graphs
- Tree LSTMs

49

## DAG syntax = multiple heads

- Collapsed/enhanced syntactic dependencies allow a word to have multiple heads:




50

## Arc-standard revisited

- LEFTARC: Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack.
- RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack;
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.

Problem operations

51

## Arc-eager revisited

- LEFTARC: Assert a head-dependent relation between the word at the front of the input buffer and the word at the top of the stack; pop the stack.
- RIGHTARC: Assert a head-dependent relation between the word on the top of the stack and the word at front of the input buffer; shift the word at the front of the input buffer to the stack.
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.
- REDUCE: Pop the stack.

Problem operation

52

## Main intuition

- Don't pop elements from the stack when an arc is created
  - This allows them to be attached to multiple heads
- This leads to 5 actions
  - Shift, LeftAttach, RightAttach, LeftReduce, RightReduce

53

## Walkthrough example 1

(a) Desired output:


Initial state:

| Stack | Input tokens |
|---|---|
| | X Y Z |

Current arcs: X Y Z

Action: *SHIFT*

| Stack | Input tokens |
|---|---|
| X | Y Z |

Current arcs: X Y Z

Shift: same as before

Action: *SHIFT*

| Stack | Input tokens |
|---|---|
| Y<br>X | Z |

Current arcs: X Y Z

54

## Walkthrough example 1

Action: *LEFT-ATTACH*

Stack Input tokens

*Z*

*Y*

*X*

Current arcs: *X Y Z*

LeftAttach: creates a left dependency arc between the top two elements on the stack, as long as a right arc does not exist. *No element is removed*!

Action: *SHIFT*

Stack Input tokens

*Z*

*Y*

*X*

Current arcs: *X Y Z*

55

## Walkthrough example 1

Action: *LEFT-REDUCE*

Stack Input tokens

*Z*

*X*

Current arcs: *X Y Z*

LeftReduce: similar to LeftAttach but remove the modifier from the stack

Action: *LEFT-REDUCE*

Stack Input tokens

*Z*

Current arcs: *X Y Z*

56

## Walkthrough example 2

(b) Desired output:

*X Y Z*

Initial state:

Stack Input tokens

*X Y Z*

Current arcs: *X Y Z*

Action: *SHIFT*

Stack Input tokens

*Y Z*

*X*

Current arcs: *X Y Z*

Action: *SHIFT*

Stack Input tokens

*Y Z*

*X*

Current arcs: *X Y Z*

Action: *SHIFT*

Stack Input tokens

*Z*

*Y*

*X*

Current arcs: *X Y Z*

57

## Walkthrough example 2

Action: *RIGHT-ATTACH*

Stack Input tokens

*Y* *Z*

*X*

Current arcs: *X* *Y* *Z*

RightAttach: creates a right dependency between the top 2 elements on the stack, *and pushes the top element on the stack back in the input buffer*

58

## Walkthrough example 2

Action: *RIGHT-REDUCE*

Stack Input

*Z*

*X*

Current arcs: *X* *Y* *Z*

RightReduce: same as LeftReduce, but creates a right dependency arc

Dependency between X and Y enabled because of the previous push back in the input buffer!

59

## Walkthrough example 2

Action: *SHIFT*

Stack Input tokens

*Z*

*X*

Current arcs: *X* *Y* *Z*

Action: *RIGHT-REDUCE*

Stack Input tokens

*X*

Current arcs: *X* *Y* *Z*

60

## Take-away

- Shift-reduce dependency parsing algorithms for projective trees
- Shift-reduce dependency parsing algorithms for directed acyclic graphs
- Tree LSTMs

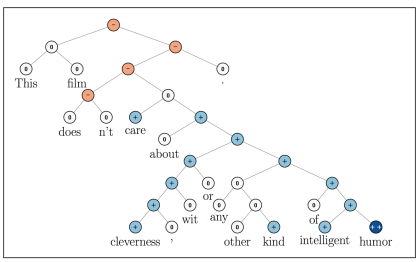
61

## Motivation

- Ok, so now we can generate a *syntactic* representation of a sentence.
- Can we use this to improve existing *semantic* tasks such as sentiment analysis?

62

## Motivation




Stanford sentiment treebank: https://nlp.stanford.edu/sentiment/

63

## Motivation

- We will use LSTMs tailored for trees to capture this information!

64

## Tree LSTMs vs. chain LSTMs

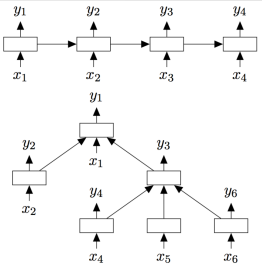



Figure 1: **Top:** A chain-structured LSTM network. **Bottom:** A tree-structured LSTM network with arbitrary branching factor.

65

## Review of chain LSTMs

$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}\right), \\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}\right), \\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}\right), \\
u_t &= \tanh\left(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)}\right), \\
c_t &= i_t \odot u_t + f_t \odot c_{t-1}, \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
$$

Strict left-to-right chaining!

66

## Child-sum Tree-LSTMs

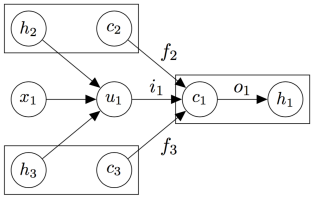



Figure 2: Composing the memory cell $c_1$ and hidden state $h_1$ of a Tree-LSTM unit with two children (subscripts 2 and 3). Labeled edges correspond to gating by the indicated gating vector, with dependencies omitted for compactness.

67

## Child-sum Tree-LSTMs

How much we are forgetting from child *k* of node *j*

C(*j*) = all children of node j

$$\tilde{h}_j = \sum_{k \in C(j)} h_k,$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right),$$

$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right),$$

$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right),$$

$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right),$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k,$$

$$h_j = o_j \odot \tanh(c_j),$$

58

## N-ary Tree-LSTMs

- Applied to tree structures where
  - The branching factor is at most *N*
  - Children are ordered from 1 to *N*
- Unlike the previous model, this one has *separate parameter matrices* per child (rather than one per sum of children)

69

## N-ary Tree-LSTMs

$$i_j = \sigma\left(W^{(i)}x_j + \sum_{\ell=1}^{N} U_\ell^{(i)} h_{j\ell} + b^{(i)}\right), \quad (9)$$

$$f_{jk} = \sigma\left(W^{(f)}x_j + \sum_{\ell=1}^{N} U_{k\ell}^{(f)} h_{j\ell} + b^{(f)}\right), \quad (10)$$

I think this is a bug in the paper. Why?

$$o_j = \sigma\left(W^{(o)}x_j + \sum_{\ell=1}^{N} U_\ell^{(o)} h_{j\ell} + b^{(o)}\right), \quad (11)$$

$$u_j = \tanh\left(W^{(u)}x_j + \sum_{\ell=1}^{N} U_\ell^{(u)} h_{j\ell} + b^{(u)}\right), \quad (12)$$

$$c_j = i_j \odot u_j + \sum_{\ell=1}^{N} f_{j\ell} \odot c_{j\ell}, \quad (13)$$

$$h_j = o_j \odot \tanh(c_j), \quad (14)$$

70

## Encoder for sentiment analysis

- The authors trained an encoder on the final output of these LSTMs ($h_N$), where the output is fed into a softmax layer to predict the sentiment label for the whole sentence
- Trained using the usual negative log likelihood cost function, and L2 regularization

71

## Results on the Stanford sentiment treebank

| Method | Fine-grained | Binary |
|---|---|---|
| RAE (Socher et al., 2013) | 43.2 | 82.4 |
| MV-RNN (Socher et al., 2013) | 44.4 | 82.9 |
| RNTN (Socher et al., 2013) | 45.7 | 85.4 |
| DCNN (Blunsom et al., 2014) | 48.5 | 86.8 |
| Paragraph-Vec (Le and Mikolov, 2014) | 48.7 | 87.8 |
| CNN-non-static (Kim, 2014) | 48.0 | 87.2 |
| CNN-multichannel (Kim, 2014) | 47.4 | **88.1** |
| DRNN (Irsoy and Cardie, 2014) | 49.8 | 86.6 |
| LSTM | 46.4 (1.1) | 84.9 (0.6) |
| Bidirectional LSTM | 49.1 (1.0) | 87.5 (0.5) |
| 2-layer LSTM | 46.0 (1.3) | 86.3 (0.6) |
| 2-layer Bidirectional LSTM | 48.5 (1.0) | 87.2 (1.0) |
| Dependency Tree-LSTM | 48.4 (0.4) | 85.7 (0.4) |
| Constituency Tree-LSTM | | |
| – randomly initialized vectors | 43.9 (0.6) | 82.0 (0.5) |
| – Glove vectors, fixed | 49.7 (0.4) | 87.5 (0.8) |
| – Glove vectors, tuned | **51.0** (0.5) | 88.0 (0.3) |

Table 2: Test set accuracies on the Stanford Sentiment Treebank. For our experiments, we report mean accuracies over 5 runs (standard deviations in parentheses). **Fine-grained:** 5-class sentiment classification. **Binary:** positive/negative sentiment classification.

72

## Take-away

- Shift-reduce dependency parsing algorithms for projective trees
- Shift-reduce dependency parsing algorithms for directed acyclic graphs
- Tree LSTMs

73