Carnegie Mellon

Stack-Pointer Networks for Dependency Parsing

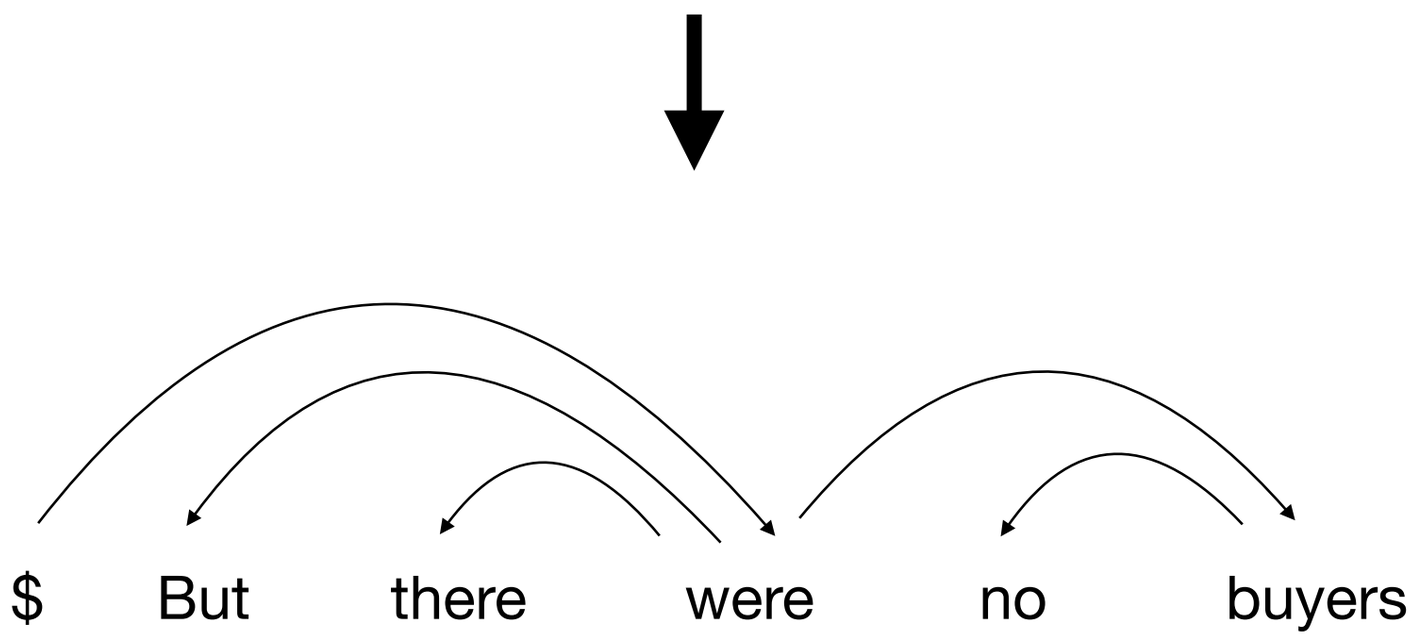
Xuezhe Ma[1], Zecong Hu[2], Jingzhou Liu[1],
Nanyun Peng[3], Graham Neubig[1] and Eduard Hovy[1]

[1]Carnegie Mellon University [2]Tsinghua University [3]University of Southern California

https://github.com/XuezheMax/NeuroNLP2

Language Technologies Institute

Dependency Parsing



Transition-based Parsing

- Process the input **sequentially** in order

- Use **actions** that build up a tree

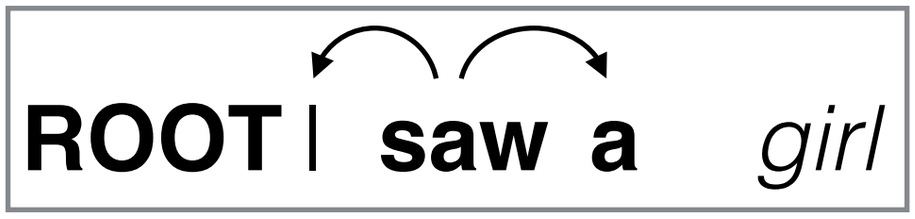
- Choose which actions to apply with a **classifier**

Example: Arc-standard Parsing [Yamada+ 2003, Nivre 2004]

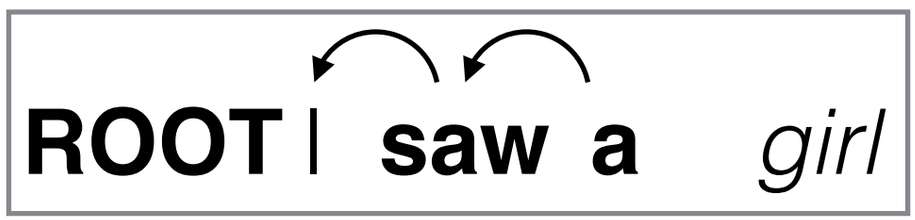
- **Order:** Left-to-right
- **Actions:** Shift, reduce-right, reduce-left

shift

right

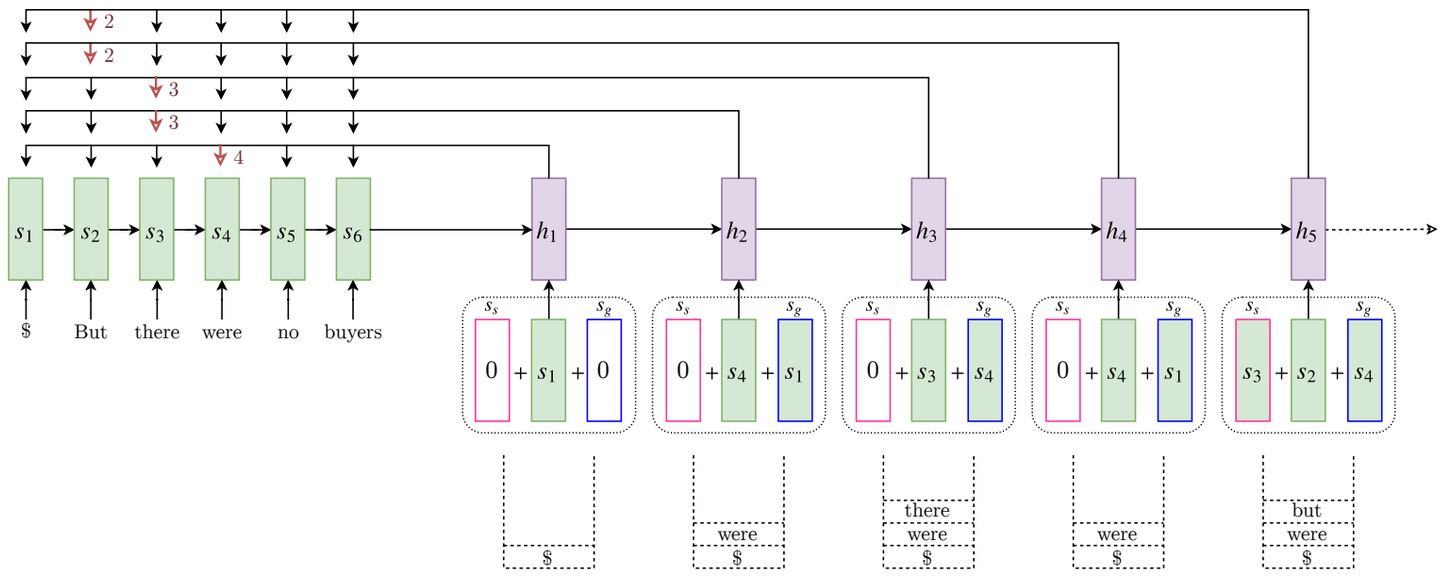


left



- **Classifier:**
  - Support vector machines [Nivre+ 2004]
  - Feed-forward neural networks [Chen+ 2014]
  - Recurrent neural networks [Dyer+ 2015]

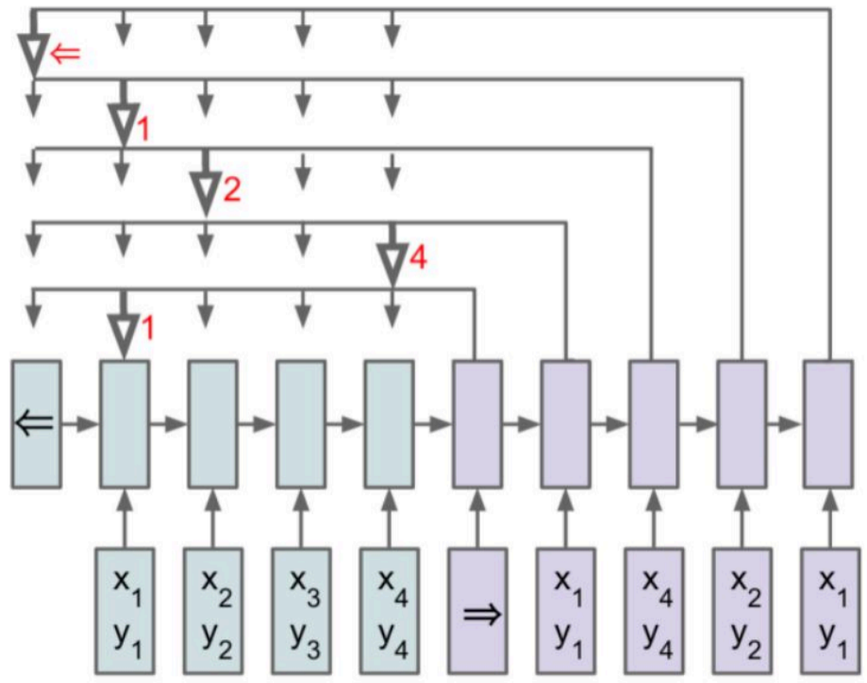
Our Proposal: Stack-pointer Networks (StackPtr)



- **Order:** Top-down, depth-first
- **Actions:** "Point" to the next word to choose as a child
- **Model:** A neural network, based on "pointer networks"
- **Advantages:**
  - Top-down parsing maintains a global view of the sentence
  - High accuracy
  - Can maintain full history, low asymptotic running time (c.f. graph-based)

Background: Pointer Network [Vinyals+ 2015]

- Output sequence with elements that are discrete tokens corresponding to positions in an input sequence



- Use attention as a pointer to select a member of the input sequence as the output

$$e_i^t = score(h_t, s_i)$$

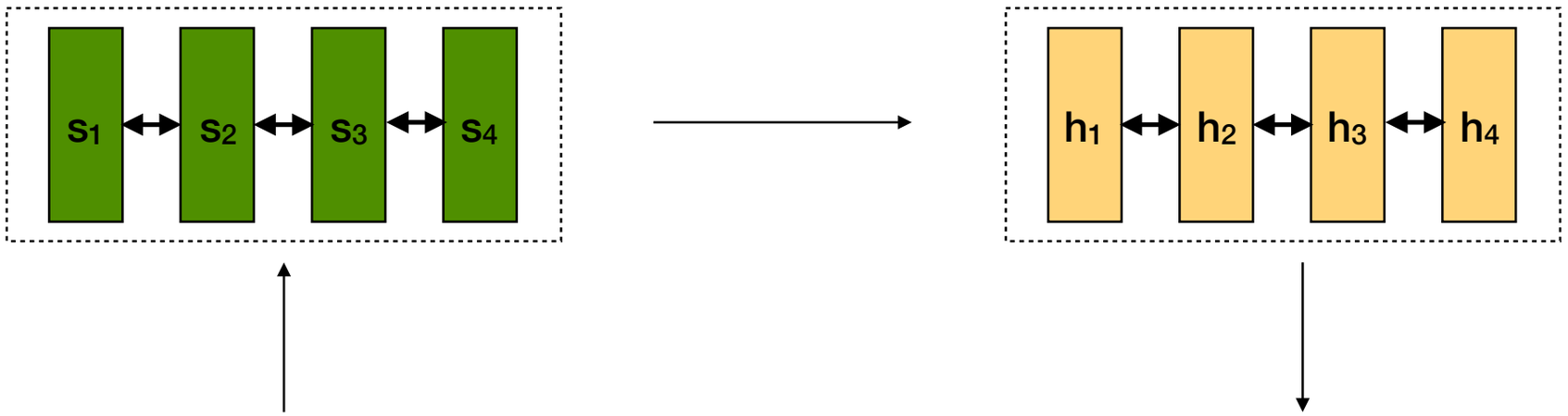
$$a^t = softmax(e^t)$$

s and h are the hidden states of encoder and decoder, and *score()* is the *attention scoring function*, *e.g.* *bi-affine attention* [Luong+ 2015; Dozat+ 2017]

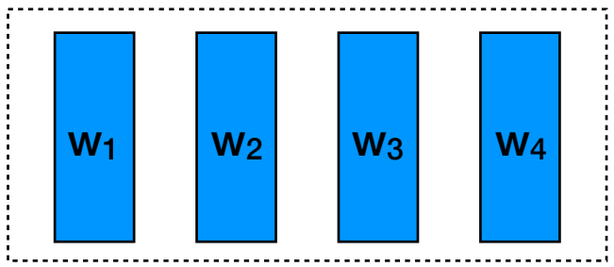
Variable Definitions

$\mathbf{s} = \{s_1, \ldots, s_n\}$: hidden states of encoder

$\mathbf{h} = \{h_1, \ldots, h_n\}$: hidden states of decoder



$\mathbf{x} = \{w_1, \ldots, w_n\}$: an input sentence



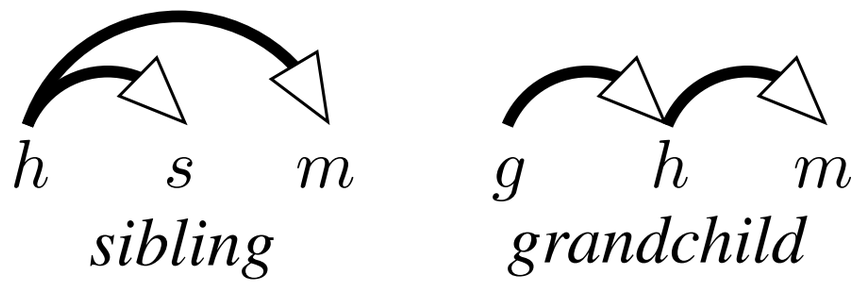
$\mathbf{y} = \{p_1, \ldots, p_n\}$: a sequence of paths, each of which is a sequence of words from root to a leaf

Transition System

- **Two data structures**
  - **List (α)**: of words whose head has not been selected
  - **Stack (σ)**: of partially processed head words whose children have not been fully selected
- **Stack σ is initialized with the root symbol** $
- **At each decoding step** t
  - receive the top element of stack σ as head word w_h, and generate the hidden state h_t
  - compute the attention vector a^t using h_t and encoder hidden states s
  - **generate an arc:** choose a specific word (w_c) from α as the child of w_h, remove w_c from α and push it onto σ
  - **complete a head:** pop w_h out of σ

Features for the Classifier

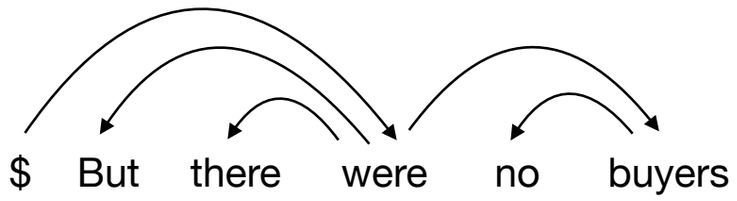
- **Utilize higher-order information at each step of the top-down decoding procedure**
- ***Sibling*** **and** ***Grandchild*** **structures**
  - proven beneficial for parsing performance (McDonald and Pereira 2006; Koo and Collins 2010)



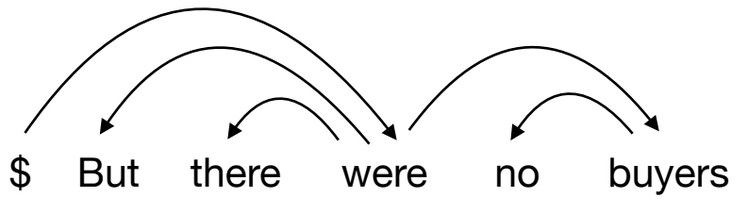
- **Use element-wise sum of the encoder hidden states instead of concatenation**
  - does not increase the dimension of β_t

$$\beta_t = s_h + s_g + s_s$$

Example

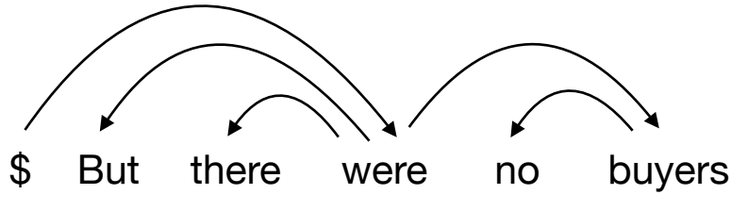


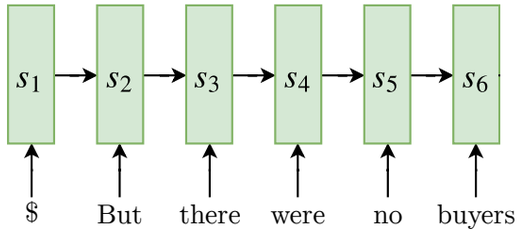
Example



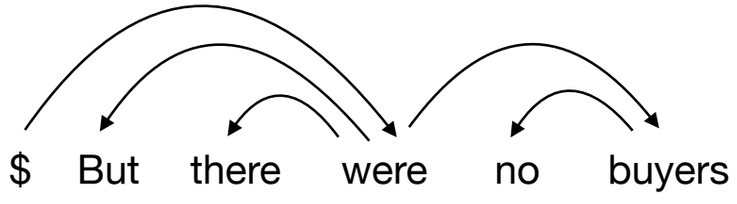
$ But there were no buyers

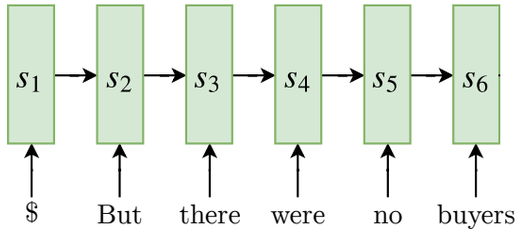
Example

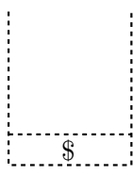

$
But
there
were
no
buyers


s_1
s_2
s_3
s_4
s_5
s_6
$
But
there
were
no
buyers

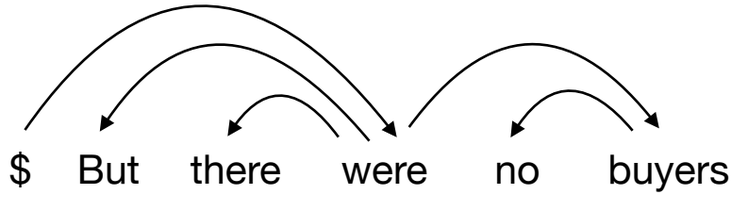
Example

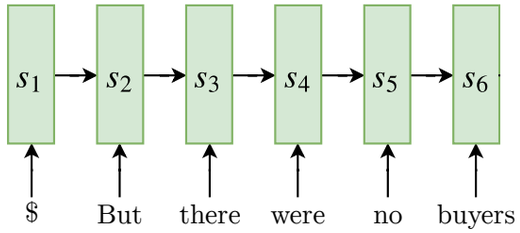

$
But
there
were
no
buyers

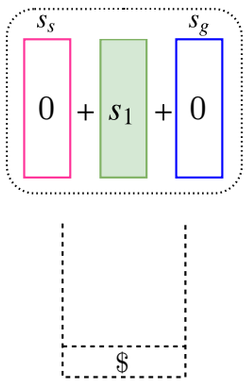

s_1
s_2
s_3
s_4
s_5
s_6
$
But
there
were
no
buyers


$

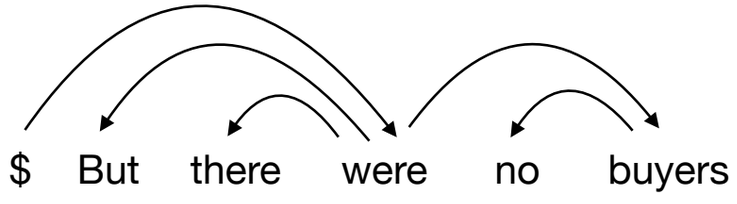
Example

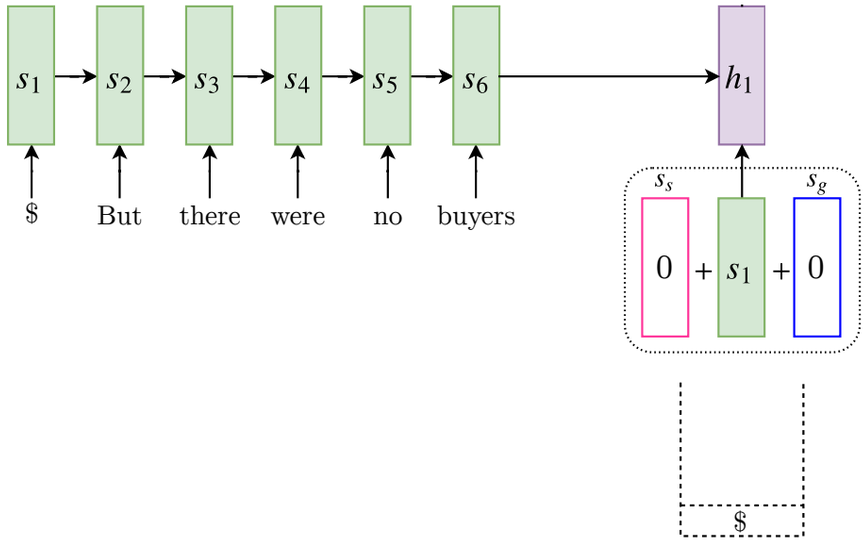

$
But
there
were
no
buyers


s_1
s_2
s_3
s_4
s_5
s_6
$
But
there
were
no
buyers

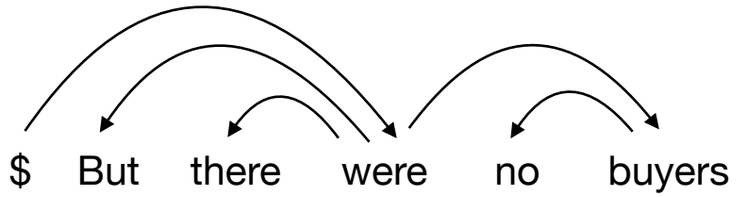

s_s
s_g
0
+
s_1
+
0
$

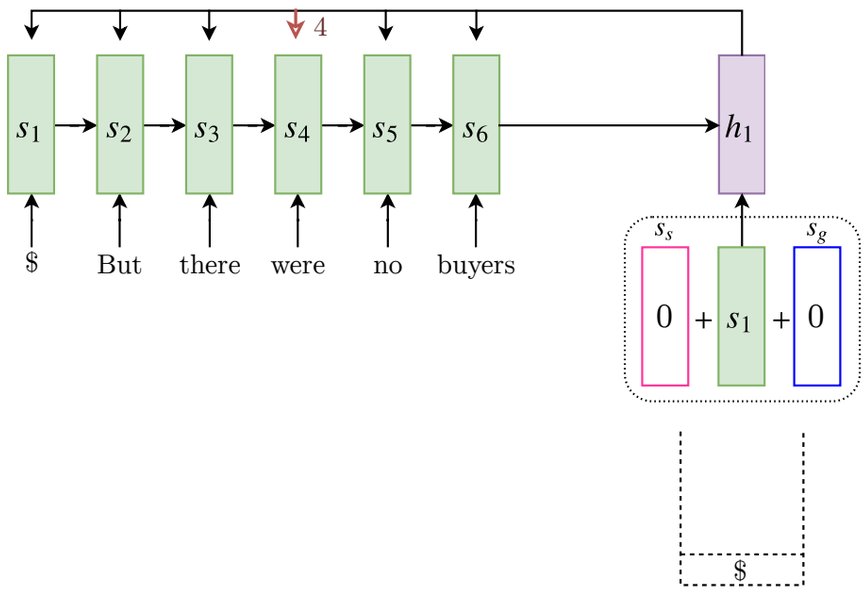
Example


$
But
there
were
no
buyers

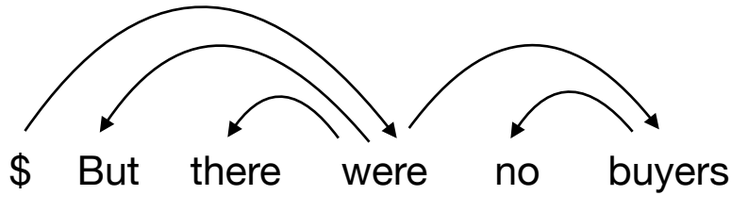

s_1
s_2
s_3
s_4
s_5
s_6
h_1
$
But
there
were
no
buyers
s_s
s_g
0
+
s_1
+
0
$

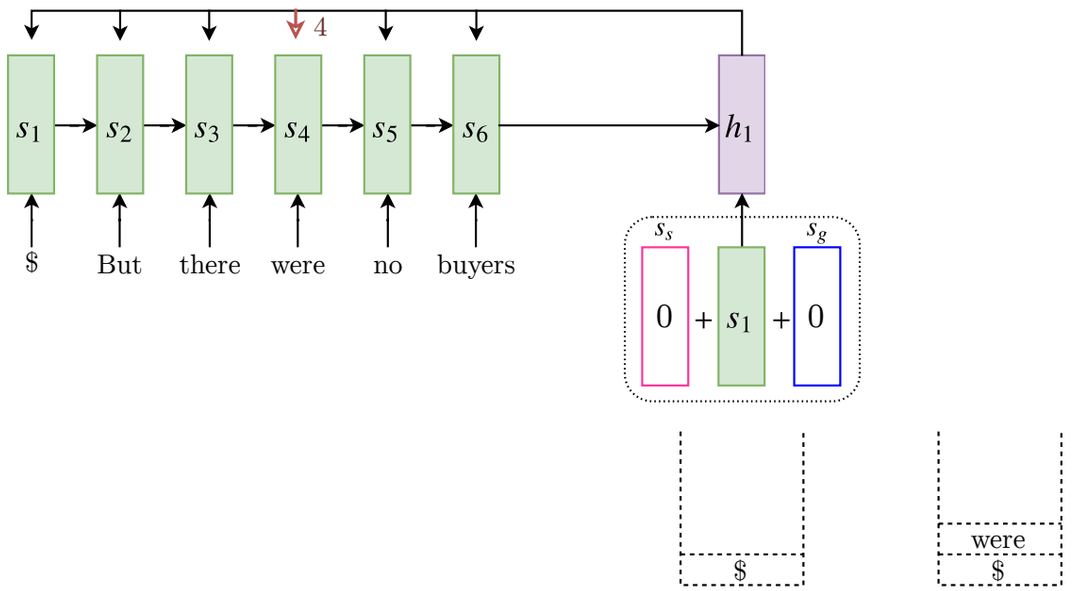
Example


$
But
there
were
no
buyers

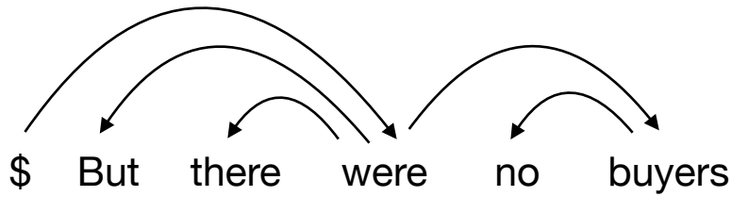

4
s_1
s_2
s_3
s_4
s_5
s_6
h_1
$
But
there
were
no
buyers
s_s
s_g
0
+
s_1
+
0
$

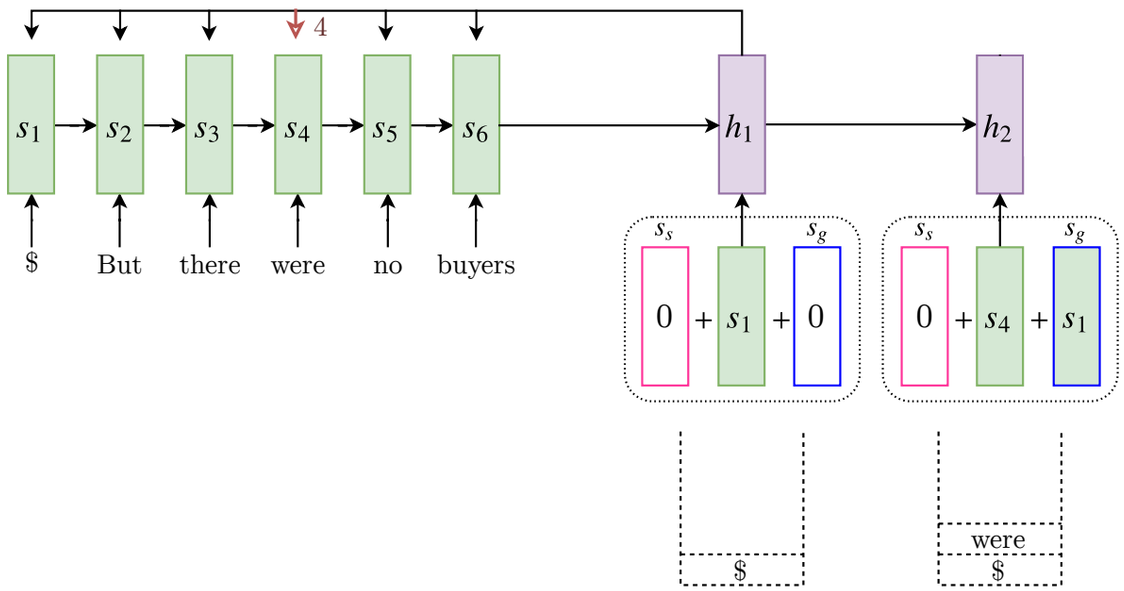
Example


$
But
there
were
no
buyers

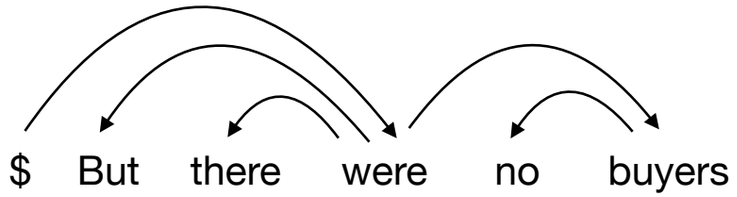

4
s_1
s_2
s_3
s_4
s_5
s_6
h_1
$
But
there
were
no
buyers
s_s
s_g
0
+
s_1
+
0
$
were
$

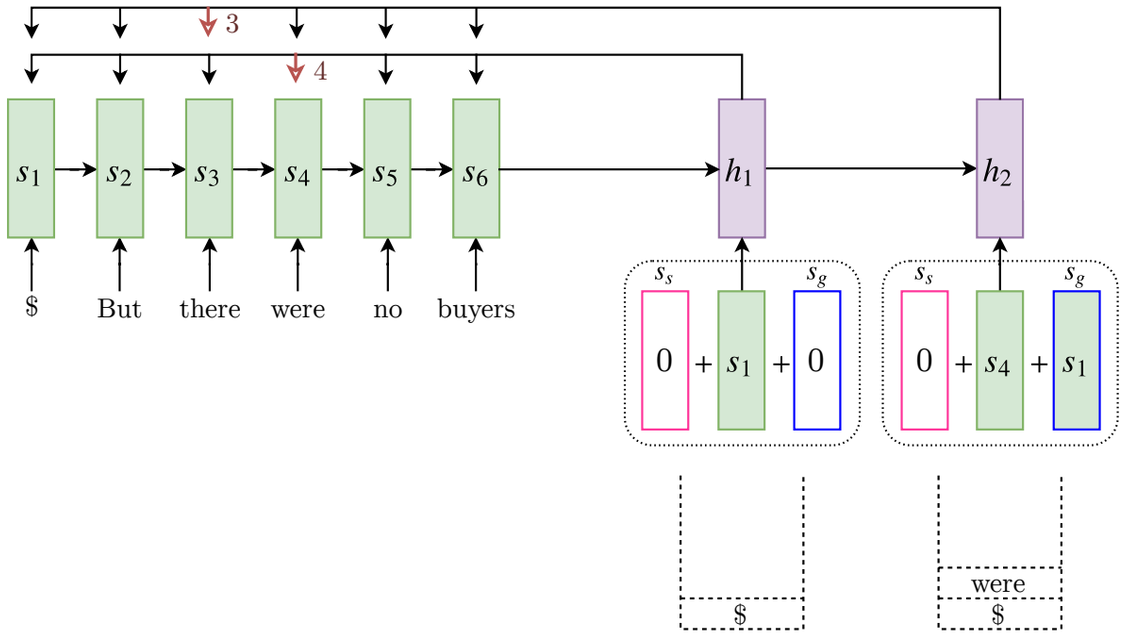
Example


$
But
there
were
no
buyers

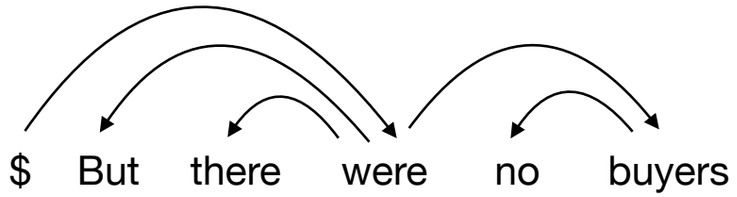

4
s_1
s_2
s_3
s_4
s_5
s_6
h_1
h_2
$
But
there
were
no
buyers
s_s
s_g
0
+
s_1
+
0
s_s
s_g
0
+
s_4
+
s_1
were
$

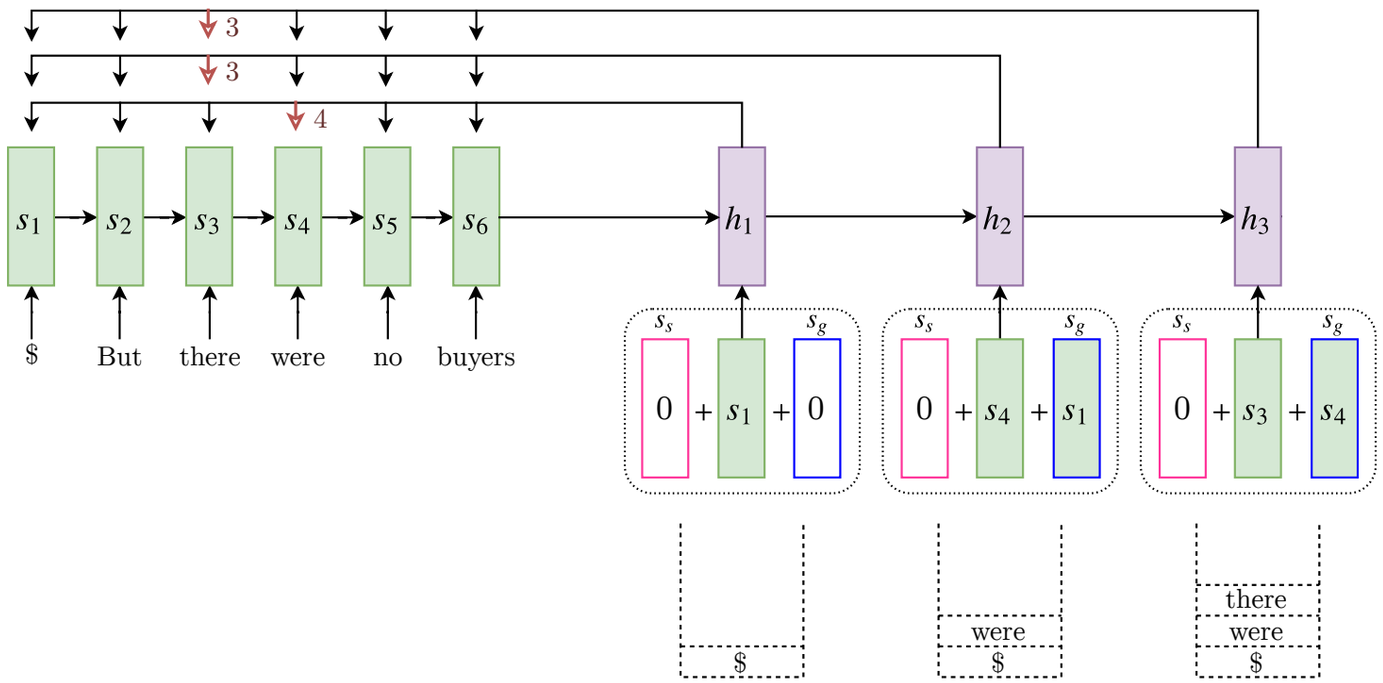
Example


$
But
there
were
no
buyers

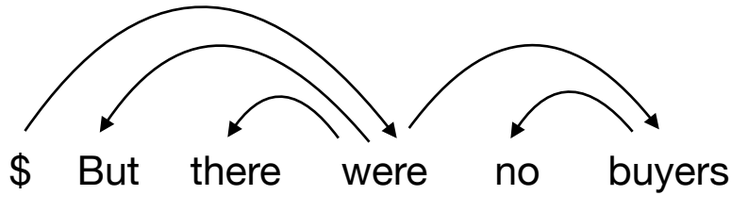

3
4
s_1
s_2
s_3
s_4
s_5
s_6
h_1
h_2
$
But
there
were
no
buyers
s_s
s_g
0
+
s_1
+
0
s_s
s_g
0
+
s_4
+
s_1
were
$

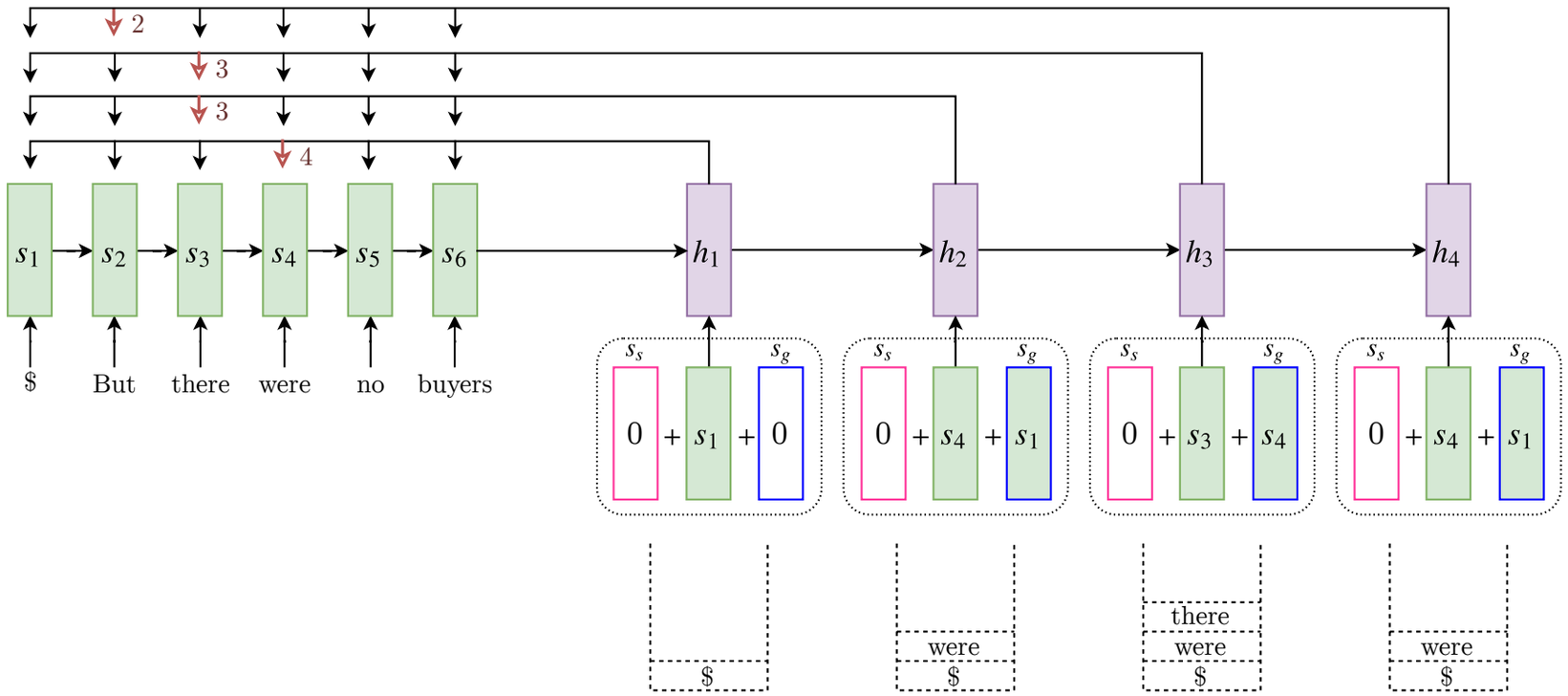
Example


$
But
there
were
no
buyers

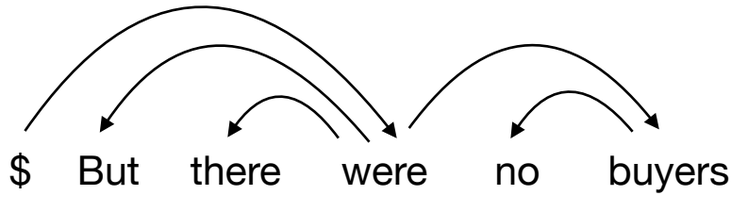

3
3
4
s_1
s_2
s_3
s_4
s_5
s_6
h_1
h_2
h_3
$
But
there
were
no
buyers
s_s
s_g
0 + s_1 + 0
0 + s_4 + s_1
0 + s_3 + s_4
$
were
$
there
were
$

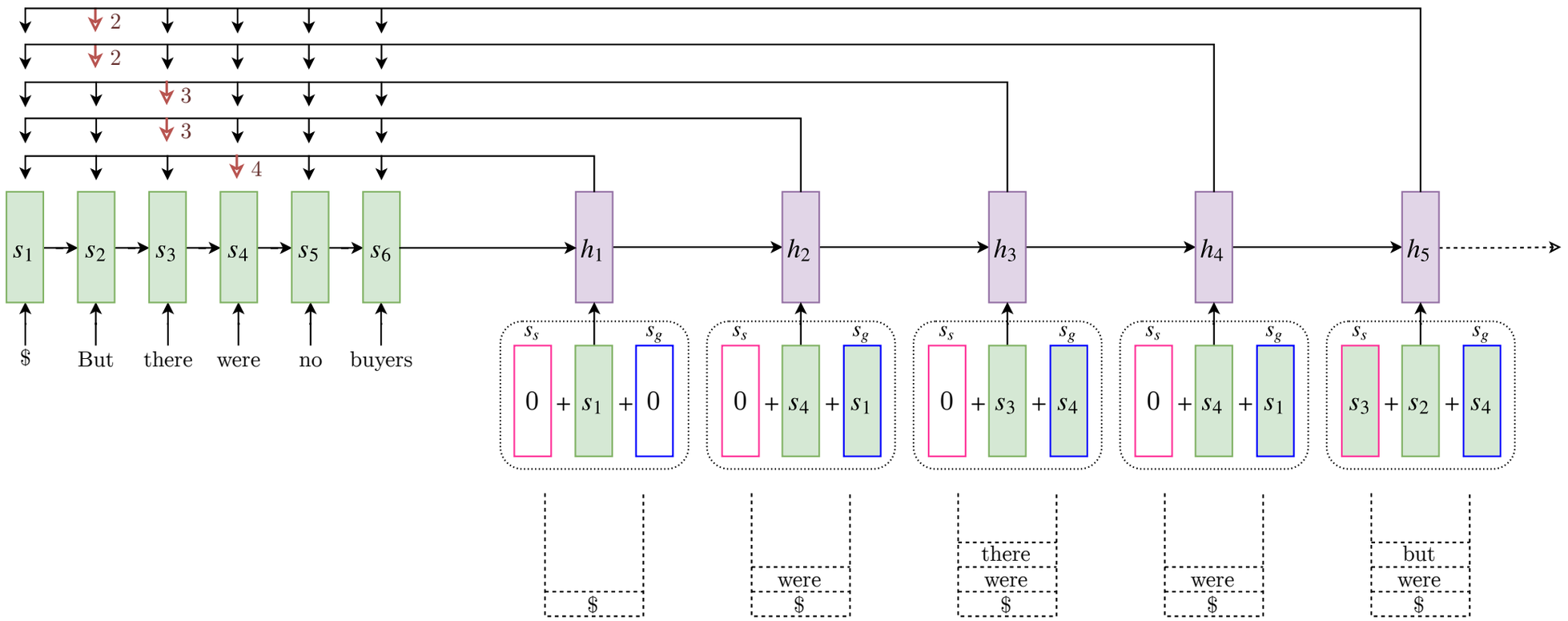
Example


$
But
there
were
no
buyers


2
3
3
4
s_1
s_2
s_3
s_4
s_5
s_6
h_1
h_2
h_3
h_4
$
But
there
were
no
buyers
s_s
s_g
0 + s_1 + 0
0 + s_4 + s_1
0 + s_3 + s_4
0 + s_4 + s_1
$
were
$
there
were
$
were
$

Example


$
But
there
were
no
buyers


2
2
3
3
4
$
But
there
were
no
buyers
$
there
were
$
but
were
$

Learning StackPtr

- **Maximum likelihood**
- **Factorize into sequence of top-down paths**

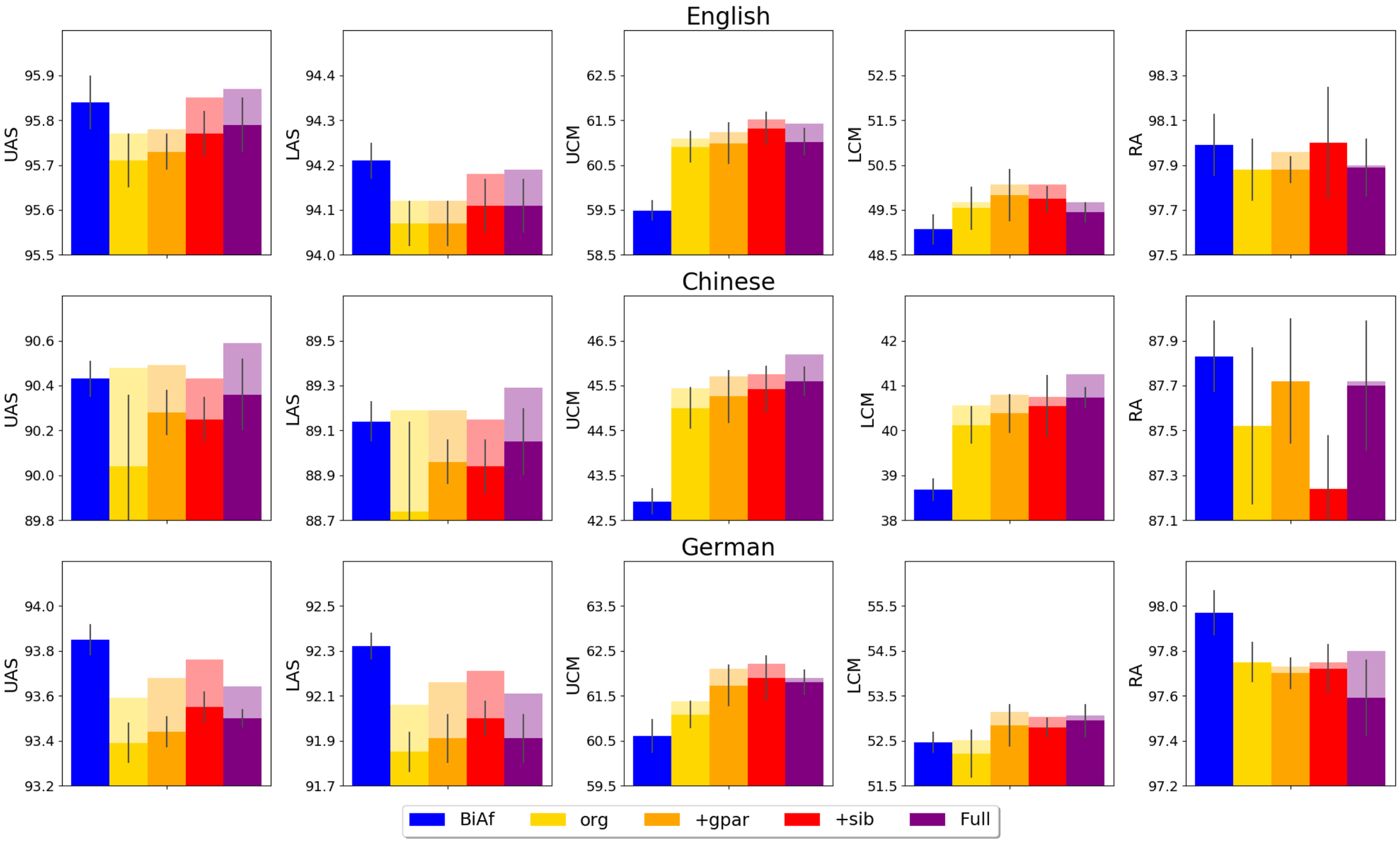
$$
\begin{aligned}
P_\theta(\mathbf{y}|\mathbf{x}) &= \prod_{i=1}^{k} P_\theta(p_i|p_{<i}, \mathbf{x}) \\
&= \prod_{i=1}^{k} \prod_{j=1}^{l_i} P_\theta(c_{i,j}|c_{i,<j}, p_{<i}, \mathbf{x}),
\end{aligned}
$$

- **Pre-defined inside-out order** for children of each head word
  - Enables parser to utilize **higher-order sibling** information
- Train separate classifier for **dependency label prediction**
  - Use head word and child information [Dozat+ 2017]

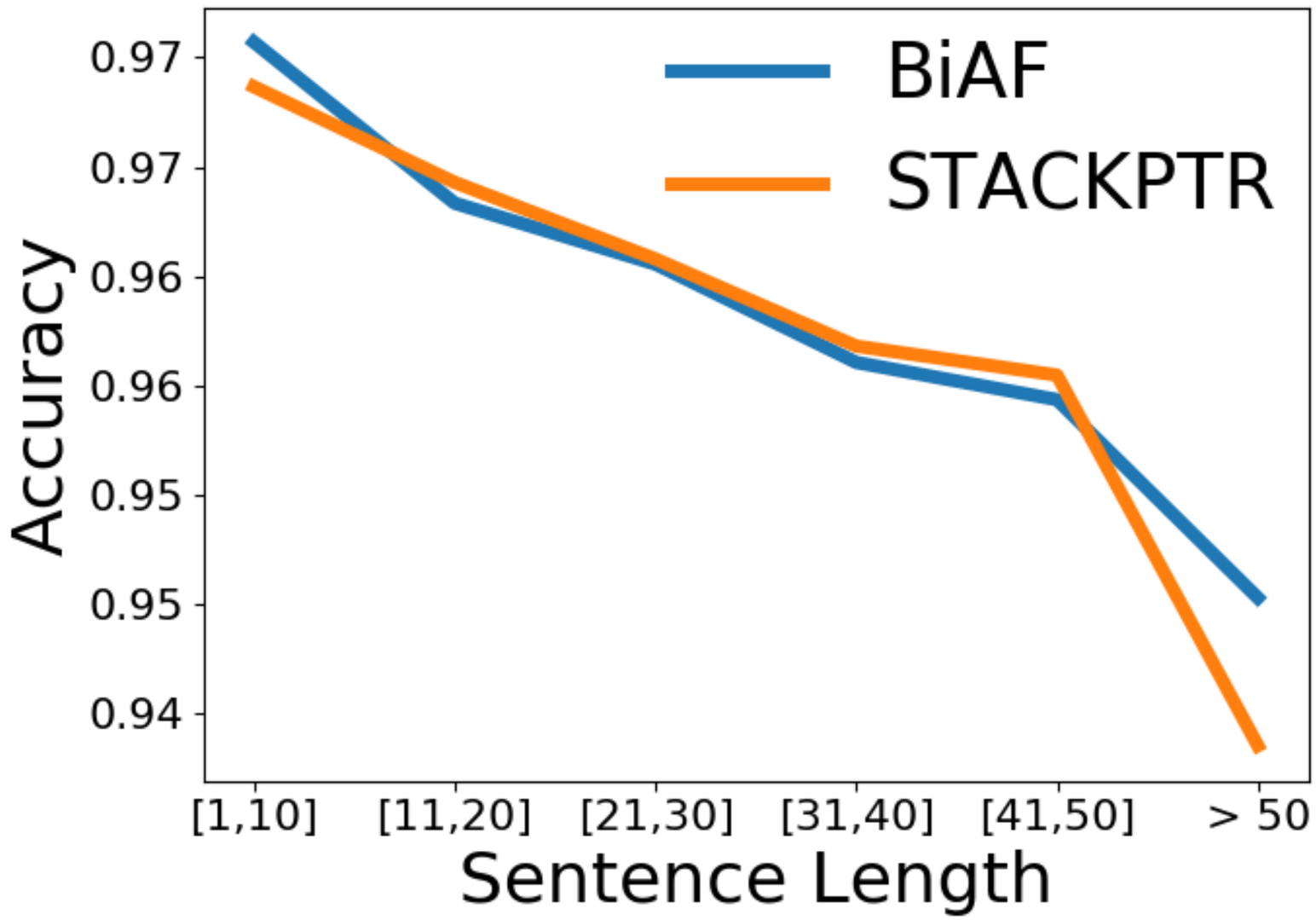
Experiment 1: Main Results & Analysis

- **Datasets:**
  - English PTB, Chinese PTB, German CoNLL 2009 shared task
- **Parsing models for comparison**
  - Baseline: Deep Biaffine (**BiAF**) parser (Dozat et al., 2017), augmented with character-level information
  - Four versions of StackPtr:
    - **Org**: utilizes only head word information
    - **+gpar**: augment Org with grandparent information
    - **+sib**: augment Org with sibling information
    - **Full**: include all the three information
- **Evaluation metrics**
  - Unlabeled Attachment Score (UAS), Labeled Attachment Score (LAS), Unlabeled Complete Match (UCM), Labeled Complete Match (LCM), Root Accuracy (RA)

Main Results

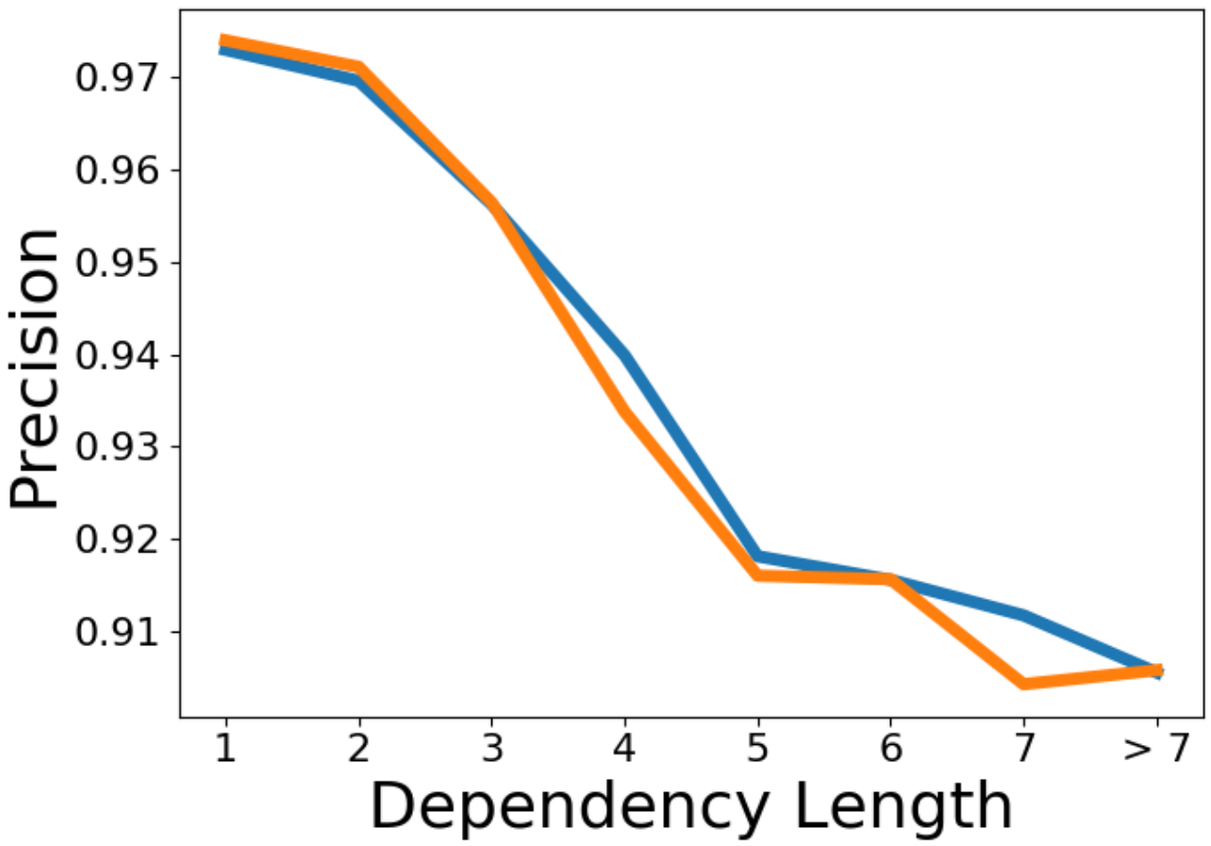


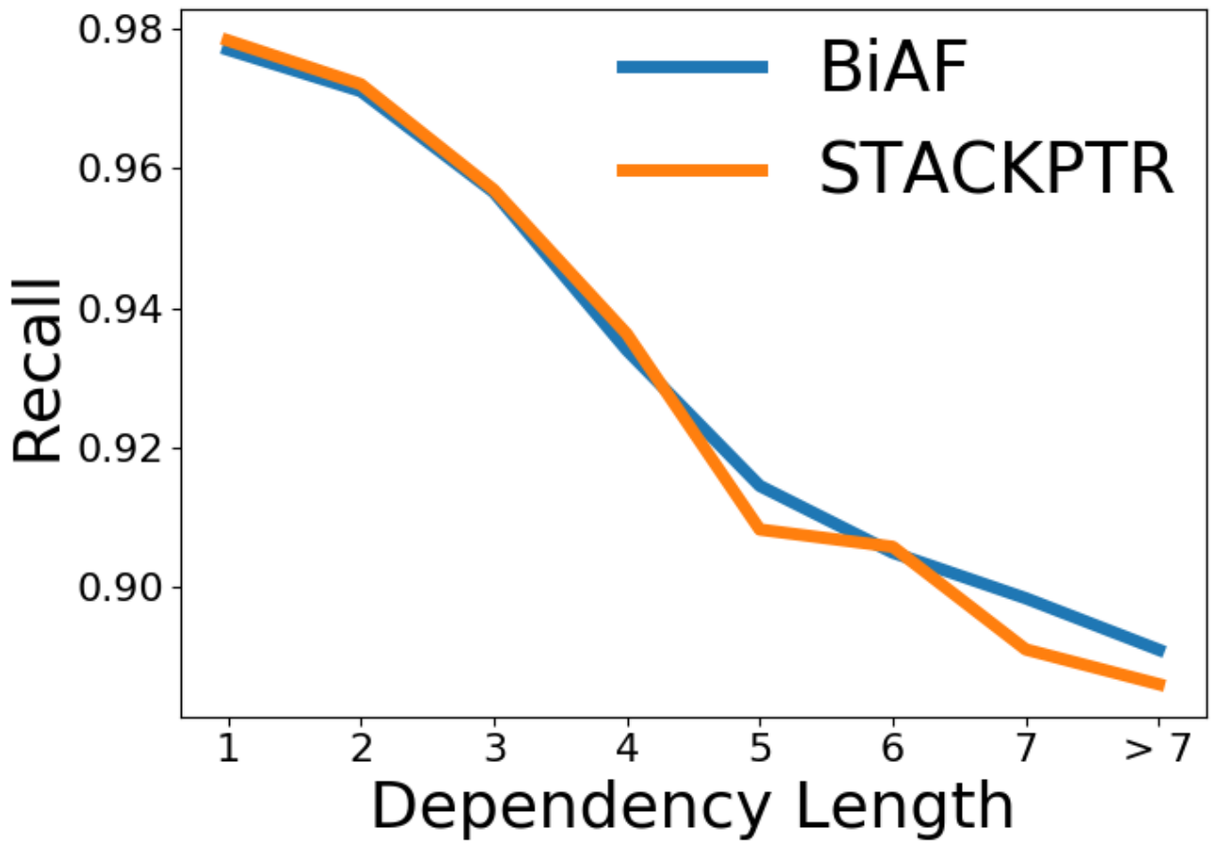
Parsing Performance on Test Data w.r.t Sentence Length



StackPtr tends to perform better on shorter sentences, consistent with transition-based/graph-based comparison in McDonald and Nivre (2011)

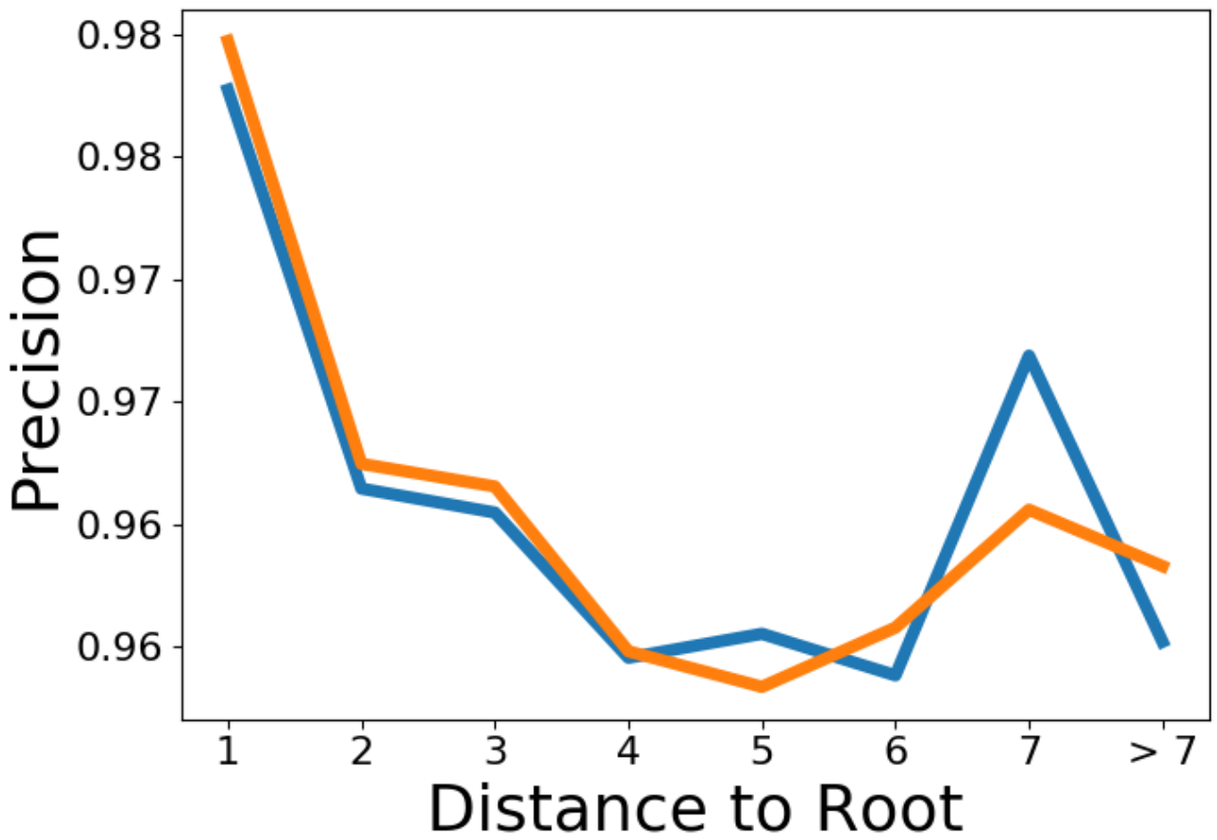
Parsing Performance w.r.t Dependency Length

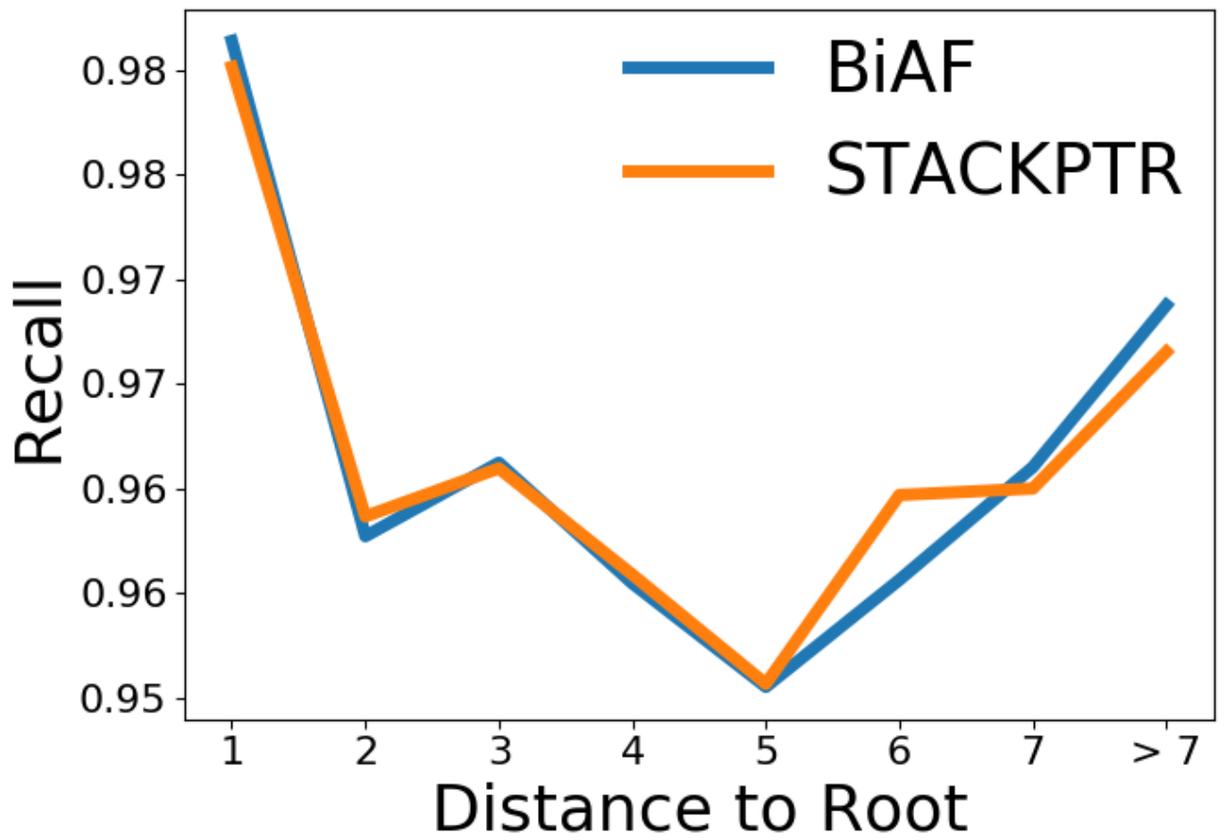




The gap between Stack-Ptr and BiAF is marginal, graph-based BiAF still performs better for longer arcs

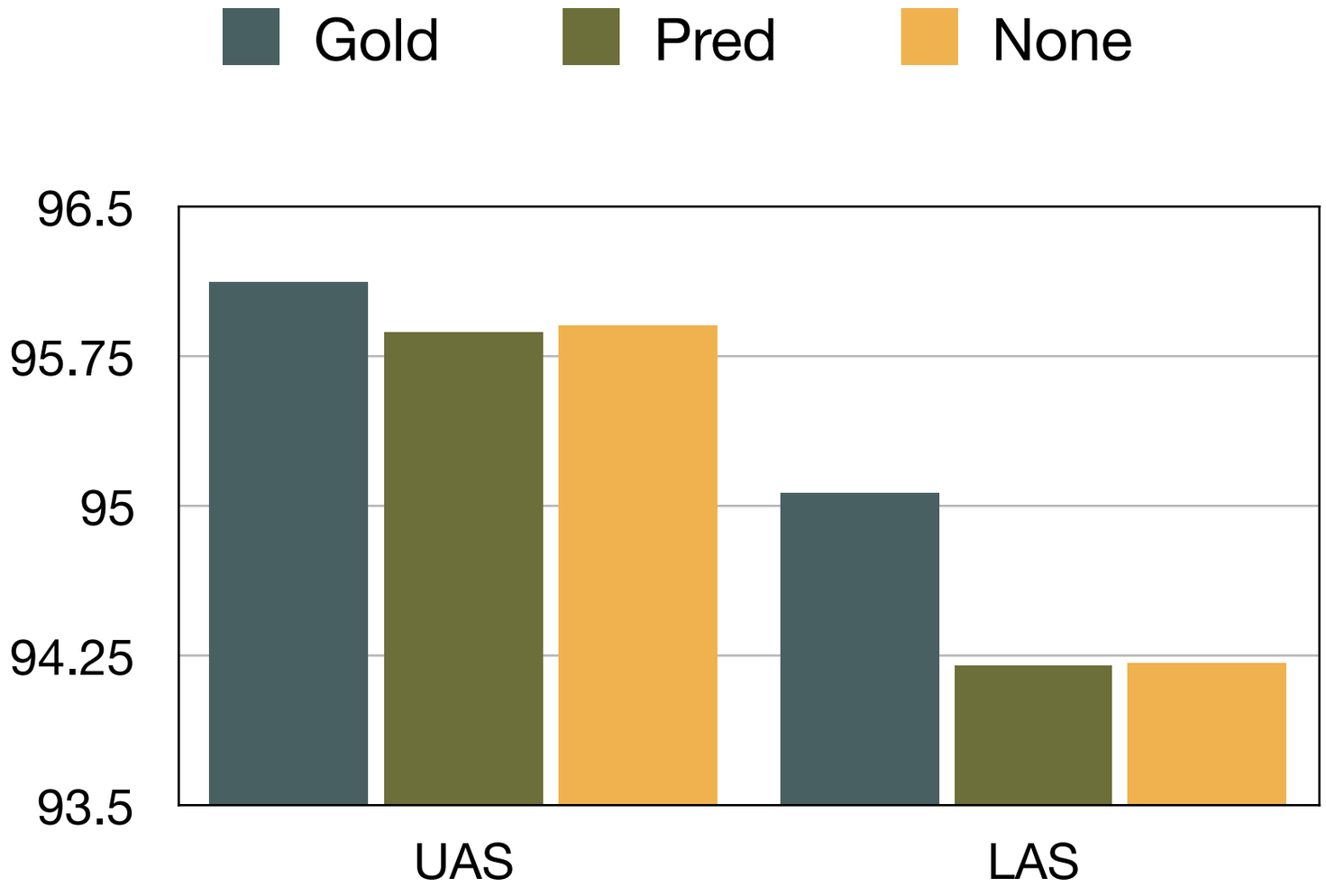
Parsing Performance w.r.t Root Distance





Different from McDonald and Nivre (2011), StackPtr and BiAf similar regardless of root distance

Effect of POS Embedding



Gold: Parser with gold-standard POS tags
Pred: Parser with predicted POS tags (97.3% accuracy)
None: Parser without POS tags

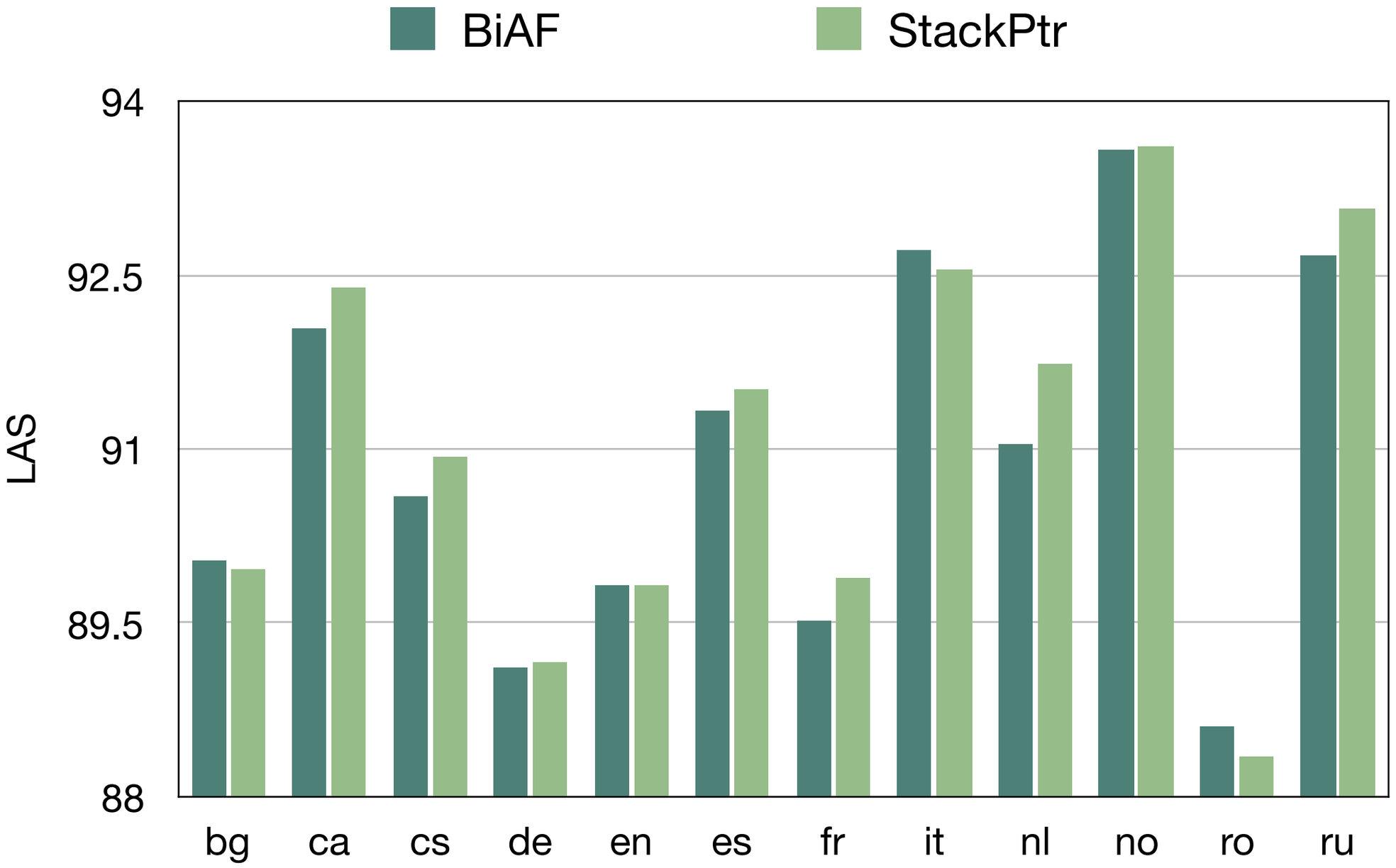
Experiment 2: Universal Dependency Treebanks

- **Datasets:**
  - Universal Dependency Treebanks (V2.2)
  - 12 languages

- **Languages:** Bulgarian, Catalan, Czech, Dutch, English, French, German, Italian, Norwegian, Romanian, Russian and Spanish

- **Note:** we also ran experiments on 14 CoNLL Treebanks. (see the paper for details)

LAS on UD Treebanks



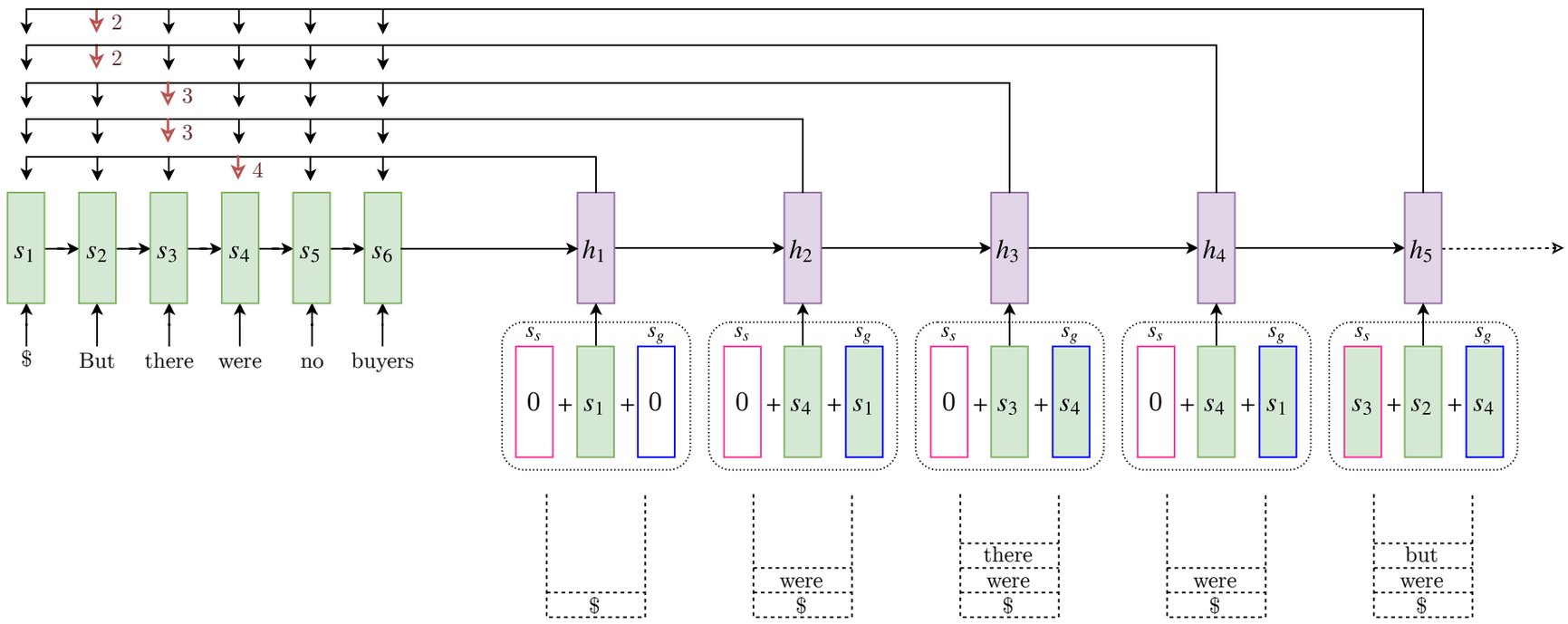
Conclusion & Future Work

- **Stack-Pointer network for dependency parsing**
  - A transition-based neural network architecture
  - Top-down, depth-first decoding procedure
  - State-of-the-art performance on 21 out of 29 treebanks

- **Future Work**
  - Learn an optimal order for the children of head words, instead of using a pre-defined fixed order
  - End-to-end training

Thank you!

Questions?



Our code is published at:
https://github.com/XuezheMax/NeuroNLP2

Language Technologies Institute

Model Details

- **Encoder**
  - Bi-directional LSTM-CNN (Chiu and Nichols 2016; Ma and Hovy 2016)
  - Three input embeddings: word, character and POS
  - CNN encodes character-level information
  - 3-layer LSTM with recurrent dropout (Gal et al., 2016)
- Decoder
  - Uni-directional LSTM
  - Use encoder hidden states as input instead of word embeddings
  - 1-layer LSTM with recurrent dropout