Recurrent Neural Network Grammars

NAACL-HLT 2016

Authors: Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, Noah A. Smith

Presenter: Che-Lin Huang

Motivation

- Sequential recurrent neural networks (RNNs) are remarkably effective models of natural language
- Despite these impressive results, sequential models are not appropriate models of natural language
- Relationships among words are largely organized in terms of latent nested structures rather than sequential order

Overview of RNNG

- A new generative probabilistic model of sentences that explicitly models nested, hierarchical relationships among words and phrases
- RNNGs maintain the algorithmic convenience of transition based parsing but incorporate top-down syntactic information
- They give two variants of the algorithm, one for parsing, and one for generation:
  - The parsing algorithm transforms a sequence of words x into a parse tree y
  - The generation algorithm stochastically generates terminal symbols and trees with arbitrary structures

Top-down variant of transition-based parsing algorithm

- Begin with the stack (S) empty, the complete sequence of words in the input buffer (B), and zero number of open nonterminals on the stack (n)
- Stack: terminal symbols, open nonterminal symbols, and complete constituents
- Input buffer: unprocessed terminal symbols
- Three classes of operations: NT(X), SHIFT, and REDUCE

Stack$_t$	**Buffer**$_t$	**Open NTs**$_t$	**Action**	**Stack**$_{t+1}$	**Buffer**$_{t+1}$	**Open NTs**$_{t+1}$
S	B	n	NT(X)	$S \mid (\mathrm{X}$	B	$n+1$
S	$x \mid B$	n	SHIFT	$S \mid x$	B	n
$S \mid (\mathrm{X} \mid \tau_1 \mid \ldots \mid \tau_\ell$	B	n	REDUCE	$S \mid (\mathrm{X}\ \tau_1\ \ldots\ \tau_\ell)$	B	$n-1$

Top-down variant of transition-based parsing algorithm

- Terminate when both criterions meet:
  1. A single completed constituent on the stack
  2. The buffer is empty
- Constraints on parser transitions:
  1. NT(X) can only be applied if B is not empty and $n < 100$
  2. SHIFT can only be applied if B is not empty and $n \geq 1$
  3. REDUCE can only be applied if $n \geq 2$ or if the buffer is empty
  4. REDUCE can only be applied if the top of the stack is not an open nonterminal symbol

Parser transitions and parsing example

Stack$_t$	**Buffer**$_t$	**Open NTs**$_t$	**Action**	**Stack**$_{t+1}$	**Buffer**$_{t+1}$	**Open NTs**$_{t+1}$
S	B	n	NT(X)	$S \mid (\text{X}$	B	$n+1$
S	$x \mid B$	n	SHIFT	$S \mid x$	B	n
$S \mid (\text{X} \mid \tau_1 \mid \ldots \mid \tau_\ell$	B	n	REDUCE	$S \mid (\text{X}\ \tau_1\ \ldots\ \tau_\ell)$	B	$n-1$

Input: *The hungry cat meows .*

	Stack	**Buffer**	**Action**
0		*The* | *hungry* | *cat* | *meows* | .	NT(S)
1	(S	*The* | *hungry* | *cat* | *meows* | .	NT(NP)
2	(S | (NP	*The* | *hungry* | *cat* | *meows* | .	SHIFT
3	(S | (NP | *The*	*hungry* | *cat* | *meows* | .	SHIFT
4	(S | (NP | *The* | *hungry*	*cat* | *meows* | .	SHIFT
5	(S | (NP | *The* | *hungry* | *cat*	*meows* | .	REDUCE
6	(S | (NP *The hungry cat*)	*meows* | .	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*meows* | .	SHIFT
8	(S | (NP *The hungry cat*) | (VP *meows*	.	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	.	SHIFT
10	(S | (NP *The hungry cat*) | (VP *meows*) | .		REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)		

Generation algorithm

- Can be adapted from parsing algorithm with minor changes
- No input buffer, instead there is an output buffer (T)
- No SHIFT operation, instead there is GEN(x) operation that generate terminal symbol and add it to the top of stack and the output buffer
- Constraints on generator transitions:

  1. GEN(x) can only be applied if $n \geq 1$

  2. REDUCE can only be applied if the top of the stack is not an open nonterminal symbol and $n \geq 1$

Generator transitions and generation example

Stack$_t$	Terms$_t$	Open NTs$_t$	Action	Stack$_{t+1}$	Terms$_{t+1}$	Open NTs$_{t+1}$
S	T	n	NT(X)	$S \mid$ (X	T	$n+1$
S	T	n	GEN(x)	$S \mid x$	$T \mid x$	n
$S \mid$ (X $\mid \tau_1 \mid \ldots \mid \tau_\ell$	T	n	REDUCE	$S \mid$ (X $\tau_1 \ldots \tau_\ell$)	T	$n-1$

	Stack	Terminals	Action
0			NT(S)
1	(S		NT(NP)
2	(S | (NP		GEN(*The*)
3	(S | (NP | *The*	*The*	GEN(*hungry*)
4	(S | (NP | *The* | *hungry*	*The* | *hungry*	GEN(*cat*)
5	(S | (NP | *The* | *hungry* | *cat*	*The* | *hungry* | *cat*	REDUCE
6	(S | (NP *The hungry cat*)	*The* | *hungry* | *cat*	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*The* | *hungry* | *cat*	GEN(*meows*)
8	(S | (NP *The hungry cat*) | (VP *meows*	*The* | *hungry* | *cat* | *meows*	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	*The* | *hungry* | *cat* | *meows*	GEN(.)
10	(S | (NP *The hungry cat*) | (VP *meows*) | .	*The* | *hungry* | *cat* | *meows* | .	REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)	*The* | *hungry* | *cat* | *meows* | .	

Generative model

- RNNGs use the generator transition set to define a joint distribution on syntax trees (y) and words (x), which is a sequence model over generator transitions that is parameterized using a continuous space embedding of the algorithm state at each time step (u_t):

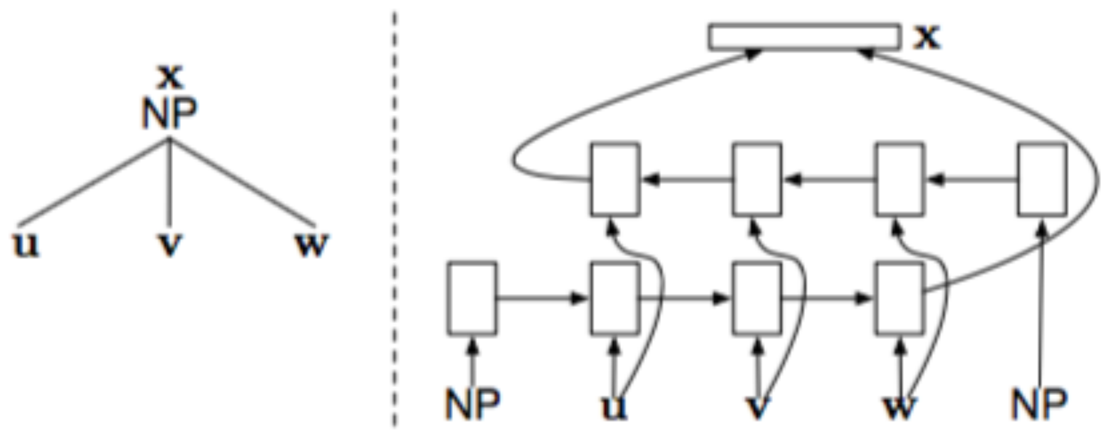
$$p(\boldsymbol{x}, \boldsymbol{y}) = \prod_{t=1}^{|\boldsymbol{a}(\boldsymbol{x},\boldsymbol{y})|} p(a_t \mid \boldsymbol{a}_{<t})$$

$$= \prod_{t=1}^{|\boldsymbol{a}(\boldsymbol{x},\boldsymbol{y})|} \frac{\exp \mathbf{r}_{a_t}^\top \mathbf{u}_t + b_{a_t}}{\sum_{a' \in \mathcal{A}_G(T_t, S_t, n_t)} \exp \mathbf{r}_{a'}^\top \mathbf{u}_t + b_{a'}}$$

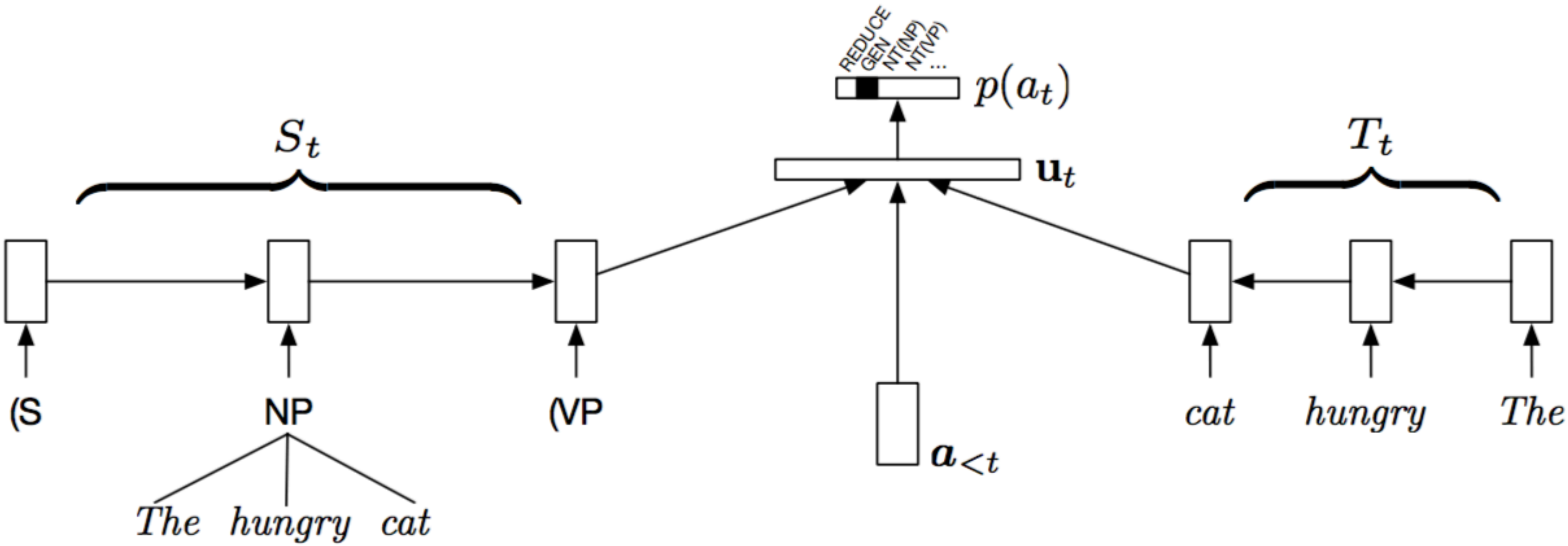
$$\mathbf{u}_t = \tanh\left(\mathbf{W}[\mathbf{o}_t; \mathbf{s}_t; \mathbf{h}_t] + \mathbf{c}\right)$$

Syntactic composition function

- The output buffer, stack, and history can grow unboundedly
- To obtain representations of them, they use RNN to encode their content
- Output buffer and history apply a standard RNN encoding
- Stack is more complicated, use stack LSTMs to encode
- To compute an embedding of this new subtree, use a composition function based on bidirectional LSTMs:



Neural architecture



- Neural architecture for defining a distribution over a_t given representations of the stack (S_t), output buffer (T_t) and history of actions ($a_{<t}$)

Inference via importance sampling

- To evaluate the generative model as a language model, we need to compute the marginal probability: $p(x) = \sum_{y' \in \mathcal{Y}} p(x, y')$
- Use a conditional proposal distribution $q(y|x)$ with properties:
  1. $p(x, y) > 0 \Longrightarrow q(y|x) > 0$
  2. Samples $y \sim q(y|x)$ can be obtained efficiently
  3. $q(y|x)$ of these samples are known
- Importance weights: $w(x, y) = p(x, y)/q(y|x)$

$$
\begin{aligned}
p(\boldsymbol{x}) &= \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} q(\boldsymbol{y} \mid \boldsymbol{x}) w(\boldsymbol{x}, \boldsymbol{y}) \\
&= \mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x})} w(\boldsymbol{x}, \boldsymbol{y}). \qquad \stackrel{\text{MC}}{\approx} \frac{1}{N} \sum_{i=1}^{N} w(\boldsymbol{x}, \boldsymbol{y}^{(i)})
\end{aligned}
$$

English parsing result

Model	**type**	$\boldsymbol{F_1}$
Vinyals et al. (2015)* – WSJ only	D	88.3
Henderson (2004)	D	89.4
Socher et al. (2013a)	D	90.4
Zhu et al. (2013)	D	90.4
Petrov and Klein (2007)	G	90.1
Bod (2003)	G	90.7
Shindo et al. (2012) – single	G	91.1
Shindo et al. (2012) – ensemble	G	92.4
Zhu et al. (2013)	S	91.3
McClosky et al. (2006)	S	92.1
Vinyals et al. (2015)	S	92.1
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})^{\dagger}$ – buggy	D	89.8
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})^{\dagger}$ – buggy	G	92.4
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	D	91.7
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	G	**93.3**

- Parsing results on Penn Treebank
- D: discriminative
- G: generative
- S: semisupervised
- F1 score:

$$F_1 = 2\frac{precision \times recall}{precision + recall} \times 100\%$$

Chinese parsing result

Model	type	F_1
Zhu et al. (2013)	D	82.6
Wang et al. (2015)	D	83.2
Huang and Harper (2009)	D	84.2
Charniak (2000)	G	80.8
Bikel (2004)	G	80.6
Petrov and Klein (2007)	G	83.3
Zhu et al. (2013)	S	85.6
Wang and Xue (2014)	S	86.3
Wang et al. (2015)	S	86.6
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})^{\dagger}$ - buggy	D	80.7
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})^{\dagger}$ - buggy	G	82.7
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	D	84.6
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	G	**86.9**

- Parsing results on Penn Chinese Treebank
- D: discriminative
- G: generative
- S: semisupervised
- F1 score:

$$F_1 = 2\frac{precision \times recall}{precision + recall} \times 100\%$$

Language model result

- Report per-word perplexities of three language models
- Cross-entropy:

$$H(p,q) = -\sum_{x} p(x)\, log_2 q(x)$$

- per-word perplexities :

$$2^{\frac{H(p,q)}{N}}$$

Model	**test ppl** (PTB)	**test ppl** (CTB)
IKN 5-gram	169.3	255.2
LSTM LM	113.4	207.3
RNNG	102.4	171.9

Conclusion

- The generative model is quite effective as a parser and a language model. This is the result of:
  - Relaxing conventional independence assumptions
  - Inferring continuous representations of symbols alongside non-linear models of their syntactic relationships

- Discriminative model performs worse than generative model:
  - Larger, unstructured conditioning contexts are harder to learn from
  - It provide opportunities to overfit

Thank you!