Stack-propagation: Improved Representation Learning for Syntax

Yuan Zhang CSAIL, MIT David Weiss Google Inc

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 1557–1566,
Berlin, Germany, August 7-12, 2016.

Zhang Shiyue

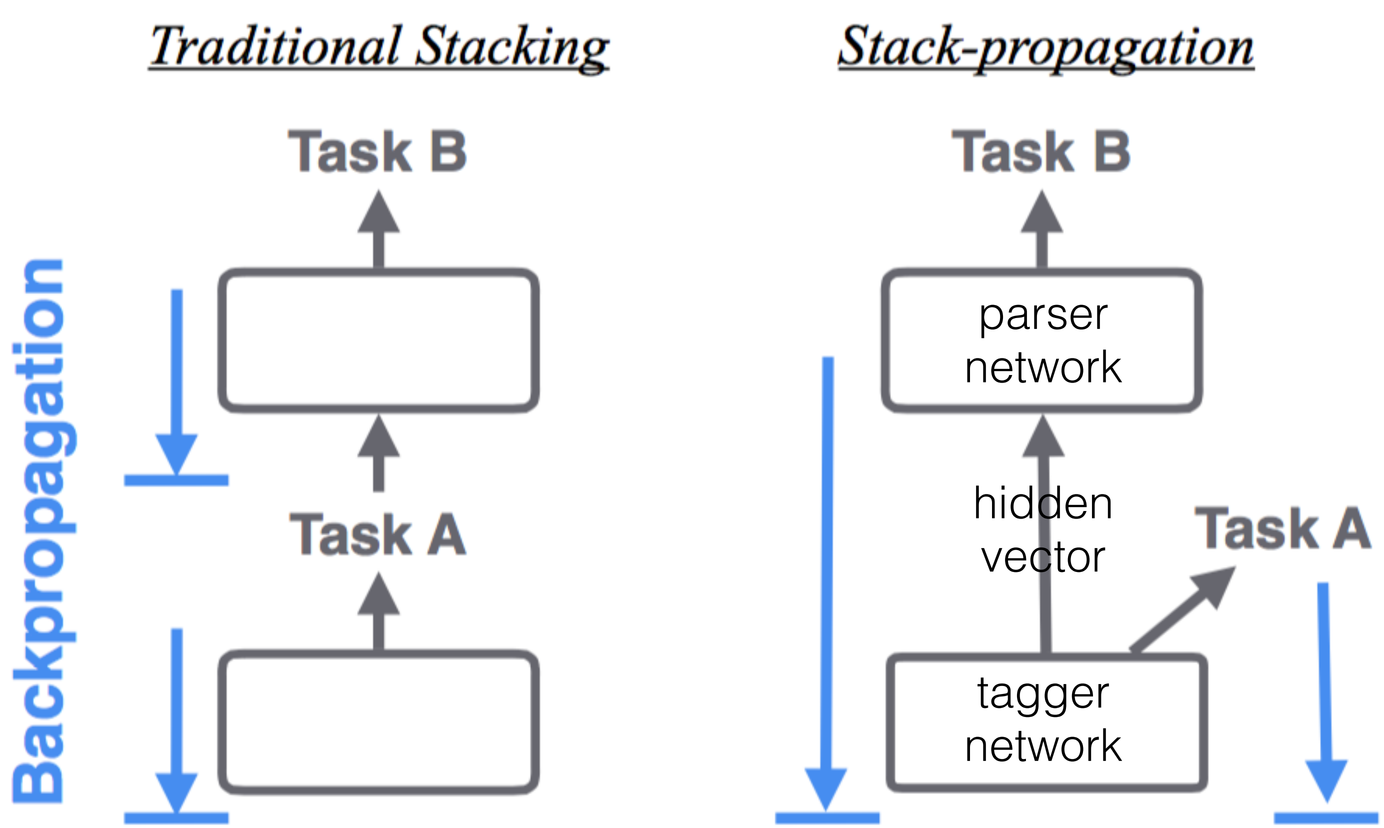
2016.9.30

Background

- Syntax Parser
- Pipeline method: first find POS, then use it to train parser
  - disadvantages:
    - The error of POS tagger will cascade to parser
    - POS tagger cannot take into account the syntactic context
  - two ways to solve this issue:
    - avoid using POS during parsing, but poor performance
    - jointly model both POS and parse trees, but sacrifice either efficiency or accuracy

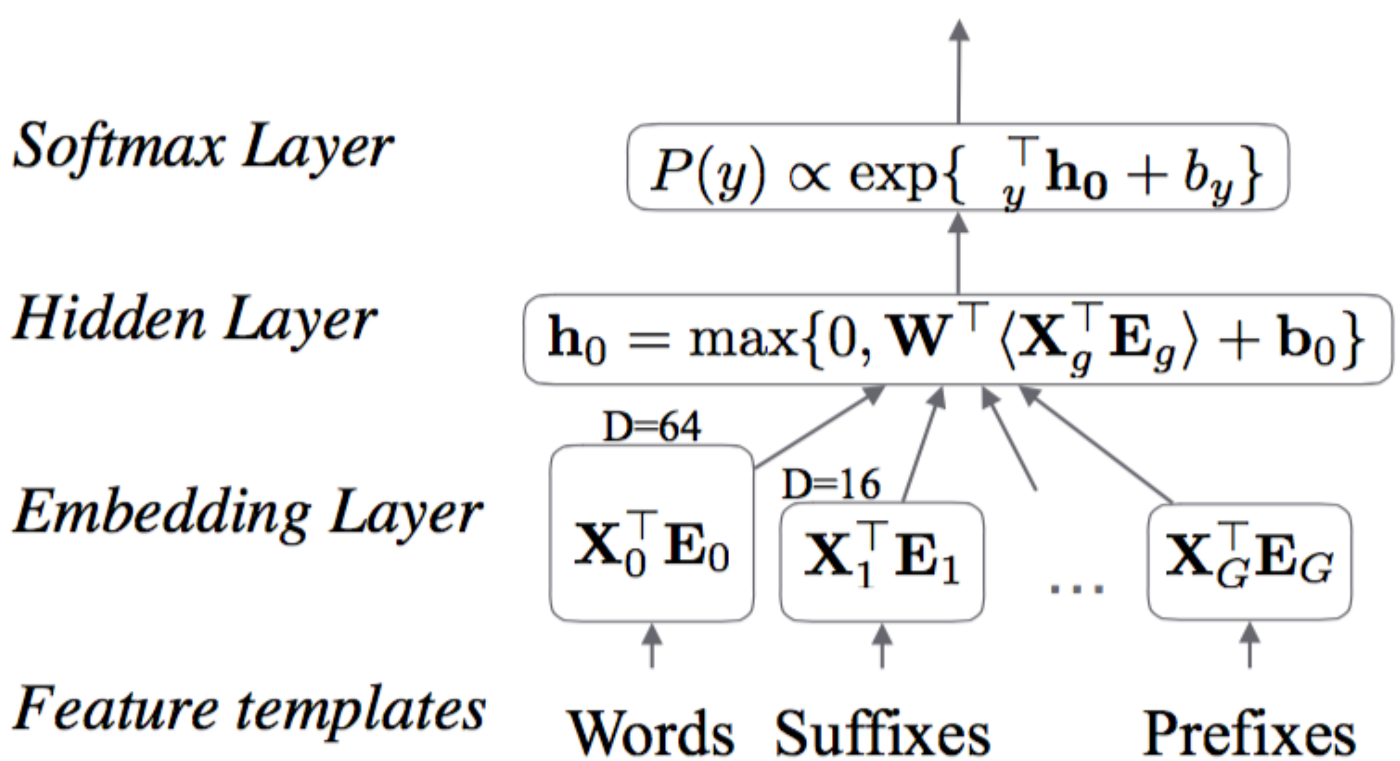
Main Idea

- So, they propose a **“stack-propagation”** model, in which the POS tags are used as **regularisation** instead of features.



Details

- Basic unit



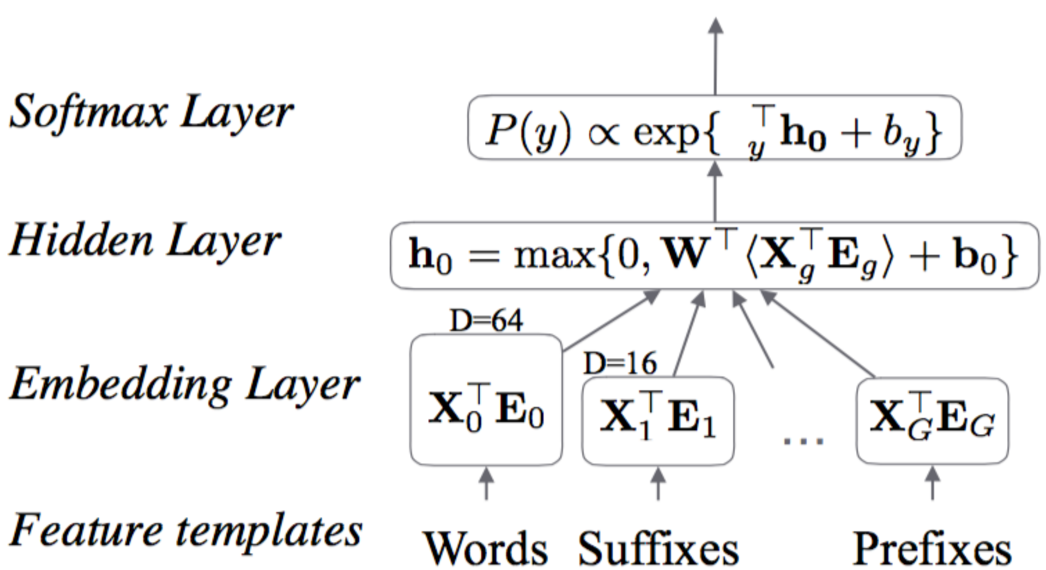
ReLU

concatenated

for parser, features are discrete labels
and continuous hidden vectors

Details

- A window-based tagger network

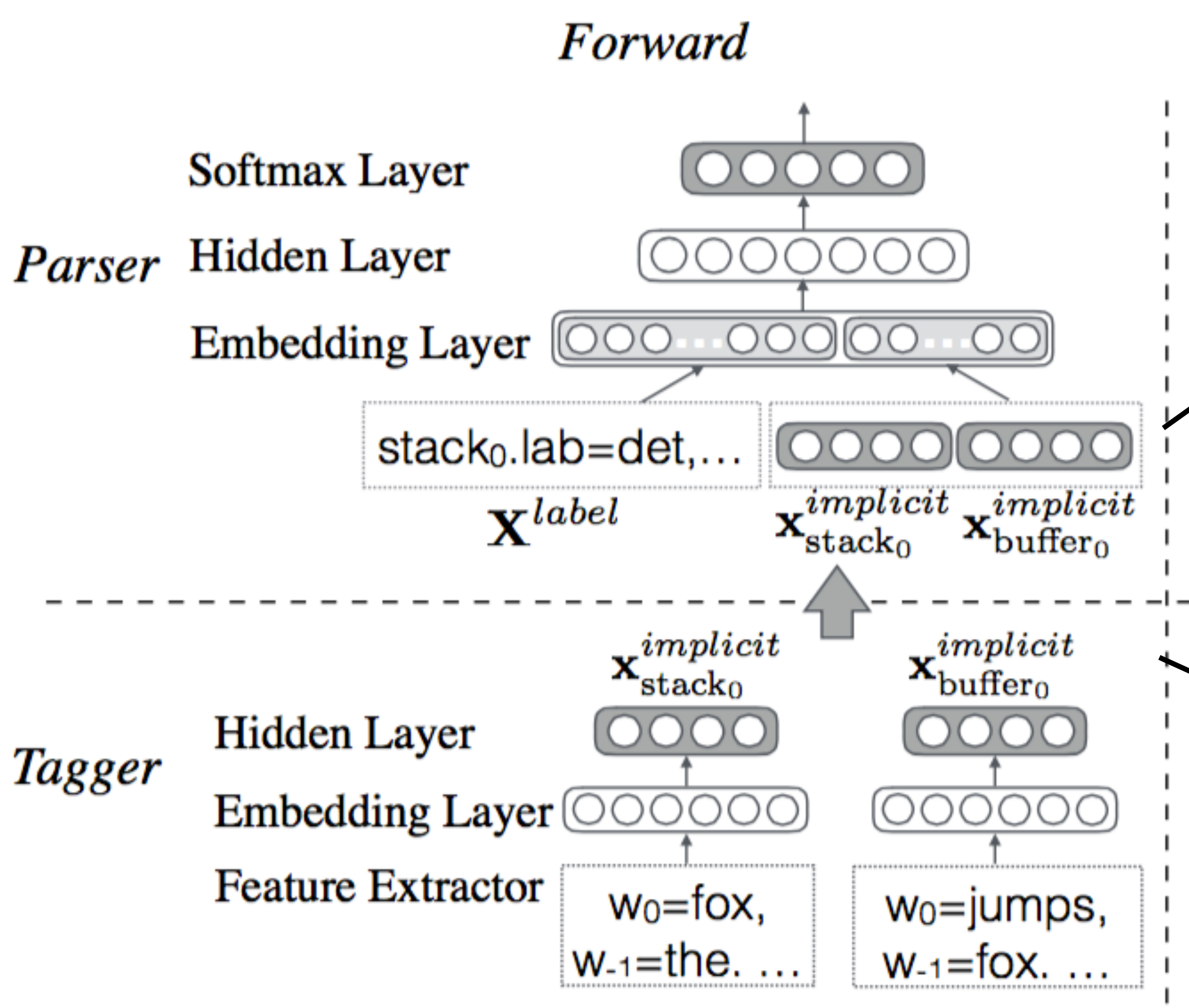


Features (g)	Window	D
Symbols	1	8
Capitalization	+/- 1	4
Prefixes/Suffixes ($n = 2, 3$)	+/- 1	16
Words	+/-3	64

$$\mathbf{h}_0 = [\mathbf{X}^g \mathbf{E}^g \mid \forall g]$$

Details

- A transition-based parser network

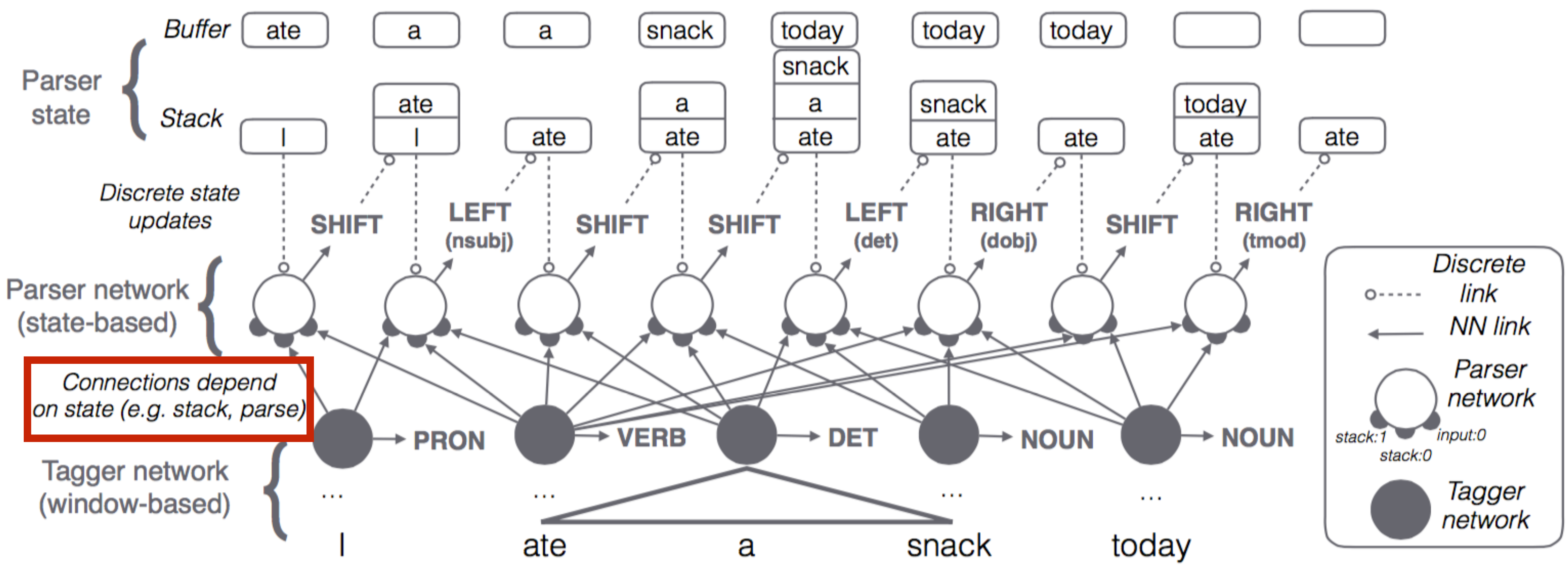


Previously, these are all discrete features(parser configuration): words, labels(from previous decisions), POS tags and morphological attributes.
But now, only labels are retained, and hidden vectors from tagger networks are added

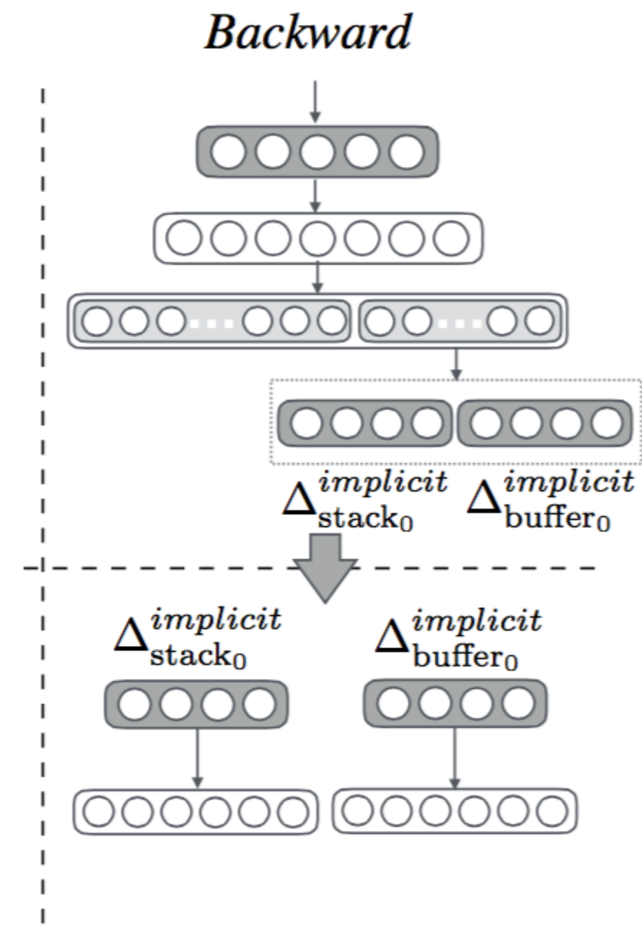
which hidden vectors to add?

Details

- Dynamic many-to-many connection



Details



- Learning with Stack-propagation
- two issues to address:
  - how to handle the dynamic many-to-many connections
  - how to incorporate the POS tags
- First one is easy to tackle: unroll the gold trees into a derivation of (state, action) pairs that produce the tree; the connection of the feed forward network are constructed incrementally as the parser state is updated.

Details

- Second issue: to incorporate the POS tag as a regularisation

$$\max_{\Theta} \lambda \sum_{\mathbf{x},y \in \mathcal{T}} \log(P_{\Theta}(y \mid \mathbf{x})) +$$

{x, y} are POS tagging examples

$$\sum_{c,a \in \mathcal{P}} \log\left(P_{\Theta}(a \mid c)\right)$$

{c, a} are parser pairs (configuration, action)

- Optimise this objective stochastically by alternating between two updates:
  - TAGGER: pick a POS tagging example and update the tagger network with BP
  - PARSER: Given a parser configuration c, BP the parsing loss through the stacked architecture to update both parser and tagger.
- 10 epochs PARSER and 5 epochs TAGGER, and pre-train TAGGER one epoch

Performance

- Universal Dependencies Treebanks

Method	ar	bg	da	de	en	es	eu	fa	fi	fr	hi	id	it	iw	nl	no	pl	pt	sl	AVG
NO TAGS																				
B'15 LSTM	75.6	**83.1**	69.6	**72.4**	77.9	78.5	67.5	74.7	**73.2**	77.4	85.9	**72.3**	**84.1**	73.1	69.5	82.4	**78.0**	**79.9**	**80.1**	76.6
Ours (window)	**76.1**	82.9	**70.9**	71.7	**79.2**	**79.3**	**69.1**	**77.5**	72.5	**78.2**	**87.1**	71.8	83.6	**76.2**	**72.3**	**83.2**	77.8	79.0	79.8	**77.3**
UNIVERSAL TAGSET																				
B'15 LSTM	74.6	82.4	68.1	73.0	77.9	77.8	66.0	75.0	73.6	78.0	86.8	72.2	84.2	74.5	68.4	83.3	74.5	80.4	78.1	76.2
Pipeline P_{tag}	73.7	83.6	72.0	73.0	79.3	79.5	63.0	78.0	66.9	78.5	87.8	73.5	84.2	75.4	70.3	83.6	73.4	79.5	79.4	76.6
RBGParser	75.8	83.6	**73.9**	73.5	79.9	79.6	68.0	**78.5**	65.4	78.9	87.7	**74.2**	84.7	**77.6**	72.4	83.9	75.4	**81.3**	80.7	77.6
Stackprop	**77.0**	**84.3**	73.8	**74.2**	**80.7**	**80.7**	**70.1**	**78.5**	**74.5**	**80.0**	**88.9**	74.1	**85.8**	77.5	**73.6**	**84.7**	**79.2**	80.4	**81.8**	**78.9**

- Window is better than RNN, and Stackprop is better than pipeline

Performance

- Stackprop vs. other representation

Method	UAS	LAS
NO TAGS		
Dyer et al. (2015)	92.70	90.30
Ours (window-based)	**92.85**	**90.77**
UNIVERSAL TAGSET		
Pipeline (P_{tag})	92.52	90.50
Stackprop	**93.23**	**91.30**
FINE TAGSET		
Chen & Manning (2014)	91.80	89.60
Dyer et al. (2015)	93.10	90.90
Pipeline (P_{tag})	93.10	91.16
Stackprop	**93.43**	**91.41**
Weiss et al. (2015)	93.99	92.05
Alberti et al. (2015)	94.23	92.36

WSJ dataset

Stackprop achieves similar accuracy using coarse tags as fine tags, while the pipelined baseline's performance drops dramatically

the most accurate models which use a deeper model and beam search

Performance

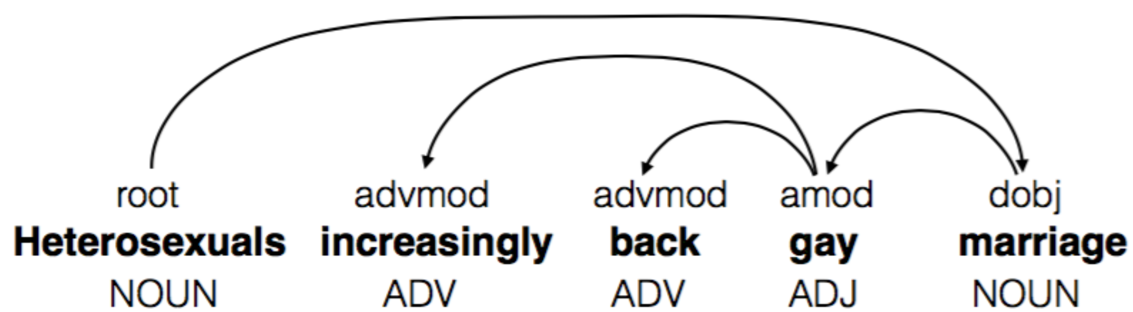
- Stackprop vs. joint modeling
- An alternative to stackprop would be to train the final layer of our architecture to predict both POS tags and dependency arcs.

Model Variant	UAS	LAS	POS
Arc-standard transition system			
Pipeline (P_{tag})	81.56	76.55	95.14
Ours (window-based)	82.08	77.08	-
Ours (Stackprop)	**83.38**	**78.78**	-
Joint parsing & tagging transition system			
Pipeline (P_{tag})	81.61	76.57	95.30
Ours (window-based)	82.58	77.76	94.92
Ours (Stackprop)	**83.21**	**78.64**	**95.43**

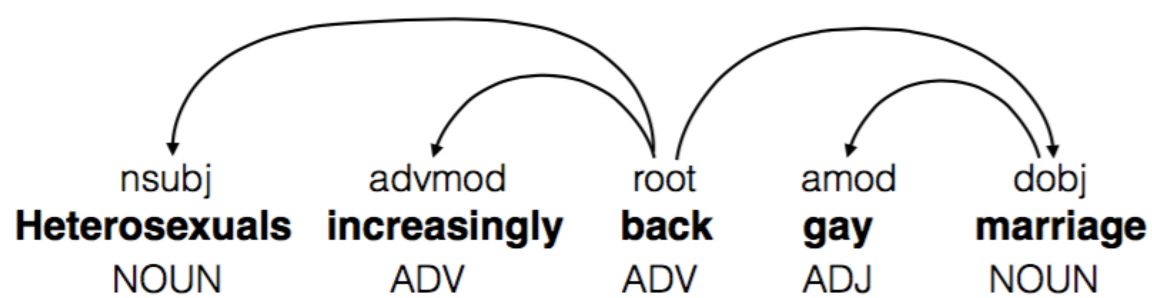
better than jointly training;
better than only window-based

Performance

- Reducing cascaded errors



(a) Tree by a pipeline model.



(b) Tree by Stackprop model.

Figure 5: Example comparison between predictions by a pipeline model and a joint model. While both models predict a wrong POS tag for the word "back" (ADV rather than VERB), the joint model is robust to this POS error and predict the correct parse tree.

observe 10.9% gain in LAS on tokens where the pipelined POS tagger makes a mistake

Performance

- Decreased model size
    - Stackprop model is reduced almost by half compared to the Pipeline model and is also roughly twice as fast
- Contextual embedding

Token	married by a **judge**.	Don't **judge** a book by	and **walked** away satisfied	when I **walk** in the door
Neighbors	mesmerizing as a *rat*. A *staple!* day at a *bar*, then go	doesn't *change* the company's won't *charge* your phone don't *waste* your money	tried, and *tried* hard and *incorporated* into and *belonged* to the	upset when I *went* to I *mean* besides me I *felt* as if I
Pattern	a [noun]	'nt [verb]	and [verb]ed	I [verb]

Thanks!