A Fast and Accurate Dependency Parser using Neural Networks

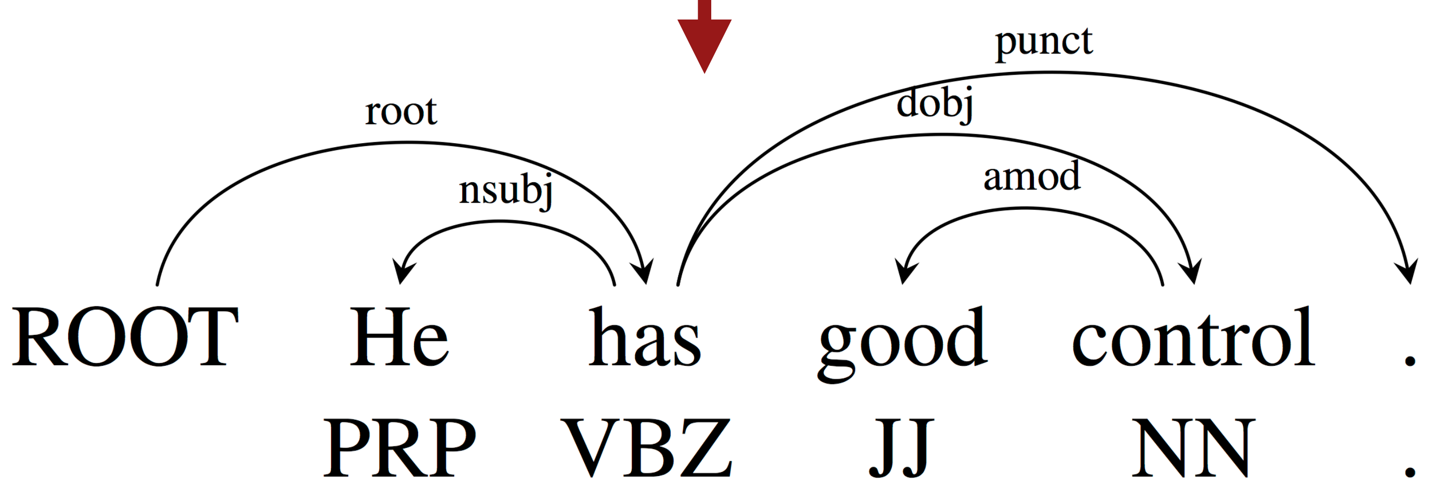
Danqi Chen and Christopher Manning

Stanford University

October 27, 2014

Dependency Parsing

He has good control.



Goal: accurate and fast parsing

(Zhang and Nirve 2011, Martins et al 2013)

Our Work

- A neural network based dependency parser!

(Zhang and Nirve 2011, Martins et al 2013)

Our Work

- A neural network based dependency parser!

Parsing on English Penn Treebank (§23):

	Unlabeled attachment score (UAS)	sent / s
Transition -based		

Our Work

- A neural network based dependency parser!

Parsing on English Penn Treebank (§23):

		Unlabeled attachment score (UAS)	sent / s
Transition -based	MaltParser (greedy)	89.9	560

Our Work

- A neural network based dependency parser!

Parsing on English Penn Treebank (§23):

		Unlabeled attachment score (UAS)	sent / s
Transition-based	MaltParser (greedy)	89.9	560
	Zpar: beam = 64	92.9*	29*

(Zhang and Nirve 2011, Martins et al 2013)

Our Work

- A neural network based dependency parser!

Parsing on English Penn Treebank (§23):

		Unlabeled attachment score (UAS)	sent / s
Transition-based	MaltParser (greedy)	89.9	560
	Zpar: beam = 64	92.9*	29*
Graph-based	MSTParser	92.0	12
	TurboParser	93.1*	31*

Our Work

- A neural network based dependency parser!

Parsing on English Penn Treebank (§23):

		Unlabeled attachment score (UAS)	sent / s
Transition -based	MaltParser (greedy)	89.9	560
	Our Parser (greedy)	92.0	1013
	Zpar: beam = 64	92.9*	29*
Graph -based	MSTParser	92.0	12
	TurboParser	93.1*	31*

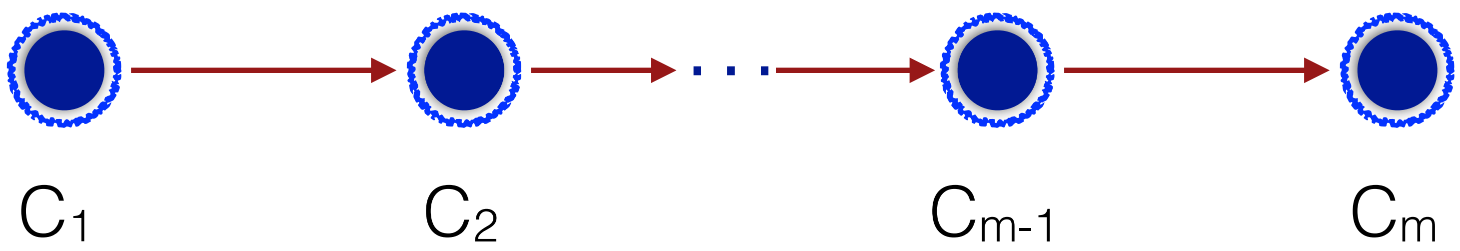
+2.1 ×1.8

Outline

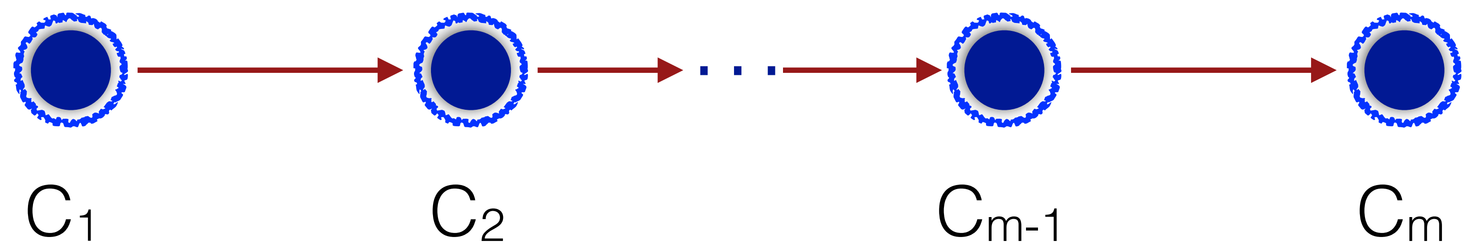
- Background & Motivation
- Model
- Experiments
- Analysis

(Nivre et al, 2004)

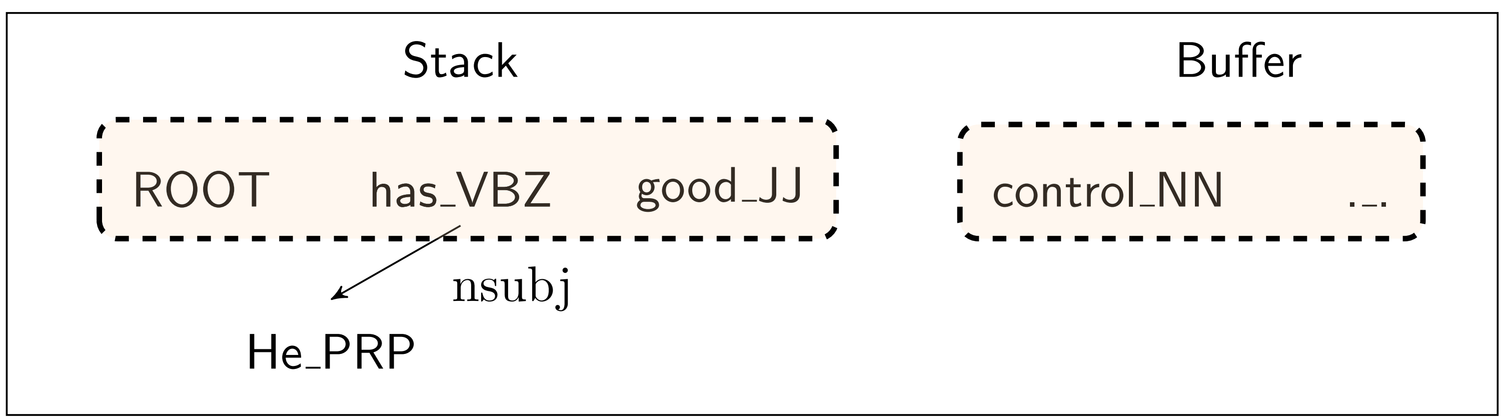
Greedy Transition-based Parsing



Greedy Transition-based Parsing

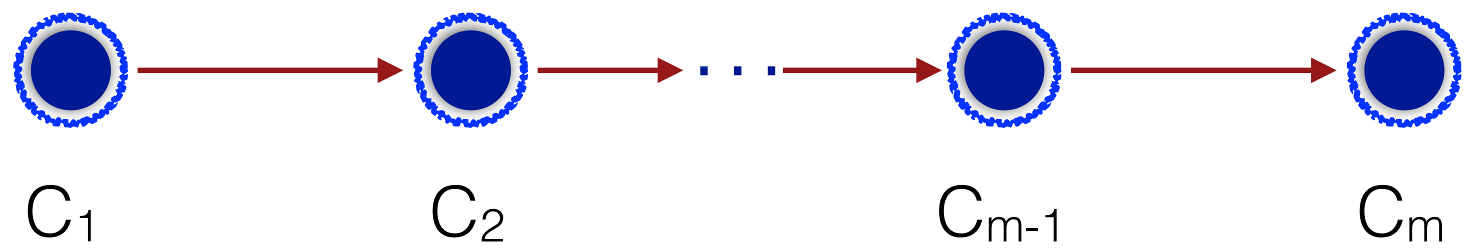


- A configuration = a stack, a buffer and some dependency arcs

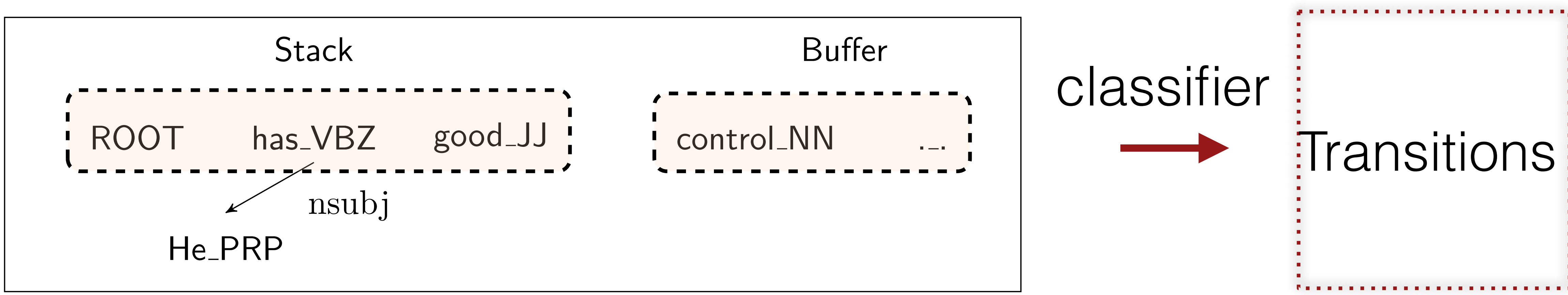


(Nivre et al, 2004)

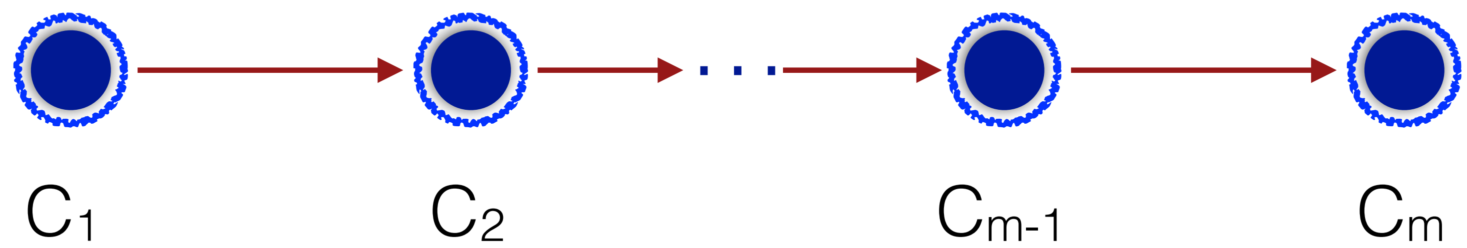
Greedy Transition-based Parsing



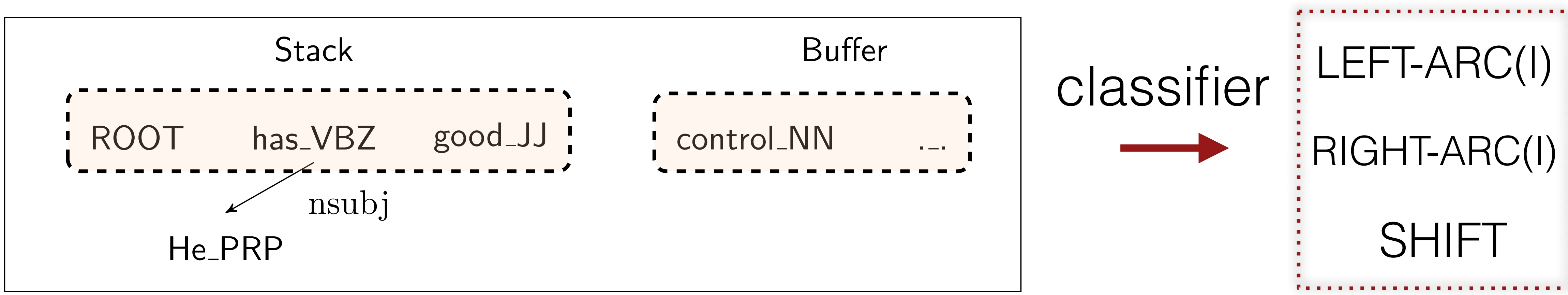
- A configuration = a stack, a buffer and some dependency arcs



Greedy Transition-based Parsing

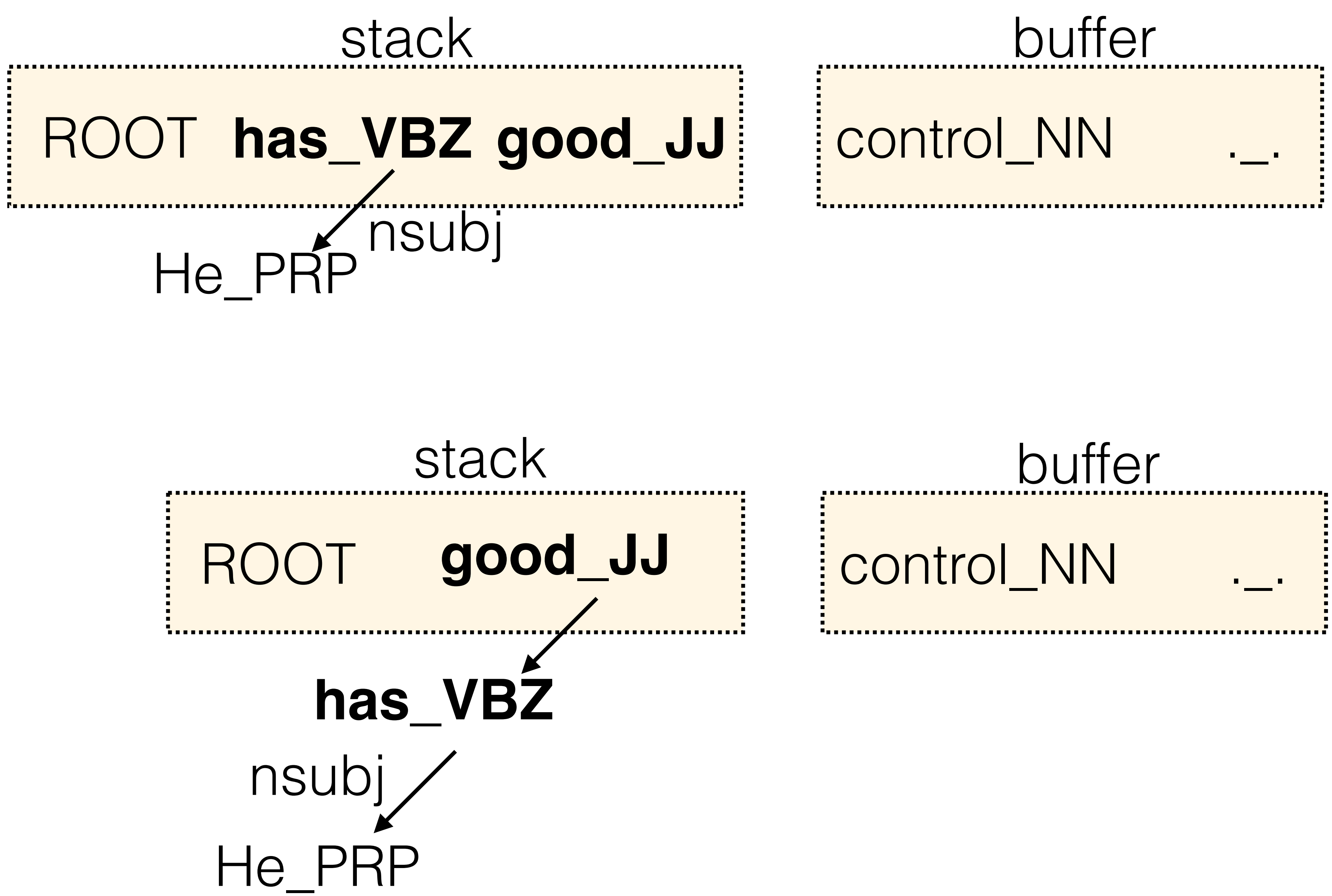


- A configuration = a stack, a buffer and some dependency arcs

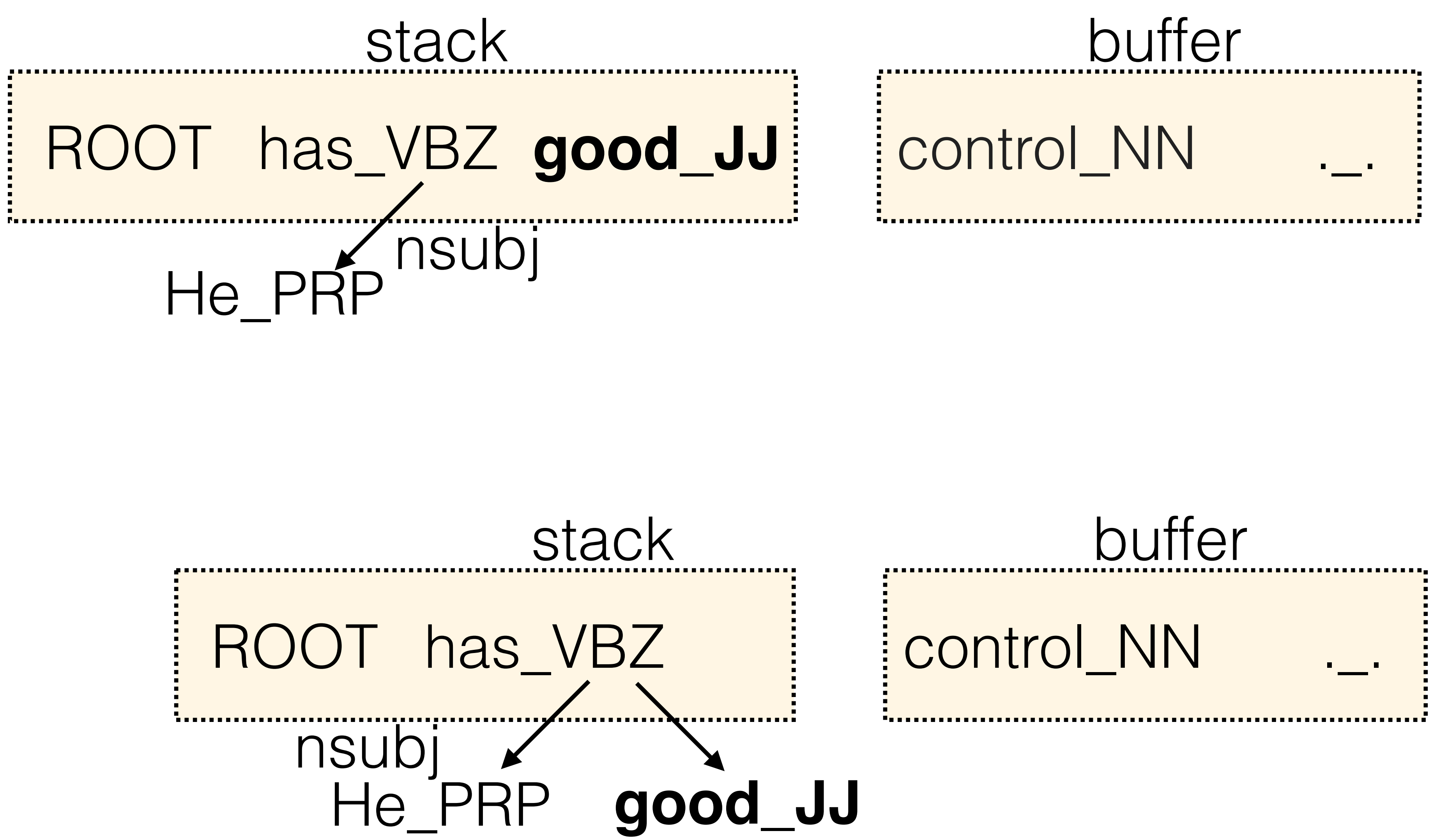


- We employ the arc-standard system.

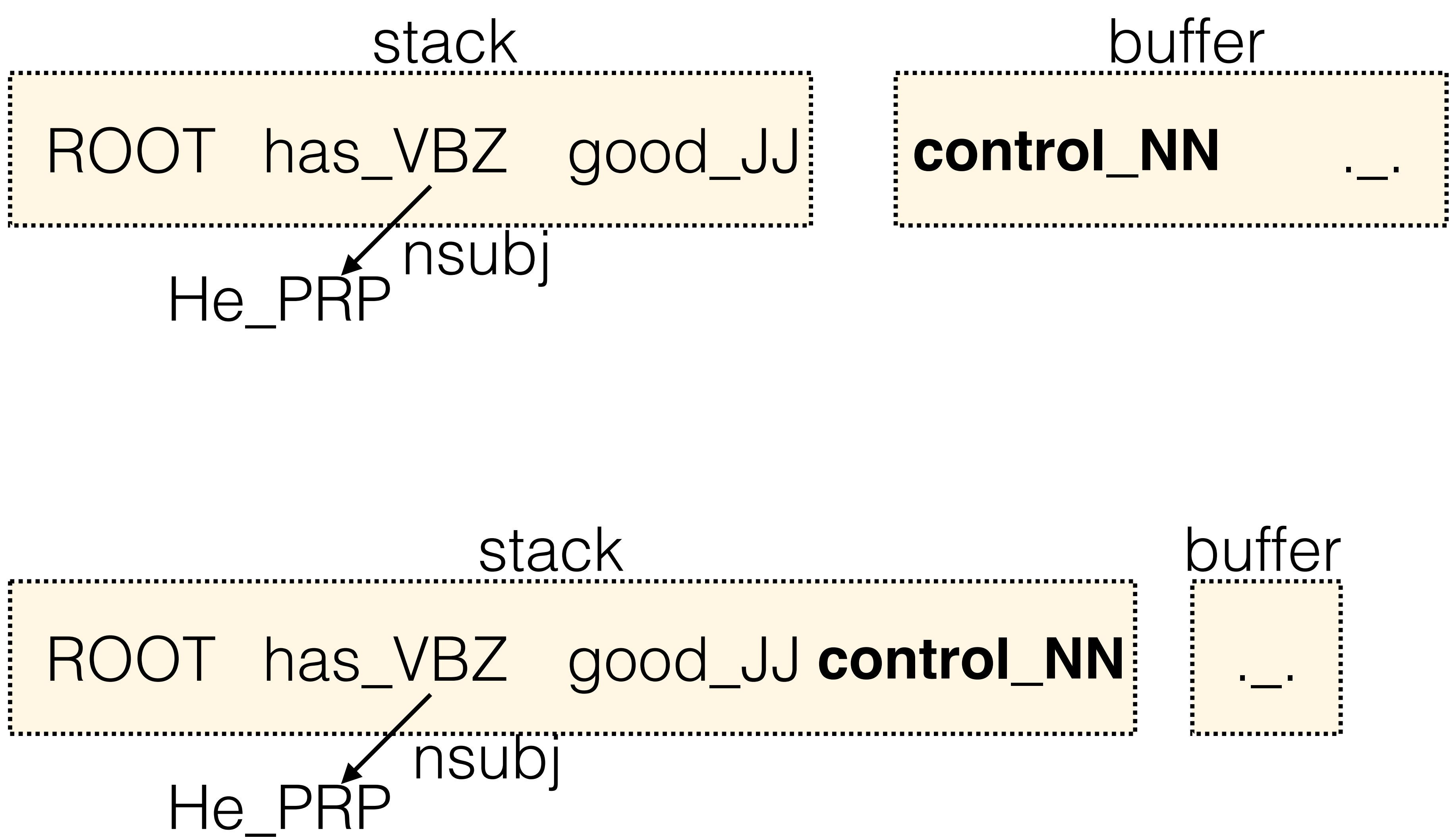
LEFT-ARC (l)


stack
buffer
ROOT has_VBZ good_JJ
control_NN ._.
nsubj
He_PRP
stack
buffer
ROOT good_JJ
control_NN ._.
has_VBZ
nsubj
He_PRP

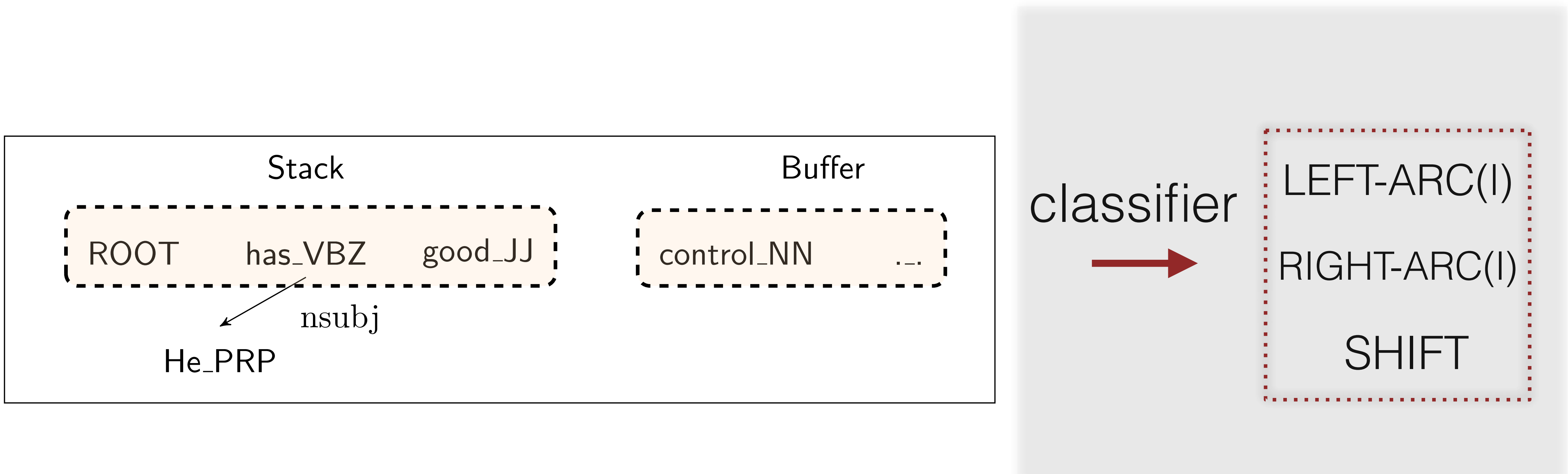
RIGHT-ARC (l)


stack
buffer
ROOT has_VBZ **good_JJ**
control_NN ._.
nsubj
He_PRP
stack
buffer
ROOT has_VBZ
control_NN ._.
nsubj
He_PRP **good_JJ**

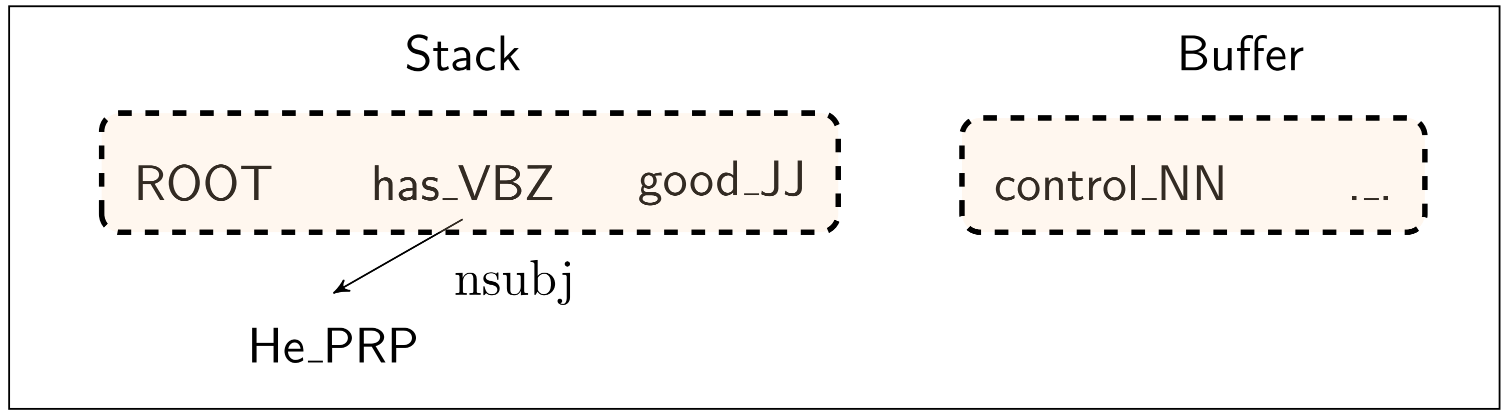
SHIFT


stack
buffer
ROOT has_VBZ good_JJ
control_NN ._.
nsubj
He_PRP
stack
buffer
ROOT has_VBZ good_JJ control_NN
._.
nsubj
He_PRP

Greedy Transition-based Parsing



Traditional Features

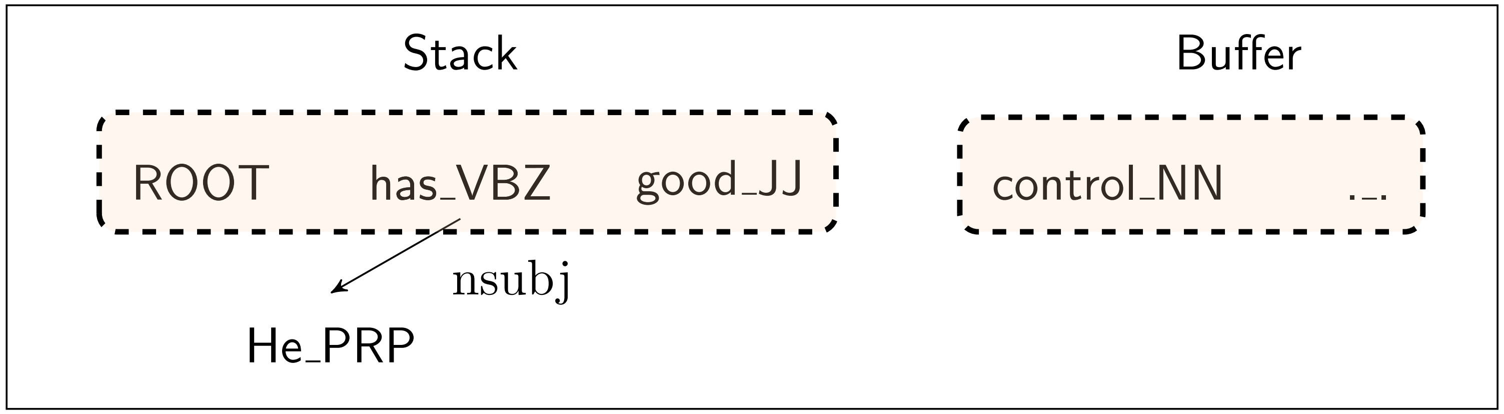


binary, sparse
dim $=10^6 \sim 10^7$

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

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

Traditional Features



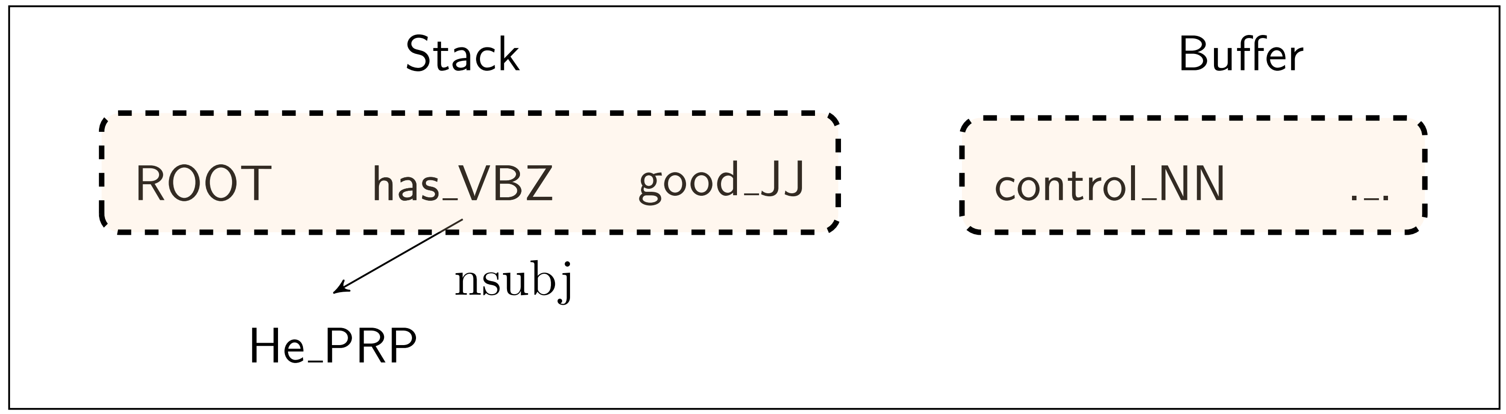
binary, sparse
dim $=10^6 \sim 10^7$

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

Indicator features

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Traditional Features



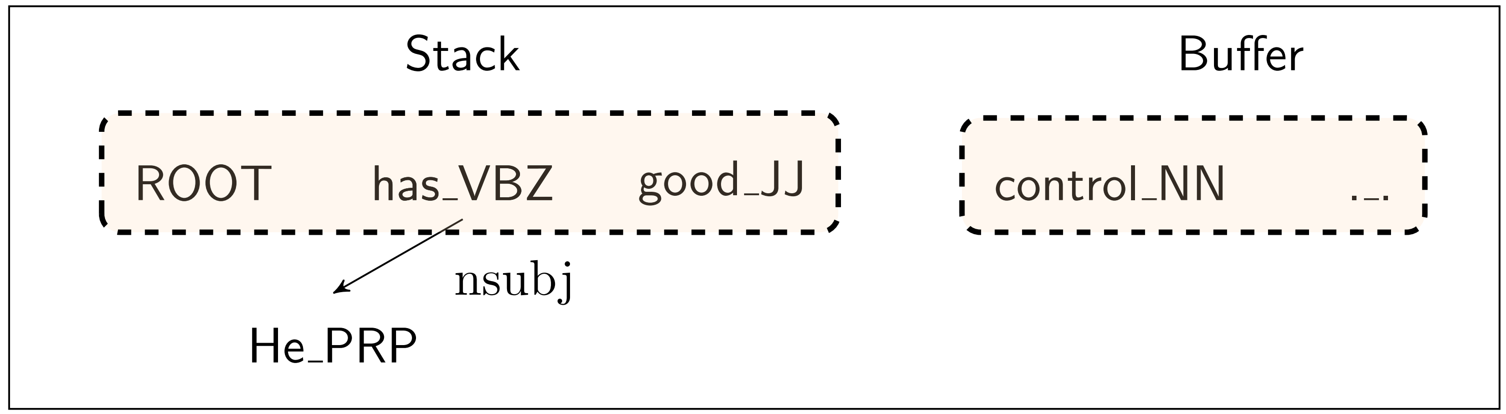
binary, sparse
dim $=10^6 \sim 10^7$

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

Indicator features

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Traditional Features



binary, sparse
dim $=10^6 \sim 10^7$

0	0	0	1	0	0	1	0	…	0	0	1	0

word part-of-speech tag

Indicator features

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$

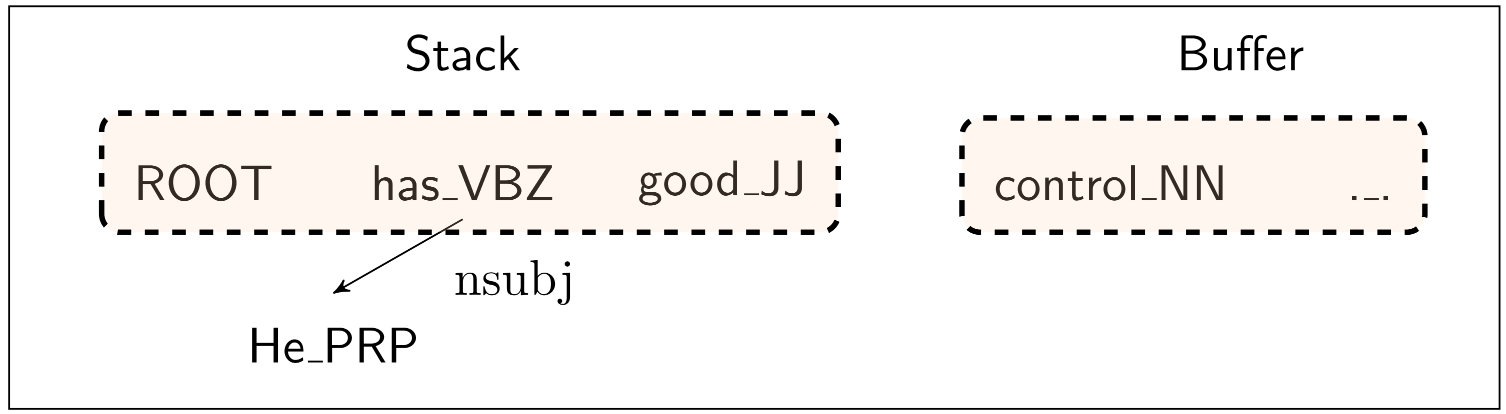
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$

$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$

$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

dep. label

Traditional Features



binary, sparse
dim $=10^6 \sim 10^7$

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

Indicator features

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

leftmost child

Indicator Features Revisited

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

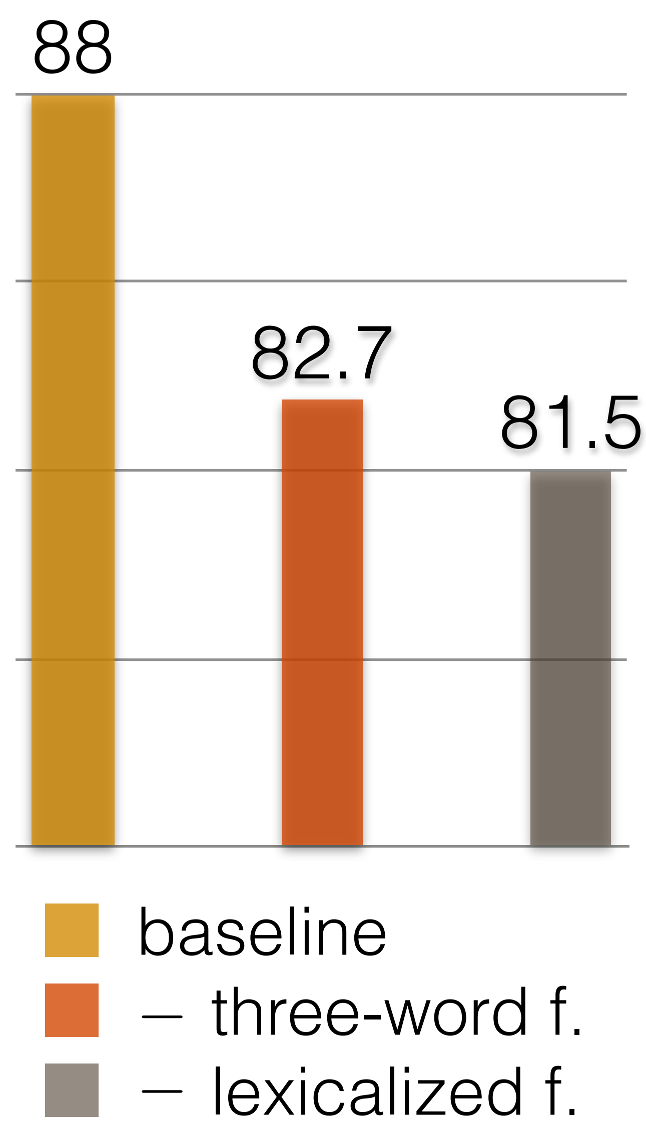
Indicator Features Revisited

- Problem #1: sparse
  - lexicalized features
  - high-order interaction features

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Indicator Features Revisited

- Problem #1: sparse
  - lexicalized features
  - high-order interaction features



$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Indicator Features Revisited

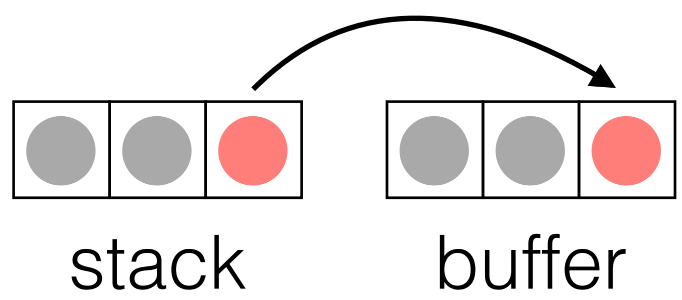
- Problem #1: sparse

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Indicator Features Revisited

- Problem #1: sparse
- Problem #2: incomplete

Unavoidable in hand-crafted feature templates.



~~RIGHT-ARC~~

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Indicator Features Revisited

- Problem #1: sparse
- Problem #2: incomplete

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

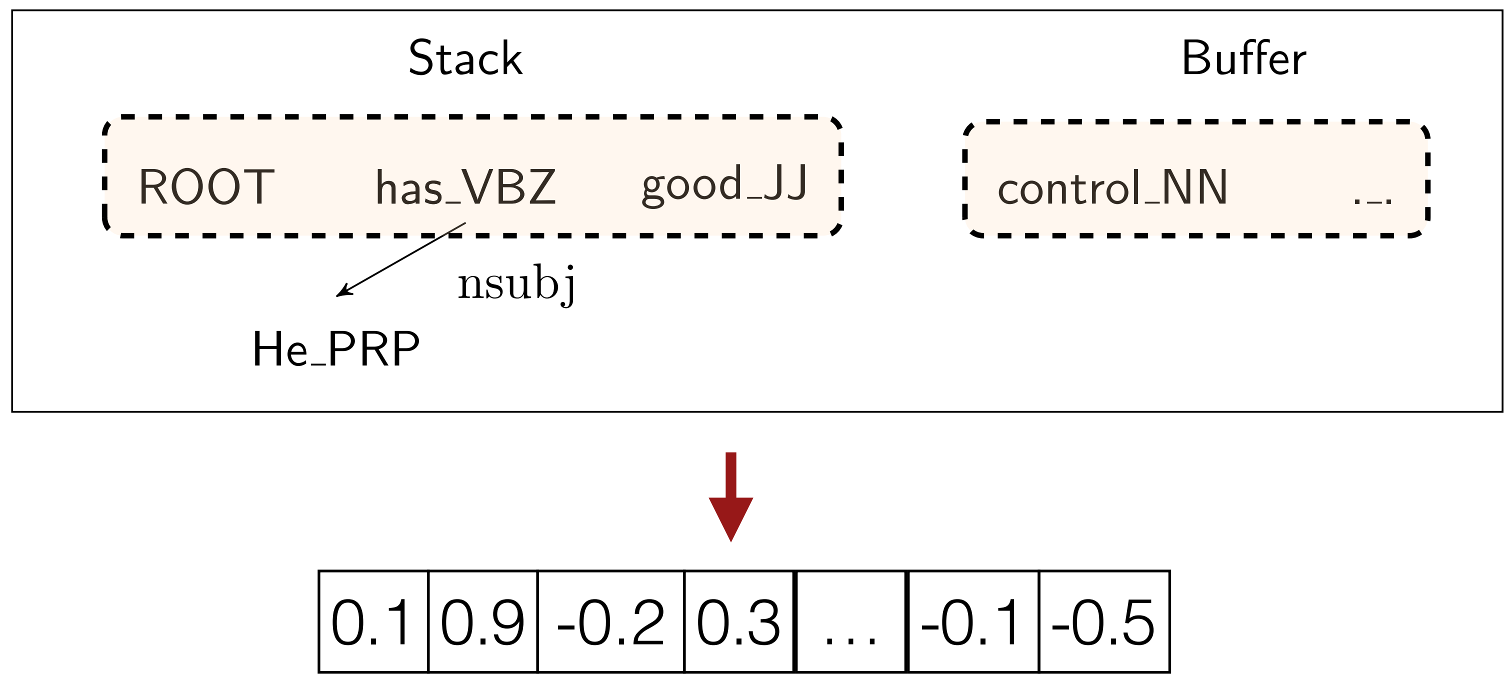
Indicator Features Revisited

- Problem #1: sparse
- Problem #2: incomplete
- Problem #3: computationally expensive

More than 95% of parsing time is consumed by feature computation.

$$s_2.w = \text{has} \wedge s_2.t = \text{VBZ}$$
$$s_1.w = \text{good} \wedge s_1.t = \text{JJ} \wedge b_1.w = \text{control}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Indicator Features Revisited

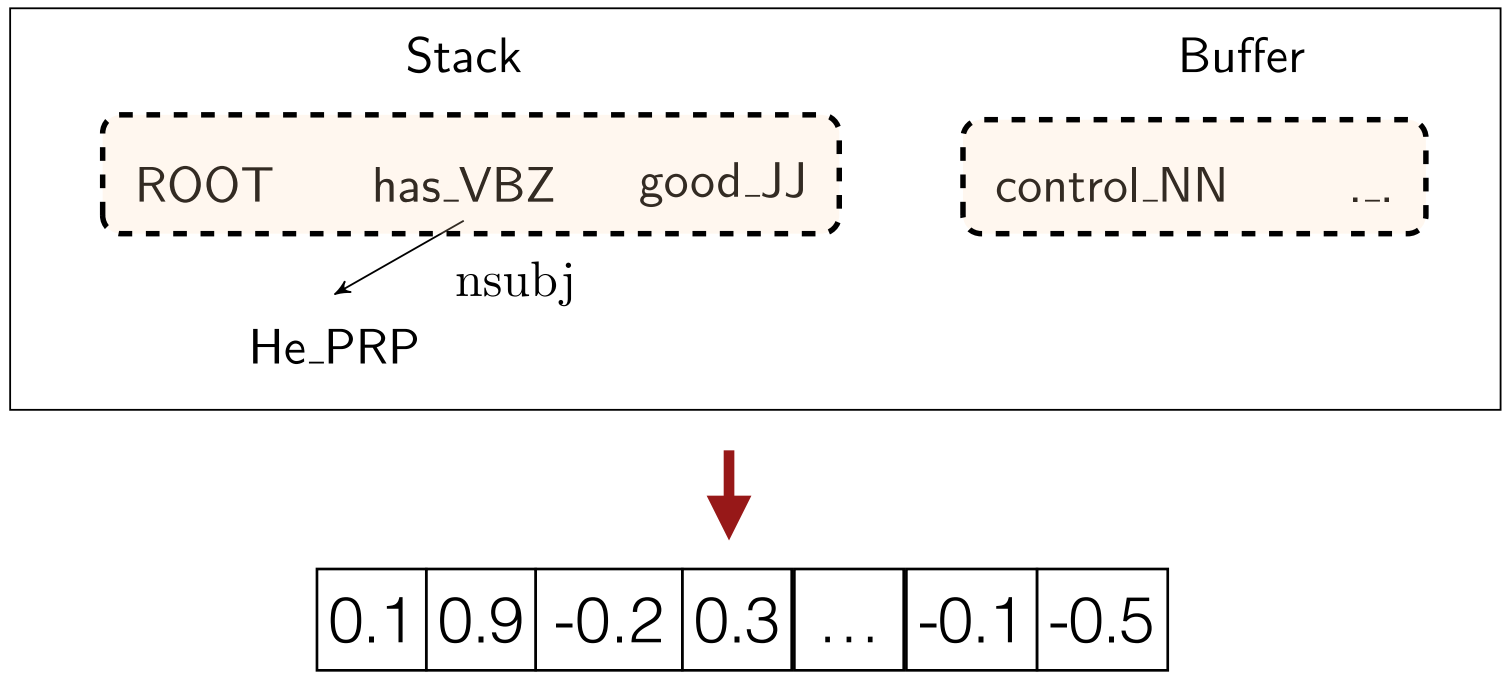


dense
dim = 200

Our Solution: Neural Networks!

Learn a **dense** and **compact** feature representation

The Challenge



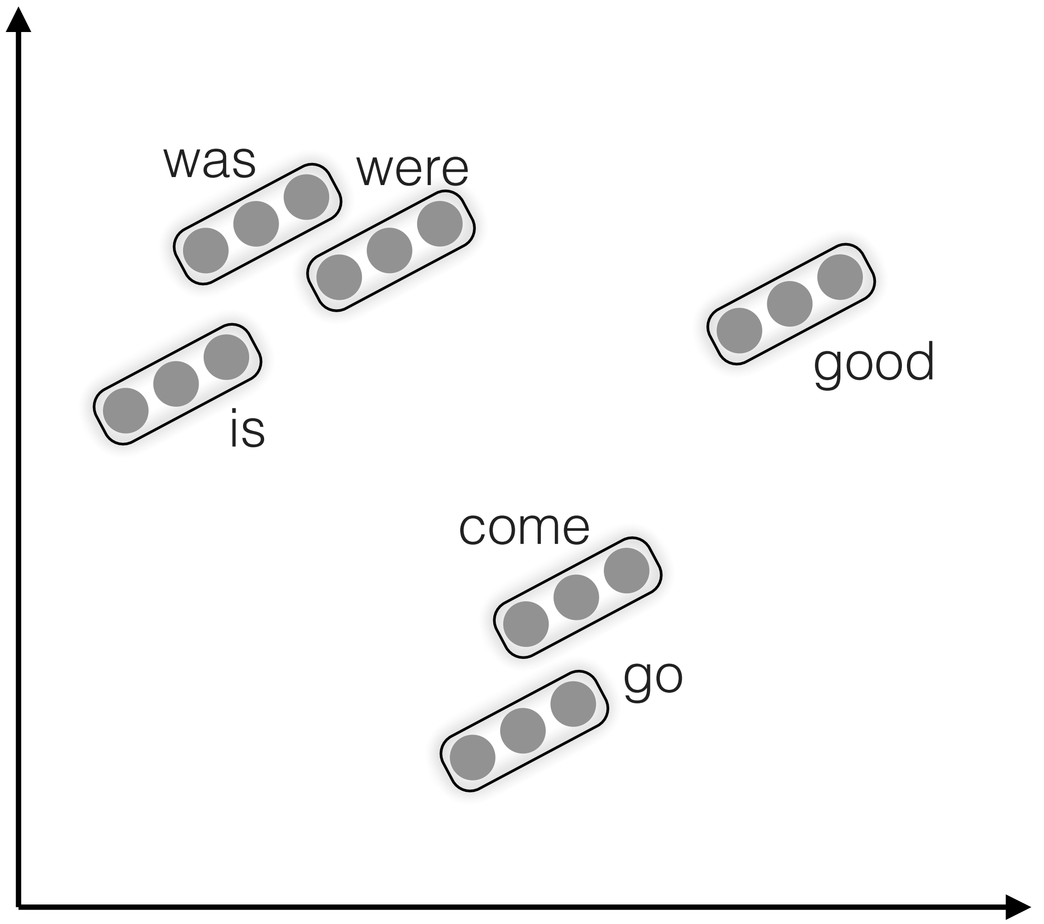
dense
dim = 200

- How to encode all the available information?
- How to model high-order features?

Distributed Representations

Distributed Representations

- We represent each word as a d-dimensional dense vector (i.e., word embeddings).
    - Similar words expect to have close vectors.



Distributed Representations

- We represent each word as a d-dimensional dense vector (i.e., word embeddings).
  - Similar words expect to have close vectors.
- Meanwhile, part-of-speech tags and dependency labels are also represented as d-dimensional vectors.
  - POS and dependency embeddings.
  - The smaller discrete sets also exhibit many semantical similarities.

Distributed Representations

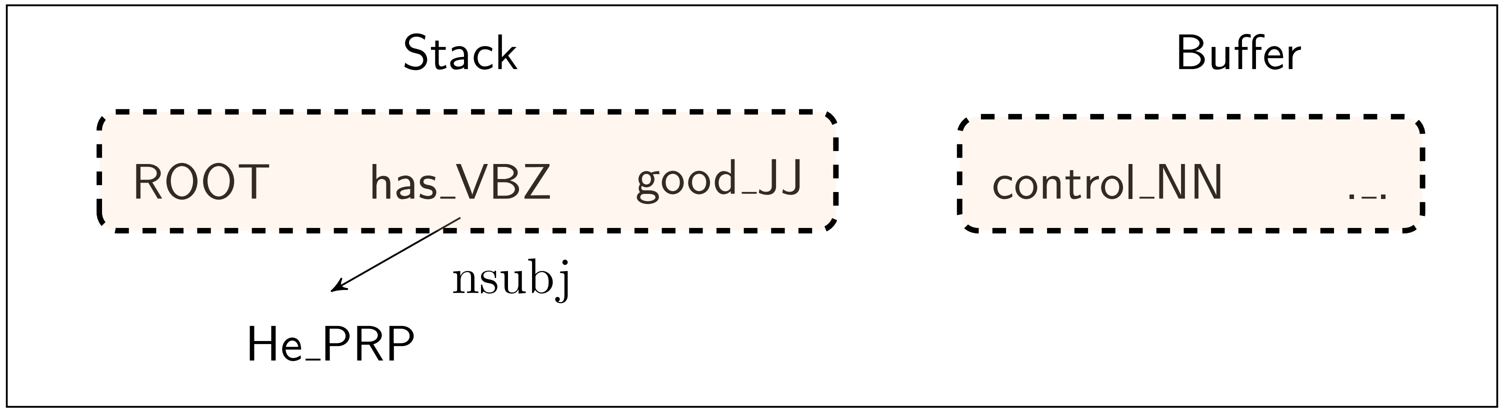
- We represent each word as a d-dimensional dense vector (i.e., word embeddings).
    - Similar words expect to have close vectors.
- Meanwhile, part-of-speech tags and dependency labels are also represented as d-dimensional vectors.
    - POS and dependency embeddings.
    - The smaller discrete sets also exhibit many semantical similarities.

NNS (plural noun) should be close to NN (singular noun).

num (numerical modifier) should be close to amod (adjective modifier).

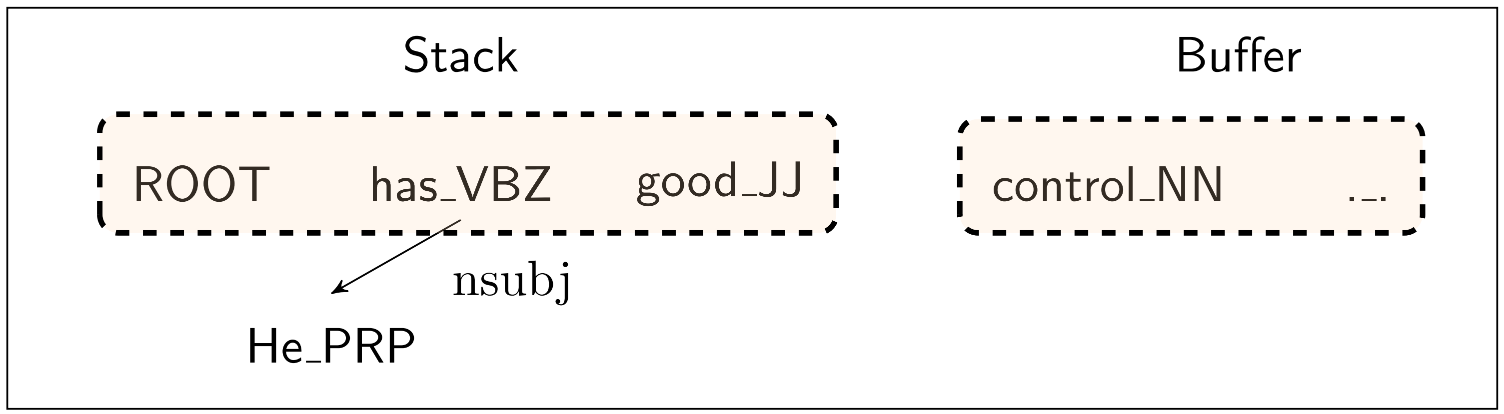
Extracting Tokens from Configuration

- We extract a set of tokens based on the positions:



Extracting Tokens from Configuration

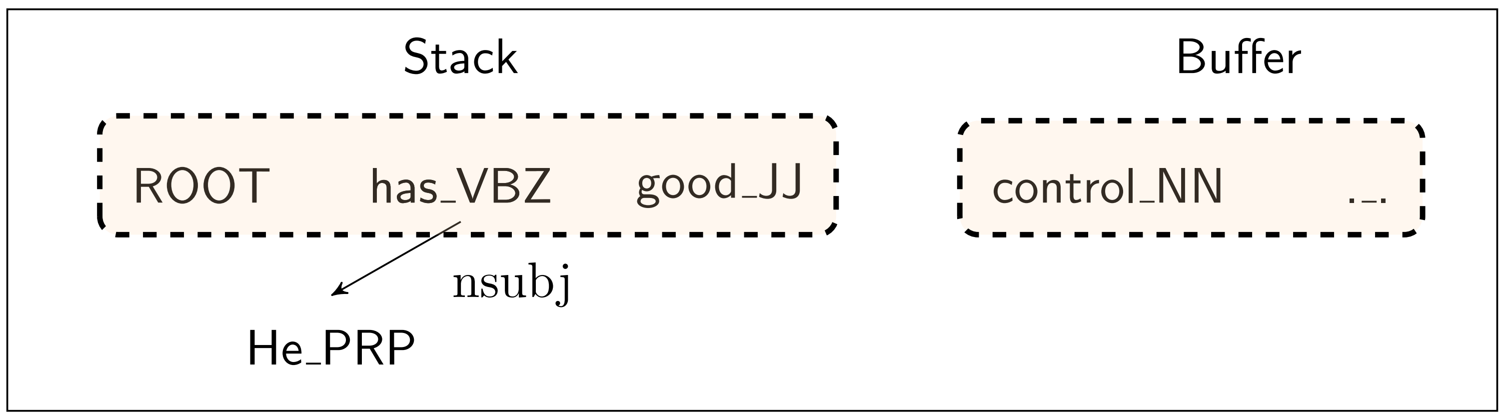
- We extract a set of tokens based on the positions:



s_1
s_2
b_1
$lc(s_1)$
$rc(s_1)$
$lc(s_2)$
$rc(s_2)$
...

Extracting Tokens from Configuration

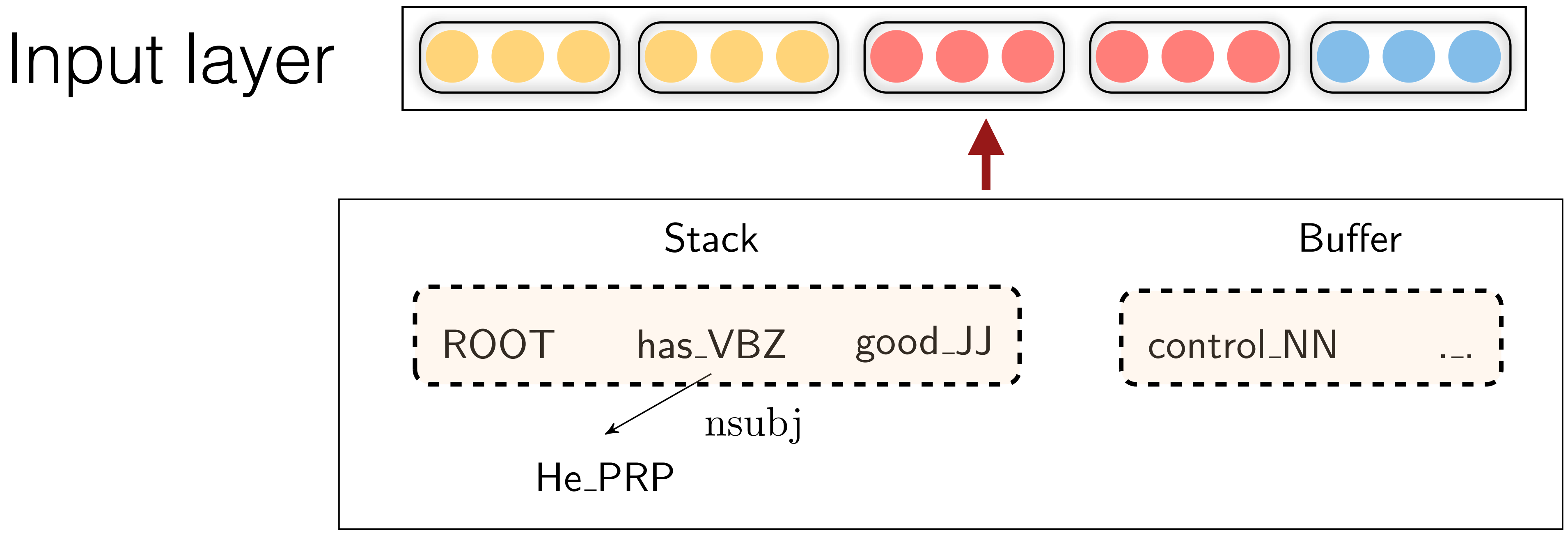
- We extract a set of tokens based on the positions:



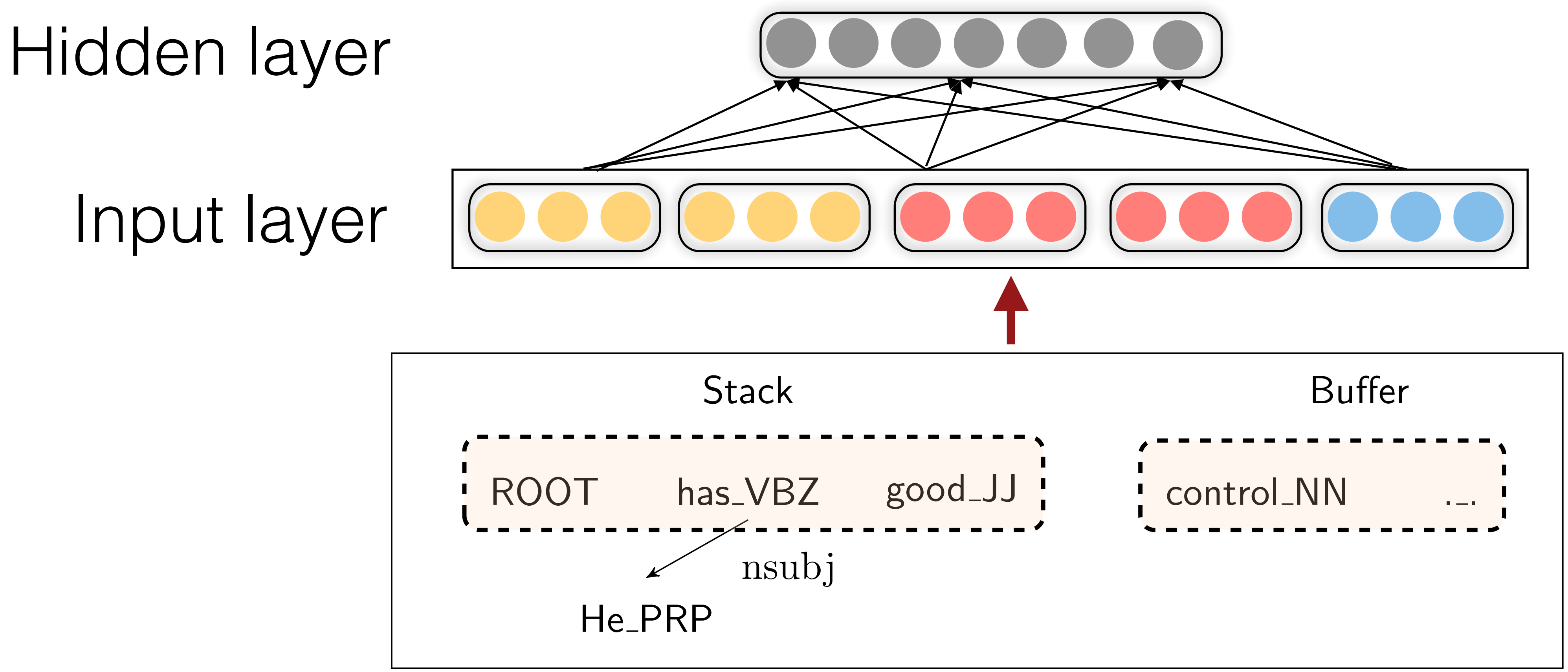
	word	POS	dep.
s_1	good	JJ	
s_2	has	VBZ	
b_1	control	NN	
$lc(s_1)$	$\emptyset$	$\emptyset$	$\emptyset$
$rc(s_1)$	$\emptyset$	$\emptyset$	$\emptyset$
$lc(s_2)$	He	PRP	nsubj
$rc(s_2)$	$\emptyset$	$\emptyset$	$\emptyset$
...	...	...	...

Model Architecture

Model Architecture

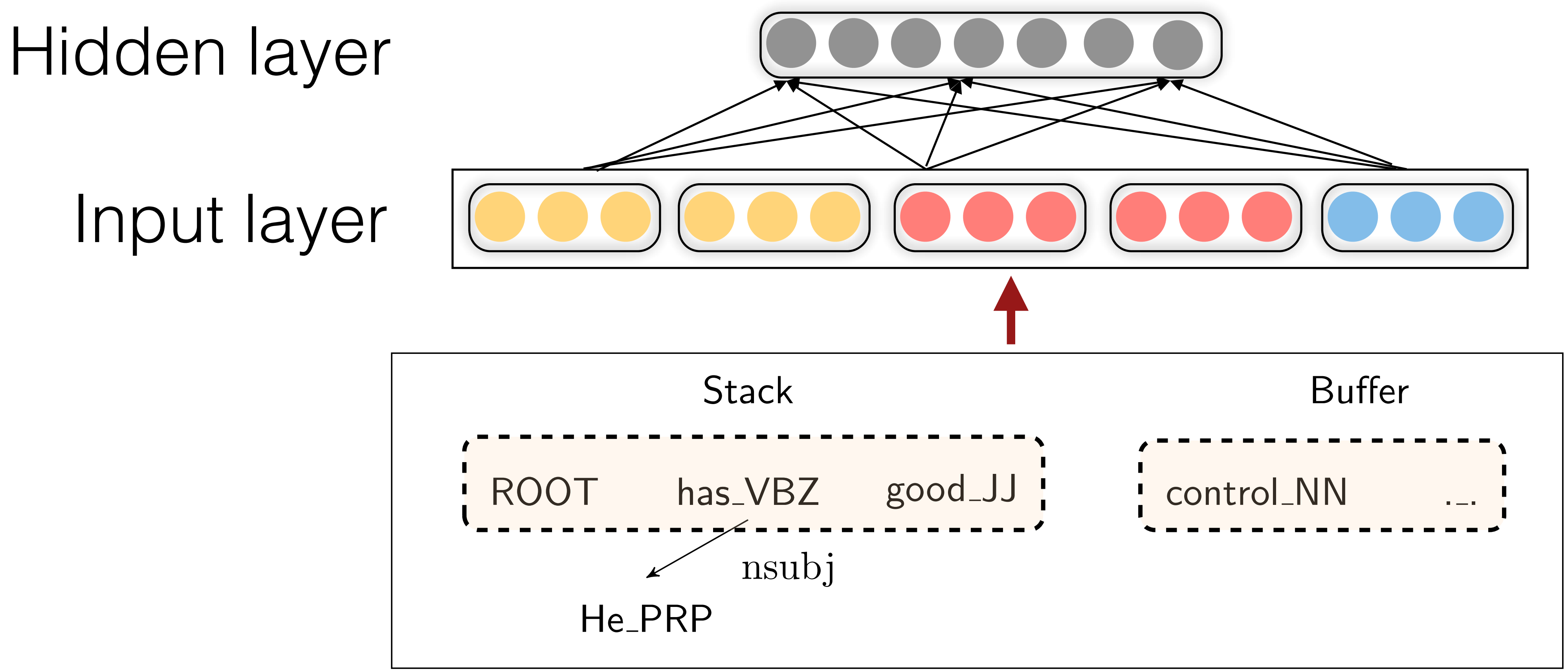


Model Architecture



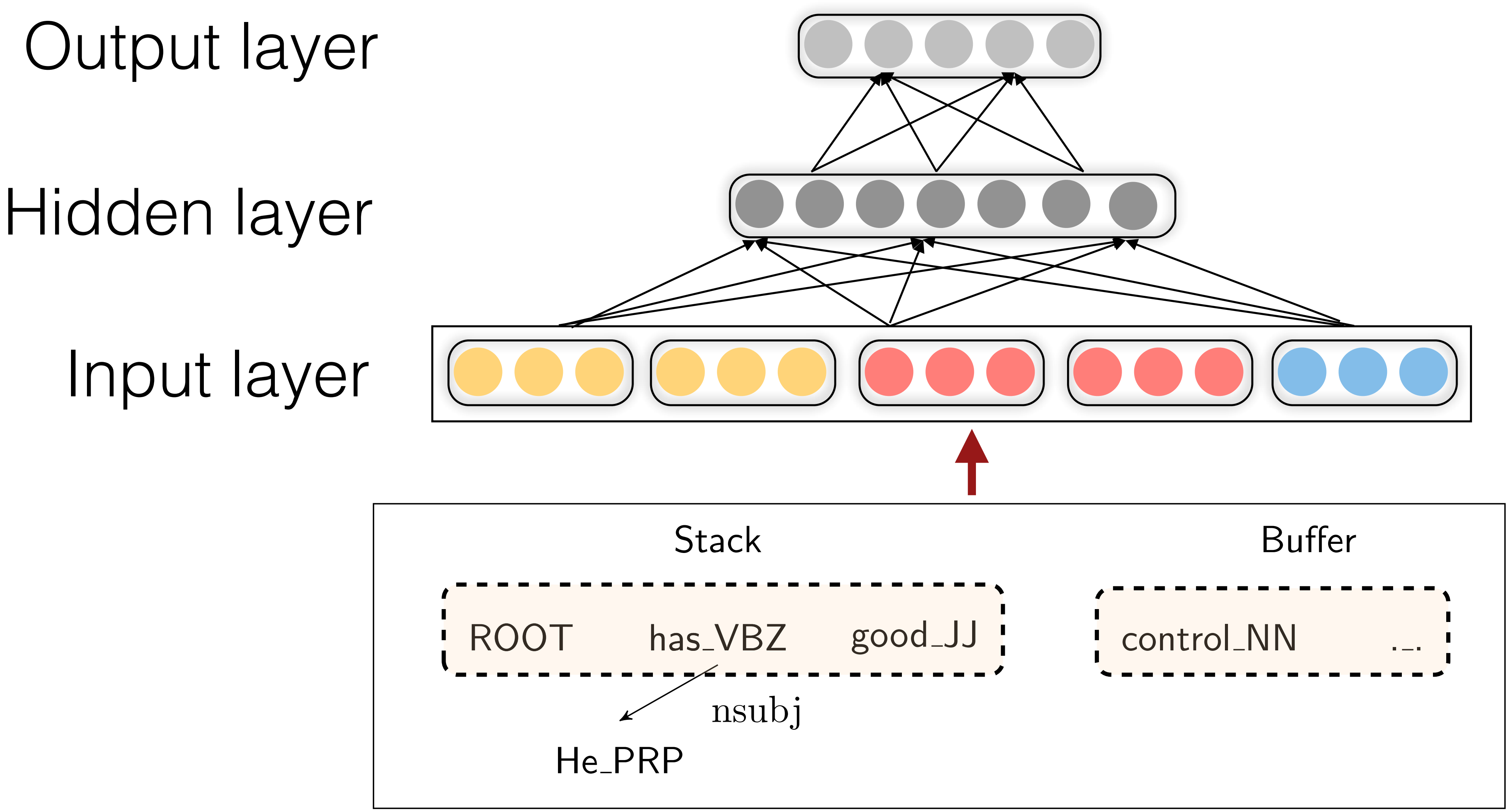
Model Architecture

Cube activation function: $g(x) = x^3$

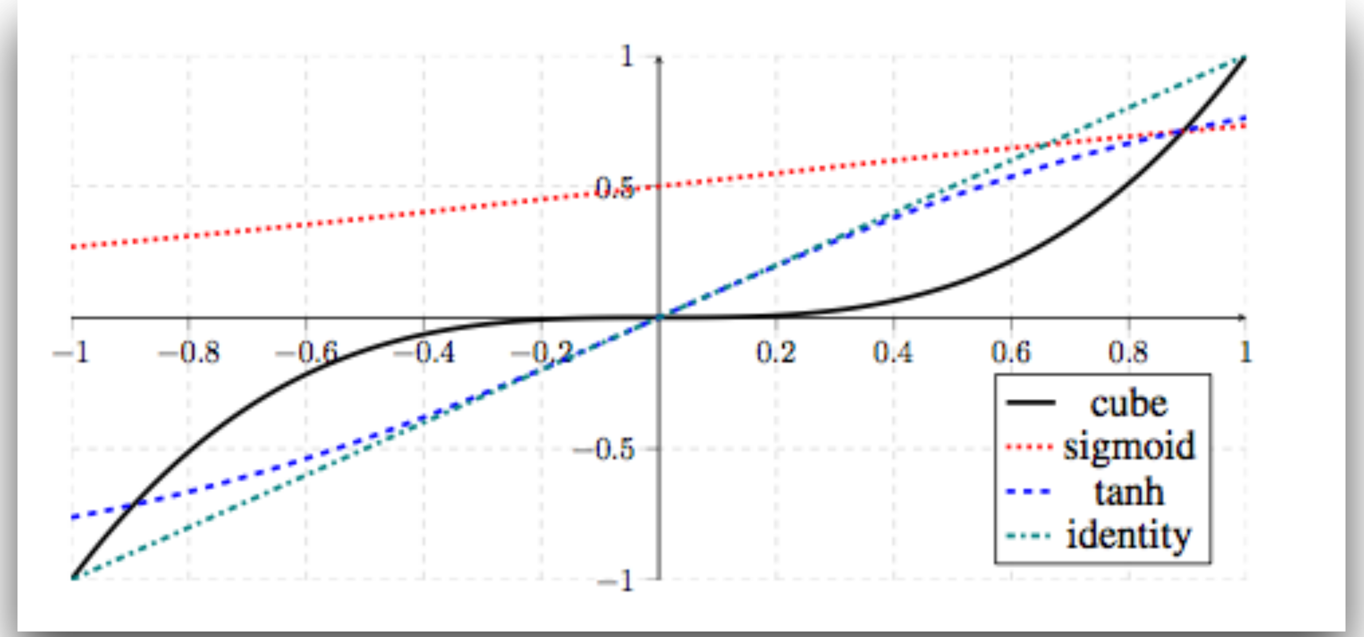


Model Architecture

Softmax probabilities



Cube Activation Function



$$g(w_1 x_1 + \ldots + w_m x_m + b) =$$
$$\sum_{i,j,k} (w_i w_j w_k) x_i x_j x_k + \sum_{i,j} b(w_i w_j) x_i x_j \ldots$$

Better capture the **interaction** terms!

Training

- Generating training examples using a fixed oracle.
- **Training objective**: cross entropy loss
- Back-propagation to all embeddings.
- **Initialization**:
  - Word embeddings from pre-trained word vectors.
  - Random initialization for others.

Parsing Speed-up

- Pre-computation trick:

	word	POS	dep.
s_1	good	JJ	
s_2	has	VBZ	
b_1	control	NN	
$lc(s_1)$	$\emptyset$	$\emptyset$	$\emptyset$
$rc(s_1)$	$\emptyset$	$\emptyset$	$\emptyset$
$lc(s_2)$	He	PRP	nsubj
$rc(s_2)$	$\emptyset$	$\emptyset$	$\emptyset$
…	…	…	…

- If we have seen (s_1, good) many times in training set, we can pre-compute matrix multiplications before parsing — reducing multiplications to additions.
- 8 ~ 10 times faster.

Indicator vs. Dense Features

- Problem #1: sparse

Distributed representations can capture similarities.

Indicator vs. Dense Features

- Problem #1: sparse

Distributed representations can capture similarities.

- Problem #2: incomplete

We don't need to enumerate the combinations.
Cube non-linearity can learn combinations automatically.

Indicator vs. Dense Features

- Problem #1: sparse

Distributed representations can capture similarities.

- Problem #2: incomplete

We don't need to enumerate the combinations.
Cube non-linearity can learn combinations automatically.

- Problem #3: computationally expensive

String concatenation + look-up in a big table $\Longrightarrow$ matrix operations. Pre-computation trick can speed up.

Experimental Setup

- **Datasets**
  - English Penn Treebank (PTB)
  - Chinese Penn Treebank (CTB)
- **Representations**
  - CoNLL representations (CD) for PTB and CTB
  - Stanford Dependencies V3.3.0 (SD) for PTB
- **Part-of-speech tags**:
  - Stanford POS tagger for PTB (97.3% accuracy)
  - Gold tags for CTB

 (Collobert et al 2011, Mikolov et al 2013)

Details

- Embedding size = 50
- Hidden size = 200
- Use mini-batched AdaGrad for optimization ($\alpha = 0.01$)
- Use 0.5 dropout on hidden layer.

- Pre-trained word vectors:
  - C & W for English
  - Word2vec for Chinese

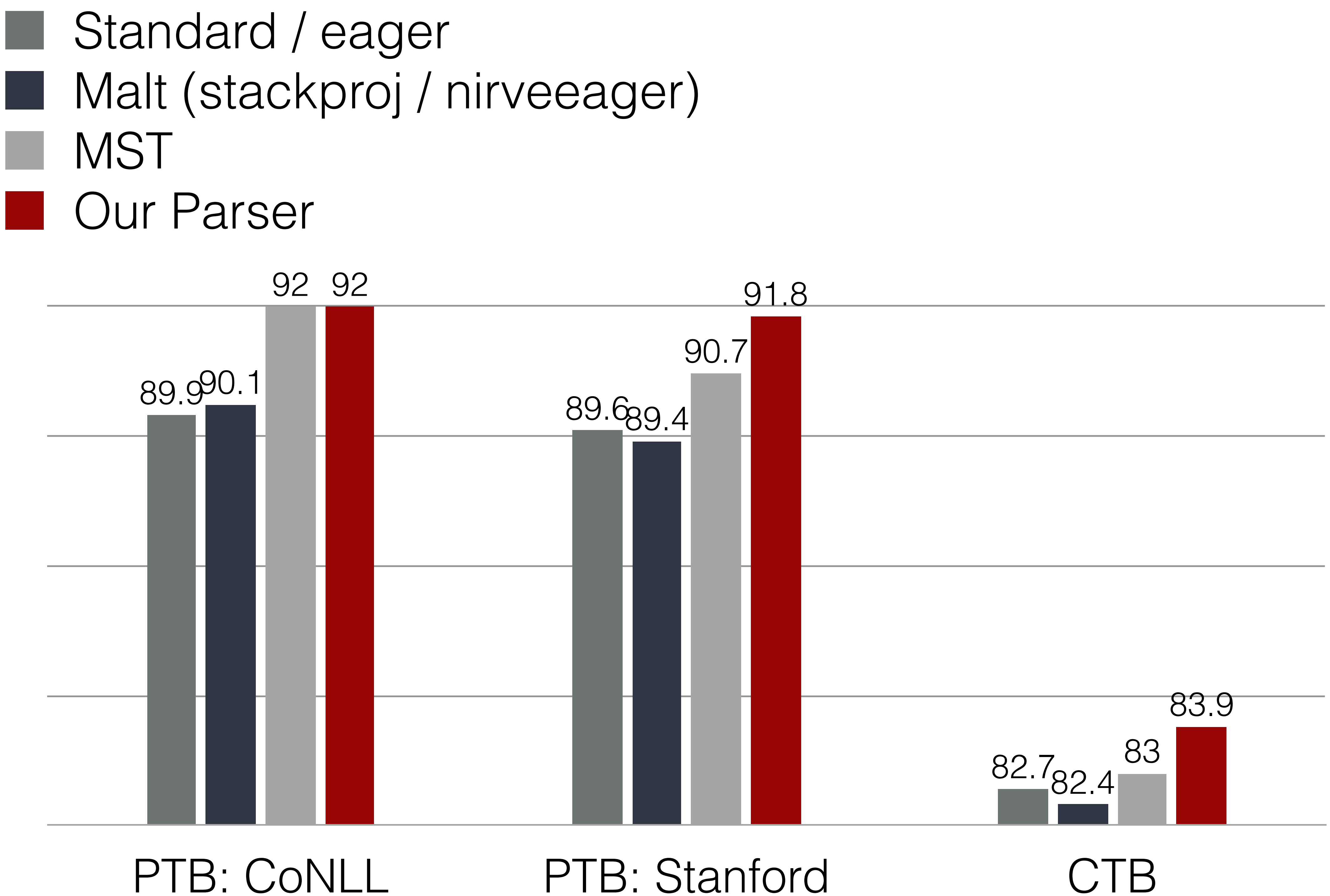
- We use a rich set of 18 tokens from the configuration.

(Nivre et al 2006, McDonald and Pereira 2006)

Baselines

- **Standard / eager**: our own implemented perceptron-based greedy parsers using arc-standard or arc-eager system, with a rich feature set from (Zhang and Nivre, 2011).
- **MaltParser**
  - two algorithms **stackproj** and **nivreeager**.
- **MSTParser**

Unlabeled Attachment Score (UAS)

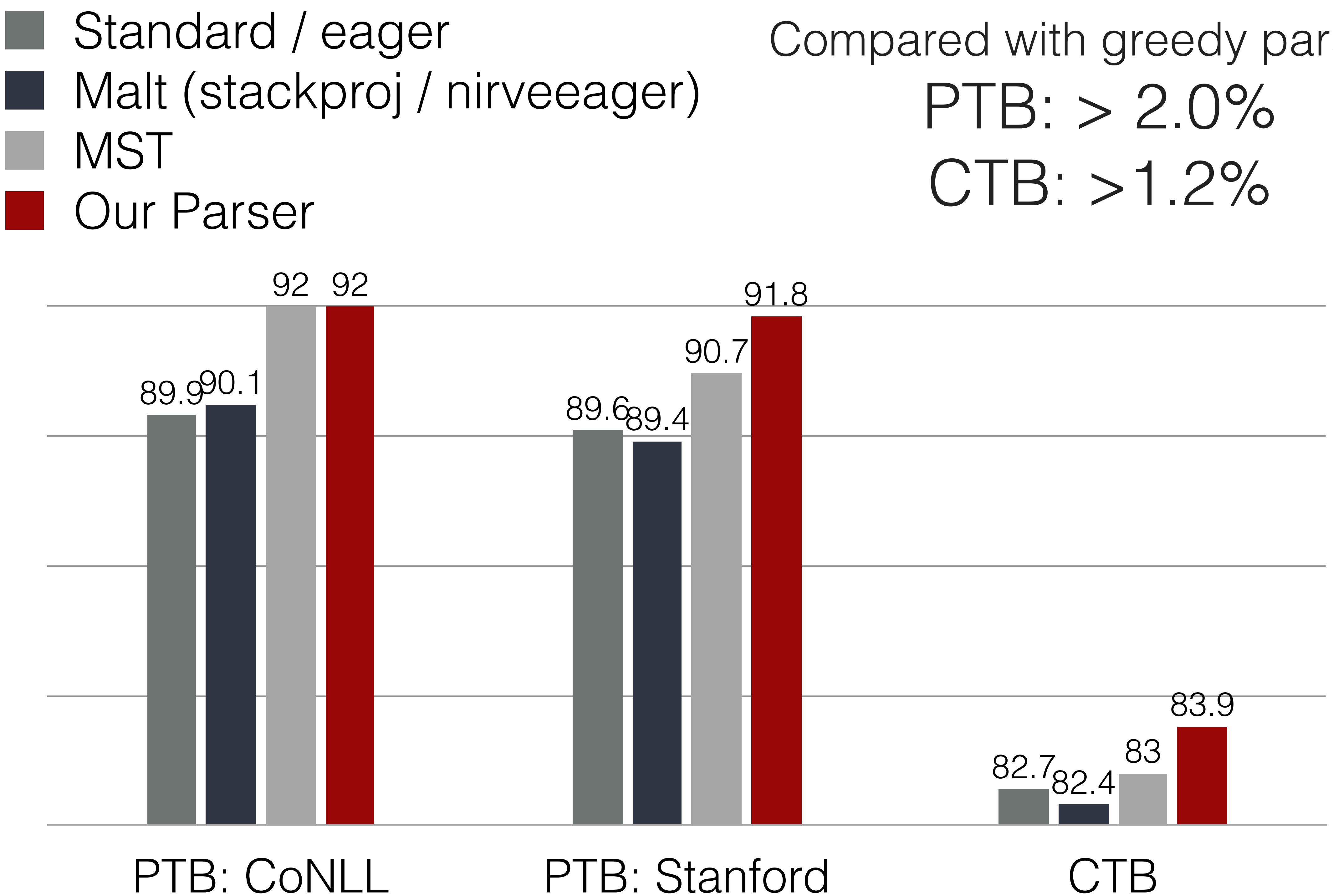


Unlabeled Attachment Score (UAS)

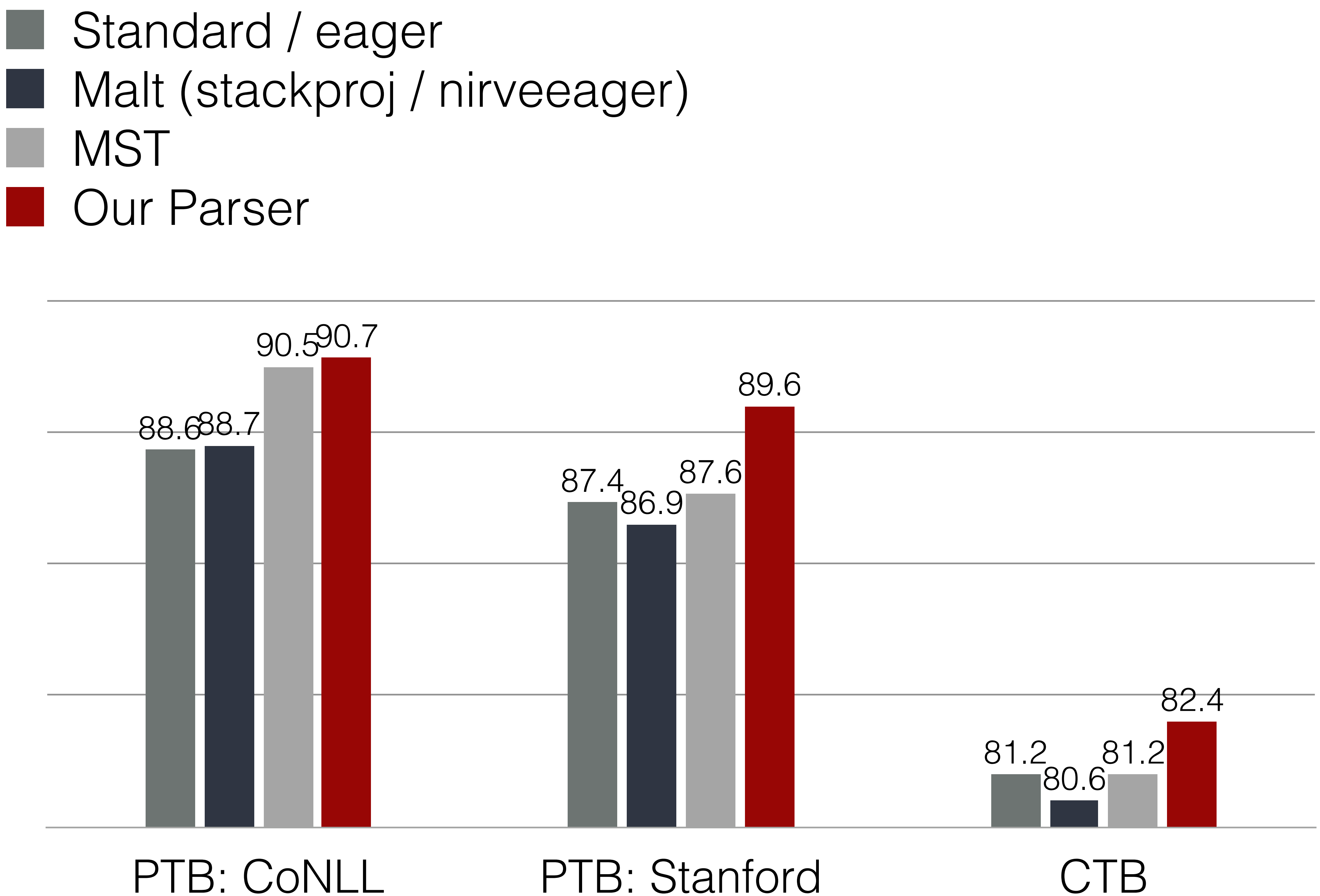
Compared with greedy parsers,

PTB: > 2.0%

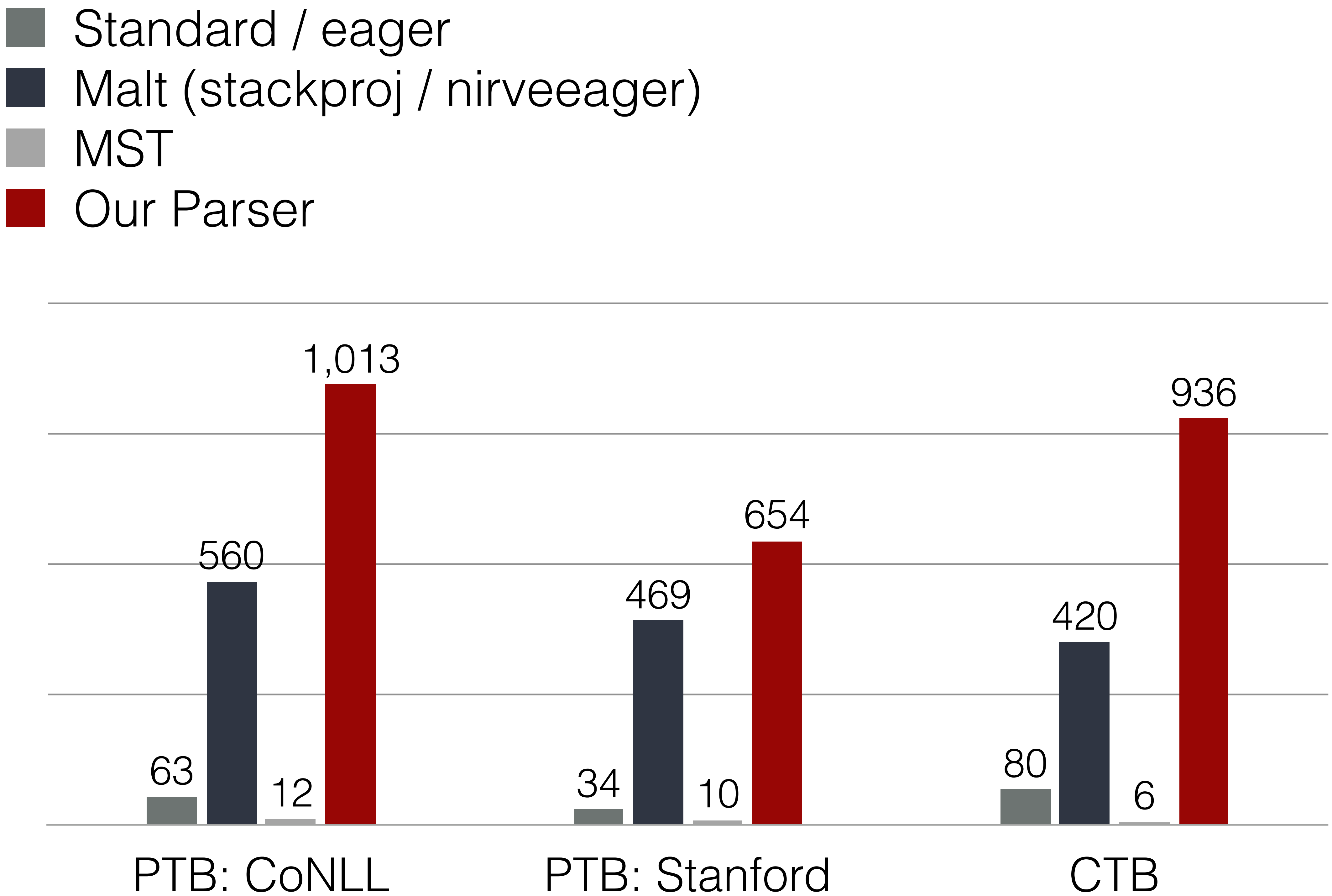
CTB: >1.2%



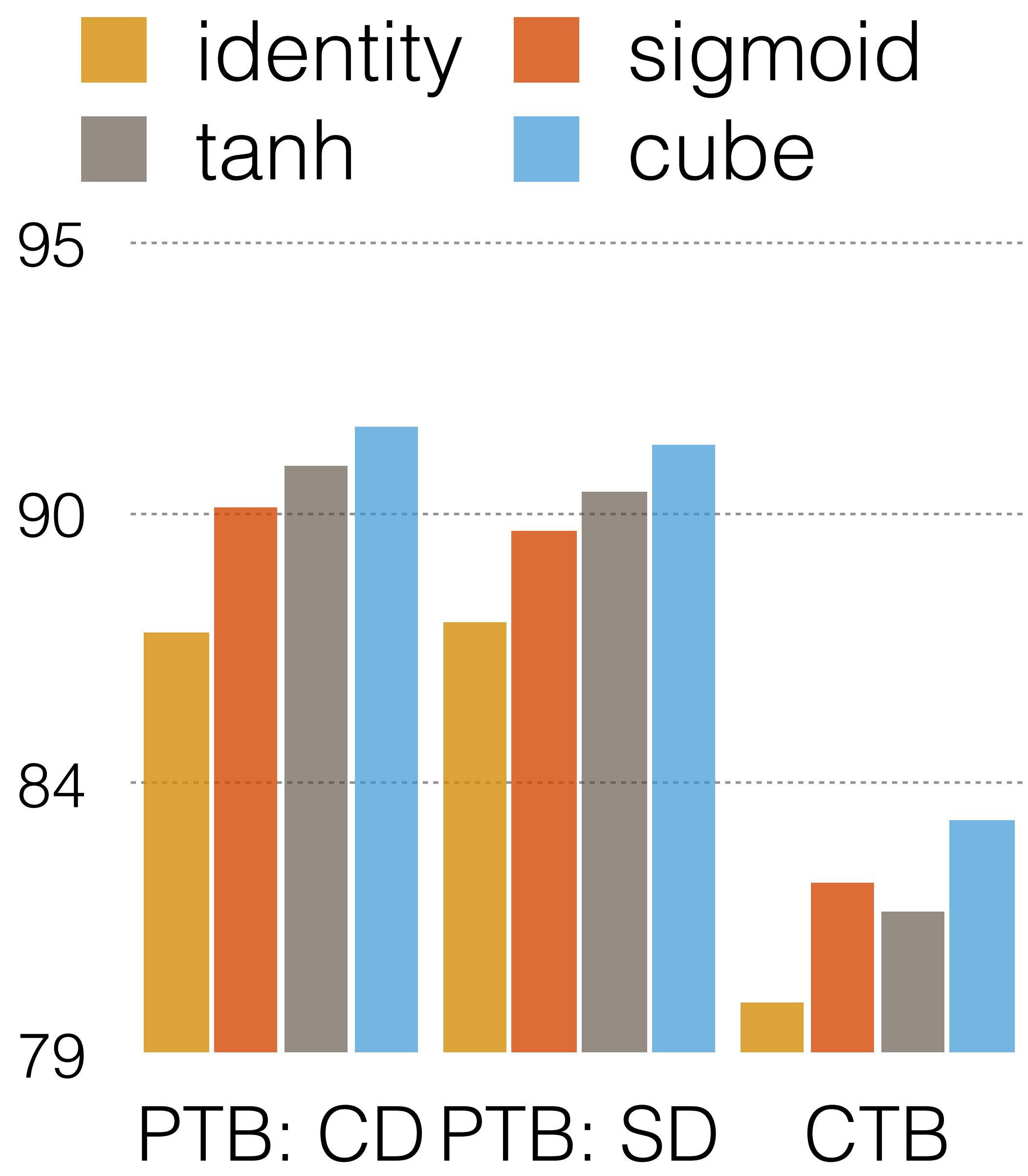
Labeled Attachment Score (LAS)



Parsing Speed (sent/s)



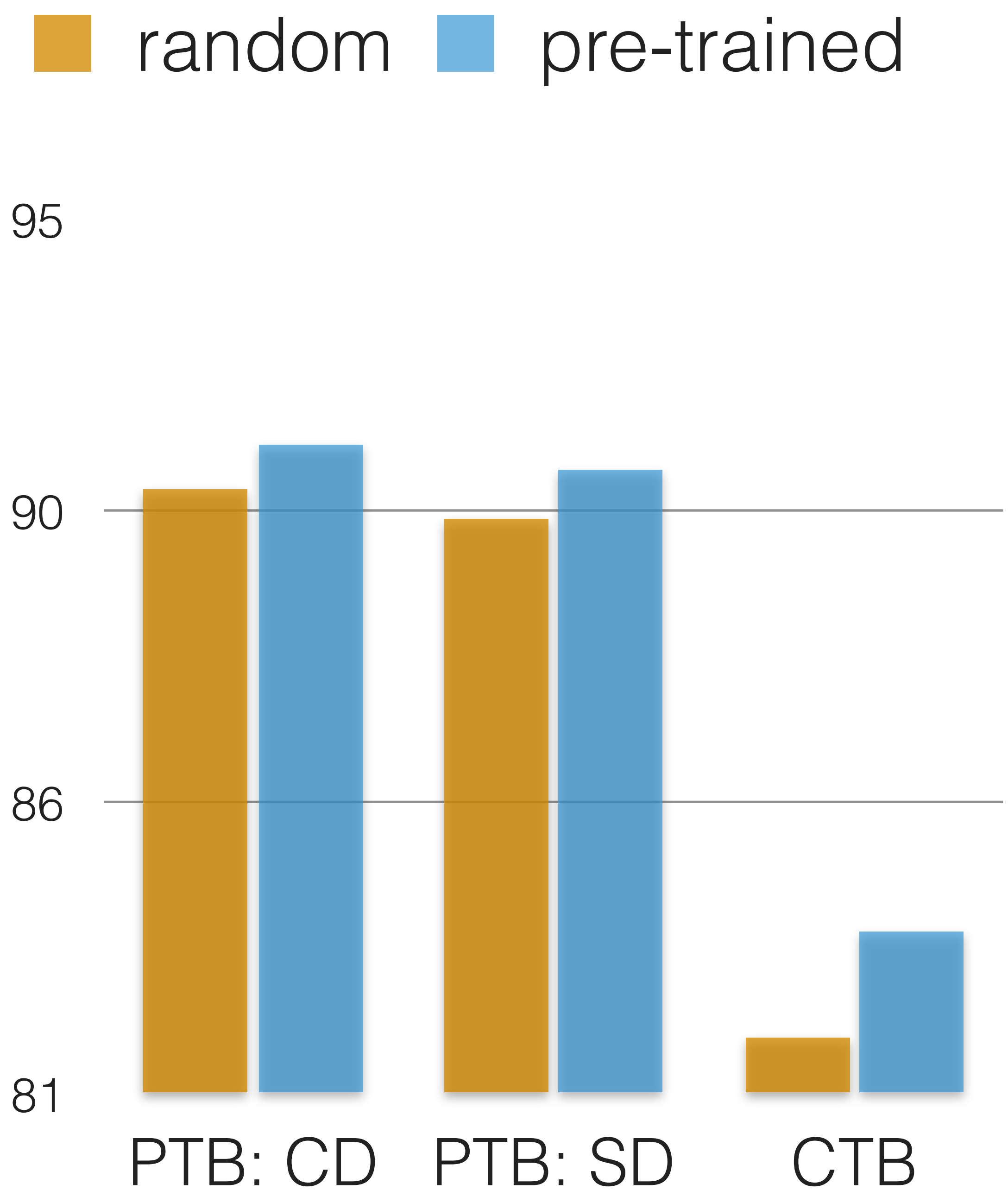
Cube Activation Function



Cube:

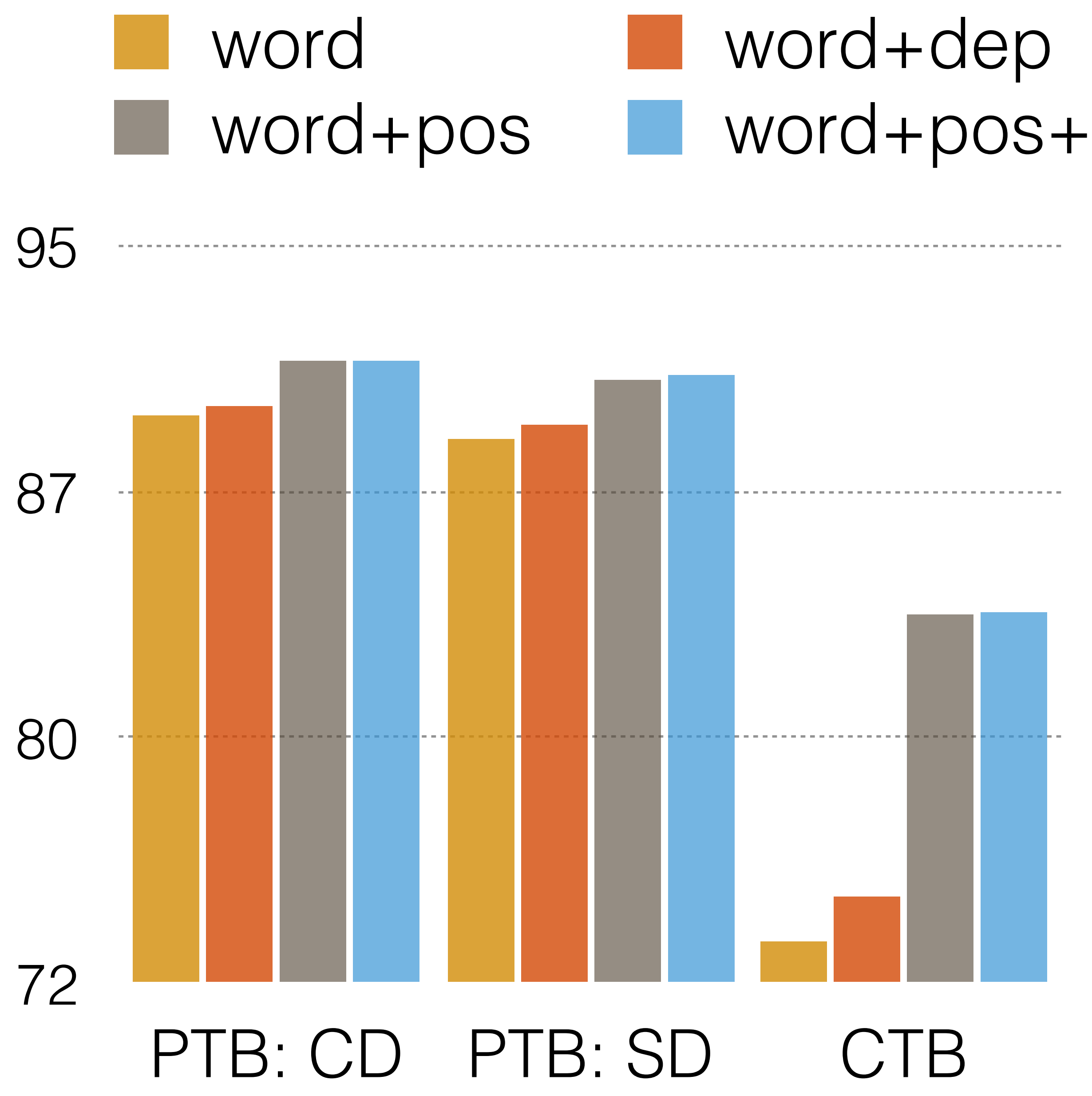
+0.8% ~ 1.2%

Pre-trained Word Vectors



- PTB: +0.7%
- CTB: +1.7%

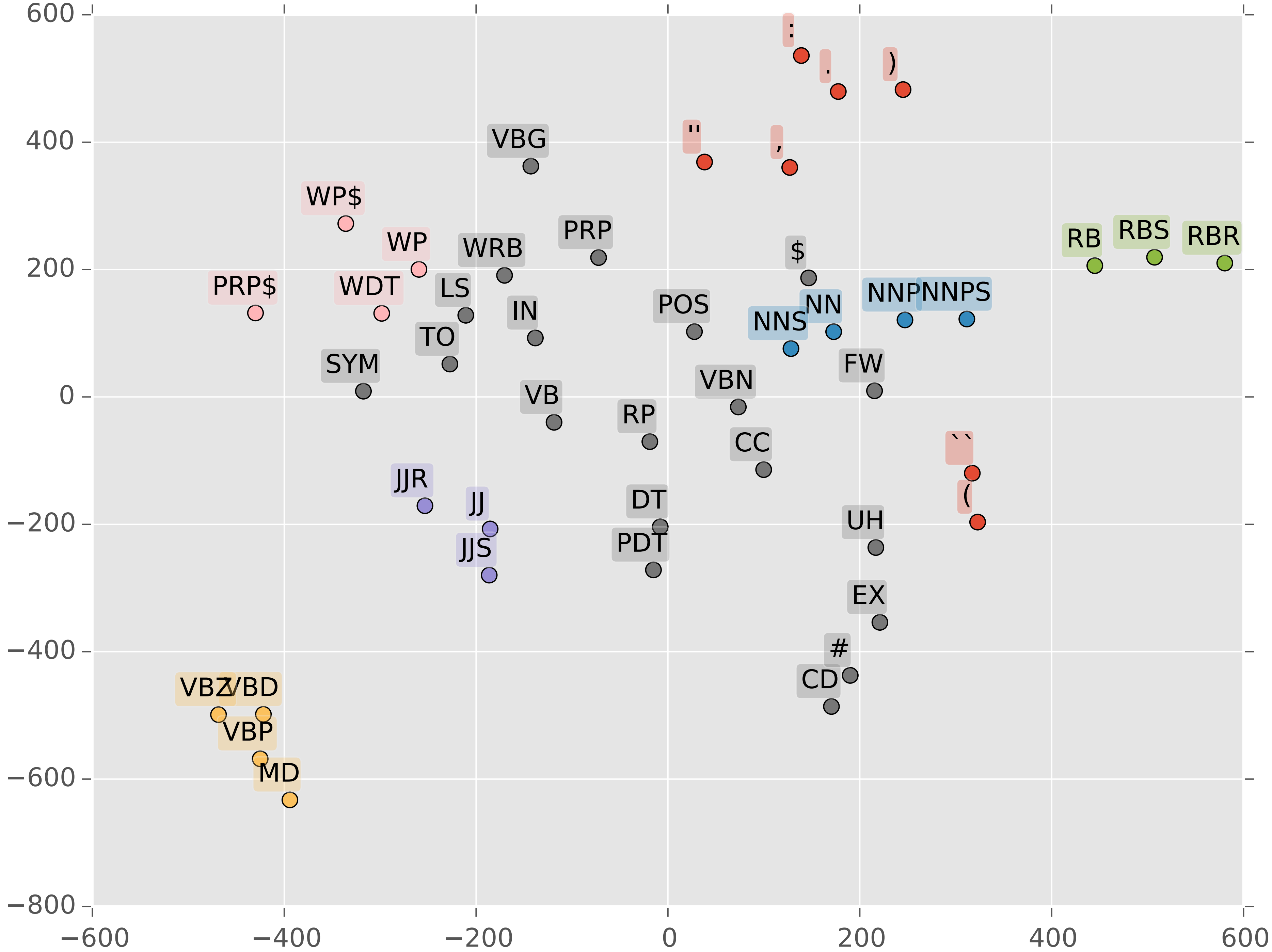
POS / Dependency Embeddings



- POS embeddings help a lot:
  - PTB: +1.7%
  - CTB: +10.2%
- Slight gain from dependency embeddings.

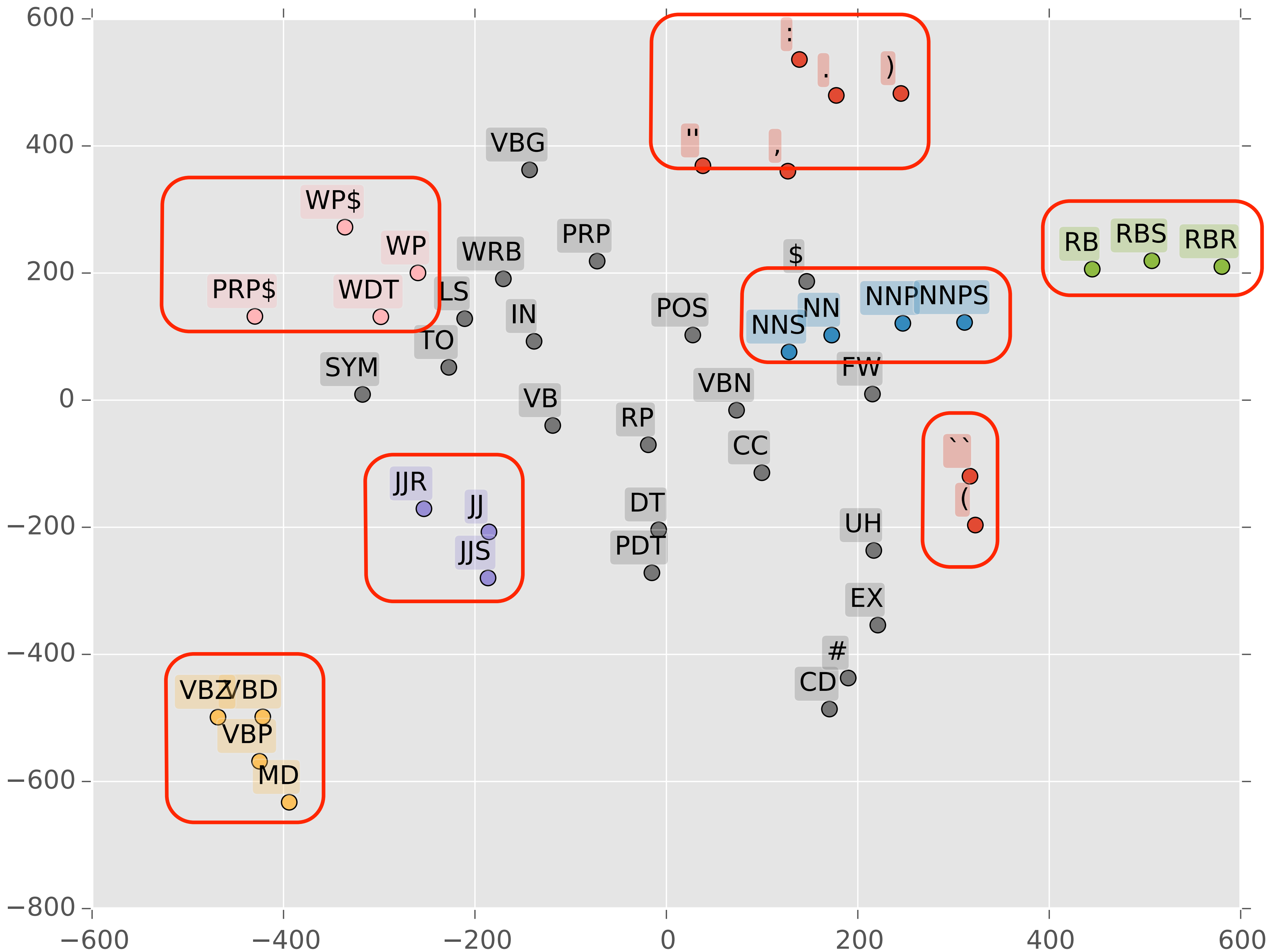
(van der Maaten and Hinton 2008)

POS Embeddings



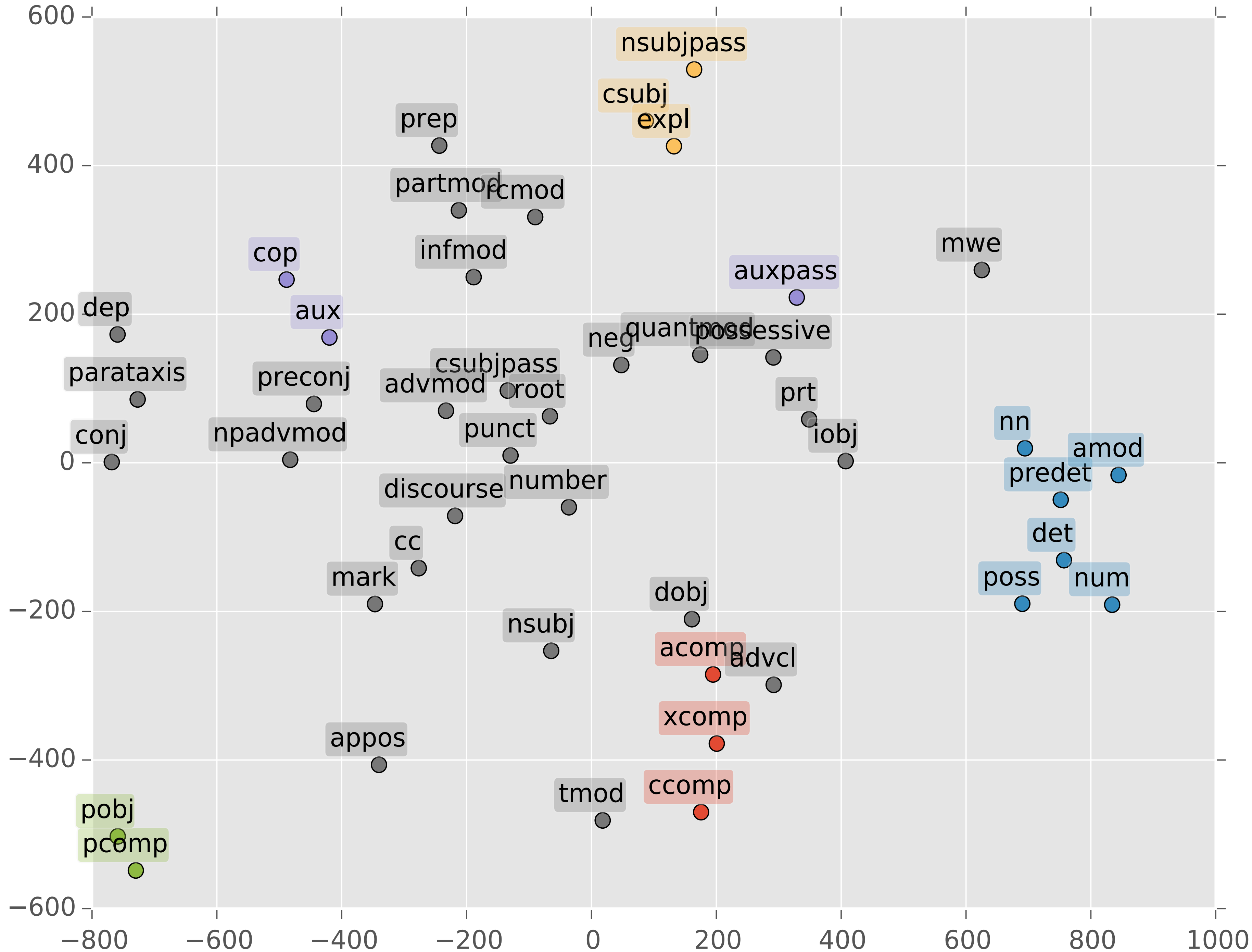
(van der Maaten and Hinton 2008)

POS Embeddings

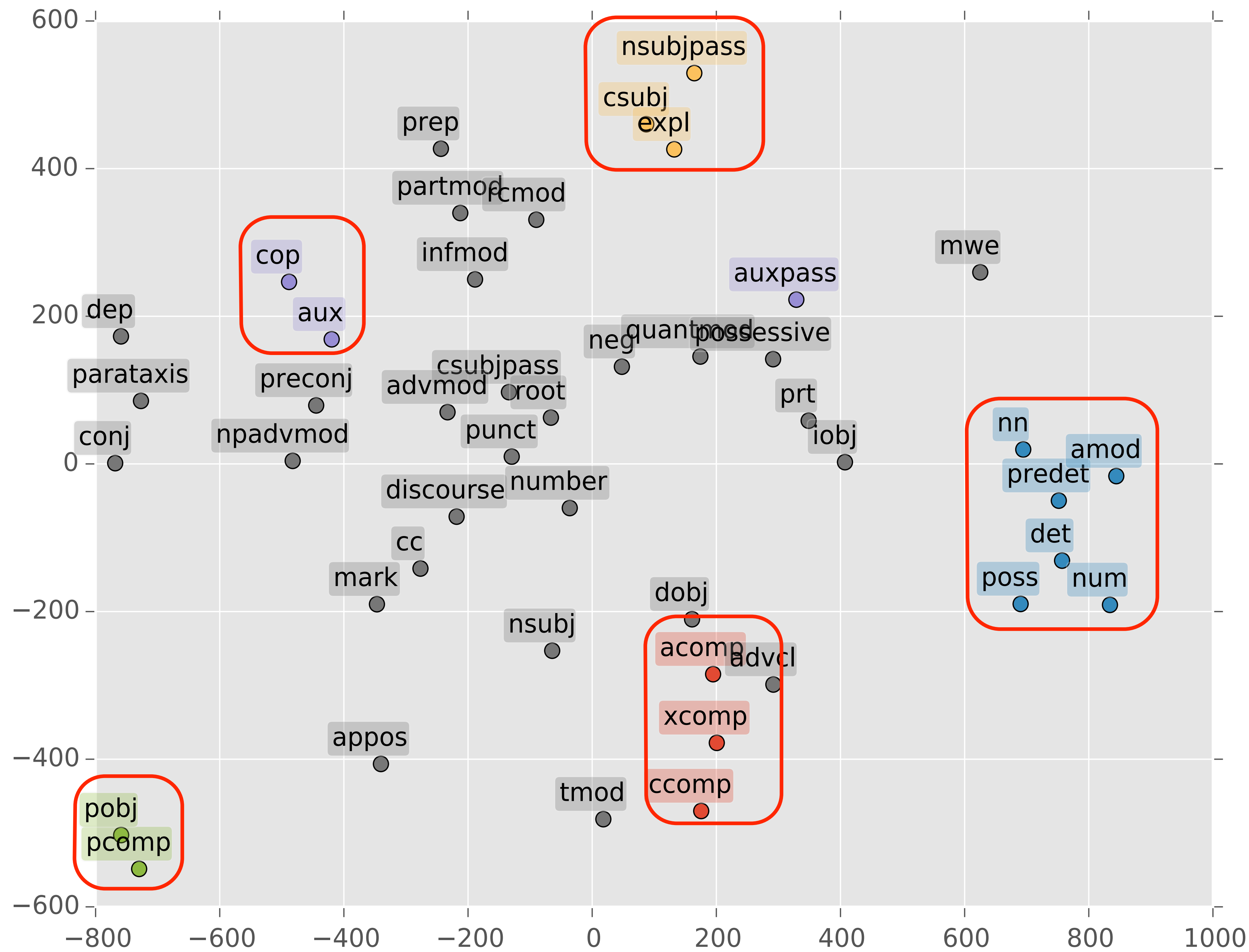


A Fast and Accurate Dependency Parser using Neural Networks

Dependency Embeddings



Dependency Embeddings



Conclusion

- **Summary**
  - Presented a state-of-the-art greedy parser using NNs.
  - Excellent accuracy and speed.
  - Introduced POS / dep. embeddings, and cube activation function.
- **Future work**
  - Beam search
  - Dynamic oracle
  - Richer features (lemma, morph, distance, etc).
  - Better representation for modeling interactions

Thanks

- Code is available!
- Try fast dependency parsing in **Stanford CoreNLP v3.5.0**,
  - annotators: tokenize,ssplit,pos,depparse
- Or check out full training / testing code at:
  - http://nlp.stanford.edu/software/nndep.shtml
- Contact: danqi@cs.stanford.edu