Breaking Out of Local Optima with Count Transforms and Model Recombination

Valentin I. Spitkovsky

with Hiyan Alshawi (Google Inc.)
and Daniel Jurafsky (Stanford University)
Problem: Unsupervised Parsing (and Grammar Induction)
Problem: Unsupervised Parsing (and Grammar Induction)

- **Input**: Raw Text

  ... By most measures, the nation’s industrial sector is now growing very slowly — if at all. Factory payrolls fell in September. So did the Federal Reserve ...
The Problem Statement

Problem: Unsupervised Parsing (and Grammar Induction)

- **Input:** Raw Text (*Sentences, Tokens* and their *Categories*)
  
  By most measures, the nation’s industrial sector is now growing very slowly — if at all. *Factory payrolls fell in September. So did the Federal Reserve.*...
**Problem**: Unsupervised Parsing (and Grammar Induction)

- **Input**: Raw Text (**Sentences**, **Tokens** and their **Categories**)
  
  ... By most measures, the nation’s industrial sector is now growing very slowly — if at all. **Factory payrolls fell in September.** So did the Federal Reserve ...

- **Output**: Syntactic Structures (and a Probabilistic Grammar)

  ![Syntactic Structure Diagram](image-url)

  N  N  V  P  N
  Factory  payrolls  fell  in  September .
Motivation: Unsupervised (Dependency) Parsing
Motivation: Unsupervised (Dependency) Parsing

- Parsing can be useful...
Motivation: Unsupervised (Dependency) Parsing

- Parsing can be useful...
  - machine translation
    — word alignment, phrase extraction, reordering;
  - web search
    — retrieval, query refinement;
  - question answering, speech recognition, etc.
Motivation: Unsupervised (Dependency) Parsing

- Parsing can be useful...
  - **machine translation**
    - word alignment, phrase extraction, reordering;
  - **web search**
    - retrieval, query refinement;
  - **question answering, speech recognition**, etc.

- But we don’t always have treebanks...
  - specialized **genres** (e.g., legal),
  - understudied **languages**, etc.
Hardness: Why is grammar induction difficult?
Hardness: Why is grammar induction difficult?

- Requires solving a non-convex optimization problem.
  - problem can be NP-hard (Cohen and Smith, 2010)
Hardness: Why is grammar induction difficult?

- Requires solving a non-convex optimization problem. — problem can be NP-hard (Cohen and Smith, 2010)

  - issue: can’t just hill-climb
    - learning is very sensitive to initialization, tie-breaking, etc.
    - hard to replicate others’ results...
Hardness: Why is grammar induction difficult?

- Requires solving a **non-convex optimization problem**.
  - problem can be NP-hard (Cohen and Smith, 2010)

  - issue: can’t just **hill-climb**
    - learning is very sensitive to initialization, tie-breaking, etc.
    - hard to replicate others’ results...

  - alternative: use sampling methods
Hardness: Why is grammar induction difficult?

• Requires solving a **non-convex optimization** problem.
  — problem can be NP-hard (Cohen and Smith, 2010)

  ▶ issue: can’t just **hill-climb**
    ★ learning is very sensitive to initialization, tie-breaking, etc.
    ★ hard to replicate others’ results...

  ▶ alternative: use sampling methods
    ★ also runs into difficulties (e.g., when to stop?)
    ★ but offers useful intuition (i.e., to move away and **restart**)
Hardness: Why is grammar induction difficult?

- Requires solving a non-convex optimization problem.
  — problem can be NP-hard (Cohen and Smith, 2010)

  ▶ issue: can’t just hill-climb
    ★ learning is very sensitive to initialization, tie-breaking, etc.
    ★ hard to replicate others’ results...

  ▶ alternative: use sampling methods
    ★ also runs into difficulties (e.g., when to stop?)
    ★ but offers useful intuition (i.e., to move away and restart)

  ▶ our approach: combining the best of both
Goal: How to not get stuck and make progress?

- Challenge:
Goal: How to not get stuck and make progress?

- **Challenge:**
  - given a (locally optimal) solution, find a better solution
    - e.g., turn a set of parse trees into a better set
Goal: How to not get stuck and make progress?

**Challenge:**
- given a (locally optimal) solution, find a better solution
  - e.g., turn a set of parse trees into a better set

**Desiderata:**
Goal: How to not get stuck and make progress?

**Challenge:**
- given a (locally optimal) solution, find a better solution
  - e.g., turn a set of parse trees into a better set

**Desiderata:**
- want an informed, medium-size step in parameter space
Goal: How to not get stuck and make progress?

Challenge:
- given a (locally optimal) solution, find a better solution
  - e.g., turn a set of parse trees into a better set

Desiderata:
- want an informed, medium-size step in parameter space
- not too big (e.g., random restarts undo all previous work)
Goal: How to not get stuck and make progress?

- **Challenge:**
  - given a (locally optimal) solution, find a better solution
    - e.g., turn a set of parse trees into a better set

- **Desiderata:**
  - want an informed, medium-size step in parameter space
  - not too big (e.g., random restarts undo all previous work)
  - not too small (i.e., not overly self-similar, as in MCMC)
Goal: How to not get stuck and make progress?

- **Challenge:**
  - given a (locally optimal) solution, find a better solution
    - e.g., turn a set of parse trees into a better set

- **Desiderata:**
  - want an informed, medium-size step in parameter space
  - not too big (e.g., random restarts undo all previous work)
  - not too small (i.e., not overly self-similar, as in MCMC)

- **Algorithm Template:**
Goal: How to not get stuck and make progress?

- **Challenge:**
  - given a (locally optimal) solution, find a better solution
    - e.g., turn a set of parse trees into a better set

- **Desiderata:**
  - want an informed, medium-size step in parameter space
  - not too big (e.g., random restarts undo all previous work)
  - not too small (i.e., not overly self-similar, as in MCMC)

- **Algorithm Template:**
  - selectively forget (or filter) some aspect of a solution
Goal: How to not get stuck and make progress?

- **Challenge:**
  - given a (locally optimal) solution, find a better solution
    - e.g., turn a set of parse trees into a better set

- **Desiderata:**
  - want an informed, medium-size step in parameter space
  - not too big (e.g., random restarts undo all previous work)
  - not too small (i.e., not overly self-similar, as in MCMC)

- **Algorithm Template:**
  - selectively forget (or filter) some aspect of a solution,
  - re-optimize from this new starting point
Goal: How to not get stuck and make progress?

- **Challenge:**
  - given a (locally optimal) solution, find a better solution
    - e.g., turn a set of parse trees into a better set

- **Desiderata:**
  - want an informed, medium-size step in parameter space
  - not too big (e.g., random restarts undo all previous work)
  - not too small (i.e., not overly self-similar, as in MCMC)

- **Algorithm Template:**
  - selectively forget (or filter) some aspect of a solution,
  - re-optimize from this new starting point,
  - and take the better of the two.
Goal: How to not get stuck and make progress?

**Challenge:**
- given a (locally optimal) solution, find a better solution
  - e.g., turn a set of parse trees into a better set

**Desiderata:**
- want an informed, medium-size step in parameter space
- not too big (e.g., random restarts undo all previous work)
- not too small (i.e., not overly self-similar, as in MCMC)

**Algorithm Template:**
- selectively forget (or filter) some aspect of a solution,
- re-optimize from this new starting point,
- and take the better of the two.
Transforms: Symmetrizer (Forget Polarity)
Transforms: Symmetrizer (Forget Polarity)

- learn from the undirected arcs of skeletal structures
Transforms: Symmetrizer (Forget Polarity)

- learn from the **undirected arcs** of skeletal structures

[Diagram]

Factory → N → V → P → N → fell → in → September
Transforms: Symmetrizer  (Forget Polarity)

- learn from the undirected arcs of skeletal structures

Factory payrolls fell in September.
Transforms: Symmetrizer (Forget Polarity)

- learn from the undirected arcs of skeletal structures

Once we kind of understand which words go together, take another whack at making heads or tails of syntax!
Transforms: Filter (Forget Incomplete Fragments)
Transforms: Filter (Forget Incomplete Fragments)

- start by splitting text on punctuation (Spitkovsky et al., 2012)
Transforms: Filter (Forget Incomplete Fragments)

- start by splitting text on punctuation (Spitkovsky et al., 2012)

*Linguistics*

*Linguistics (sometimes called philology) is the science that studies language. Scientists who study language are called linguists.*
Transforms: Filter (Forget Incomplete Fragments)

- start by splitting text on punctuation (Spitkovsky et al., 2012)

*Linguistics* from (simple) Wikipedia

*Linguistics* (sometimes called philology)
is the science that studies language.
*Scientists who study language are called linguists.*
Transforms: Filter (Forget Incomplete Fragments)

- start by splitting text on punctuation (Spitkovsky et al., 2012)

**Linguistics** from (simple) Wikipedia

*Linguistics* (sometimes called philology) is the science that studies language. Scientists who study language are called linguists.
Transforms: Filter (Forget Incomplete Fragments)

- start by splitting text on punctuation (Spitkovsky et al., 2012)

Linguistics from (simple) Wikipedia

Linguistics (sometimes called philology) is the science that studies language. Scientists who study language are called linguists.

- once we’ve bootstrapped a rudimentary grammar, retry from just the clean, simple complete sentences!
Transforms: Filter (Forget Incomplete Fragments)

- start by splitting text on punctuation (Spitkovsky et al., 2012)

Stage II

Scientists who study language are called linguists.

- once we've bootstrapped a rudimentary grammar, retry from just the clean, simple complete sentences!
Transforms: Decoder (Forget Unlikely Parses)
Transforms: Decoder  (Forget Unlikely Parses)

- discard most interpretations (a step of Viterbi training)
Transforms: Decoder (Forget Unlikely Parses)

- discard most interpretations (a step of Viterbi training)
Transforms: Decoder  (Forget Unlikely Parses)

- discard most interpretations (a step of Viterbi training)
Transforms: Decoder (Forget Unlikely Parses)

- discard most interpretations (a step of Viterbi training)

1.0
N N V P N
Factory payrolls fell in September

- many reasons why Viterbi steps are a good idea:
  e.g., M-step initialization (Klein and Manning, 2004)
  (Cohen and Smith, 2010)
  (Spitkovsky et al., 2010)
  (Allahverdyan and Galstyan, 2011)
Pop-up: This is not specific to grammar induction!
Pop-up: This is not specific to grammar induction!

- proposed primitive transform operators (unary):
Pop-up: This is not specific to grammar induction!

- proposed primitive transform operators (unary):
  - model ablation (i.e., forget something you learned)
Pop-up: This is not specific to grammar induction!

- proposed primitive transform operators (unary):
  - model ablation (i.e., forget something you learned);
  - data filtering (e.g., drop complex inputs);
Pop-up: This is not specific to grammar induction!

- proposed primitive transform operators (unary):
  - model ablation (i.e., forget something you learned);
  - data filtering (e.g., drop complex inputs);
  - Viterbi stepping (i.e., decode your data).
Pop-up: This is not specific to grammar induction!

- proposed primitive transform operators (unary):
  - model ablation (i.e., forget something you learned);
  - data filtering (e.g., drop complex inputs);
  - Viterbi stepping (i.e., decode your data).

- just need operators (binary or higher) to combine them:
  - a robust way to merge alternatives of varying quality...
Pop-up: This is not specific to grammar induction!

- proposed primitive transform operators (unary):
  - model ablation (i.e., forget something you learned);
  - data filtering (e.g., drop complex inputs);
  - Viterbi stepping (i.e., decode your data).

- just need operators (binary or higher) to combine them:
  - a robust way to merge alternatives of varying quality...

- could construct complex networks that fork/join inputs:
  - useful for many (non-convex) optimization problems!
Goal: How to not get stuck and make progress?

- **Challenge #2:**
Goal: How to not get stuck and make progress?

- **Challenge #2:**
  - given multiple (local) solutions, find a better one
Goal: How to not get stuck and make progress?

- **Challenge #2:**
  - given *multiple* (local) solutions, find a better one

- **Algorithm #2:**
Goal: How to not get stuck and make progress?

- **Challenge #2:**
  - given *multiple* (local) solutions, find a better one

- **Algorithm #2:**
  - compute a mixture model
Goal: How to not get stuck and make progress?

- **Challenge #2:**
  - given multiple (local) solutions, find a better one

- **Algorithm #2:**
  - compute a mixture model,
  - re-optimize from this new starting point
Goal: How to not get stuck and make progress?

- **Challenge #2:**
  - given *multiple* (local) solutions, find a better one

- **Algorithm #2:**
  - compute a mixture model,
  - re-optimize from this new starting point,
  - and take the better of the three.
Goal: How to not get stuck and make progress?

- **Challenge #2:**
  - given multiple (local) solutions, find a better one

- **Algorithm #2:**
  - compute a mixture model,
  - re-optimize from this new starting point,
  - and take the better of the three.
Goal: How to not get stuck and make progress?

- **Challenge #2:**
  - given multiple (local) solutions, find a better one

- **Algorithm #2:**
  - compute a mixture model,
  - re-optimize from this new starting point,
  - and take the better of the three.

- **Improved Algorithm #2:**

Spitkovsky et al. (Stanford & Google)

Breaking out of Local Optima

EMNLP (2013-10-21)
Goal: How to not get stuck and make progress?

- **Challenge #2:**
  - given multiple (local) solutions, find a better one

- **Algorithm #2:** Model Combination
  - compute a mixture model,
  - re-optimize from this new starting point,
  - and take the better of the three.

- **Improved Algorithm #2:**
  - don’t have to stop there...
Goal: How to not get stuck and make progress?

- **Challenge #2:**
  - given multiple (local) solutions, find a better one

- **Algorithm #2:**
  - compute a mixture model,
  - re-optimize from this new starting point,
  - and take the better of the three.

- **Improved Algorithm #2:**
  - don’t have to stop there...
  - if output is better than the worst input, replace and recurse!
Goal: How to not get stuck and make progress?

- **Challenge #2:**
  - given multiple (local) solutions, find a better one

- **Algorithm #2:**
  - compute a mixture model,
  - re-optimize from this new starting point,
  - and take the better of the three.

- **Improved Algorithm #2:**
  - don’t have to stop there...
  - if output is better than the worst input, replace and recurse!
Theme: Try, try again!!
Theme: Try, try again!!

Story-telling time...
Theme: Try, try again!!

Story-telling time...
Theme: Try, try again!!

Story-telling time...
Theme: Try, try again!!

Story-telling time...

Dr. Wiesner
Theme: Try, try again!!

Story-telling time...

Dr. Wiesner, you said “Keep on moving; keep on moving!”

Theme: Many many ways to “keep on moving!”

- **Challenge #3:**
Theme: Many many ways to “keep on moving!”

- **Challenge #3:**
  - everything else has failed,
Theme: Many many ways to “keep on moving!”

- **Challenge #3:**
  - everything else has failed,
  - all transformers and combiners are stuck...
Theme: Many many ways to “keep on moving!"

- **Challenge #3:**
  - everything else has failed,
  - all transformers and combiners are stuck...

- **Algorithm #3:** “lateen EM” (Spitkovsky et al., 2011)
  - use multiple objectives (they are all wrong anyway)
  - e.g., if soft EM is stuck, use hard EM to dig it out...
Theme: Many many ways to “keep on moving!”

- **Challenge #3:**
  - everything else has failed,
  - all transformers and combiners are stuck...

- **Algorithm #3:** “lateen EM” (Spitkovsky et al., 2011)
  - use multiple objectives (they are all wrong anyway)
  - e.g., if soft EM is stuck, use hard EM to dig it out...

- many useful alternative ways to view data:
  - sentence **strings** or parse **trees** (Spitkovsky et al., 2010; 2011)
Theme: Many many ways to “keep on moving!”

- **Challenge #3:**
  - everything else has failed,
  - all transformers and combiners are stuck...

- **Algorithm #3:** “lateen EM” (Spitkovsky et al., 2011)
  - use multiple objectives (they are all wrong anyway)
  - e.g., if soft EM is stuck, use hard EM to dig it out...

- many useful alternative ways to view data:
  - sentence **strings** or parse **trees** (Spitkovsky et al., 2010; 2011)
  - all **data** or just **short** sentences (Klein and Manning, 2004)
Theme: Many many ways to “keep on moving!”

- **Challenge #3:**
  - everything else has failed,
  - all transformers and combiners are stuck...

- **Algorithm #3:** “lateen EM” (Spitkovsky et al., 2011)
  - use multiple objectives (they are all wrong anyway)
  - e.g., if soft EM is stuck, use hard EM to dig it out...

- many useful alternative ways to view data:
  - sentence **strings** or parse **trees** (Spitkovsky et al., 2010; 2011)
  - all data or just **short** sentences (Klein and Manning, 2004)
  - **words** or **categories** (Paskin, 2001; vs. Carroll and Charniak, 1992)
Theme: Many many ways to “keep on moving!”

- **Challenge #3:**
  - everything else has failed,
  - all transformers and combiners are stuck...

- **Algorithm #3:** “lateen EM” (Spitkovsky et al., 2011)
  - use multiple objectives (they are all wrong anyway)
  - e.g., if soft EM is stuck, use hard EM to dig it out...

- many useful alternative ways to view data:
  - sentence strings or parse trees (Spitkovsky et al., 2010; 2011)
  - all data or just short sentences (Klein and Manning, 2004)
  - words or categories (Paskin, 2001; vs. Carroll and Charniak, 1992)
  - feature-rich or bare-bones models (Cohen and Smith, 2009; vs. Spitkovsky et al., 2012)
Theme: Many many ways to “keep on moving!”

- **Challenge #3:**
  - everything else has failed,
  - all transformers and combiners are stuck...

- **Algorithm #3:** “lateen EM” (Spitkovsky et al., 2011)
  - use multiple objectives (they are all wrong anyway)
  - e.g., if soft EM is stuck, use hard EM to dig it out...

- many useful alternative ways to view data:
  - sentence **strings** or **parse trees** (Spitkovsky et al., 2010; 2011)
  - all data or just **short** sentences (Klein and Manning, 2004)
  - **words** or **categories** (Paskin, 2001; vs. Carroll and Charniak, 1992)
  - feature-rich or **bare-bones** models (Cohen and Smith, 2009; vs. Spitkovsky et al., 2012)

- never let convergence interfere with your (non-convex) optimization...
Networks: Fork/Join (FJ)

counts
Networks: Fork/Join (FJ)

- Simple Filter
- Symmetrizer

counts
Networks: Fork/Join (FJ)
Networks: Fork/Join (FJ)

- Simple Filter
- Full Model Optimizer
- Symmetrizer
- Sparse Model Optimizer

counts
Networks: Fork/Join (FJ)

Counts → Simple Filter → Full Model Optimizer

Counts → Symmetrizer → Sparse Model Optimizer
Networks: Fork/Join (FJ)

- Simple Filter
- Full Model Optimizer
- Symmetrizer
- Sparse Model Optimizer
- Combiner

Counts
Networks: Fork/Join (FJ)
Networks: Fork/Join (FJ)
Networks: Fork/Join (FJ)

- Full
- Sparse

Counts

Soft EM

Fork

Join

Lexicalized

Hard EM

Spitkovsky et al. (Stanford & Google)

Breaking out of Local Optima

EMNLP (2013-10-21)
Networks: Fork/Join (FJ)
Networks: Fork/Join (FJ)

- a “grammar inductor” will represent FJ subnetworks:

\[ \text{counts} \rightarrow F \rightarrow \text{full} \rightarrow \text{sparse} \rightarrow \text{full} \]
Networks: Iterated Fork/Join (IFJ)
Networks: Iterated Fork/Join (IFJ)

- daisy-chain inductors, as in “baby steps” (Spitkovsky et al., 2009)
Networks: Iterated Fork/Join (IFJ)

- daisy-chain inductors, as in “baby steps” (Spitkovsky et al., 2009)

  ![Diagram](image)

  - inputs up to length one
  - up to length two
  - up to length

- start with inputs up to length one
  - they have unique parses — an easy (convex) case
Networks: Iterated Fork/Join (IFJ)

- daisy-chain inductors, as in “baby steps” (Spitkovsky et al., 2009)

- start with inputs up to length one
  - they have unique parses — an easy (convex) case
- output initializes training with slightly longer inputs
  - gradually extend solutions to the fully complex target task
Networks: Iterated Fork/Join (IFJ)

- daisy-chain inductors, as in “baby steps” (Spitkovsky et al., 2009)

\[
\begin{aligned}
\text{inputs up to length one} & \quad \text{up to length two} & \quad \text{up to length } / \\
\end{aligned}
\]

- start with inputs up to length one
  - they have unique parses — an easy (convex) case
- output initializes training with slightly longer inputs
  - gradually extend solutions to the fully complex target task
Networks: Iterated Fork/Join (IFJ)

- daisy-chain inductors, as in “baby steps” (Spitkovsky et al., 2009)

- start with inputs up to length one
  - they have unique parses — an easy (convex) case

- output initializes training with slightly longer inputs
  - gradually extend solutions to the fully complex target task

— an instance of deterministic annealing (Allgower and Georg, 1990; Rose, 1998)
Networks: Grounded Iterated Fork/Join (GIFJ)
Networks: Grounded Iterated Fork/Join (GIFJ)

- combine purely iterative (IFJ) and static (FJ) networks:

  counts-up-to-$(l - 1) \xrightarrow{}$

  empty-set-of-counts $\xrightarrow{}$

  up to

  length /
Networks: Grounded Iterated Fork/Join (GIFJ)

- combine purely iterative (IFJ) and static (FJ) networks:
  - counts-up-to-$(l - 1)$
  - empty-set-of-counts
  - up to length $l$
Networks: Grounded Iterated Fork/Join (GIFJ)

- combine purely iterative (IFJ) and static (FJ) networks:
  - counts-up-to-(\(l - 1\))
  - empty-set-of-counts
  - up to length /
Networks: Grounded Iterated Fork/Join (GIFJ)

- combine purely iterative (IFJ) and static (FJ) networks:

  empty-set-of-counts \[\dashrightarrow\] counts-up-to-\((l - 1)\) \[\dashrightarrow\] full \[\dashrightarrow\] counts-up-to-\(/\)

Spitkovsky et al. (Stanford & Google)
Breaking out of Local Optima
EMNLP (2013-10-21) 15 / 21
Networks: Grounded Iterated Fork/Join (GIFJ)

- combine purely iterative (IFJ) and static (FJ) networks:

  counts-up-to-$(l - 1)$  ⨿  full  ⨿  counts-up-to-$/l$
  empty-set-of-counts  ⨿  full network obtained by unrolling the template (as a DBN)

  up to length $l$
Networks: Grounded Iterated Fork/Join (GIFJ)

- combine purely iterative (IFJ) and static (FJ) networks:

  counts-up-to-$(l - 1)$ \rightarrow \text{full} \rightarrow \text{counts-up-to-}l

  empty-set-of-counts \rightarrow \text{full network obtained by unrolling the template (as a DBN)}

  up to length $/$

  - full network obtained by unrolling the template (as a DBN)
    - can specify relatively “deep” learning architectures
    - without sacrificing (too much) clarity or simplicity
Networks: Grounded Iterated Fork/Join (GIFJ)

- combine purely iterative (IFJ) and static (FJ) networks:

  counts-up-to-$(l - 1)$ \rightarrow \text{full} \rightarrow \text{counts-up-to-}$/\$

  empty-set-of-counts \rightarrow \text{full network obtained by unrolling the template (as a DBN)}$

  - can specify relatively “deep” learning architectures
  - without sacrificing (too much) clarity or simplicity

- a structured way of organizing optimizers into networks:
  - only a handful of primitives here
  - would be hard to do without modularity and abstraction
  - can understand and improve components in isolation
## Results: Directed Dependency Accuracies

Section 23 of English WSJ (all sentences)

<table>
<thead>
<tr>
<th>System</th>
<th>DDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Gimpel and Smith, 2012)</td>
<td>53.1</td>
</tr>
<tr>
<td>(Gillenwater et al., 2010)</td>
<td>53.3</td>
</tr>
<tr>
<td>(Bisk and Hockenmaier, 2012)</td>
<td>53.3</td>
</tr>
<tr>
<td>(Blunsom and Cohn, 2010)</td>
<td>55.7</td>
</tr>
<tr>
<td>(Tu and Honavar, 2012)</td>
<td>57.0</td>
</tr>
<tr>
<td>(Spitkovsky et al., 2013)</td>
<td><strong>64.4</strong></td>
</tr>
</tbody>
</table>
## Results: Unlabeled Constituents

### Section 23 of English WSJ (all sentences)

<table>
<thead>
<tr>
<th>System</th>
<th>F(_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-CCM</td>
<td>45.1</td>
</tr>
<tr>
<td>LLCCM</td>
<td>47.6</td>
</tr>
<tr>
<td>CCL</td>
<td>52.8</td>
</tr>
<tr>
<td>PRLG</td>
<td>54.6</td>
</tr>
<tr>
<td>(Spitkovsky et al., 2013)</td>
<td>54.2</td>
</tr>
</tbody>
</table>
Results: Unlabeled Constituents

Section 23 of English WSJ (all sentences)

<table>
<thead>
<tr>
<th>System</th>
<th>$F_1$</th>
<th>P</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-CCM (Huang et al., 2012)</td>
<td>45.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLCCM (Golland et al., 2012)</td>
<td>47.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCL (Seginer, 2007)</td>
<td>52.8</td>
<td>54.6</td>
<td>51.1</td>
</tr>
<tr>
<td>PRLG (Ponvert et al., 2011)</td>
<td>54.6</td>
<td>60.4</td>
<td>49.8</td>
</tr>
<tr>
<td>(Spitkovsky et al., 2013)</td>
<td>54.2</td>
<td>55.6</td>
<td>52.8</td>
</tr>
<tr>
<td>Dependency-Based Upper Bound</td>
<td>87.2</td>
<td>100</td>
<td>77.3</td>
</tr>
</tbody>
</table>
Results: Multi-Lingual Dependencies

2006/7 CoNLL Data (19 languages): Arabic, Basque, Bulgarian, Catalan, Chinese, Czech, Danish, Dutch, English, German, Greek, Hungarian, Italian, Japanese, Portuguese, Slovenian, Spanish, Swedish, Turkish
## Results: Multi-Lingual Dependencies

2006/7 CoNLL Data (19 languages): Arabic, Basque, Bulgarian, Catalan, Chinese, Czech, Danish, Dutch, English, German, Greek, Hungarian, Italian, Japanese, Portuguese, Slovenian, Spanish, Swedish, Turkish

<table>
<thead>
<tr>
<th>System</th>
<th>DDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Mareček and Žabokrtský, 2012)</td>
<td>40.0</td>
</tr>
<tr>
<td>(Spitkovsky et al., 2012b)</td>
<td>42.9</td>
</tr>
<tr>
<td>(Spitkovsky et al., 2013)</td>
<td>48.6</td>
</tr>
</tbody>
</table>
Results: Multi-Lingual Dependencies

2006/7 CoNLL Data (19 languages): Arabic, Basque, Bulgarian, Catalan, Chinese, Czech, Danish, Dutch, English, German, Greek, Hungarian, Italian, Japanese, Portuguese, Slovenian, Spanish, Swedish, Turkish

<table>
<thead>
<tr>
<th>System</th>
<th>DDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Mareček and Žabokrtský, 2012)</td>
<td>40.0</td>
</tr>
<tr>
<td>(Spitkovsky et al., 2012b)</td>
<td>42.9</td>
</tr>
<tr>
<td>(Spitkovsky et al., 2013)</td>
<td>48.6</td>
</tr>
<tr>
<td>(Mareček and Straka, 2013)</td>
<td>48.7</td>
</tr>
</tbody>
</table>
Conclusion: Summary

- useful way of merging grammars of different quality:
**Conclusion: Summary**

- useful way of **merging grammars** of different quality:
  - not always easy, e.g., in machine translation (Xiao et al., 2010)
Conclusion: Summary

- useful way of **merging grammars** of different quality:
  - not always easy, e.g., in machine translation (Xiao et al., 2010)

- exploited **multiple views** of data:
Conclusion: Summary

- useful way of **merging grammars** of different quality:
  - not always easy, e.g., in machine translation (Xiao et al., 2010)

- exploited **multiple views** of data:
  - simple sentences — easy to recognize root words
Conclusion: Summary

- useful way of **merging grammars** of different quality:
  - not always easy, e.g., in machine translation (Xiao et al., 2010)

- exploited **multiple views** of data:
  - simple sentences — easy to recognize root words
  - fragments split on punctuation — learn word associations
**Conclusion: Summary**

- **useful way of merging grammars of different quality:**
  - not always easy, e.g., in machine translation (Xiao et al., 2010)

- **exploited multiple views of data:**
  - simple sentences — easy to recognize root words
  - fragments split on punctuation — learn word associations
  - skeleton parses — for recovering correct arc polarities
## Conclusion: Summary

- **useful way of merging grammars** of different quality:
  - not always easy, e.g., in machine translation (Xiao et al., 2010)

- **exploited multiple views** of data:
  - simple sentences — easy to recognize root words
  - fragments split on punctuation — learn word associations
  - skeleton parses — for recovering correct arc polarities

- **state-of-the-art results** for **grammar induction**:
  - English WSJ (both dependency and constituency)
  - 19 languages of the 2006/7 CoNLL data (dependency)
Conclusion: Implications (Why This Matters)
Conclusion: Implications (Why This Matters)

- applicable all over NLP
Conclusion: Implications

- applicable all over NLP, even within sampling methods:
Conclusion: Implications  (Why This Matters)

- applicable all over NLP, even within sampling methods:
  - transformed models as seeds to multi-chain MCMC
    - e.g., symmetrized models, which would tend to be rejected
Conclusion: Implications (Why This Matters)

- applicable all over NLP, even within sampling methods:
  - transformed models as seeds to multi-chain MCMC
    - e.g., symmetrized models, which would tend to be rejected
  - combining as an alternative to swapping adjacent chains
    - e.g., in MCMC (Geyer, 1991)
Conclusion: Implications  (Why This Matters)

- applicable all over NLP, even within sampling methods:
  - transformed models as seeds to multi-chain MCMC
    - e.g., symmetrized models, which would tend to be rejected
  - combining as an alternative to swapping adjacent chains
    - e.g., in MCMC (Geyer, 1991)

- working title: “the power of forgetting and starting over”
Conclusion: Implications (Why This Matters)

- applicable all over NLP, even within sampling methods:
  - transformed models as seeds to multi-chain MCMC
    - e.g., symmetrized models, which would tend to be rejected
  - combining as an alternative to swapping adjacent chains
    - e.g., in MCMCMC (Geyer, 1991)

- working title: “the power of forgetting and starting over”
  - “unlearning” — an old idea in machine learning
    - e.g., regularization, pruning of decision trees, etc.
Conclusion: Implications (Why This Matters)

- applicable all over NLP, even within sampling methods:
  - transformed models as seeds to multi-chain MCMC
    - e.g., symmetrized models, which would tend to be rejected
  - combining as an alternative to swapping adjacent chains
    - e.g., in MCMCMC (Geyer, 1991)

- working title: “the power of forgetting and starting over”
  - “unlearning” — an old idea in machine learning
    - e.g., regularization, pruning of decision trees, etc.
  - also important in neuroscience
    - e.g., neuronal shedding (Craik and Bialystok, 2006; Low and Cheng, 2006)
Conclusion: Implications (Why This Matters)

- applicable all over NLP, even within sampling methods:
  - transformed models as seeds to multi-chain MCMC
    - e.g., symmetrized models, which would tend to be rejected
  - combining as an alternative to swapping adjacent chains
    - e.g., in MCMCMC (Geyer, 1991)

- working title: “the power of forgetting and starting over”
  - “unlearning” — an old idea in machine learning
    - e.g., regularization, pruning of decision trees, etc.
  - also important in neuroscience (Craik and Bialystok, 2006)
    - e.g., neuronal shedding (Low and Cheng, 2006)
  - some things we learn, which are responsible for our early success, are also what holds us back later in life...
Thanks!

Questions?