A Fast and Accurate Dependency Parser using Neural Networks

Danqi Chen and Christopher Manning
Stanford University
October 27, 2014
Dependency Parsing

He has good control.

Goal: **accurate** and **fast** parsing
Our Work

- A neural network based dependency parser!

(Zhang and Nirve 2011, Martins et al 2013)
Our Work

• A neural network based dependency parser!

Parsing on English Penn Treebank (§23):

Unlabeled attachment score (UAS) sent / s

Transition-based
Our Work

• A neural network based dependency parser!

Parsing on English Penn Treebank (§23):

<table>
<thead>
<tr>
<th>Transition-based</th>
<th>Unlabeled attachment score (UAS)</th>
<th>sent / s</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaltParser (greedy)</td>
<td>89.9</td>
<td>560</td>
</tr>
</tbody>
</table>

(Zhang and Nirve 2011, Martins et al 2013)
Our Work

- A neural network based dependency parser!

Parsing on English Penn Treebank (§23):

<table>
<thead>
<tr>
<th>Unlabeled attachment score (UAS)</th>
<th>sent / s</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaltParser (greedy)</td>
<td>89.9</td>
</tr>
<tr>
<td>Zpar: beam = 64</td>
<td>92.9*</td>
</tr>
</tbody>
</table>
Our Work

- A neural network based dependency parser!

Parsing on English Penn Treebank (§23):

<table>
<thead>
<tr>
<th>Transition-based</th>
<th>MaltParser (greedy)</th>
<th>89.9</th>
<th>560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zpar: beam = 64</td>
<td>92.9*</td>
<td>29*</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph-based</th>
<th>MSTParser</th>
<th>92.0</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>TurboParser</td>
<td>93.1*</td>
<td>31*</td>
<td></td>
</tr>
</tbody>
</table>
## Our Work

- A neural network based dependency parser!

### Parsing on English Penn Treebank (§23):

<table>
<thead>
<tr>
<th>Method</th>
<th>UAS</th>
<th>sent / s</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaltParser (greedy)</td>
<td>89.9</td>
<td>560</td>
</tr>
<tr>
<td>Our Parser (greedy)</td>
<td>92.0</td>
<td>1013</td>
</tr>
<tr>
<td>Zpar: beam = 64</td>
<td>92.9*</td>
<td>29*</td>
</tr>
<tr>
<td>MSTParser</td>
<td>92.0</td>
<td>12</td>
</tr>
<tr>
<td>TurboParser</td>
<td>93.1*</td>
<td>31*</td>
</tr>
</tbody>
</table>

*(Zhang and Nirve 2011, Martins et al 2013)*
Outline

• Background & Motivation
• Model
• Experiments
• Analysis
Greedy Transition-based Parsing

\[ C_1 \rightarrow C_2 \rightarrow \cdots \rightarrow C_{m-1} \rightarrow C_m \]

(Nivre et al, 2004)
Greedy Transition-based Parsing

- A configuration = a stack, a buffer and some dependency arcs

Motivation | Model | Experiments | Analysis

(Nivre et al, 2004)
Greedy Transition-based Parsing

- A configuration = a stack, a buffer and some dependency arcs

Motivation | Model | Experiments | Analysis

(Nivre et al, 2004)
**Greedy Transition-based Parsing**

- A configuration = a stack, a buffer and some dependency arcs

- We employ the **arc-standard** system.
LEFT-ARC (I)

Motivation | Model | Experiments | Analysis
RIGHT-ARC (I)

Motivation | Model | Experiments | Analysis
A Fast and Accurate Dependency Parser using Neural Networks
Greedy Transition-based Parsing

A Fast and Accurate Dependency Parser using Neural Networks
Traditional Features

- Feature templates: usually a combination of 1 ~ 3 elements from the configuration.

- Stack:
  - ROOT
  - has_VBZ
  - good_JJ
  - nsubj
  - He_PRP

- Buffer:
  - control_NN

- Binary, sparse
  - dim = $10^6 \sim 10^7$

---

Motivation | Model | Experiments | Analysis
Traditional Features

```
<table>
<thead>
<tr>
<th>Stack</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT</td>
<td>control_NN</td>
</tr>
<tr>
<td>has_VBZ</td>
<td></td>
</tr>
<tr>
<td>good_JJ</td>
<td></td>
</tr>
<tr>
<td>He_PRP</td>
<td></td>
</tr>
<tr>
<td>nsubj</td>
<td></td>
</tr>
</tbody>
</table>
```

Indicates features:

- $s_2.w = \text{has} \land s_2.t = \text{VBZ}$
- $s_1.w = \text{good} \land s_1.t = \text{JJ} \land b_1.w = \text{control}$
- $lc(s_2).t = \text{PRP} \land s_2.t = \text{VBZ} \land s_1.t = \text{JJ}$
- $lc(s_2).w = \text{He} \land lc(s_2).l = \text{nsubj} \land s_2.w = \text{has}$

Binary, sparse features:

- Dimension: $10^6 \sim 10^7$

Motivation | Model | Experiments | Analysis
Traditional Features

Motivation | Model | Experiments | Analysis

Stack

(ROOT, has_VBZ, good_JJ, nsbuj)

Buffer

(control_NN)

Binary, sparse

$dim = 10^6 \sim 10^7$

Indicator features

$s_2.w = \text{has} \land s_2.t = \text{VBZ}$

$s_1.w = \text{good} \land s_1.t = \text{JJ} \land b_1.w = \text{control}$

$lc(s_2).t = \text{PRP} \land s_2.t = \text{VBZ} \land s_1.t = \text{JJ}$

$lc(s_2).w = \text{He} \land lc(s_2).l = \text{nsbuj} \land s_2.w = \text{has}$
Traditional Features

Motivation | Model | Experiments | Analysis

Stack

ROOT has_VBZ good_JJ
He_PRP

nssubj

Buffer

control_NN

Stack Bu↵er

Correct transition:

Shift

S2.w = has \land S2.t = VBZ
S1.w = good \land S1.t = JJ \land b1.w = control
lc(s2).t = PRP \land s2.t = VBZ \land s1.t = JJ
lc(s2).w = He \land lc(s2).l = nsubj \land s2.w = has

word

part-of-speech tag

nssubj

dep. label

binary, sparse

dim = 10^6 \sim 10^7

[0 0 0 1 0 0 1 0 ... 0 0 1 0]
Traditional Features

Stack

ROOT has_VBZ good_JJ
He_PRP

nsubj

Buffer
control_NN...

binary, sparse
dim $=10^6 \sim 10^7$

Indicator features

$s_2.w = \text{has} \land s_2.t = \text{VBZ}$
$s_1.w = \text{good} \land s_1.t = \text{JJ} \land b_1.w = \text{control}$
$lc(s_2).t = \text{PRP} \land s_2.t = \text{VBZ} \land s_1.t = \text{JJ}$
$lc(s_2).w = \text{He} \land lc(s_2).l = \text{nsubj} \land s_2.w = \text{has}$

leftmost child
Indicator Features Revisited

\[ s_2.w = \text{has} \land s_2.t = \text{VBZ} \]
\[ s_1.w = \text{good} \land s_1.t = \text{JJ} \land b_1.w = \text{control} \]
\[ lc(s_2).t = \text{PRP} \land s_2.t = \text{VBZ} \land s_1.t = \text{JJ} \]
\[ lc(s_2).w = \text{He} \land lc(s_2).l = \text{nsubj} \land s_2.w = \text{has} \]
Indicator Features Revisited

- Problem #1: sparse

- lexicalized features
- high-order interaction features

\[ s_2.w = \text{has} \land s_2.t = \text{VBZ} \]
\[ s_1.w = \text{good} \land s_1.t = \text{JJ} \land b_1.w = \text{control} \]
\[ \text{lc}(s_2).t = \text{PRP} \land s_2.t = \text{VBZ} \land s_1.t = \text{JJ} \]
\[ \text{lc}(s_2).w = \text{He} \land \text{lc}(s_2).l = \text{nsubj} \land s_2.w = \text{has} \]
Indicator Features Revisited

- Problem #1: sparse
  - lexicalized features
  - high-order interaction features

\[
\begin{align*}
  s_2.w &= \text{has} \land s_2.t = \text{VBZ} \\
  s_1.w &= \text{good} \land s_1.t = \text{JJ} \land b_1.w = \text{control} \\
  lc(s_2).t &= \text{PRP} \land s_2.t = \text{VBZ} \land s_1.t = \text{JJ} \\
  lc(s_2).w &= \text{He} \land lc(s_2).l = \text{nsubj} \land s_2.w = \text{has}
\end{align*}
\]
Indicator Features Revisited

- Problem #1: sparse

\[ s_2.w = \text{has} \land s_2.t = \text{VBZ} \]
\[ s_1.w = \text{good} \land s_1.t = \text{JJ} \land b_1.w = \text{control} \]
\[ lc(s_2).t = \text{PRP} \land s_2.t = \text{VBZ} \land s_1.t = \text{JJ} \]
\[ lc(s_2).w = \text{He} \land lc(s_2).l = \text{nsubj} \land s_2.w = \text{has} \]
Indicator Features Revisited

• Problem #1: sparse
• Problem #2: incomplete

Unavoidable in hand-crafted feature templates.

\[
\begin{align*}
    s_2.w &= \text{has} \land s_2.t = \text{VBZ} \\
    s_1.w &= \text{good} \land s_1.t = \text{JJ} \land b_1.w = \text{control} \\
    lc(s_2).t &= \text{PRP} \land s_2.t = \text{VBZ} \land s_1.t = \text{JJ} \\
    lc(s_2).w &= \text{He} \land lc(s_2).l = \text{nsubj} \land s_2.w = \text{has}
\end{align*}
\]
Indicator Features Revisited

- Problem #1: sparse
- Problem #2: incomplete

\[
s_{2}.w = \text{has} \land s_{2}.t = \text{VBZ}
\]
\[
s_{1}.w = \text{good} \land s_{1}.t = \text{JJ} \land b_{1}.w = \text{control}
\]
\[
lc(s_{2}).t = \text{PRP} \land s_{2}.t = \text{VBZ} \land s_{1}.t = \text{JJ}
\]
\[
lc(s_{2}).w = \text{He} \land lc(s_{2}).l = \text{nsubj} \land s_{2}.w = \text{has}
\]
Indicator Features Revisited

• Problem #1: sparse
• Problem #2: incomplete
• Problem #3: computationally expensive

More than 95% of parsing time is consumed by feature computation.

\[
\begin{align*}
  s_2.w &= \text{has} \land s_2.t = \text{VBZ} \\
  s_1.w &= \text{good} \land s_1.t = \text{JJ} \land b_1.w = \text{control} \\
  lc(s_2).t &= \text{PRP} \land s_2.t = \text{VBZ} \land s_1.t = \text{JJ} \\
  lc(s_2).w &= \text{He} \land lc(s_2).l = \text{nsubj} \land s_2.w = \text{has}
\end{align*}
\]
Indicator Features Revisited

Motivation

| Model | Experiments | Analysis |

Our Solution: Neural Networks!
Learn a dense and compact feature representation
The Challenge

- How to encode all the available information?
- How to model high-order features?

Motivation | Model | Experiments | Analysis

Stanford University NLP

A Fast and Accurate Dependency Parser using Neural Networks
Distributed Representations
Distributed Representations

- We represent each word as a d-dimensional dense vector (i.e., word embeddings).
  - Similar words expect to have close vectors.
Distributed Representations

- We represent each word as a d-dimensional dense vector (i.e., word embeddings).
  - Similar words expect to have close vectors.
- Meanwhile, part-of-speech tags and dependency labels are also represented as d-dimensional vectors.
  - POS and dependency embeddings.
  - The smaller discrete sets also exhibit many semantical similarities.
Distributed Representations

• We represent each word as a \( d \)-dimensional dense vector (i.e., word embeddings).
  • Similar words expect to have close vectors.

• Meanwhile, part-of-speech tags and dependency labels are also represented as \( d \)-dimensional vectors.
  • POS and dependency embeddings.
  • The smaller discrete sets also exhibit many semantical similarities.

\[
\text{NNS} \text{ (plural noun) should be close to } \text{NN} \text{ (singular noun).} \\
\text{num} \text{ (numerical modifier) should be close to } \text{amod} \text{ (adjective modifier).}
\]
Extracting Tokens from Configuration

- We extract a set of tokens based on the positions:

```
Stack
ROOT has_VBZ good_JJ
    nsubj
He_PRP
```

```
Buffer
color =_NN ...
```
- We extract a set of tokens based on the positions:
Extracting Tokens from Configuration

- We extract a set of tokens based on the positions:

```
Stack
ROOT has_VBZ good_JJ
He_PRP nsubj

Buffer
control_NN ...
```

```
s1
s2
b1
lc(s1)
rc(s1)
lc(s2)
rc(s2)
...
```

```
word
good has control
He

POS
JJ
VBZ
NN

dep.
∅ + ∅ + ∅ + nsubj
∅ + ∅ + ∅...
Model Architecture
Model Architecture

Input layer

Stack

Buffer

ROOT  has_VBZ  good_JJ

He_PRP

nsubj

control_NN
Model Architecture

Motivation | Model | Experiments | Analysis

Hidden layer

Input layer

Stack

Buffer

ROOT  has_VBZ  good_JJ

He_PRP

control_NN

A Fast and Accurate Dependency Parser using Neural Networks
Model Architecture

Cube activation function: $g(x) = x^3$
A Fast and Accurate Dependency Parser using Neural Networks

Model Architecture

Motivation | Model | Experiments | Analysis

Output layer

Hidden layer

Input layer

Softmax probabilities

Stack

Buffer

ROOT has_VBZ good_JJ

He_PRP

control_NN
Cube Activation Function

$$g(w_1x_1 + \ldots + w_mx_m + b) = \sum_{i,j,k} (w_iw_jw_k)x_ix_jx_k + \sum_{i,j} b(w_iw_j)x_ix_j \ldots$$

Better capture the **interaction** terms!
Training

• Generating training examples using a fixed oracle.
• **Training objective:** cross entropy loss
• Back-propagation to all embeddings.
• **Initialization:**
  • Word embeddings from **pre-trained** word vectors.
  • Random initialization for others.
A Fast and Accurate Dependency Parser using Neural Networks

**Parsing Speed-up**

- **Pre-computation trick:**

  - If we have seen \((s_1, \text{good})\) many times in training set, we can pre-compute matrix multiplications before parsing — reducing multiplications to additions.
  - 8 ~ 10 times faster.
Indicator vs. Dense Features

• Problem #1: sparse

  Distributed representations can capture similarities.
Indicator vs. Dense Features

- **Problem #1: sparse**
  
  Distributed representations can capture similarities.

- **Problem #2: incomplete**
  
  We don’t need to enumerate the combinations. Cube non-linearity can learn combinations automatically.
Indicator vs. Dense Features

- **Problem #1: sparse**
  
  Distributed representations can capture similarities.

- **Problem #2: incomplete**
  
  We don’t need to enumerate the combinations. Cube non-linearity can learn combinations automatically.

- **Problem #3: computationally expensive**
  
  String concatenation + look-up in a big table $\rightarrow$ matrix operations. Pre-computation trick can speed up.
• Datasets
  • English Penn Treebank (PTB)
  • Chinese Penn Treebank (CTB)

• Representations
  • CoNLL representations (CD) for PTB and CTB
  • Stanford Dependencies V3.3.0 (SD) for PTB

• Part-of-speech tags:
  • Stanford POS tagger for PTB (97.3% accuracy)
  • Gold tags for CTB
Details

- Embedding size = 50
- Hidden size = 200
- Use mini-batched AdaGrad for optimization ($\alpha = 0.01$)
- Use 0.5 dropout on hidden layer.

- Pre-trained word vectors:
  - C & W for English
  - Word2vec for Chinese

- We use a rich set of 18 tokens from the configuration.
Baselines

- **Standard / eager**: our own implemented perceptron-based greedy parsers using arc-standard or arc-eager system, with a rich feature set from (Zhang and Nivre, 2011).

- **MaltParser**
  - two algorithms `stackproj` and `nivreeager`.

- **MSTParser**
Unlabeled Attachment Score (UAS)

- Standard / eager
- Malt (stackproj / nirveeager)
- MST
- Our Parser

<table>
<thead>
<tr>
<th>Value Axis</th>
<th>PTB: CoNLL</th>
<th>PTB: Stanford</th>
<th>CTB</th>
</tr>
</thead>
<tbody>
<tr>
<td>89.9</td>
<td>90.1</td>
<td>92</td>
<td>83</td>
</tr>
<tr>
<td>89.6</td>
<td>90.4</td>
<td>91.8</td>
<td>83.9</td>
</tr>
<tr>
<td>89.4</td>
<td>90.7</td>
<td>92</td>
<td>82.7</td>
</tr>
<tr>
<td>90.1</td>
<td>90.7</td>
<td>91.8</td>
<td>82.4</td>
</tr>
<tr>
<td>90.4</td>
<td>90.7</td>
<td>91.8</td>
<td>83</td>
</tr>
</tbody>
</table>
Unlabeled Attachment Score (UAS)

<table>
<thead>
<tr>
<th>Value Axis</th>
<th>82</th>
<th>84.5</th>
<th>87</th>
<th>89.5</th>
<th>92</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTB: CoNLL</td>
<td>89.9</td>
<td>90.1</td>
<td>92</td>
<td>92</td>
<td>91.8</td>
</tr>
<tr>
<td>PTB: Stanford</td>
<td>89.6</td>
<td>89.4</td>
<td>90.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTB</td>
<td>82.7</td>
<td>82.4</td>
<td>83</td>
<td></td>
<td>83.9</td>
</tr>
</tbody>
</table>

Compared with greedy parsers,
PTB: > 2.0%
CTB: >1.2%
Labeled Attachment Score (LAS)

- Standard / eager
- Malt (stackproj / nirveeager)
- MST
- Our Parser

<table>
<thead>
<tr>
<th></th>
<th>PTB: CoNLL</th>
<th>PTB: Stanford</th>
<th>CTB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard / eager</td>
<td>88.6</td>
<td>87.4</td>
<td>81.2</td>
</tr>
<tr>
<td>Malt (stackproj / nirveeager)</td>
<td>88.7</td>
<td>86.9</td>
<td>80.6</td>
</tr>
<tr>
<td>MST</td>
<td>90.5</td>
<td>87.6</td>
<td>81.2</td>
</tr>
<tr>
<td>Our Parser</td>
<td>90.7</td>
<td>89.6</td>
<td>82.4</td>
</tr>
</tbody>
</table>
Parsing Speed (sent/s)

- Standard / eager
- Malt (stackproj / nirveeager)
- MST
- Our Parser

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Standard / eager</th>
<th>Malt (stackproj / nirveeager)</th>
<th>MST</th>
<th>Our Parser</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTB: CoNLL</td>
<td>560</td>
<td>1,013</td>
<td>63</td>
<td>12</td>
</tr>
<tr>
<td>PTB: Stanford</td>
<td>469</td>
<td>654</td>
<td>34</td>
<td>10</td>
</tr>
<tr>
<td>CTB</td>
<td>420</td>
<td>936</td>
<td>80</td>
<td>6</td>
</tr>
</tbody>
</table>

A Fast and Accurate Dependency Parser using Neural Networks
Cube Activation Function

- identity
- sigmoid
- tanh
- cube

Motivation | Model | Experiments | Analysis

Cube:
+0.8% ~ 1.2%

A Fast and Accurate Dependency Parser using Neural Networks
Pre-trained Word Vectors

- **PTB**: +0.7%
- **CTB**: +1.7%

Bar chart showing:
- PTB: CD (random: 90, pre-trained: 91)
- PTB: SD (random: 86, pre-trained: 87)
- CTB (random: 81, pre-trained: 81)
POS / Dependency Embeddings

- POS embeddings help a lot:
  - PTB: +1.7%
  - CTB: +10.2%

- Slight gain from dependency embeddings.
POS Embeddings

A Fast and Accurate Dependency Parser using Neural Networks

(van der Maaten and Hinton 2008)
POS Embeddings
Dependency Embeddings

Motivation | Model | Experiments | Analysis
Conclusion

• **Summary**
  - Presented a state-of-the-art greedy parser using NNs.
  - Excellent accuracy and speed.
  - Introduced POS / dep. embeddings, and cube activation function.

• **Future work**
  - Beam search
  - Dynamic oracle
  - Richer features (lemma, morph, distance, etc).
  - Better representation for modeling interactions

(Goldberg et al 2014)
• Code is available!

• Try fast dependency parsing in Stanford CoreNLP v3.5.0,
  • annotators: tokenize, ssplit, pos, depparse

• Or check out full training / testing code at:
  • http://nlp.stanford.edu/software/software/nndep.shtml

• Contact: danqi@cs.stanford.edu