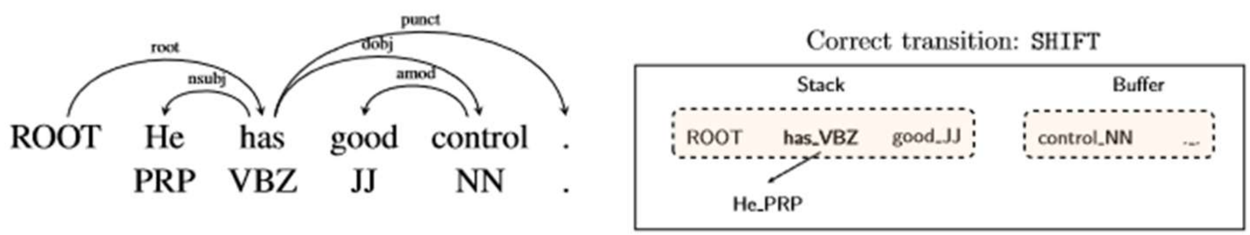
CCG Parsing: Ambati et al., 2016

AUSTIN BLODGETT

Review of Transition-based Parsing



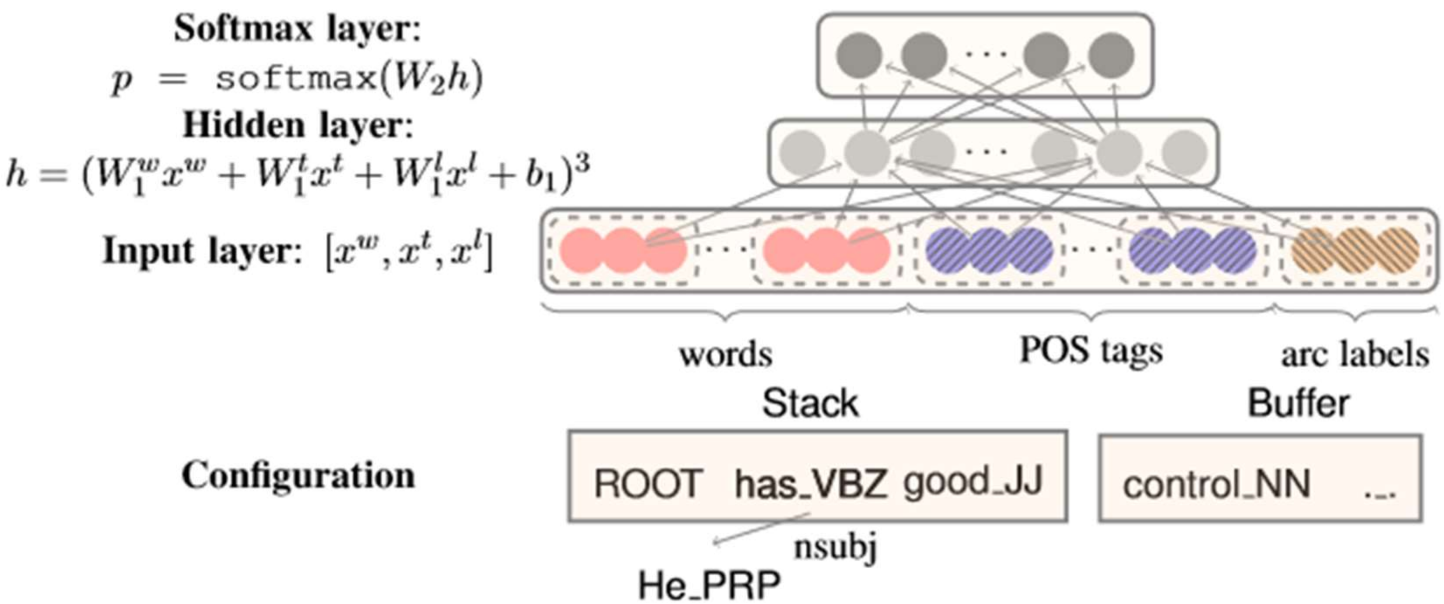
Transition	Stack	Buffer	A
	[ROOT]	[He has good control .]	$\emptyset$
SHIFT	[ROOT He]	[has good control .]	
SHIFT	[ROOT He has]	[good control .]	
LEFT-ARC(nsubj)	[ROOT has]	[good control .]	$A \cup$ nsubj(has,He)
SHIFT	[ROOT has good]	[control .]	
SHIFT	[ROOT has good control]	[.]	
LEFT-ARC(amod)	[ROOT has control]	[.]	$A \cup$amod(control,good)
RIGHT-ARC(dobj)	[ROOT has]	[.]	$A \cup$ dobj(has,control)
...	...	...	...
RIGHT-ARC(root)	[ROOT]	[]	$A \cup$ root(ROOT,has)

Figure 1: An example of transition-based dependency parsing. Above left: a desired dependency tree, above right: an intermediate configuration, bottom: a transition sequence of the arc-standard system.

Review of Transition-based Parsing

- LEFT-ARC(l): adds an arc $s_1 \rightarrow s_2$ with label l and removes s_2 from the stack. Precondition: $|s| \geq 2$.
- RIGHT-ARC(l): adds an arc $s_2 \rightarrow s_1$ with label l and removes s_1 from the stack. Precondition: $|s| \geq 2$.
- SHIFT: moves b_1 from the buffer to the stack. Precondition: $|b| \geq 1$.

Review of Transition-based Parsing



Ambati et al., 2016

How to parse CCG with Shift and Reduce?

New Transition Rules

- REDUCE-LEFT(cat): remove **s1** from the stack and tag constituent as **cat**
- REDUCE-RIGHT(cat): remove **s2** from the stack and tag constituent as **cat**
- REDUCE-UNARY(cat): remove **s1** (or s2?) from the stack and tag constituent as **cat**
- SHIFT: moves **b1** from the buffer to the stack

Ambati et al., 2016

Ambati et al.'s parser uses 2296 total Transitions:

- 340 REDUCE-LEFT(cat)
- 593 REDUCE-RIGHT(cat)
- 78 REDUCE-UNARY(cat)
- 1285 SHIFT

Ambati et al., 2016

Nodes to consider:

- a) top 4 nodes in the **stack**
- b) next 4 nodes in the **input**
- c) **left and right children** of the top 2 nodes in the stack

34 features:

- Word embeddings from (a-c)
- POS embeddings from (a-c)
- CCG tag embeddings from (a, b) + lexical heads of 2 nodes in the stack

Input layer = 34 x 50 (embedding size)

Ambati et al., 2016

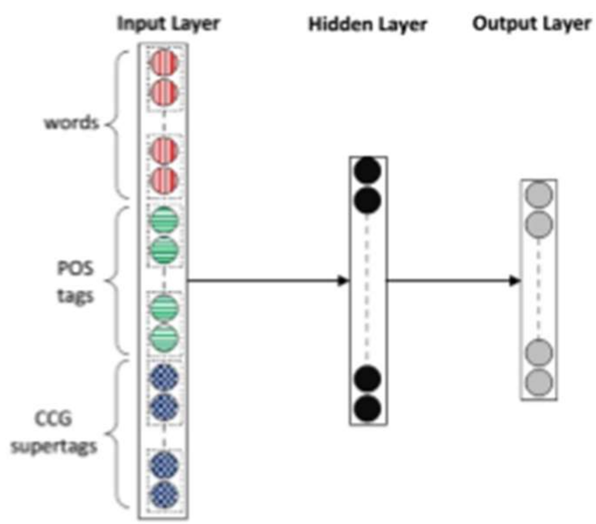


Figure 1: Our Neural Network Architecture (adapted from Chen and Manning (2014)).

Ambati et al., 2016

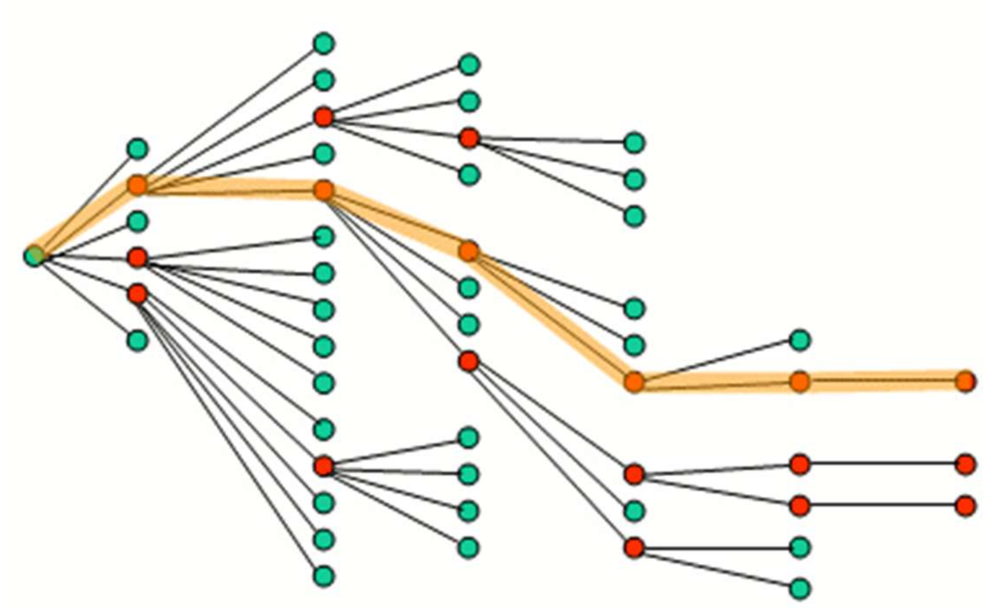
Greedy Search

Model	*Tagger*	*UF*	*LF*	*Cat.*
Z&C*	C&C	87.24	80.25	91.09
Our NNPar	C&C	89.38	82.65	91.72
Z&C*	NNT	87.00	79.78	90.52
Our NNPar	NNT	**90.09**	**83.33**	**92.03**

Table 1: Performance of greedy CCG parsers on CCGbank development data (Sec. 00).

Ambati et al., 2016

Beam Search



Ambati et al., 2016

Beam Search

Model	*Beam*	*UF*	*LF*	*Cat.*
Z&C*	1	87.28	80.78	91.44
Our NNPar	1	89.78	83.27	91.89
Z&C*	16	91.28	85.00	92.79
Our NNPar	16	91.14	84.44	92.22
Our Structured NNPar	16	**91.95**	85.57	**92.86**
Zhang and Clark (2011)	16	-	85.48	92.77
Xu et al. (2014)	128	-	**86.00**	92.75

Table 3: Results on CCGbank test data (Sec. 23).

Evaluation

1. Supertag Prediction (F1)
2. Unlabelled F1 (per constituent)
3. ✔ Labelled F1 (per constituent)
4. ✕ Exact Match

Other Approaches

1. LSTM CCG Parsing (Lewis et al. 2016)
2. A* CCG Parsing with a Supertag-factored Model (Lewis and Steedman, 2014)

CCG Leaderboard

C&C + RNN (Xu et al., 2015)

EasyCCG (Lewis and Steedman, 2014)

Leader (Lewis et al. 2016)

Model	P	R	F1
C&C	86.2	84.2	85.2
C&C + RNN	87.7	86.4	87.0
EASYCCG	83.7	83.0	83.3
Dependencies	86.5	85.8	86.1
LSTM	87.7	86.7	87.2
LSTM + Dependencies	88.2	87.3	87.8
LSTM + Tri-training	**88.6**	**87.5**	**88.1**
LSTM + Tri-training + Dependencies	88.2	87.3	87.8

Table 2: Labelled F1 for CCGbank dependencies on the CCGbank test set (Section 23).

CCG Leaderboard: Speed

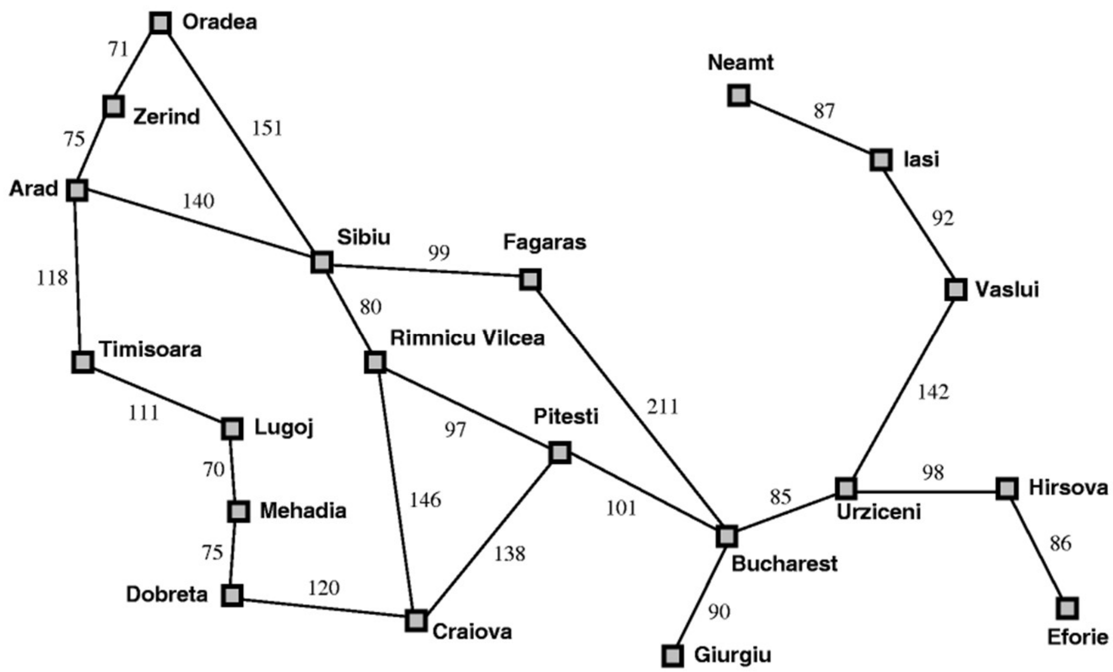
EasyCCG (Lewis and Steedman, 2014)

LSTM GPU (Lewis et al. 2016)

Parser	**Sentences per second**
SpaCy*[4]	778
Berkeley GPU* (Hall et al., 2014)	687
Chen and Manning (2014)*	391
C&C	66
EasyCCG	606
LSTM	214
LSTM + Dependencies	58
LSTM GPU	2670

Table 4: Sentences parsed per second on our hardware. Parsers marked * use non-CCG formalisms but are the fastest available CPU and GPU parsers.

A* search (Lewis and Steedman, 2014)



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

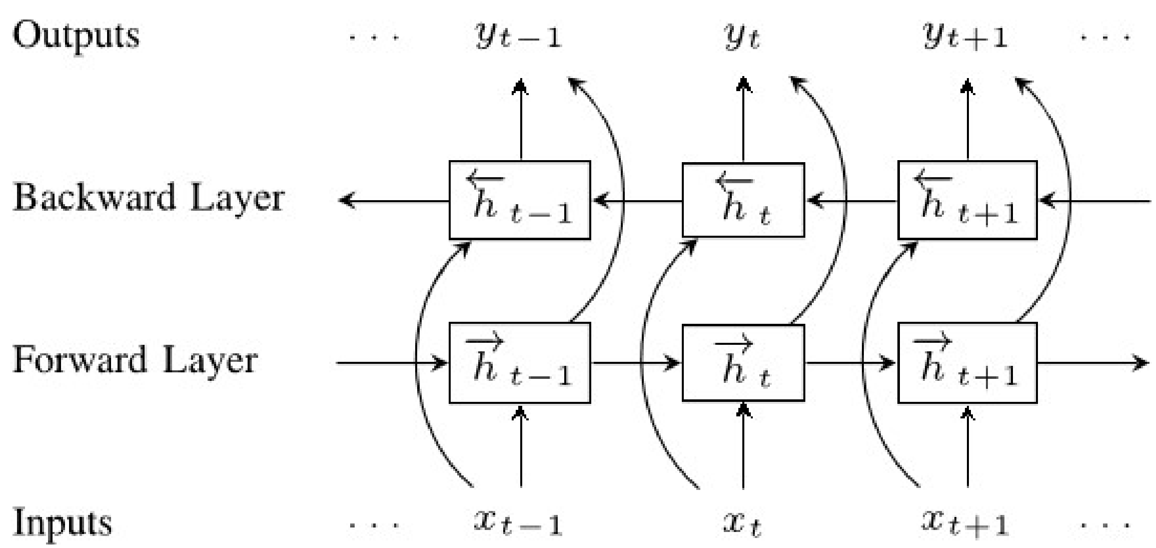
A* Parsing (Lewis and Steedman, 2014)

Choose next action **x** by minimizing:

- f(x) = dist(current, x) + dist(x, endpoint)

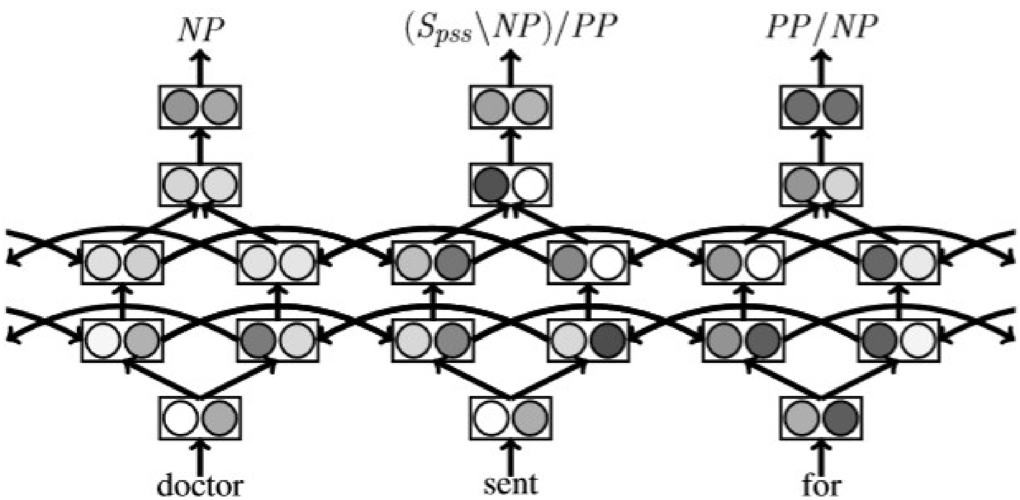
LSTM (Lewis et al. 2016)

Stacked BiLSTM Supertagger



LSTM (Lewis et al. 2016)

Contribution: What we need is a strict deterministic grammar and a great lexical tagger



Reflections

1. Computer Scientists like CCG for its syntax-semantics interface
2. Much of effort spent on efficient search
3. CCG Parsers tend to learn ad hoc combinators
4. Graph-based approaches better, Transition-based approaches faster…
5. Best Accuracy comes from smaller grammar