Dependency Parsing

Lecture 13

Kilian Evang Jakub Waszczuk

Heinrich Heine University Düsseldorf

Summer semester 2021

Today

- Previous homework
- Dependency Parsing in the Neural Age
  - Quick Revision
  - Slides accompanying Eliyahu Kiperwasser and Yoav Goldberg's 2016 paper: *Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations*

Data-driven Dependency Parsing: Methods

Transition-based

- Projective: Arc-Standard, Arc-Eager, Arc-Hybrid...
- Non-projective: Swap transitions, Pseudo-projective parsing...

Graph-based

- Projective: Chart-based (Eisner's algorithm)
- Non-projective: Maximum Spanning Tree (Chu-Liu-Edmonds algorithm)

Key problem with all these: *Feature extraction*

Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations

Eliyahu Kiperwasser
Yoav Goldberg

Bar-Ilan University

Contribution

I will show that BiLSTMs are great feature extractors, because:

- Global considerations (in local scoring)
- Trainable (for whatever task)
- Plug&Play (two different parsers)

Main Results

In this talk:

- We apply BiLSTM feature extraction for both greedy and global inference parsing
- We achieve very high parsing scores
- using a very efficient algorithm (93.9 UAS, 800 words/sec, CPU, Python)
- Using this frustratingly simple feature extraction

Parsing Background

There are two main frameworks for parsing:

- Graph-based:
  - Global inference
  - Score factorized over parts
  - There are first, second & third order parsers.
- Transition-based:
  - Greedy local inference
  - Score relies on current configuration, which is dependent on all previous transitions

Parsing Background

There are two main frameworks for parsing:

- Graph-based:
  - Global inference
  - Score factorized over parts
  - There are first, second & third order parsers.
- Transition-based:
  - Greedy local inference
  - Score relies on current configuration, which is dependent on all previous transitions

Parsing Background

There are two main frameworks for parsing:

- Graph-based:
  - Global inference
  - Score factorized over parts
  - There are first, second & third order parsers.
- Transition-based:
  - Greedy local inference
  - Score relies on current configuration, which is dependent on all previous transitions

Graph-based Parsing

$$Score\Big(\ \overset{\text{nsubj}}{\frown}\ \overset{\text{dobj}}{\frown}\ \overset{\text{det}}{\frown}\ \overset{\text{prep}}{\frown}\ \overset{\text{nsubj}}{\frown}\ \Big) =$$

They ate the pizza with anchovies

$$Score\Big(\ \overset{}{\frown}\ \Big) + Score\Big(\ \overset{}{\frown}\ \Big) + Score\Big(\ \overset{}{\frown}\ \Big) +$$

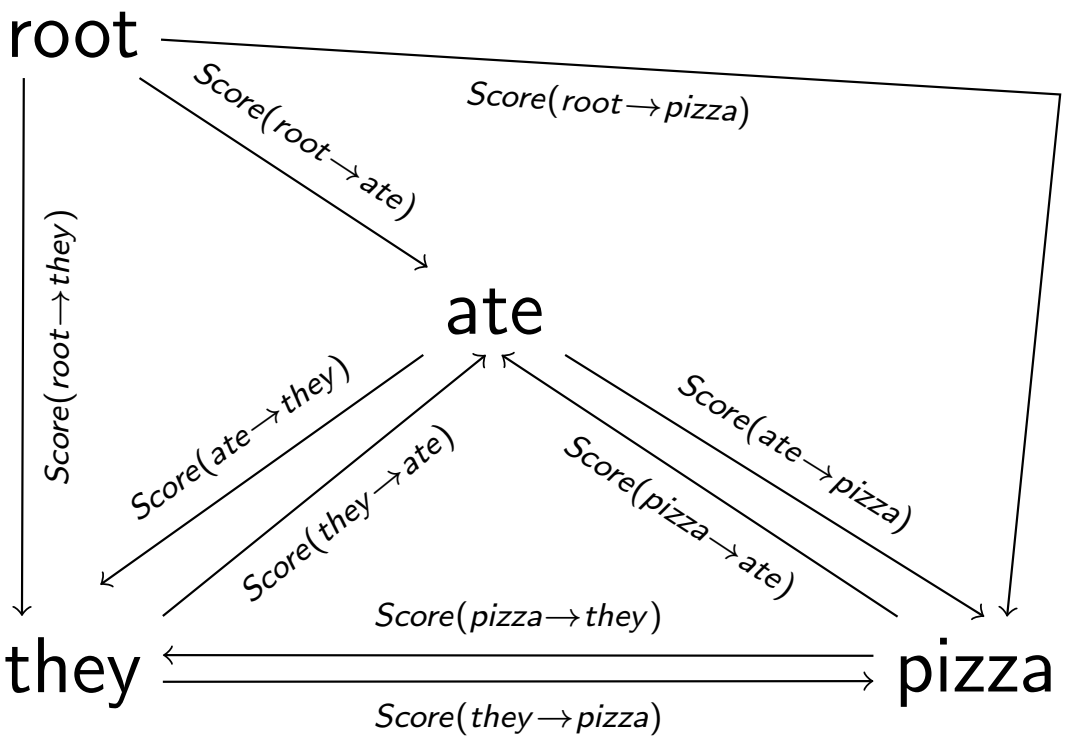
They ate ate pizza the pizza

$$Score\Big(\ \overset{}{\frown}\ \Big) + Score\Big(\ \overset{}{\frown}\ \Big)$$

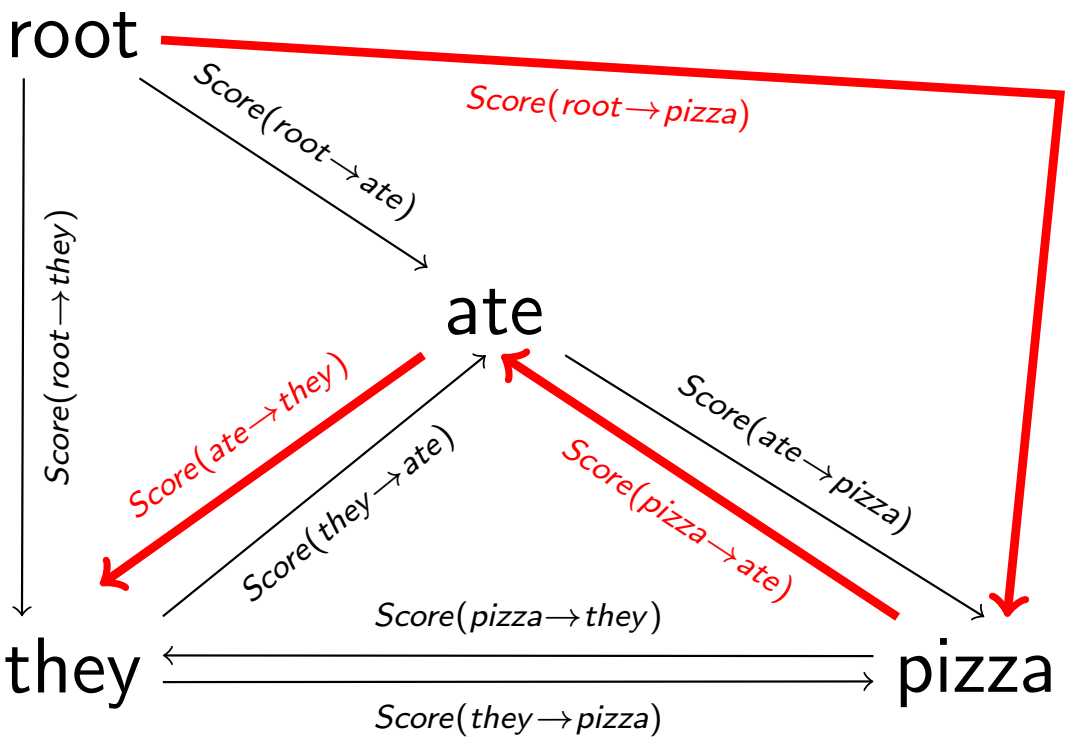
pizza with with anchovies

Graph-based Parsing (Inference)

Input Sentence: "They ate pizza"

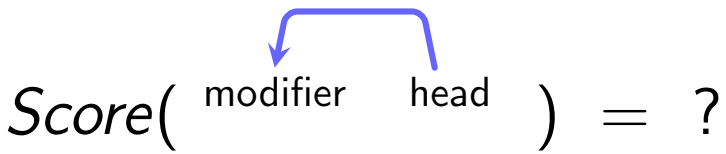


Graph-based Parsing (Inference)



Spanning tree with maximal score

Arc Score Function



Arc Score Function

$$Score(\ \overset{\text{modifier}}{\underset{\text{head}}{\curvearrowleft}}\ ) = F(\ \phi(modifier, head; sentence)\ )$$

- Similar story for transition-based parser
- **The choice of features is very important**

Arc Score Function

$$Score(\ \overset{\text{modifier}}{\underset{\text{head}}{\curvearrowleft}}\ ) = F(\ \phi(modifier, head; sentence)\ )$$

- Similar story for transition-based parser
- **The choice of features is very important**

Arc Score Function

$$Score(\ \overset{\text{modifier}}{\underset{\text{head}}{\curvearrowleft}}\ ) = F(\ \phi(modifier, head; sentence)\ )$$

- Similar story for transition-based parser
- The choice of features is very important

Arc Score Function

$$Score(\ \overset{\text{modifier}}{\underset{\text{head}}{\curvearrowleft}}\ ) = F(\ \phi(modifier, head; sentence)\ )$$

- Similar story for transition-based parser
- **The choice of features is very important**

Let's Talk about Features

(Previous Work)

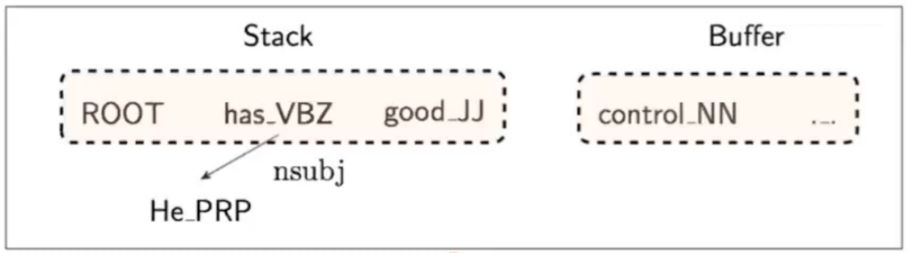
Manual Feature Templates

Core Features +
Feature Combinations

[went]	[VBD]	[As]	[ADP]	[went]
[VERB]	[As]	[IN]	[went, VBD]	[As, ADP]
[went, As]	[VBD, ADP]	[went, VERB]	[As, IN]	[went, As]
[VERB, IN]	[VBD, As, ADP]	[went, As, ADP]	[went, VBD, ADP]	[went, VBD, As]
[ADJ, *, ADP]	[VBD, *, ADP]	[VBD, ADJ, ADP]	[VBD, ADJ, *]	[NNS, *, ADP]
[NNS, VBD, ADP]	[NNS, VBD, *]	[ADJ, ADP, NNP]	[VBD, ADP, NNP]	[VBD, ADJ, NNP]
[NNS, ADP, NNP]	[NNS, VBD, NNP]	[went, left, 5]	[VBD, left, 5]	[As, left, 5]
[ADP, left, 5]	[VERB, As, IN]	[went, As, IN]	[went, VERB, IN]	[went, VERB, As]
[JJ, *, IN]	[VERB, *, IN]	[VERB, JJ, IN]	[VERB, JJ, *]	[NOUN, *, IN]
[NOUN, VERB, IN]	[NOUN, VERB, *]	[JJ, IN, NOUN]	[VERB, IN, NOUN]	[VERB, JJ, NOUN]
[NOUN, IN, NOUN]	[NOUN, VERB, NOUN]	[went, left, 5]	[VERB, left, 5]	[As, left, 5]
[IN, left, 5]	[went, VBD, As, ADP]	[VBD, ADJ, *, ADP]	[NNS, VBD, *, ADP]	[VBD, ADJ, ADP, NNP]
[NNS, VBD, ADP, NNP]	[went, VBD, left, 5]	[As, ADP, left, 5]	[went, As, left, 5]	[VBD, ADP, left, 5]
[went, VERB, As, IN]	[VERB, JJ, *, IN]	[NOUN, VERB, *, IN]	[VERB, JJ, IN, NOUN]	[NOUN, VERB, IN, NOUN]
[went, VERB, left, 5]	[As, IN, left, 5]	[went, As, left, 5]	[VERB, IN, left, 5]	[VBD, As, ADP, left, 5]
[went, As, ADP, left, 5]	[went, VBD, ADP, left, 5]	[went, VBD, As, left, 5]	[ADJ, *, ADP, left, 5]	[VBD, *, ADP, left, 5]
[VBD, ADJ, ADP, left, 5]	[VBD, ADJ, *, left, 5]	[NNS, *, ADP, left, 5]	[NNS, VBD, ADP, left, 5]	[NNS, VBD, *, left, 5]
[ADJ, ADP, NNP, left, 5]	[VBD, ADP, NNP, left, 5]	[VBD, ADJ, NNP, left, 5]	[NNS, ADP, NNP, left, 5]	[NNS, VBD, NNP, left, 5]
[VERB, As, IN, left, 5]	[went, As, IN, left, 5]	[went, VERB, IN, left, 5]	[went, VERB, As, left, 5]	[JJ, *, IN, left, 5]
[VERB, *, IN, left, 5]	[VERB, JJ, IN, left, 5]	[VERB, JJ, *, left, 5]	[NOUN, *, IN, left, 5]	[NOUN, VERB, IN, left, 5]

Example from slides of Rush and Petrov (2012)

Conventional Feature Representation



binary, sparse
dim =10^6 ~ 10^7

0	0	0	1	0	0	1	0	...	0	0	1	0

Feature templates: usually a combination of 1 ~ 3 elements from the configuration.

Indicator features

$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$
$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Slide credits: Shira Wein, Christopher Manning

Conventional Feature Representation, cont.

- **Core feature templates:** atomic elements of input/structure built so far, e.g.,
  - "word on top of stack" ($s_1.w$)
  - "POS tag of word on top of stack" ($s_1.t$)
  - "word at the beginning of buffer" ($b_1.w$)
  - "POS tag of left child of second-topmost word on stack" ($lc(s_2).t$)
  - ...
- **Combination feature templates:** combinations of about 1–3 of these, e.g.,
  - $s_1.w \wedge s_1.t$
  - $lc(s_2).t \wedge s_2.t \wedge s_1.t$
  - ...
- **Combination features:** concrete instantiations of feature templates, e.g.,
  - $s_1.w = \text{good} \wedge s_1.t = \text{JJ}$
  - $lc(s_2).t = \text{good} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{good}$
  - ...
- **Feature vector:** long vector (1-10 M dimensions), indicating 0 or 1 for each combination feature

Slide credits: Shira Wein, Christopher Manning

Problems with Conventional Feature Vectors

- Optimal feature set hard to choose
- May differ for each language/domain!
- Incomplete
- Sparse
- Expensive to compute

Solution: Dense Vector Representations

- Real-valued rather than binary
- Dense rather than sparse (few 0's)
- Lower dimensionality (a few 100)
- Computed efficiently by deep neural networks
- Vector dimensions no longer have straightforward interpretations
- With exposure to enough training data, can pick up signals that human feature engineers might have never thought of

Core Features + ~~Feature Combinations~~

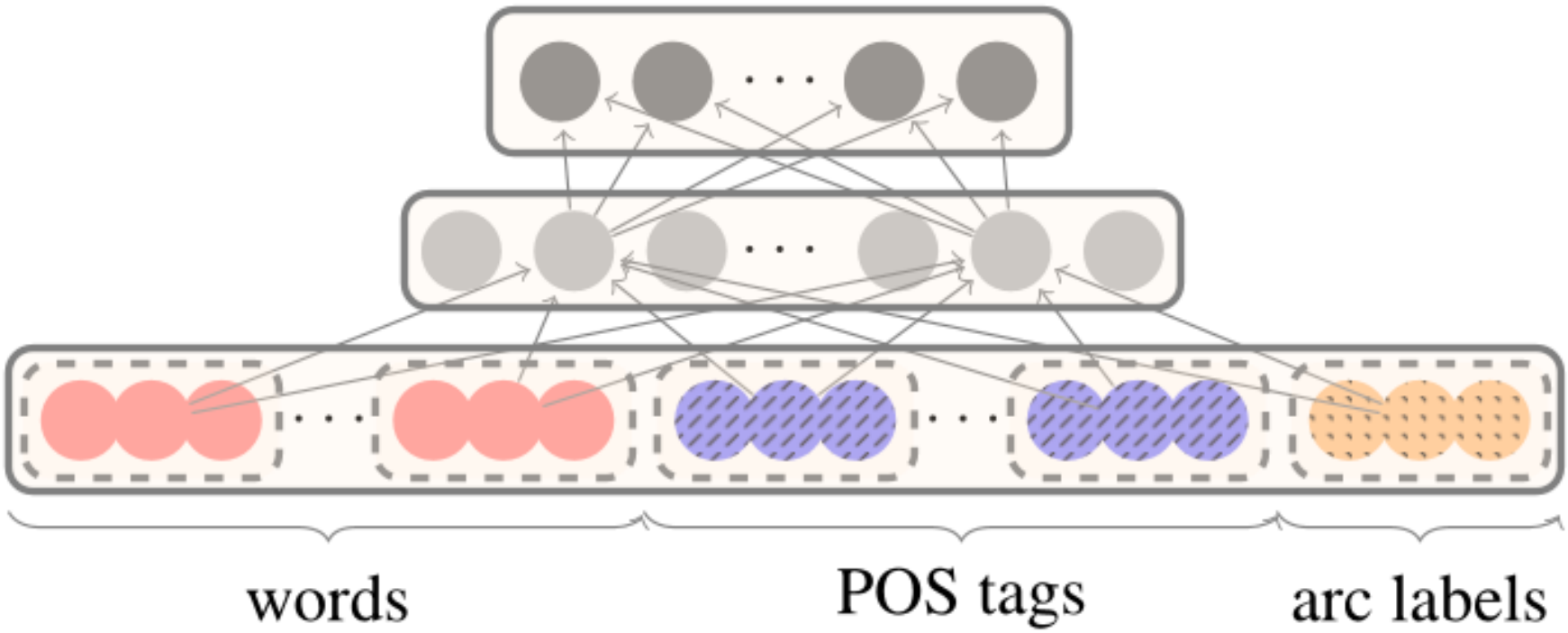


Figure from Chen and Manning (2014)
Similiar approach in Pei et al, Weiss et al, Andor et al

Core Features + Non-Linear Classifier

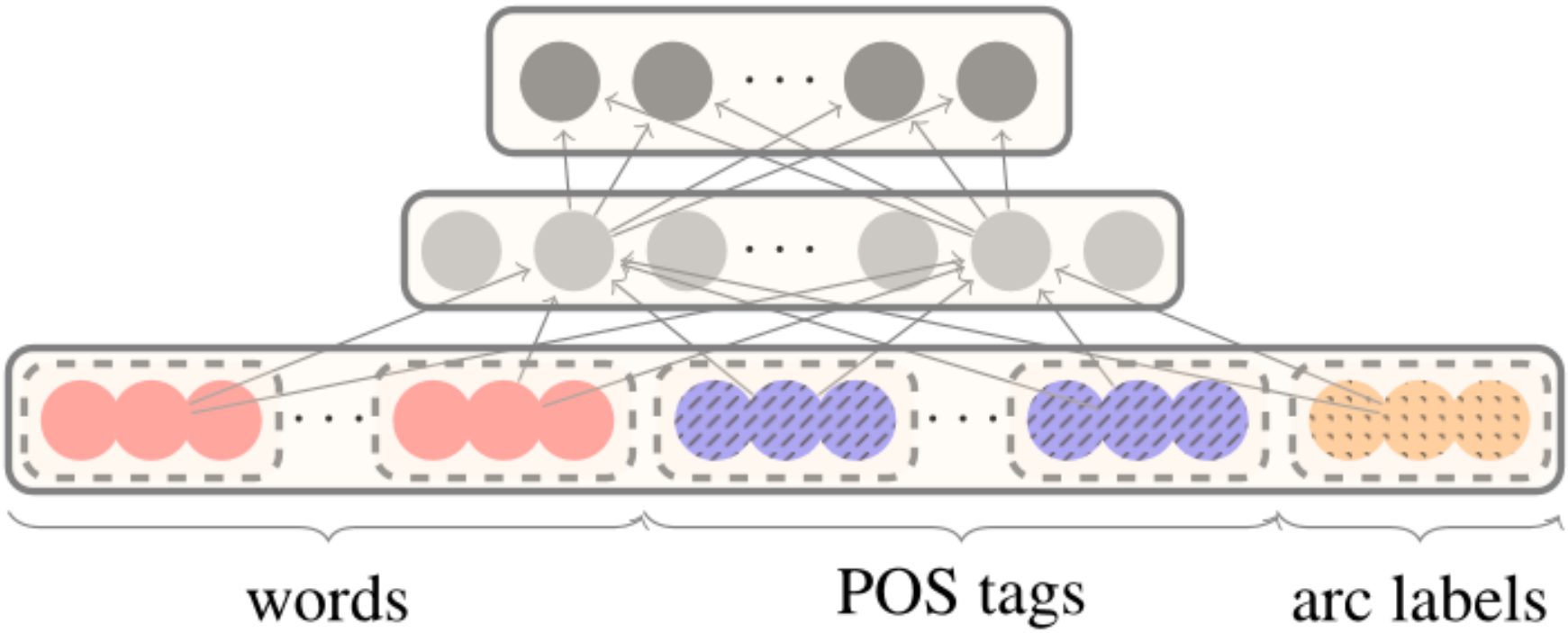
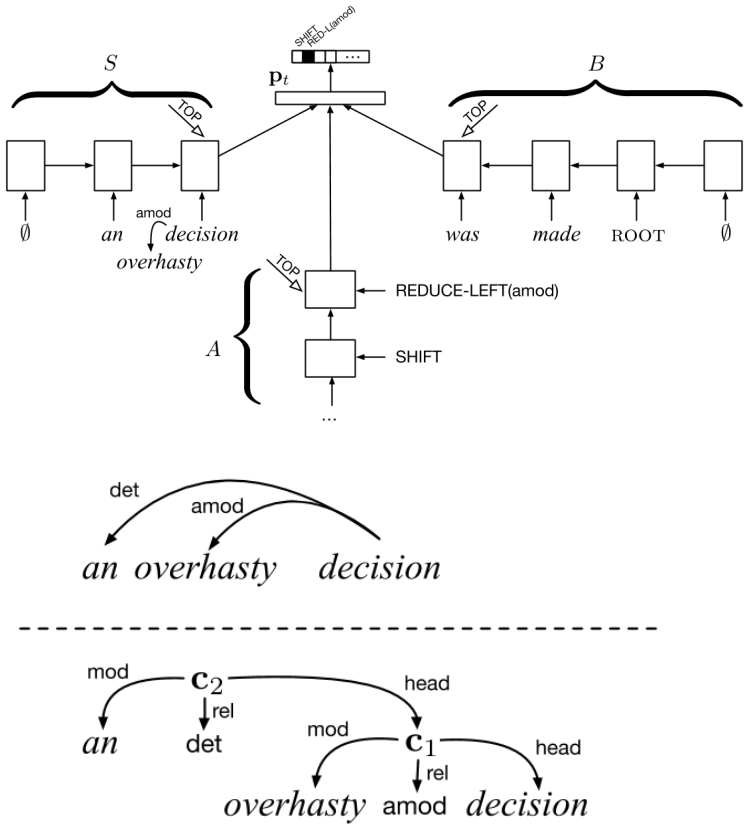


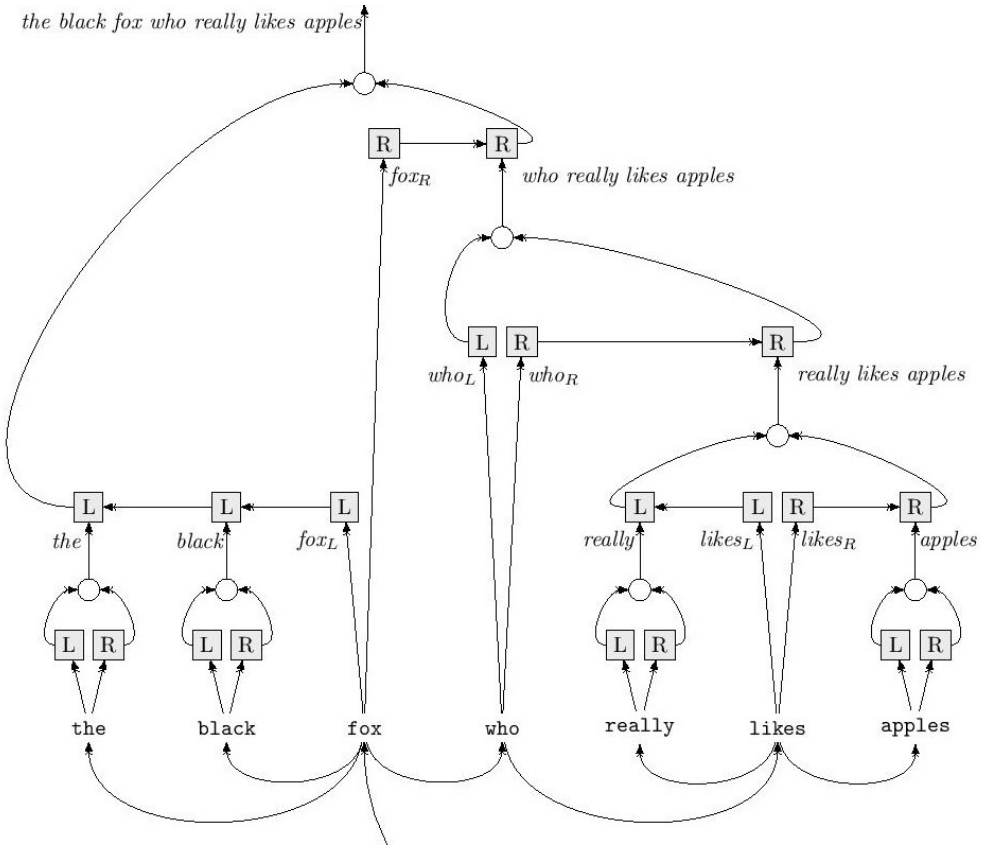
Figure from Chen and Manning (2014)
Similiar approach in Pei et al, Weiss et al, Andor et al

Tailored Neural Architecture

Dyer et al. (2015) [1]



Kiperwasser and Goldberg (2016) [2]



[1] Transition-based Parsing (S-LSTM + Composition)

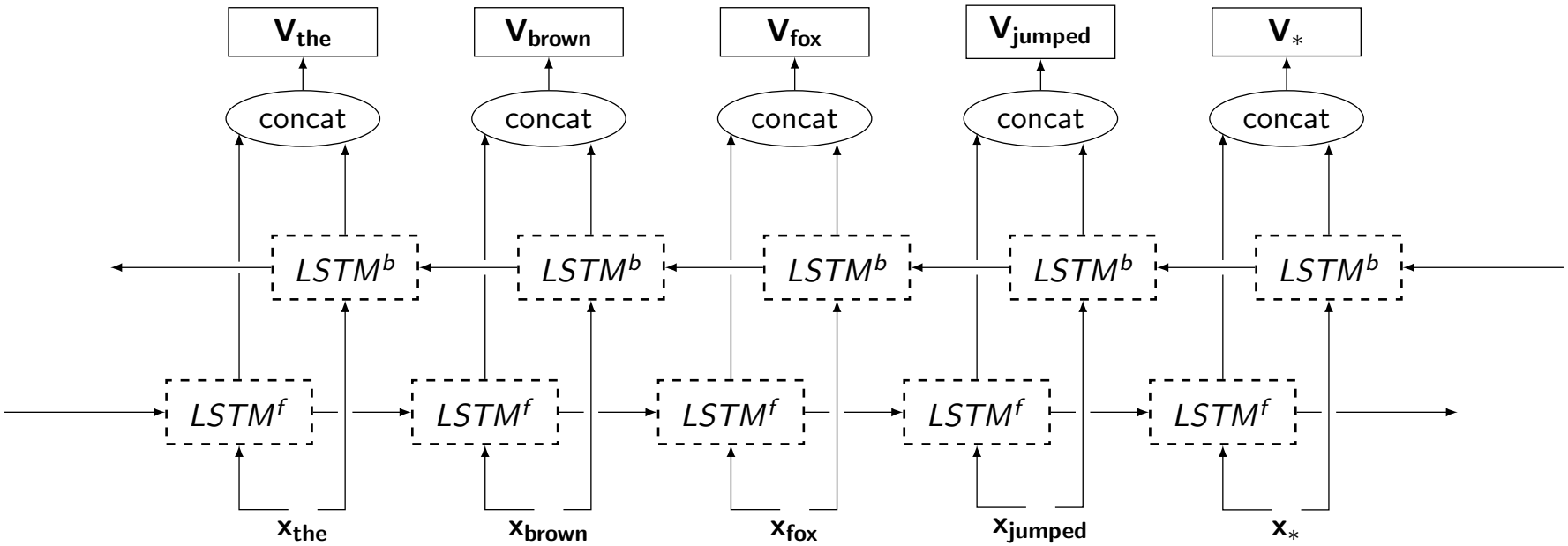
[2] Easy-First Parsing (HT-LSTM)

Our Approach

- We use a simple and elegant BiLSTM encoding as a feature extractor
- Achieving accuracies comparable to state-of-the-art parsers
- While using very simple features definitions

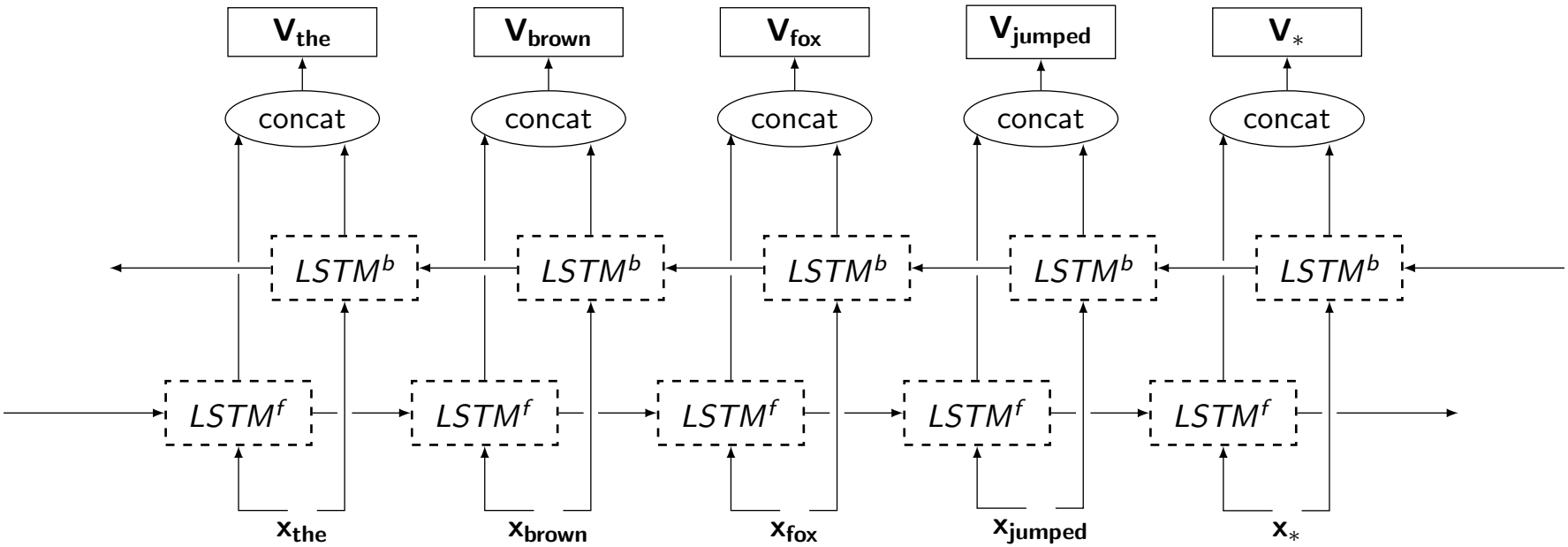
BiLSTM

BiLSTM Word Representation



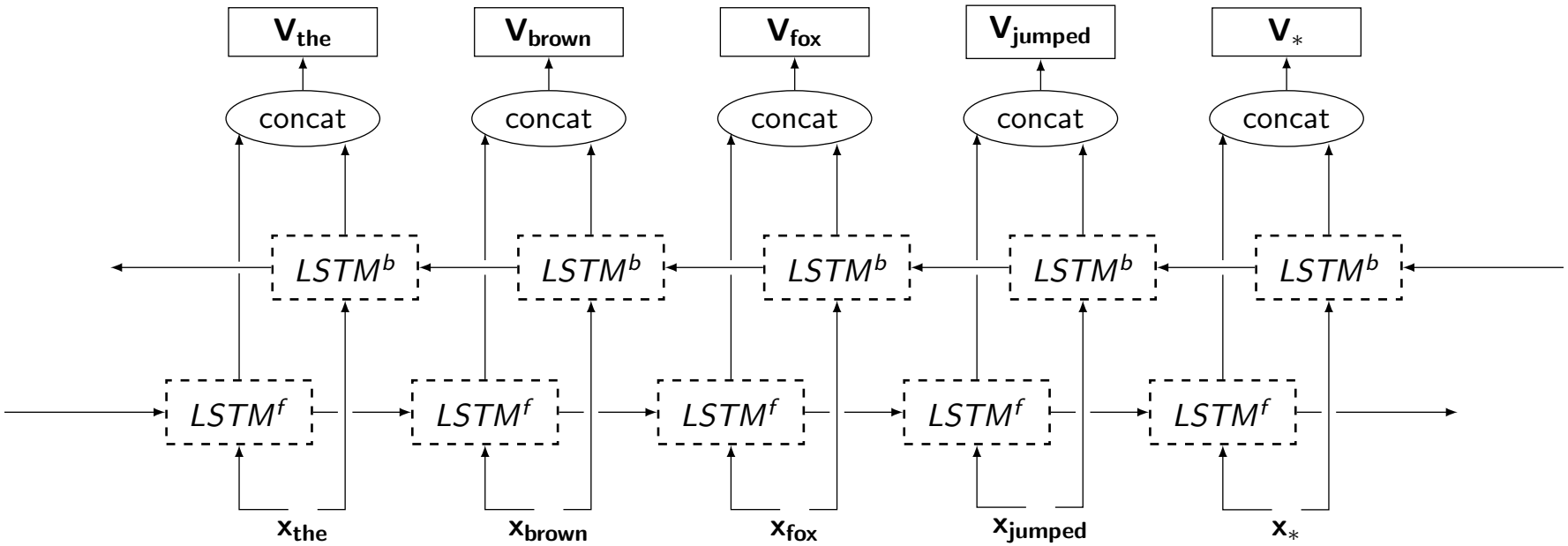
- Encodes an "infinite" window around each word
- Trains jointly with the rest of the network (end-to-end)
- Learn to capture the relevant syntactic information

BiLSTM Word Representation



- Encodes an "infinite" window around each word
- Trains jointly with the rest of the network (end-to-end)
- Learn to capture the relevant syntactic information

BiLSTM Word Representation



- Encodes an "infinite" window around each word
- Trains jointly with the rest of the network (end-to-end)
- Learn to capture the relevant syntactic information

Plug&Play of BiLSTM Features

- We use well known parsing techinques
- We simply replace each word vector with its BiLSTM representation

Plug&Play of BiLSTM Features

- **Graph-based Parsing**:
  - Use the BiLSTM encoding of the *head* and *modifier*
  - Use a multi-layer perceptron to score the attachment

- **Transition-based Parsing**:
  - Use the BiLSTM encoding of the *three words on top of the stack* and the *first word on the buffer*
  - Use a multi-layer perceptron to score the transition

Plug&Play of BiLSTM Features

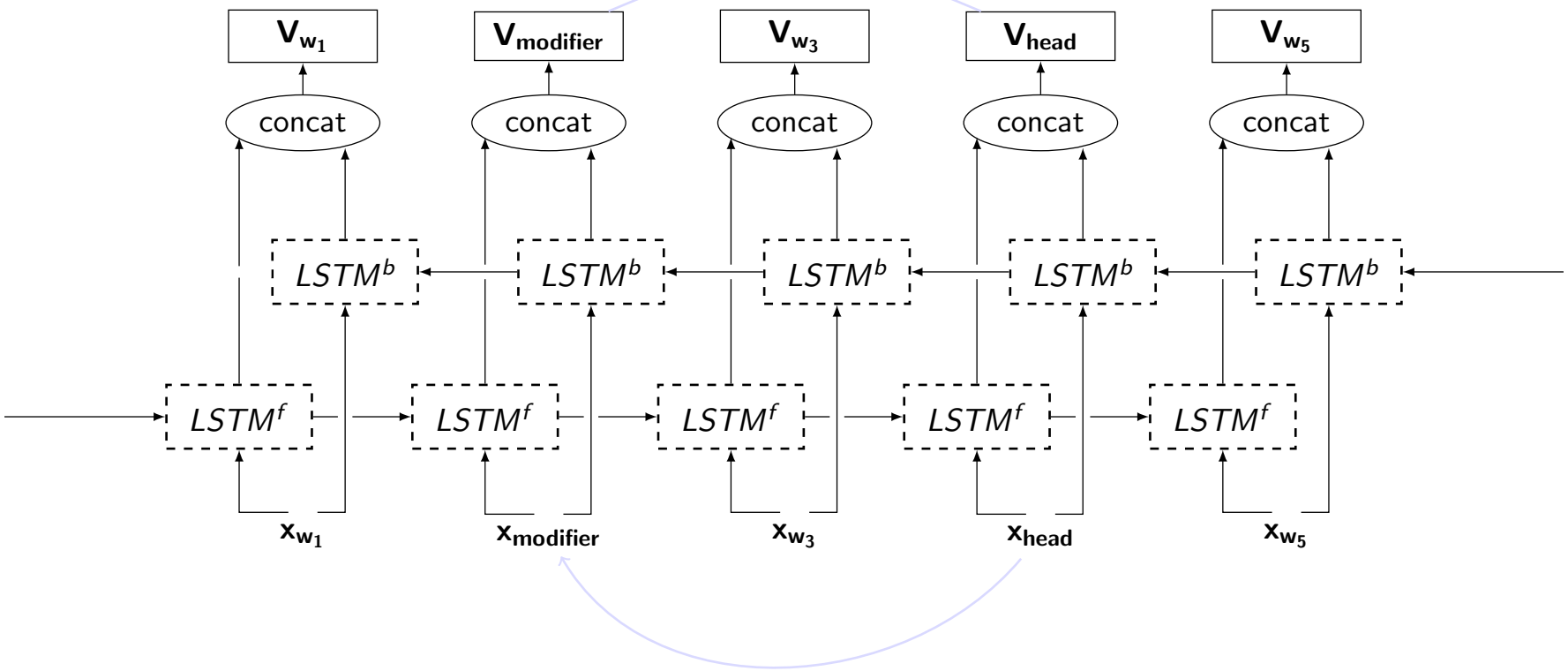
- **Graph-based Parsing**:
  - Use the BiLSTM encoding of the *head* and *modifier*
  - Use a multi-layer perceptron to score the attachment
- **Transition-based Parsing**:
  - Use the BiLSTM encoding of the *three words on top of the stack* and the *first word on the buffer*
  - Use a multi-layer perceptron to score the transition

Plug&Play

Graph-based Parsing

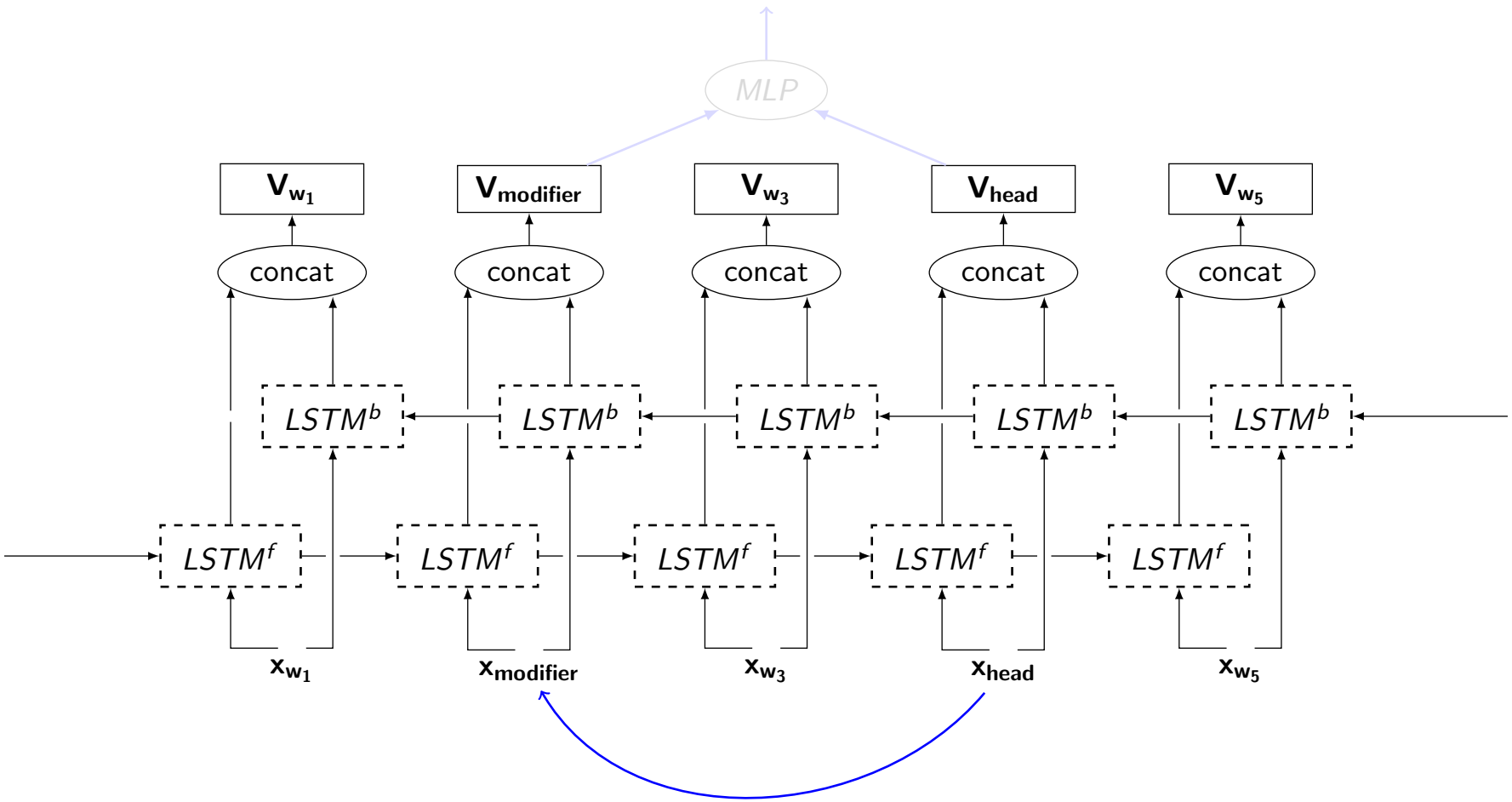
Arc Score

$Score($ modifier head $) = MLP(\phi(head, modifier; sentence))$



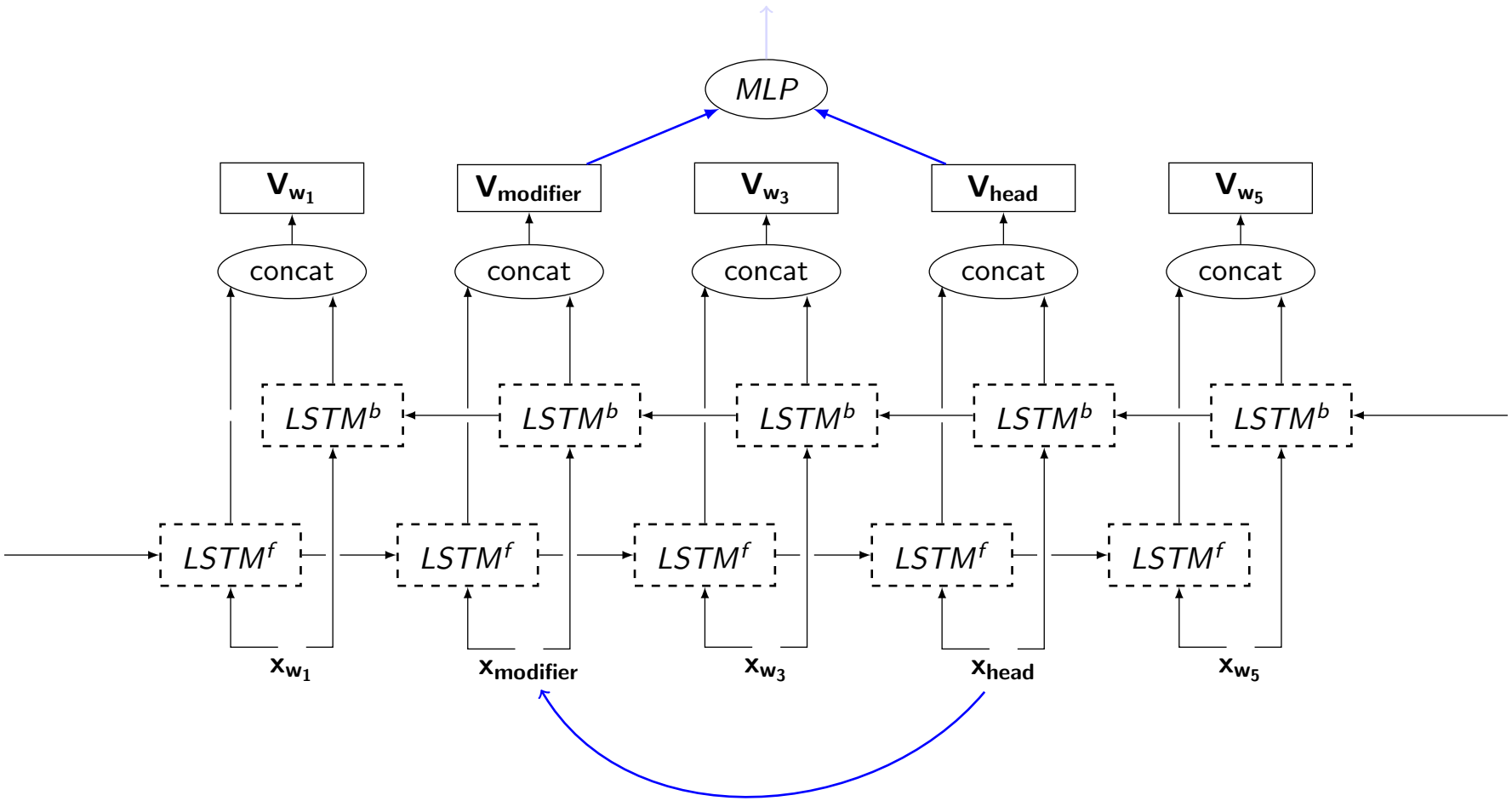
Arc Score

$Score($ modifier head $) =$ $MLP(\phi(head, modifier; sentence))$



Arc Score

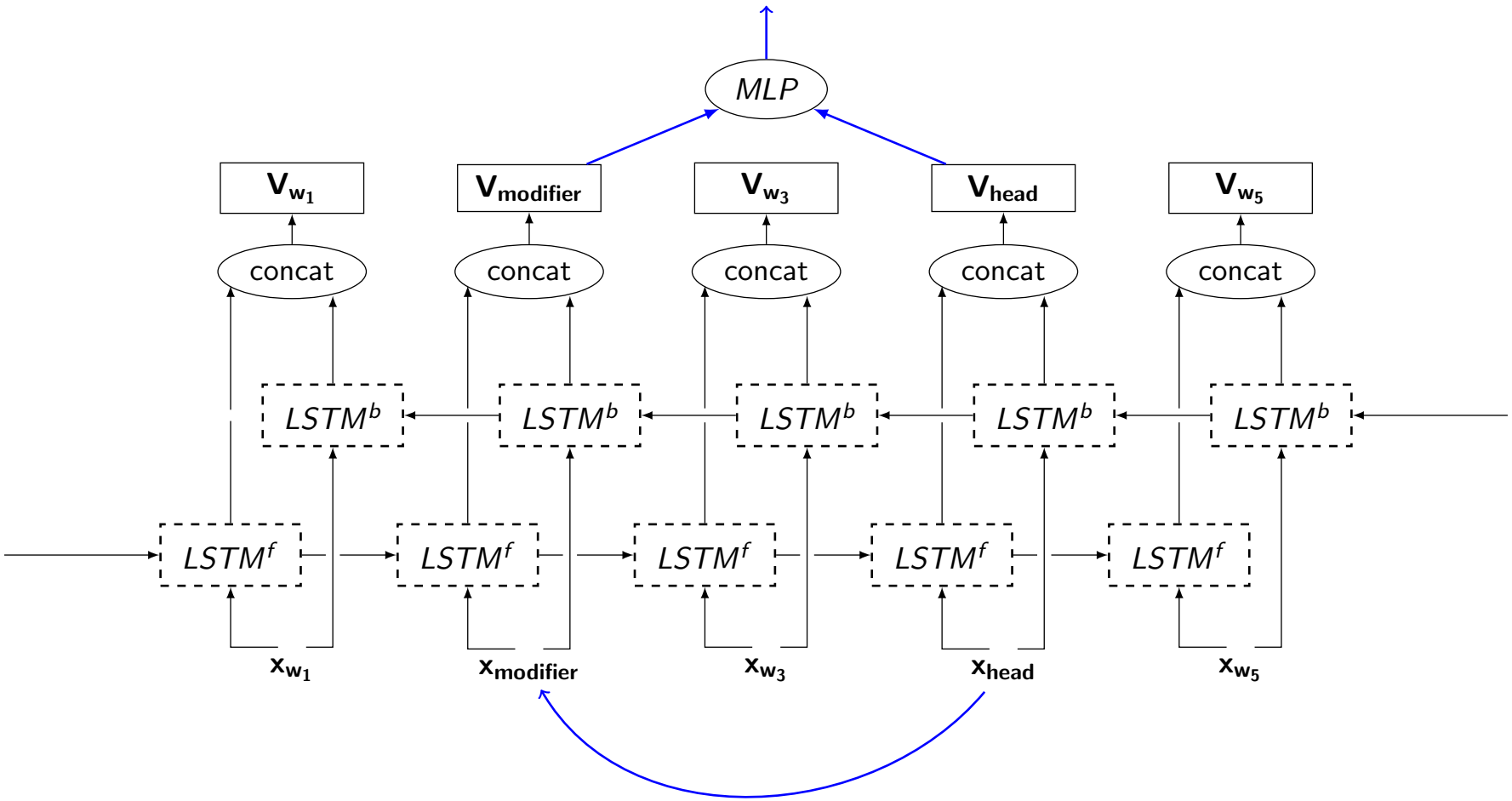
$Score($ modifier head $) =$ $MLP(\phi(head, modifier; sentence))$



Arc Score

$Score($ 

$) = MLP(\phi(head, modifier; sentence))$



Arc Score (Intuition)

- The BiLSTM encoding of a word holds information about its attachment preferences
- The score is dependent on the BiLSTM encoding which in turn depends on the entire sentence
- Therefore, the score function focused on a specific arc is considering also the entire sentence attachement preferences

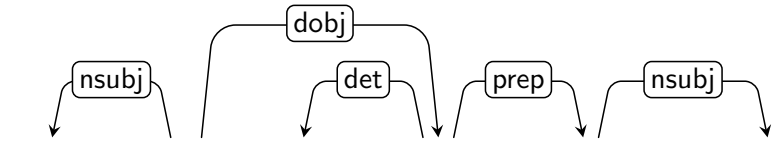
Arc Score (Intuition)

- The BiLSTM encoding of a word holds information about its attachment preferences
- The score is dependent on the BiLSTM encoding which in turn depends on the entire sentence
- Therefore, the score function focused on a specific arc is considering also the entire sentence attachement preferences

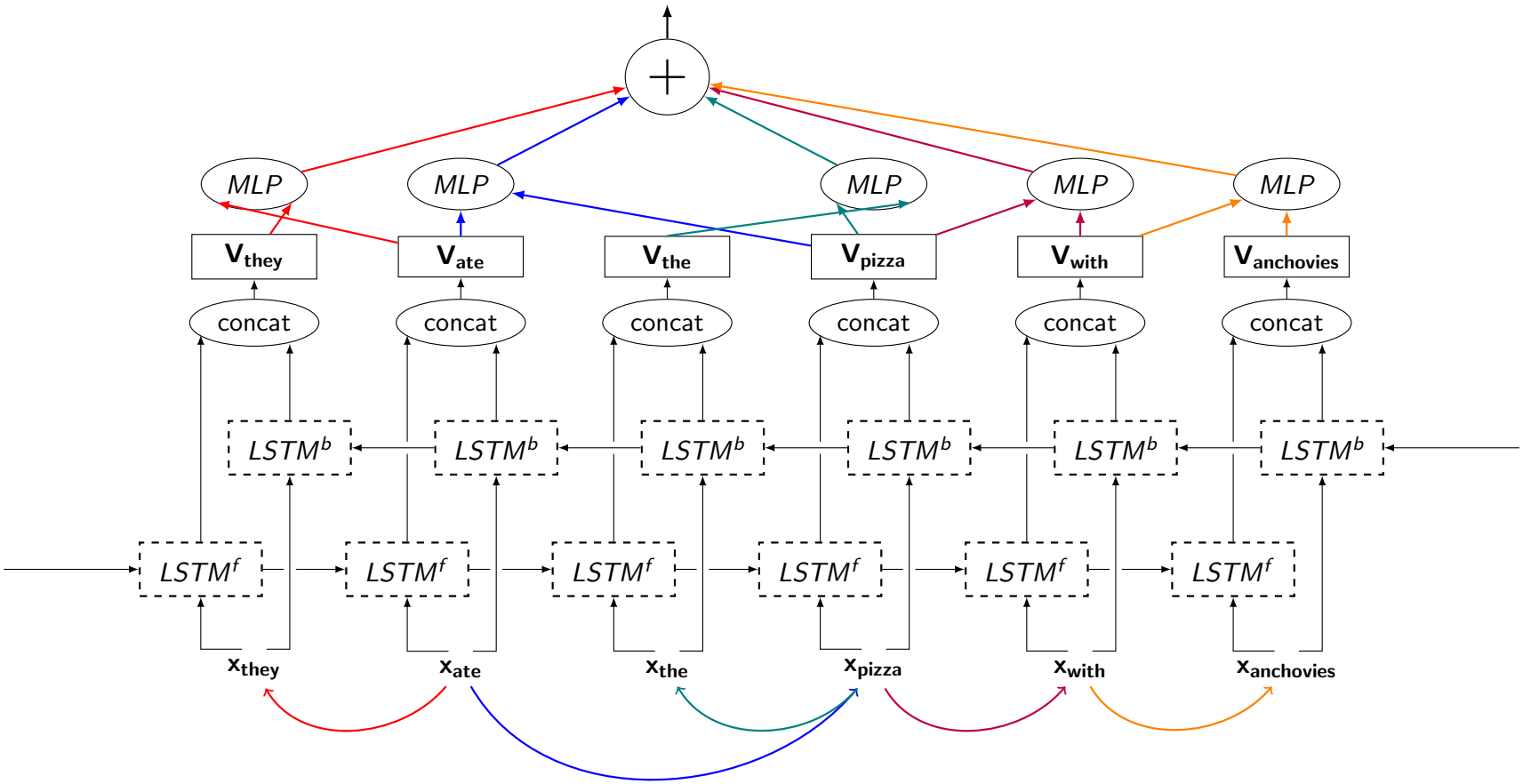
Arc Score (Intuition)

- The BiLSTM encoding of a word holds information about its attachment preferences
- The score is dependent on the BiLSTM encoding which in turn depends on the entire sentence
- Therefore, the score function focused on a specific arc is considering also the entire sentence attachement preferences

Tree Score

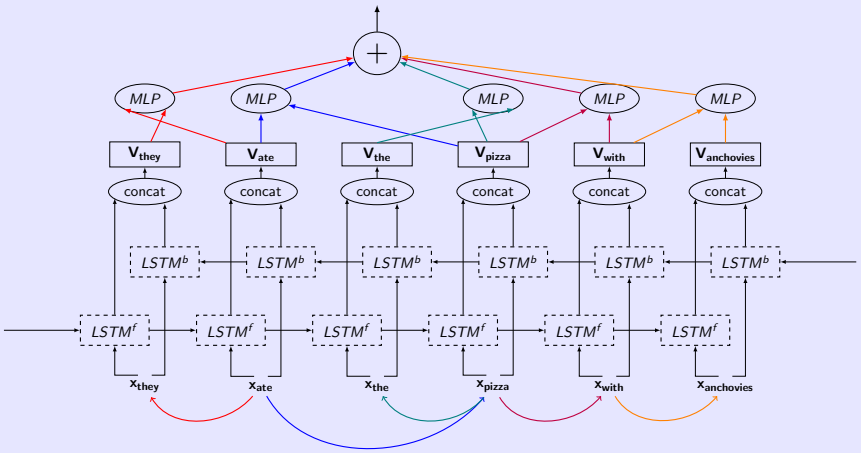
$Score($ 

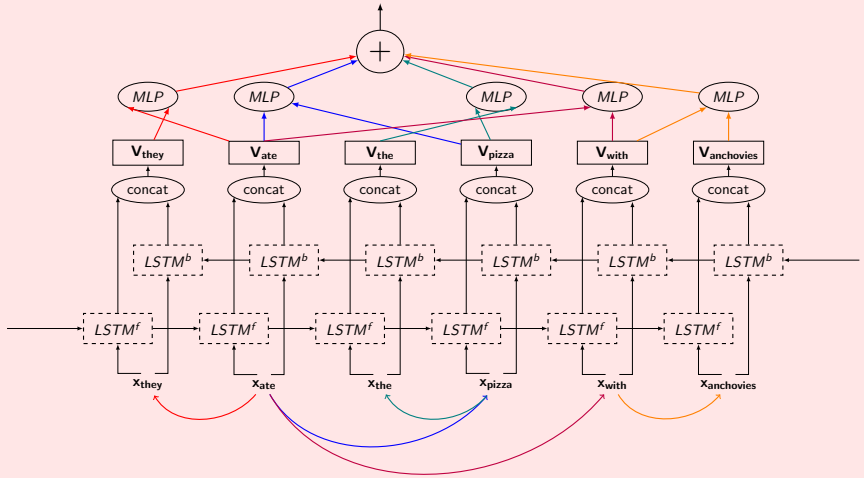
$) =$

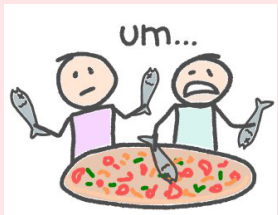


Large Margin Objective

$$max(0, 1 - \square + \square)$$







Graph-based Parsing (More Details)

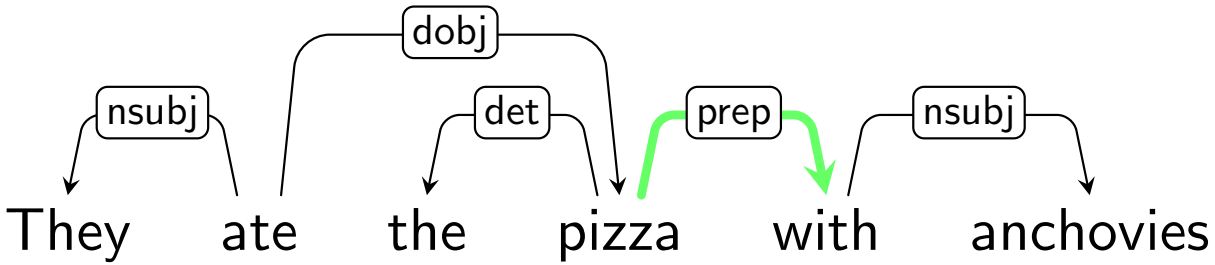
- **Cost Augmentation**: Make non-gold attachments more attractive in training by adding a constant to their score
- **Multi-Task Learning**: Learning the label on the same BiLSTM representation helps both in terms of accuracy and performance.
- **For Speed**: Simple algebric "trick" reduces the number of matrix multiplication significantly.

Graph-based Parsing (More Details)

- **Cost Augmentation**: Make non-gold attachments more attractive in training by adding a constant to their score
- **Multi-Task Learning**: Learning the label on the same BiLSTM representation helps both in terms of accuracy and performance.
- **For Speed**: Simple algebric "trick" reduces the number of matrix multiplication significantly.

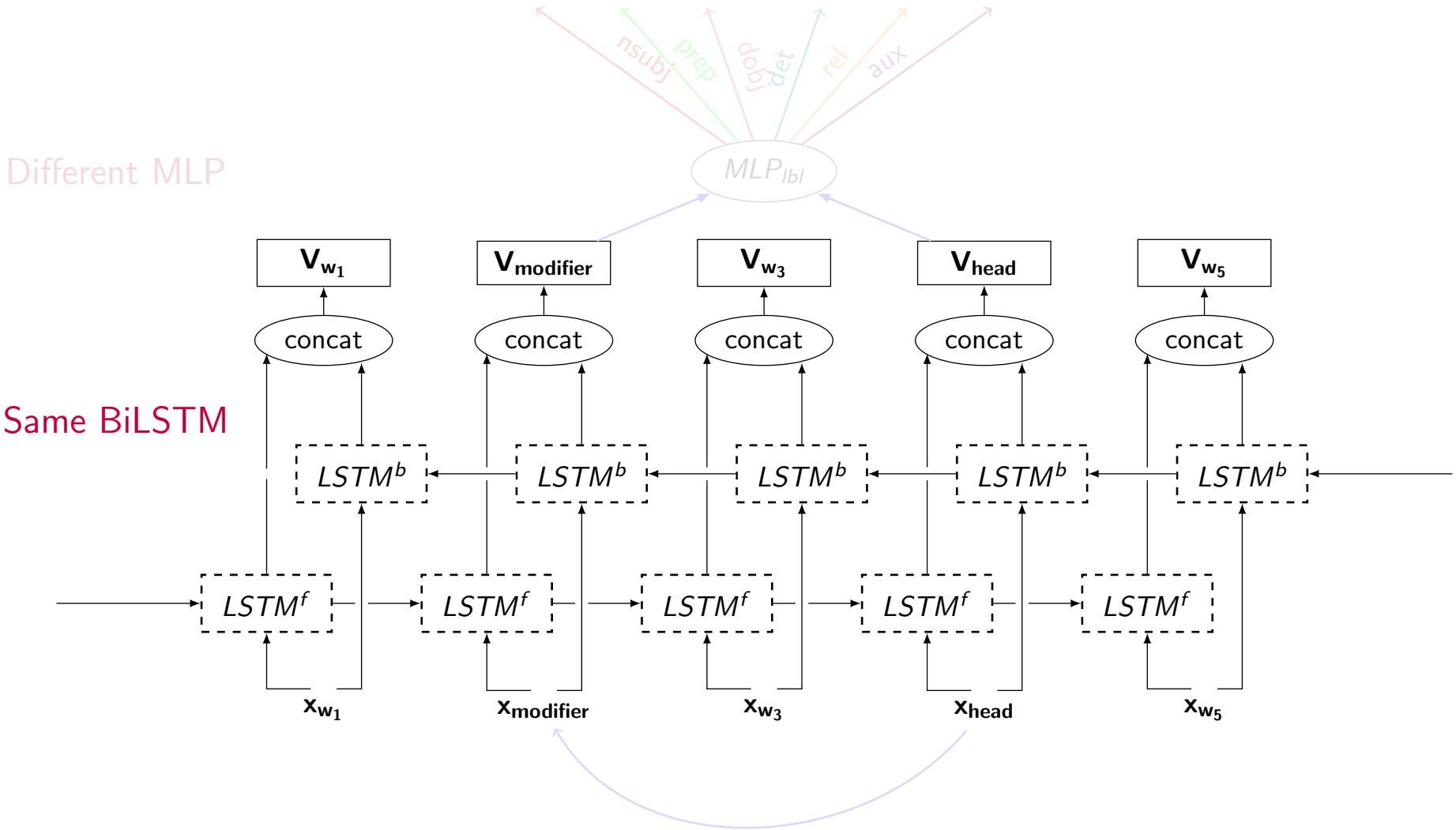
Multi-Task Learning

Arc Labels (Multi-Task Learning)



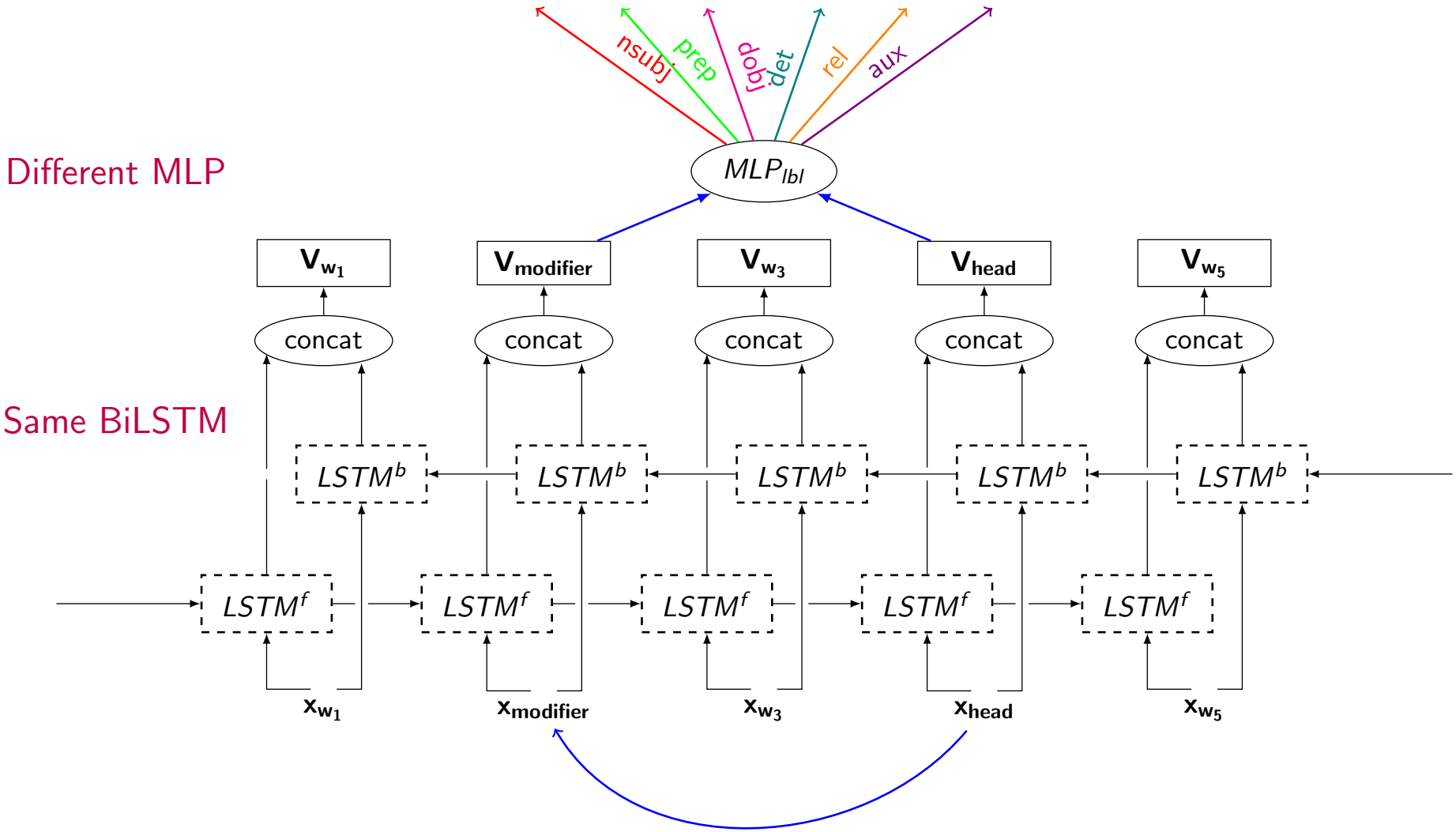
- The arc labels hold important additional syntactic information
- The labels contribute information useful for the unlabeled case too

Arc Labels (Multi-Task Learning)



Enrich BiLSTM representation by learning labels

Arc Labels (Multi-Task Learning)



Enrich BiLSTM representation by learning labels

Graph-based Parsing

and this works:

93.2 UAS with two features,

first-order parser,

without external embeddings.

Graph-based Parsing

and this works:

93.2 UAS with two features,

first-order parser,

without external embeddings.

Graph-based Parsing

and this works:

93.2 UAS with two features,

first-order parser,

without external embeddings.

Graph-based Parsing

and this works:

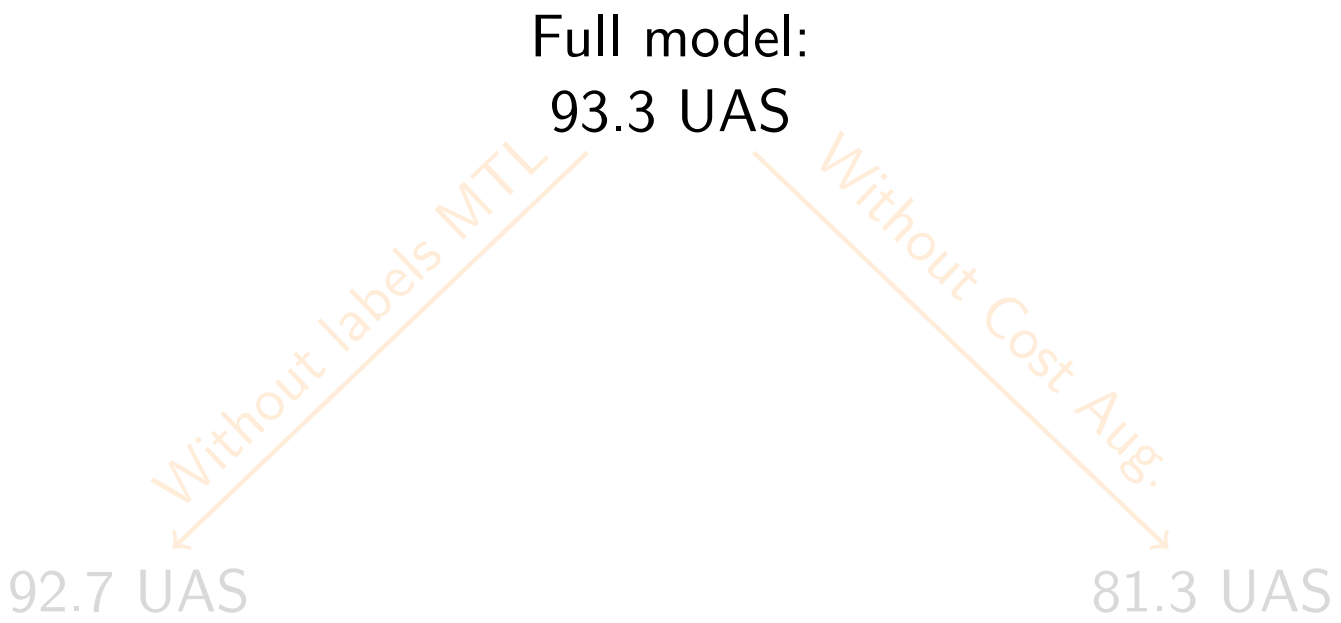
93.2 UAS with two features,

first-order parser,

without external embeddings.

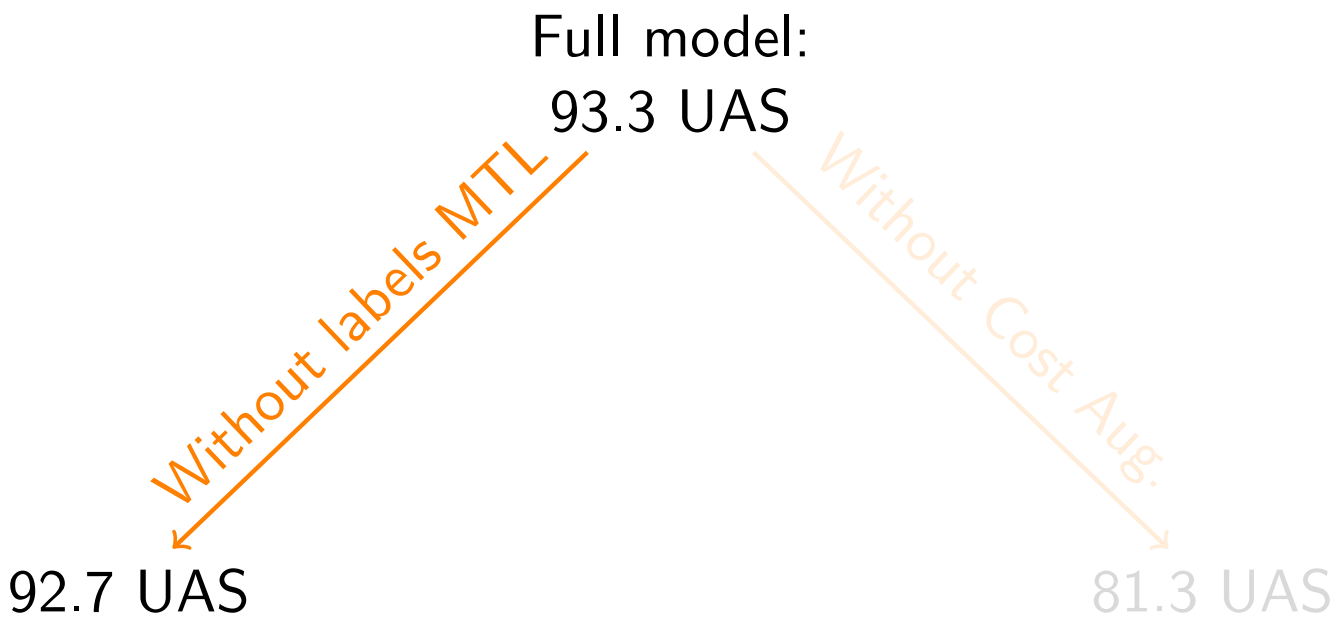
Ablations

Ablation results on the development set:



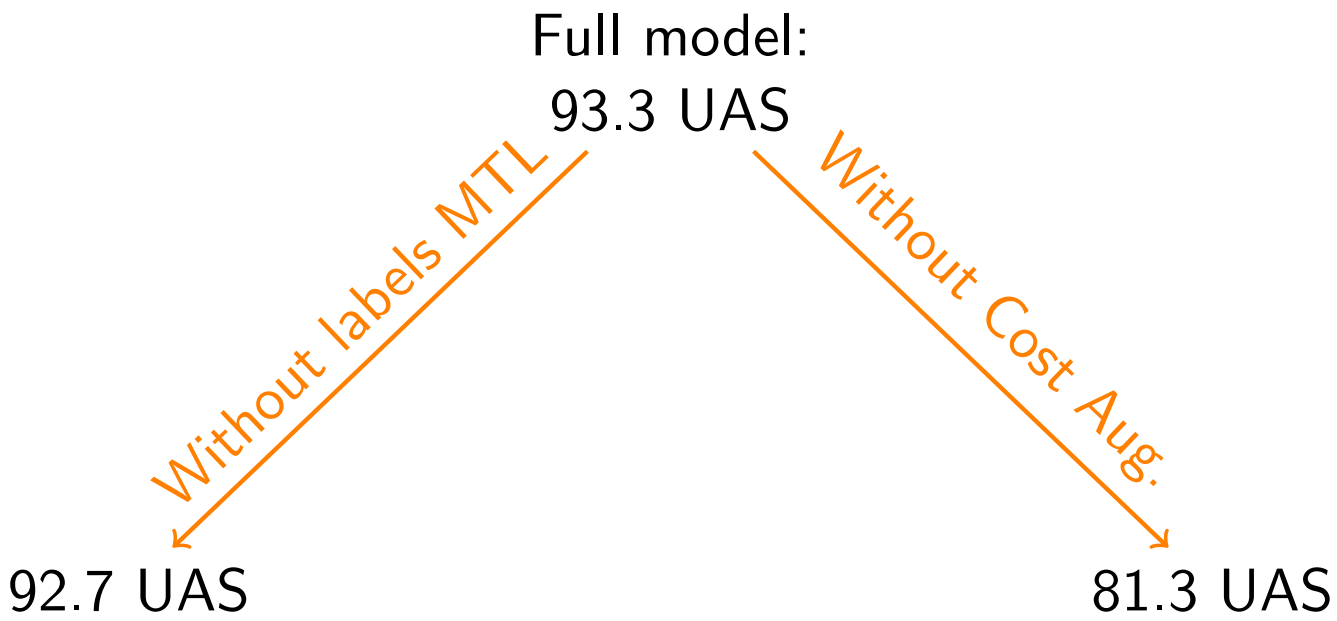
Ablations

Ablation results on the development set:



Ablations

Ablation results on the development set:



Plug&Play

Transition-based Parsing

Transition-based Parsing (Oracle)

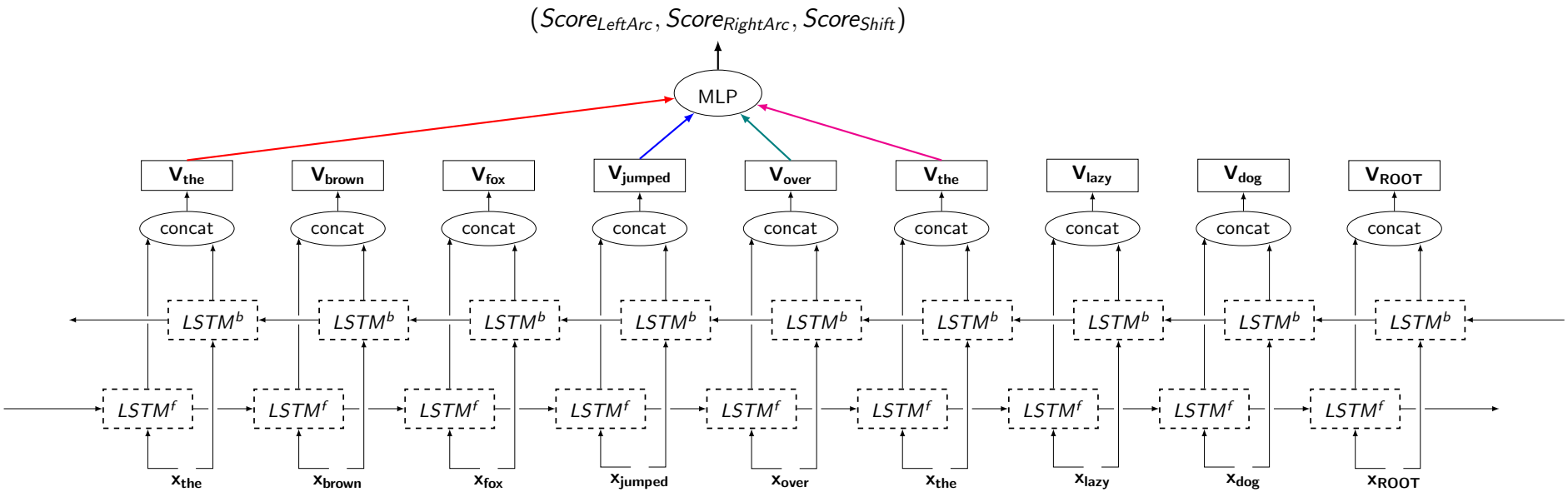
Configuration:

s_2	s_1	s_0	b_0	b_1	b_2	b_3
the	jumped	over	the	lazy	dog	ROOT

fox

brown

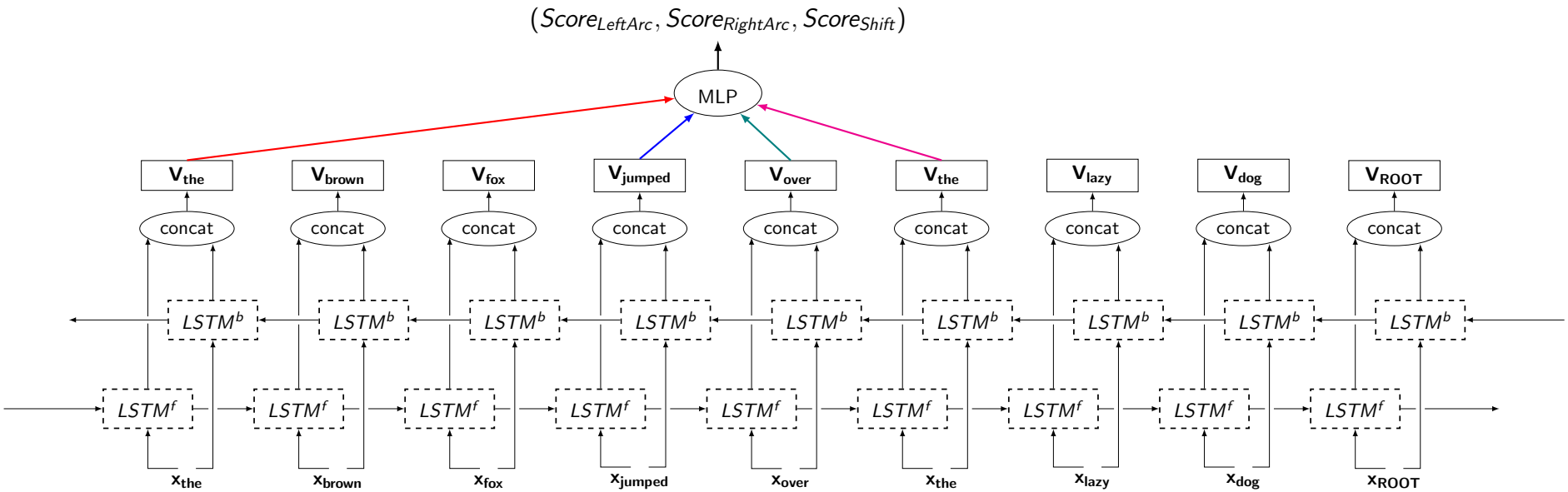
Scoring:



Transition-based Parsing (Learning)

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

Scoring:



Transition-based Parsing (More Details)

- **Dynamic Oracle**: Train the parser to recover from mistaken transitions by allowing the parser to make mistakes in training and consider the gold transition as the one making the least "damage".
- **Aggressive Exploration**: Force the parser to make mistakes in training in order for it to train on more data.
- **Labeling Network**: Our neural network is the sum of two neural networks, one assigning score for unlabled arc and the other scores pairs of arcs and labels. Useful in particular when an arc is connected because is makes least future mistakes.

Results

Results

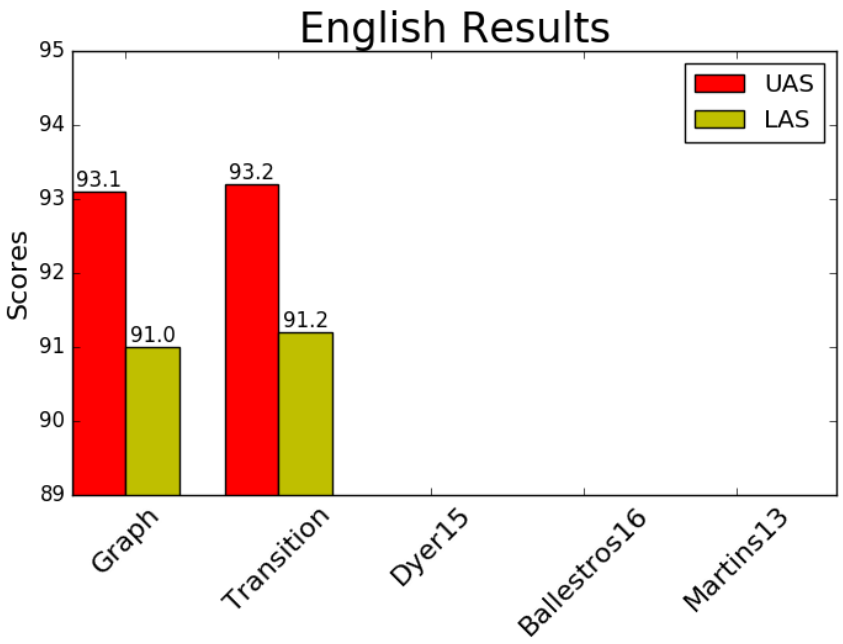
We focus on two kinds of scenarios:

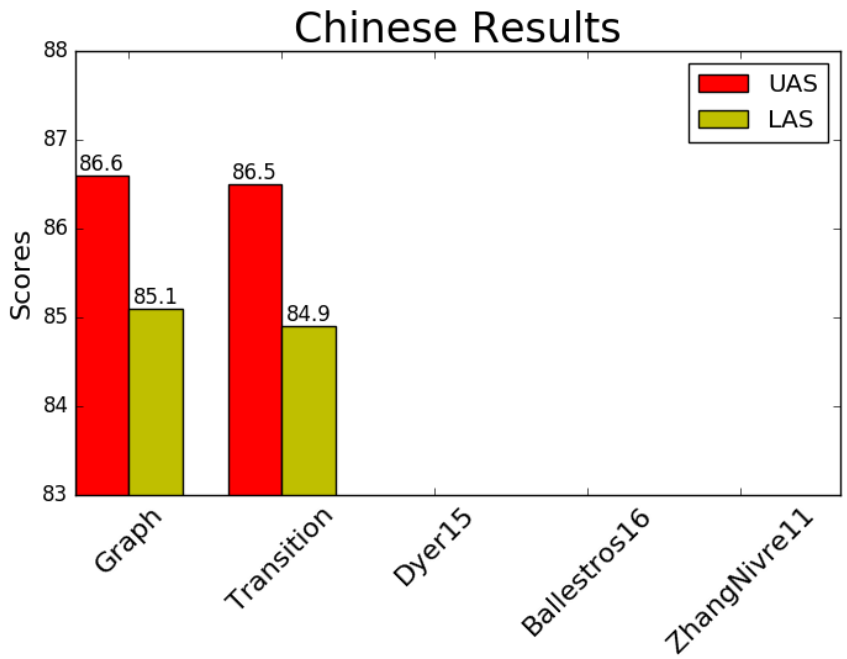
- **Strictly Supervised**: Using only the train data without external embeddings
- **Semi-Supervised**: Using the train data and external embeddings

Results

Strictly Supervised

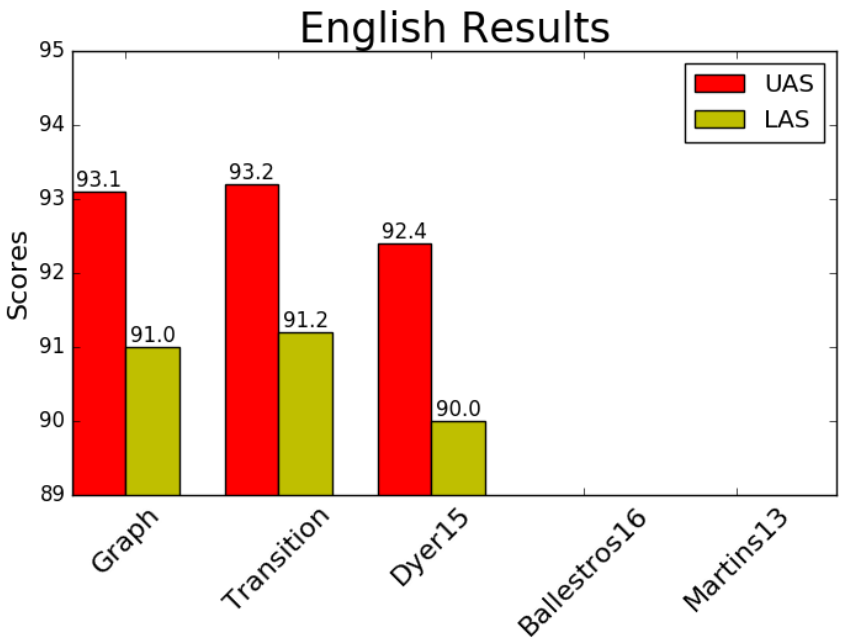
Results (Strictly Supervised)

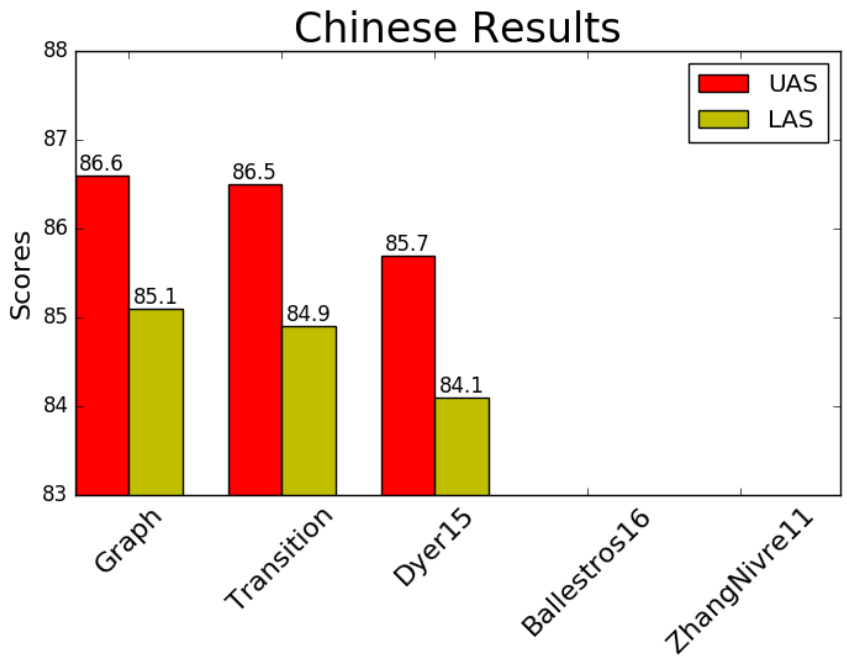




Results (Strictly Supervised)

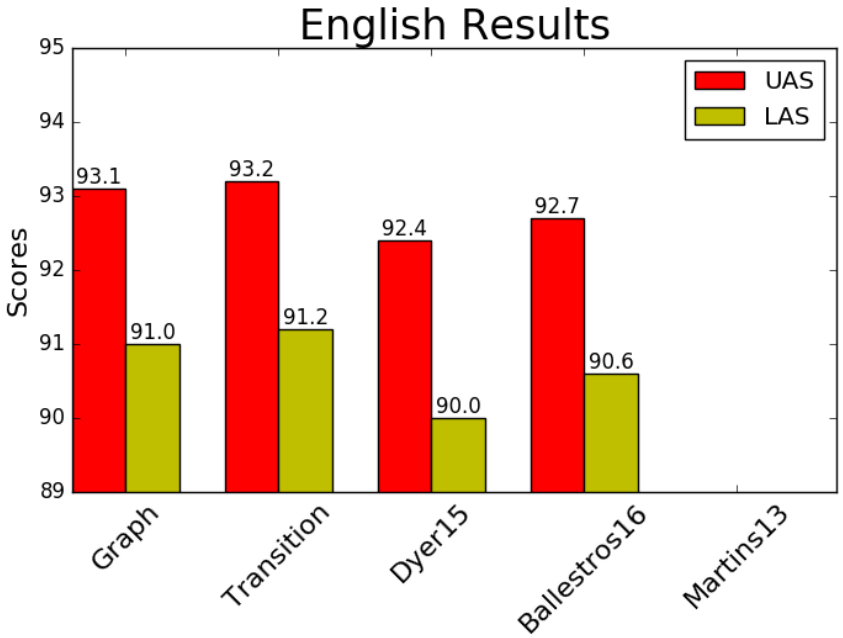
Dyer15: transition (greedy), Stack-LSTM

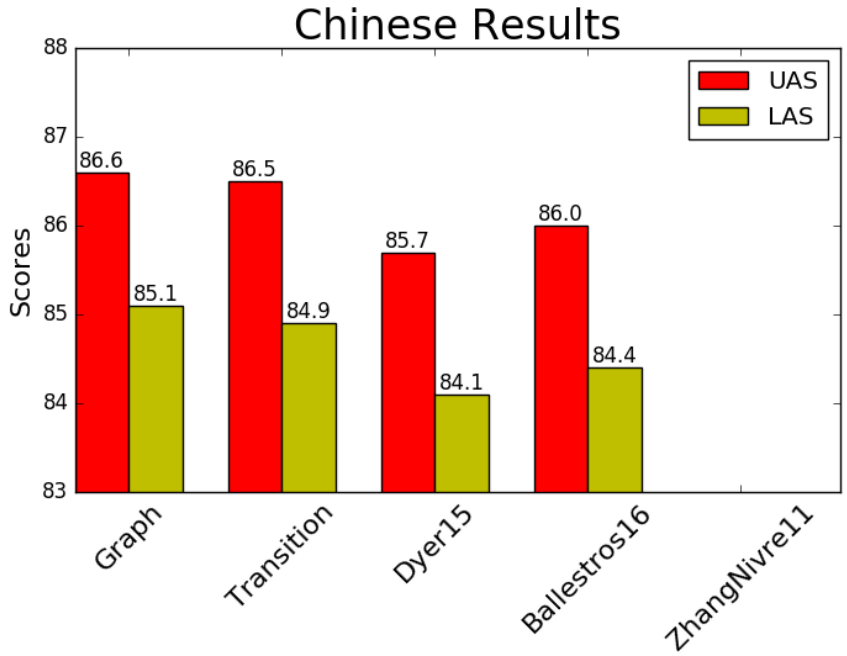




Results (Strictly Supervised)

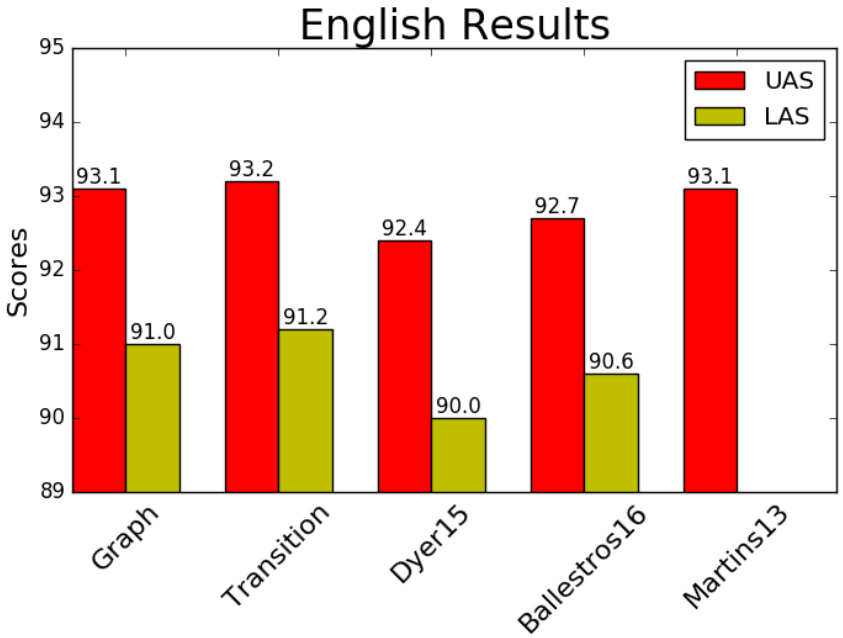
Ballesteros16: transition (greedy, dyn-oracle), Stack-LSTM

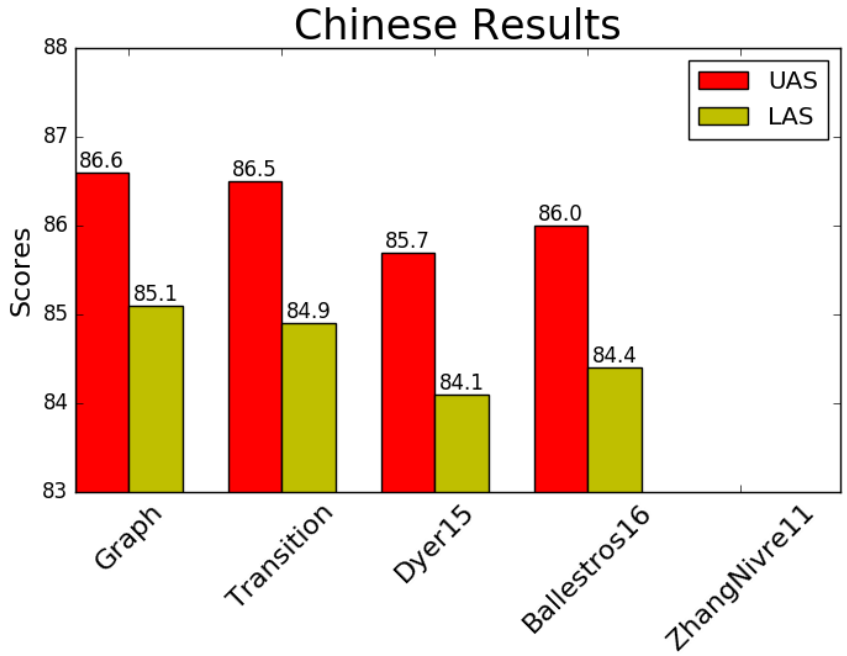




Results (Strictly Supervised)

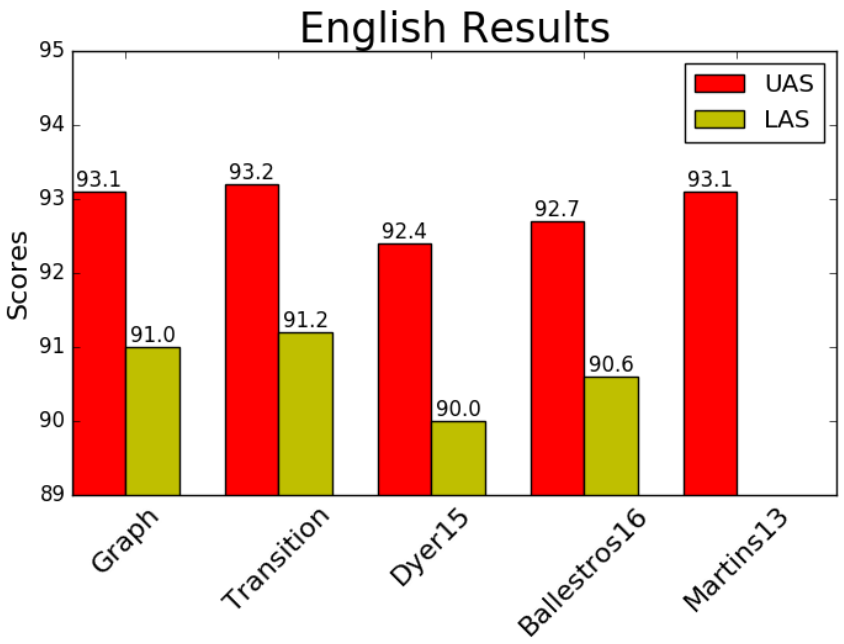
Martins13: graph, 3rd order+, large feature set (sparse)

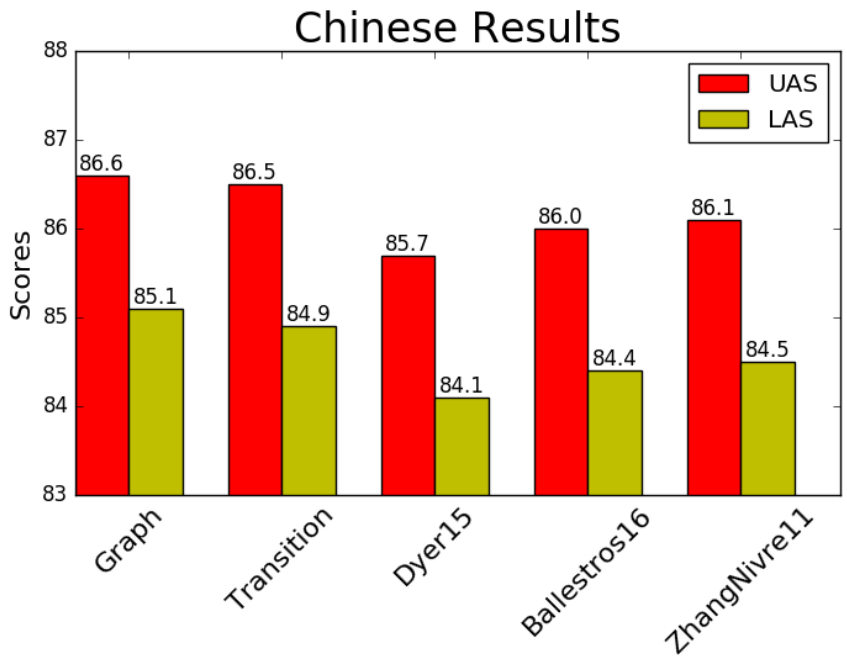




Results (Strictly Supervised)

ZhangNivre11: transition (beam), large feature set (sparse)

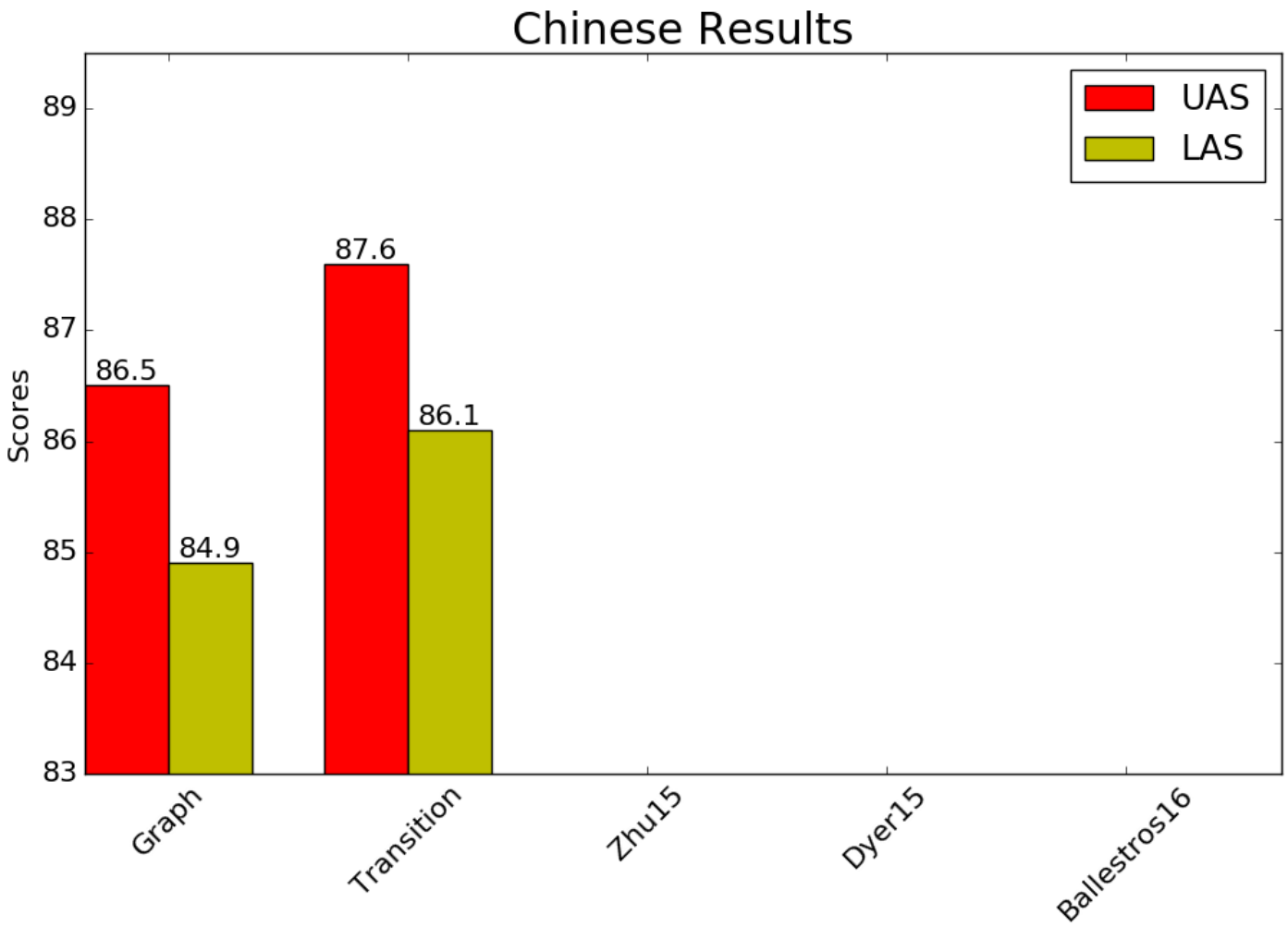




Results

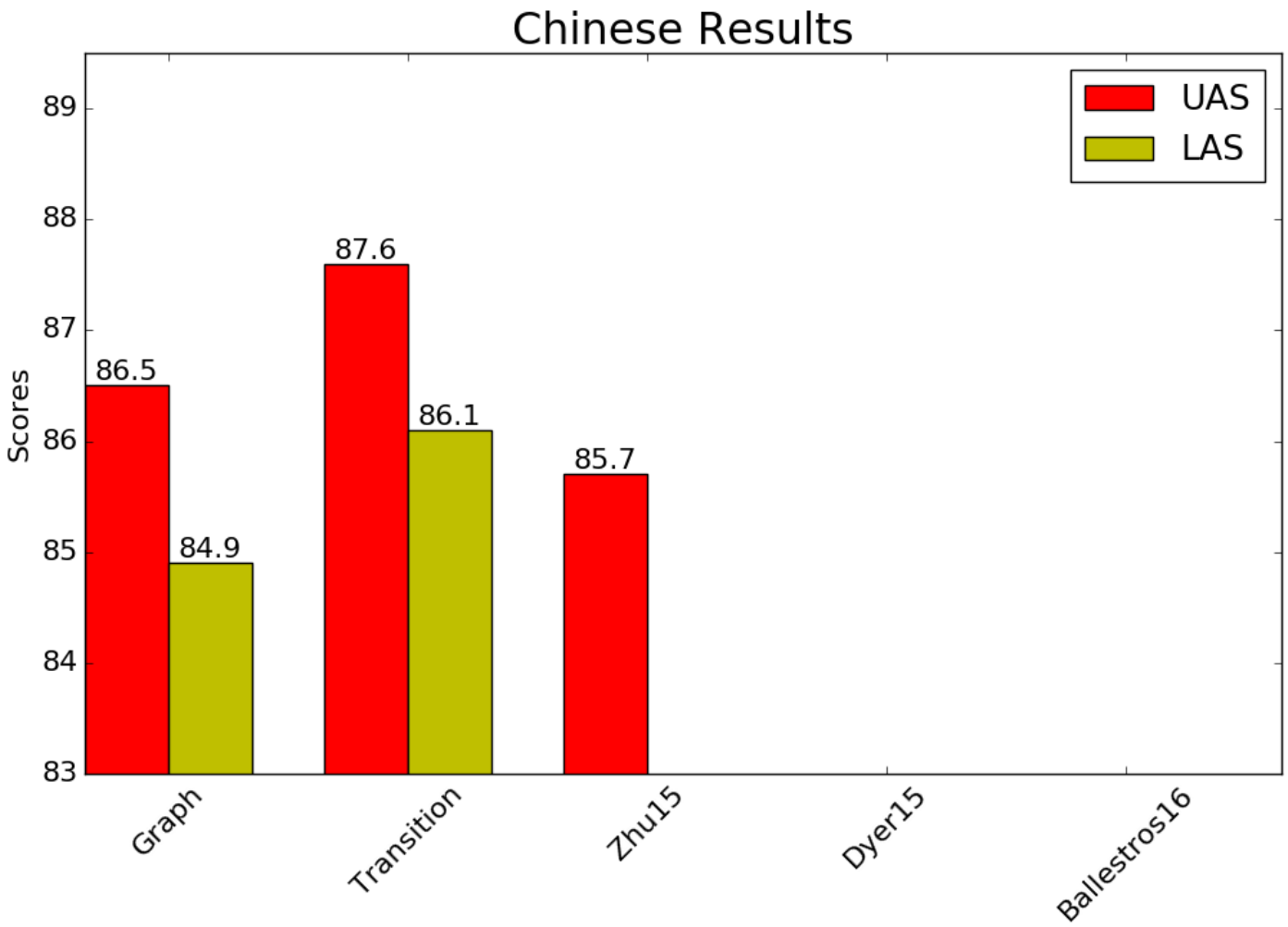
Semi-Supervised

Results CTB (Semi-Supervised)



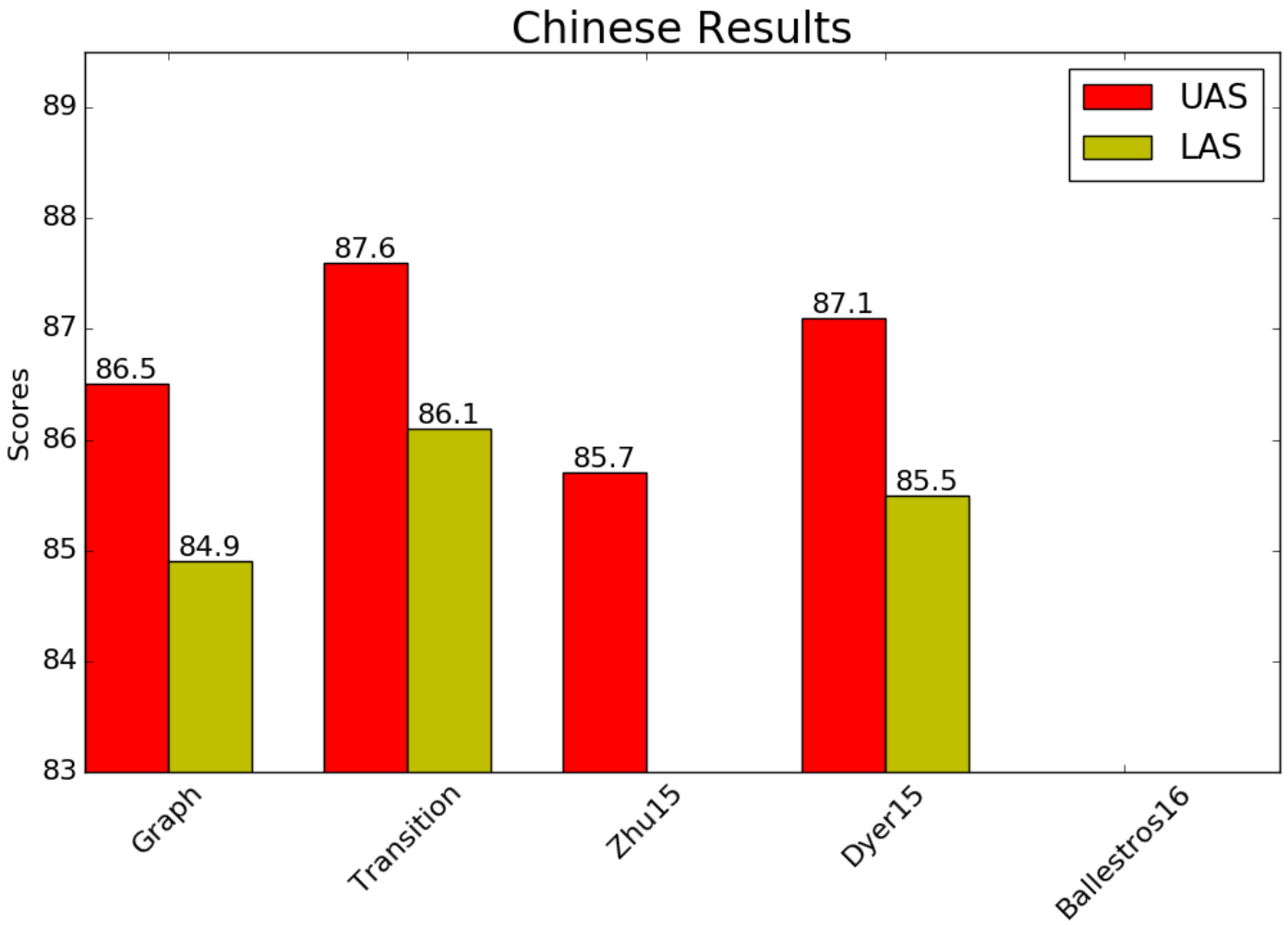
Results CTB (Semi-Supervised)

Zhu15: reranking /blend, recursive conv-net



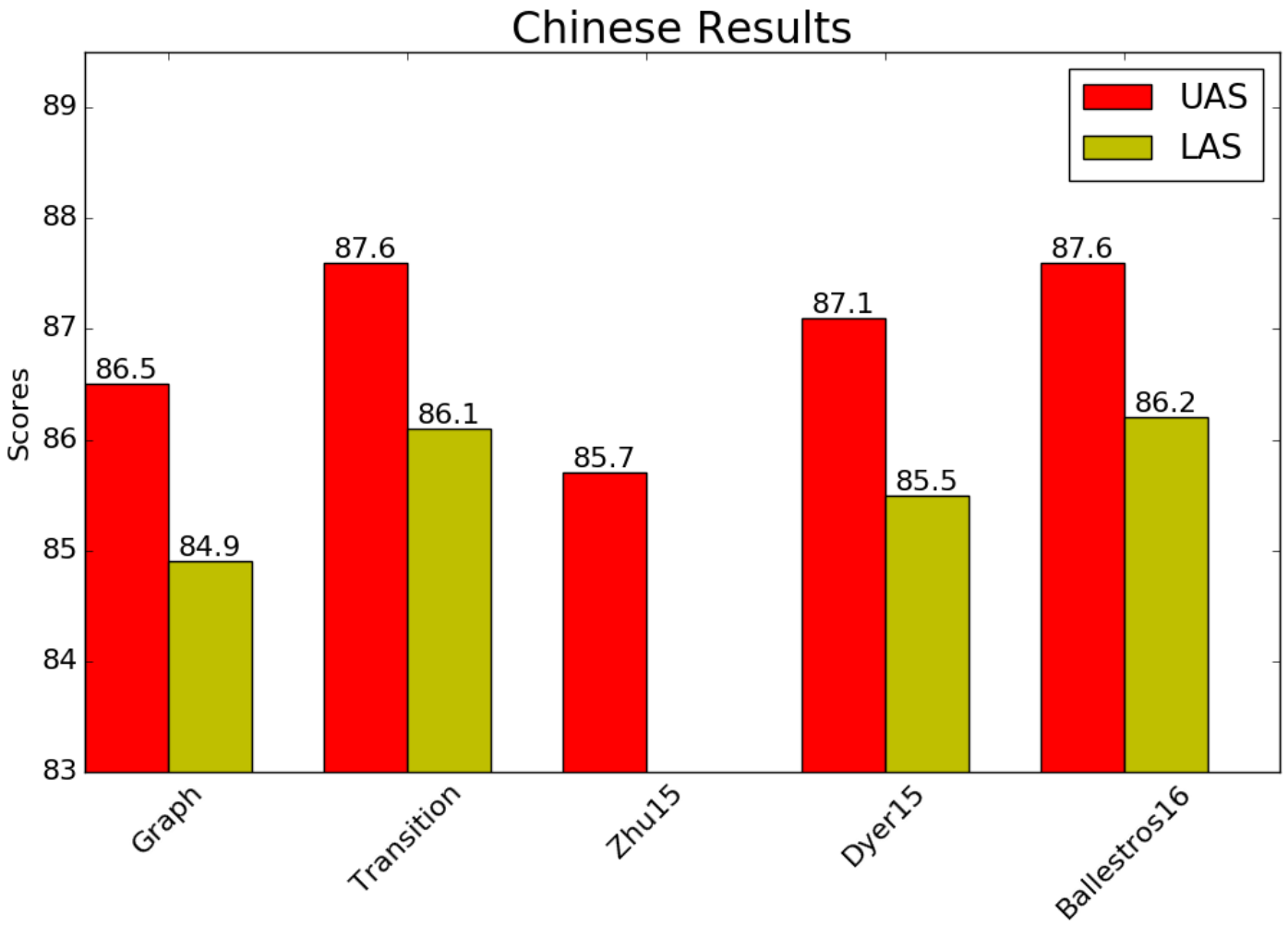
Results CTB (Semi-Supervised)

Dyer15: transition (greedy), Stack-LSTM

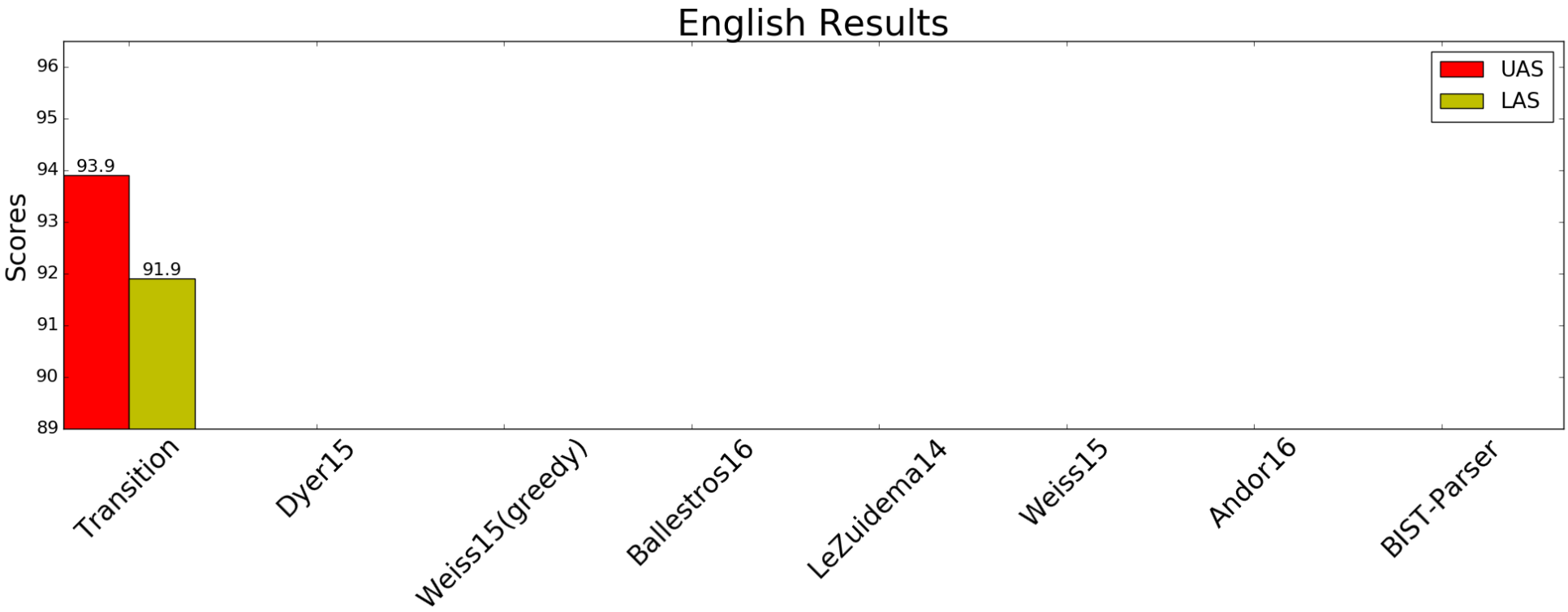


Results CTB (Semi-Supervised)

Ballesteros16: transition (greedy, dyn-oracle), Stack-LSTM

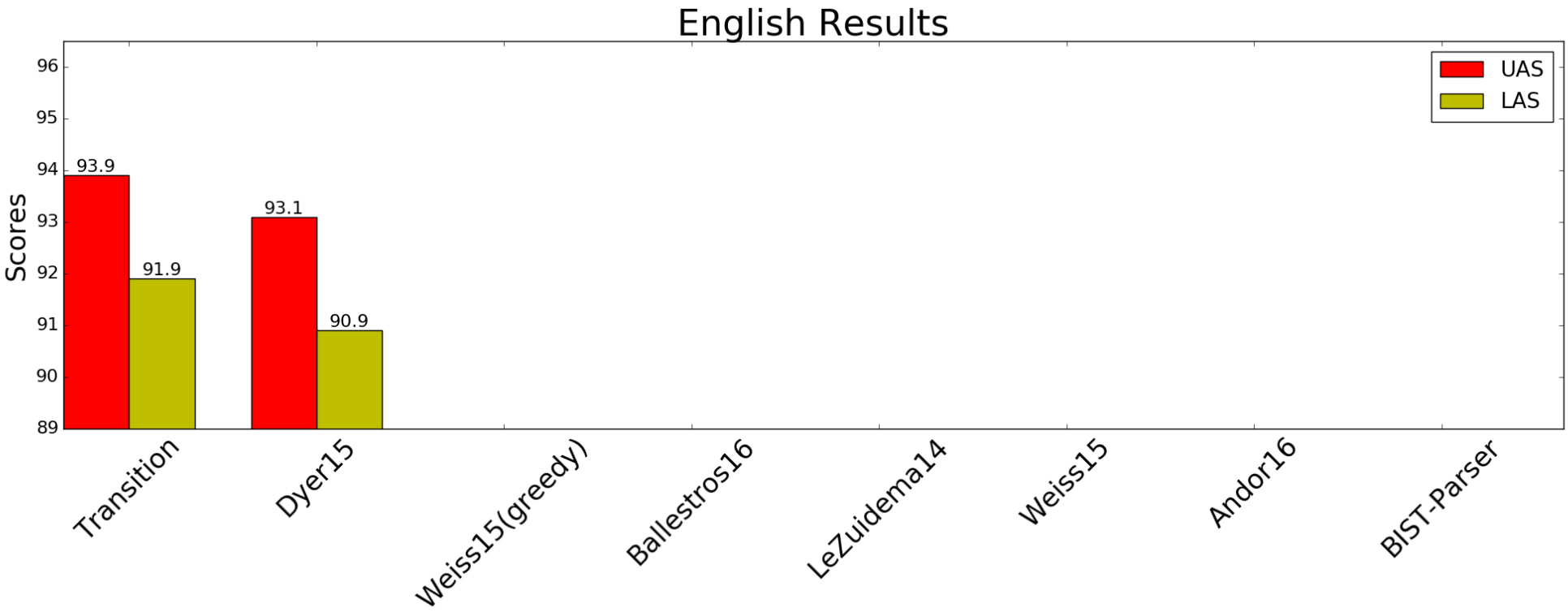


Results PTB (Semi-Supervised)



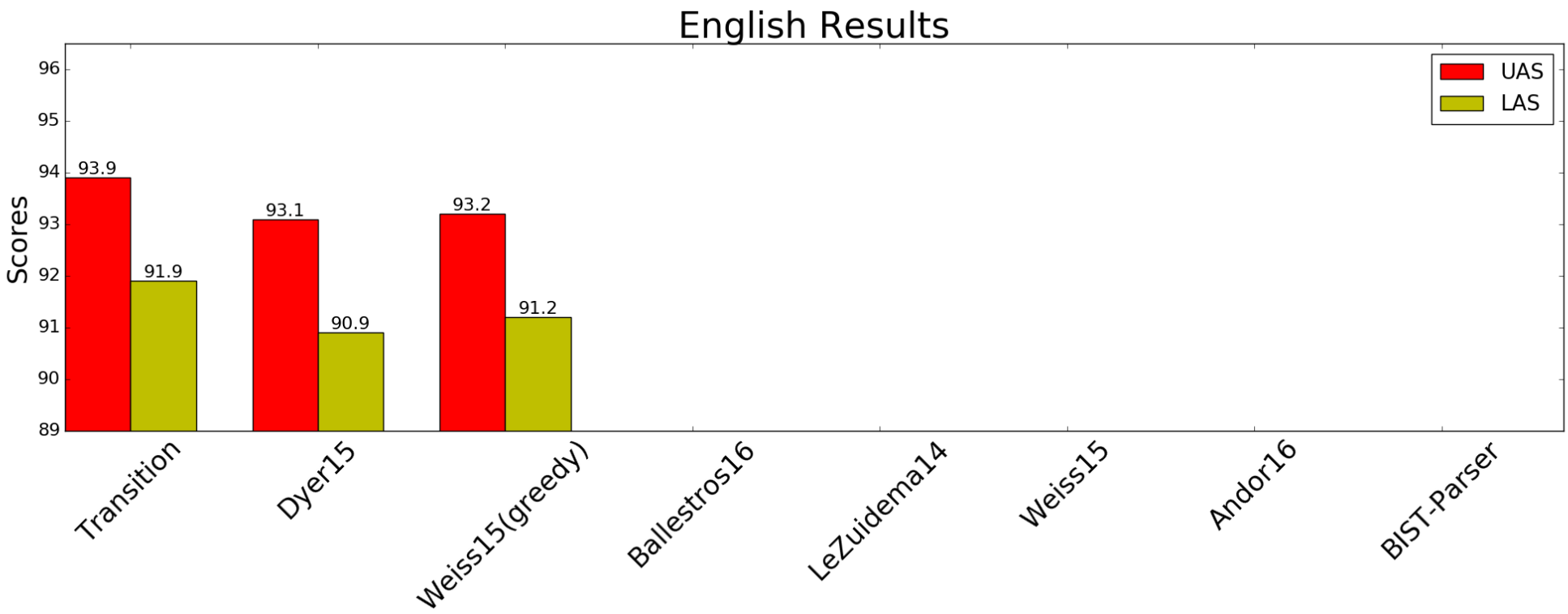
Results PTB (Semi-Supervised)

Dyer15: transition (greedy), Stack-LSTM



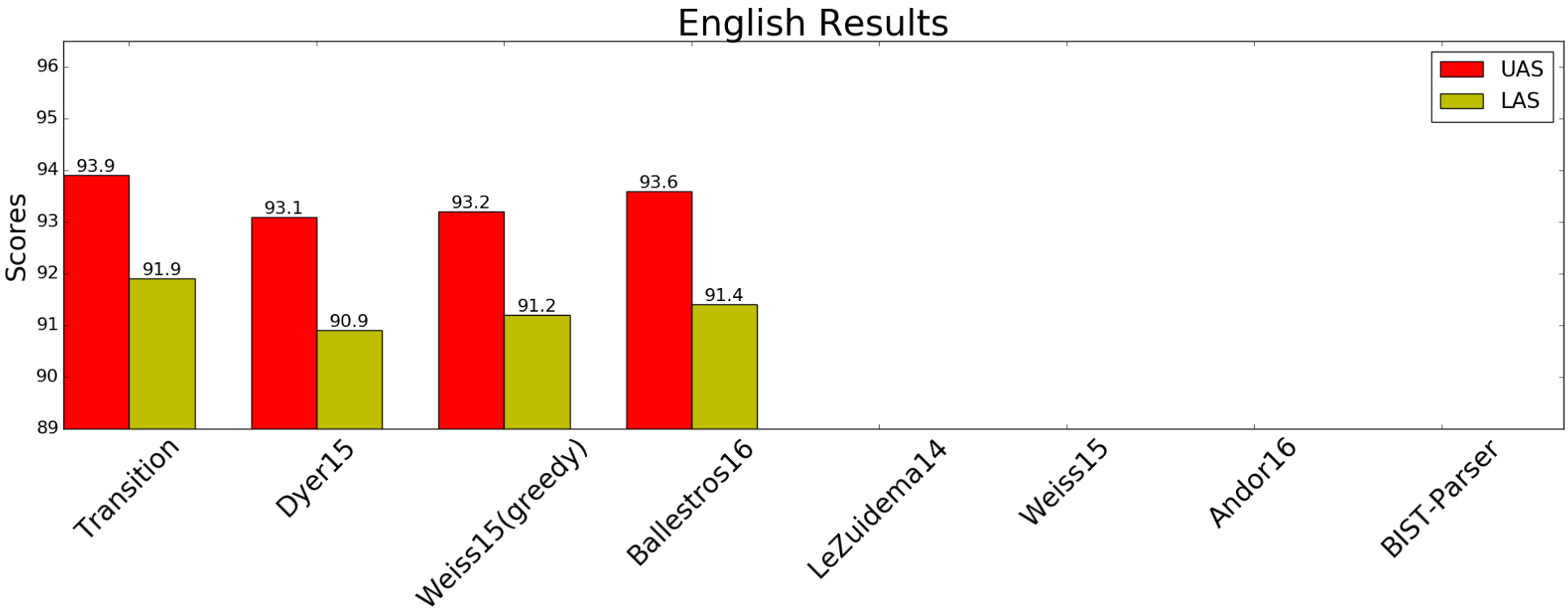
Results PTB (Semi-Supervised)

Weiss15(greedy): transition (greedy), large feature set (dense)



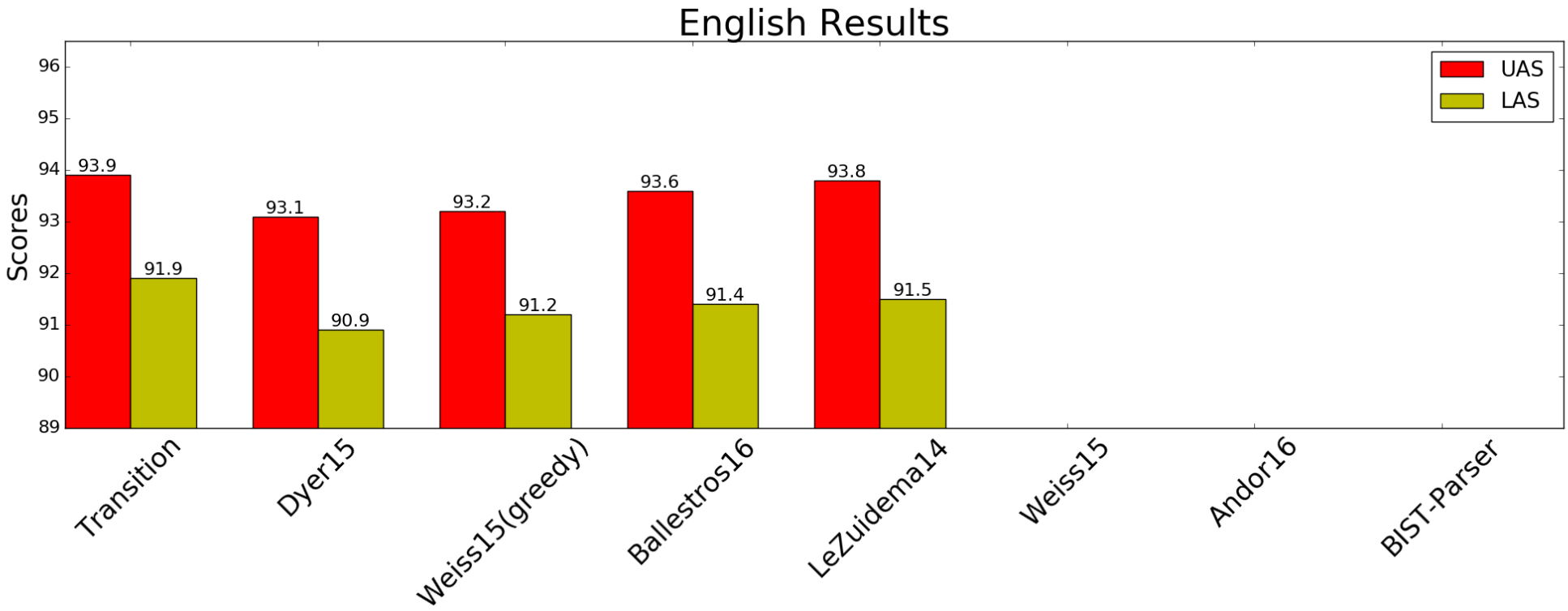
Results PTB (Semi-Supervised)

Ballesteros16: transition (greedy, dyn-oracle), Stack-LSTM



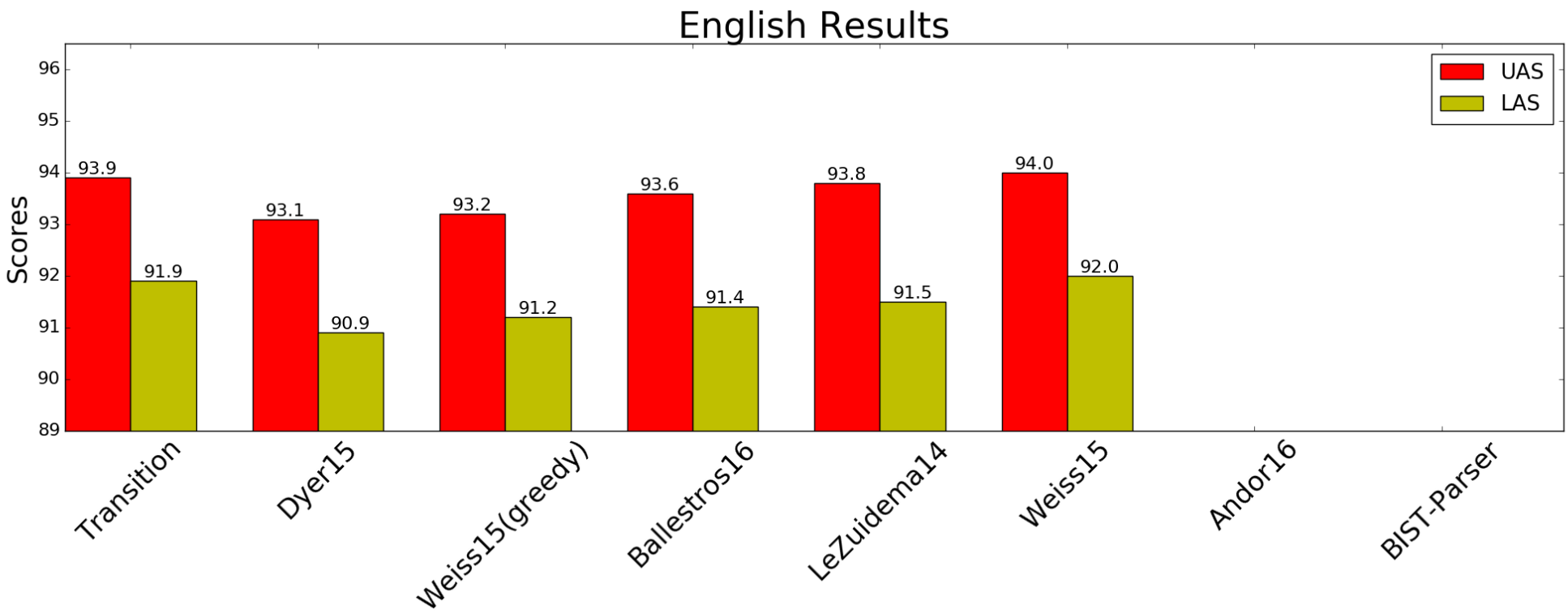
Results PTB (Semi-Supervised)

LeZuidema14: reranking /blend, inside-outside recursive net



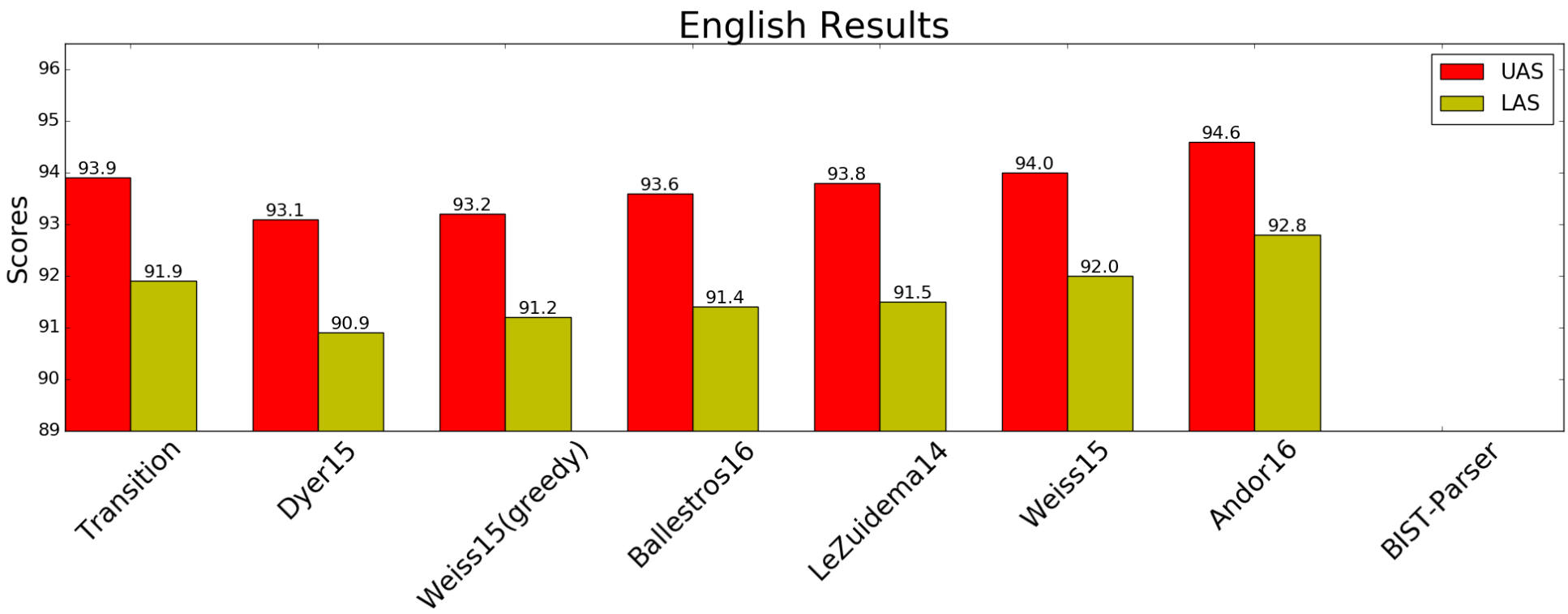
Results PTB (Semi-Supervised)

Weiss15: transition (beam), large feature set (dense)



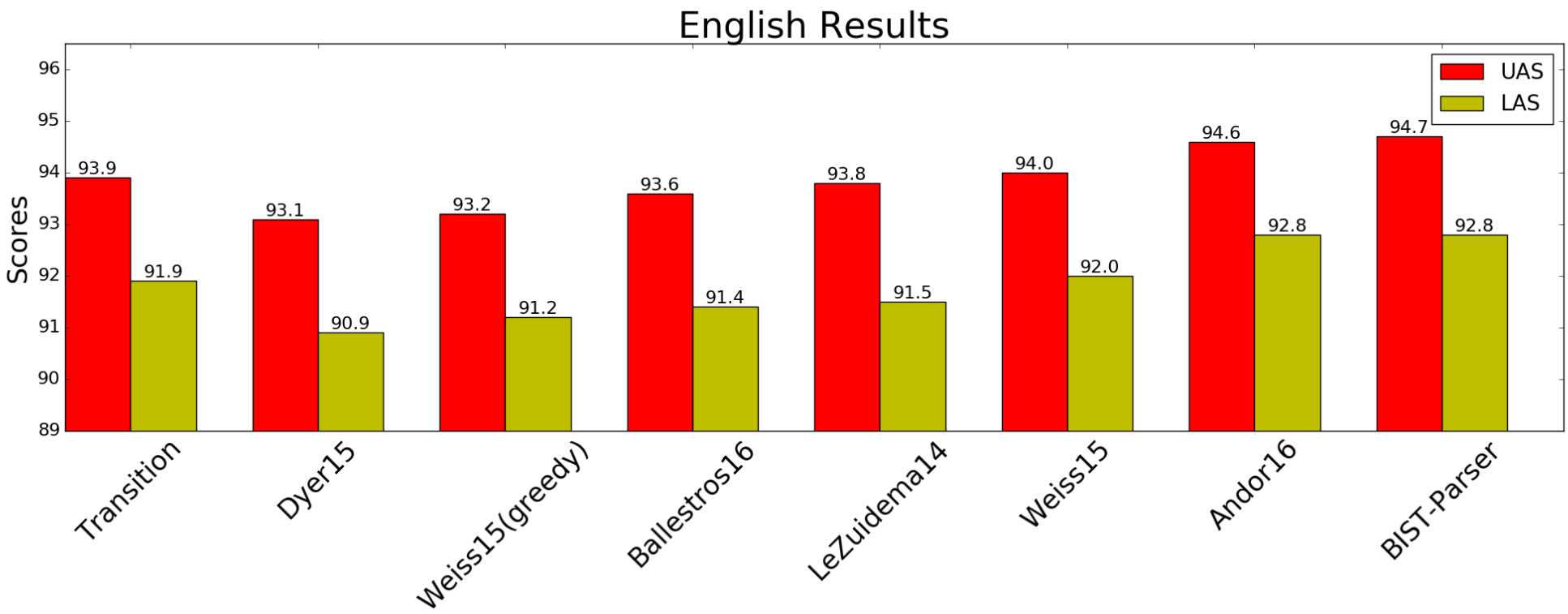
Results PTB (Semi-Supervised)

Andor16: transition (beam), large feature set (dense)



Results PTB (Semi-Supervised)

BIST-Parser: This work with a twist. Available on github.



Summary

We have seen that BiLSTMs are powerful feature extractors

- Easy to train end-to-end
- Simple to plug&play
- Add global information
- Strong results

So we encourage using them

Summary

We have seen that BiLSTMs are powerful feature extractors

- Easy to train end-to-end
- Simple to plug&play
- Add global information
- Strong results

So we encourage using them

Thank You

Parsers Code: `https://github.com/elikip/bist-parser`

2021 Update

- Timothy Dozat and Christopher Manning (ICLR 2017): *Deep Biaffine Attention for Neural Dependency Parsing*
  - Similar to Kiperwasser and Goldberg's graph-based parser, with some tweaks
  - Now the most popular model for dependency parsing (I think)
  - Most work focusing on further improving the feature extractors
- No big new breakthroughs in transition-based parsing
  - slightly below state-of-the-art
  - advantages: fast, incremental
  - popular especially in semantic parsing, cf. e.g., Daniel Hershcovich, Omri Abend, Ari Rappoport (ACL 2017): *A Transition-Based Directed Acyclic Graph Parser for UCCA*