Introduction to NLP
Data-Driven Dependency Parsing

Prof. Reut Tsarfaty
Bar Ilan University

November 24, 2020
Statistical Parsing

The Big Picture

sentence -> parsing system -> tree
Statistical Parsing

The Big Picture

sentence $\rightarrow$ decoding $\rightarrow$ tree

parsing system

$\lambda$ $\uparrow$

training

treebank
Statistical Parsing

The Big Picture
Statistical Parsing: The Big Picture

The Questions

- What kind of Trees?
- What kind of Models?
  - Generative
  - Discriminative
- Which Search Algorithm (Decoding)?
- Which Learning Algorithm (Training)?
- What kind of Evaluation?
# Statistical Parsing: The Big Picture

Previously on NLP@BIU

<table>
<thead>
<tr>
<th>Representation</th>
<th>Phrase-Structure Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td>Generative</td>
</tr>
<tr>
<td><strong>Objective</strong></td>
<td>Probabilistic</td>
</tr>
<tr>
<td><strong>Search</strong></td>
<td>CKY</td>
</tr>
<tr>
<td><strong>Train</strong></td>
<td>Maximum Likelihood</td>
</tr>
<tr>
<td><strong>Evaluation</strong></td>
<td>Precision/Recall/F1</td>
</tr>
</tbody>
</table>
Today: Introduction to Dependency Parsing

Today: More Modeling Choices:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Constituency Trees</th>
<th>Dependency Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Generative</td>
<td>?</td>
</tr>
<tr>
<td>Objective</td>
<td>Probailistic</td>
<td>?</td>
</tr>
<tr>
<td>Search</td>
<td>Exhaustive</td>
<td>?</td>
</tr>
<tr>
<td>Train</td>
<td>MLE</td>
<td>?</td>
</tr>
<tr>
<td>Evaluation</td>
<td>$F_1$-Scores</td>
<td>Attachment Scores</td>
</tr>
</tbody>
</table>
Introduction to Dependency Parsing

- The purpose of Syntactic Structures:
  - Encode *Predicate Argument* Structures
  - *Who Does What to Whom?* (When, Where, Why...)

- Properties of Dependency Structures:
  - Defined as (labeled) binary relations between words
  - Reflect a long linguistic (European) tradition
  - Explicitly represent *Argument Structure*
Representation: Labeled vs. Unlabeled

Unlabeled Dependency Tree:

-ROOT-
  dumped
  workers sacks into bins

Labeled Dependency Tree:

-ROOT-
  dumped
  subj workers
d  obj sacks
  prep into
  pobj bins
Representation: Functional vs. Lexical

Functional Dependencies:

-ROOT-
  dumped
  subj workers
dobj sacks
  prep into
  pobj bins

Lexical Dependencies:

-ROOT-
  dumped
  subj workers
dobj sacks
  nmod bins
  case into
Discussions: Options and Schemes

Vertical vs. Horizontal Representation
http://nlp.stanford.edu:8080/corenlp/

The Universal Dependencies Initiative
https://universaldependencies.org/
Let’s Analyse!

The cat sat on the mat.
Let’s Analyse!

The cat is on the mat.
Let’s Analyse!

The cat, which I met, is sitting on the mat.
Let’s Analyse!

The dog and the cat sat on the big and fluffy mat
Let’s Analyse!

The dog and the cat sat on the big and fluffy mat

*You should know how to read/analyse these!*
A labeled dependency tree is a labeled directed tree $T$:
- a set $V$ of nodes, labeled with words (including ROOT)
- a set $A$ of arcs, labeled with dependency types
- a linear precedence order $<$ on $V$

Notation:
- $\text{Arc } \langle v_1, v_2 \rangle$ connects head $v_1$ with dep $v_2$
- $\text{Arc } \langle v_1, l, v_2 \rangle$ connects head $v_1$ with dep $v_2$ with label $l \in L$
- A node $v_0$ (ROOT) serves as a unique root of the tree
Properties of Dependency Trees

A dependency $T$ tree is:

- **connected:**
  For every node $i$ there is a node $j$ such that $i \rightarrow j$ or $j \rightarrow i$

- **acyclic:**
  If $i \rightarrow j$ then not $j \rightarrow^* i$

- **single head:**
  If $i \rightarrow j$ then not $k \rightarrow j$ for any $k \neq i$

- **projective:**
  If $i \rightarrow j$ then $i \rightarrow^* k$ for any $k$ such that $i < k < j$
Non-Projective Dependency Trees

Figure 1: A projective dependency graph.

Figure 2: Non-projective dependency graph.
Non-Projective Dependency Trees

Many parsing algorithms are restricted to projective dependency trees.

Is this a problem?

Statistics from CoNLL-X Shared Task 2006

- 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>

We will (mostly) focus on projective dependencies.
Evaluation Metrics

- **Unlabeled Attachment Scores (UAS)**
  The percentage of identical arcs from the total number or arcs in the tree
  \[
  UAS = \frac{A_{\text{intersect}(i,j)}}{n}
  \]

- **Labeled Attachment Scores (LAS)**
  The percentage of identical arcs with identical labels from the total number or arcs in the tree
  \[
  LAS = \frac{A_{\text{intersect}(i,l,j)}}{n}
  \]

- **Root Accuracy**
  The percentage of sentences with correct root dependency

- **Exact Match**
  The percentage of sentences with parses identical to gold
Models for Dependency Parsing

The Parsing Objective:

\[ y^* = \arg \max_{\{y | y \in \text{GEN}(x)\}} \text{Score}(y) \]

The Modeling Choices:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Dependency Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>?</td>
</tr>
<tr>
<td>Decoder</td>
<td>?</td>
</tr>
<tr>
<td>Trainer</td>
<td>?</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Attachment Scores</td>
</tr>
</tbody>
</table>
Modeling Methods

Our Modeling Tasks:

\[ y^* = \arg \max_{\{y \mid y \in \text{GEN}(x)\}} \text{Score}(y) \]

- **GEN**: How do we generate all \( t \)?
- **Score**: How do we score any \( t \)?
- **argmax**: How do we find the best \( t \)?
Modeling Methods

- **Conversion-Based**
  - Convert Phrase-Structure to Dependency Trees

- **Grammar-Based**
  - Generative methods based on PCFGs

- **Graph-Based**
  - Globally Optimised, Restricted features

- **Transition-Based**
  - Locally Optimal, Unrestricted features

- **Neural-Based**
Modeling Methods (1)

- Conversion-Based: Convert PS trees Using a Head Table
- Grammar-Based
- Graph-Based
- Transition-Based
**Conversion-Based**: Convert PS trees Using a Head Table

<table>
<thead>
<tr>
<th></th>
<th>VP</th>
<th>NP</th>
<th>ADJP</th>
<th>ADVP</th>
<th>S</th>
<th>SQ</th>
<th>SBAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VBD VBN MD VBZ VB VBG VBP VP</td>
<td>NN NX JJR CD JJ JJS RB</td>
<td>NNS QP NN ADVP JJ VBN VBG</td>
<td>RB RBR RBS FW ADVP TO CD JJR</td>
<td>VP S SBAR ADJP UCP NP</td>
<td>VBZ VBD VBP VB MD PRD VP SQ</td>
<td>S SQ SINV SBAR FRAG IN DT</td>
</tr>
</tbody>
</table>
Where Do Dependency Trees Come From?

```
S
  NP
    workers
  VP
    V
      dumped
    NP
      sacks
  PP
    P
      into
    NP
      bins
```
Where Do Dependency Trees Come From?

S/dumped
   NP/workers
      workers
   VP/dumped
      VP/dumped
         V/dumped
            dumped
         NP/sacks
            sacks
      PP/into
         P/into
            into
         NP/bins
Where do Dependency Trees Come From?

```
  dumped
/    \
/     /
/      /
workers dumped into
/      /
/       /
/        /
workers dumped sacks
/    /
/     /
dumped sacks
/     \
/      
/       
dumped
/     \
/      
/       
/        
dumped
/     \
/      
/       
sacks
```

"dumped" workers dumped sacks into bins
Where do Dependency Trees Come From?

dumped

workers

workers

dumped

dumped

sacks

dumped

sacks

into

into

bins
Where do Dependency Trees Come From?

-ROOT-

dumped

workers  sacks  into

  bins
Modeling Methods (2)

- Conversion-Based
  - Grammar-Based
  - Graph-Based
  - Transition-Based

```
TOP
  dumped
  /
workers       dumped
  /
workers
  /
dumped  sacks  into  bins
  /
dumped  sacks  into  bins
```
Grammar-Based Dependency Parsing

The Basic Idea

▶ Treat bi-lexical dependencies as constituents
▶ Decode using chart based algorithm (e.g., CKY)
▶ Learn using standard MLE methods
▶ Evaluate over the set of resulting dependencies as usual
Grammar-Based Dependency Parsing

The Basic Idea
▶ Treat bi-lexical dependencies as constituents
▶ Decode using chart based algorithm (e.g., CKY)
▶ Learn using standard MLE methods
▶ Evaluate over the set of resulting dependencies as usual

Relevant Studies
▶ Original version: [Hays 1964]
▶ Link Grammar: [Sleator and Temperley 1991]
▶ Earley-style left-corner: [Lombardo and Lesmo 1996]

http://cs.jhu.edu/~jason/papers/eisner.coling96.pdf
Grammar-Based Dependency Parsing

The Objective Function:

\[ t^* = \arg\max_{t \in \text{GEN}(x)} P(t) \]

The Modeling Choices:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Dependency Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>PCFG</td>
</tr>
<tr>
<td>Decoder</td>
<td>Adapted CKY</td>
</tr>
<tr>
<td>Trainer</td>
<td>Smoothed MLE</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Attachment Scores</td>
</tr>
</tbody>
</table>
Modeling Methods (3)

✓ Conversion-Based
✓ Grammar-Based
▶ Graph-Based
▶ Transition-Based
The Basic Idea

- Define a global Arc-Factored model
- Treat the search as an MST problem
- Treat the learning as a classification problem
- Evaluate over the set of gold dependencies as usual
Graph-Based Dependency Parsing

The Basic Idea

- Define a global Arc-Factored model
- Treat the search as an MST problem
- Treat the learning as a classification problem
- Evaluate over the set of gold dependencies as usual
Graph-Based Dependency Parsing

Step 1: Defining the Arc Factored Model

\[ t^* = \arg \max_{t \in GEN(V)} w\Phi(t) \]

\[ = \arg \max_{t \in GEN(V)} \sum_{(i \rightarrow j) \in t} w_{\phi_{arc}}(i \rightarrow j) \]
Graph-Based Dependency Parsing

Step 2: Defining Feature Templates

Name $\phi_i$ (had,OBJ, effect)

Unigram head $w_i$

Unigram dep $w_i$

Unigram head pos VB $w_i$

Unigram dep pos NN $w_i$

Bigram head-dep "had-effect" $w_i$

Bigram headpos-deppos VB-NN $w_i$

Labeled Bigram head-dep "had-OBJ-effect" $w_i$

Labeled Bigram headpos-deppos VB-obj-NN $w_i$

In-Between pos VB-IN-NN $w_i$
## Step 2: Defining Feature Templates

<table>
<thead>
<tr>
<th>Name</th>
<th>( \phi_i(\text{had,OBJ, effect}) )</th>
<th>( w_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unigram head</td>
<td>“had”</td>
<td>( w_{\text{unihead}} )</td>
</tr>
<tr>
<td>Unigram dep</td>
<td>“effect”</td>
<td>( w_{\text{unidep}} )</td>
</tr>
<tr>
<td>Unigram head pos</td>
<td>VB</td>
<td>( w_{\text{uniheadpos}} )</td>
</tr>
<tr>
<td>Unigram dep pos</td>
<td>NN</td>
<td>( w_{\text{unidepos}} )</td>
</tr>
<tr>
<td>Bigram head-dep</td>
<td>“had-effect”</td>
<td>( w_{\text{bigram}} )</td>
</tr>
<tr>
<td>Bigram headpos-depospos</td>
<td>VB-NN</td>
<td>( w_{\text{bigrampos}} )</td>
</tr>
<tr>
<td>Labeled Bigram head-dep</td>
<td>“had-OBJ-effect”</td>
<td>( w_{\text{bigramlabel}} )</td>
</tr>
<tr>
<td>Labeled Bigram headpos-depospos</td>
<td>VB-obj-NN</td>
<td>( w_{\text{bigramposlabel}} )</td>
</tr>
<tr>
<td>In-Between pos</td>
<td>VB-IN-NN</td>
<td>( w_{\text{inbetween}} )</td>
</tr>
</tbody>
</table>
Graph-Based Dependency Parsing

Step 3: Learning

E.g., Perceptron

▶ Theory:
  - Find $w$ that assigns higher scores to $y_i$ than any $y \in \mathcal{Y}$
  - If separation exists, will learn to separate the correct structure from the incorrect structures

▶ Practice:
  - Training requires repeated inference-update
  - Computing feature values is time consuming
  - The Averaged-Perceptron variant preferred
Graph-Based Dependency Parsing

Step 4: Finding the Max-Spanning Tree

The Chu-Liu-Edmonds Algorithm

Runtime complexity: $O(n^2)$
Graph-Based Dependency Parsing

Step 3: Online Learning
Perceptron/MIRA (Margin Infused Relaxed Algorithm)

Step 4: Max-Spanning Tree Decoding
The Chu-Liu-Edmonds Algorithm (CLE)

http://repository.upenn.edu/cgi/viewcontent.cgi?article=1056&context=cis_reports
Graph-Based Dependency Parsing

The Objective Function:

\[ t^* = \arg\max_{\{t | \in GEN(x)\}} \sum_{a \in arcs(t)} w^T \Phi(a) \]

The Modeling Choices:

<table>
<thead>
<tr>
<th>Representation</th>
<th>Dependency Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Graph-Based</td>
</tr>
<tr>
<td></td>
<td>Arc-Factored</td>
</tr>
<tr>
<td>Decoder</td>
<td>MST/CLE (O(n^2))</td>
</tr>
<tr>
<td>Trainer</td>
<td>Perceptron/MIRA</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Attachment Scores</td>
</tr>
</tbody>
</table>
Modeling Methods (4)

✓ Conversion-Based
✓ Grammar-Based
✓ Graph-Based
► Transition-Based
Transition-Based Dependency Parsing

The Basic Idea

- Define a transition system
- Define an Oracle Algorithm for Decoding
- Approximate the Oracle Algorithm via Learning
- Evaluate over Dependency Arcs as Usual

http://stp.lingfil.uu.se/~nivre/docs/BeyondMaltParser.pdf
Transition-Based Dependency Parsing

Defining Configurations

A parser **Configuration** is a triplet \( c = (S, Q, A) \), where

- \( S \) = a stack \([..., w_i]_S\) of partially processed nodes
- \( Q \) = a queue \([w_j, ...]_Q\) of remaining input nodes
- \( A \) = a set of labeled arcs \((w_i, l, w_j)\)

**Initialization:**

- \( c_0 = ([w_0]_S, [w_1, ..., w_n]_Q, \{\}) \)
  - Note: \( w_0 = \text{ROOT} \)

**Termination:**

- \( c_t = ([w_0]_S, [], A) \)
Transition-Based Dependency Parsing

Defining Transitions

- **Shift:**
  \[(\ldots S, [w_i, \ldots]_Q, A) \rightarrow ([\ldots, w_i]_S, [\ldots]_Q, A)\]

- **Arc-Left(/):**
  \[(\ldots, w_i, w_j]_S, Q, A) \rightarrow ([\ldots, w_j]_S, Q, A \cup (w_j, l, w_i))\]

- **Arc-Right(/):**
  \[(\ldots, w_i, w_j]_S, Q, A) \rightarrow ([\ldots, w_i]_S, Q, A \cup (w_i, l, w_j))\]
Transition-Based Dependency Parsing

Demo Deck
Transition-Based Dependency Parsing

Deterministic Parsing

Given an oracle $O$ that correctly predicts the next transition $O(c)$, parsing is deterministic:

\[
\text{PARSE}(w_1, \ldots, w_n)
\]

1. $c \leftarrow ([w_0]_S, [w_1, \ldots, w_n]_Q, )$
2. \textbf{while} $Q_c \neq []$ \textbf{or} $|S_c| = 1$
3. \quad $t \leftarrow O(c)$
4. \quad $c \leftarrow t(c)$
5. \textbf{return} $T = (w_0, w_1, \ldots, w_n, A_c)$
Transition-Based Dependency Parsing

Data-Driven Parsing
We approximate the Oracle $O$ using a Classifier $\text{Predict}(c)$ that predicts the next transition using $\text{Features}$ of $c$, feats($c$).

\[
\text{PARSE}(w_1, \ldots, w_n)
\]
1. $c \leftarrow ([w_0]_S, [w_1, \ldots, w_n]_Q, )$
2. while $Q_c \neq []$ or $|S_c| = 1$
3. \hspace{1em} $t \leftarrow \text{Predict}(w, \text{feats}(c))$
4. \hspace{1em} $c \leftarrow t(c)$
5. return $T = (w_0, w_1, \ldots, w_n, A_c)$
Transition-Based Dependency Parsing

Feature Engineering

\[ \text{[ROOT, had, little, effect]}_s \ [on, financial, markets, .]_Q \]
Transition-Based Dependency Parsing

Feature Engineering

\[ [\text{ROOT}, \text{had}, \text{little}, \text{effect}]_s \ [\text{on}, \text{financial}, \text{markets}, .]_q \]

<table>
<thead>
<tr>
<th>name</th>
<th>feature</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>S[0] word</td>
<td>effect</td>
<td>$w_1$</td>
</tr>
<tr>
<td>S[0] pos</td>
<td>NN</td>
<td>$w_2$</td>
</tr>
<tr>
<td>S[1] word</td>
<td>little</td>
<td>$w_3$</td>
</tr>
<tr>
<td>S[1] pos</td>
<td>JJ</td>
<td>$w_4$</td>
</tr>
<tr>
<td>Q[0] word</td>
<td>on</td>
<td>$w_5$</td>
</tr>
<tr>
<td>Q[0] pos</td>
<td>P</td>
<td>$w_6$</td>
</tr>
<tr>
<td>Q[1] word</td>
<td>financial</td>
<td>$w_7$</td>
</tr>
<tr>
<td>Q[1] pos</td>
<td>JJ</td>
<td>$w_8$</td>
</tr>
<tr>
<td>Root(A) word</td>
<td>had</td>
<td>$w_9$</td>
</tr>
<tr>
<td>Root(A) POS</td>
<td>VB</td>
<td>$w_{10}$</td>
</tr>
<tr>
<td>s[0]-S[1]</td>
<td>effect $\rightarrow$ little</td>
<td>$w_{11}$</td>
</tr>
<tr>
<td>s[1]-S[0]</td>
<td>little $\rightarrow$ effect</td>
<td>$w_{12}$</td>
</tr>
</tbody>
</table>
An Oracle $O$ can be approximated by a (linear) classifier:

$$\text{Predict}(t) = \arg\max_t \mathbf{w} \Phi(c, t)$$

History-Based Features $\Phi(c, t)$
- Features over input words relative to $S$ and $Q$
- Features over the (partial) dependency tree defined by $A$
- Features over the (partial) transition sequence so far

Learning $\mathbf{w}$ from Treebank Data
- Reconstruct Oracle sequence for each sentence
- Construct training data set $D = \{(c, t) | O(c) = t\}$
- Maximize accuracy of local predictions $O(c) = t$
Transition-Based Dependency Parsing

Online Learning
Online learning algorithms

Step 4: Greedy Decoding
Greedy: At each step, select the maximum scoring transition.
Recent Advances: Neural-Network Models

The Basic Claim: Both graph based and transition-based models benefit from the move to Neural Networks.

▷ Same overall approach and algorithm as before, but:
  → Replace linear classifier with non-linear to MLP.
  → Use pre-trained word embeddings.
  → Replace feature-extractor with Bi-LSTM.

▷ Further explorations:
  → Semi-supervised learning.
  → Multi-task learning

▷ Remaining Challenges:
  → Out-of-domain parsing (e.g. twitter)
  → Parsing Morphologically-Rich Languages (e.g. Hebrew)
Summarising Dependency Parsing

- Dependency trees as labeled bi-lexical dependencies
  - Data-Driven parsing trained over Dependency Treebanks

- Varied Methods:
  - Conversion-Based (Rules)
  - Grammar-Based (Probabilistic)
  - Graph-Based (Linear, Globally Optimized)
  - Transition-Based (Linear, Locally Optimized)

- Neural Network models work the same but:
  - Non-linear objective eg. MLP
  - Better word-representations eg. Word Embeddings
  - Better (automatic) feature-extraction eg. BiLSTM

- English is "solved" — What about other languages?
  - Stanford CoreNLP
    - https://corenlp.run
  - The UD Initiative:
    - https://universaldependencies.org/
  - UDPipe:
    - http://lindat.mff.cuni.cz/services/udpipe/
  - ONLP:
    - nlp.biu.ac.il/~rtsarfaty/onlp/hebrew/
Summarising Dependency Parsing

- Dependency trees as labeled bi-lexical dependencies
  - Data-Driven parsing trained over Dependency Treebanks
- Varied Methods:
  - Conversion-Based (Rules)
  - Grammar-Based (Probabilistic)
  - Graph-Based (Linear, Globally Optimized)
  - Transition-Based (Linear, Locally Optimized)
- Neural Network models work the same but:
  - Non-linear objective eg. MLP
  - Better word-representations eg. Word Embeddings
  - Better (automatic) feature-extraction eg. BiLSTM
- English is “solved” — What about other languages?
  - Stanford CoreNLP https://corenlp.run
  - The UD Initiative: https://universaldependencies.org/
  - UDPipe: http://lindat.mff.cuni.cz/services/udpipe/
  - ONLP: nlp.biu.ac.il/~rtsarfaty/onlp/hebrew/
So Far

✓ Part 1: Introduction (classes 1-2)
✓ Part 2: Words/Sequences (classes 3-4)
✓ Part 3: Sentences/Trees (classes 5-6)
→ Part 4: Meanings (Prof. Ido Dagan, starting class 7)

To Be Continued...