CS 4650/7650: Natural Language Processing

Dependency Parsing

Diyi Yang
Presenting: Yuval Pinter (uvp@)
Representing Sentence Structure
Constituent (Phrase-Structure) Representation
Dependency Representation

I prefer the morning flight through Denver
Dependency Representation

- A dependency structure can be defined as a directed graph $G$, consisting of
  - A set $V$ of nodes – **vertices**, words, punctuation, morphemes
  - A set $A$ of arcs – **directed edges**
  - A linear precedence order $<$ on $V$ (**word order**)

```
I prefer the morning flight through Denver
```
Dependency Representation

- Labeled graphs
  - Nodes in $V$ are labeled with word forms (and annotation)
  - Arcs in $A$ are labeled with dependency types
  - $L = \{l_1, l_2, \ldots, l_{|L|}\}$ is the set of permissible arc labels
  - Every arc in $A$ is a triple $(i, j, k)$, representing a dependency form $w_i$ to $w_j$ with label $l_k$

I prefer the morning flight through Denver
Dependency vs Constituency

- Constituency structures explicitly represent
  - Phrases (nonterminal nodes)
  - Structural categories (nonterminal labels)

- Dependency structures explicitly represent
  - Head-dependent relations (directed arcs)
  - Functional categories (arc labels)
  - Possibly some structural categories (parts of speech)
Dependency vs Constituency

The diagram illustrates the difference between dependency and constituency parsing. In dependency parsing, the focus is on the relationships between words, showing how each word depends on another word for meaning. In contrast, constituency parsing focuses on the structure of the sentence, breaking it down into phrases and clauses.
Dependency Representation

I prefer the morning flight through Denver

```
0  ROOT  -  -
1  I     nsubj  2
2  prefer root  0
3  the    det   5
4  morning nmod  5
5  flight  dobj  2
6  through case  7
7  Denver  nmod  5
```

“CoNLL format”
Dependency Relations

- Label
- Relation
- Type

eat<sub>2</sub> → obj → fish<sub>4</sub>

- Head
- Governor
- Parent

- Modifier
- Dependent
- Child
Grammatical Functions

<table>
<thead>
<tr>
<th>Clausal Argument Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSUBJ</td>
<td>Nominal subject</td>
</tr>
<tr>
<td>DOBJ</td>
<td>Direct object</td>
</tr>
<tr>
<td>IOBJ</td>
<td>Indirect object</td>
</tr>
<tr>
<td>CCOMP</td>
<td>Clausal complement</td>
</tr>
<tr>
<td>XCOMP</td>
<td>Open clausal complement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nominal Modifier Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMOD</td>
<td>Nominal modifier</td>
</tr>
<tr>
<td>AMOD</td>
<td>Adjectival modifier</td>
</tr>
<tr>
<td>NUMMOD</td>
<td>Numeric modifier</td>
</tr>
<tr>
<td>APPOS</td>
<td>Appositional modifier</td>
</tr>
<tr>
<td>DET</td>
<td>Determiner</td>
</tr>
<tr>
<td>CASE</td>
<td>Prepositions, postpositions and other case markers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Notable Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONJ</td>
<td>Conjunct</td>
</tr>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
</tbody>
</table>

Selected dependency relations from the Universal Dependency Set
Dependency Constraints

- Syntactic structure is complete (*connectedness*)
  - Connectedness can be enforced by adding a special root node
- Syntactic structure is hierarchical (*acyclicity*)
  - There is a unique pass from the root to each vertex
- Every word has at most one syntactic head (*single-head constraint*)
  - Except root that does not have incoming arcs
- This makes the dependencies a tree
Projectivity

- Projective parse
  - Arcs don’t across each other
  - Mostly true for English
- Non-projective structures are needed to account for
  - Long-distance dependencies
  - Flexible word order
Projectivity

- Dependency grammars do not normally assume that all dependency-trees are projective, because some linguistic phenomena can only be achieved using non-projective trees.

- But a lot of parsers assume that the output trees are projective

- Reasons:
  - Conversion from constituency to dependency
  - The most widely used families of parsing algorithms impose projectivity
The major English dependency treebanks converted from the WSJ sections of the PTB (Marcus et al., 1993)

OntoNotes project (Hovy et al., 2006, Weischedel et al., 2011) adds conversational telephone speech, weblogs, usenet newsgroups, broadcast, and talk shows in English, Chinese and Arabic

Annotated dependency treebanks created for morphologically rich languages such as Czech, Hindi and Finnish, e.g., Prague Dependency Treebank (Bejcek et al., 2013)

https://universaldependencies.org/ (122 treebanks, 71 languages)

Different schemas exist - not all treebanks follow the same attachment rules
The Parsing Problem

- The parsing problem for a dependency parser is to find the optimal dependency tree $y$ given an input sentence $x$.

- This amounts to assigning a syntactic head $i$ and a label $l$ to every node $j$ corresponding to a word $x_j$ in such a way that the resulting graph is a tree rooted at the node 0.
The Parsing Problem

- This is equivalent to finding a spanning tree in the complete graph containing all possible arcs.
Evaluation

- # correct dependencies
  # of dependencies

- LAS – labeled attachment score
- UAS – unlabeled attachment score
- Which is bigger?
Evaluation

\[
\frac{\text{# correct dependencies}}{\text{# of dependencies}}
\]

- LAS – labeled attachment score
- UAS – unlabeled attachment score
- Which is bigger?
- Does 90% sound like a lot?
Parsing Algorithms

- Graph based
  - Minimum Spanning Tree for a sentence
  - McDonald et al.’s (2005) MSTParser
  - Martins et al.’s (2009) Turbo Parser

- Transition based
  - Greedy choice of local transitions guided by a good classifier
  - Deterministic
  - MaltParser (Nivre et al., 2008)
Graph-Based Parsing Algorithms

- Start with a fully-connected directed graph
- Find a Minimum Spanning Tree
  - Chu and Liu (1965) and Edmonds (1967) algorithm
Chu-Liu Edmonds Algorithm

```plaintext
function MAXSPANNINGTREE(G=(V,E), root, score) returns spanning tree

F ← []
T' ← []
score’ ← []

for each v ∈ V do
    bestInEdge ← argmax_{e=(u,v)∈ E} score[e]
    F ← F ∪ bestInEdge
    for each e=(u,v) ∈ E do
        score'[e] ← score[e] − score[bestInEdge]

if T=(V,F) is a spanning tree then return it
else
    C ← a cycle in F
    G' ← CONTRACT(G, C)
    T' ← MAXSPANNINGTREE(G', root, score’)
    T ← EXPAND(T’, C)
return T

function CONTRACT(G, C) returns contracted graph

function EXPAND(T, C) returns expanded graph
```

Select best incoming edge for each node
Subtract its score from all incoming edges
Stopping condition
Contract nodes if there are cycles
Recursively compute MST
Expand contracted nodes
Chu-Liu Edmonds Algorithm

- Select best incoming edge for each node
Chu-Liu Edmonds Algorithm

- Subtract its score from all incoming edges
Chu-Liu Edmonds Algorithm

- Contract nodes if there are cycles
Chu-Liu Edmonds Algorithm

- Recursively compute MST
Chu-Liu Edmonds Algorithm

- Expand contracted nodes
Chu-Liu Edmonds Algorithm

- Expand contracted nodes

Who sees a potential problem?
Scores

\[ \text{score}(S, e) = w \cdot f \]

- Word forms, lemmas, and parts of speech of the headword and its dependent.
- Corresponding features from the contexts before, after, between the words
- Word embeddings / contextual embeddings from LSTM or Transformer
- The dependency relation itself
- The direction of the relation (to the right or left)
- The distance from the head to the dependent
Parsing Algorithms

- Graph based
  - Minimum Spanning Tree for a sentence
  - McDonald et al.’s (2005) MSTParser
  - Martins et al.’s (2009) Turbo Parser

- Transition based
  - Greedy choice of local transitions guided by a good classifier
  - Deterministic
  - MaltParser (Nivre et al., 2008)
Transition Based Parsing

- Greedy discriminative dependency parser
- Motivated by a stack-based approach called **shift-reduce parsing** originally developed for analyzing programming languages (Aho & Ullman, 1972)
Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.
Configuration

**Stack:** partially processed words

**Buffer:** unprocessed words

**Oracle:** a classifier
Operations

At each step choose:
- Shift
At each step choose:
- Shift
- LeftArc (Reduce left)
Operations

At each step choose:

- Shift
- LeftArc (Reduce left)
- RightArc (Reduce right)
Shift-Reduce Parsing

- **Configuration:**
  - Stack, Buffer, Oracle, Set of dependency relations

- **Operations by a classifier at each step:**
  - **Shift**
    - Remove $w_1$ from the buffer, add it to the top of the stack as $s_1$
  - **LeftArc or Reduce Left**
    - Assert a head-dependent relation between $s_1$ and $s_2$
    - Remove $s_2$ from the stack
  - **RightArc or Reduce Right**
    - Assert a head-dependent relation between $s_2$ and $s_1$
    - Remove $s_1$ from the stack
### Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

```
root
  ↓
  dobj
    ↓
    det
      ↓
      nmod

Book me the morning flight
```
## Shift-Reduce Parsing

**Diagram:**
- **root**
- **iobj**
- **dobj**
- **det**
- **nmod**

**Sentence:** Book me the morning flight

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
# Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
</tbody>
</table>
## Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
## Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(morning ← flight)</td>
</tr>
</tbody>
</table>
### Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(the ← flight)</td>
</tr>
</tbody>
</table>
### Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[flight]</td>
<td>LEFTARC</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(the ← flight)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RIGHTARC</td>
<td>(book → flight)</td>
</tr>
</tbody>
</table>

The diagram illustrates the parsing process of the sentence "Book me the morning flight."
### Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book → me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(morning ← flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(the ← flight)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RIGHTARC</td>
<td>(book → flight)</td>
</tr>
</tbody>
</table>
### Shift-Reduce Parsing

#### Table:

<table>
<thead>
<tr>
<th>Step</th>
<th>Stack</th>
<th>Word List</th>
<th>Action</th>
<th>Relation Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[root]</td>
<td>[book, me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[root, book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[root, book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHTARC</td>
<td>(book -&gt; me)</td>
</tr>
<tr>
<td>3</td>
<td>[root, book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[root, book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[root, book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[root, book, the, morning, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(morning &lt;-&gt; flight)</td>
</tr>
<tr>
<td>7</td>
<td>[root, book, the, flight]</td>
<td>[]</td>
<td>LEFTARC</td>
<td>(the &lt;-&gt; flight)</td>
</tr>
<tr>
<td>8</td>
<td>[root, book, flight]</td>
<td>[]</td>
<td>RIGHTARC</td>
<td>(book -&gt; flight)</td>
</tr>
<tr>
<td>10</td>
<td>[root]</td>
<td>[]</td>
<td>Done</td>
<td></td>
</tr>
</tbody>
</table>

#### Diagram:

```
  root
   /  
  dobj det
     /   
    dobj
```

**Sentence:** Book me the morning flight
Shift-Reduce Parsing

- **Configuration:**
  - Stack, Buffer, Oracle, Set of dependency relations

- **Operations by a classifier at each step:**
  - **Shift**
    - Remove $w_1$ from the buffer, add it to the top of the stack as $s_1$
  - **LeftArc or Reduce Left**
    - Assert a head-dependent relation between $s_1$ and $s_2$
    - Remove $s_2$ from the stack
  - **RightArc or Reduce Right**
    - Assert a head-dependent relation between $s_2$ and $s_1$
    - Remove $s_1$ from the stack

Oracle decisions can correspond to unlabeled or labeled arcs
The Oracle is a supervised classifier that learns a function from the configuration to the next operation

How to extract the training set?
Training an Oracle

- Oracle is a supervised classifier that learns a function from the configuration to the next operation
- How to extract the training set?
  - If LeftArc $\rightarrow$ LeftArc
  - If RightArc
    - If $s_1$ dependents have been processed $\rightarrow$ RightArc
    - Else $\rightarrow$ shift
- What features to use?
Training an Oracle: Features

- POS, word-forms, lemmas on the stack/buffer
- Morphological features for some languages
- Previous relations
- Conjunction features

<table>
<thead>
<tr>
<th>Source</th>
<th>Feature templates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One word</strong></td>
<td></td>
</tr>
<tr>
<td>(s_1.w)</td>
<td>(s_1.t)</td>
</tr>
<tr>
<td>(s_2.w)</td>
<td>(s_2.t)</td>
</tr>
<tr>
<td>(b_1.w)</td>
<td>(b_1.w)</td>
</tr>
<tr>
<td><strong>Two word</strong></td>
<td></td>
</tr>
<tr>
<td>(s_1.w \circ s_2.w)</td>
<td>(s_1.t \circ s_2.t)</td>
</tr>
<tr>
<td>(s_1.t \circ s_2.wt)</td>
<td>(s_1.w \circ s_2.w \circ s_2.t)</td>
</tr>
</tbody>
</table>
Learning

- Before 2014: SVMs
- After 2014: Neural Nets
Figure 1: An example of transition-based dependency parsing. Above left: a desired dependency tree, above right: an intermediate configuration, bottom: a transition sequence of the arc-standard system.
**Chen & Manning 2014**

**Softmax layer:**

\[ p = \text{softmax}(W_2h) \]

**Hidden layer:**

\[ h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3 \]

**Input layer:** \([x^w, x^t, x^l]\)

**Configuration**

Stack

<table>
<thead>
<tr>
<th>ROOT</th>
<th>has.VBZ</th>
<th>good.JJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>He_PRP</td>
<td>nsubj</td>
<td>control.NN</td>
</tr>
</tbody>
</table>

**Diagram**

- Words
- POS tags
- Arc labels
- Stack
- Buffer
Stack LSTM (Dyer et al. 2015)

- Instead of recalculating features, configuration updates via NN
Limitations of Transition Parsers

- Oracle prediction - early mistakes are very expensive. Solutions:
  - Different transition systems (arc-standard vs. arc-eager)
  - Beam Search
Limitations of Transition Parsers

- Oracle prediction - early mistakes are very expensive. Solutions:
  - Different transition systems (arc-standard vs. arc-eager)
  - Beam Search

- Can only produce projective trees. Solutions:
  - Complicate the transition system (SWAP action)
  - Apply post-parsing, language-specific rules
Summary

- Graph based
  - + Exact or close-to-exact decoding
  - - Weaker features

- Transition based
  - + Fast
  - + Rich features of context
  - - Greedy decoding
  - - Projective only