Transition-Based Dependency Parsing with Stack Long Short-Term Memory

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, Noah A. Smith

Association for Computational Linguistics (ACL), 2015

Presented By: Lavisha Aggarwal (lavisha2)

Overview

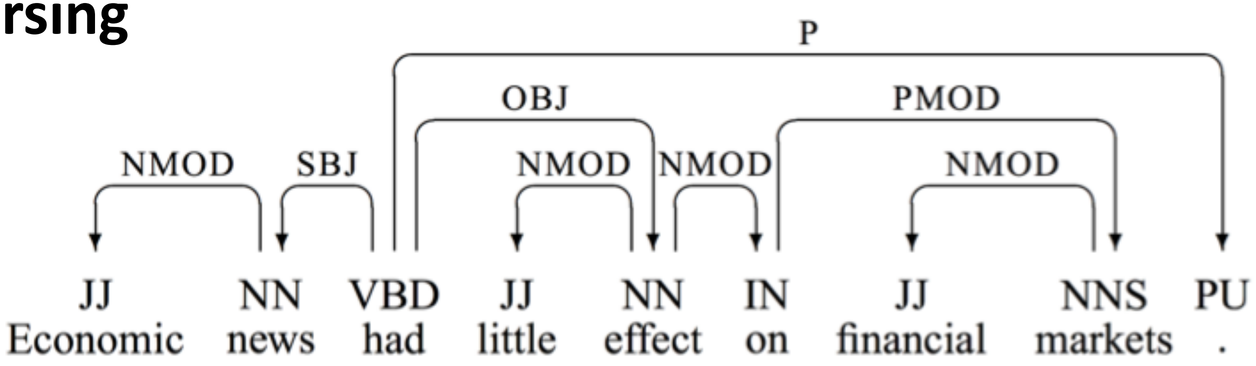
- Parsing
- Transition based dependency Parsing
- Example
- Stack LSTM’s
- Dependency parser transitions and operations
- Token Embeddings
- Experimental details
- Data
- Chen and Manning (2014)
- Results
- Conclusion
- References

What is Parsing ?

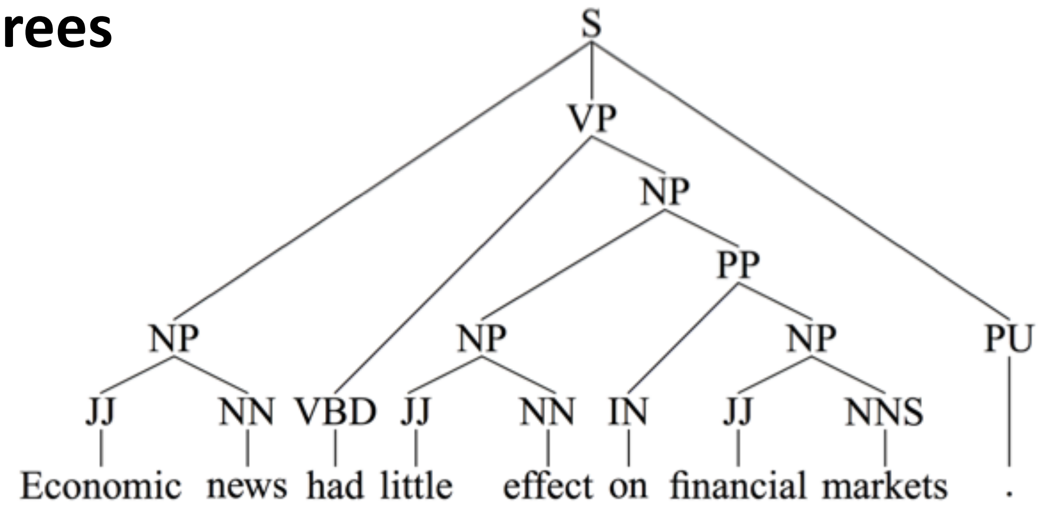
Analyzing a sentence by taking each word and determining the structure of the sentence.

Two types of Parsing

Dependency Parsing

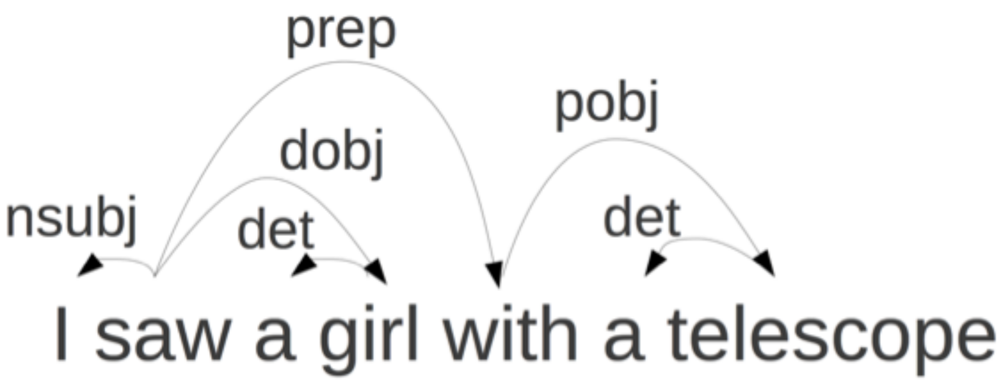


Phase structure trees



Dependency Parsing

- Represent relations between words using directed edges from Head (H) to the Dependent (D). Eg. saw(H) → girl (D)
- Dependencies can be of 2 types:



Labeled

Unlabeled

Transition-based dependency Parsing

- The parser is made up of:
  1. Stack (S) of partially processed words (Initially contains the ROOT of sentence)
  2. Buffer (B) of remaining input words (Initially contains the entire input sentence)
  3. Set of dependency arcs (A) representing actions (Initially empty)
- Series of decisions that read words sequentially from a buffer and combine them incrementally into syntactic structures

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 .]	

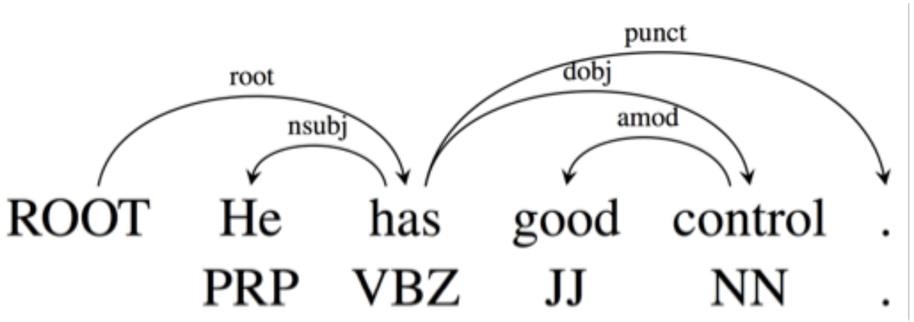
Arc-standard transition-based parser

- Notation : s_1 – Top element in stack, s_2 – 2nd element from the top in Stack
- We can have 3 types of transition actions:

1. SHIFT : Move one word from Buffer to Stack
2. LEFT-ARC (Reduce-Left): Add an arc $s_2 \Leftarrow s_1$; Remove s_2 from Stack
3. RIGHT-ARC (Reduce Right): Add an arc $s_2 \Rightarrow s_1$; Remove s_1 from Stack

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 .]	

An Example



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)

[Image credits: Chen and Manning (2014)]

Transition-based dependency parsing with Stack LSTM's

- Predict the transition actions (Shift, Left-Arc or Right-Arc) at each time step
- Based on the state of the parser (contents of Stack, Buffer and Action-set)
- Use Long short-term memory models
- Goal – Learn a representation for the various parser components that helps us determine the sequence of actions

Long Short-term Memory (LSTM)

[Input gate($\mathbf{i}_t$), Output gate ($\mathbf{o}_t$) and Forget gate ($\mathbf{f}_t$), Cell state ($\mathbf{c}_t$), Output ($\mathbf{y}_t$)]

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i)$$

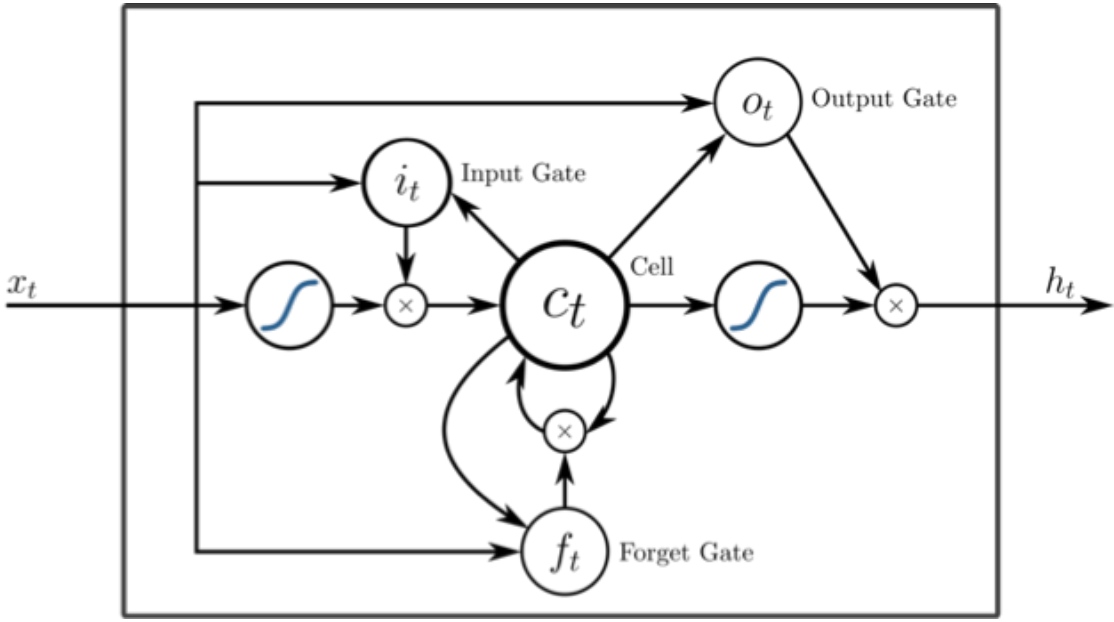
$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fc}\mathbf{c}_{t-1} + \mathbf{b}_f)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c),$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{b}_o)$$

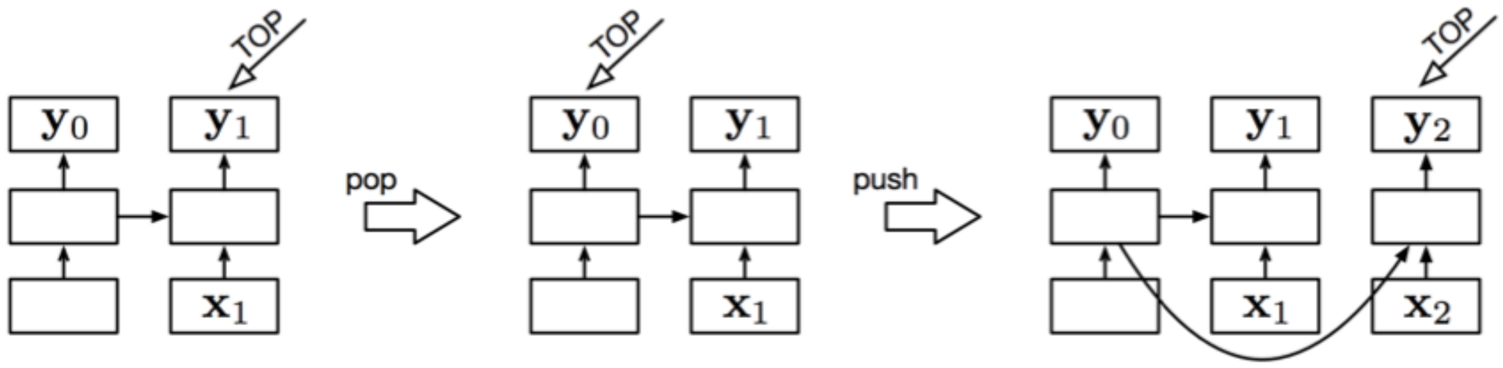
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t).$$

$$\mathbf{y}_t = g(\mathbf{h}_t)$$



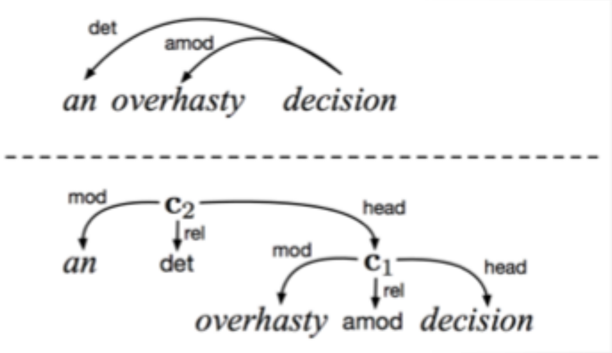
Stack LSTM

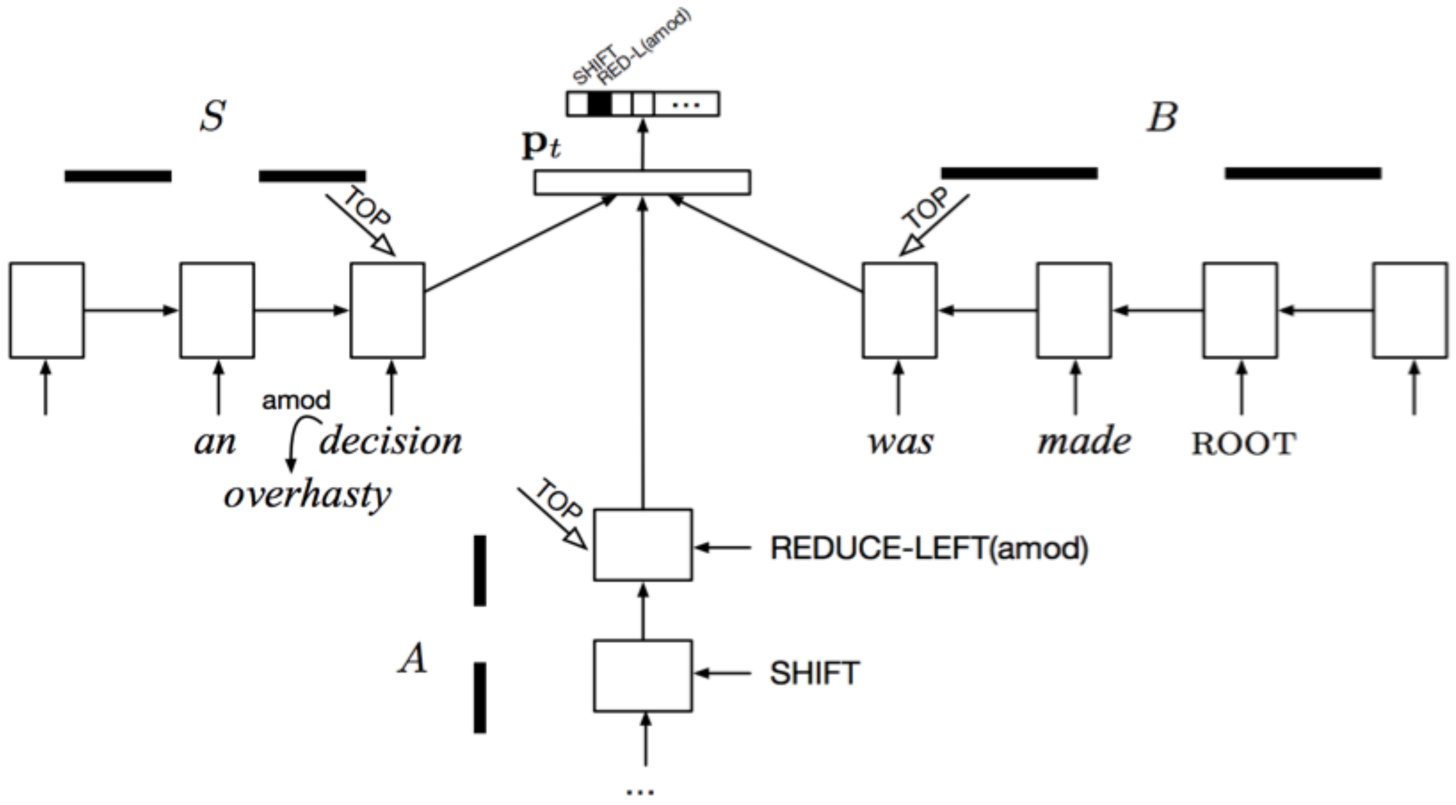
- Variation of recurrent neural networks with long short-term memory units
- Interpret LSTM as a stack that grows towards right (in the image below)
- At time t, the input $\mathbf{x}_t$, cell states and gate values output $\mathbf{y}_t$ are added as a new element to the stack – PUSH operation
- A Stack Pointer points to the TOP of stack
- For POP, simply move the stack pointer to the previous element



Dependency parser

- Buffer of words (B), Stack of syntactic trees (S) and Set of dependency actions (A)
- Each is represented by a Stack LSTM
- State of the parser at time t : $\mathbf{p}_t$





Parser Transitions

- At each time-step, perform either of the 3 Actions
- REDUCE left and right linked with a relation (r) label (amod, nmod, obj, nsubj, dobj, etc.)
- If there are K relations, total number of possible actions: 2K+1
- Store words u,v along with their respective embeddings **u,v** in S and B.
- For dependencies, store the head with the relation embedding gr(**u,v**)

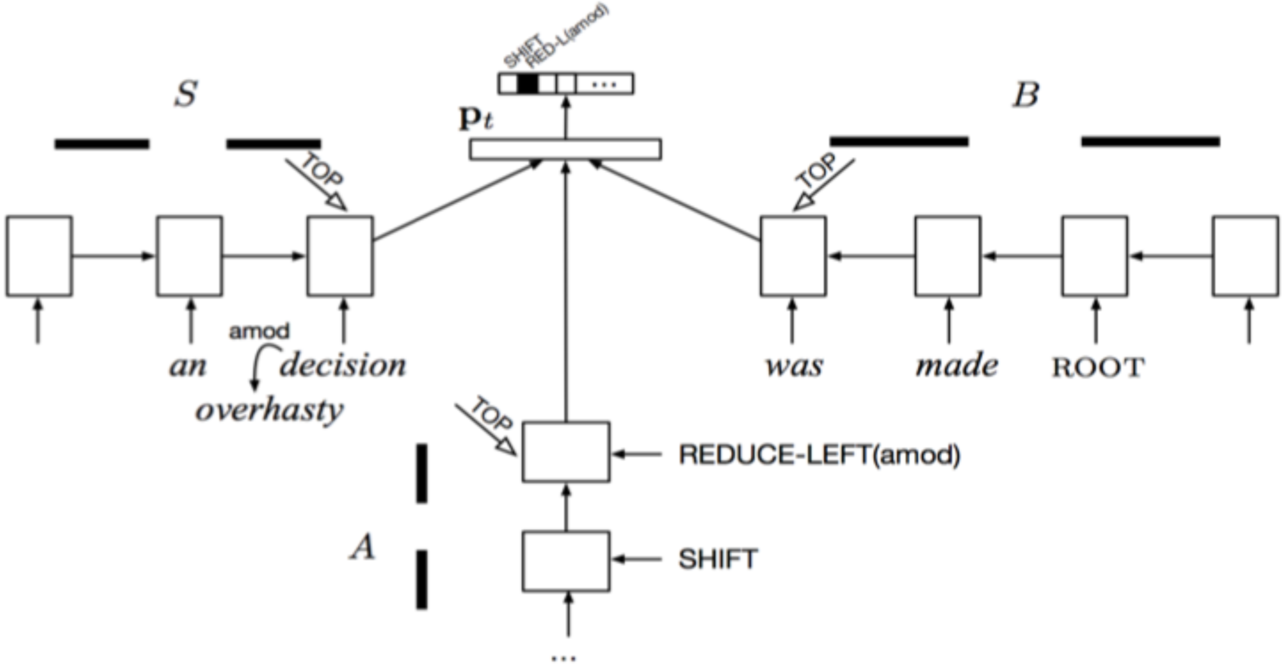
$\textbf{Stack}_t$	$\textbf{Buffer}_t$	**Action**	$\textbf{Stack}_{t+1}$	$\textbf{Buffer}_{t+1}$	**Dependency**
$(\mathbf{u}, u), (\mathbf{v}, v), S$	B	REDUCE-RIGHT(r)	$(g_r(\mathbf{u}, \mathbf{v}), u), S$	B	$u \xrightarrow{r} v$
$(\mathbf{u}, u), (\mathbf{v}, v), S$	B	REDUCE-LEFT(r)	$(g_r(\mathbf{v}, \mathbf{u}), v), S$	B	$u \xleftarrow{r} v$
S	$(\mathbf{u}, u), B$	SHIFT	$(\mathbf{u}, u), S$	B	—

Parser Operation I

- The state of the parser $\mathbf{p}_t$ at time t depends on the stack LSTM encodings of buffer B ($\mathbf{b}_t$), stack S ($\mathbf{s}_t$) and action ($\mathbf{a}_t$)

$$\mathbf{p}_t = \max\{\mathbf{0}, \mathbf{W}[\mathbf{s}_t; \mathbf{b}_t; \mathbf{a}_t] + \mathbf{d}\}$$

- **W** is a learned parameter matrix and **d** is a bias term



Parser Operation II

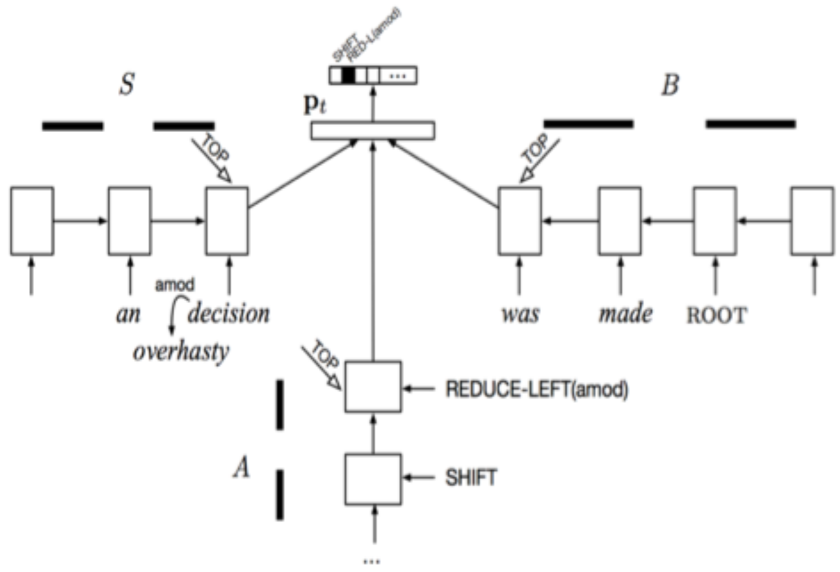
- For each possible action z_t at time t, the likelihood is determined by

$$p(z_t \mid \mathbf{p}_t) = \frac{\exp\left(\mathbf{g}_{z_t}^{\top}\mathbf{p}_t + q_{z_t}\right)}{\sum_{z' \in \mathcal{A}(S,B)} \exp\left(\mathbf{g}_{z'}^{\top}\mathbf{p}_t + q_{z'}\right)}$$

- $\mathbf{g}_z$ represents embedding of parser action z, q_z is the bias for action z
- $A(S,B)$ represents the possible actions given stack S and buffer B
- Probability of a sequence of parse actions *z*

$$p(\boldsymbol{z} \mid \boldsymbol{w}) = \prod_{t=1}^{|\boldsymbol{z}|} p(z_t \mid \mathbf{p}_t)$$

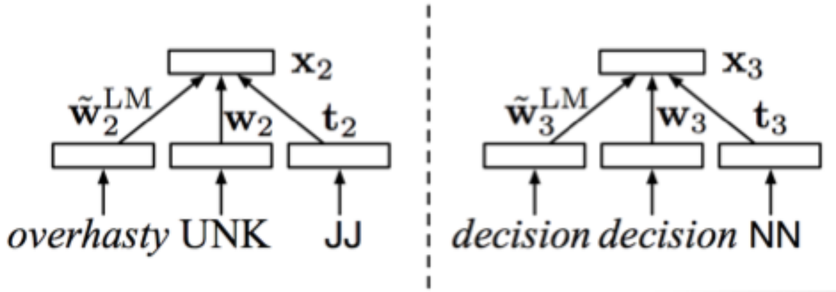
- $\boldsymbol{w}$ corresponds to the set of words of the given sentence
- Goal – Find the sequence of actions that maximize this



Token Embeddings

- Each input token $\mathbf{x}_t$ is a concatenation of 3 vectors:
  1. Learned vector representation ($\mathbf{w}$)
  2. Neural language model representation ($\mathbf{w}_{LM}$)
  3. POS tag representation ($\mathbf{t}$)

$$\mathbf{x} = \max\{\mathbf{0}, \mathbf{V}[\mathbf{w}; \tilde{\mathbf{w}}_{\mathrm{LM}}; \mathbf{t}] + \mathbf{b}\}$$



- **V** is a linear map and **b** is the bias term
- Syntactic trees represented as a composition function **c** in terms of the Syntactic head (**h**), dependent (**d**) and relation (**r**)

$$\mathbf{c} = \tanh(\mathbf{U}[\mathbf{h}; \mathbf{d}; \mathbf{r}] + \mathbf{e})$$

- **U** is a parameter matrix and **e** is a constant bias term

Experiment details

- The model is trained to learn the representations of the parser states
- Goal - Maximize the likelihood of the correct sequence of parse actions
- Training time – 8 to 12 hours
- Stochastic gradient descent with standard backpropogation
- Matrix, vector parameters initialized with uniform samples in $\pm\sqrt{6/(r+c)}$. (r rows, c columns)
- Dimensionality
  - LSTM hidden state size - 100
  - Parser actions dimensions – 16
  - Output embedding size – 20
  - Pretrained word embeddings - 100 for English and 80 for Chinese
  - Learned word embedding – 32
  - POS tag embeddings – 12

Training Data

- English
  - Stanford Dependency treebank
  - POS tags – Stanford Tagger (Accuracy – 97.3%)
  - Language model embeddings – AFP portion of English Gigaword corpus
- Chinese
  - Penn Chinese Treebank
  - Gold POS tags
  - Language model embeddings - Chinese Gigaword corpus

Experimental configurations

Testing was done on 5 experimental configurations:

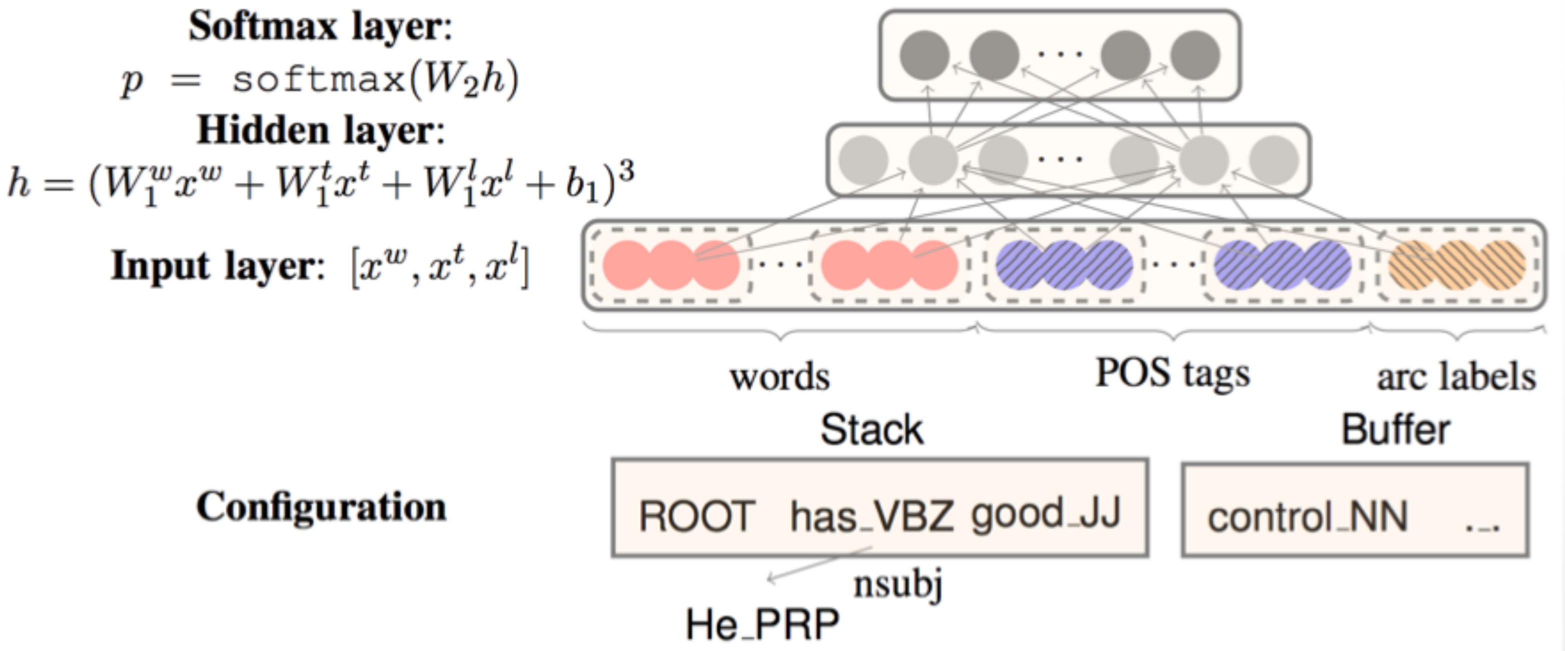
- Full stack LSTM parsing (S-LSTM)
- Without POS tags (-POS)
- Without pre-trained language model embeddings (-pre-training)
- Instead of composed representations only head words used (-composition)
- Full parsing model with RNN instead of LSTM (S-RNN)

Compared the model with Chen and Manning (2014)

Chen and Manning (EMNLP, 2014)

(A Fast and Accurate Dependency Parser using Neural Networks)

- Feed-forward Neural network architecture with 1 hidden layer (h)
- Cube activation function
- Features used - s_1, s_2, s_3, b_1, b_2, b_3
  - $lc_1(s_i)$, $lc_2(s_i)$, $rc_1(s_i)$, $rc_2(s_i)$, i=1,2 [lc - leftchild, rc - rightchild]
  - $lc_1(lc_1(s_i))$, $rc_1(rc_1(s_i))$, i=1,2



Results

- Report Unlabeled attachment scores (UAS) and Labeled attachment scores (LAS)

	Development		Test	
	UAS	LAS	UAS	LAS
S-LSTM	**93.2**	**90.9**	**93.1**	**90.9**
−POS	93.1	90.4	92.7	90.3
−pretraining	92.7	90.4	92.4	90.0
−composition	92.7	89.9	92.2	89.6
S-RNN	92.8	90.4	92.3	90.1
C&M (2014)	92.2	89.7	91.8	89.6

Table 1: English parsing results (SD)

	Dev. set		Test set	
	UAS	LAS	UAS	LAS
S-LSTM	**87.2**	**85.9**	**87.2**	**85.7**
−composition	85.8	84.0	85.3	83.6
−pretraining	86.3	84.7	85.7	84.1
−POS	82.8	79.8	82.2	79.1
S-RNN	86.3	84.7	86.1	84.6
C&M (2014)	84.0	82.4	83.9	82.4

Table 2: Chinese parsing results (CTB5)

- POS, Composition different effect in English and Chinese
- RNN and Chen&Manning lack Stack LSTM

Conclusion

- All configurations except –POS for Chinese, better than Chen and Manning
- Composition function seems to be the most important factor as the accuracy drop is largest in -composition
- Pre-training and parts of speech tagging follow as the next important things
- In English, POS do not play much role
- But in Chinese POS play a significant role
- LSTM’s outperform RNN’s but they are still better than Chen and manning
- Stack memory offers intriguing possibilities
- Achieve parsing and training in linear time (length of the input sentence)
- Beam search had minimal impact on scores

References

- Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, Noah A. Smith. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In ACL. 2015
- Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In Proc. EMNLP.
- Bernd Bohnet and Joakim Nivre. 2012. A transition based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In Proc. EMNLP.
- Jurafsky and Martin. Dependency Parsing. Speech and Language Processing, Chapter 14, Stanford
- Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, Noah A. Smith. Greedy Transition-Based Dependency Parsing with Stack LSTMs. In Proc. ACL 2017
- Jinho D. Choi and Andrew McCallum. 2013.Transition-based dependency parsing with selectional branching. In Proc. ACL.
- Julia Hockenmaier. Dependency Parsing Lecture 8, Natural Language Processing CS447, UIUC
- Richard Socher. Natural Language Processing with Deep Learning. CS224N, Stanford
- Graham Neubig. Neural Networks for NLP Transition-based Parsing with Neural Nets. CS11-747, CMU