Dependency Parsing

CMSC 723 / LING 723 / INST 725

M ARINE C ARPUAT
marine@cs.umd.edu

Slides credit: Joakim Nivre & Ryan McDonald
Agenda

• Formalizing dependency graphs
• Formalizing transition-based parsing
  – Graph-based
  – Transition-based

most material based on Kubler, McDonald & Nivre
Dependencies

- **Typed**: Label indicating relationship between words
  
  ![Diagram showing typed dependency between words]

- **Untyped**: Only which words depend
  
  ![Diagram showing untyped dependency between words]
Data-driven dependency parsing

Goal: learn a good predictor of dependency graphs

Input: x
Output: dependency graph/tree G

Can be framed as a structured prediction task
- very large output space
- with interdependent labels
FORMALIZING DEPENDENCY REPRESENTATIONS
Dependency Graphs

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

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

- For a dependency graph $G = (V, A)$
- With label set $L = \{l_1, \ldots, l_{|L|}\}$
  - $i \rightarrow j \equiv \exists k : (i, j, k) \in A$
  - $i \leftarrow j \equiv i \rightarrow j \lor j \rightarrow i$
  - $i \rightarrow^* j \equiv i = j \lor \exists i' : i \rightarrow i', i' \rightarrow^* j$
  - $i \leftarrow^* j \equiv i = j \lor \exists i' : i \leftarrow i', i' \leftarrow^* j$
Properties of Dependency Trees

- **G** is (weakly) connected:
  - If \( i, j \in V \), \( i \leftrightarrow^* j \).

- **G** is acyclic:
  - If \( i \to j \), then not \( j \to^* i \).

- **G** obeys the single-head constraint:
  - If \( i \to j \), then not \( i' \to j \), for any \( i' \neq i \).

- **G** is projective:
  - If \( i \to j \), then \( i \to^* i' \), for any \( i' \) such that \( i < i' < j \) or \( j < i' < i \).
Economic news had little effect on financial markets.
Non-Projectivity

- Most theoretical frameworks do not assume projectivity
- Non-projective structures are needed to represent
  - Long-distance dependencies
  - Free word order
Directed Spanning Trees

- A directed spanning tree of a (multi-)digraph $G = (V, A)$, is a subgraph $G' = (V', A')$ such that:
  - $V' = V$
  - $A' \subseteq A$, and $|A'| = |V'| - 1$
  - $G'$ is a tree (acyclic)
- A spanning tree of the following (multi-)digraphs
Maximum Spanning Tree

• Assume we have an **arc factored** model
  i.e. weight of graph can be factored as sum or product of weights of its arcs

• Chu-Liu-Edmonds algorithm can find the maximum spanning tree for us!
  – Greedy recursive algorithm
  – Naïve implementation: $O(n^3)$
Chu-Liu-Edmonds illustrated
Chu-Liu-Edmonds illustrated

- Find highest scoring incoming arc for each vertex

- If this is a tree, then we have found MST!!
Chu-Liu-Edmonds illustrated

- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle
Chu-Liu-Edmonds illustrated

- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John → Mary is 3 and saw → Mary is 30
Chu-Liu-Edmonds illustrated

- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root → saw → John is 40 (**)
  - root → John → saw is 29
This is a tree and the MST for the contracted graph!!

Go back up recursive call and reconstruct final graph
Arc weights as linear classifiers

\[ w_{ij}^k = e^{w \cdot f(i,j,k)} \]

- Arc weights are a linear combination of features of the arc, \( f \), and a corresponding weight vector \( w \).
- Raised to an exponent (simplifies some math ...)
- What arc features?
- [McDonald et al. 2005] discuss a number of binary features.
Example of classifier features

John saw Mary McGuire yesterday with his telescope

Features from [McDonald et al. 2005]:
- Identities of the words $w_i$ and $w_j$ and the label $l_k$

head=saw & dependent=with
How to score a graph $G$ using features?

By definition of arc weights as linear classifiers

$$G = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} e^{w \cdot f(i,j,k)}$$

$$= \arg \max_{G \in T(G_x)} \log \prod_{(i,j,k) \in G} e^{w \cdot f(i,j,k)}$$

$$= \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} w \cdot f(i,j,k)$$

$$= \arg \max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k) = \arg \max_{G \in T(G_x)} w \cdot f(G)$$
How can we learn the classifier from data?

e.g., The Perceptron

Training data: $\mathcal{T} = \{(x_t, G_t)\}_{t=1}^{T}$

1. $w^{(0)} = 0; \quad i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $G' = \arg\max_{G'} w^{(i)} \cdot f(G')$
5. if $G' \neq G_t$
6. $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$
7. $i = i + 1$
8. return $w^i$
TRANSITION-BASED DEPENDENCY PARSER
Transition-based parsing

- A transition system for dependency parsing is a quadruple $S = (C, T, c_s, C_t)$, where
  1. $C$ is a set of configurations, each of which contains a buffer $\beta$ of (remaining) nodes and a set $A$ of dependency arcs,
  2. $T$ is a set of transitions, each of which is a (partial) function $t : C \rightarrow C$,
  3. $c_s$ is an initialization function, mapping a sentence $x = w_0, w_1, \ldots, w_n$ to a configuration with $\beta = [1, \ldots, n]$,
  4. $C_t \subseteq C$ is a set of terminal configurations.

Note:

- A configuration represents a parser state.
- A transition represents a parsing action (parser state update).
Transition-based parsing

- Let $S = (C, T, c_s, C_t)$ be a transition system.
- A transition sequence for a sentence $x = w_0, w_1, \ldots, w_n$ in $S$ is a sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ of configurations, such that
  1. $c_0 = c_s(x)$,
  2. $c_m \in C_t$,
  3. for every $i$ ($1 \leq i \leq m$), $c_i = t(c_{i-1})$ for some $t \in T$.
- The parse assigned to $x$ by $C_{0,m}$ is the dependency graph $G_{c_m} = (\{0, 1, \ldots, n\}, A_{c_m})$, where $A_{c_m}$ is the set of dependency arcs in $c_m$. 
Deterministic parsing with an oracle

- An oracle for a transition system $S = (C, T, c_s, C_t)$ is a function $o : C \rightarrow T$.
- Given a transition system $S = (C, T, c_s, C_t)$ and an oracle $o$, deterministic parsing can be achieved by the following simple algorithm:

\[
\text{Parse}(x = (w_0, w_1, \ldots, w_n))
\]

1. $c \leftarrow c_s(x)$
2. \textbf{while} $c \notin C_t$
3. \hspace{1em} $c = [o(c)](c)$
4. \hspace{1em} \textbf{return} $G_c$
Stack-based transition system

- A stack-based configuration for a sentence \( x = w_0, w_1, \ldots, w_n \) is a triple \( c = (\sigma, \beta, A) \), where
  1. \( \sigma \) is a stack of tokens \( i \leq m \) (for some \( m \leq n \)),
  2. \( \beta \) is a buffer of tokens \( j > m \),
  3. \( A \) is a set of dependency arcs such that \( G = (\{0, 1, \ldots, n\}, A) \) is a dependency graph for \( x \).

- A stack-based transition system is a quadruple \( S = (C, T, c_s, C_t) \), where
  1. \( C \) is the set of all stack-based configurations,
  2. \( c_s(x = w_0, w_1, \ldots w_n) = ([0], [1, \ldots, n], \emptyset) \),
  3. \( T \) is a set of transitions, each of which is a function \( t : C \rightarrow C \),
  4. \( C_t = \{ c \in C | c = (\sigma, [], A) \} \).
Transitions & Preconditions

- **Transitions:**
  - **Left-Arc\(_k\):**
    \[
    (\sigma|i,j|\beta,A) \Rightarrow (\sigma,j|\beta,A \cup \{(j,i,k)\})
    \]
  - **Right-Arc\(_k\):**
    \[
    (\sigma|i,j|\beta,A) \Rightarrow (\sigma,i|\beta,A \cup \{(i,j,k)\})
    \]
  - **Shift:**
    \[
    (\sigma,i|\beta,A) \Rightarrow (\sigma|i,\beta,A)
    \]

- **Preconditions:**
  - **Left-Arc\(_k\):**
    \[-[i = 0]\]
    \[-\exists i' \exists k'[(i',i,k') \in A]\]
  - **Right-Arc\(_k\):**
    \[-\exists i' \exists k'[(i',j,k') \in A]\]
Let's try it out...

Transitions:
- **Left-Arc**: \[(\sigma | i, j | \beta, A) \Rightarrow (\sigma, j | \beta, A \cup \{(j, i, k)\})\]
- **Right-Arc**: \[(\sigma | i, j | \beta, A) \Rightarrow (\sigma, i | \beta, A \cup \{(i, j, k)\})\]
- **Shift**: \[(\sigma, i | \beta, A) \Rightarrow (\sigma | i, \beta, A)\]

Preconditions:
- **Left-Arc**:
  \[\neg [i = 0]
  \neg \exists i' \exists k' [(i', i, k') \in A]\]
- **Right-Arc**:
  \[\neg \exists i' \exists k' [(i', j, k') \in A]\]
A few steps illustrated...
A few steps illustrated...

[right-Arc]
Properties of this transition-based parsing algorithm

• A shift-reduce parser

• Time & space complexity
  - $O(n)$, where $n =$ input sentence length
  - Assuming oracle & transition functions can be computed in constant time
Trees & Forests

- A dependency forest (here) is a dependency graph satisfying
  - Root
  - Single-Head
  - Acyclicity
  - but **not** Connectedness
Properties of this transition-based parsing algorithm

- Correctness
  - For every transition sequence \( C_{0,m} \), the resulting graph is a projective dependency forest (soundness)
  - For every projective dependency forest \( G \), there is a transition sequence \( C_{0,m} \) that generates \( G \) (completeness)

- Trick: forest can be turned into tree by adding links to \( \text{ROOT}_0 \)
Replacing the oracle by classifier-based prediction

- Data-driven deterministic parsing:
  - Deterministic parsing requires an oracle.
  - An oracle can be approximated by a classifier.
  - A classifier can be trained using treebank data.

- Learning problem:
  - Approximate a function from configurations (represented by feature vectors) to transitions, given a training set of (gold standard) transition sequences.
  - Three issues:
    - How do we represent configurations by feature vectors?
    - How do we derive training data from treebanks?
    - How do we learn classifiers?
Feature representations

- A feature representation $f(c)$ of a configuration $c$ is a vector of simple features $f_i(c)$.
- Typical features are defined in terms of attributes of nodes in the dependency graph.
  - Nodes:
    - Target nodes (top of $\sigma$, head of $\beta$)
    - Linear context (neighbors in $\sigma$, $\beta$)
    - Structural context (parents, children, siblings given $A$)
  - Attributes:
    - Word form (and/or lemma)
    - Part-of-speech (and morpho-syntactic features)
    - Dependency type (if labeled)
    - Distance (between target tokens)
Typical feature templates

\[
\begin{array}{c}
\vdots \quad w_i \\
\downarrow \quad l_j \\
h_j \\
\downarrow \quad w_j \\
\downarrow \quad r_j \\
\sigma \\
\downarrow \quad w_k \\
\downarrow \quad w_{k+1} \\
\downarrow \quad w_{k+2} \\
\downarrow \quad w_{k+3} \\
\vdots \\
\beta
\end{array}
\]

<table>
<thead>
<tr>
<th>Feature</th>
<th>FORM</th>
<th>LEMMA</th>
<th>CPOS</th>
<th>POS</th>
<th>FEATS</th>
<th>DEPREL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

Formulas and diagrams
Where do training examples come from?

- Training instances have the form \((f(c), t)\), where
  1. \(f(c)\) is a feature representation of a configuration \(c\),
  2. \(t\) is the correct transition out of \(c\) (i.e., \(o(c) = t\)).

- Given a dependency treebank, we can sample the oracle function \(o\) as follows:
  - For each sentence \(x\) with (gold standard) dependency graph \(G_x\), we construct a transition sequence \(C_{0,m} = (c_0, c_1, \ldots, c_m)\) such that
    1. \(c_0 = c_s(x)\),
    2. \(G_{c_m} = G_x\),
  - For each configuration \(c_i (i < m)\), we construct a training instance \((f(c_i), t_i)\), where \(t_i(c_i) = c_{i+1}\).
Learning classifiers

• Many options
  – Perceptron, Logistic regression, Neural Networks, ...

• Extensions
  – Capture sequence of actions with recurrent neural network [Dyer et al. 2015]
Example: perceptron training

Algorithm 2 Online training with a static oracle

1: \( w \leftarrow 0 \)
2: \( \textbf{for } I = 1 \rightarrow \text{ITERATIONS} \textbf{ do} \)
3: \( \textbf{for } \text{sentence } x \text{ with gold tree } G_{\text{gold}} \text{ in corpus } \textbf{ do} \)
4: \( c \leftarrow c_s(x) \)
5: \( \textbf{while } c \text{ is not terminal } \textbf{ do} \)
6: \( t_p \leftarrow \arg \max_t w \cdot \phi(c, t) \)
7: \( t_o \leftarrow o(c, G_{\text{gold}}) \)
8: \( \textbf{if } t_p \neq t_o \textbf{ then} \)
9: \( w \leftarrow w + \phi(c, t_o) - \phi(c, t_p) \)
10: \( c \leftarrow t_o(c) \)
11: \( \textbf{return } w \)

[Goldberg & Nivre, 2012]
Extension: dynamic oracle

Problem with standard classifier-based oracle:
- Is it “static” ie tied to optimal config sequence that produces gold tree
- What if there are multiple sequences for a single gold tree?
- How can we recover if the parser deviates from gold sequence?

One solution: “dynamic oracle” [Goldberg & Nivre 2012]

See also Locally Optimal Learning to Search [Chang et al. ICML 2015]
Extension: dynamic oracle

Algorithm 3 Online training with a dynamic oracle

1: \( w \leftarrow 0 \)
2: for \( I = 1 \rightarrow \text{ITERATIONS} \) do
3:  for sentence \( x \) with gold tree \( G_{\text{gold}} \) in corpus do
4:   \( c \leftarrow c_s(x) \)
5:   while \( c \) is not terminal do
6:     \( t_p \leftarrow \arg\max_t w \cdot \phi(c, t) \)
7:     \( \text{ZERO\_COST} \leftarrow \{ t | o(t; c, G_{\text{gold}}) = \text{true} \} \)
8:     \( t_o \leftarrow \arg\max_{t \in \text{ZERO\_COST}} w \cdot \phi(c, t) \)
9:     if \( t_p \notin \text{ZERO\_COST} \) then
10:    \( w \leftarrow w + \phi(c, t_o) - \phi(c, t_p) \)
11: \( t_n \leftarrow \text{CHOOSE\_NEXT}(I, t_p, \text{ZERO\_COST}) \)
12: \( c \leftarrow t_n(c) \)
13: return \( w \)

See [Goldberg & Nivre 2012] for details
Extension: stack-LSTMs

• Instead of hand-crafted features
• Predict next transition using recurrent neural networks to learn representation of
  – Stack
  – Buffer
  – Sequence of transitions

[Dyer et al. 2015]
ALTERNATE TRANSITION SYSTEMS (OTHER THAN ARC-STANDARD)
A weakness of arc-standard parsing

Right dependents cannot be attached to their head until all their dependents have been attached.
Variation: Arc-Eager Parsing

- **Transitions:**
  - **Left-Arc$_k$:**
    \[(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})\]
  - **Right-Arc$_k$:**
    \[(\sigma|i, j|\beta, A) \Rightarrow (\sigma|i|j, \beta, A \cup \{(i, j, k)\})\]
  - **Reduce:**
    \[(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A)\]
  - **Shift:**
    \[(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)\]

- **Preconditions:**
  - **Left-Arc$_k$:**
    \[\neg[i = 0]\]
    \[\neg \exists i' \exists k'[(i', i, k') \in A]\]
  - **Right-Arc$_k$:**
    \[\neg \exists i' \exists k'[(i', j, k') \in A]\]
  - **Reduce:**
    \[\exists i' \exists k'[(i', i, k') \in A]\]

[Nivre 2003]
Standard vs. Arc-Eager Transition Systems

Transitions:
- **Left-Arc$_k$**:
  \[(\sigma|i,j|\beta, A) \Rightarrow (\sigma,j|\beta, A \cup \{(j, i, k)\})\]
- **Right-Arc$_k$**:
  \[(\sigma|i,j|\beta, A) \Rightarrow (\sigma,i|\beta, A \cup \{(i, j, k)\})\]
- **Shift**:
  \[(\sigma,i|\beta, A) \Rightarrow (\sigma|i, \beta, A)\]

Preconditions:
- **Left-Arc$_k$**:
  \[-[i = 0]
  \neg \exists i' \exists k'[(i', i, k') \in A]\]
- **Right-Arc$_k$**:
  \[-\exists i' \exists k'[(i', j, k') \in A]\]

Transitions:
- **Left-Arc$_k$**:
  \[(\sigma|i,j|\beta, A) \Rightarrow (\sigma,j|\beta, A \cup \{(j, i, k)\})\]
- **Right-Arc$_k$**:
  \[(\sigma|i,j|\beta, A) \Rightarrow (\sigma|i,j, \beta, A \cup \{(i, j, k)\})\]
- **Reduce**:
  \[(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A)\]
- **Shift**:
  \[(\sigma,i|\beta, A) \Rightarrow (\sigma|i, \beta, A)\]

Preconditions:
- **Left-Arc$_k$**:
  \[-[i = 0]
  \neg \exists i' \exists k'[(i', i, k') \in A]\]
- **Right-Arc$_k$**:
  \[-\exists i' \exists k'[(i', j, k') \in A]\]
- **Reduce**:
  \[\exists i' \exists k'[(i', i, k') \in A]\]
DEALING WITH NON-PROJECTIVITY
Arc-standard parsing can’t produce non-projective trees
[root Z] [je] jen jedna na kvalitu
(out-of) (them) (is) (only) (one) (to) (quality)
How frequent are non-projective structures?

- Statistics from CoNLL shared task
  - NPD = non projective dependencies
  - NPS = non projective sentences

<table>
<thead>
<tr>
<th>Language</th>
<th>%NPD</th>
<th>%NPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>5.4</td>
<td>36.4</td>
</tr>
<tr>
<td>German</td>
<td>2.3</td>
<td>27.8</td>
</tr>
<tr>
<td>Czech</td>
<td>1.9</td>
<td>23.2</td>
</tr>
<tr>
<td>Slovene</td>
<td>1.9</td>
<td>22.2</td>
</tr>
<tr>
<td>Portuguese</td>
<td>1.3</td>
<td>18.9</td>
</tr>
<tr>
<td>Danish</td>
<td>1.0</td>
<td>15.6</td>
</tr>
</tbody>
</table>
How to deal with non-projectivity?

(1) change the transition system

<table>
<thead>
<tr>
<th>Transition</th>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP-Left$_r$</td>
<td>$i \neq 0$</td>
</tr>
<tr>
<td>NP-Right$_r$</td>
<td>$i \neq 0$</td>
</tr>
</tbody>
</table>

- Add new transitions
  - That apply to 2$^{\text{nd}}$ word of the stack
  - Top word of stack is treated as context

[Attardi 2006]
How to deal with non-projectivity?

(2) pseudo-projective parsing

Solution:

• “projectivize” a non-projective tree by creating new projective arcs
• That can be transformed back into non-projective arcs in a post-processing step
How to deal with non-projectivity? 

(2) pseudo-projective parsing

Solution:
• "projectivize" a non-projective tree by creating new projective arcs
• That can be transformed back into non-projective arcs in a post-processing step
Recap: Dependency Parsing

– Graph-based parsing
  • Arc-factored models
  • Maximum Spanning Tree
  • Perceptron learning

Tools: http://mstparser.sourceforge.net

– Transition-based parsing
  • Learn to predict transitions given input and history
  • Predict new graphs using shift-reduce parsing
  • Greedy search through arc-standard transition system
  • Guided by classifier learned from treebank

http://www.maltparser.org/
https://github.com/JohnLangford/vowpal_wabbit