Constituent Syntax (II)

CS 685, Spring 2021

Advanced Topics in Natural Language Processing

http://brenocon.com/cs685

https://people.cs.umass.edu/~brenocon/cs685_s21/

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

- Syntax: how do words structurally combine to form sentences and meaning?

- Representations
  - Constituents
    - [the big dogs] chase cats
    - [colorless green clouds] chase cats
  - Dependencies
    - The **dog** ← **chased** the cat.
    - My **dog**, a big old one, **chased** the cat.

- Idea of a *grammar (G)*: global template for how sentences / utterances / phrases ***w*** are formed, via latent syntactic structure ***y***
  - Linguistics: what do G and P(w,y | G) look like?
  - Generation: score with, or sample from, P(w, y | G)
  - Parsing: predict P(y | w, G)

- Explicit grammars: How to parse, give one?
- Learning a grammar
- Do RNN/Transformers implicitly learn constituency syntax?

→ RNN Grammar

Parsing with a CFG

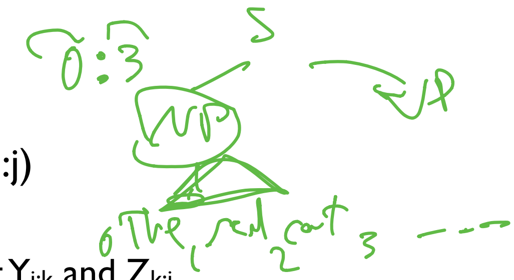
- Task: given text and a CFG, answer:
  - Does there exist at least one parse? "Is it grammatical?"
  - Enumerate parses (backpointers)
- Ambiguity is key problem: there exist multiple possible analyses

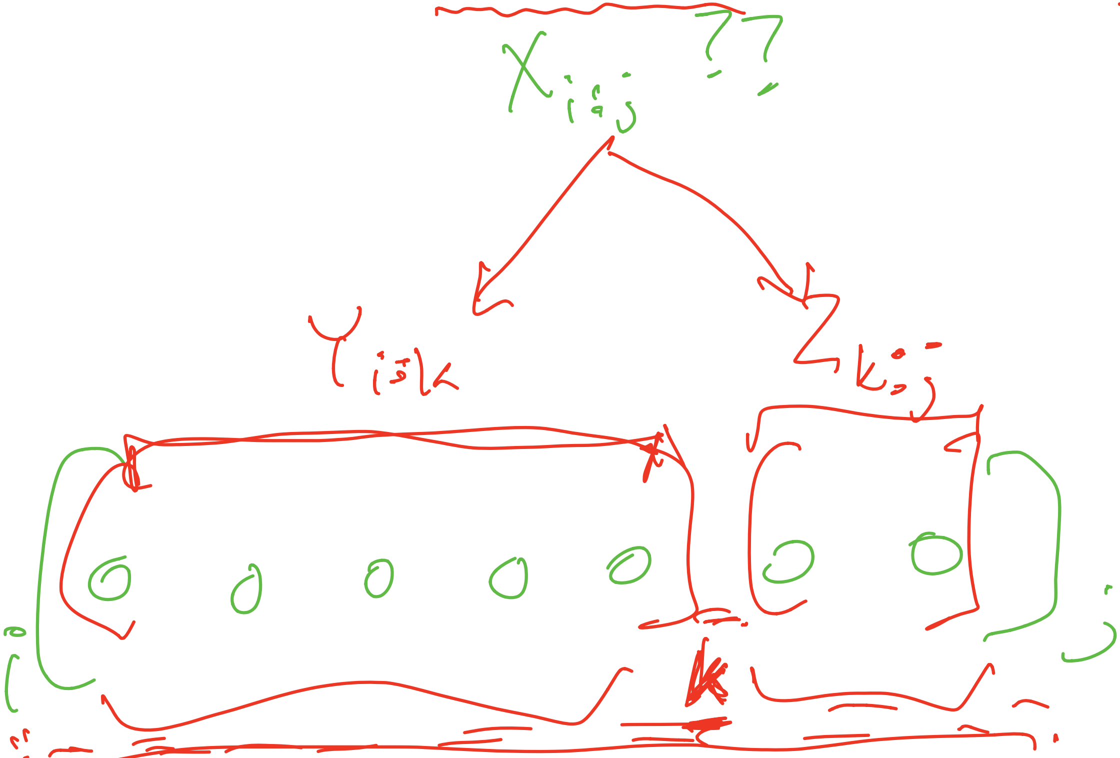
- Cocke-Kasami-Younger algorithm
  - Bottom-up dynamic programming:
    Find possible nonterminals for short spans of sentence, then possible combinations for higher spans
  - Requires converting CFG to Chomsky Normal Form (a.k.a. binarization): always one or two RHS terms

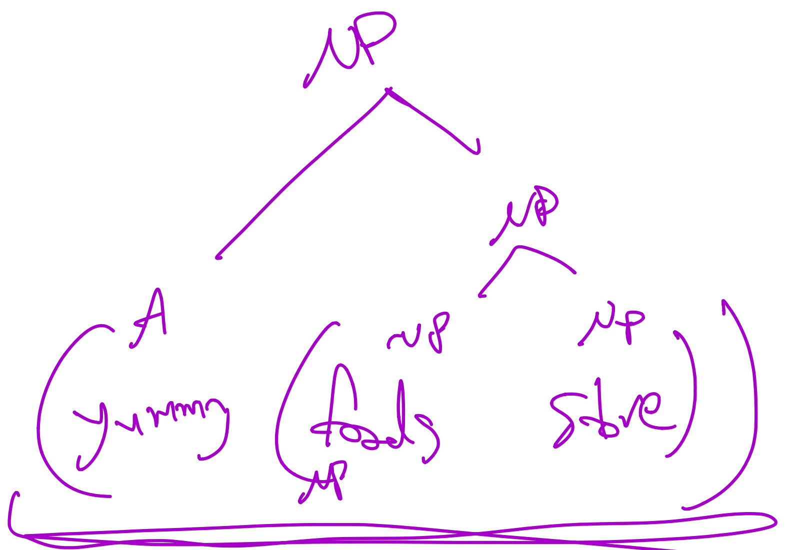
$A \to B\ C\ D$

CKY big idea

- Each nonterminal X is associated with its ***span*** (i:j)
- To merge Y, Z into X via rule X -> Y Z:
  - we can create $X_{i:j}$ only from neighboring children at $Y_{i:k}$ and $Z_{k:j}$
  - *and* we don't care about Y's or Z's internal substructure (Markov property!) thus, we get a dynamic programming speedup!






NP
NP
A
NP
NP
yummy
foods
store
NP

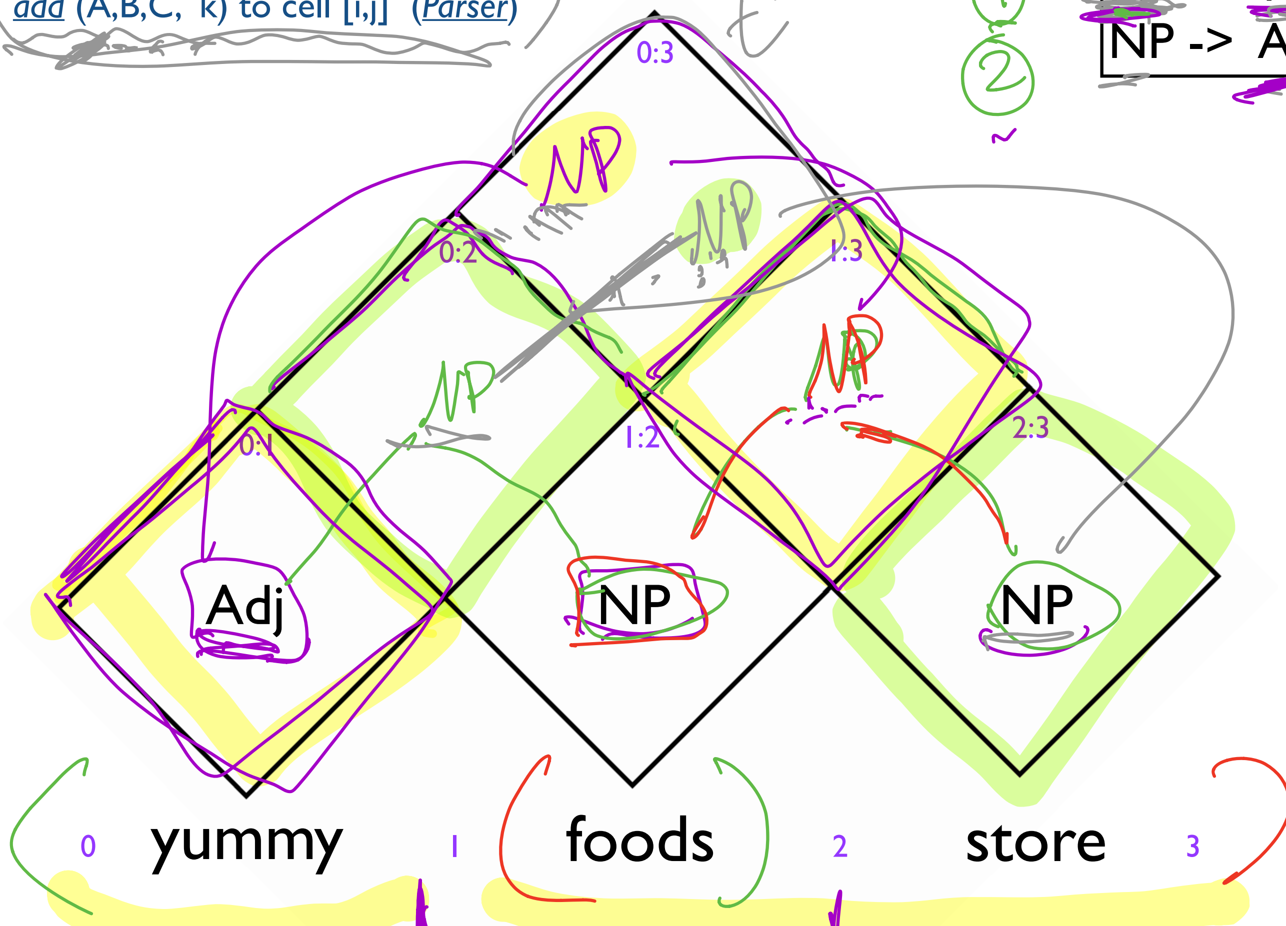

NP
NP
A
NP
NP
yummy
foods
store

CKY

For cell [i:j] (loop through them bottom-up)
    For possible splitpoint k=i..j:
        For every B in [i:k] and C in [k:j],
            If exists rule A -> B C,
                add A to cell [i,j]   (*Recognizer*)
or...   *add* (A,B,C, k) to cell [i,j]   (*Parser*)

Grammar
Adj -> yummy
NP -> foods
NP -> store
NP -> NP NP
NP -> Adj NP

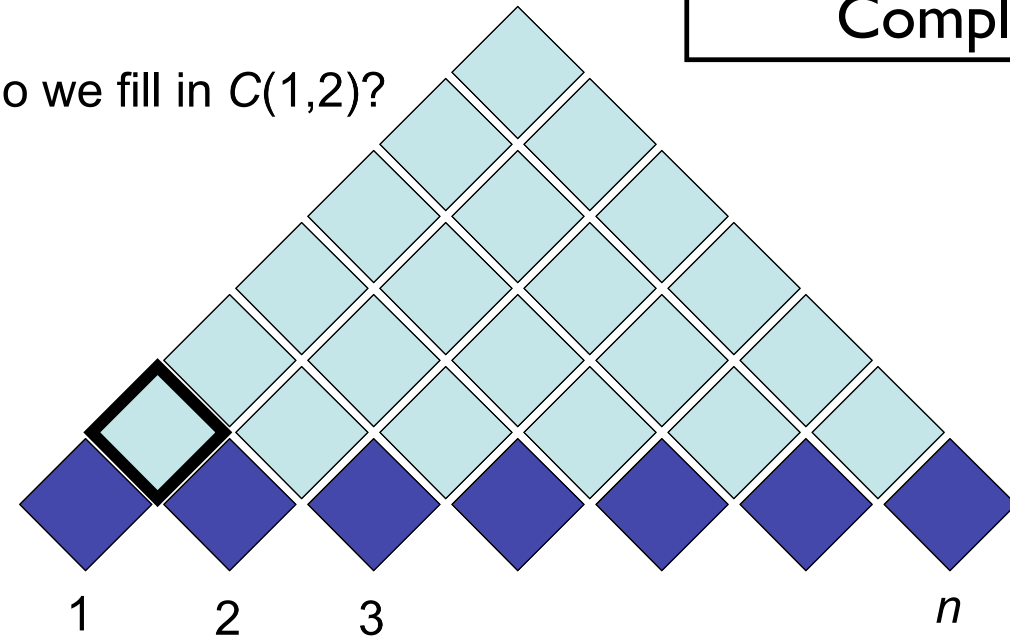
1
2



```
For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add  A to cell [i,j]
```

Computational Complexity ?

How do we fill in $C(1,2)$?



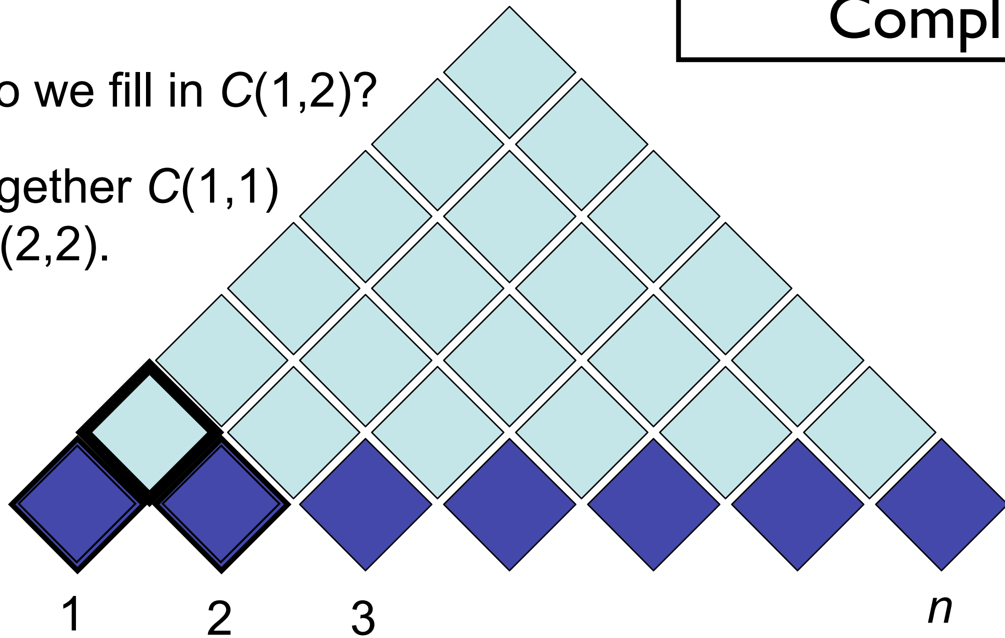
[Example from Noah Smith]

For cell [i,j]
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        add A to cell [i,j]

Computational Complexity ?

How do we fill in *C*(1,2)?

Put together *C*(1,1) and *C*(2,2).

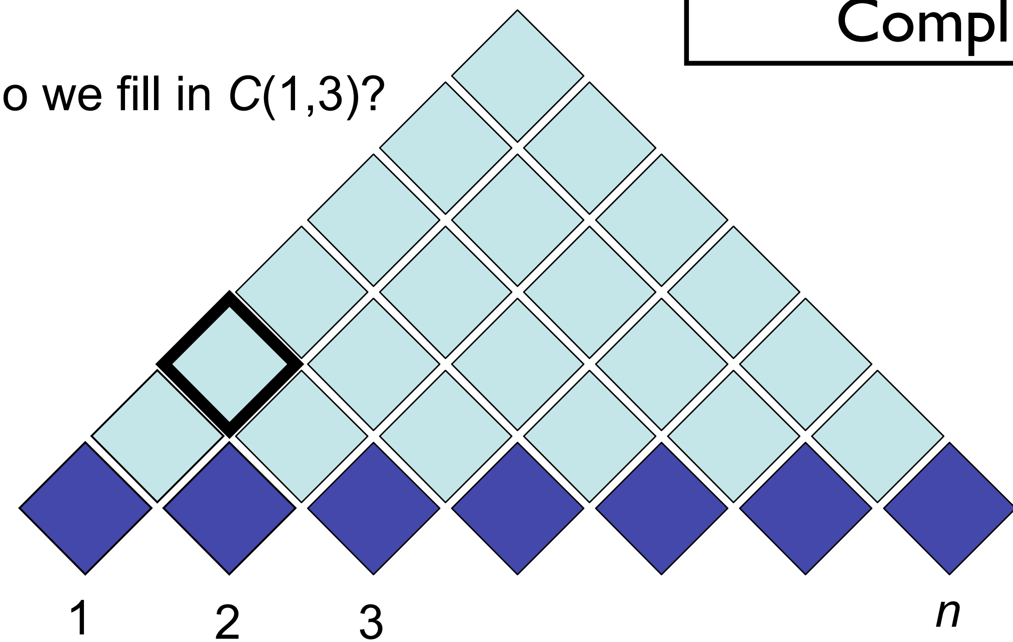


[Example from Noah Smith]

For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]

Computational Complexity ?

How do we fill in *C*(1,3)?



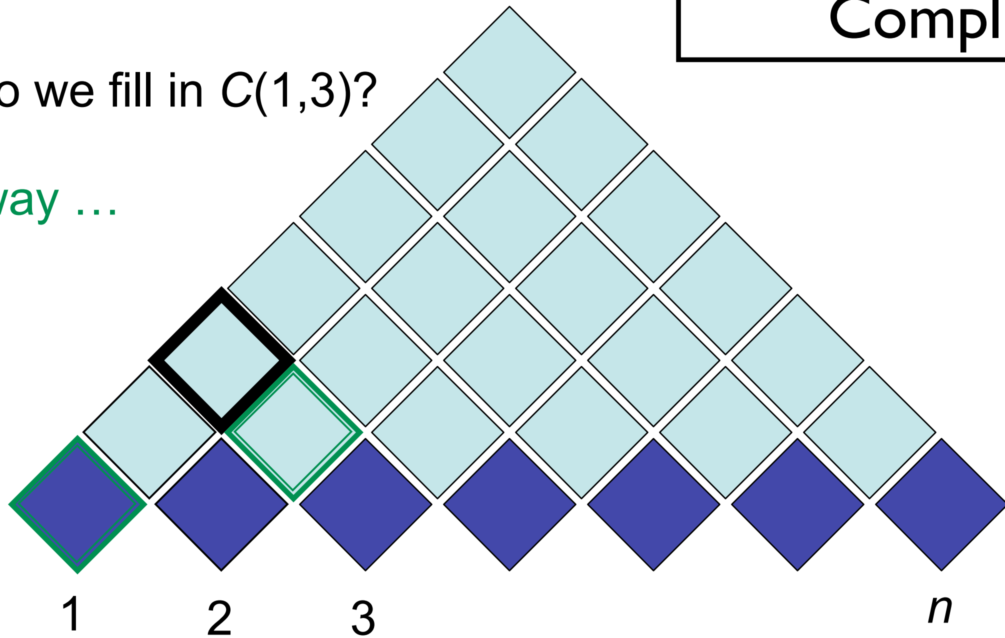
[Example from Noah Smith]

For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]

Computational Complexity ?

How do we fill in $C(1,3)$?

One way …



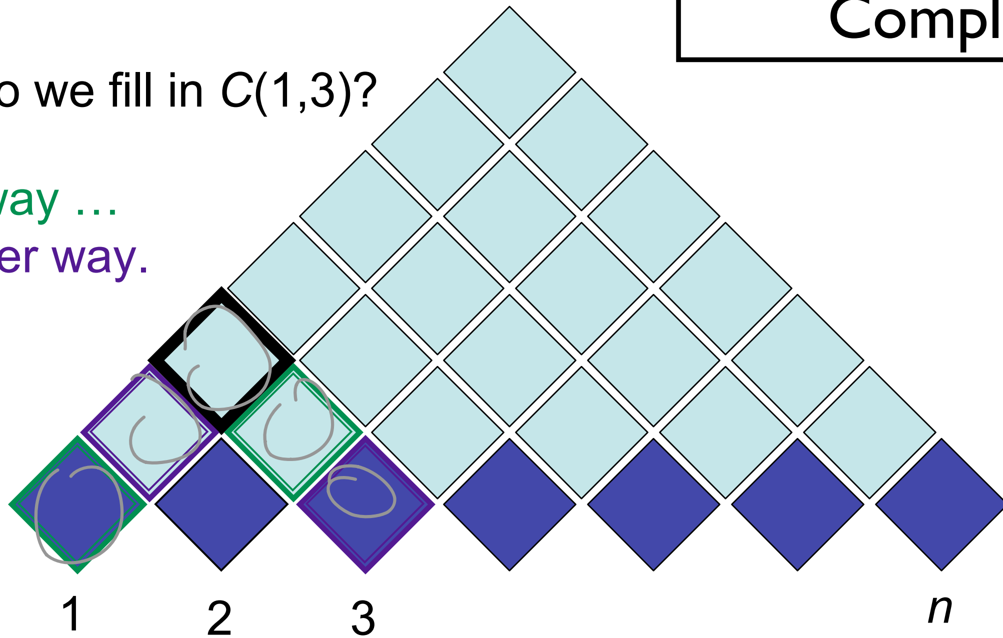
[Example from Noah Smith]

For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]

Computational Complexity ?

How do we fill in *C*(1,3)?

One way …
Another way.

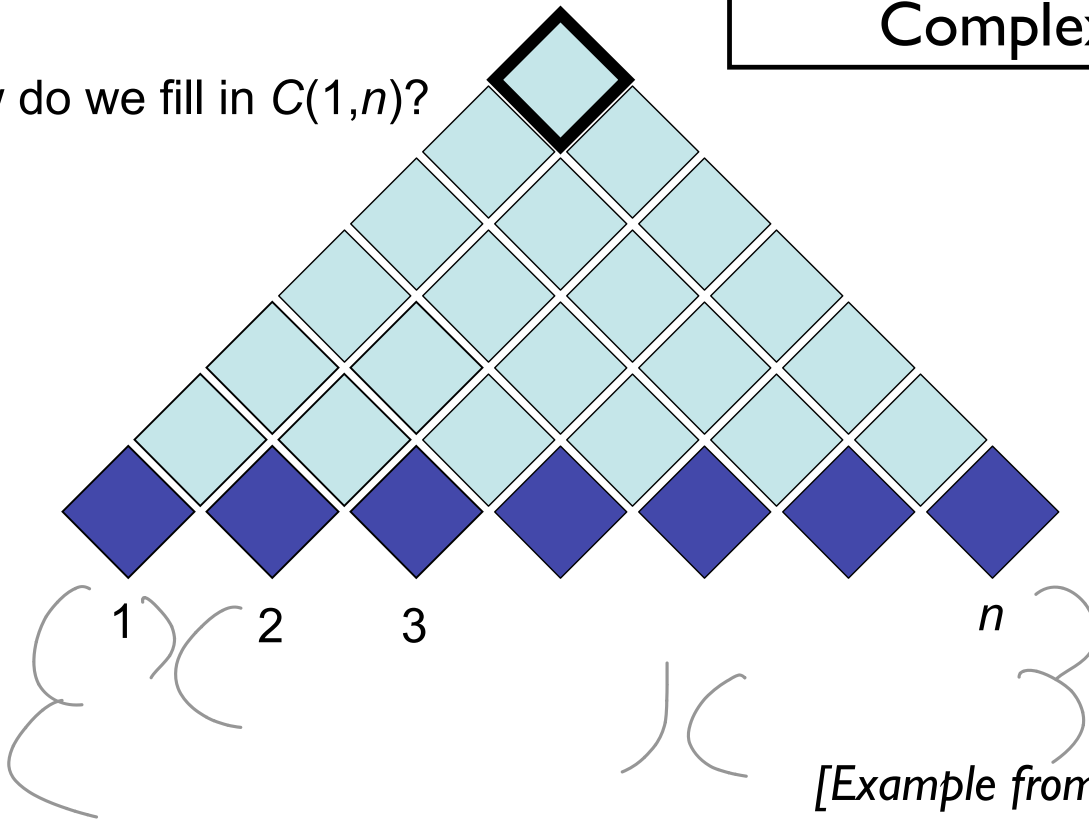


[Example from Noah Smith]

```
For cell [i,j]
    For possible splitpoint k=(i+1)..(j-1):
        For every B in [i,k] and C in [k,j],
            If exists rule A -> B C,
                add A to cell [i,j]
```

Computational Complexity ?

How do we fill in $C(1,n)$?



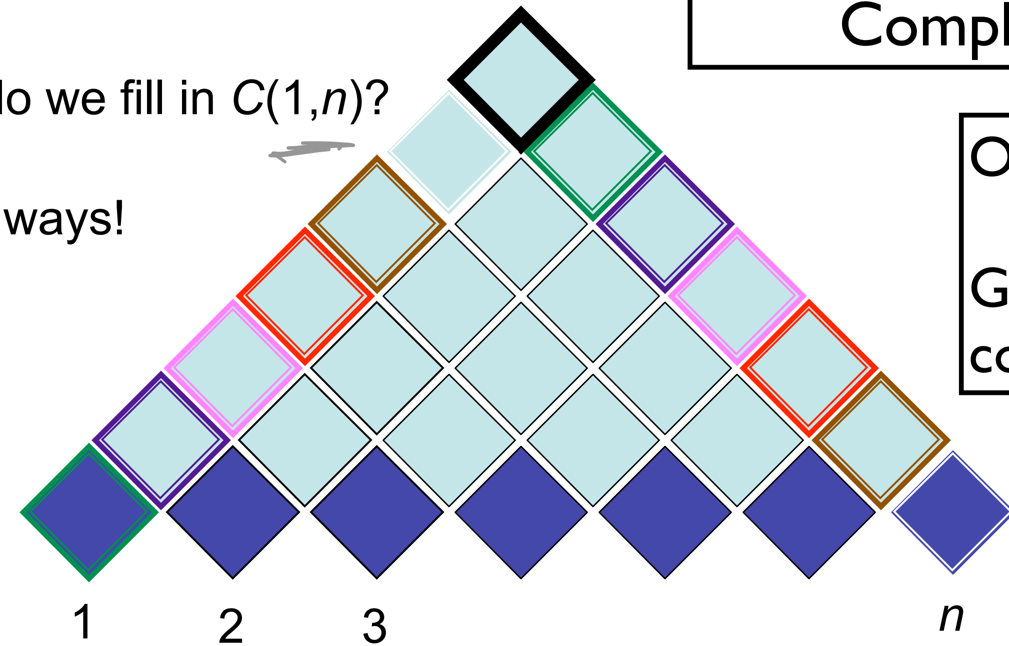
[Example from Noah Smith]

For cell [i,j]
  For possible splitpoint k=(i+1)..(j-1):
    For every B in [i,k] and C in [k,j],
      If exists rule A -> B C,
        <u>*add*</u> A to cell [i,j]

Computational Complexity ?

How do we fill in $C(1,n)$?

$n - 1$ ways!



$O(G\ n^3)$

G = grammar constant

[Example from Noah Smith]

Probabilistic CFGs

Grammar		Lexicon
$S \rightarrow NP\ VP$	[.80]	$Det \rightarrow$ *that* [.10] | *a* [.30] | *the* [.60]
$S \rightarrow Aux\ NP\ VP$	[.15]	$Noun \rightarrow$ *book* [.10] | *flight* [.30]
$S \rightarrow VP$	[.05]	| *meal* [.15] | *money* [.05]
$NP \rightarrow Pronoun$	[.35]	| *flights* [.40] | *dinner* [.10]
$NP \rightarrow Proper\text{-}Noun$	[.30]	$Verb \rightarrow$ *book* [.30] | *include* [.30]
$NP \rightarrow Det\ Nominal$	[.20]	| *prefer*; [.40]
$NP \rightarrow Nominal$	[.15]	$Pronoun \rightarrow$ *I* [.40] | *she* [.05]
$Nominal \rightarrow Noun$	[.75]	| *me* [.15] | *you* [.40]
$Nominal \rightarrow Nominal\ Noun$	[.20]	$Proper\text{-}Noun \rightarrow$ *Houston* [.60]
$Nominal \rightarrow Nominal\ PP$	[.05]	| *TWA* [.40]
$VP \rightarrow Verb$	[.35]	$Aux \rightarrow$ *does* [.60] | *can* [40]
$VP \rightarrow Verb\ NP$	[.20]	$Preposition \rightarrow$ *from* [.30] | *to* [.30]
$VP \rightarrow Verb\ NP\ PP$	[.10]	| *on* [.20] | *near* [.15]
$VP \rightarrow Verb\ PP$	[.15]	| *through* [.05]
$VP \rightarrow Verb\ NP\ NP$	[.05]	
$VP \rightarrow VP\ PP$	[.15]	
$PP \rightarrow Preposition\ NP$	[1.0]	

Emission

- Defines a probabilistic generative p(y,w)
    - Can parameterize *local* production probs. Must maintain Markov property for efficient inference.
- Learning
    - Fully supervised: if you have a treebank (Penn TB, Chinese TB)
    - Unsupervised: EM or other latent-variable learning methods

Penn Treebank

```
( (S
    (NP-SBJ (NNP General) (NNP Electric) (NNP Co.) )
    (VP (VBD said)
      (SBAR (-NONE- 0)
        (S
          (NP-SBJ (PRP it) )
          (VP (VBD signed)
            (NP
              (NP (DT a) (NN contract) )
              (PP (-NONE- *ICH*-3) ))
            (PP (IN with)
              (NP
                (NP (DT the) (NNS developers) )
                (PP (IN of)
                  (NP (DT the) (NNP Ocean) (NNP State) (NNP Power) (NN project) ))))
            (PP-3 (IN for)
              (NP
                (NP (DT the) (JJ second) (NN phase) )
                (PP (IN of)
                  (NP
                    (NP (DT an) (JJ independent)
                      (ADJP
                        (QP ($ $) (CD 400) (CD million) )
                        (-NONE- *U*) )
                      (NN power) (NN plant) )
                    (, ,)
                    (SBAR
                      (WHNP-2 (WDT which) )
                      (S
                        (NP-SBJ-1 (-NONE- *T*-2) )
                        (VP (VBZ is)
                          (VP (VBG being)
                            (VP (VBN built)
                              (NP (-NONE- *-1) )
                              (PP-LOC (IN in)
                                (NP
                                  (NP (NNP Burrillville) )
                                  (, ,)
                                  (NP (NNP R.I) ))))))))))))))))
```

(P)CFG model, (P)CKY algorithm

- CKY: given CFG and sentence w
  - Does there exist at least one parse?
  - Enumerate parses (backpointers)

- Probabilistic/Weighted CKY: given PCFG and sentence w
  - Likelihood of sentence P(w)
  - Most probable parse ("Viterbi parse")
    $\mathrm{argmax}_y\ P(y \mid w) = \mathrm{argmax}_y\ P(y, w)$
  - Non-terminal span marginals (Inside-outside algorithm)

- Discriminative (Tree-CRF) parsing:
  $\mathrm{argmax}_y\ P(y \mid w)$

- Parsing model accuracy: lots of ambiguity!!
  - PCFGs lack lexical information to resolve ambiguities (sneak in world knowledge?)
  - Need to add word embeddings or other lexical information to enrich phrase representations
- Parsers' computational efficiency
  - Grammar constant; pruning & heuristic search
  - $O(N^3)$ for CKY (ok? sometimes...)
  - $O(N)$ left-to-right incremental algorithms
- Evaluate: precision and recall of labeled spans
- Treebank data

Better PCFG grammars

- Lexicalization: encode semantic preferences

Non-terminal	Direction	Priority
S	right	VP SBAR ADJP UCP NP
VP	left	VBD VBN MD VBZ TO VB VP VBG VBP ADJP NP
NP	right	N* EX $ CD QP PRP . . .
PP	left	IN TO FW

Table 11.3: A fragment of head percolation rules

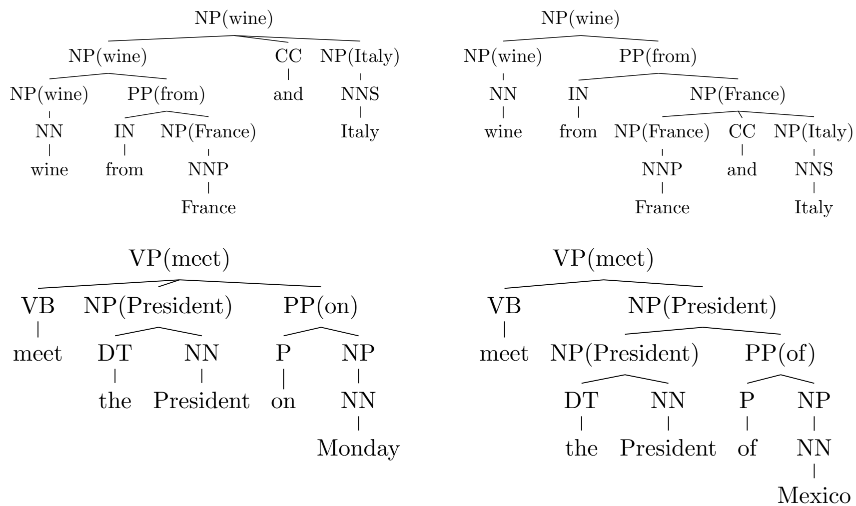


Figure 11.9: Lexicalization can address ambiguity on coordination scope (upper) and PP attachment (lower)

[From Eisenstein (2017)]

Reranking

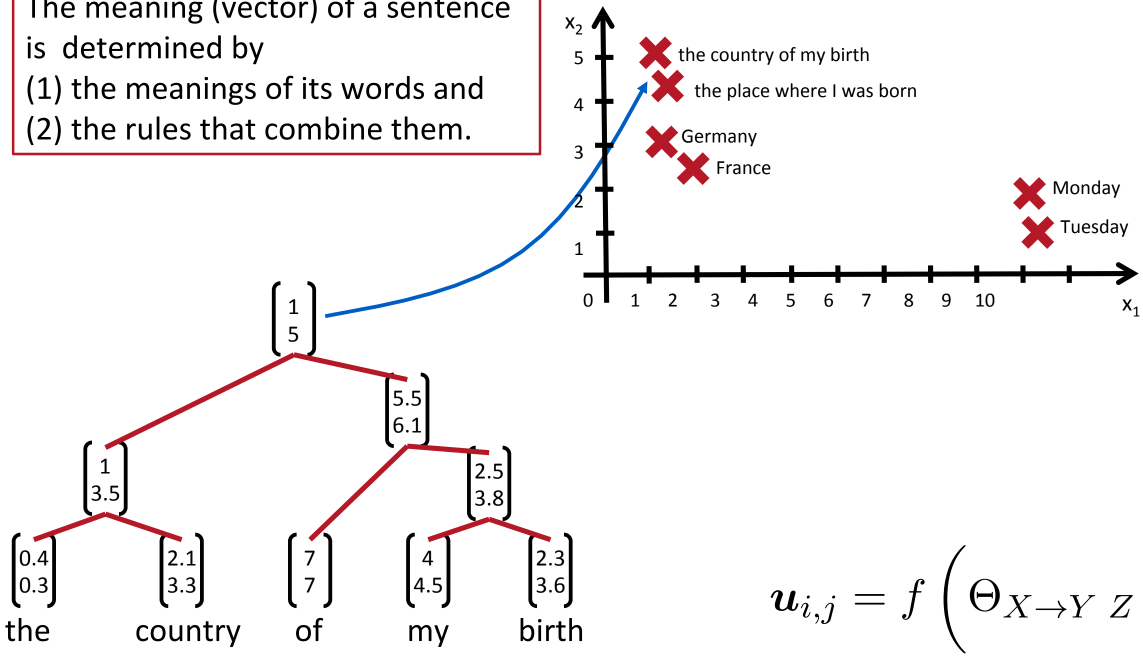
- CFGs are fast, but only use local info
- Whole-structure scoring (features, tree RNNs, etc.) is slow, but can use global info
- Solution: **Reranking**
  - CKY/Viterbi to infer top-K parses from fast CFG model
  - Score each one with NN/features for K-way multiclass problem
    - or use a ranking loss, etc.
- Reranking (fast->slow) is a very general approach in NLP & other areas (IR, etc.)

Reranking: TreeRNN

[Socher et al. (2013)]

Use principle of compositionality

The meaning (vector) of a sentence is determined by
(1) the meanings of its words and
(2) the rules that combine them.



$$\boldsymbol{u}_{i,j} = f\left(\Theta_{X \to Y\ Z} \begin{bmatrix} \boldsymbol{u}_{i,k} \\ \boldsymbol{u}_{k,j} \end{bmatrix}\right)$$

(Can also be used for classification or other tasks, not just parsing itself)

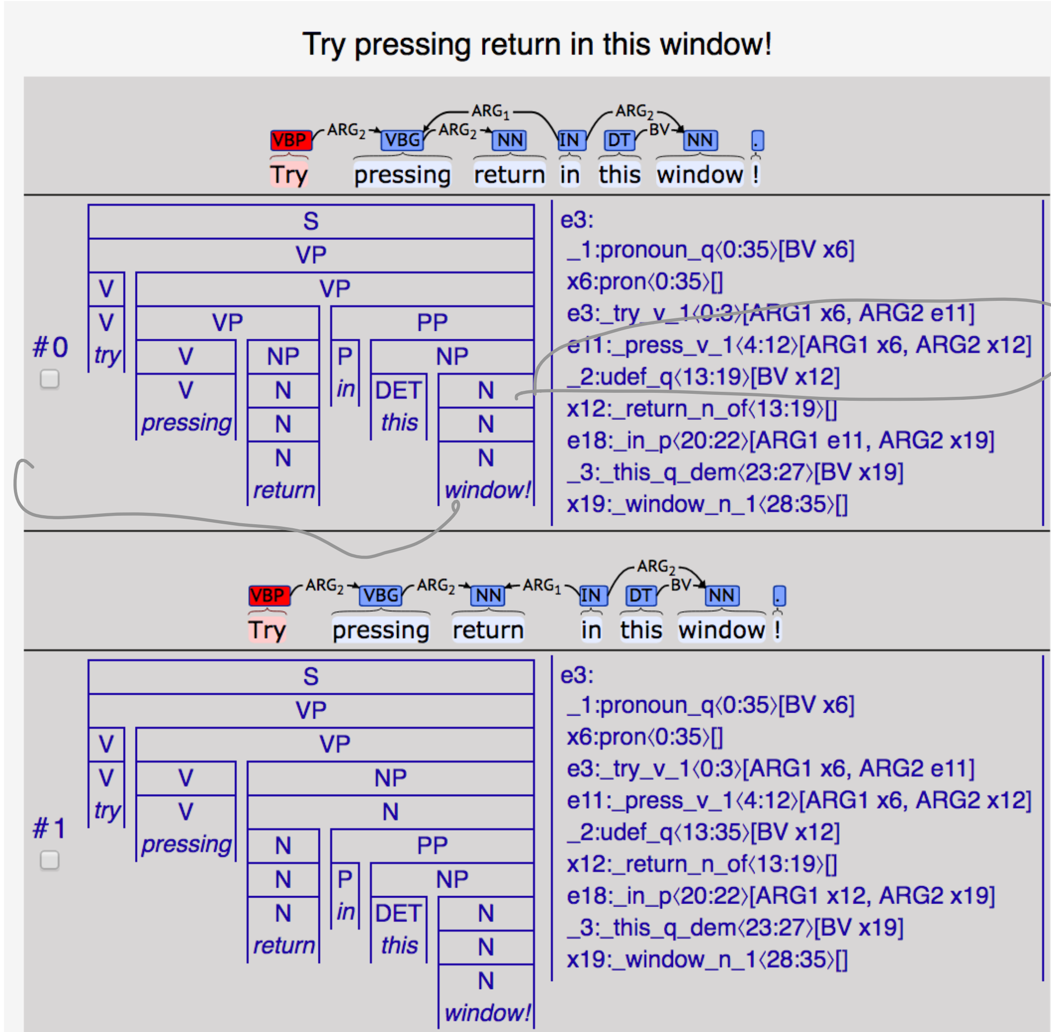
Model performance

Vanilla PCFG	72%
Parent-annotations (Johnson, 1998)	80%
Lexicalized (Charniak, 1997)	86%
Lexicalized (Collins, 2003)	87%
Lexicalized, reranking, self-training (McClosky et al., 2006)	92.1%
State splitting (Petrov and Klein, 2007)	90.1%
CRF Parsing (Finkel et al., 2008)	89%
TAG Perceptron Parsing (Carreras et al., 2008)	91.1%
Compositional Vector Grammars (Socher et al., 2013a)	90.4%
Neural CRF (Durrett and Klein, 2015)	91.1%

Table 11.7: Penn Treebank parsing scoreboard, circa 2015 (Durrett and Klein, 2015)

[From Eisenstein (2017)]

ERG


Try pressing return in this window!
VBP ARG2 VBG ARG1 ARG2 NN IN DT ARG2 BV NN .
Try pressing return in this window !
#0
S
VP
V VP
V VP PP
try V NP P NP
V N in DET N
pressing N this N
N N
return window!
e3:
_1:pronoun_q⟨0:35⟩[BV x6]
x6:pron⟨0:35⟩[]
e3:_try_v_1⟨0:3⟩[ARG1 x6, ARG2 e11]
e11:_press_v_1⟨4:12⟩[ARG1 x6, ARG2 x12]
_2:udef_q⟨13:19⟩[BV x12]
x12:_return_n_of⟨13:19⟩[]
e18:_in_p⟨20:22⟩[ARG1 e11, ARG2 x19]
_3:_this_q_dem⟨23:27⟩[BV x19]
x19:_window_n_1⟨28:35⟩[]
VBP ARG2 VBG ARG2 NN ARG1 IN DT ARG2 BV NN .
Try pressing return in this window !
#1
S
VP
V VP
V V NP
try V N
pressing N PP
N P NP
N in DET N
return this N
N
window!
e3:
_1:pronoun_q⟨0:35⟩[BV x6]
x6:pron⟨0:35⟩[]
e3:_try_v_1⟨0:3⟩[ARG1 x6, ARG2 e11]
e11:_press_v_1⟨4:12⟩[ARG1 x6, ARG2 x12]
_2:udef_q⟨13:35⟩[BV x12]
x12:_return_n_of⟨13:19⟩[]
e18:_in_p⟨20:22⟩[ARG1 x12, ARG2 x19]
_3:_this_q_dem⟨23:27⟩[BV x19]
x19:_window_n_1⟨28:35⟩[]

Left-to-right models

- Can sequence models learn hierarchical syntactic phenomena?

- Case study:
  Subject-Verb agreement on grammatical number

(1) a. The **key is** on the table.
b. *The **key are** on the table.
c. *The **keys is** on the table.
d. The **keys are** on the table.

- N-grams can't capture long-distance dependencies

(2) The **keys** to the <u>cabinet</u> **are** on the table.

(3) The **building** on the far right that's quite old and run down **is** the Kilgore Bank Building.

[Linzen et al. 2016]

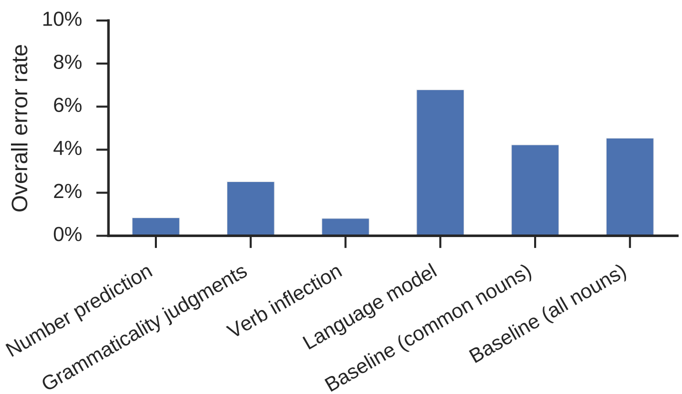
Left-to-right sequence RNN

- Does an LSTM LM implicitly learn these syntactic rules?
  - Assess number prediction by comparing e.g. P(writes | ...) vs. P(write | ...)

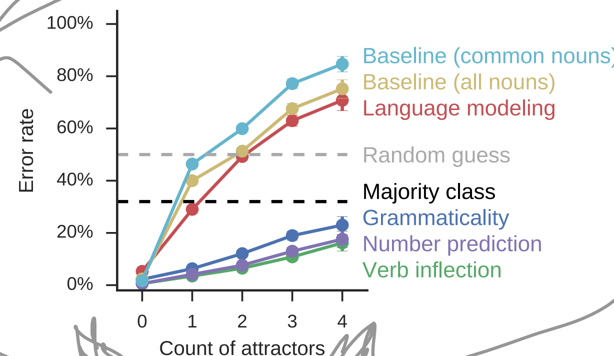
Training objective	Sample input	Training signal	Prediction task	Correct answer
Number prediction	*The keys to the cabinet*	PLURAL	SINGULAR/PLURAL?	PLURAL
Verb inflection	*The keys to the cabinet [is/are]*	PLURAL	SINGULAR/PLURAL?	PLURAL
Grammaticality	*The keys to the cabinet are here.*	GRAMMATICAL	GRAMMATICAL/UNGRAMMATICAL?	GRAMMATICAL
Language model	*The keys to the cabinet*	are	$P(are) > P(is)$?	True

Table 1: Examples of the four training objectives and corresponding prediction tasks.

(a)



(b)



Left-to-right grammatical RNN

- Shift-reduce parsing
  - One form of left-to-right / top-down parsing
  - Incrementally build up the parse tree, scanning words left-to-right.
    - Parser as a state machine
  - No dynamic programming! O(n) runtime (typically)
  - Potentially related to cognitive processing?
  - Most practically efficient for constituent parsing -- e.g. *zpar* and *CoreNLP* implementations
- "RNN Grammars": LSTM-based stack automaton (not merely a traditional sequence RNN)

Shift-reduce parsing

- State machine: <u>*stack*</u> and input <u>*buffer*</u>
- Decide on one of 3 *actions*

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	$n-1$

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*) .)		

<u>[Dyer et al. 2016]</u>

Generation as well

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	$n-1$

Figure 3: Generator transitions. Symbols defined as in Fig. 1 with the addition of T representing the history of generated terminals.

	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* | .	

Figure 4: Joint generation of a parse tree and sentence.

[Dyer et al. 2016]

Shift-reduce parsing

- Models for shift-reduce
  - Any (P)CFG can be parsed in this manner *[Stolcke 1995]*
- History based models: select next action given information about *current state and history*
  - Infinite history, no future (contrast to PCFG assumptions!)
  - ***a***: action
  - ***u***: features/embedding of current state
- Generative form (discriminative also possible):

current state

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

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

[Dyer et al. 2016]

- Vector representation of current stack/buffer state
  - Explicit log-linear features over the current stack, buffer etc. *[Ratnaparkhi 1998, Zhang+Clark 2011]*
  - Neural network representation of current state *[e.g. Henderson 2004, Dyer et al. 2016, Bowman et al. 2016]*
- Training: extract oracle decisions paths from labeled data
  - Generative model: use importance sampling to calculate feature expectations

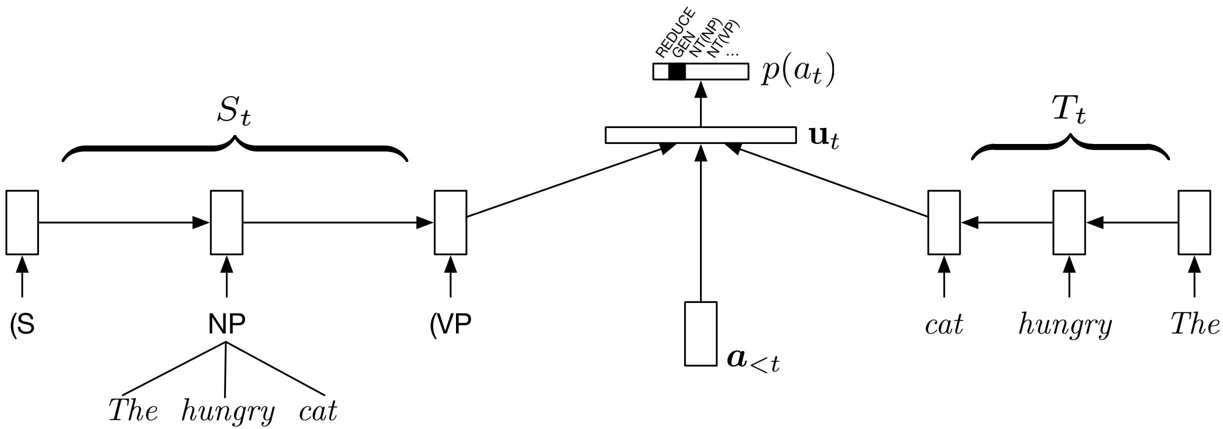


Figure 5: Neural architecture for defining a distribution over a_t given representations of the stack (S_t), output buffer (T_t) and history of actions ($\boldsymbol{a}_{<t}$). Details of the composition architecture of the NP, the action history LSTM, and the other elements of the stack are not shown. This architecture corresponds to the generator state at line 7 of Figure 4.

[Dyer et al. 2016]

Results: Look out for bugs.

Due to an implentation bug in the RNNG's recursive composition function, the results reported in Dyer et al. (2016) did not correspond to the model as it was presented. This corri-

- Even the experts have bugs!
- Many, MANY unreported bugs in results are likely out there
- Replication and reimplementation are often good ways of finding them

Model	**type**	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**

Table 5: Parsing results with fixed composition function on PTB §23 (D=discriminative, G=generative, S=semisupervised). * indicates the (Vinyals et al., 2015) model trained only on the WSJ corpus without ensembling. † indicates RNNG models with the buggy composition function implementation.