Transition-Based Dependency Parsing with Stack Long Short-Term Memory

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, Noah A. Smith
Association for Computational Linguistics (ACL), 2015

Presented By: Lavisha Aggarwal (lavisha2)
Overview

• Parsing
• Transition based dependency Parsing
• Example
• Stack LSTM’s
• Dependency parser transitions and operations
• Token Embeddings
• Experimental details
• Data
• Chen and Manning (2014)
• Results
• Conclusion
• References
What is Parsing?

Analyzing a sentence by taking each word and determining the structure of the sentence.
Two types of Parsing

Dependency Parsing

Phase structure trees
Dependency Parsing

• Represent relations between words using directed edges from Head (H) to the Dependent (D). Eg. saw(H) → girl (D)

• Dependencies can be of 2 types:

I saw a girl with a telescope

Labeled

I saw a girl with a telescope

Unlabeled
Transition-based dependency Parsing

• The parser is made up of:
  1. Stack (S) of partially processed words (Initially contains the ROOT of sentence)
  2. Buffer (B) of remaining input words (Initially contains the entire input sentence)
  3. Set of dependency arcs (A) representing actions (Initially empty)

• Series of decisions that read words sequentially from a buffer and combine them incrementally into syntactic structures

<table>
<thead>
<tr>
<th>Transition</th>
<th>Stack</th>
<th>Buffer</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT</td>
<td>[ROOT]</td>
<td>[He has good control .]</td>
<td>\emptyset</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT He]</td>
<td>[has good control .]</td>
<td></td>
</tr>
<tr>
<td>LEFT-ARC(nsubj)</td>
<td>[ROOT He has]</td>
<td>[good control .]</td>
<td></td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT has]</td>
<td>[good control .]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ROOT has good]</td>
<td>[control .]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A \cup \text{nsubj(has,He)}</td>
</tr>
</tbody>
</table>
Arc-standard transition-based parser

- Notation: \( s_1 \) – Top element in stack, \( s_2 \) – 2\(^{nd} \) element from the top in Stack
- We can have 3 types of transition actions:
  1. **SHIFT**: Move one word from Buffer to Stack
  2. **LEFT-ARC (Reduce-Left)**: Add an arc \( s_2 \) \( \rightarrow \) \( s_1 \), Remove \( s_2 \) from Stack
  3. **RIGHT-ARC (Reduce Right)**: Add an arc \( s_2 \) \( \rightarrow \) \( s_1 \), Remove \( s_1 \) from Stack

<table>
<thead>
<tr>
<th>Transition</th>
<th>Stack</th>
<th>Buffer</th>
<th>( A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT</td>
<td>[ROOT]</td>
<td>[He has good control .]</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT He]</td>
<td>[has good control .]</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT He has]</td>
<td>[good control .]</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>LEFT-ARC (nsubj)</td>
<td>[ROOT has]</td>
<td>[good control .]</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT has good]</td>
<td>[control .]</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( A \cup \text{nsubj(has,He)} )</td>
</tr>
</tbody>
</table>
An Example

### Transition

<table>
<thead>
<tr>
<th>Transition</th>
<th>Stack</th>
<th>Buffer</th>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT</td>
<td>[ROOT]</td>
<td>[He has good control .]</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT He]</td>
<td>[has good control .]</td>
<td>$A \cup \text{nsubj(has,He)}$</td>
</tr>
<tr>
<td>LEFT-ARC (nsubj)</td>
<td>[ROOT has]</td>
<td>[good control .]</td>
<td></td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT has good]</td>
<td>[control .]</td>
<td></td>
</tr>
<tr>
<td>SHIFT</td>
<td>[ROOT has good control]</td>
<td>[.]</td>
<td>$A \cup \text{amod(control,good)}$</td>
</tr>
<tr>
<td>LEFT-ARC (amod)</td>
<td>[ROOT has control]</td>
<td>[.]</td>
<td>$A \cup \text{dobj(has,control)}$</td>
</tr>
<tr>
<td>RIGHT-ARC (dobj)</td>
<td>[ROOT has]</td>
<td>[.]</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>$A \cup \text{root(ROOT,has)}$</td>
</tr>
<tr>
<td>RIGHT-ARC (root)</td>
<td>[ROOT]</td>
<td>[.]</td>
<td></td>
</tr>
</tbody>
</table>

[Image credits: Chen and Manning (2014)]
Transition-based dependency parsing with Stack LSTM’s

• Predict the transition actions (Shift, Left-Arc or Right-Arc) at each time step
• Based on the state of the parser (contents of Stack, Buffer and Action-set)
• Use Long short-term memory models
• Goal – Learn a representation for the various parser components that helps us determine the sequence of actions
Long Short-term Memory (LSTM)

[Input gate ($i_t$), Output gate ($o_t$) and Forget gate ($f_t$), Cell state ($c_t$), Output ($y_t$)]

\[
\begin{align*}
  i_t &= \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i) \\
  f_t &= \sigma(W_{fx}x_t + W_{fh}h_{t-1} + W_{fc}c_{t-1} + b_f) \\
  c_t &= f_t \odot c_{t-1} + \\
  &\quad i_t \odot \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c), \\
  o_t &= \sigma(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_t + b_o) \\
  h_t &= o_t \odot \tanh(c_t). \\
  y_t &= g(h_t)
\end{align*}
\]
Stack LSTM

• Variation of recurrent neural networks with long short-term memory units
• Interpret LSTM as a stack that grows towards right (in the image below)
• At time $t$, the input $x_t$, cell states and gate values output $y_t$ are added as a new element to the stack – PUSH operation
• A Stack Pointer points to the TOP of stack
• For POP, simply move the stack pointer to the previous element
Dependency parser

- Buffer of words (B), Stack of syntactic trees (S) and Set of dependency actions (A)
- Each is represented by a Stack LSTM
- State of the parser at time t: $p_t$
Parser Transitions

- At each time-step, perform either of the 3 Actions
- REDUCE left and right linked with a relation (r) label (amod, nmod, obj, nsubj, dobj, etc.)
- If there are K relations, total number of possible actions: $2K+1$
- Store words $u,v$ along with their respective embeddings $u,v$ in S and B.
- For dependencies, store the head with the relation embedding $gr(u,v)$

<table>
<thead>
<tr>
<th>Stack$_t$</th>
<th>Buffer$_t$</th>
<th>Action</th>
<th>Stack$_{t+1}$</th>
<th>Buffer$_{t+1}$</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(u,u),(v,v),S$</td>
<td>B</td>
<td>REDUCE-LEFT$(r)$</td>
<td>$(gr(u,v),u),S$</td>
<td>B</td>
<td>$u \rightarrow v$</td>
</tr>
<tr>
<td>$(u,u),(v,v),S$</td>
<td>B</td>
<td>REDUCE-RIGHT$(r)$</td>
<td>$(gr(v,u),v),S$</td>
<td>B</td>
<td>$u \leftarrow v$</td>
</tr>
<tr>
<td>S</td>
<td>$(u,u),B$</td>
<td>SHIFT</td>
<td>$(u,u),S$</td>
<td>B</td>
<td>—</td>
</tr>
</tbody>
</table>
Parser Operation I

• The state of the parser $p_t$ at time $t$ depends on the stack LSTM encodings of buffer $B$ ($b_t$), stack $S$ ($s_t$) and action ($a_t$)

$$p_t = \max \{0, W[s_t; b_t; a_t] + d\}$$

• $W$ is a learned parameter matrix and $d$ is a bias term
Parser Operation II

• For each possible action $z_t$ at time $t$, the likelihood is determined by

$$p(z_t \mid \mathbf{p}_t) = \frac{\exp(g_{z_t}^T \mathbf{p}_t + q_{z_t})}{\sum_{z' \in A(S,B)} \exp(g_{z'}^T \mathbf{p}_t + q_{z'})}.$$  

• $g_z$ represents embedding of parser action $z$, $q_z$ is the bias for action $z$  
• $A(S,B)$ represents the possible actions given stack $S$ and buffer $B$  
• Probability of a sequence of parse actions $z$

$$p(z \mid w) = \prod_{t=1}^{\left| z \right|} p(z_t \mid \mathbf{p}_t)$$

• $w$ corresponds to the set of words of the given sentence  
• Goal – Find the sequence of actions that maximize this
Token Embeddings

- Each input token $x_t$ is a concatenation of 3 vectors:
  1. Learned vector representation ($w$)
  2. Neural language model representation ($w_{LM}$)
  3. POS tag representation ($t$)

$$x = \max \{0, V[w; \tilde{w}_{LM}; t] + b\}$$

- $V$ is a linear map and $b$ is the bias term
- Syntactic trees represented as a composition function $c$ in terms of the Syntactic head ($h$), dependent ($d$) and relation ($r$)

$$c = \tanh (U[h; d; r] + e)$$

- $U$ is a parameter matrix and $e$ is a constant bias term
Experiment details

• The model is trained to learn the representations of the parser states
• Goal - Maximize the likelihood of the correct sequence of parse actions
• Training time – 8 to 12 hours
• Stochastic gradient descent with standard backpropogation
• Matrix, vector parameters initialized with uniform samples in $\pm \sqrt{6/(r + c)}$ (r rows, c columns )
• Dimensionality
  ▪ LSTM hidden state size - 100
  ▪ Parser actions dimensions – 16
  ▪ Output embedding size – 20
  ▪ Pretrained word embeddings - 100 for English and 80 for Chinese
  ▪ Learned word embedding – 32
  ▪ POS tag embeddings – 12
Training Data

• English
  ▪ Stanford Dependency treebank
  ▪ POS tags – Stanford Tagger (Accuracy – 97.3%)
  ▪ Language model embeddings – AFP portion of English Gigaword corpus

• Chinese
  ▪ Penn Chinese Treebank
  ▪ Gold POS tags
  ▪ Language model embeddings - Chinese Gigaword corpus
Experimental configurations

Testing was done on 5 experimental configurations:

- Full stack LSTM parsing (S-LSTM)
- Without POS tags (-POS)
- Without pre-trained language model embeddings (-pre-training)
- Instead of composed representations only head words used (-composition)
- Full parsing model with RNN instead of LSTM (S-RNN)

Compared the model with Chen and Manning (2014)
Chen and Manning (EMNLP, 2014) 
(A Fast and Accurate Dependency Parser using Neural Networks)

- Feed-forward Neural network architecture with 1 hidden layer (h)
- Cube activation function
- Features used - $s_1$, $s_2$, $s_3$, $b_1$, $b_2$, $b_3$
  - $lc_1(s_i)$, $lc_2(s_i)$, $rc_1(s_i)$, $rc_2(s_i)$, $i=1,2$ [lc - leftchild, rc - rightchild]
  - $lc_1(lc_1(s_i))$, $rc_1(rc_1(s_i))$, $i=1,2$
Results

• Report Unlabeled attachment scores (UAS) and Labeled attachment scores (LAS)

<table>
<thead>
<tr>
<th>Development</th>
<th>Test</th>
<th>Dev. set</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UAS</td>
<td>LAS</td>
<td>UAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S-LSTM</td>
<td>93.2</td>
<td>90.9</td>
<td>93.1</td>
</tr>
<tr>
<td>-POS</td>
<td>93.1</td>
<td>90.4</td>
<td>92.7</td>
</tr>
<tr>
<td>-pretraining</td>
<td>92.7</td>
<td>90.4</td>
<td>92.4</td>
</tr>
<tr>
<td>-composition</td>
<td>92.7</td>
<td>89.9</td>
<td>92.2</td>
</tr>
<tr>
<td>S-RNN</td>
<td>92.8</td>
<td>90.4</td>
<td>92.3</td>
</tr>
<tr>
<td>C&amp;M (2014)</td>
<td>92.2</td>
<td>89.7</td>
<td>91.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S-LSTM</td>
<td>87.2</td>
<td>85.9</td>
<td>87.2</td>
</tr>
<tr>
<td>-composition</td>
<td>85.8</td>
<td>84.0</td>
<td>85.3</td>
</tr>
<tr>
<td>-pretraining</td>
<td>86.3</td>
<td>84.7</td>
<td>85.7</td>
</tr>
<tr>
<td>-POS</td>
<td>82.8</td>
<td>79.8</td>
<td>82.2</td>
</tr>
<tr>
<td>S-RNN</td>
<td>86.3</td>
<td>84.7</td>
<td>86.1</td>
</tr>
<tr>
<td>C&amp;M (2014)</td>
<td>84.0</td>
<td>82.4</td>
<td>83.9</td>
</tr>
</tbody>
</table>

Table 1: English parsing results (SD)

Table 2: Chinese parsing results (CTB5)

• POS, Composition different effect in English and Chinese
• RNN and Chen&Manning lack Stack LSTM
Conclusion

• All configurations except –POS for Chinese, better than Chen and Manning
• Composition function seems to be the most important factor as the accuracy drop is largest in -composition
• Pre-training and parts of speech tagging follow as the next important things
• In English, POS do not play much role
• But in Chinese POS play a significant role
• LSTM’s outperform RNN’s but they are still better than Chen and manning
• Stack memory offers intriguing possibilities
• Achieve parsing and training in linear time (length of the input sentence)
• Beam search had minimal impact on scores
References

- Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, Noah A. Smith. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In ACL. 2015
- Jurafsky and Martin. Dependency Parsing. Speech and Language Processing, Chapter 14, Stanford
- Julia Hockenmaier. Dependency Parsing Lecture 8, Natural Language Processing CS447, UIUC
- Graham Neubig. Neural Networks for NLP Transition-based Parsing with Neural Nets. CS11-747, CMU