Neural Networks in Structured Prediction

November 19, 2015
Last Time

• We talked about using non-structured neural networks to solve structured problems

• Intuition: neural nets are powerful learners—maybe we don’t need to model statistical dependencies among output variables?

• Some support for this: POS tagging results…
Goals for Today

• Neural networks in structured prediction:
  • Option 1: locally nonlinear factors in globally linear models
  • Option 2: operation sequence models
  • Option 3: global, nonlinear structured models [speculative]
Locally Nonlinear Models

\[ score(x, y) = \sum_{i=1}^{n} w^T f(y_{i-1}, y_i, x) \]

\[ = w^T \sum_{i=1}^{n} f(y_{i-1}, y_i, x) \]
Locally Nonlinear Models

\[
\text{score}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \mathbf{w}^\top f(y_{i-1}, y_i, \mathbf{x})
\]

\[
= \mathbf{w}^\top \sum_{i=1}^{n} f(y_{i-1}, y_i, \mathbf{x})
\]

\[
= \mathbf{w}^\top \sum_{i=1}^{n} \text{NN}(y_{i-1}, y_i, \mathbf{x})
\]
Local Nonlinear Model

• Neural net returns a vector (a feature vector!) for each local factor

• We still get fast, global decoding using standard linear models

• Feature induction operates locally

• Best of both worlds?
CRF

Peng, Bo, Xu (NIPS 2009)
Protein secondary structure prediction (Peng et al., 2009)

<table>
<thead>
<tr>
<th>Methods</th>
<th>Q3(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional Random Fields</td>
<td>72.9</td>
</tr>
<tr>
<td>SVM-struct (Linear Kernel)</td>
<td>73.1</td>
</tr>
<tr>
<td>Neural Networks (one hidden layer)</td>
<td>72</td>
</tr>
<tr>
<td>Neural Networks (two hidden layer)</td>
<td>74</td>
</tr>
<tr>
<td>Semimarkov HMM</td>
<td>72.8</td>
</tr>
<tr>
<td>SVMpro</td>
<td>73.5</td>
</tr>
<tr>
<td>SVMpsi</td>
<td>76.6</td>
</tr>
<tr>
<td>PSIPRED</td>
<td>76</td>
</tr>
<tr>
<td>YASSPP</td>
<td>77.8</td>
</tr>
<tr>
<td>SPINE*</td>
<td>76.8</td>
</tr>
<tr>
<td>Conditional Neural Fields</td>
<td>80.1 ±0.3</td>
</tr>
<tr>
<td>Conditional Neural Fields*</td>
<td>80.5 ±0.3</td>
</tr>
</tbody>
</table>
### Protein secondary structure prediction (Peng et al., 2009)

<table>
<thead>
<tr>
<th>Methods</th>
<th>Q3(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional Random Fields</td>
<td>72.9</td>
</tr>
<tr>
<td>SVM-struct (Linear Kernel)</td>
<td>73.1</td>
</tr>
<tr>
<td>Neural Networks (one hidden layer)</td>
<td>72</td>
</tr>
<tr>
<td>Neural Networks (two hidden layer)</td>
<td>74</td>
</tr>
<tr>
<td>Semimarkov HMM</td>
<td>72.8</td>
</tr>
<tr>
<td>SVMpro</td>
<td>73.5</td>
</tr>
<tr>
<td>SVMpsi</td>
<td>76.6</td>
</tr>
<tr>
<td>PSIPRED</td>
<td>76</td>
</tr>
<tr>
<td>YASSPP</td>
<td>77.8</td>
</tr>
<tr>
<td>SPINE*</td>
<td>76.8</td>
</tr>
<tr>
<td>Conditional Neural Fields</td>
<td>80.1±0.3</td>
</tr>
<tr>
<td>Conditional Neural Fields*</td>
<td>80.5±0.3</td>
</tr>
</tbody>
</table>

### Constituency parsing (Durrett & Klein, 2015)

<table>
<thead>
<tr>
<th></th>
<th>Arabic</th>
<th>Basque</th>
<th>French</th>
<th>German</th>
<th>Hebrew</th>
<th>Hungarian</th>
<th>Korean</th>
<th>Polish</th>
<th>Swedish</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev, all lengths</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hall et al. (2014)</td>
<td>78.89</td>
<td>83.74</td>
<td>79.40</td>
<td>83.28</td>
<td>88.06</td>
<td>87.44</td>
<td>91.10</td>
<td>75.95</td>
<td>83.30</td>
<td>83.30</td>
</tr>
<tr>
<td>This work*</td>
<td>80.68</td>
<td>84.37</td>
<td>80.65</td>
<td>85.25</td>
<td>89.37</td>
<td>89.46</td>
<td>92.10</td>
<td>77.93</td>
<td>84.68</td>
<td></td>
</tr>
<tr>
<td>Test, all lengths</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Berkeley</td>
<td>79.19</td>
<td>70.50</td>
<td>80.38</td>
<td>78.30</td>
<td>86.96</td>
<td>81.62</td>
<td>71.42</td>
<td>79.23</td>
<td>79.18</td>
<td>78.53</td>
</tr>
<tr>
<td>Berkeley-Tags</td>
<td>78.66</td>
<td>74.74</td>
<td>79.76</td>
<td>78.28</td>
<td>85.42</td>
<td>85.22</td>
<td>78.56</td>
<td>86.75</td>
<td>80.64</td>
<td>80.89</td>
</tr>
<tr>
<td>Crabbé and Seddah (2014)</td>
<td>77.66</td>
<td>85.35</td>
<td>79.68</td>
<td>77.15</td>
<td>86.19</td>
<td>87.51</td>
<td>79.35</td>
<td>91.60</td>
<td>82.72</td>
<td>83.02</td>
</tr>
<tr>
<td>Hall et al. (2014)</td>
<td>78.75</td>
<td>83.39</td>
<td>79.70</td>
<td>78.43</td>
<td>87.18</td>
<td>88.25</td>
<td>80.18</td>
<td>90.66</td>
<td>82.00</td>
<td>83.17</td>
</tr>
<tr>
<td>This work*</td>
<td>80.24</td>
<td>85.41</td>
<td>81.25</td>
<td>80.95</td>
<td>88.61</td>
<td>90.66</td>
<td>82.23</td>
<td>92.97</td>
<td>83.45</td>
<td>85.08</td>
</tr>
</tbody>
</table>
Operation Sequence Models
while $y_t \neq \text{STOP}$

$$h_t = f(h_{t-1}, x_t)$$

$$y_t \sim g(h_t)$$

$$t \leftarrow t + 1$$

What is the probability of a sequence $y$?

$$p(y) = \prod_i p(y_i \mid y_{<i})$$
RNN Language Models

while $y_t \neq \text{STOP}$

$$h_t = f(h_{t-1}, x_t)$$

$$y_t \sim g(h_t)$$

$$t \leftarrow t + 1$$

What is the probability of a sequence $y$?

$$p(y) = \prod_{i} p(y_i \mid y_{<i})$$
RNN Language Models

while $y_t \neq $STOP

$h_t = f(h_{t-1}, x_t)$

$y_t \sim g(h_t)$

$t \leftarrow t + 1$

What is the probability of a sequence $y$?

$$p(y) = \prod_{i} p(y_i \mid y_{<i})$$
RNN Language Models

while $y_t \neq \text{STOP}$

$h_t = f(h_{t-1}, x_t)$

$y_t \sim g(h_t)$

$t \leftarrow t + 1$

What is the probability of a sequence $\mathbf{y}$?

$$p(\mathbf{y}) = \prod_i p(y_i \mid \mathbf{y}_{<i})$$
RNN Language Models

while $y_t \neq \text{STOP}$

\[
\begin{align*}
  h_t &= f(h_{t-1}, x_t) \\
  y_t &\sim g(h_t) \\
  t &\leftarrow t + 1
\end{align*}
\]

What is the probability of a sequence $y$?

\[
p(y) = \prod_i p(y_i \mid y_{<i})
\]
RNN Language Models

while \( y_t \neq \text{STOP} \)
\[
\begin{align*}
    h_t &= f(h_{t-1}, x_t) \\
    y_t &\sim g(h_t) \\
    t &\leftarrow t + 1
\end{align*}
\]

What is the probability of a sequence \( y \)?
\[
p(y) = \prod_i p(y_i \mid y_{<i})
\]
RNN Language Models

while $y_t \neq \text{STOP}$

$$h_t = f(h_{t-1}, x_t)$$

$$y_t \sim g(h_t)$$

$$t \leftarrow t + 1$$

What is the probability of a sequence $y$?

$$p(y) = \prod_{i} p(y_i \mid y_{<i})$$
while $y_t \neq \text{STOP}$

$$h_t = f(h_{t-1}, x_t)$$

$$y_t \sim g(h_t)$$

$$t \leftarrow t + 1$$

What is the probability of a sequence $y$?

$$p(y) = \prod_{i} p(y_i \mid y_{<i})$$
RNNLMs for Structured Prediction

• Intuition \[ p(y) = \prod_i p(y_i \mid y_{<i}) \]
RNNLMs for Structured Prediction

• Intuition

\[ p(y) = \prod_i p(y_i | y_{<i}) \]

\[ p(y | x) = \prod_i p(y_i | y_{<i}, x) \]
Transition-Based Models

• Break the structure you want to build down into a sequence of structure-building operations (or transitions)

• sequence tagging can be done with a single operation: ReadAndLabel(X) - remove the next input symbol and label it with an X

• more complicated structures (trees, graphs) require auxiliary data structures that are manipulated (more later)
Dependency parsing

I saw her duck
Dependency parsing

I saw her duck

ROOT

I saw her duck
Transition-based parsing

- Build trees by pushing words ("shift") onto a stack and combing elements at the top of the stack into a syntactic constituent ("reduce")

- *Given current stack and buffer of unprocessed words, what action should the algorithm take?*

- Widely used

  - Good accuracy

  - $O(n)$ runtime [much faster than other parsing algos]
Transition-based parsing

- There are actually perhaps 5 or 6 different “transition sets” for transition-based parsing (the one we are presenting is called “arc standard”)

- They use the stack and buffer in slightly different ways and may make predicting certain tree structures more or less difficult

- When designing your transition sets for your problem, keep in mind that there may be many possibilities
<table>
<thead>
<tr>
<th>Stack</th>
<th>Buffer</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I saw</td>
<td>her</td>
</tr>
<tr>
<td></td>
<td>duck</td>
<td>ROOT</td>
</tr>
</tbody>
</table>

I saw her duck ROOT
<table>
<thead>
<tr>
<th>Stack</th>
<th>Buffer</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I saw  her duck  ROOT</td>
<td>SHIFT</td>
</tr>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Action</td>
</tr>
<tr>
<td>-------</td>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>1</td>
<td>I saw her duck</td>
<td><strong>ROOT</strong></td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td><strong>ROOT</strong></td>
</tr>
<tr>
<td></td>
<td><strong>SHIFT</strong></td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Action</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>ROOT</td>
</tr>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Action</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>I</td>
<td>I saw her duck [ROOT]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>I saw</td>
<td>saw her duck [ROOT]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>I saw</td>
<td>her duck [ROOT]</td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Action</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>l saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Action</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>SHIFT</td>
</tr>
<tr>
<td>I saw</td>
<td>saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td>I saw</td>
<td>her duck</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>ROOT</td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td>ROOT</td>
</tr>
<tr>
<td></td>
<td>I saw</td>
<td>ROOT</td>
</tr>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Action</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td>![Stack Diagram]</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>![Buffer Diagram]</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>![Buffer Diagram]</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Action</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td>I</td>
<td>saw</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td>I saw</td>
<td>her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Action</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td>SHIFT</td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>SHIFT</td>
</tr>
<tr>
<td>I saw</td>
<td></td>
<td>REDUCE-L</td>
</tr>
<tr>
<td>l saw</td>
<td></td>
<td>SHIFT</td>
</tr>
<tr>
<td>I saw her</td>
<td></td>
<td>SHIFT</td>
</tr>
</tbody>
</table>
I saw her duck ROOT

Stack | Buffer | Action
--- | --- | ---
| I | saw her duck ROOT | SHIFT
| I saw | her duck ROOT | SHIFT
| I saw her | duck ROOT | REDUCE-L
| I saw her duck | | SHIFT

Action:
- SHIFT
- SHIFT
- REDUCE-L
- SHIFT
- SHIFT
<table>
<thead>
<tr>
<th>Stack</th>
<th>Buffer</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>l</td>
<td>SHIFT</td>
</tr>
<tr>
<td>I saw</td>
<td>l</td>
<td>SHIFT</td>
</tr>
<tr>
<td>I saw</td>
<td>I saw l</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td>I saw</td>
<td>her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td>I saw</td>
<td>her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td>l</td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td>l</td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td>l</td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td>I</td>
<td>I saw her duck</td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Action</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td><strong>SHIFT</strong></td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td><strong>SHIFT</strong></td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td><strong>REDUCE-L</strong></td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td><strong>SHIFT</strong></td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td><strong>SHIFT</strong></td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td><strong>REDUCE-L</strong></td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td><strong>SHIFT</strong></td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td><strong>SHIFT</strong></td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td><strong>REDUCE-L</strong></td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td><strong>SHIFT</strong></td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td><strong>SHIFT</strong></td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td><strong>REDUCE-L</strong></td>
</tr>
</tbody>
</table>
I saw her duck
<table>
<thead>
<tr>
<th>Stack</th>
<th>Buffer</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>saw</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>saw</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td></td>
<td>i saw</td>
<td>REDUCE-R</td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td></td>
<td>i saw</td>
<td>REDUCE-R</td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td>REDUCE-R</td>
</tr>
<tr>
<td></td>
<td>i saw</td>
<td>REDUCE-R</td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td>REDUCE-R</td>
</tr>
</tbody>
</table>

The diagram shows a top-down parsing process with the sentence "I saw her duck." The actions performed are `SHIFT` and `REDUCE-L` and `REDUCE-R`.
<table>
<thead>
<tr>
<th>Stack</th>
<th>Buffer</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>ROOT</strong></td>
</tr>
<tr>
<td>I</td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>her duck</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td></td>
<td>duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>ROOT</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>REDUCE-R</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td><strong>ROOT</strong></td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td><strong>ROOT</strong></td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td><strong>ROOT</strong></td>
</tr>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Action</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>REDUCE-L</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>REDUCE-R</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>SHIFT</td>
</tr>
<tr>
<td></td>
<td>I saw her duck</td>
<td>REDUCE-L</td>
</tr>
</tbody>
</table>
Making Predictions

• In transition based models, you need to look at the current "state" of the algorithm and make a decision about what to do next

• The current state in sequence models is pretty simple
  • The things you’ve labeled
  • The labels you’ve produced
  • The unlabeled part of the string

• What about in trees?
Transition-based parsing

Challenges
Transition-based parsing

Challenges

I saw her duck ROOT unbounded length
Transition-based parsing

Challenges

unbounded depth

unbounded length

I saw
her duck ROOT
Transition-based parsing

Challenges

arbitrarily complex trees \rightleftharpoons \text{unbounded depth} \rightleftharpoons \text{unbounded length} \rightleftharpoons \text{ROOT}

I saw her duck
Transition-based parsing

Challenges

unbounded depth

unbounded length

arbitrarily complex trees

I saw

her duck

ROOT

I saw her duck

I saw her

duck

ROOT

ROOT

unbounded history
Transition-based parsing

Challenges

unbounded depth

unbounded length

arbitrarily complex trees

reading and forgetting

I saw her duck

her duck ROOT

duck ROOT

ROOT

I saw her duck

I saw her
Transition-based parsing

Solutions

• Use a new variant of LSTMs—stack LSTMs—to embed buffer, stack, and history of actions

• Embeddings are sensitive to full lookahead, full stack contents, and full history of actions

• Incremental construction of parser state embedding means runtime remains linear
Augment LSTM with a **stack pointer**

Two constant-time operations

- **Push** - read input, add to top of stack
- **Pop** - move stack pointer back

A **summary** of stack contents is obtained by accessing the output of the LSTM at location of the stack pointer
Transition-based parsing
Stack LSTMs

$y_0$

$\emptyset$

PUSH
Transition-based parsing
Stack LSTMs

\[
\begin{array}{c}
\emptyset \\
\end{array}
\quad \begin{array}{c}
\emptyset \\
\end{array}
\]

\[
\begin{array}{c}
y_0 \\
\end{array}
\quad \begin{array}{c}
y_1 \\
\end{array}
\]

\[
\begin{array}{c}
x_1 \\
\end{array}
\quad \begin{array}{c}
\end{array}
\]

POP
Transition-based parsing

Stack LSTMs
Transition-based parsing

Stack LSTMs

\[ y_0 \]
\[ \emptyset \]

\[ y_1 \]
\[ x_1 \]

PUSH
Transition-based parsing

Stack LSTMs

\[ y_0 \]
\[ \emptyset \]
\[ y_1 \]
\[ x_1 \]
\[ y_2 \]
\[ x_2 \]

POP
Transition-based parsing

Stack LSTMs

\[ y_0 \]
\[ \emptyset \]
\[ y_1 \]
\[ x_1 \]
\[ y_2 \]
\[ x_2 \]
Transition-based parsing
Stack LSTMs

\[ y_0 \]
\[ \emptyset \]
\[ y_1 \]
\[ x_1 \]
\[ y_2 \]
\[ x_2 \]

PUSH
Transition-based parsing
Stack LSTMs
an overhasty decision
overhasty
overhasty

an decision

overhasty

pt

SHIFT RED-L(amod)

...
Representing Tree(let)s

\[ \text{an overhasty decision} \]
Representing Tree(let)s

\[
\text{an} \quad \text{overhasty} \quad \text{decision}
\]

\[
\text{an} \quad \text{det} \quad \text{c}_2 \quad \text{rel} \quad \text{overhasty} \quad \text{amod} \quad \text{decision}
\]

\[
\text{c}_1 \quad \text{mod} \quad \text{head} \quad \text{rel} \quad \text{overhasty} \quad \text{amod} \quad \text{decision}
\]
Inference

\[ y^* = \arg \max_y \ p(y \mid x) \]

\[ = \arg \max_y \ \prod_i p(y_i \mid y_{<i}, x) \]

RNNs never forget anything! Decoding is difficult.

- Greedy left-to-right decoding
- Beam search
- Particle filtering
<table>
<thead>
<tr>
<th>Method</th>
<th>Development</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UAS</td>
<td>LAS</td>
</tr>
<tr>
<td>S-LSTM</td>
<td>93.2</td>
<td>90.9</td>
</tr>
<tr>
<td>—POS</td>
<td>93.1</td>
<td>90.4</td>
</tr>
<tr>
<td>—pretraining</td>
<td>92.7</td>
<td>90.4</td>
</tr>
<tr>
<td>—composition</td>
<td>92.7</td>
<td>89.9</td>
</tr>
<tr>
<td>S-RNN</td>
<td>92.8</td>
<td>90.4</td>
</tr>
<tr>
<td>C&amp;M (2014)</td>
<td>92.2</td>
<td>89.7</td>
</tr>
</tbody>
</table>
Other examples

• Constituency parsing
  • both top-down and bottom-up “unrollings” exist

• bottom-up
  • \texttt{shift} behaves as it did before
  • \texttt{reduce} builds a unary or binary constituent, also takes a label type (VP, NP, …)

• top-down
  • addition of a new operation: \texttt{NT}
\[ p(S) \]
\[ p(\text{NP VP} \mid S) \]
\[ p(S) \]
\[ p(\text{NP } \text{VP} \mid S) \]
\[ p(\text{DT } \text{NN} \mid \text{S}, \text{NP}) \]
p(S)
p(NP VP | S)
p(DT NN | S, NP)
p(the | S, NP, DT)
The diagram represents a parse tree for the sentence "the cats". The root node is the sentence (S), which is divided into a noun phrase (NP) on the left and a verb phrase (VP) on the right. The noun phrase contains a determiner (DT) "the" and a noun (NNS) "cats". The verb phrase contains a verb phrase (VP) that is a function of the sentence (S). The probabilities associated with the parse tree are:

- \( p(S) \)
- \( p(NP \ VP | S) \)
- \( p(DT \ NN | S, NP) \)
- \( p(the | S, NP, DT) \)
- \( p(cats | S, NP, NN) \)
the cats
The diagram illustrates a parse tree for the sentence "the cats". The tree is labeled with categories like NP, VP, DT, NNS, VB, and RB, and the probabilities for each category given the sentence structure are also shown.

- $p(S)$
- $p(NP \ VP | S)$
- $p(DT \ NN | S, NP)$
- $p(\text{the} | S, NP, DT)$
- $p(\text{cats} | S, NP, NN)$
- $p(VP \ RB | S, VP)$
- $p(VB | S, VP, VP)$
the cats meow

$p(S)$
$p(NP \ VP | S)$
$p(DT \ NN | S, NP)$
$p(\text{the} | S, NP, DT)$
$p(\text{cats} | S, NP, NN)$
$p(VP \ RB | S, VP)$
$p(VB | S, VP, VP)$
$p(meow | S, VP, VP, VB)$
The cats meow loudly.
Top-Down Tree RNN

\[
\pi(y) = \text{parent of node } y \\
\mathbf{h}_y = \tanh (\mathbf{W} \mathbf{h}_{\pi(y)} + \mathbf{b})
\]
Top-Down Tree RNN

\[ \pi(y) = \text{parent of node } y \]

\[ h_y = \tanh (W h_{\pi(y)} + b) \]
Top-Down Tree RNN

\[ \pi(y) = \text{parent of node } y \]

\[ h_y = \tanh \left( W h_{\pi(y)} + b \right) \]
$\pi(y) = \text{parent of node } y$

$h_y = \tanh (W h_{\pi(y)} + b)$
Top-Down Tree RNN

\[ \pi(y) = \text{parent of node } y \]

\[ h_y = \tanh (W h_{\pi(y)} + b) \]
Top-Down Tree RNN

• By changing the initial state, we can build an encoder-decoder architecture on trees

• Intuitively, the initial vector “encodes” everything you want to generate.

• But- is this enough??
The cats meow loudly.

- p(S)
- p(NP VP | S)
- p(DT NN | S, NP)
- p(the | S, NP, DT)
- p(cats | S, NP, NN)
- p(VP RB | S, VP)
- p(VB | S, VP, VP)
- p(meow | S, VP, VP, VB)
- p(loudly | S, VP, RB)
the cats meow loudly
Problem: model doesn’t condition on the noun decision! Agreement??
Problem: model doesn’t condition on the noun decision! Agreement??
the
the
the cats
the cats
the cats
the cats
the cats
the cats meow
the cats meow
the cats meow
the cats meow loudly
the cats meow loudly
the cats meow loudly
Composition Functions
Top-down transition-based parsing

• Can be used for both generation and parsing
Other Neural Architectures

- Hidden RNNs?
Hidden RNNs

Replace the Markov model in an HMM with an RNN.

\[
y_0 = \text{START} \\
y_i \mid y_{<i} \sim \text{RNNLM}(y_{<i}) \\
x_i \mid y_i \sim \text{Categorical}(\theta_{y_i})
\]

Is this a valid model? Yes!

Can you perform supervised training? Yes, easily!

Can you perform posterior inference on \( y \mid x \)?

Well … the naive algorithm works. What about Viterbi?
Summary

- Neural Networks are expressive
  - …but structured prediction is too!
- Hybrid architectures give us the best of both worlds.