

Recurrent neural network grammars

Slide credits: Chris Dyer, Adhiguna Kuncoro

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:

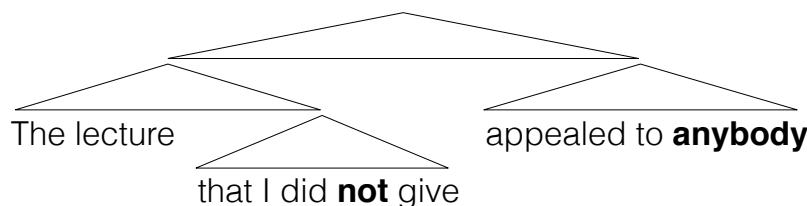
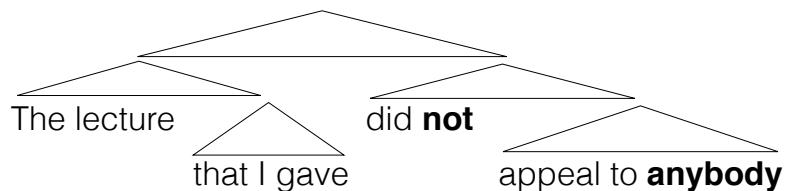
* *The lecture that I gave appealed to **anybody***

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

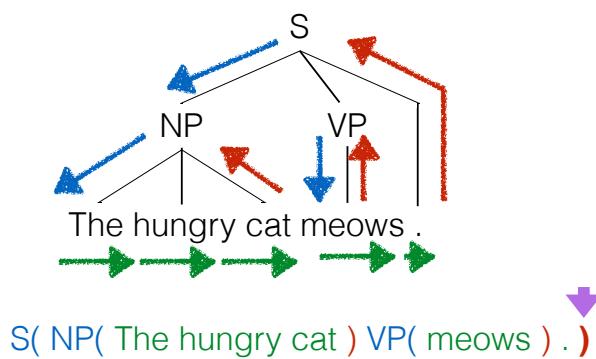
The licensing context depends on recursive structure (syntax)



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 (-> “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



Terminals	Stack	Action
	(S	NT(S)
	(S (NP	NT(NP)
The	(S (NP The	GEN(The)
	(S (NP The hungry	GEN(hungry)
	(S (NP The hungry cat	GEN(cat)
	(S (NP The hungry cat)	REDUCE
	(S (NP The hungry cat) .	

Terminals	Stack	Action
	(S	NT(S)
	(S (NP	NT(NP)
The	(S (NP The	GEN(The)
	(S (NP The hungry	GEN(hungry)
	(S (NP The hungry cat	GEN(cat)
	(S (NP The hungry cat)	REDUCE
	(S (NP The hungry cat) .	
Compress "The hungry cat" into a single composite symbol		

Terminals	Stack	Action
	(S	NT(S)
	(S (NP	NT(NP)
The	(S (NP The	GEN(The)
	(S (NP The hungry	GEN(hungry)
	(S (NP The hungry cat	GEN(cat)
	(S (NP The hungry cat)	REDUCE
	(S (NP The hungry cat) .	NT(VP)

Terminals	Stack	Action	Terminals	Stack	Action
<i>The</i>	(S)	NT(S)	<i>The</i>	(S)	NT(S)

Q: What information can we use to predict the next action, and how can we encode it with an RNN?

Terminals	Stack	Action	Terminals	Stack	Action
<i>The</i>	(S)	NT(S)	<i>The</i>	(S)	NT(S)

A: We can use an RNN for each of:

1. Previous terminal symbols
2. Previous actions
3. Current stack contents

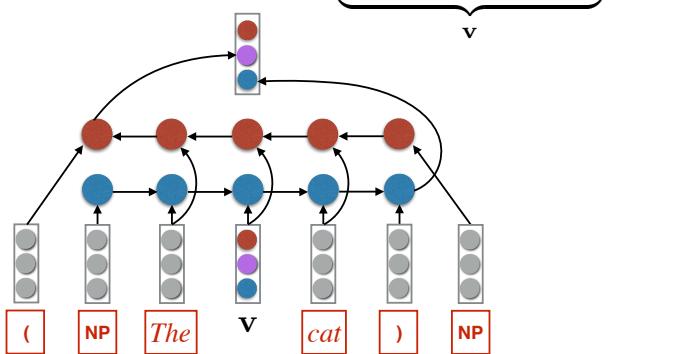
Terminals	Stack	Action
		NT(S)
The	(S	NT(NP)
The hungry	(S (NP	GEN(The)
The hungry cat	(S (NP The hungry	GEN(hungry)
The hungry cat	(S (NP The hungry cat	GEN(cat)
The hungry cat		REDUCE
The hungry cat		NT(VP)
The hungry cat		GEN(meows)
The hungry cat meows		REDUCE
The hungry cat meows	(S (NP The hungry cat) (VP meows)	GEN(.)
The hungry cat meows .	(S (NP The hungry cat) (VP meows) .	REDUCE
The hungry cat meows .	(S (NP The hungry cat) (VP meows) .)	

Recursion

Need representation for:

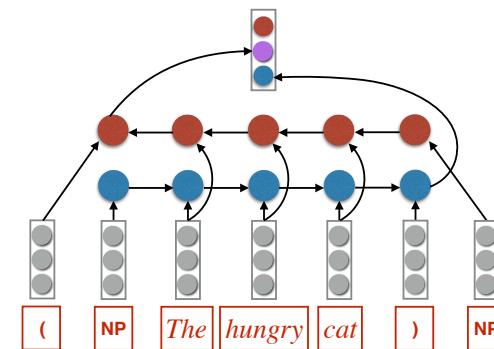
(NP *The hungry cat*)

(NP *The* (ADJP *very hungry*) *cat*)

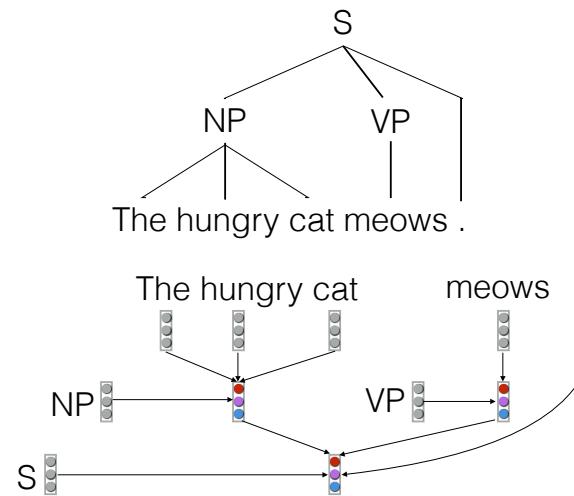


Syntactic Composition

Need representation for: (NP *The hungry cat*)



Stack symbols composed recursively
mirror corresponding tree structure



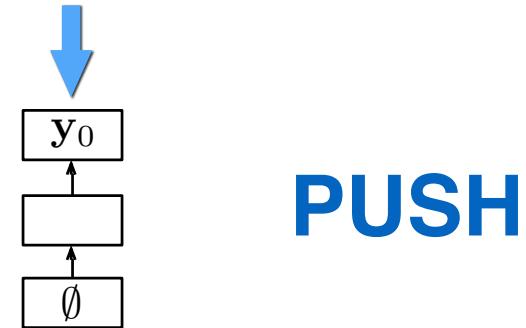
Effect

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

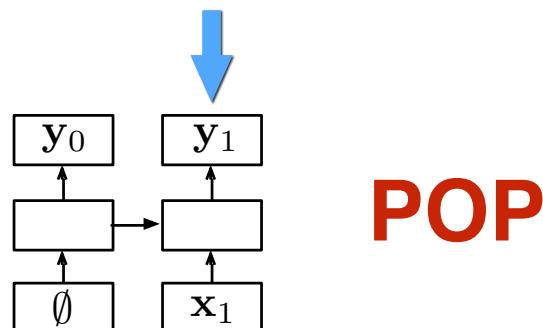
Implementing RNNGs Stack RNNs

- 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

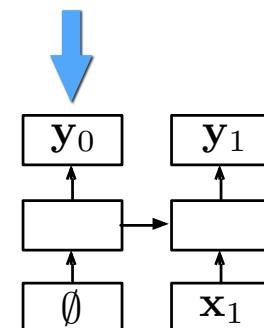
Implementing RNNGs Stack RNNs



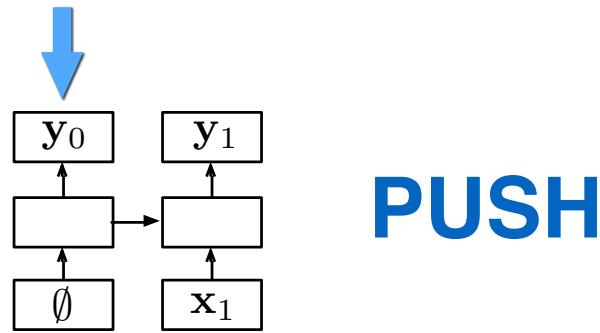
Implementing RNNGs Stack RNNs



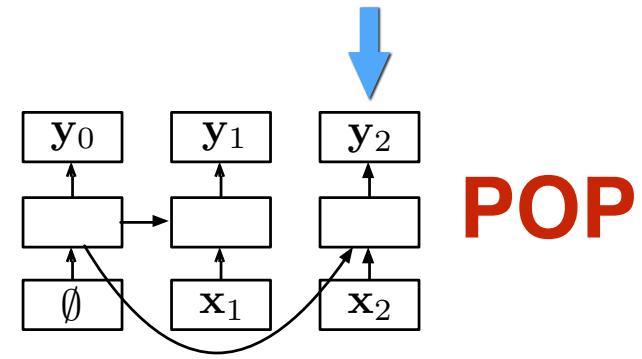
Implementing RNNGs Stack RNNs



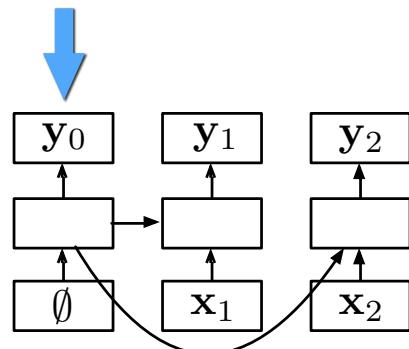
Implementing RNNs Stack RNNs



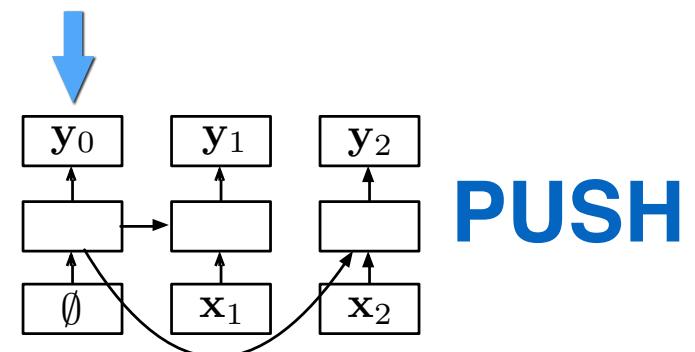
Implementing RNNs Stack RNNs



Implementing RNNs Stack RNNs

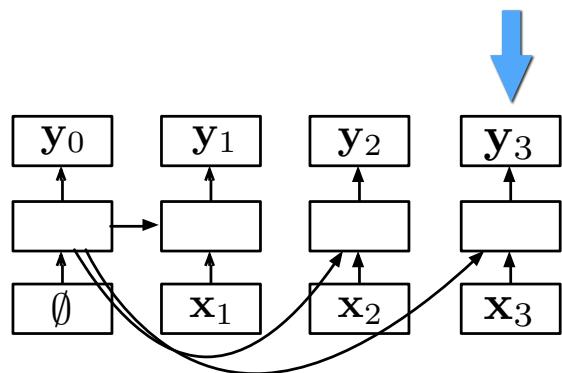


Implementing RNNs Stack RNNs

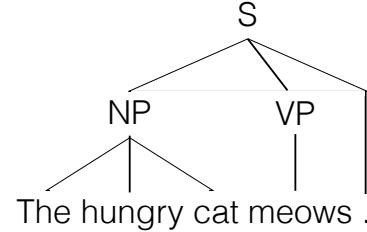


Implementing RNNs

Stack RNNs



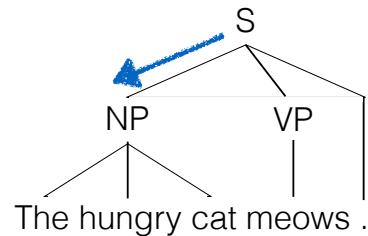
The evolution of the stack LSTM over time mirrors tree structure



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

stack ← top

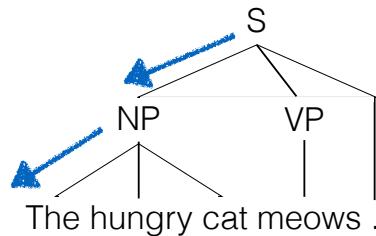
The evolution of the stack LSTM over time mirrors tree structure



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

stack → S ← top

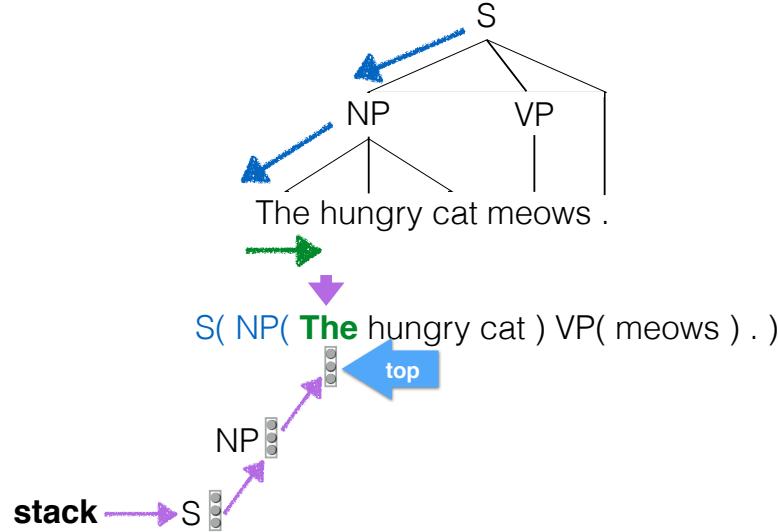
The evolution of the stack LSTM over time mirrors tree structure



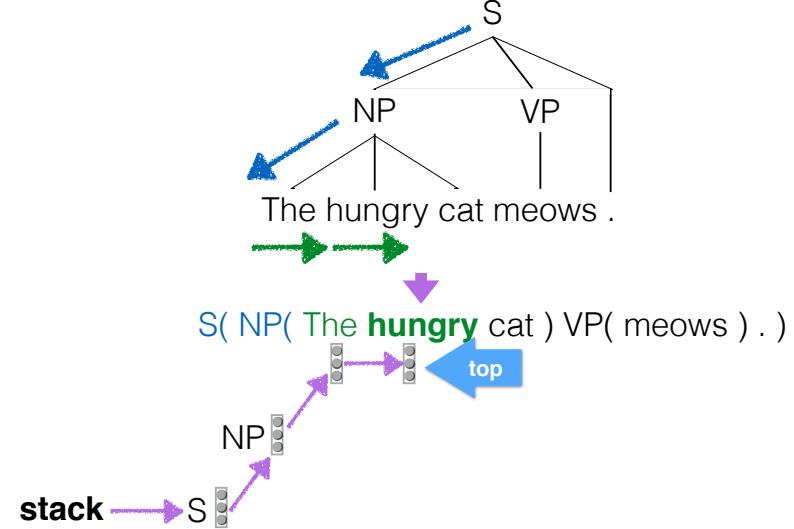
$S(NP(The hungry) VP(meows) .)$

stack → S ← top

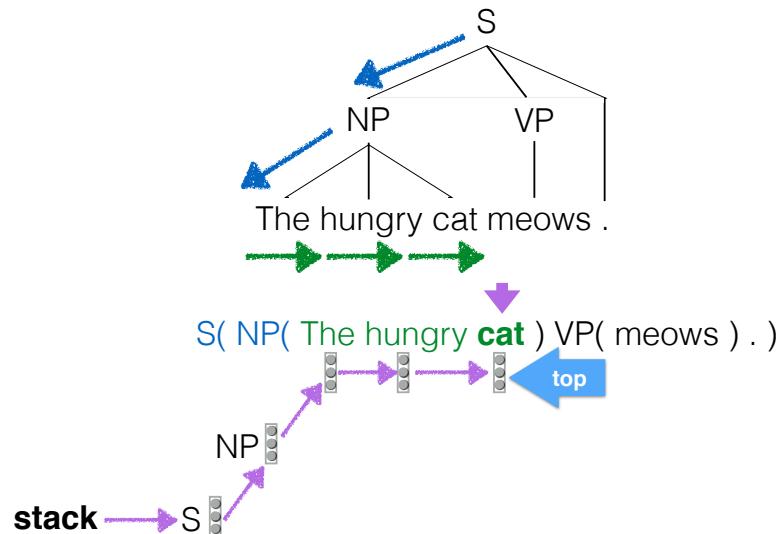
The evolution of the stack LSTM over time mirrors tree structure



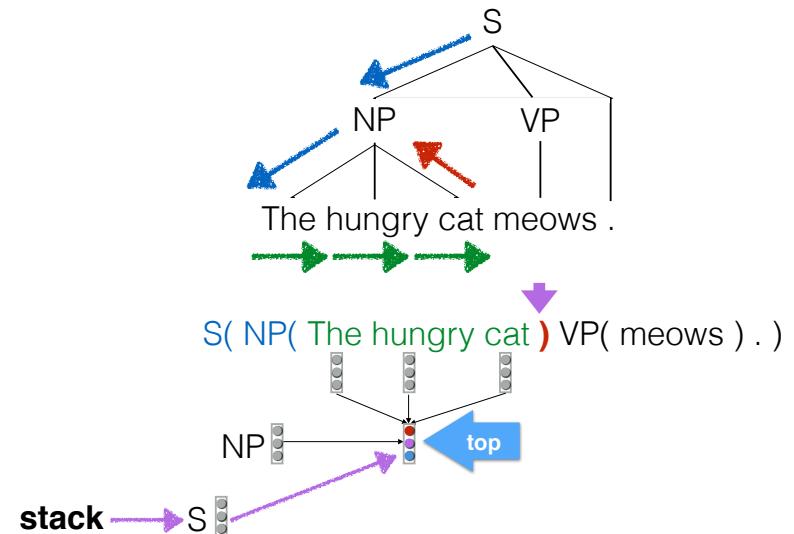
The evolution of the stack LSTM over time mirrors tree structure



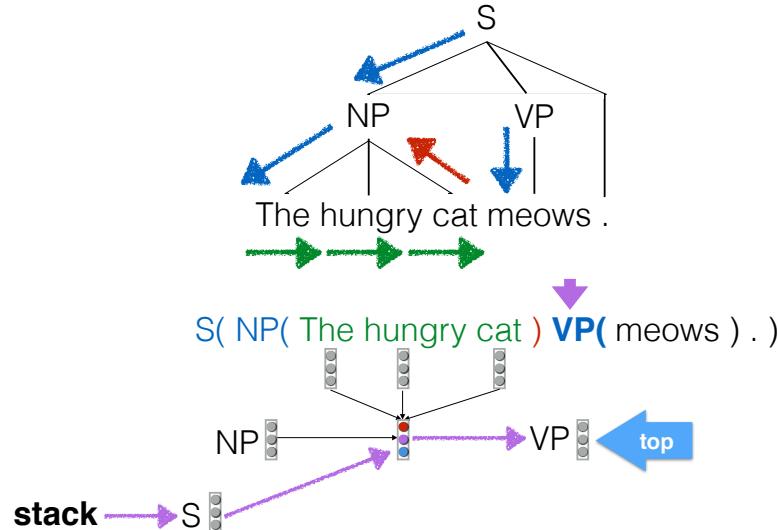
The evolution of the stack LSTM over time mirrors tree structure



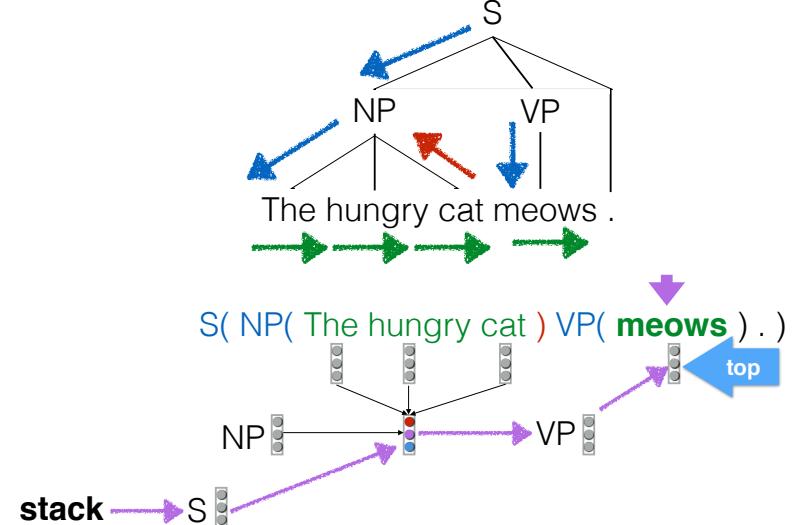
The evolution of the stack LSTM over time mirrors tree structure



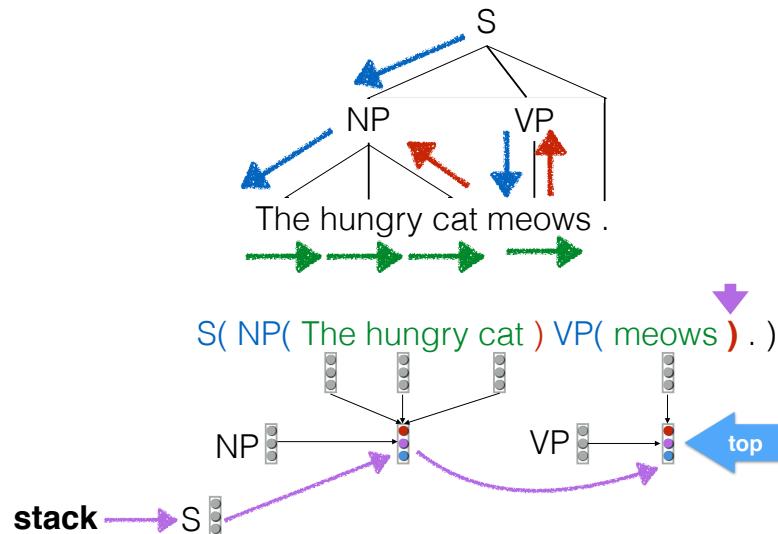
The evolution of the stack LSTM over time mirrors tree structure



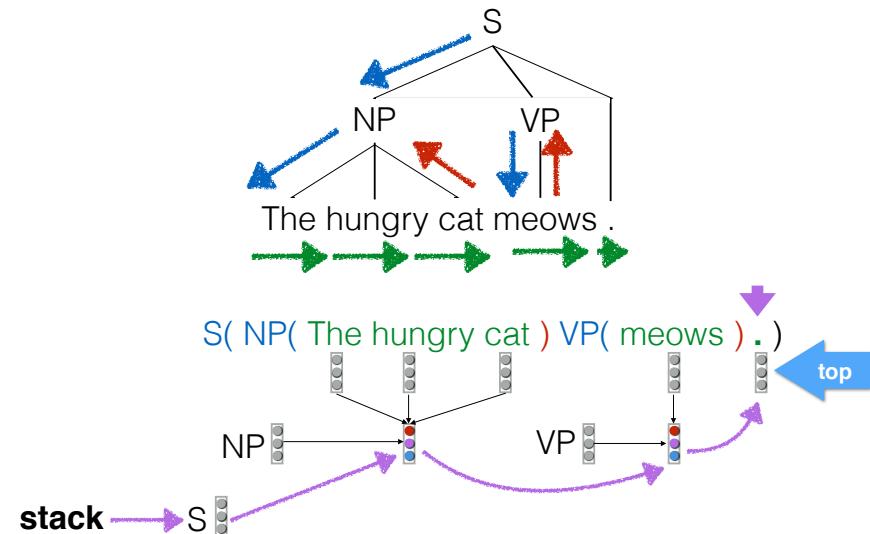
The evolution of the stack LSTM over time mirrors tree structure



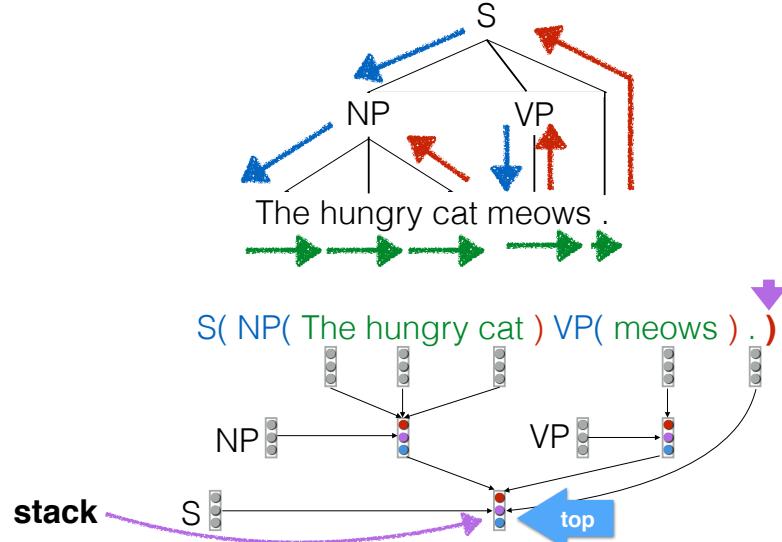
The evolution of the stack LSTM over time mirrors tree structure



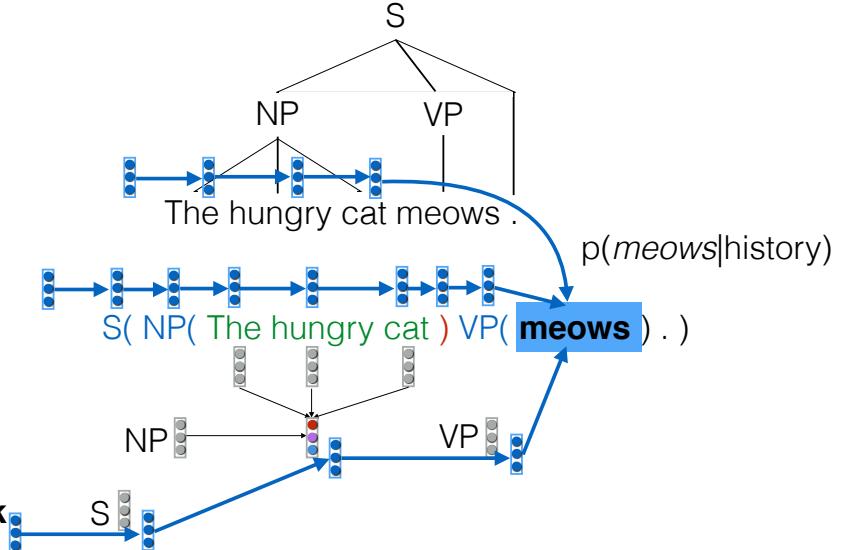
The evolution of the stack LSTM over time mirrors tree structure



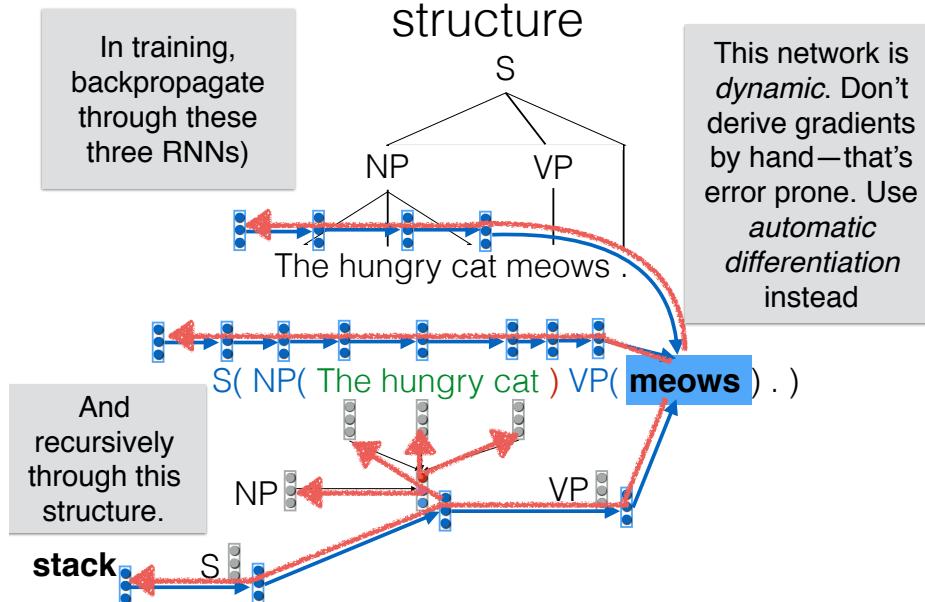
The evolution of the stack LSTM over time mirrors tree structure



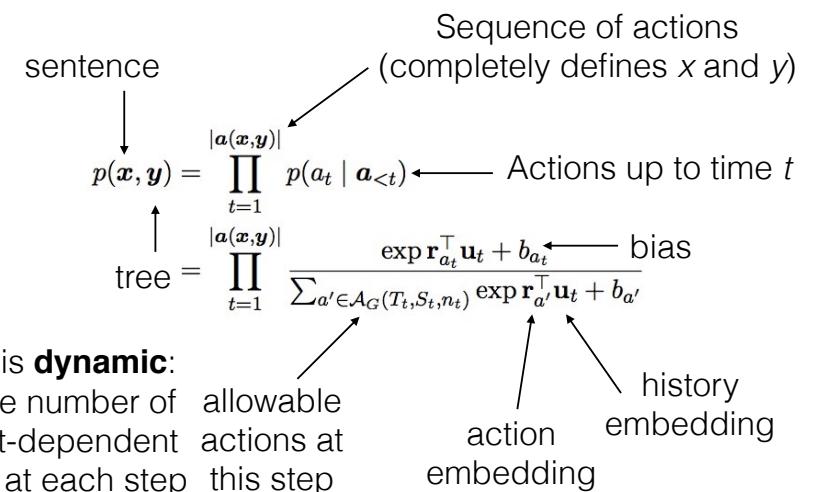
Each word is conditioned on history represented by a trio of RNNs



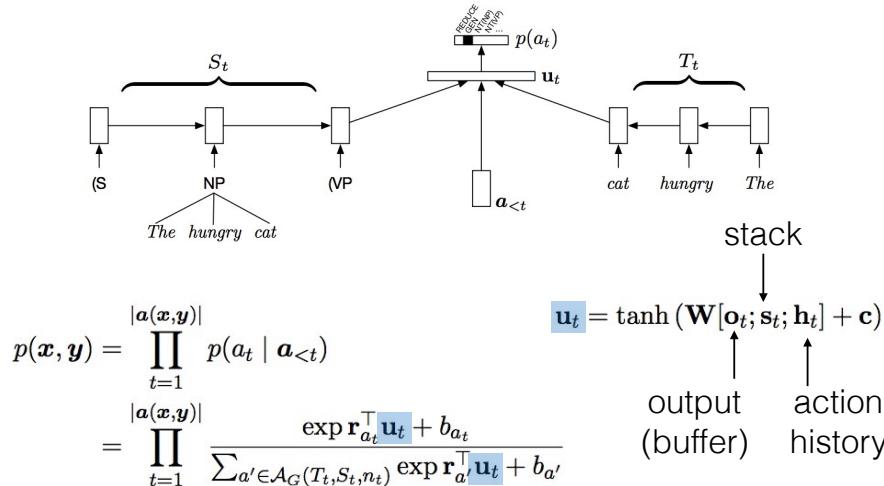
Train with backpropagation through structure



Complete model



Complete model



Implementing RNNGs Parameter Estimation

- RNNGs jointly model sequences of words together with a “tree structure”, $p_{\theta}(\mathbf{x}, \mathbf{y})$
- 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 $p(\mathbf{x}, \mathbf{y})$ over strings (\mathbf{x}) and parse trees (\mathbf{y})
- We are interested in two inference questions:
 - What is $p(\mathbf{x})$ for a given \mathbf{x} ? [\[language modeling\]](#)
 - What is $\max_{\mathbf{y}} p(\mathbf{y} | \mathbf{x})$ for a given \mathbf{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)

	Type	F1
Petrov and Klein (2007)	G	90.1
Shindo et al (2012) Single model	G	91.1
Shindo et al (2012) Ensemble	~G	92.4
Vinyals et al (2015) PTB only	D	90.5
Vinyals et al (2015) Ensemble	S	92.8
<i>Discriminative</i>	D	89.8
<i>Generative (IS)</i>	G	92.4

Importance Sampling

Assume we've got a conditional distribution $q(\mathbf{y} | \mathbf{x})$

- s.t. (i) $p(\mathbf{x}, \mathbf{y}) > 0 \implies q(\mathbf{y} | \mathbf{x}) > 0$
- (ii) $\mathbf{y} \sim q(\mathbf{y} | \mathbf{x})$ is tractable and
- (iii) $q(\mathbf{y} | \mathbf{x})$ is tractable

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

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} p(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} w(\mathbf{x}, \mathbf{y})q(\mathbf{y} | \mathbf{x}) \\ &= \mathbb{E}_{\mathbf{y} \sim q(\mathbf{y} | \mathbf{x})} w(\mathbf{x}, \mathbf{y}) \end{aligned}$$

Importance Sampling

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} p(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} w(\mathbf{x}, \mathbf{y})q(\mathbf{y} | \mathbf{x}) \\ &= \mathbb{E}_{\mathbf{y} \sim q(\mathbf{y} | \mathbf{x})} w(\mathbf{x}, \mathbf{y}) \end{aligned}$$

Replace this expectation with its Monte Carlo estimate.

$$\mathbf{y}^{(i)} \sim q(\mathbf{y} | \mathbf{x}) \quad \text{for } i \in \{1, 2, \dots, N\}$$

$$\mathbb{E}_{q(\mathbf{y} | \mathbf{x})} w(\mathbf{x}, \mathbf{y}) \stackrel{\text{MC}}{\approx} \frac{1}{N} \sum_{i=1}^N w(\mathbf{x}, \mathbf{y}^{(i)})$$

English PTB (LM)

Perplexity	
5-gram IKN	169.3
LSTM + Dropout	113.4
Generative (IS)	102.4

Chinese CTB (LM)

Perplexity	
5-gram IKN	255.2
LSTM + Dropout	207.3
Generative (IS)	171.9

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.

Model	F_1
Vinyals et al. (2015) [†]	92.1
Choe and Charniak (2016)	92.6
Choe and Charniak (2016) [†]	93.8
Baseline RNNG	93.3
Ablated RNNG (no history)	93.2
Ablated RNNG (no buffer)	93.3
Ablated RNNG (no stack)	92.5
Stack-only RNNG	93.6
GA-RNNG	93.5

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?

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?

Noun phrases

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)**
the (0.0) final (0.18) **hour (0.81)**
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)

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?

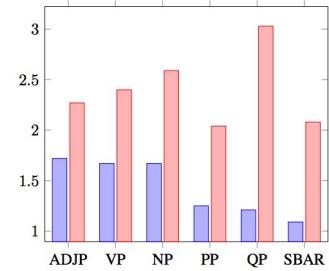


Figure 3: Average perplexity of the learned attention vectors on the test set (blue), as opposed to the average perplexity of the uniform distribution (red), computed for each major phrase type.

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?

Verb phrases

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

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?

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

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.