Linear Time Constituency Parsing with RNNs and Dynamic Programming

Juneki Hong \(^1\) Liang Huang \(^{1,2}\)

\(^1\) Oregon State University
\(^2\) Baidu Research Silicon Valley AI Lab
Span Parsing is SOTA in Constituency Parsing

- Cross+Huang 2016 introduced Span Parsing
  - But with greedy decoding.
- Stern et al. 2017 had Span Parsing with Exact Search and Global Training
  - But was too slow: $O(n^3)$
- Can we get the best of both worlds?
  - Something that is both fast and accurate?

---

Accuracy

Cross + Huang 2016

Stern et al. 2017

Kitaev + Klein 2018

New at ACL 2018! Also Span Parsing!

Joshi et al. 2018

Our Work
Both Fast and Accurate!

Baseline Chart Parser (Stern et al. 2017a) | 91.79
---|---
Our Linear Time Parser | 91.97
In this talk, we will discuss:

- Linear Time Constituency Parsing using dynamic programming
- Going slower in order to go faster: $O(n^3) \rightarrow O(n^4) \rightarrow O(n)$
- Cube Pruning to speed up Incremental Parsing with Dynamic Programming
  - From $O(n \ b^2)$ to $O(n \ b \ \log b)$
- An improved loss function for Loss-Augmented Decoding
  - 2nd highest accuracy among single systems trained on PTB only

$$O(2^n) \rightarrow O(n^3) \rightarrow O(n^4) \sim O(nb^2) \sim O(nb \ \log b)$$
Span Parsing

- Span differences are taken from an encoder (in our case: a bi-LSTM)
- A span is scored and labeled by a feed-forward network.
- The score of a tree is the sum of all the labeled span scores

\[ s_{tree}(t) = \sum_{(i,j,X) \in t} s(i, j, X) \]

\[ s(i, j, X) = (f_j - f_i, b_i - b_j) \]

Cross + Huang 2016  Stern et al. 2017  Wang + Chang 2016
Incremental Span Parsing Example

Cross + Huang 2016
Incremental Span Parsing Example

Eat ice cream after lunch

<table>
<thead>
<tr>
<th>Action</th>
<th>Label</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shift</td>
<td>ø</td>
</tr>
</tbody>
</table>

Cross + Huang 2016
Incremental Span Parsing Example

<table>
<thead>
<tr>
<th>Action</th>
<th>Label</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shift</td>
<td>ø</td>
</tr>
<tr>
<td>2</td>
<td>Shift</td>
<td>ø</td>
</tr>
</tbody>
</table>

Cross + Huang 2016
Incremental Span Parsing Example

<table>
<thead>
<tr>
<th>Action</th>
<th>Label</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shift</td>
<td>ø</td>
</tr>
<tr>
<td>2</td>
<td>Shift</td>
<td>ø</td>
</tr>
<tr>
<td>3</td>
<td>Shift</td>
<td>ø</td>
</tr>
</tbody>
</table>

Cross + Huang 2016
### Incremental Span Parsing Example

<table>
<thead>
<tr>
<th>Action</th>
<th>Label</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shift</td>
<td>Ø</td>
</tr>
<tr>
<td>2</td>
<td>Shift</td>
<td>Ø</td>
</tr>
<tr>
<td>3</td>
<td>Shift</td>
<td>Ø</td>
</tr>
<tr>
<td>4</td>
<td>Reduce</td>
<td>NP</td>
</tr>
</tbody>
</table>

Cross + Huang 2016
Incremental Span Parsing Example

Cross + Huang 2016
Incremental Span Parsing Example

Cross + Huang 2016
Incremental Span Parsing Example

<table>
<thead>
<tr>
<th>Action</th>
<th>Label</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shift</td>
<td>ø</td>
</tr>
<tr>
<td>2</td>
<td>Shift</td>
<td>ø</td>
</tr>
<tr>
<td>3</td>
<td>Shift</td>
<td>ø</td>
</tr>
<tr>
<td>4</td>
<td>Reduce</td>
<td>NP</td>
</tr>
<tr>
<td>5</td>
<td>Reduce</td>
<td>ø</td>
</tr>
<tr>
<td>6</td>
<td>Shift</td>
<td>ø</td>
</tr>
<tr>
<td>7</td>
<td>Shift</td>
<td>NP</td>
</tr>
</tbody>
</table>

Cross + Huang 2016
Incremental Span Parsing Example

<table>
<thead>
<tr>
<th>Action</th>
<th>Label</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shift</td>
<td>Ø</td>
</tr>
<tr>
<td>2</td>
<td>Shift</td>
<td>Ø (0, 1)</td>
</tr>
<tr>
<td>3</td>
<td>Shift</td>
<td>Ø (0, 1) (1, 2)</td>
</tr>
<tr>
<td>4</td>
<td>Reduce</td>
<td>NP (0, 1) (1, 3)</td>
</tr>
<tr>
<td>5</td>
<td>Reduce</td>
<td>Ø (0, 3)</td>
</tr>
<tr>
<td>6</td>
<td>Shift</td>
<td>Ø (0, 3) (3, 4)</td>
</tr>
<tr>
<td>7</td>
<td>Shift</td>
<td>NP (0, 3) (3, 4) (4, 5)</td>
</tr>
<tr>
<td>8</td>
<td>Reduce</td>
<td>PP (0, 3) (3, 5)</td>
</tr>
</tbody>
</table>

Cross + Huang 2016
Incremental Span Parsing Example

Cross + Huang 2016
How Many Possible Parsing Paths?

- 2 actions per state.
- $O(2^n)$
Equivalent Stacks?

- Observe that all stacks that end with \((i,j)\) will be treated the same!
- …Until \((i,j)\) is popped off.

\[
\begin{align*}
[(0, 2), (2, 7), (7, 9)] & \quad \text{becomes} \quad [(0, 3), (3, 7), (7, 9)] \\
\end{align*}
\]

- So we can treat these as “temporarily equivalent”, and merge.
Equivalent Stacks?

- Observe that all stacks that end with \((i, j)\) will be treated the same!
- …Until \((i, j)\) is popped off.

\[
\begin{align*}
\ldots, (0, 2) & \rightarrow [\ldots, (2, 7)] & \rightarrow [\ldots, (7, 9)] \\
[\ldots, (0, 3)] & \rightarrow [\ldots, (3, 7)]
\end{align*}
\]

- This is our new stack representation.

Graph-Structured Stack (Tomita 1988; Huang + Sagae 2010)
Equivalent Stacks?

- Observe that all stacks that end with \((i, j)\) will be treated the same!
- …Until \((i, j)\) is popped off.

Reduce Actions: \[
\frac{[\ldots, (k, i)] \quad [\ldots, (i, j)]}{[\ldots, (k, j)]} \quad O(n^3)
\]

Graph-Structured Stack (Tomita 1988; Huang + Sagae 2010)
Dynamic Programming: Merging Stacks

- Temporarily merging stacks will make our state space polynomial.

- And our parsing state is represented by top span \((i, j)\).
Shift-Reduce Parsers are traditionally action synchronous.

This makes beam-search straightforward.

We will also do the same.

But will show that this will slow down our DP (before applying beam-search).
Action Synchronous Parsing Example

Gold: Shift (0,1)
Action Synchronous Parsing Example

Gold:

<table>
<thead>
<tr>
<th></th>
<th>Shift (0,1)</th>
<th>Shift (1,2)</th>
</tr>
</thead>
</table>

$\epsilon \rightarrow_{\text{sh}} (0,1) \rightarrow_{\text{sh}} (1,2)$

Left Pointers

Gold Parse
Action Synchronous Parsing Example

Gold:

<table>
<thead>
<tr>
<th></th>
<th>Shift (0,1)</th>
<th>Shift (1,2)</th>
<th>Shift (2,3)</th>
</tr>
</thead>
</table>

Left Pointers

Gold Parse
Action Synchronous Parsing Example

Gold:

<table>
<thead>
<tr>
<th></th>
<th>Shift</th>
<th>Shift</th>
<th>Shift</th>
<th>Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0,1)</td>
<td>(1,2)</td>
<td>(2,3)</td>
<td>(1,3)</td>
</tr>
</tbody>
</table>

Left Pointers

Gold Parse
Action Synchronous Parsing Example

Gold:

<table>
<thead>
<tr>
<th></th>
<th>Shift (0,1)</th>
<th>Shift (1,2)</th>
<th>Shift (2,3)</th>
<th>Reduce (1,3)</th>
<th>Reduce (0,3)</th>
</tr>
</thead>
</table>

Left Pointers

Gold Parse
Action Synchronous Parsing Example

Gold:

<table>
<thead>
<tr>
<th>Action</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift</td>
<td>(0,1)</td>
<td>Shift</td>
<td>(1,2)</td>
<td>Shift</td>
<td>(2,3)</td>
<td>Shift</td>
</tr>
<tr>
<td>Reduce</td>
<td>(1,3)</td>
<td>Shift</td>
<td>(2,4)</td>
<td>Shift</td>
<td>(3,4)</td>
<td>Shift</td>
</tr>
<tr>
<td>Reduce</td>
<td>(1,4)</td>
<td>Shift</td>
<td>(3,5)</td>
<td>Shift</td>
<td>(0,4)</td>
<td>Shift</td>
</tr>
<tr>
<td>Reduce</td>
<td>(2,5)</td>
<td>Shift</td>
<td>(3,5)</td>
<td>Shift</td>
<td>(2,5)</td>
<td>Shift</td>
</tr>
<tr>
<td>Reduce</td>
<td>(0,5)</td>
<td>Shift</td>
<td>(3,5)</td>
<td>Shift</td>
<td>(2,5)</td>
<td>Shift</td>
</tr>
</tbody>
</table>

Left Pointers

Gold Parse

ɛ → sh → (0,1) → sh → (1,2) → sh → (2,3) → sh → (3,4) → sh → (4,5) → r → (3,5)
(0,2) → r → (0,3)
(2,3) → sh → (0,3)
(2,4) → sh → (2,4)
(3,4) → sh → (3,4)
Action Synchronous Parsing Example

Gold:

<table>
<thead>
<tr>
<th></th>
<th>Shift (0,1)</th>
<th>Shift (1,2)</th>
<th>Shift (2, 3)</th>
<th>Reduce (1, 3)</th>
<th>Reduce (0, 3)</th>
<th>Shift (3, 4)</th>
<th>Shift (4, 5)</th>
<th>Reduce (3, 5)</th>
<th>Reduce (0, 5)</th>
</tr>
</thead>
</table>
Runtime Analysis: $O(n^4)$

Huang+Sagae 2010
Runtime Analysis: $O(n^4)$

#steps: $2n - 1 = O(n)$
Runtime Analysis: $O(n^4)$

#states per step: $O(n^2)$

$\#steps: 2n - 1 = O(n)$

Huang+Sagae 2010
Runtime Analysis: $O(n^4)$

#steps: $2n - 1 = O(n)$

$O(n^3)$ states
Runtime Analysis: $O(n^4)$

Check out the paper for our new theorem:

$$l' = l - 2(j - i) + 1$$

Thanks to Dezhong Deng!

#left pointers per state: $O(n)$

#states per step: $O(n^2)$

#steps: $2n - 1 = O(n)$

$O(n^3)$ states

$l' : [\ldots, (k, i)] \quad l : [\ldots, (i, j)]$

$l+1 : [\ldots, (k, j)]$

$O(n^4)$

Huang+Sagae 2010
Going slower to go faster

- Our Action-Synchronous algorithm has a slower runtime than CKY!
- However, it also becomes straightforward to prune using beam search.
- So we can achieve a linear runtime in the end.
Now our runtime is $O(n)$. 
But this $O(n)$ is hiding a constant.
But this $O(n)$ is hiding a constant.

$O(b)$ left pointers per state

$O(nb^2)$ runtime

$b$ states per action step
We can apply cube pruning to make $O(nb \log b)$.
We can apply cube pruning to make $O(nb \log b)$.

By pushing all states and their left pointers into a heap.
Cube Pruning

- We can apply cube pruning to make $O(nb \log b)$

- By pushing all states and their left pointers into a heap
- And popping the top $b$ unique subsequent states

[Diagram of state transitions]
Cube Pruning

- We can apply cube pruning to make $O(nb \log b)$

- By pushing all states and their left pointers into a heap
- And popping the top $b$ unique subsequent states
- First time Cube-Pruning has been applied to Incremental Parsing

Chiang 2007
Huang+Chiang 2007
Runtime on PTB and Discourse Treebank

- Chart Parsing: $O(n^{2.26})$
- Beam 20 No Cube-Pruning: $O(n^{1.26})$
- Beam 20 Cube Pruned: $O(n^{1.08})$
- Beam 5 Cube Pruned: $O(n^{0.97})$

Chart parsing vs Sentence Length and Discourse Length (words)
Training

• Structured SVM approach (Taskar et al. 2003; Stern et al. 2017):
  • Goal: Score the gold tree higher than all others by a margin:
    \[ \forall t, s(t^*) - s(t) \geq \Delta(t, t^*) \]

• Loss Augmented Decoding:
  • During Training: Return the most violated tree (i.e., highest augmented score):
    \[ \hat{t} = \arg \max_t (s(t) + \Delta(t, t^*)) \]

• Minimize:
  \[ (s(\hat{t}) + \Delta(\hat{t}, t^*)) - s(t^*) \]
Loss Function

- Counts the incorrectly labeled spans in the tree (Stern et al. 2017)
- Happens to be decomposable, so can even be used to compare partial trees.

\[
\Delta(t, t^*) = \sum_{(i, j, X) \in t} 1\left(X \neq t^*_{(i, j)}\right)
\]
We observe that the null label $\emptyset$ is used in two different ways:

- To facilitate ternary and n-ary branching trees.
- As a default label for incorrect spans that violate other gold spans.

$t^*(i, j) = \emptyset$
We modify the loss to account for incorrect spans in the tree.

$$\Delta(t, t^*) = \sum_{(i,j,X) \in t} 1(X \neq t^* \downarrow_{(i,j)})$$
Novel Cross-Span Loss

• We modify the loss to account for incorrect spans in the tree.

\[
\text{cross}(i, j, t^*)
\]

• Indicates whether \((i, j)\) is crossing a span in the gold tree

\[
\Delta(t, t^*) = \sum_{(i, j, X) \in t} 1 \left( X \neq t^*_{(i, j)} \lor \text{cross}(i, j, t^*) \right)
\]

• Still decomposable over spans, so can be used to compare partial trees.
• Take the largest augmented loss value across all time steps.
• This is the Max-Violation, that we use to train.

Max-Violation Updates

Huang et. al. 2012
## Comparison with Baseline Chart Parser

<table>
<thead>
<tr>
<th>Model</th>
<th>Note</th>
<th>F1 (PTB test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stern et al. (2017a)</td>
<td>Baseline Chart Parser</td>
<td>91.79</td>
</tr>
<tr>
<td></td>
<td>+our cross-span loss</td>
<td>91.81</td>
</tr>
<tr>
<td>Our Work</td>
<td>Beam 15</td>
<td>91.84</td>
</tr>
<tr>
<td></td>
<td>Beam 20</td>
<td>91.97</td>
</tr>
</tbody>
</table>
## Comparison to Other Parsers

<table>
<thead>
<tr>
<th>Model</th>
<th>Note</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durett + Klein 2015</td>
<td></td>
<td>91.1</td>
</tr>
<tr>
<td>Cross + Huang 2016</td>
<td>Original Span Parser</td>
<td>91.3</td>
</tr>
<tr>
<td>Liu + Zhang 2016</td>
<td></td>
<td>91.7</td>
</tr>
<tr>
<td>Dyer et al. 2016</td>
<td>Discriminative</td>
<td>91.7</td>
</tr>
<tr>
<td>Stern et al. 2017a</td>
<td>Baseline Chart Parser</td>
<td>91.79</td>
</tr>
<tr>
<td>Stern et al. 2017c</td>
<td>Separate Decoding</td>
<td>92.56</td>
</tr>
<tr>
<td>Our Work</td>
<td>Beam 20</td>
<td>91.97</td>
</tr>
</tbody>
</table>

### PTB only, Single Model, End-to-End

<table>
<thead>
<tr>
<th>Model</th>
<th>Note</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vinyals et al. 2015</td>
<td>Ensemble</td>
<td>90.5</td>
</tr>
<tr>
<td>Dyer et al. 2016</td>
<td>Generative Reranking</td>
<td>93.3</td>
</tr>
<tr>
<td>Choe + Charniak 2016</td>
<td>Reranking</td>
<td>93.8</td>
</tr>
<tr>
<td>Fried et al. 2017</td>
<td>Ensemble Reranking</td>
<td>94.25</td>
</tr>
</tbody>
</table>

### Reranking, Ensemble, Extra Data
Conclusions

• Linear-Time, Span-Based Constituency Parsing with Dynamic Programming
• Cube-Pruning to speedup Incremental Parsing with Dynamic Programming
• Cross-Span Loss extension for improving Loss-Augmented Decoding
• Result: Faster and more accurate than cubic-time Chart Parsing
  • 2nd highest accuracy for single-model end-to-end systems trained on PTB only
    • Stern et al. 2017c is more accurate, but with separate decoding, and is much slower
  • After this ACL, definitely no longer true. (e.g. Joshi et al. 2018, Kitaev+Klein 2018)
  • But both are Span-Based Parsers and can be linearized in the same way!

\[
O(2^n) \rightarrow O(n^3) \rightarrow O(n^4) \sim O(nb^2) \sim O(nb \log b)
\]
Dezhong Deng for his theorem for predecessor states.
And his mathematical proofreading of the training sections.
Mitchell Stern for releasing his code and his suggestions.
Thank you! Questions?

Chart Parsing: $O(n^{2.26})$
Beam 20 No Cube-Pruning: $O(n^{1.26})$
Beam 20 Cube Pruned: $O(n^{1.08})$
Beam 5 Cube Pruned: $O(n^{0.97})$

Time (sec)
Sentence Length

Chart Parsing

Chart Parsing
This Work Beam 10

Time (sec)
Discourse Length (words)