Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representation

Eliyahu Kiperwasser & Yoav Goldberg 2016

Presented by: Yaoyang Zhang

Outline

Simple and Accurate **Dependency Parsing** Using **Bidirectional LSTM Feature Representation**

- Background – **Bidirectional RNN**
- Background – **Dependency Parsing**
- Motivation– **Bidirectional RNN as feature functions**
- Model for transition-based parser
- Model for graph-based parser
- Results and conclusion

Bidirectional Recurrent Neural Network

- RNN has memory of the past up to time *i*, at step *i*
- What if we also have memory of the "future"? Since we are talking about text, the preceding and succeeding context should both carry some weight
- Use two RNNs, with different directions
- Each direction has its own set of parameters
- Use LSTM cells

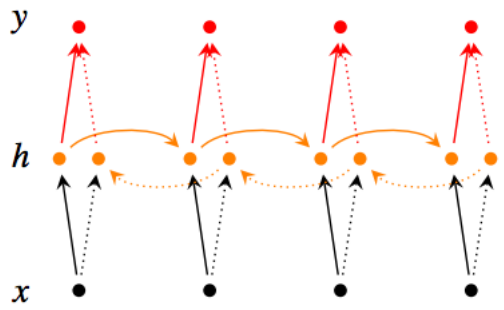
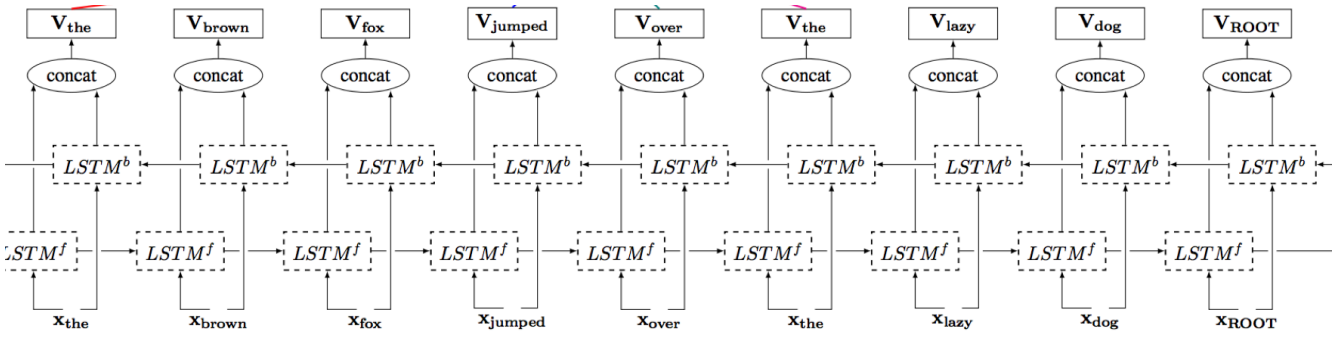


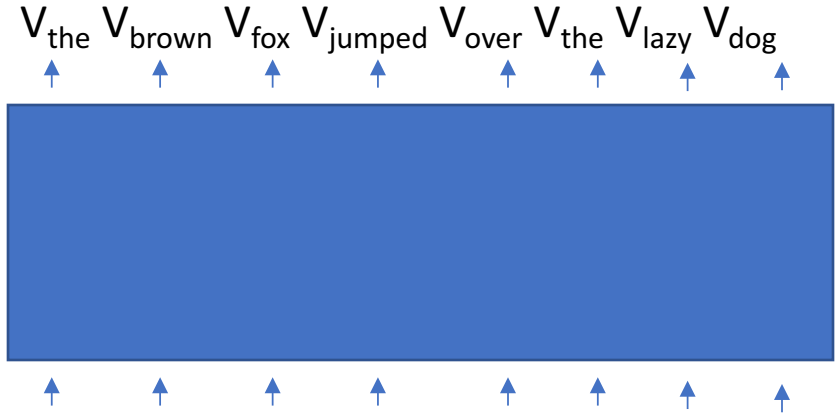
Figure 6: A bi-directional RNN model

[1]Figures borrowed from Stanford CS 244d notes

Bidirectional Recurrent Neural Network

- Why and how to use BiRNN for dependency parsing
  - Motivation: get a vector representation for each word in a sentence, which will later be used as feature input for parsing algorithm
  - One BiRNN per sentence
  - Will be trained jointly with a classifier/regressor depending on the parsing model

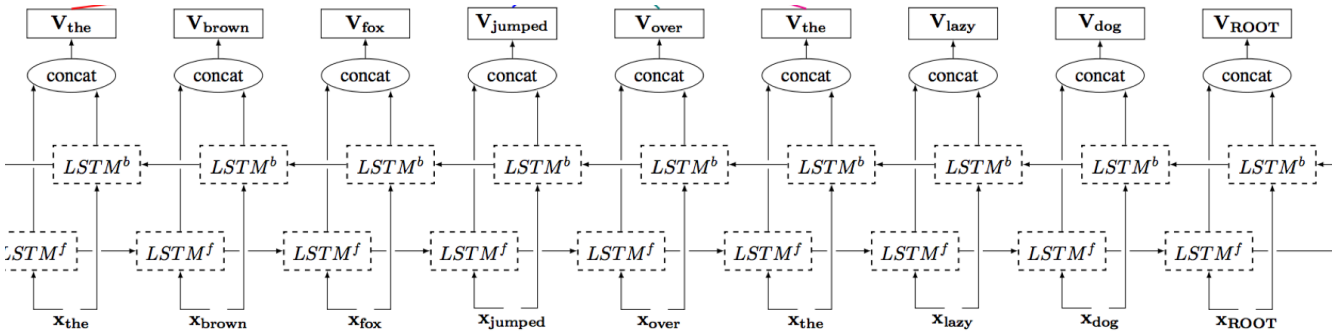




The brown fox jumped over the lazy dog

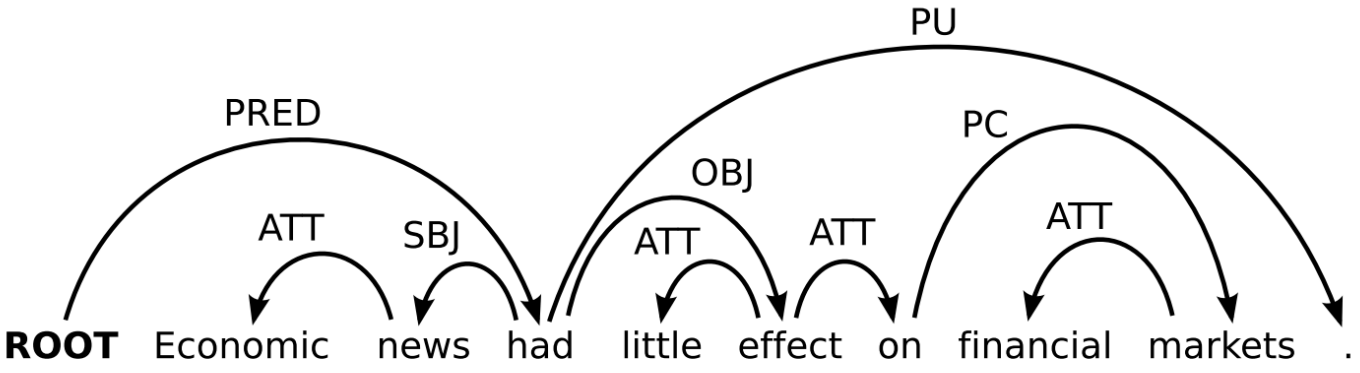
Bidirectional Recurrent Neural Network

- Input: words $w_1, w_2, \ldots, w_n$, POS tag $t_1, t_2, \ldots, t_n$
- Input to BiLSTM: $x_i = e(w_i) | e(t_i)$
  - e(): embedding of word/tag, jointly trained with the network
  - |: concatenation
- Output from BiLSTM: $v_i =$ $\mathrm{BiRNN}_\theta(x_{1:n}, i) = \mathrm{RNN}_F(x_{1:i}) \circ \mathrm{RNN}_R(x_{n:i})$
  - Feature representation
  - Output $\mathrm{BiRNN}_\theta(x_{1:n}, i)$ is the concatenation of the outputs from two directions



Dependency Grammar

- A grammar model
- The syntactic structure of a sentence is described solely in terms of the words in a sentence and an associated set of directed binary grammatical relations that hold among the words[1]
- TL; DR: Dependency grammar assumes that syntactic structure consists only of dependencies[2]



[1] Speech and Language Processing, Chapter 14

[2] CS447 slide

Dependency Grammar

- There are other grammar models out there such as context-free grammar, but we are focusing on dependency grammar here
- Dependency parsing the process of getting the parse tree out of a sentence
- Dependency structures:
  - Each dependency is a directed edge from one word to another
  - Dependencies form a connected and acyclic graph over the words in a sentence
  - Every node (word) has at most one incoming edge
  - $\Longrightarrow$ It is a rooted tree
- Universal dependencies: 37 syntactic relations for any language (with modification)

Parsing Algorithms

- Transition-based v.s. Graph-based
- Transition-based:
  - start with an initial state (empty stack, all words in a queue/buffer, empty dependencies)
  - greedily choose an action (shift, left-arc, right-arc) based on the current state
  - Repeat until reaching a terminal state (empty stack, empty queue, parse tree)
- Graph-based:
  - All the possible edges are associated with some scores
  - Different parse trees have different total scores
  - Use an (usually dynamic programming) algorithm to find the tree with the highest score

Transition-based Dependency Parsing

- s: sentence
- w: word
- t: transition (action)
- c: configuration (state)
- Initial: empty stack, all words in the queue, empty dependencies
- Terminal: empty stack, empty queue, dependency tree
- Legal: shift, reduce, left-arc(label), right-arc(label)
- Scorer($\phi(c)$, t): given feature $\phi(c)$, outputs score for action t

Algorithm 1 Greedy transition-based parsing

Input: sentence $s = w_1, \ldots, x_w,\ t_1, \ldots, t_n$, parameterized function $\textsc{Score}_\theta(\cdot)$ with parameters θ.
$c \leftarrow \textsc{Initial}(s)$
while not $\textsc{Terminal}(c)$ **do**
  $\hat{t} \leftarrow \arg\max_{t \in \textsc{Legal}(c)} \textsc{Score}_\theta(\phi(c), t)$
  $c \leftarrow \hat{t}(c)$
return $tree(c)$

Transition-based Dependency Parsing

Economic news had little effect on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	$\emptyset$)
SH $\Rightarrow$	([ROOT, Economic],	[news, . . . , .],	$\emptyset$)
LA_{ATT} $\Rightarrow$	([ROOT],	[news, . . . , .],	$A_1 = \{(\text{news, ATT, Economic})\}$)
SH $\Rightarrow$	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} $\Rightarrow$	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(\text{had, SBJ, news})\}$)
SH $\Rightarrow$	([ROOT, had],	[little, . . . , .],	A_2)
SH $\Rightarrow$	([ROOT, had, little],	[effect, . . . , .],	A_2)
LA_{ATT} $\Rightarrow$	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup \{(\text{effect, ATT, little})\}$)
SH $\Rightarrow$	([ROOT, had, effect],	[on, . . . , .],	A_3)
SH $\Rightarrow$	([ROOT, . . . on],	[financial, markets, .],	A_3)
SH $\Rightarrow$	([ROOT, . . . , financial],	[markets, .],	A_3)
LA_{ATT} $\Rightarrow$	([ROOT, . . . on],	[markets, .],	$A_4 = A_3 \cup \{(\text{markets, ATT, financial})\}$)
RA_{PC} $\Rightarrow$	([ROOT, had, effect],	[on, .],	$A_5 = A_4 \cup \{(\text{on, PC, markets})\}$)
RA_{ATT} $\Rightarrow$	([ROOT, had],	[effect, .],	$A_6 = A_5 \cup \{(\text{effect, ATT, on})\}$)
RA_{OBJ} $\Rightarrow$	([ROOT],	[had, .],	$A_7 = A_6 \cup \{(\text{had, OBJ, effect})\}$)
SH $\Rightarrow$	([ROOT, had],	[.],	A_7)
RA_{PU} $\Rightarrow$	([ROOT],	[had],	$A_8 = A_7 \cup \{(\text{had, PU, .})\}$)
RA_{PRED} $\Rightarrow$	([ ],	[ROOT],	$A_9 = A_8 \cup \{(\text{ROOT, PRED, had})\}$)
SH $\Rightarrow$	([ROOT],	[ ],	A_9)

Actions

States = (stack, queue, set)

[1]Borrowed from CS 447 slides

Transition-based Dependency Parsing - Motivation

- How to get feature $\phi(c)$, given the current state c?
  - Old-school: “Hand-crafted” features (templates) – can have as many as 72 templates
  - Now: Deep learning (Bidirectional LSTM)
  - $\phi(c)$ is actually a simple function of the BiRNN output vectors!
- Once we get the feature $\phi(c)$, the rest is straightforward
  - Train a classifier based on $\phi(c)$ and output t

Transition-based Dependency Parsing

$$MLP_\theta(x) = W^2 \cdot \tanh(W^1 \cdot x + b^1) + b^2$$

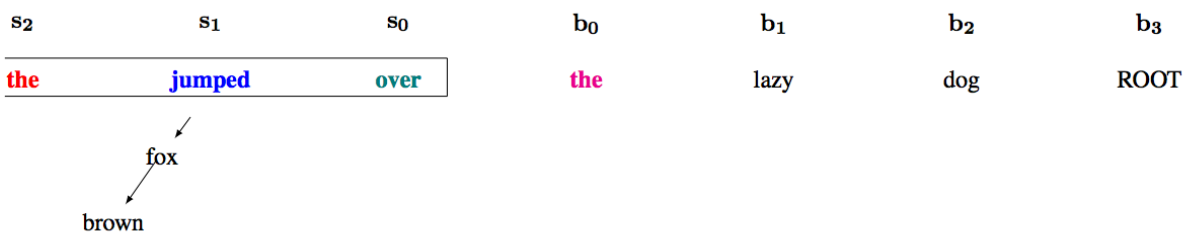
- Output from BiLSTM: v_i
  - Feature representation
- Input to classifier (Multi-layer perceptron, MLP): $\phi(c)$
  - c: state at time i, $(\ldots|s_2|s_1|s_0, \quad b_0|\ldots, \quad T)$
  - $\phi(c) = v_{s_2} \circ v_{s_1} \circ v_{s_0} \circ v_{b_0}$
- Output from MLP: a vector of scores for all possible actions
- Objective (max-margin): Maximize the difference between the score of the correct action and the maximum score of all incorrect actions
  - G: correct (gold) actions
  - A: all actions

$$max\Big(0, 1 - \max_{t_o \in G} MLP\big(\phi(c)\big)[t_o] + \max_{t_p \in A \setminus G} MLP\big(\phi(c)\big)[t_p]\Big)$$

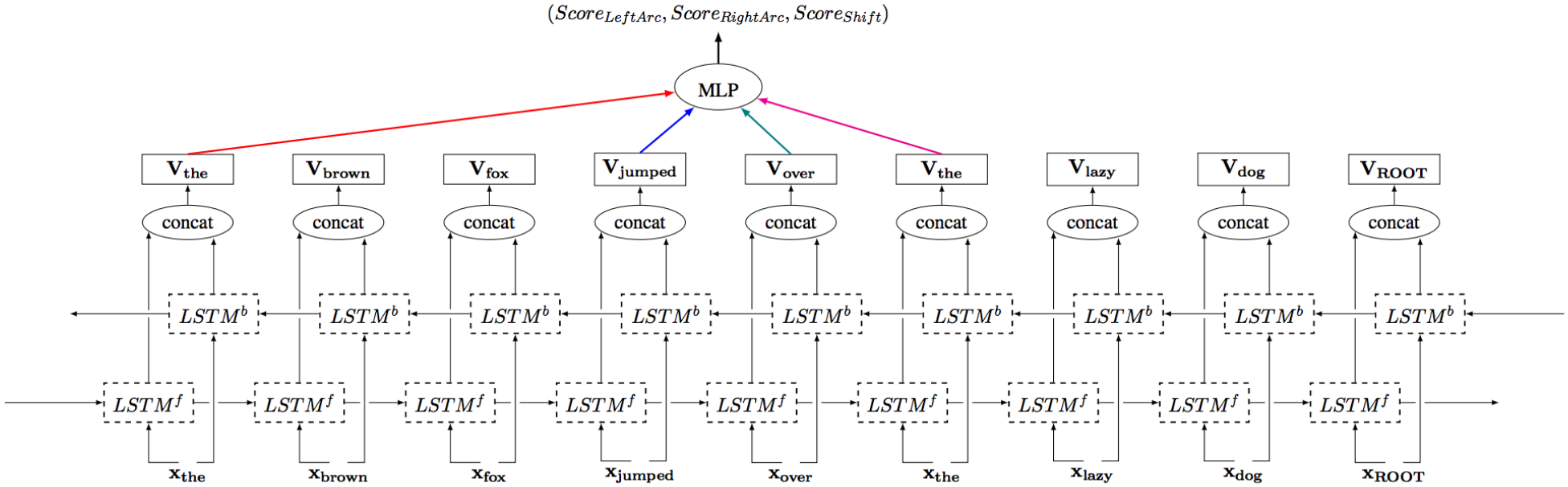
Transition-based Dependency Parsing

- Put everything together:

Configuration:



Scoring:



Transition-based Dependency Parsing

- Other things to note:
  - Error exploration and dynamic oracle: a technique to explore wrong configurations to reduce overfitting, needs to redefine G (called dynamic oracle)
  - Aggressive exploration: with some (small) probability to follow the wrong configuration if the difference of scores between the correct and incorrect actions are small enough. Further reduces overfitting

Graph-based Dependency Parsing

- Input: sentence s, chooses a tree y that score the highest (general form)
  - Score of a tree y is the summation of scores of all its subtrees

$$predict(s) = \arg\max_{y \in \mathcal{Y}(s)} score_{global}(s, y)$$

$$score_{global}(s, y) = \sum_{part \in y} score_{local}(s, part)$$

Graph-based Dependency Parsing

- Arc-factored graph: relaxes the assumption. Decompose the score of a tree into the sum of scores of arcs.
  - $\phi(s, h, m)$: feature function of edge (h, m) in the sentence s

$$parse(s) = \arg\max_{y \in \mathcal{Y}(s)} \sum_{(h,m) \in y} score\big(\phi(s, h, m)\big)$$

- Efficient DP algorithm to find the parse tree if $\phi(s, h, m)$ is given (Eisner's decoding algorithm)
- Again, how to get the feature function $\phi(s, h, m)$?
  - Of course use vector representation from BiRNN. Concatenation of the two vectors for h and m

Graph-based Dependency Parsing

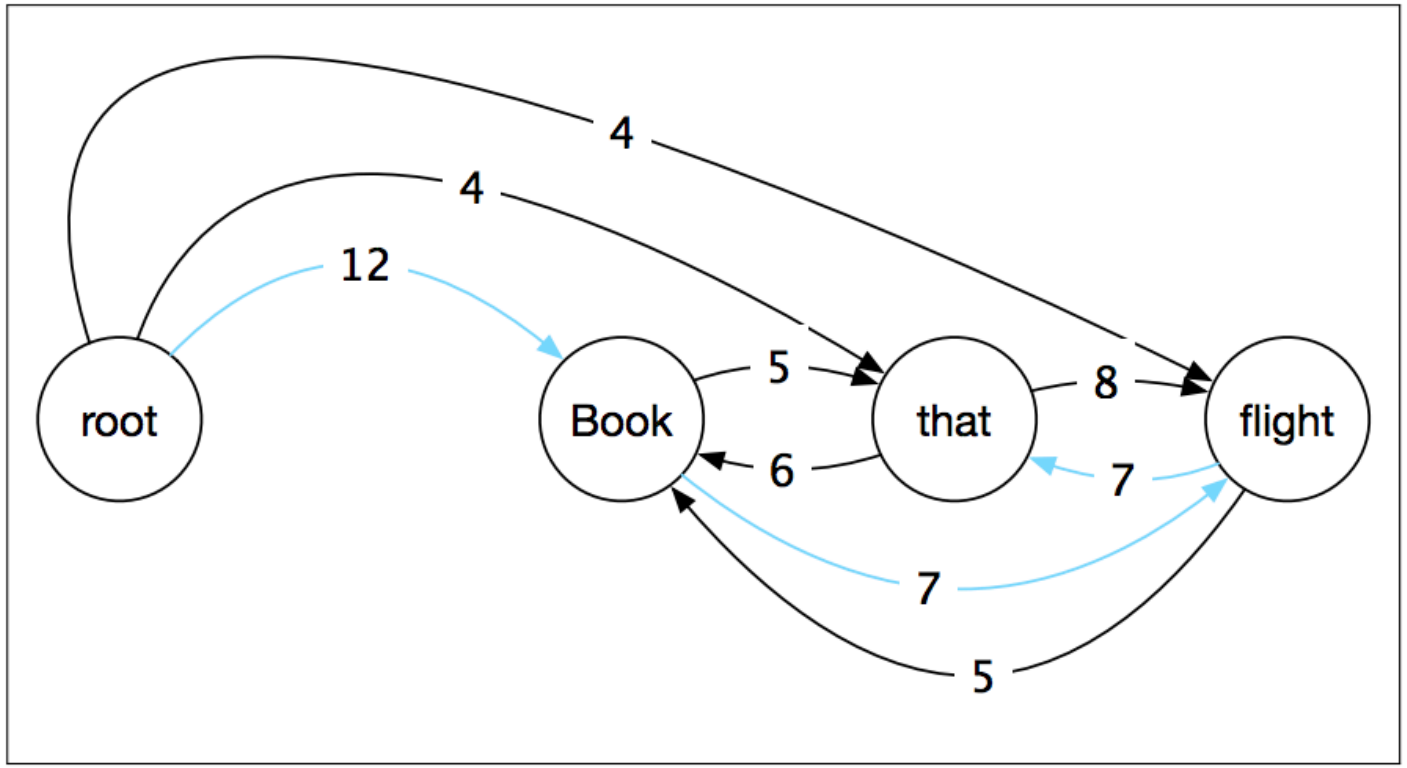


Figure 14.12 Initial rooted, directed graph for *Book that flight*.

[1]Speech and Language Processing, Chapter 14

The Model (for Graph-Based)

- Output from BiLSTM: v_i=BIRNN($x_{1:n}$,i)
  - Feature representation
- Input to regressor (Multi-layer perceptron, MLP): $\phi(s, h, m)$

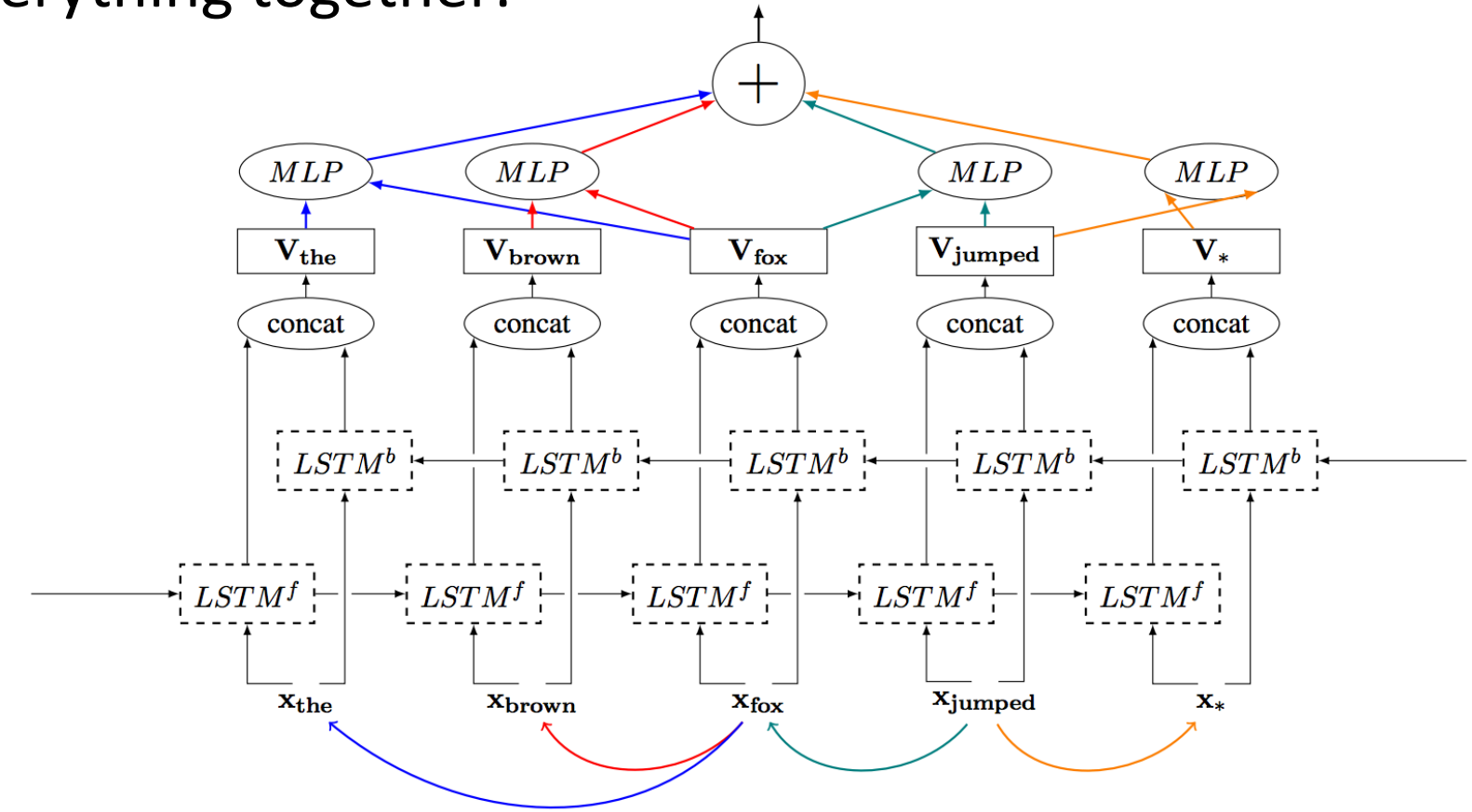
$$\phi(s, h, m) = \textsc{BiRNN}(x_{1:n}, h) \circ \textsc{BiRNN}(x_{1:n}, m)$$

- Output from MLP: score for this edge
- Objective (max-margin, similar to transition-based):
  - y: correct tree, y': incorrect tree

$$max\Big(0, 1 - \max_{y' \neq y} \sum_{(h,m) \in y'} MLP(v_h \circ v_m) + \sum_{(h,m) \in y} MLP(v_h \circ v_m)\Big)$$

The Model (for Graph-Based)

- Put everything together:



The Model (for Graph-Based)

- Other things to note:
  - Labeled parsing: (similar to transition based)

$$label(h, m) = \arg\max_{\ell \in labels} MLP_{LBL}(v_h \circ v_m)[\ell]$$

  - Loss augmented inference: prevent overfitting. Penalize trees that have high scores but are also VERY wrong

$$max(0, 1 + score(x, y) - \max_{y' \neq y} \sum_{part \in y'} (score_{local}(x, part) + \mathbb{1}_{part \notin y}))$$

Experiment and Results

- Training:
  - Dataset: Stanford Dependency (SD) for English, Penn Chinese Treebank 5.1 (CTB5)
  - Word dropout: a word is replaced with an unknown symbol with probability proportional to the inverse of its frequency
  - 30 iterations
  - Hyper-parameters

Word embedding dimension	100
POS tag embedding dimension	25
Hidden units in MLP	100
Hidden units in MLP_{LBL}	100
BI-LSTM Layers	2
BI-LSTM Dimensions (hidden/output)	125 / 125
α (for word dropout)	0.25
p_{agg} (for exploration training)	0.1

Table 2: Hyper-parameter values used in experiments

Experiment and Results

- UAS: unlabeled attachment score, LAS: labeled attachment score
- Model much simpler but very competitive results

System	Method	Representation	Emb	PTB-YM	PTB-SD		CTB	
				UAS	UAS	LAS	UAS	LAS
This work	graph, 1st order	2 BiLSTM vectors	–	–	93.1	91.0	**86.6**	**85.1**
This work	transition (greedy, dyn-oracle)	4 BiLSTM vectors	–	–	93.1	91.0	86.2	85.0
This work	transition (greedy, dyn-oracle)	11 BiLSTM vectors	–	–	**93.2**	**91.2**	86.5	84.9
ZhangNivre11	transition (beam)	large feature set (sparse)	–	92.9	–	–	86.0	84.4
Martins13 (TurboParser)	graph, 3rd order+	large feature set (sparse)	–	92.8	93.1	–	–	–
Pei15	graph, 2nd order	large feature set (dense)	–	93.0	–	–	–	–
Dyer15	transition (greedy)	Stack-LSTM + composition	–	–	92.4	90.0	85.7	84.1
Ballesteros16	transition (greedy, dyn-oracle)	Stack-LSTM + composition	–	–	92.7	90.6	86.1	84.5
This work	graph, 1st order	2 BiLSTM vectors	YES	–	93.0	90.9	86.5	84.9
This work	transition (greedy, dyn-oracle)	4 BiLSTM vectors	YES	–	93.6	91.5	87.4	85.9
This work	transition (greedy, dyn-oracle)	11 BiLSTM vectors	YES	–	93.9	91.9	**87.6**	86.1
Weiss15	transition (greedy)	large feature set (dense)	YES	–	93.2	91.2	–	–
Weiss15	transition (beam)	large feature set (dense)	YES	–	**94.0**	**92.0**	–	–
Pei15	graph, 2nd order	large feature set (dense)	YES	93.3	–	–	–	–
Dyer15	transition (greedy)	Stack-LSTM + composition	YES	–	93.1	90.9	87.1	85.5
Ballesteros16	transition (greedy, dyn-oracle)	Stack-LSTM + composition	YES	–	93.6	91.4	**87.6**	**86.2**
LeZuidema14	reranking /blend	inside-outside recursive net	YES	93.1	93.8	91.5	–	–
Zhu15	reranking /blend	recursive conv-net	YES	93.8	–	–	85.7	–