Recurrent neural network grammars

Widespread phenomenon: Polarity items can only appear in certain contexts

Example: “anybody” is a polarity item that tends to appear only in specific contexts:

* The lecture that I gave did not appeal to anybody

but not:

* The lecture that I gave appealed to anybody

We might infer that the licensing context is the word “not” appearing among the preceding words, and you could use an RNN to model this. However:

* The lecture that I did not give appealed to anybody

Language is hierarchical

The licensing context depends on recursive structure (syntax)

One theory of hierarchy

- Generate symbols sequentially using an RNN
- Add some “control symbols” to rewrite the history periodically
  - Periodically “compress” a sequence into a single “constituent”
  - Augment RNN with an operation to compress recent history into a single vector (i.e., “reduce”)
- RNN predicts next symbol based on the history of compressed elements and non-compressed terminals (“shift” or “generate”)
- RNN must also predict “control symbols” that decide how big constituents are
- We call such models recurrent neural network grammars.
(Ordered) tree traversals are sequences

```
The hungry cat meows .
```

<table>
<thead>
<tr>
<th>Terminals</th>
<th>Stack</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT(S)</td>
<td>(S)</td>
<td></td>
</tr>
<tr>
<td>NT(NP)</td>
<td>(S)</td>
<td></td>
</tr>
<tr>
<td>GEN(The)</td>
<td>(S)</td>
<td></td>
</tr>
<tr>
<td>GEN(hungry)</td>
<td>(S)</td>
<td></td>
</tr>
<tr>
<td>GEN(cat)</td>
<td>(S)</td>
<td>REDUCE</td>
</tr>
</tbody>
</table>

Compress “The hungry cat” into a single composite symbol
Q: What information can we use to predict the next action, and how can we encode it with an RNN?

A: We can use an RNN for each of:
1. Previous terminal symbols
2. Previous actions
3. Current stack contents
**Syntactic Composition**

Need representation for: 

(NP The hungry cat)

---

**Recursion**

Need representation for: 

1. (NP The hungry cat) 
2. (NP The (ADJP very hungry) cat)

---

**Effect**

Stack encodes top-down syntactic recency, rather than left-to-right string recency.
Implementing RNNGs

Stack RNNs

- Augment a sequential RNN with a stack pointer
- Two constant-time operations
  - push - read input, add to top of stack, connect to current location of the stack pointer
  - pop - move stack pointer to its parent
- A summary of stack contents is obtained by accessing the output of the RNN at location of the stack pointer

Note: push and pop are discrete actions here (cf. Grefenstette et al., 2015)

Implementing RNNGs

Stack RNNs
Implementing RNNGs
Stack RNNs

PUSH

Implementing RNNGs
Stack RNNs

POP

Implementing RNNGs
Stack RNNs

PUSH

Implementing RNNGs
Stack RNNs

PUSH
Implementing RNNGs

Stack RNNs

The hungry cat meows.

NP  VP
S

S( NP( The hungry cat ) VP( meows ) . )

The evolution of the stack LSTM over time mirrors tree structure

stack  top

The evolution of the stack LSTM over time mirrors tree structure

stack  top
The evolution of the stack LSTM over time mirrors tree structure.

The hungry cat meows.

The evolution of the stack LSTM over time mirrors tree structure.

The hungry cat meows.
The evolution of the stack LSTM over time mirrors tree structure.
The evolution of the stack LSTM over time mirrors tree structure

Each word is conditioned on history represented by a trio of RNNs

Train with backpropagation through structure

Complete model

sentence

Sequence of actions (completely defines $x$ and $y$)

Model is dynamic: variable number of context-dependent actions at each step allowable actions at this step

And recursively through this structure.

This network is dynamic. Don’t derive gradients by hand—that’s error prone. Use automatic differentiation instead.

In training, backpropagate through these three RNNs

$S(\text{NP( The hungry cat ) VP( meows ) })$
Complete model

\[
p(x, y) = \prod_{t=1}^{a(x, y)} p(a_t | a_{<t}) \prod_{t=1}^{a(x, y)} \frac{\exp \sum_{x' \in A_G(T_t, s_t, h_t)} r(a_t, x_t) + b_{a_t}}{\sum_{y' \in A_G(T_t, s_t, h_t)} \exp \sum_{x' \in A_G(T_t, s_t, h_t)} r(a_t, x_t) + b_{a_t}}
\]

\[
u_t = \tanh(W[a_t; s_t; h_t] + c)
\]

Implementing RNNGs

Parameter Estimation

- RNNGs jointly model sequences of words together with a “tree structure”, \( p_\theta(x, y) \)
- Any parse tree can be converted to a sequence of actions (depth first traversal) and vice versa (subject to wellformedness constraints)
- We use trees from the Penn Treebank
- We could treat the non-generation actions as latent variables or learn them with RL, effectively making this a problem of grammar induction. Future work…

Implementing RNNGs

Inference

- An RNNG is a joint distribution \( p(x, y) \) over strings (\( x \)) and parse trees (\( y \))
- We are interested in two inference questions:
  - What is \( p(x) \) for a given \( x \)? [language modeling]
  - What is max \( p(y | x) \) for a given \( x \)? [parsing]
- Unfortunately, the dynamic programming algorithms we often rely on are of no help here
- We can use importance sampling to do both by sampling from a discriminatively trained model

English PTB (Parsing)

<table>
<thead>
<tr>
<th>Type</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Petrov and Klein (2007)</td>
<td>G</td>
</tr>
<tr>
<td>Shindo et al (2012) Ensemble</td>
<td>~G</td>
</tr>
<tr>
<td>Vinyals et al (2015) PTB only</td>
<td>D</td>
</tr>
<tr>
<td>Discriminative</td>
<td>D</td>
</tr>
<tr>
<td>Generative (IS)</td>
<td>G</td>
</tr>
</tbody>
</table>
Importance Sampling

Assume we've got a conditional distribution \( q(y \mid x) \)

\[ p(x, y) > 0 \implies q(y \mid x) > 0 \]

\[ y \sim q(y \mid x) \text{ is tractable and} \]

\[ q(y \mid x) \text{ is tractable} \]

Let the importance weights \( w(x, y) = \frac{p(x, y)}{q(y \mid x)} \)

\[ p(x) = \sum_{y \in \mathcal{Y}(x)} p(x, y) = \sum_{y \in \mathcal{Y}(x)} w(x, y)q(y \mid x) \]

\[ = \mathbb{E}_{y \sim q(y \mid x)}w(x, y) \]

English PTB (LM)

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-gram IKN</td>
<td>169.3</td>
</tr>
<tr>
<td>LSTM + Dropout</td>
<td>113.4</td>
</tr>
<tr>
<td>Generative (IS)</td>
<td>102.4</td>
</tr>
</tbody>
</table>

Chinese CTB (LM)

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-gram IKN</td>
<td>255.2</td>
</tr>
<tr>
<td>LSTM + Dropout</td>
<td>207.3</td>
</tr>
<tr>
<td>Generative (IS)</td>
<td>171.9</td>
</tr>
</tbody>
</table>

Do we need a stack?

Kuncoro et al., Oct 2017

- Both stack and action history encode the same information, but expose it to the classifier in different ways.

<table>
<thead>
<tr>
<th>Model</th>
<th>( F_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vinayals et al. (2015)†</td>
<td>92.1</td>
</tr>
<tr>
<td>Choe and Charniak (2016)</td>
<td>92.6</td>
</tr>
<tr>
<td>Choe and Charniak (2016)†</td>
<td><strong>93.8</strong></td>
</tr>
<tr>
<td>Baseline RNNG</td>
<td>93.3</td>
</tr>
<tr>
<td>Ablated RNNG (no history)</td>
<td>93.2</td>
</tr>
<tr>
<td>Ablated RNNG (no buffer)</td>
<td>93.3</td>
</tr>
<tr>
<td>Ablated RNNG (no stack)</td>
<td>92.5</td>
</tr>
<tr>
<td>Stack-only RNNG</td>
<td><strong>93.6</strong></td>
</tr>
<tr>
<td>GA-RNNG</td>
<td>93.5</td>
</tr>
</tbody>
</table>

Leaving out stack is harmful; using it on its own works slightly better than complete model!
RNNG as a mini-linguist

• Replace composition with one that computes attention over objects in the composed sequence, using embedding of NT for similarity.

• What does this learn?

<table>
<thead>
<tr>
<th>Noun phrases</th>
<th>Verb phrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canadian (0.09) Auto (0.31) Workers (0.2) union (0.22) president (0.18) no (0.29) major (0.05) Eurobond (0.32) or (0.01) foreign (0.01) bond (0.1) offerings (0.22) Saatchi (0.12) client (0.14) Philips (0.21) Lighting (0.24) Co. (0.29) nonperforming (0.18) commercial (0.23) real (0.25) estate (0.1) assets (0.25) the (0.1) Jamaica (0.1) Tourist (0.03) Board (0.17) ad (0.20) account (0.40) their (0.0) first (0.23) test (0.77) Apple (0.62) , (0.02) Compaq (0.1) and (0.01) IBM (0.25) both (0.02) stocks (0.03) and (0.06) futures (0.88) NP (0.01) , (0.0) and (0.98) NP (0.01)</td>
<td>buying (0.31) and (0.25) selling (0.21) NP (0.23) ADVP (0.27) show (0.29) PRT (0.23) PP (0.21) pleaded (0.48) ADJP (0.23) PP (0.15) PP (0.08) PP (0.06) received (0.33) PP (0.18) NP (0.32) PP (0.17) cut (0.27) NP (0.37) PP (0.22) PP (0.14) to (0.99) VP (0.01) were (0.77) n’t (0.22) VP (0.01) did (0.39) n’t (0.60) VP (0.01) handle (0.09) NP (0.91) VP (0.15) and (0.83) VP (0.02)</td>
</tr>
</tbody>
</table>

Figure 3: Average perplexity of the learned attention vectors on the test set (blue), as opposed to the average perplexity of the uniform distribution (red), computed for each major phrase type.
RNNG as a mini-linguist

- Replace composition with one that computes attention over objects in the composed sequence, using embedding of NT for similarity.

- What does this learn?

<table>
<thead>
<tr>
<th>Prepositional phrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVP (0.14) on (0.72) NP (0.14)</td>
</tr>
<tr>
<td>ADVP (0.05) for (0.54) NP (0.40)</td>
</tr>
<tr>
<td>ADVP (0.02) because (0.73) of (0.18) NP (0.07)</td>
</tr>
<tr>
<td>such (0.31) as (0.65) NP (0.04)</td>
</tr>
<tr>
<td>from (0.39) NP (0.49) PP (0.12)</td>
</tr>
<tr>
<td>(0.97) NP (0.03)</td>
</tr>
<tr>
<td>in (0.93) NP (0.07)</td>
</tr>
<tr>
<td>by (0.96) S (0.04)</td>
</tr>
<tr>
<td>at (0.99) NP (0.01)</td>
</tr>
<tr>
<td>NP (0.1) after (0.83) NP (0.06)</td>
</tr>
</tbody>
</table>

Summary

- Language is hierarchical, and this inductive bias can be encoded into an RNN-style model.

- RNNGs work by simulating a tree traversal—like a pushdown automaton, but with continuous rather than finite history.

- Modeled by RNNs encoding (1) previous tokens, (2) previous actions, and (3) stack contents.

- A stack LSTM evolves with stack contents.

- The final representation computed by a stack LSTM has a top-down recency bias, rather than left-to-right bias, which might be useful in modeling sentences.

- Effective for parsing and language modeling, and seems to capture linguistic intuitions about headedness.