Recurrent Neural Network Grammars

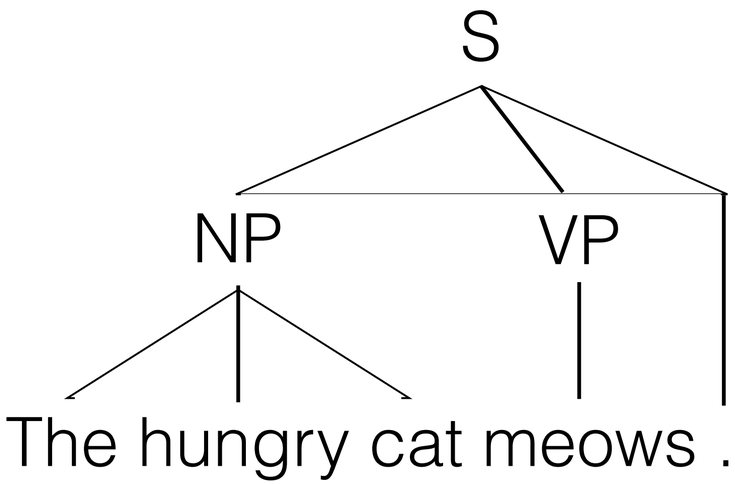
COMP3361 — Week 8

Lingpeng Kong

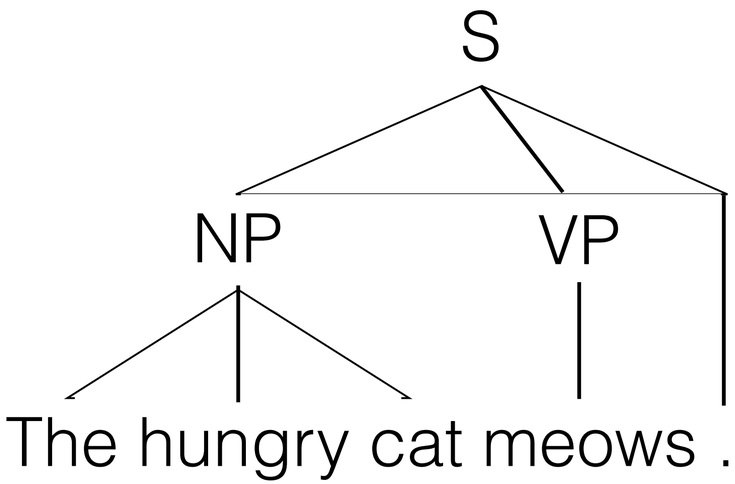
Department of Computer Science, The University of Hong Kong

Slide credits: Chris Dyer, Adhiguna Kuncoro

(Ordered) tree traversals are sequences

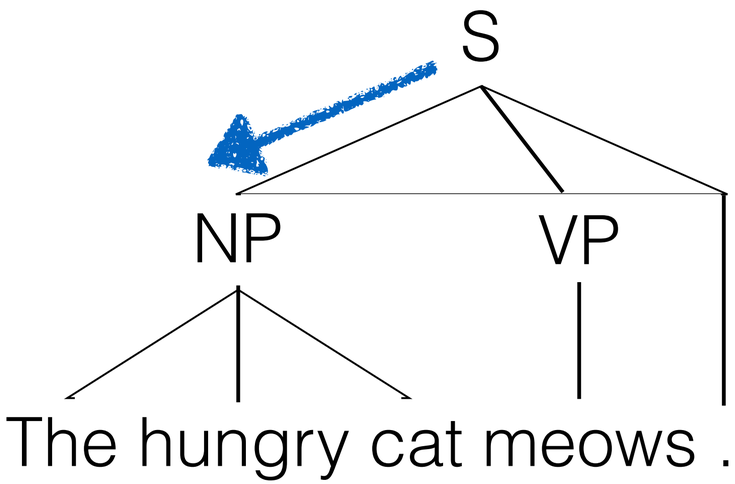


(Ordered) tree traversals are sequences



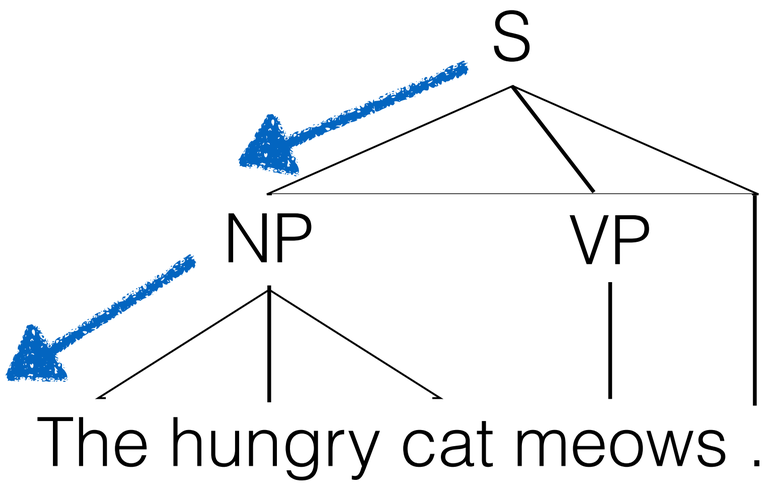
S(NP(The hungry cat) VP(meows) .)

(Ordered) tree traversals are sequences



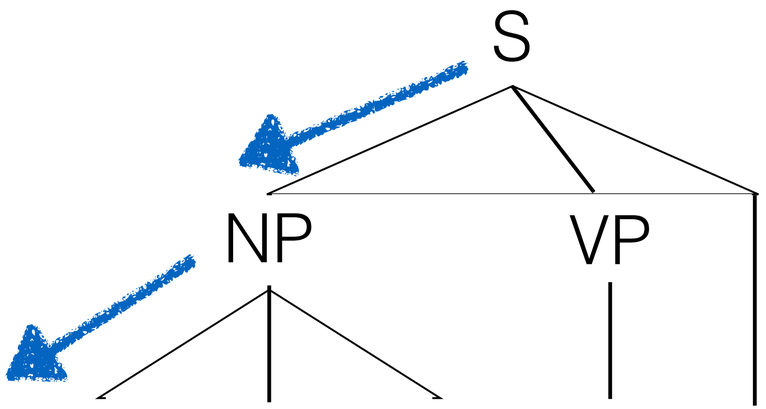
S(NP(The hungry cat) VP(meows) .)

(Ordered) tree traversals are sequences



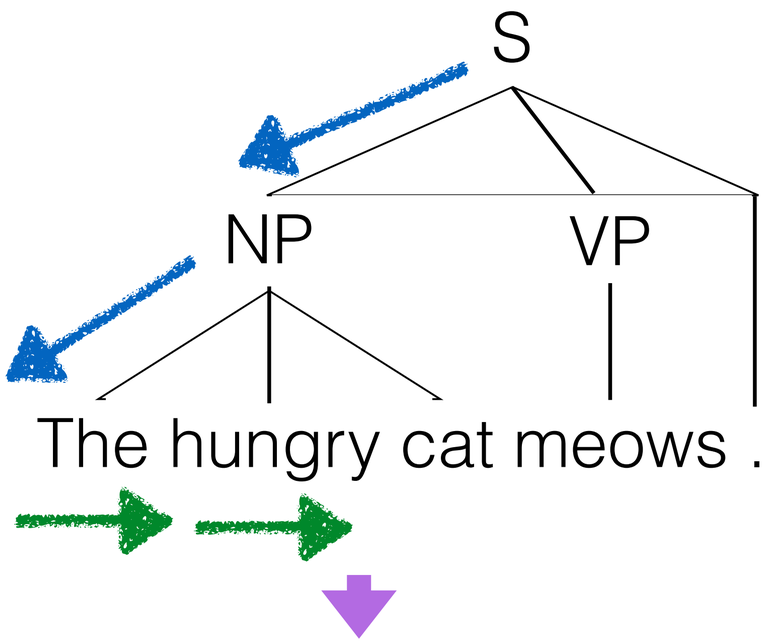
S(**NP(** The hungry cat) VP(meows) .)

(Ordered) tree traversals are sequences



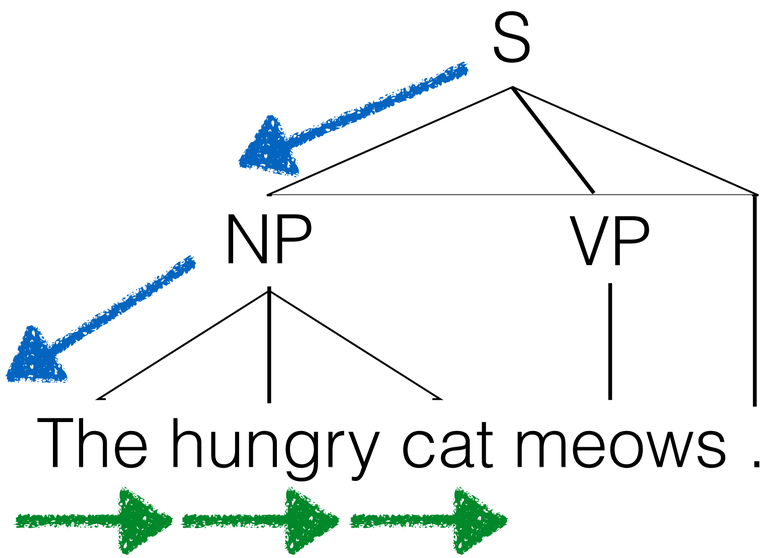
S(NP(**The** hungry cat) VP(meows) .)

(Ordered) tree traversals are sequences



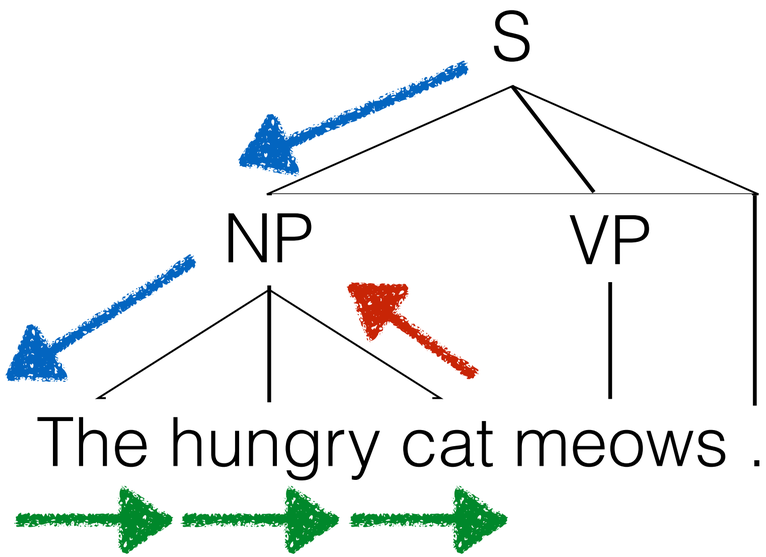
S(NP(The **hungry** cat) VP(meows) .)

(Ordered) tree traversals are sequences



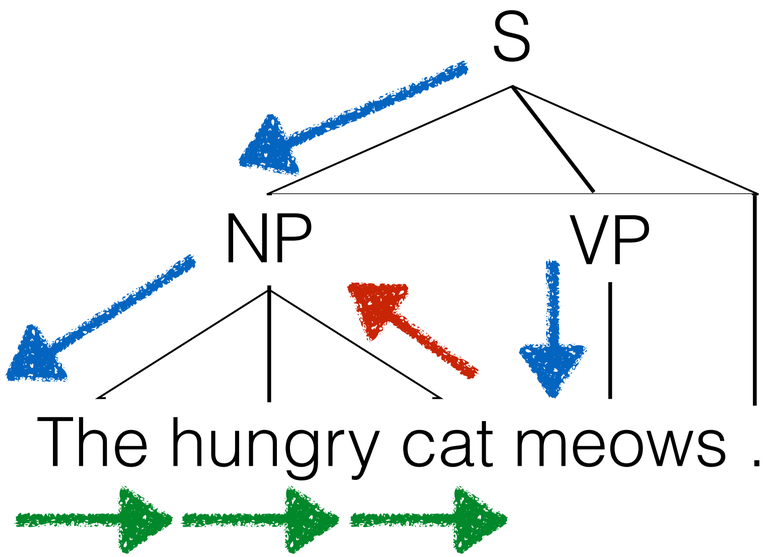
S(NP(The hungry **cat**) VP(meows) .)

(Ordered) tree traversals are sequences



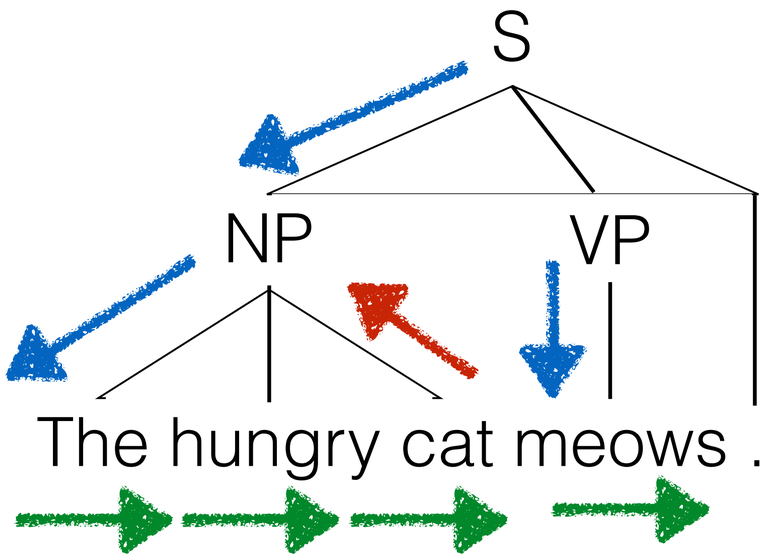
S(NP(The hungry cat) VP(meows) .)

(Ordered) tree traversals are sequences



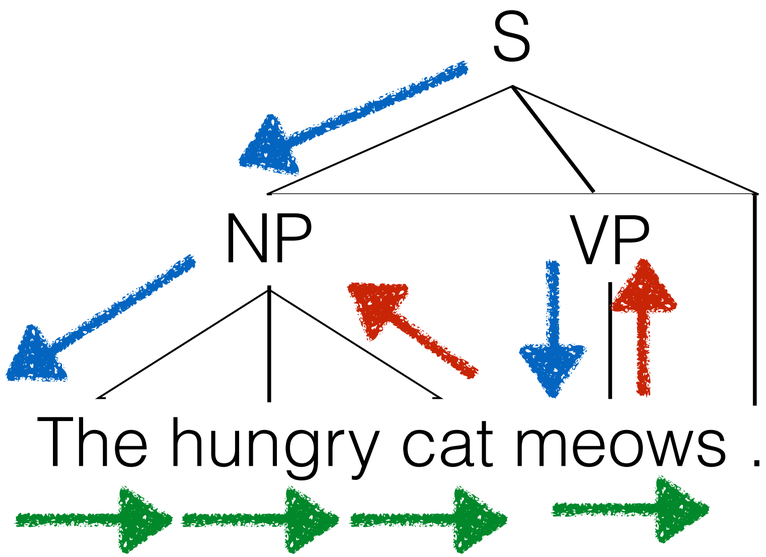
S(NP(The hungry cat) **VP(** meows) .)

(Ordered) tree traversals are sequences



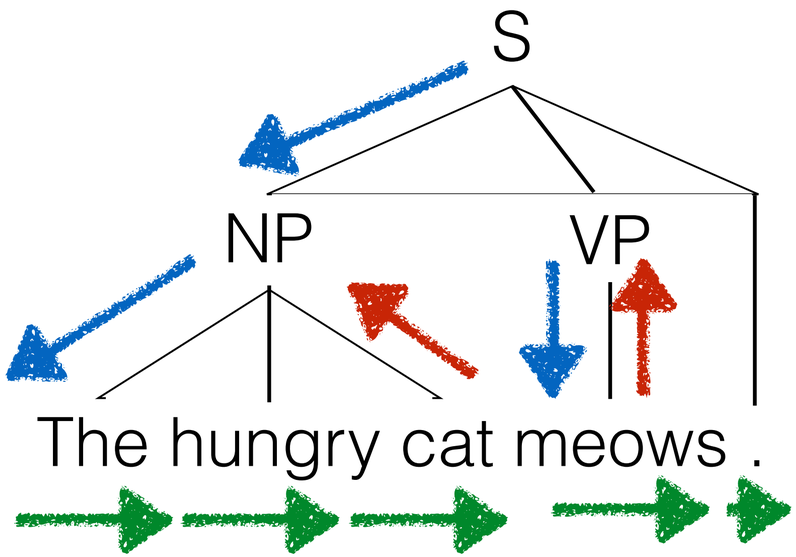
S(NP(The hungry cat) VP(**meows**) .)

(Ordered) tree traversals are sequences



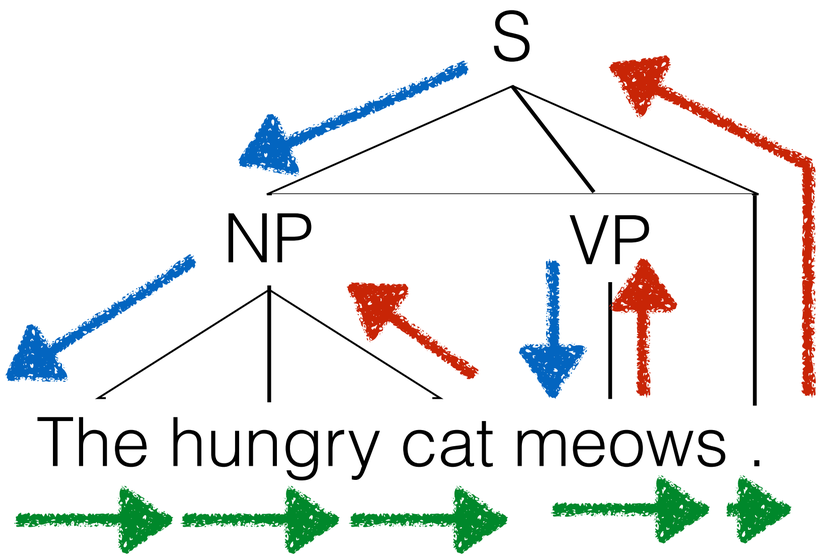
S(NP(The hungry cat) VP(meows) .)

(Ordered) tree traversals are sequences



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

(Ordered) tree traversals are sequences



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

Terminals	Stack	Action

Terminals	Stack	Action
		NT(S)

Terminals	Stack	Action
		NT(S**)**
	(S	**NT(**NP**)**

Terminals	Stack	Action
		NT(S**)**
	(S	**NT(**NP**)**
	(S (NP	

Terminals	Stack	Action
		NT(S**)**
	(S	**NT(**NP**)**
	(S (NP	**GEN(***The***)**

Terminals	Stack	Action
		NT(S)
	(S	**NT**(NP)
	(S (NP	**GEN**(*The*)
The	(S (NP *The*	

Terminals	Stack	Action
		NT(S**)**
	(S	**NT(**NP**)**
	(S (NP	**GEN(***The***)**
The	(S (NP *The*	**GEN(***hungry***)**

Terminals	Stack	Action
		NT(S**)**
	(S	**NT(**NP**)**
	(S (NP	**GEN(***The***)**
The	(S (NP *The*	**GEN(***hungry***)**
The hungry	(S (NP *The hungry*	

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

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

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

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

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

Compress “The hungry cat” into a single composite symbol

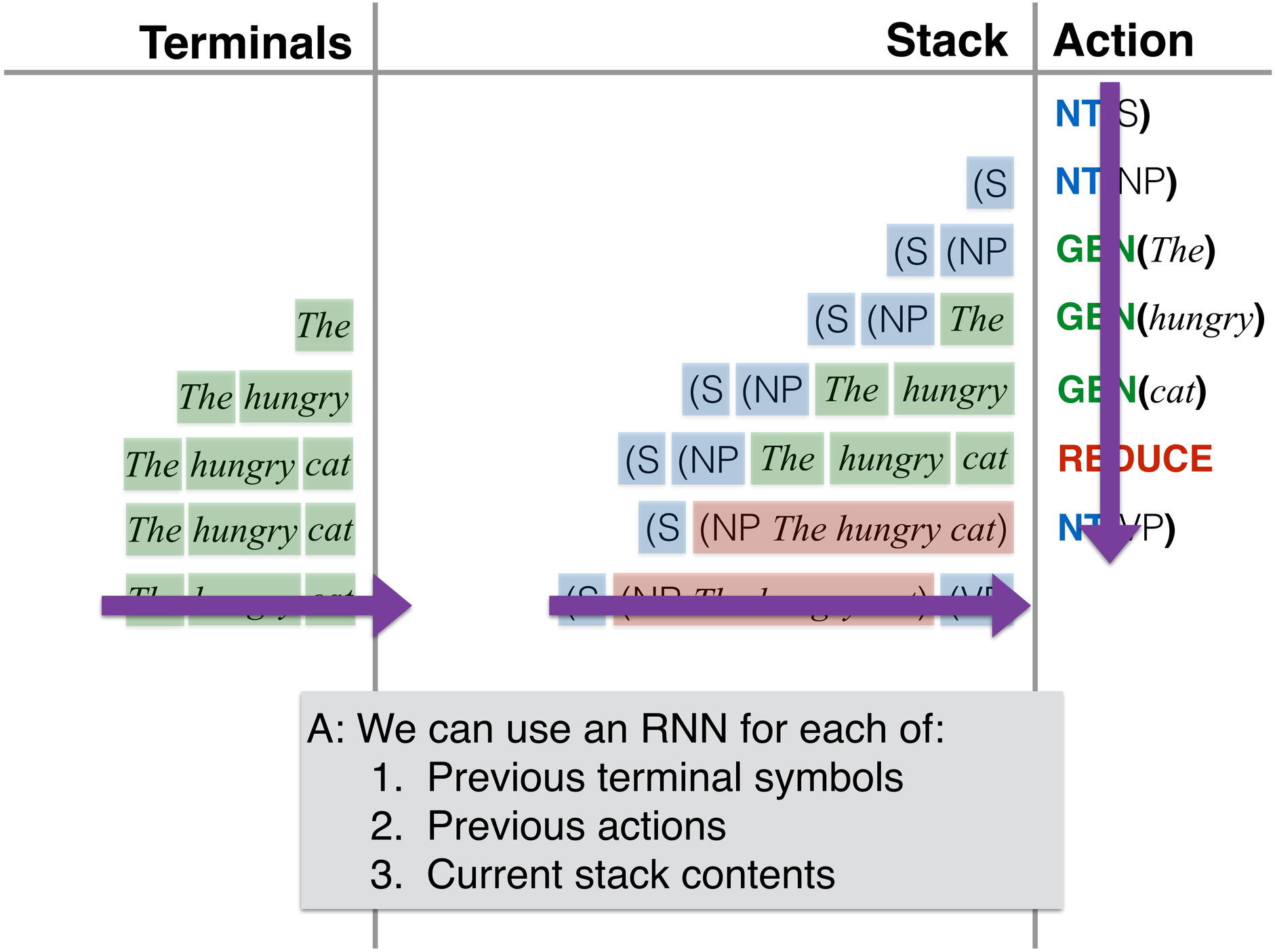
Terminals	Stack	Action
		NT(S)
	(S	**NT**(NP)
	(S (NP	**GEN**(*The*)
The	(S (NP *The*	**GEN**(*hungry*)
The hungry	(S (NP *The hungry*	**GEN**(*cat*)
The hungry cat	(S (NP *The hungry cat*	**REDUCE**
The hungry cat	(S (NP *The hungry cat*)	

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

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

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

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

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

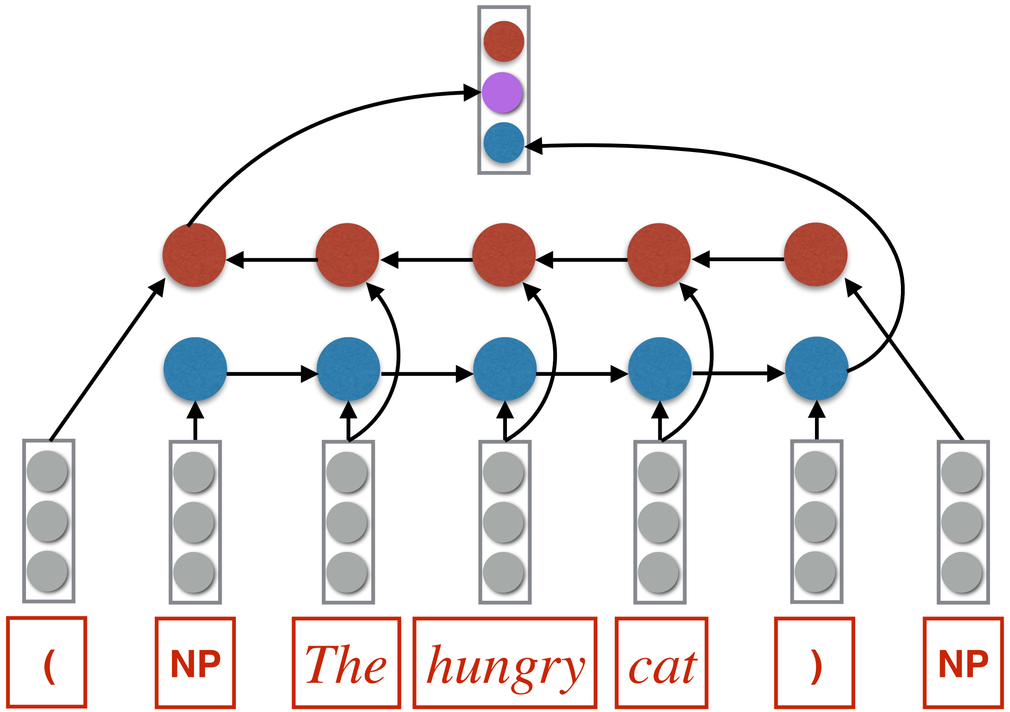
Terminals	Stack	Action
		NT(S)
	(S	**NT**(NP)
	(S (NP	**GEN**(*The*)
The	(S (NP *The*	**GEN**(*hungry*)
The hungry	(S (NP *The hungry*	**GEN**(*cat*)
The hungry cat	(S (NP *The hungry cat*	**REDUCE**
The hungry cat	(S (NP *The hungry cat*)	**NT**(VP)
The hungry cat	(S (NP *The hungry cat*) (VP	**GEN**(*meows*)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

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

Final stack symbol is
(a vector representation of)
the complete tree.

Syntactic Composition

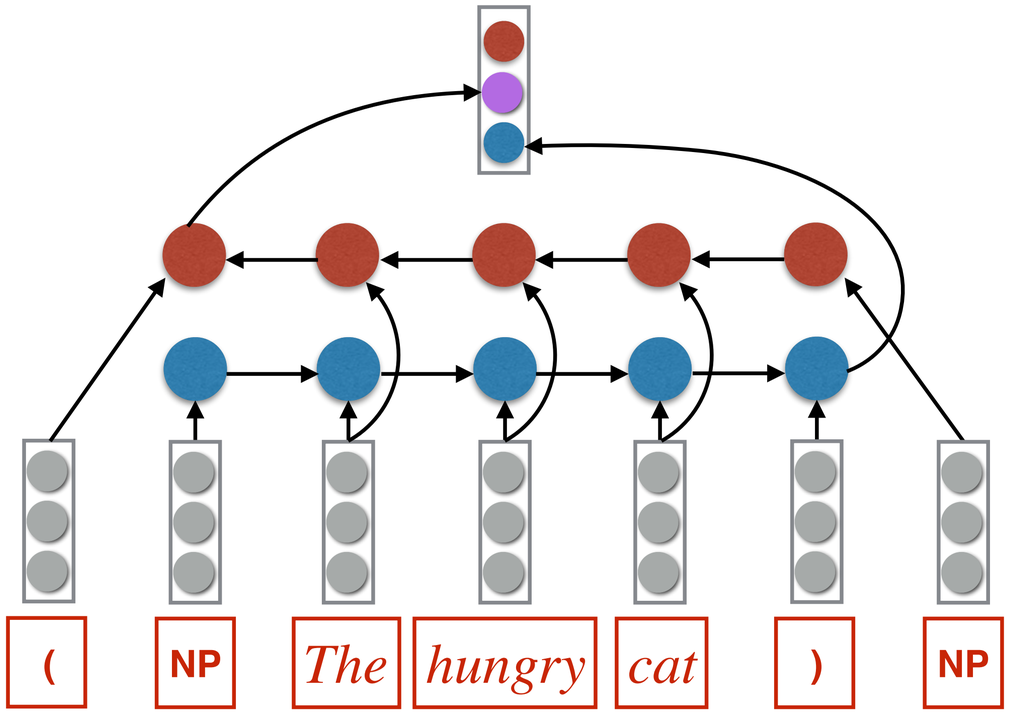
Need representation for: (NP *The hungry cat*)



Recursion

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

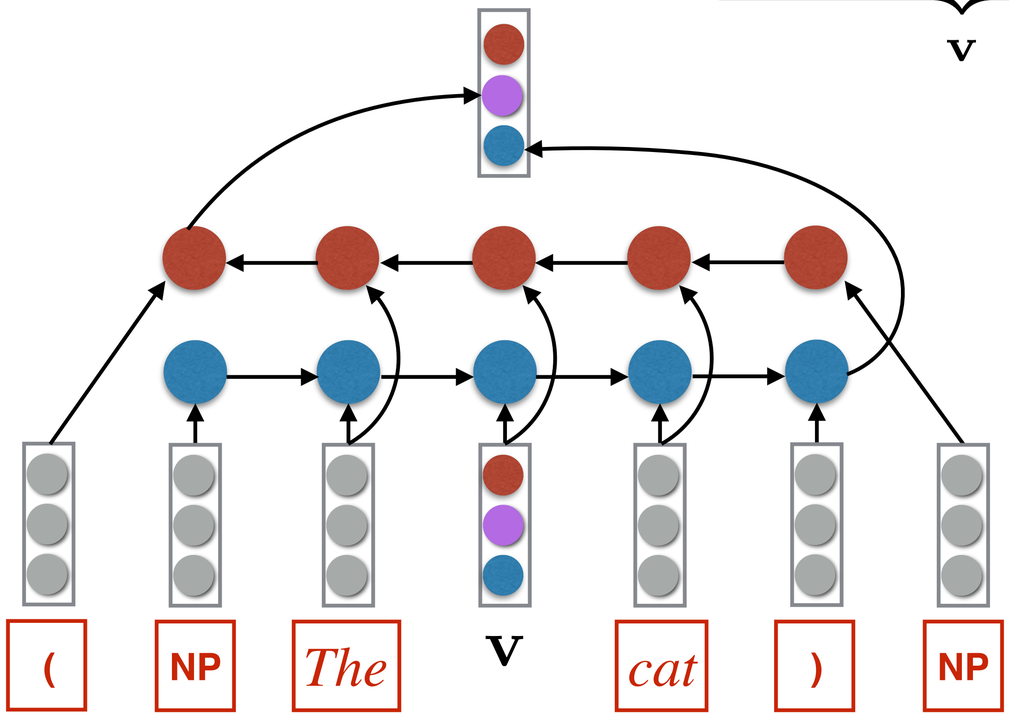
(NP *The* (ADJP *very hungry*) *cat*)



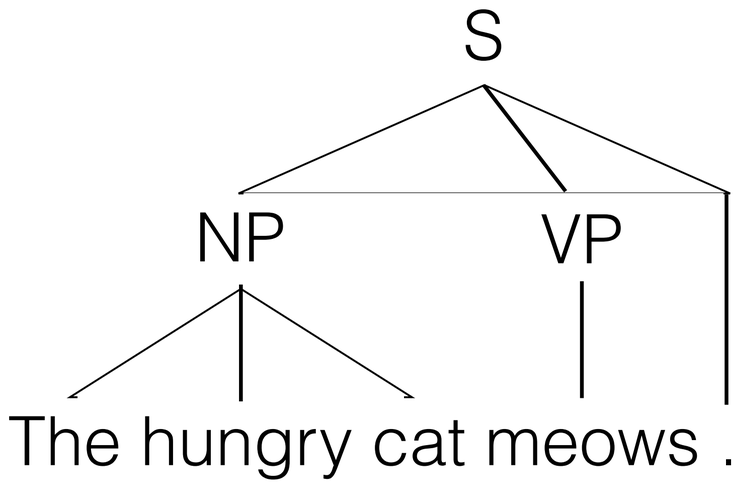
Recursion

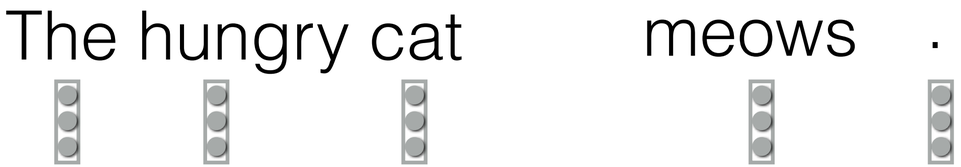
Need representation for: (NP *The hungry cat*)

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

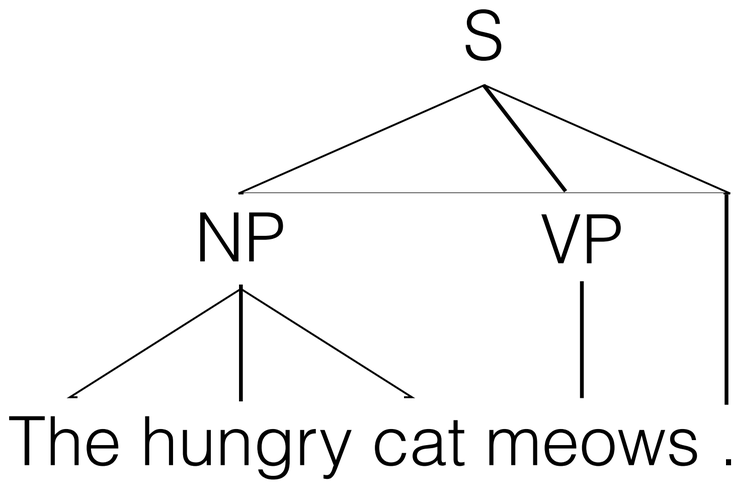


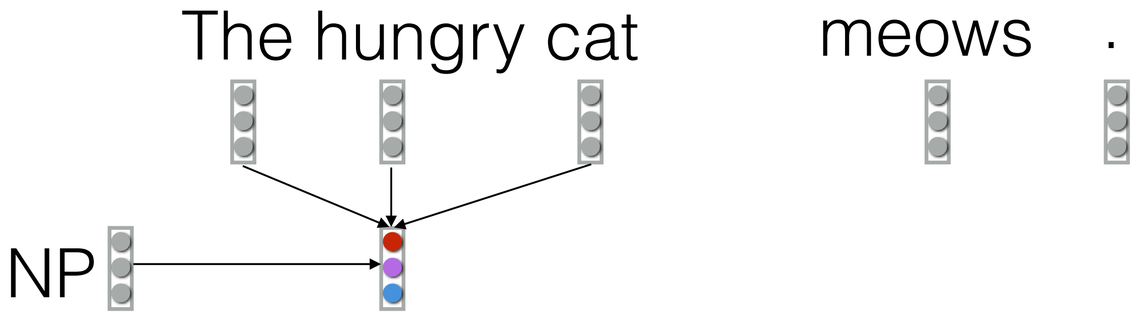
Stack symbols composed recursively mirror corresponding tree structure



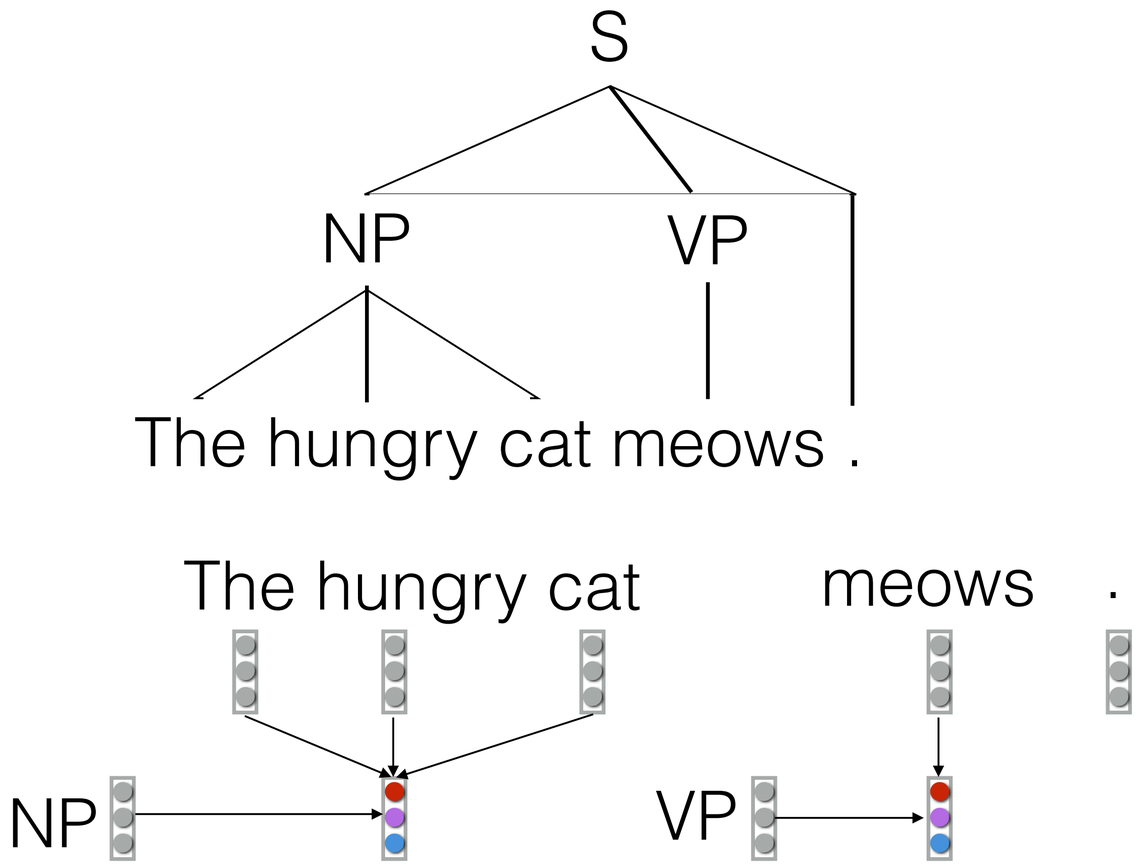


Stack symbols composed recursively mirror corresponding tree structure

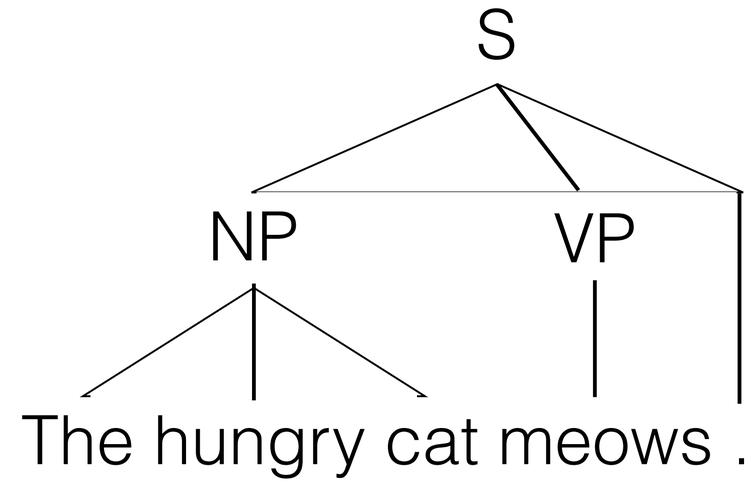




Stack symbols composed recursively mirror corresponding tree structure

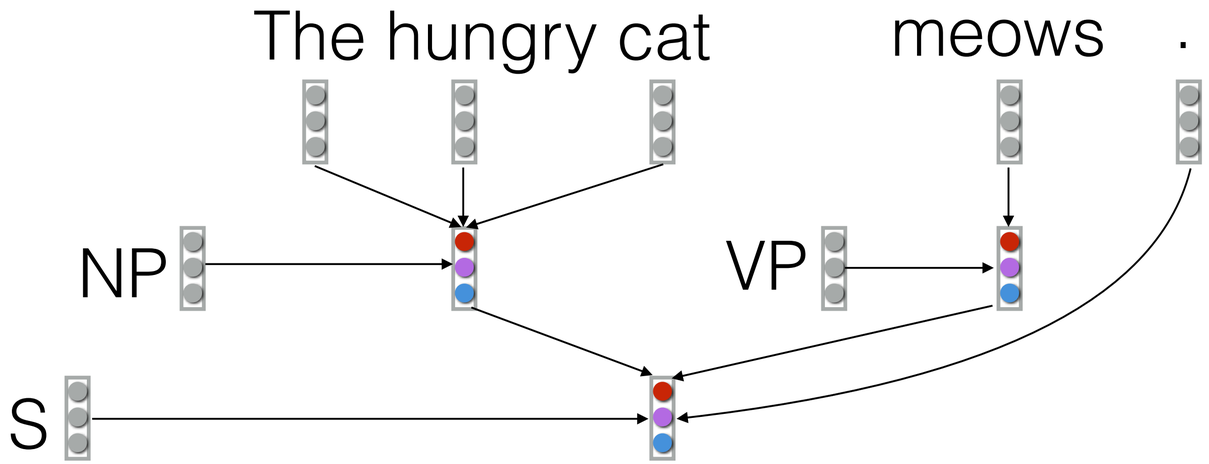


Stack symbols composed recursively mirror corresponding tree structure



Effect

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

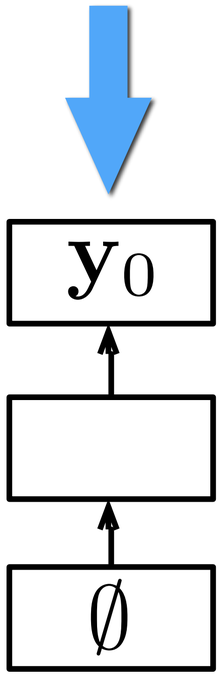


Implementing RNNGs
Stack RNNs

- 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

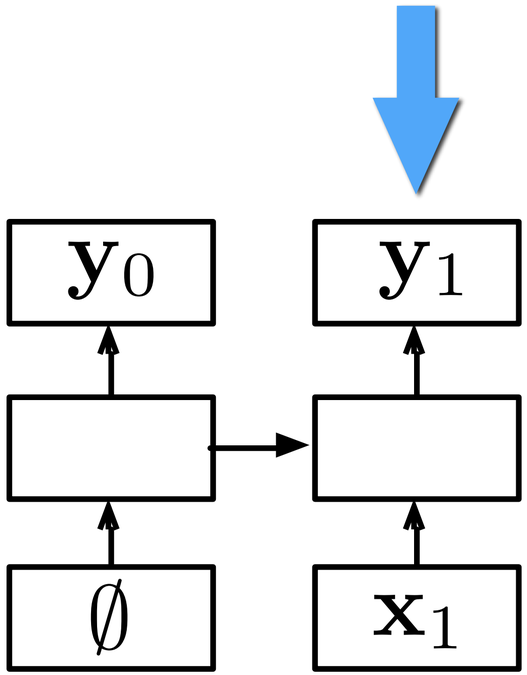
Stack RNNs



PUSH

Implementing RNNGs

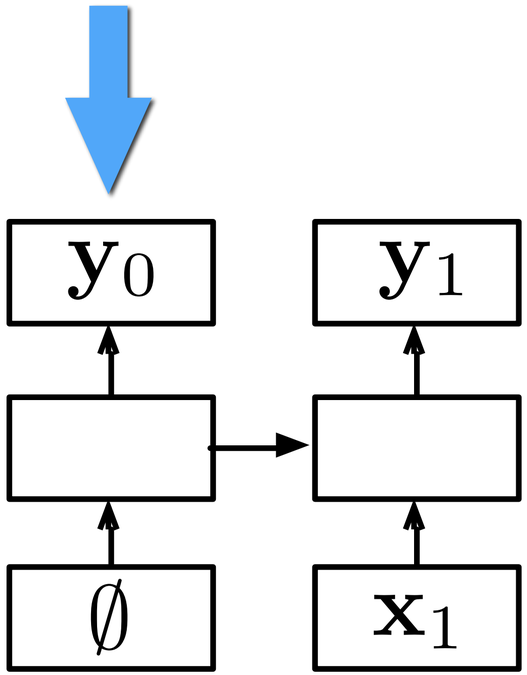
Stack RNNs



POP

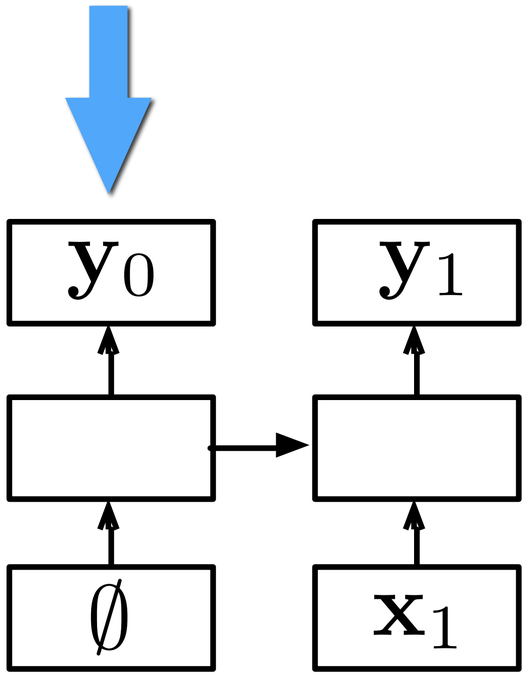
Implementing RNNGs

Stack RNNs



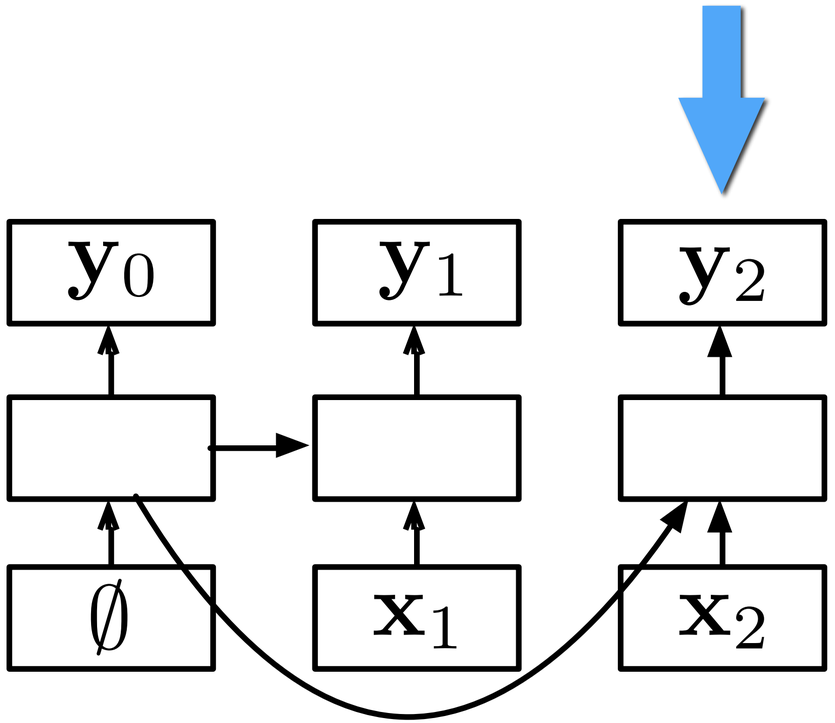
Implementing RNNGs

Stack RNNs



PUSH

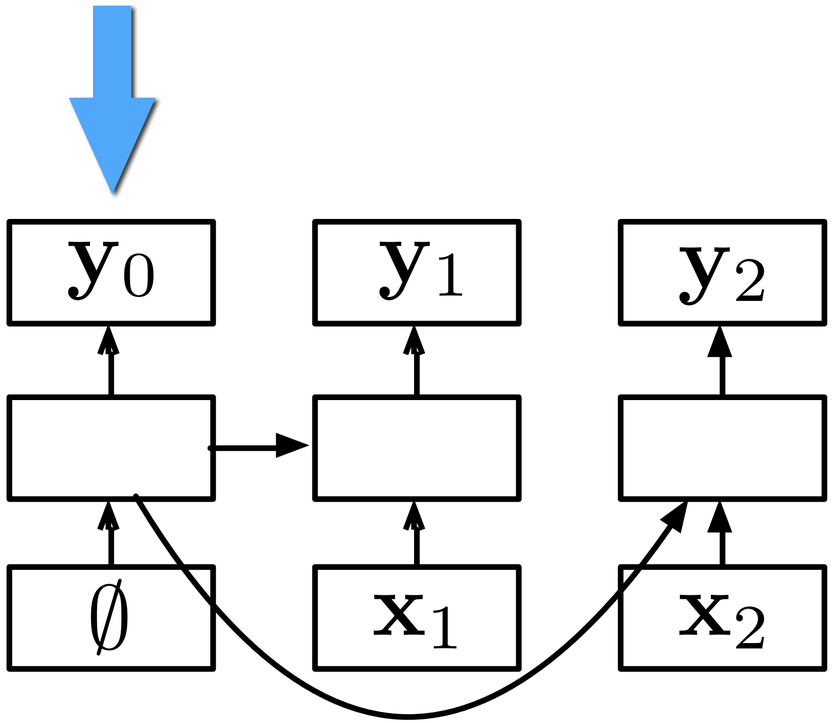
Implementing RNNGs
Stack RNNs



POP

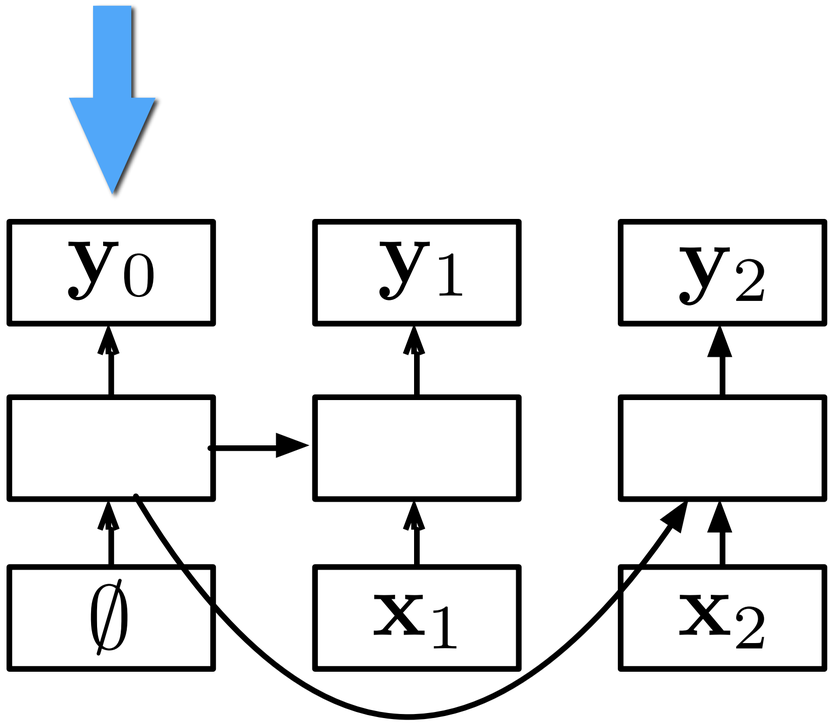
Implementing RNNGs

Stack RNNs



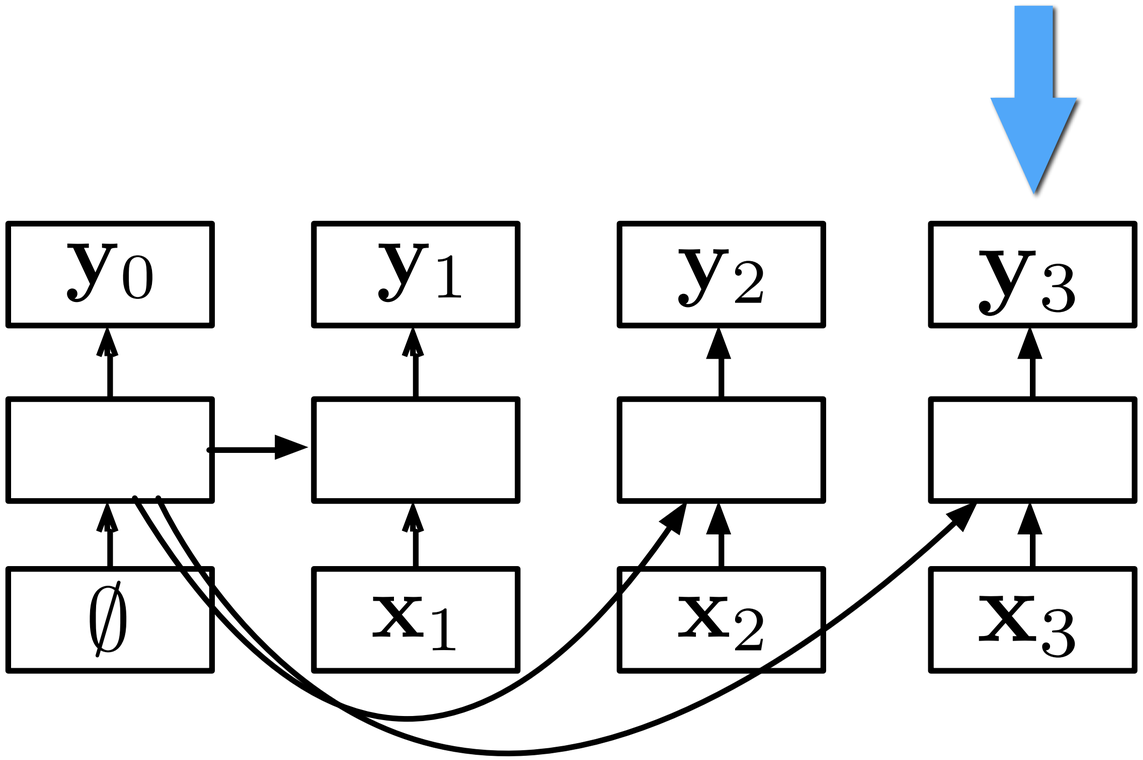
Implementing RNNGs

Stack RNNs

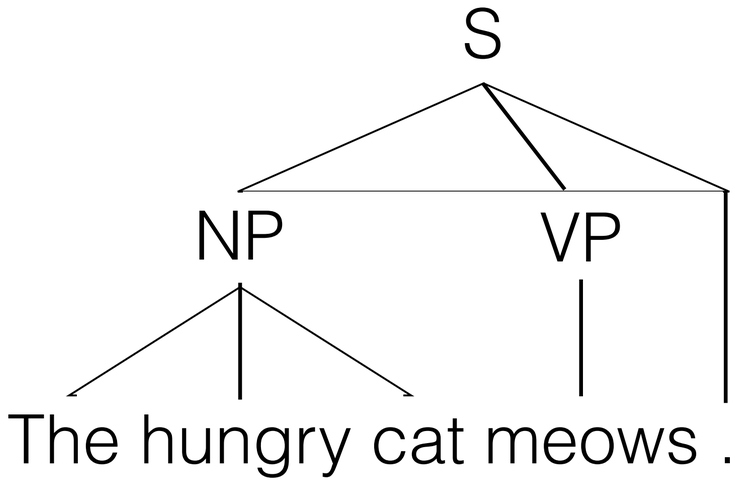


PUSH

Implementing RNNGs
Stack RNNs



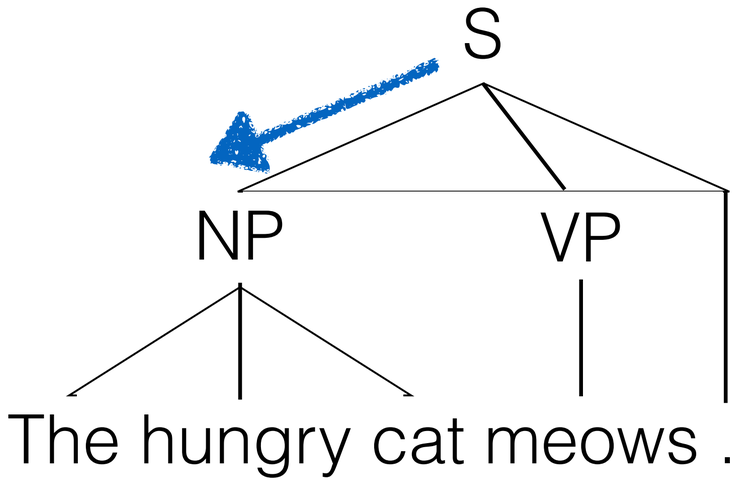
The evolution of the stack LSTM over time mirrors tree structure



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

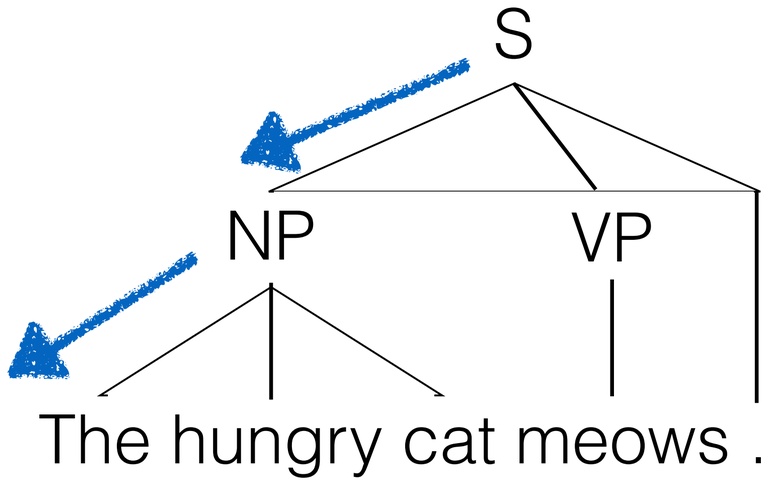
stack top

The evolution of the stack LSTM over time mirrors tree structure

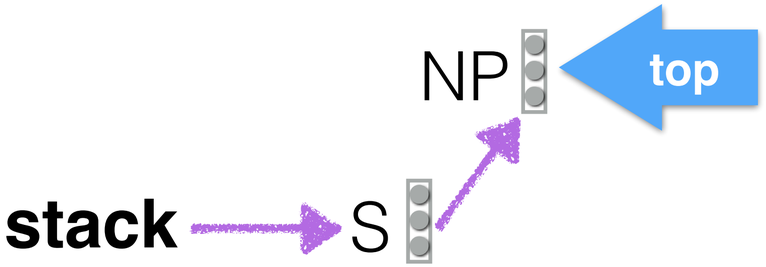


S(NP(The hungry cat) VP(meows) .)

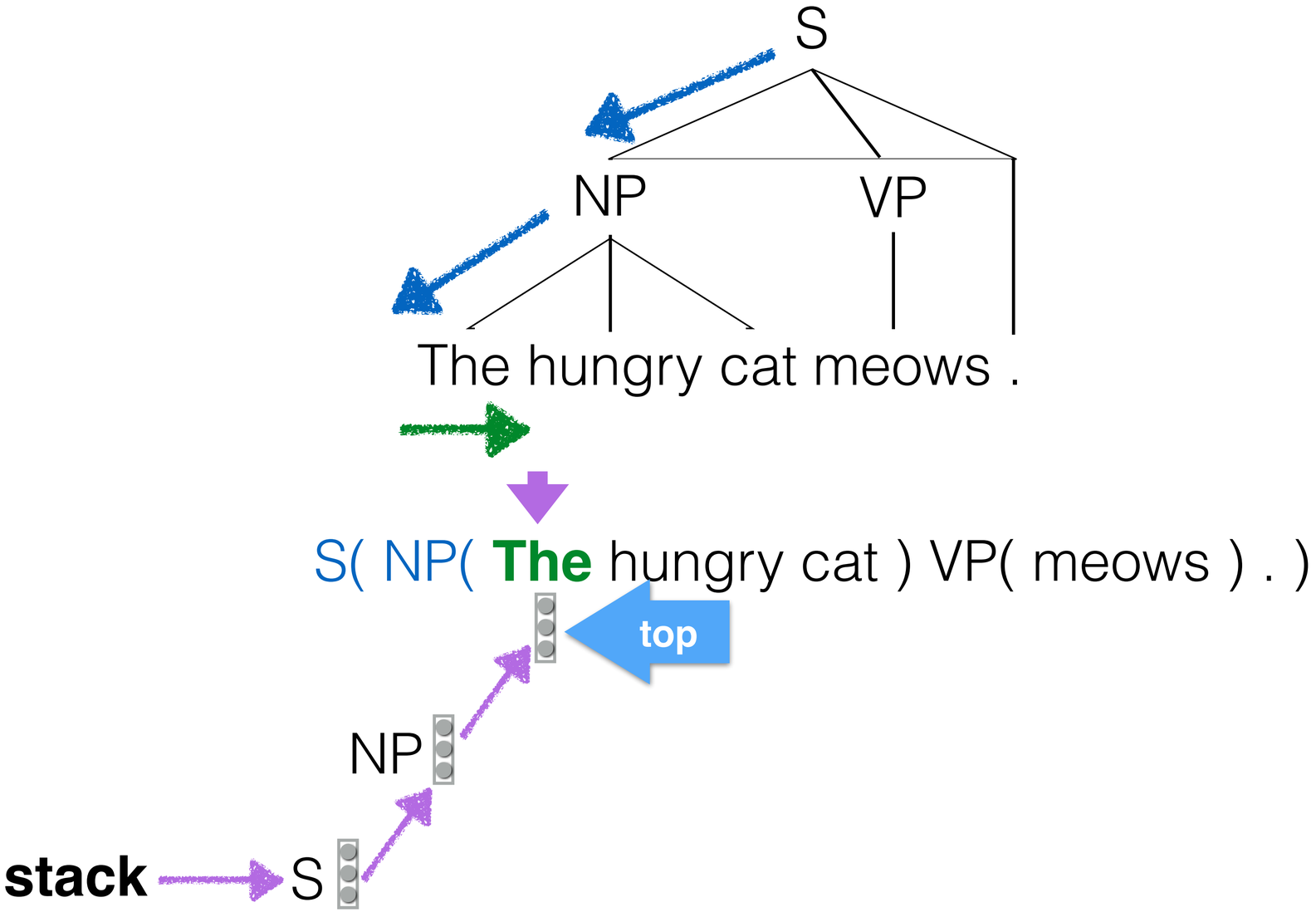
The evolution of the stack LSTM over time mirrors tree structure



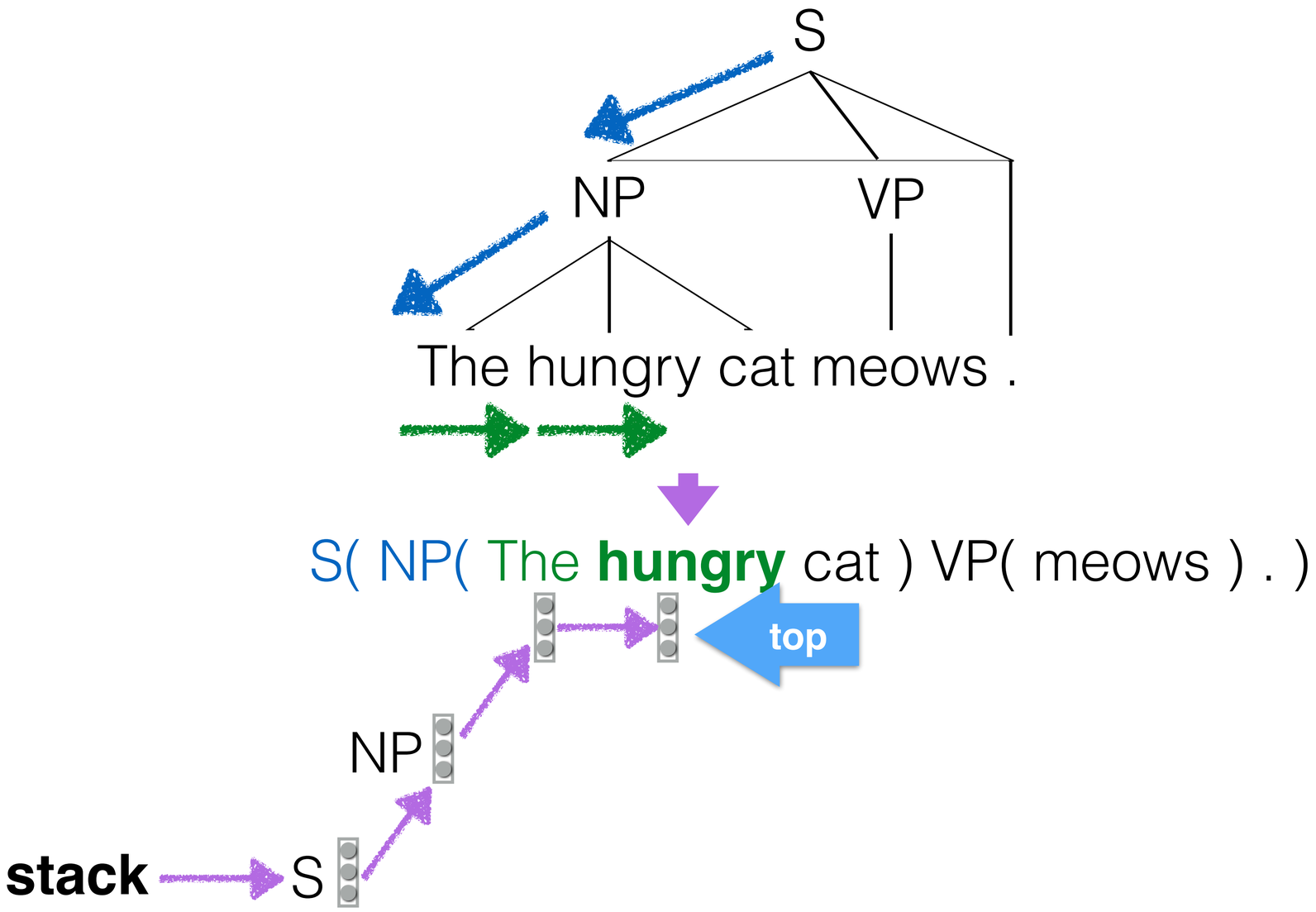
S(**NP(** The hungry cat) VP(meows) .)



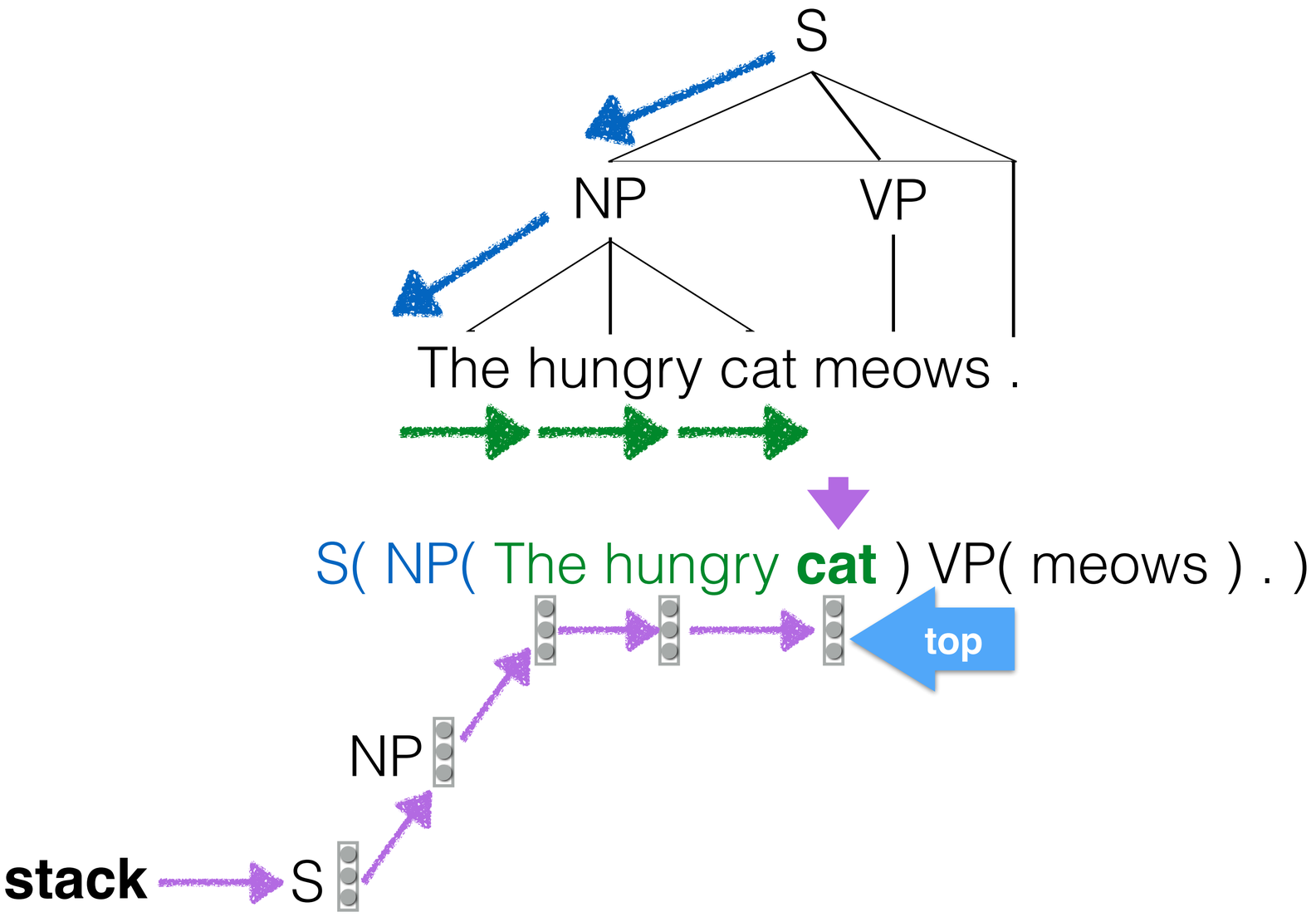
The evolution of the stack LSTM over time mirrors tree structure



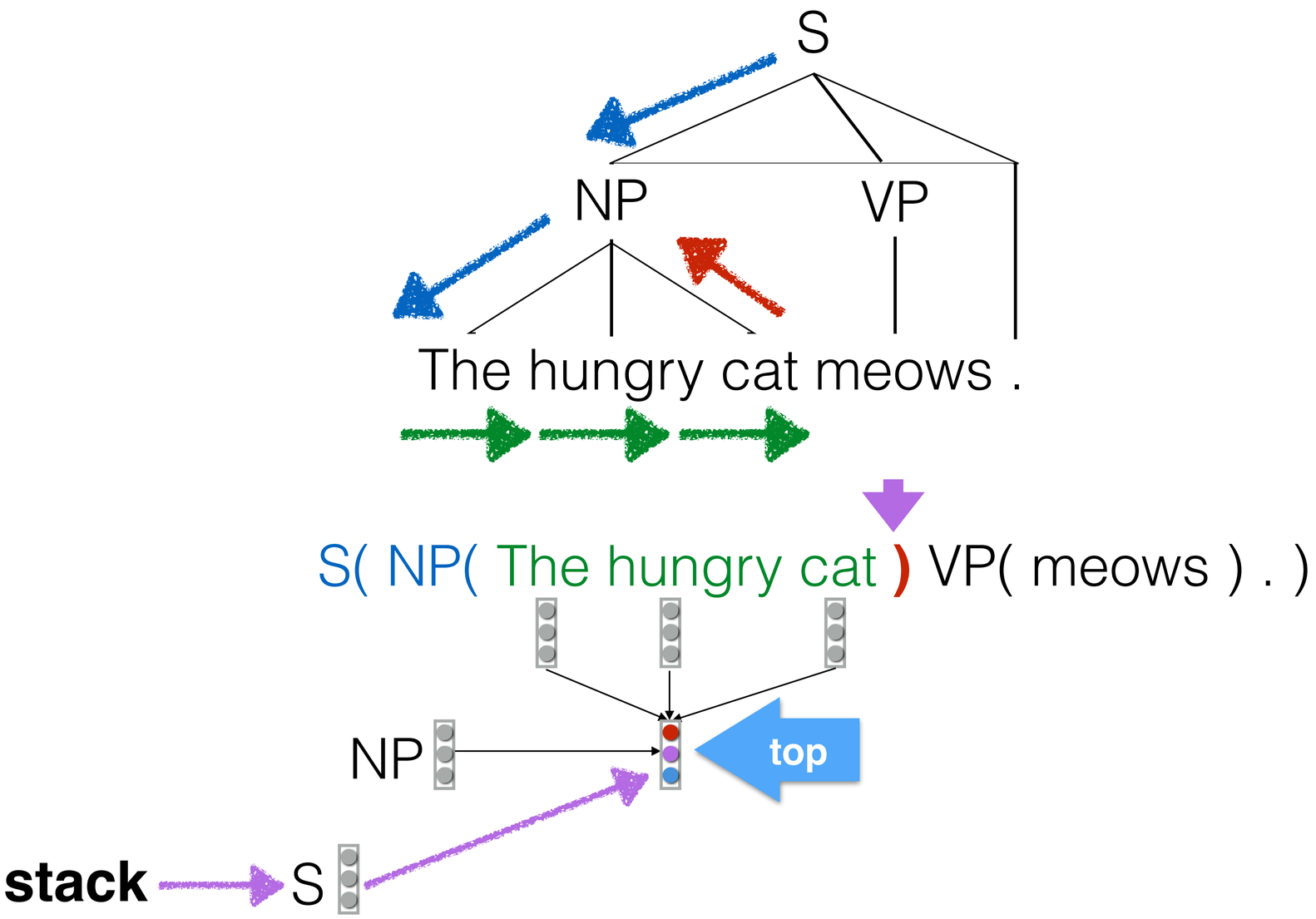
The evolution of the stack LSTM over time mirrors tree structure



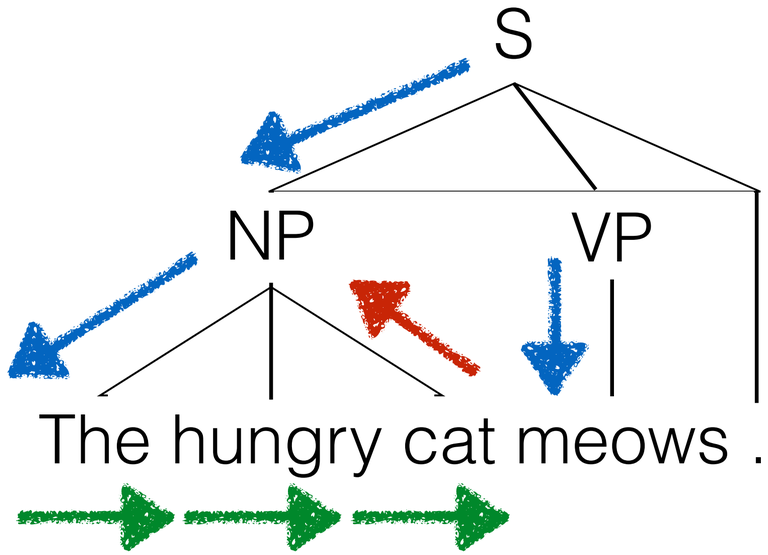
The evolution of the stack LSTM over time mirrors tree structure



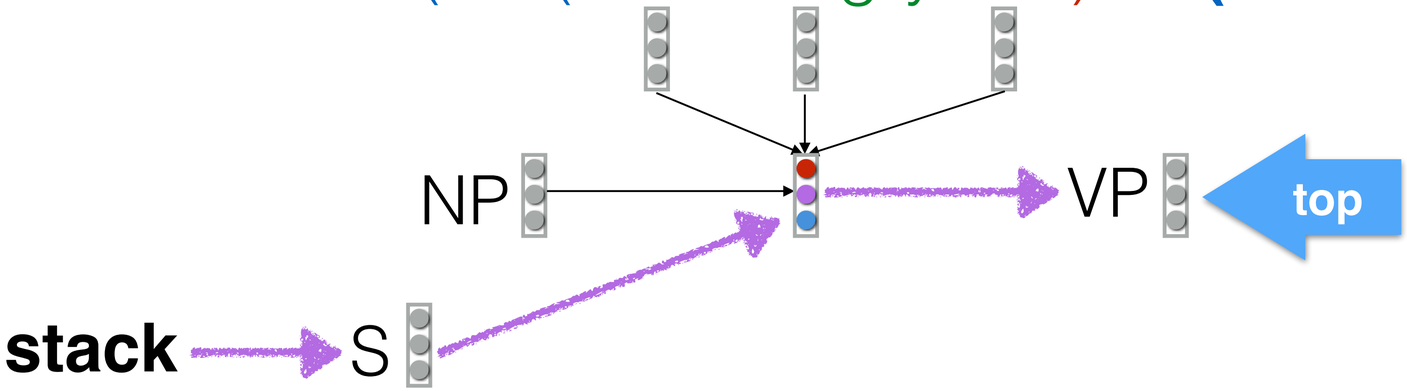
The evolution of the stack LSTM over time mirrors tree structure



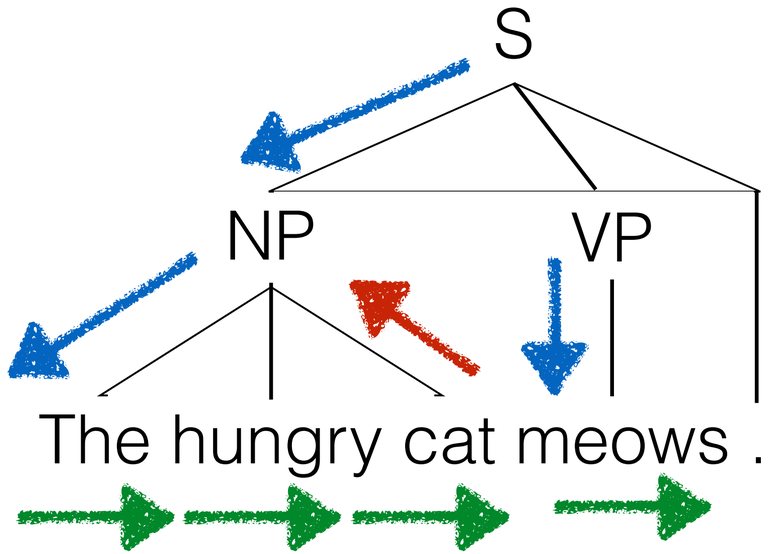
The evolution of the stack LSTM over time mirrors tree structure

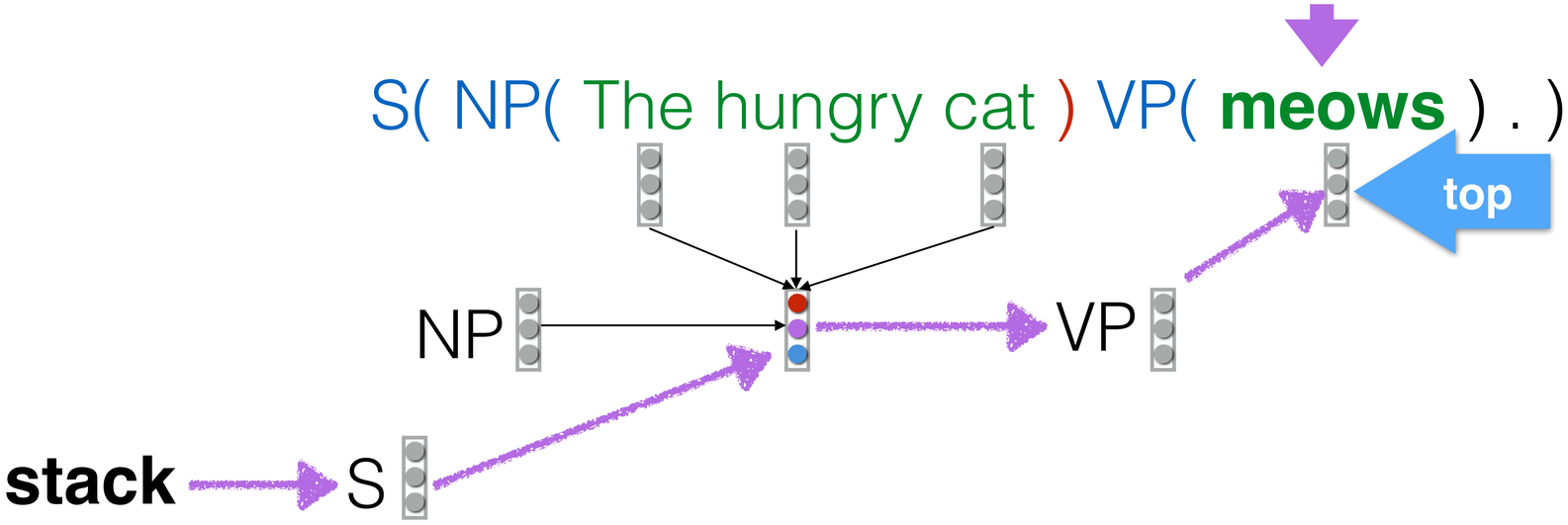


S(NP(The hungry cat) **VP(** meows) .)

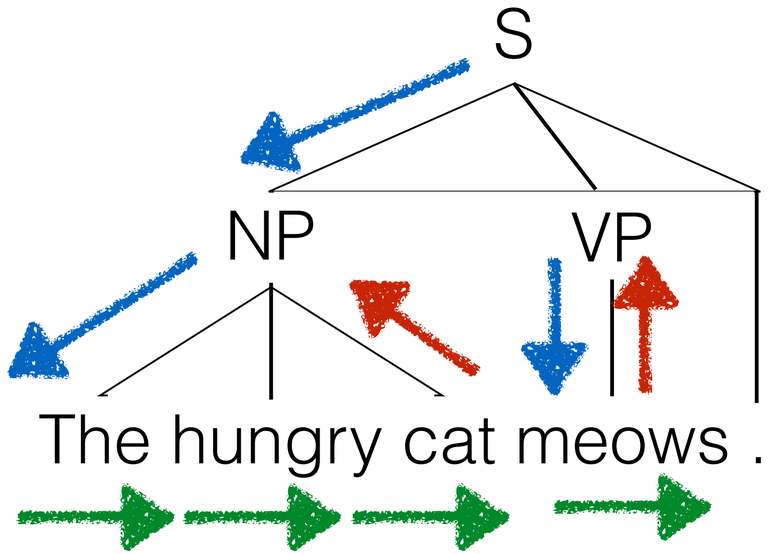


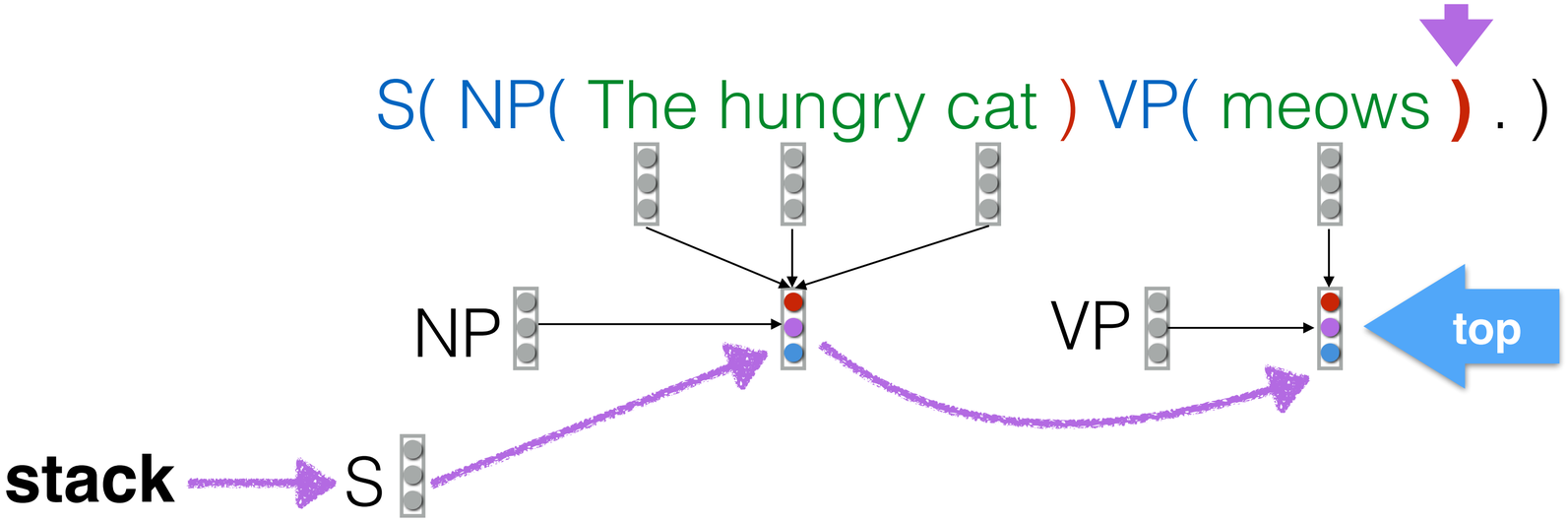
The evolution of the stack LSTM over time mirrors tree structure



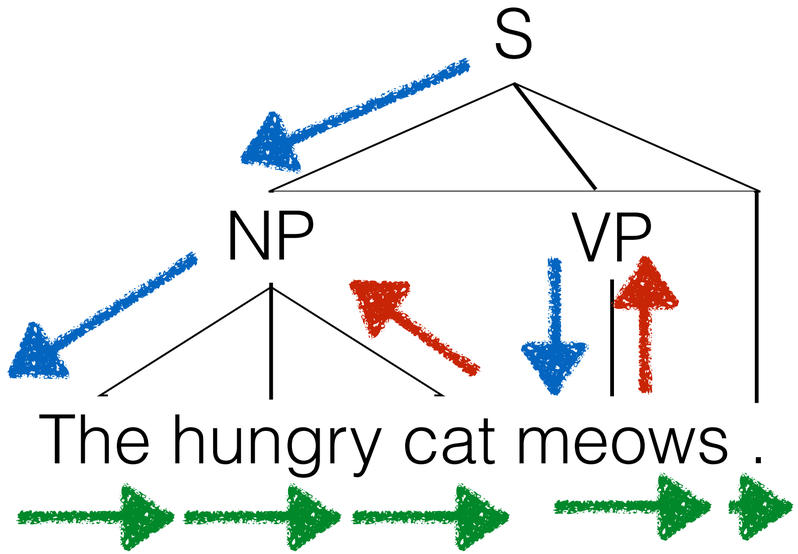


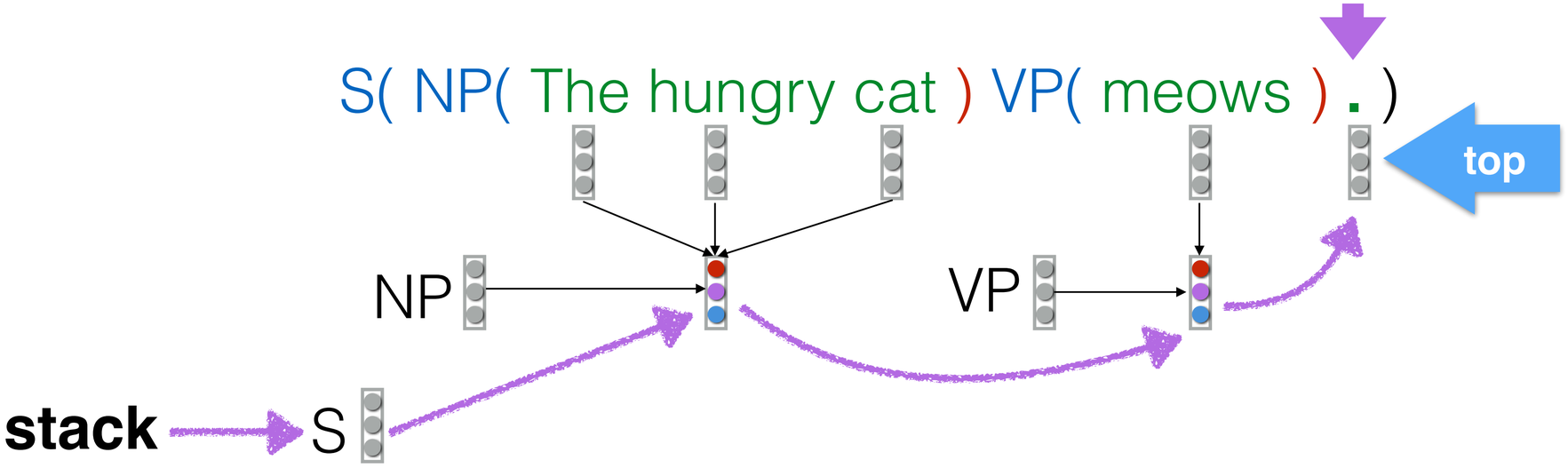
The evolution of the stack LSTM over time mirrors tree structure



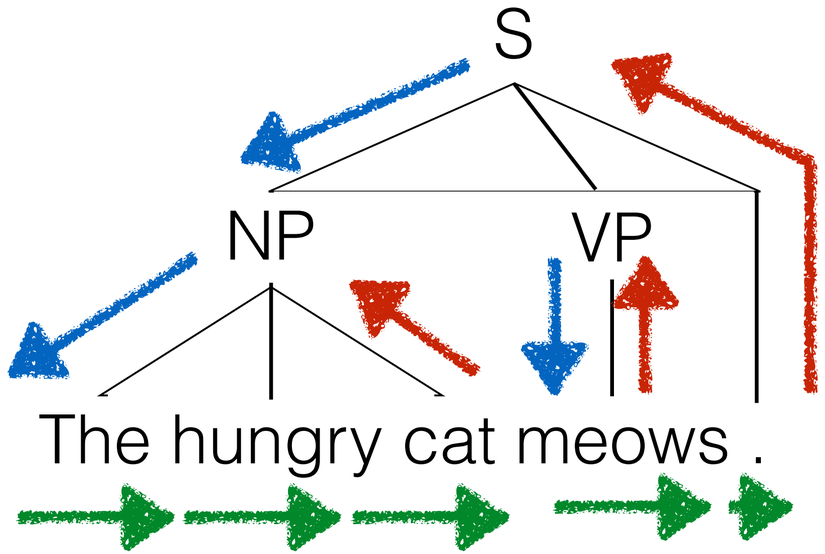


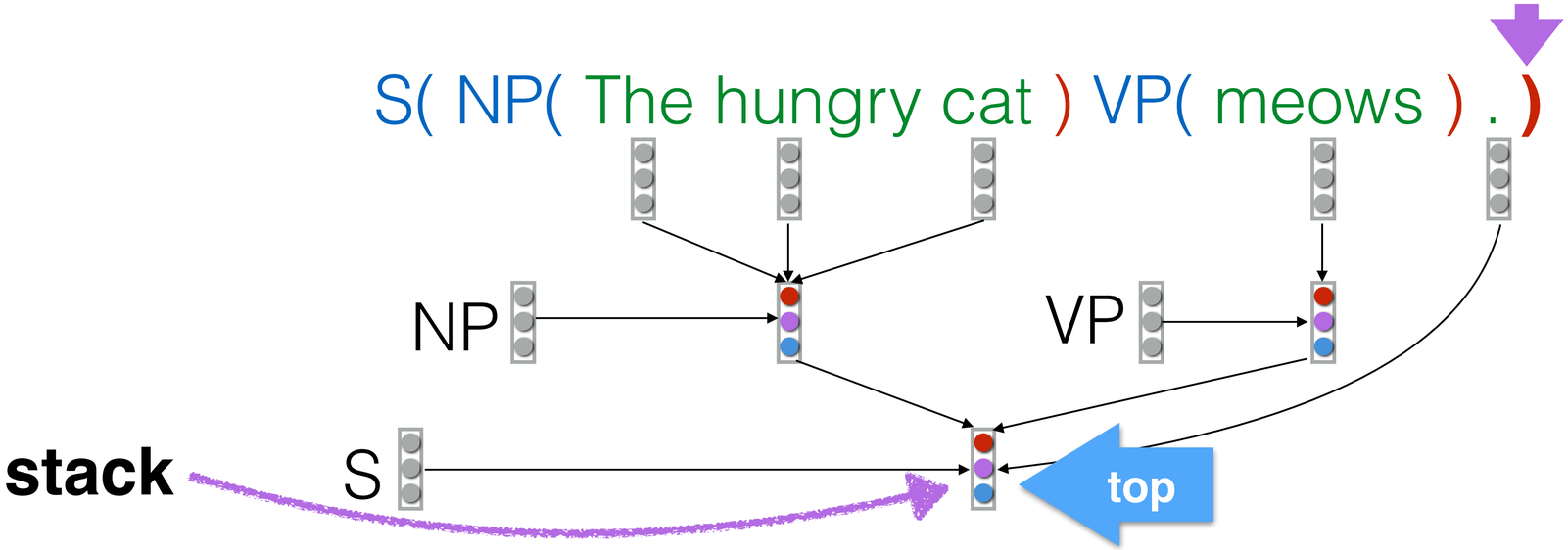
The evolution of the stack LSTM over time mirrors tree structure



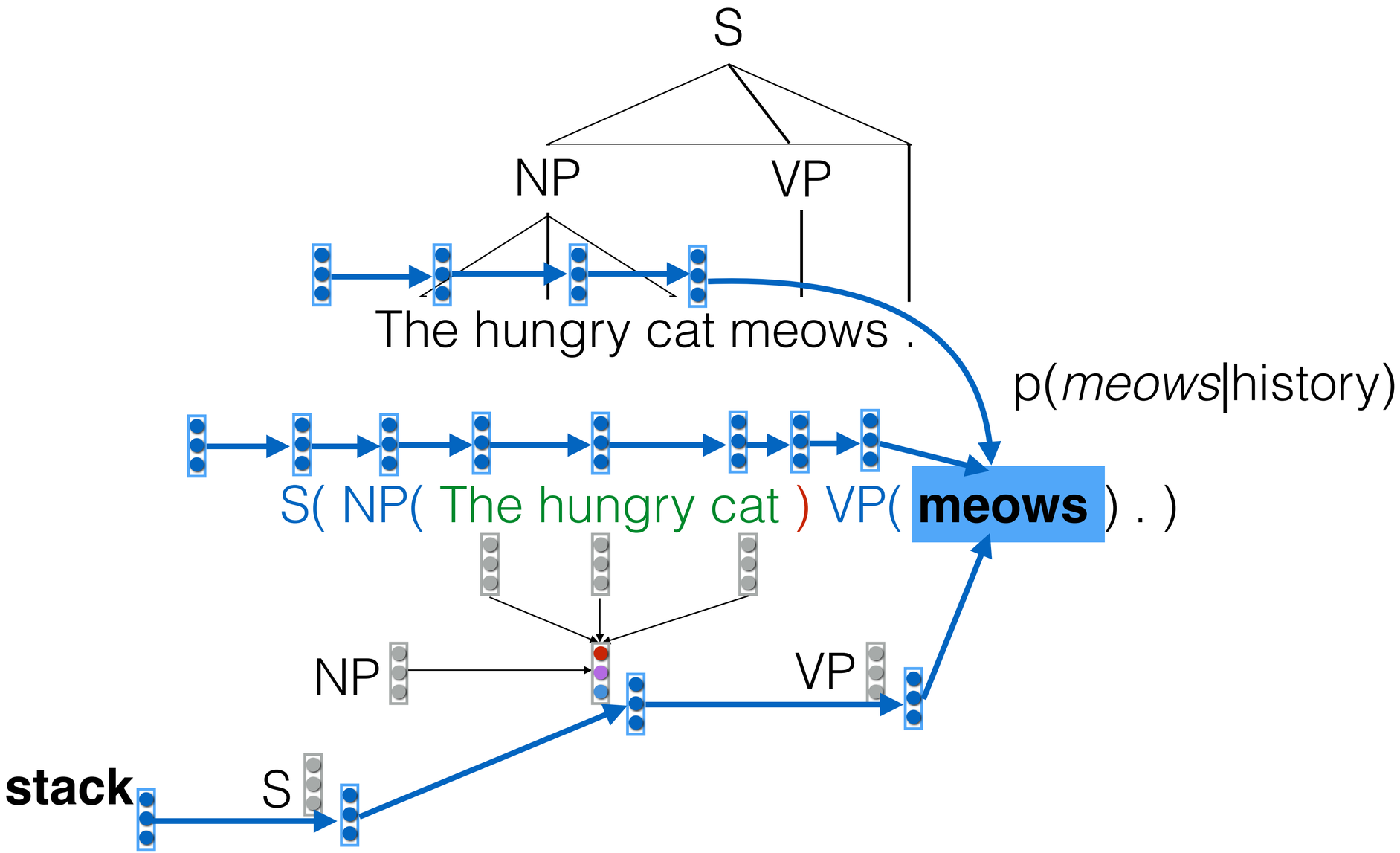


The evolution of the stack LSTM over time mirrors tree structure



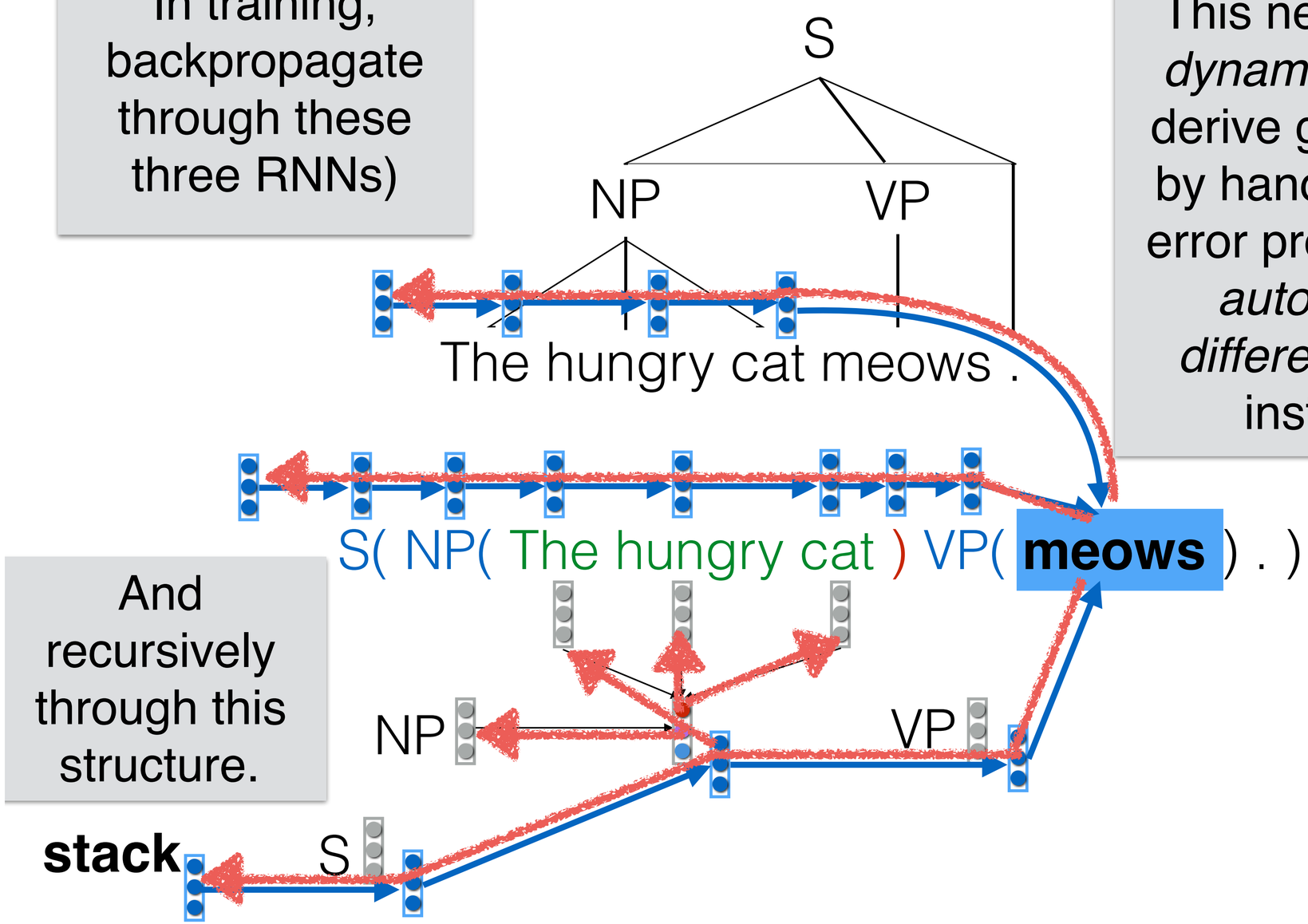


Each word is conditioned on history represented by a trio of RNNs



Train with backpropagation through structure

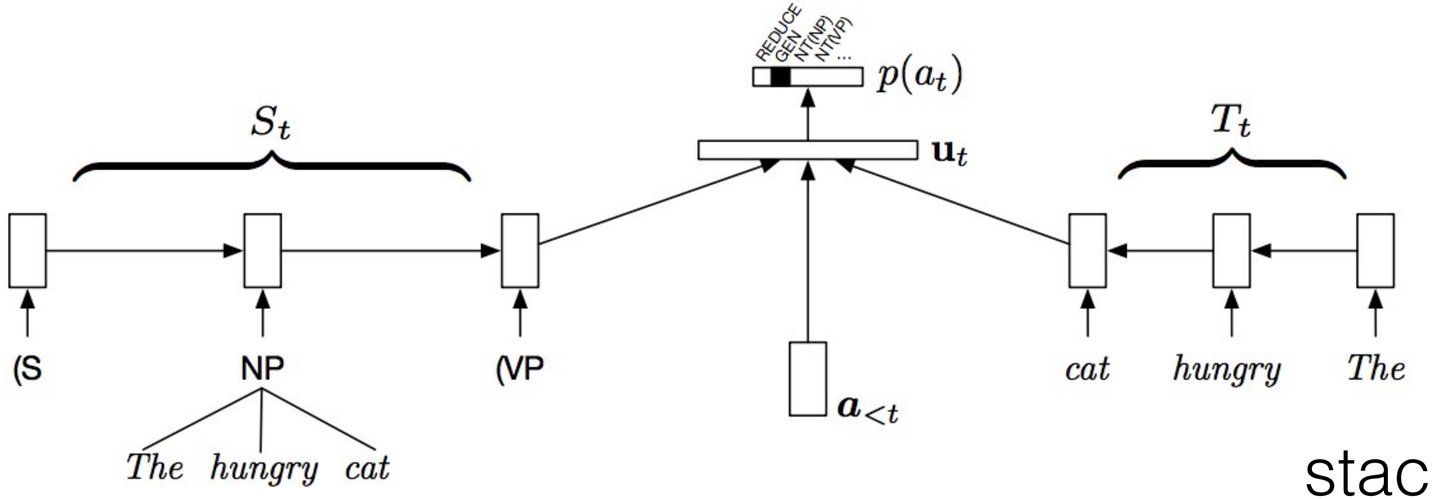
In training, backpropagate through these three RNNs)



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

And recursively through this structure.

Complete model



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

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

stack

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

output (buffer)

action history

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 $\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x})$ 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

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 $\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x})$ 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

Importance Sampling

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

s.t. (i) $p(\boldsymbol{x}, \boldsymbol{y}) > 0 \implies q(\boldsymbol{y} \mid \boldsymbol{x}) > 0$

(ii) $\boldsymbol{y} \sim q(\boldsymbol{y} \mid \boldsymbol{x})$ is tractable and

(iii) $q(\boldsymbol{y} \mid \boldsymbol{x})$ is tractable

Importance Sampling

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

s.t. (i) $p(\boldsymbol{x}, \boldsymbol{y}) > 0 \implies q(\boldsymbol{y} \mid \boldsymbol{x}) > 0$

(ii) $\boldsymbol{y} \sim q(\boldsymbol{y} \mid \boldsymbol{x})$ is tractable and

(iii) $q(\boldsymbol{y} \mid \boldsymbol{x})$ is tractable

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

Importance Sampling

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

s.t. (i) $p(\boldsymbol{x}, \boldsymbol{y}) > 0 \implies q(\boldsymbol{y} \mid \boldsymbol{x}) > 0$

(ii) $\boldsymbol{y} \sim q(\boldsymbol{y} \mid \boldsymbol{x})$ is tractable and

(iii) $q(\boldsymbol{y} \mid \boldsymbol{x})$ is tractable

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

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

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

Importance Sampling

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

Importance Sampling

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

Replace this expectation with its Monte Carlo estimate.

$$
\boldsymbol{y}^{(i)} \sim q(\boldsymbol{y} \mid \boldsymbol{x}) \quad \text{for } i \in \{1, 2, \ldots, N\}
$$

Importance Sampling

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

Replace this expectation with its Monte Carlo estimate.

$$
\boldsymbol{y}^{(i)} \sim q(\boldsymbol{y} \mid \boldsymbol{x}) \quad \text{for } i \in \{1, 2, \ldots, N\}
$$

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

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?

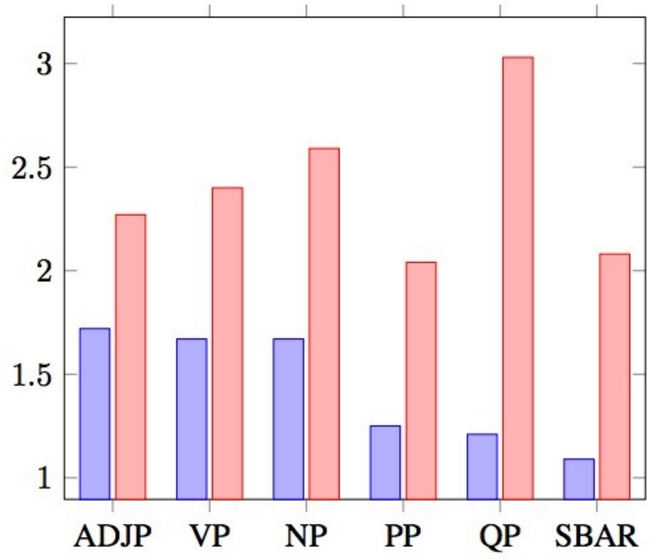


Figure 3: Average perplexity of the learned attention vectors on the test set (blue), as opposed to the average perplexity of the uniform distribution (red), computed for each major phrase type.

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?

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?

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.