A Fast and Accurate Dependency Parser using Neural Networks

2025-03-24

Proceedings of EMNLP 2014

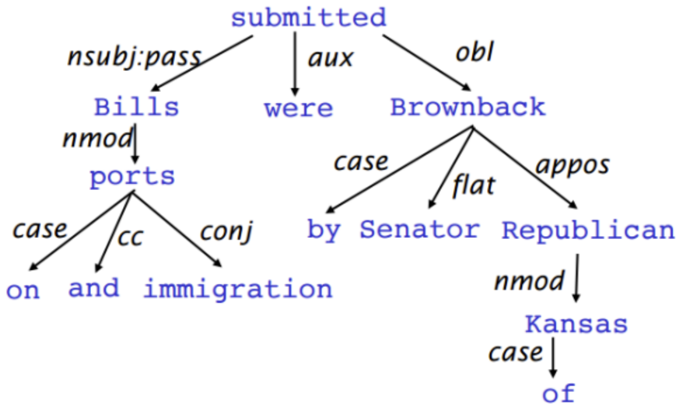
Danqi Chen, Christopher D. Manning

Content

- Introduction
- Related Work
- Transition-based Dependency Parsing
- Goal
- Neural Network Based Parser
- Experiments
- Conclusion

Introduction

- **What is Dependency Parsing?**
  - The task of analyzing the syntactic dependency structure of a given input sentence



Introduction

- **Why is dependency parsing needed?**
  - A model needs to understand sentence structure to be able to interpret language correctly

Scientists count whales from space

✓

Scientists count whales from space

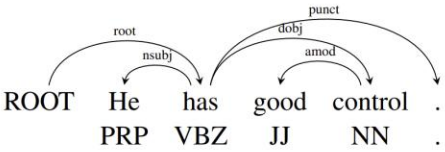
✗

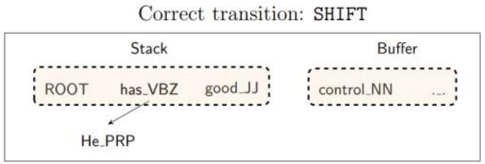
Related Work

- **Dynamic programming**
  - Clever algorithm with complexity $O(n^3)$
- **Graph algorithms**
  - Create a Minimum Spanning Tree for a sentence
- **Constraint Satisfaction**
  - Edges are eliminated that don't satisfy hard constraints
- **Transition-based parsing**
  - Greedy choice of attachments guided by good machine learning classifiers

Transition-based Dependency Parsing

- **Predict a transition sequence**
  - Initial configuration → some terminal configuration
  - Based on indicator features extracted from the configuration
  - Derive a target dependency parse tree



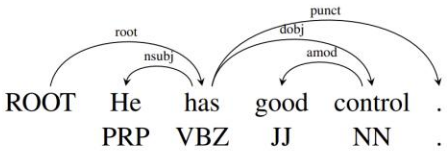


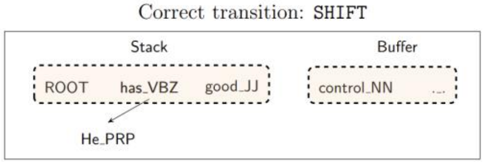
Transition	Stack	Buffer	A
	[ROOT]	[He has good control .]	$\emptyset$
SHIFT	[ROOT He]	[has good control .]	
SHIFT	[ROOT He has]	[good control .]	
LEFT-ARC(nsubj)	[ROOT has]	[good control .]	$A\cup$ nsubj(has,He)
SHIFT	[ROOT has good]	[control .]	
SHIFT	[ROOT has good control]	[.]	
LEFT-ARC(amod)	[ROOT 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)

Transition-based Dependency Parsing

☐ Arc-standard system

- Configuration $c = (s, b, A)$, s : stack, b : buffer, A : a ser of dependency arcs
- Initial configuration
  - ☐ $s = [ROOT], b = [w_1, \ldots, w_n], A = \emptyset$
- Terminal configuration
  - ☐ $s = [ROOT], b = \emptyset$





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)

Transition-based Dependency Parsing

Three types of transitions

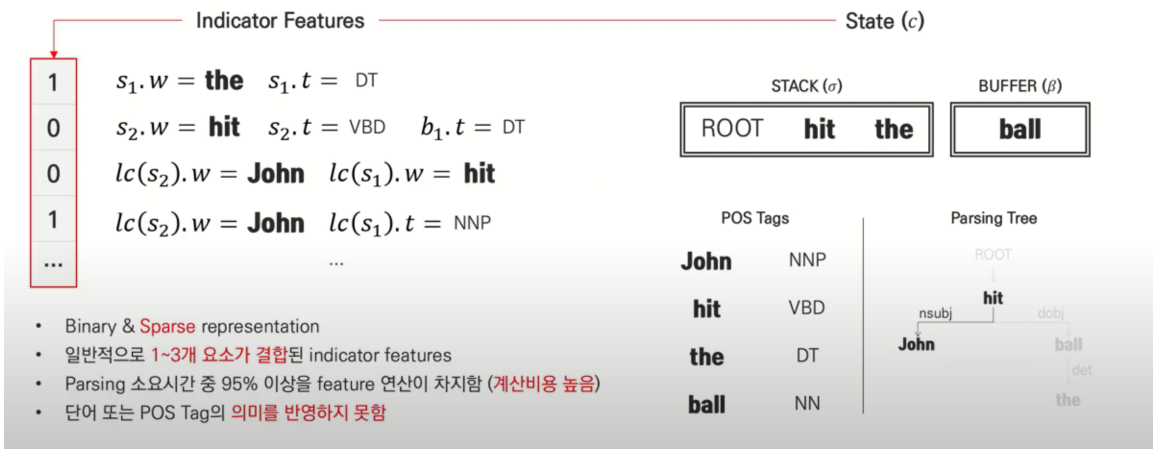
- LEFT-ARC(l) : Add an arc $s_1 \rightarrow s_2$ with label l and removes s_2 from the stack
- RIGHT-ARC(l) : Add an arc $s_2 \rightarrow s_1$ with label l and removes s_1 from the stack
- SHIFT : Move b_1 from the buffer to the stack

Transition-based Dependency Parsing

☐ Indicator features

- Conjunction of 1 ~ 3 elements from the stack/buffer using their words, POS tags or arc labels

Single-word features (9)
$s_1.w$; $s_1.t$; $s_1.wt$; $s_2.w$; $s_2.t$;
$s_2.wt$; $b_1.w$; $b_1.t$; $b_1.wt$
Word-pair features (8)
$s_1.wt \circ s_2.wt$; $s_1.wt \circ s_2.w$; $s_1.wts_2.t$;
$s_1.w \circ s_2.wt$; $s_1.t \circ s_2.wt$; $s_1.w \circ s_2.w$
$s_1.t \circ s_2.t$; $s_1.t \circ b_1.t$
Three-word feaures (8)
$s_2.t \circ s_1.t \circ b_1.t$; $s_2.t \circ s_1.t \circ lc_1(s_1).t$;
$s_2.t \circ s_1.t \circ rc_1(s_1).t$; $s_2.t \circ s_1.t \circ lc_1(s_2).t$;
$s_2.t \circ s_1.t \circ rc_1(s_2).t$; $s_2.t \circ s_1.w \circ rc_1(s_2).t$;
$s_2.t \circ s_1.w \circ lc_1(s_1).t$; $s_2.t \circ s_1.w \circ b_1.t$



Transition-based Dependency Parsing

The problems of Indicator features

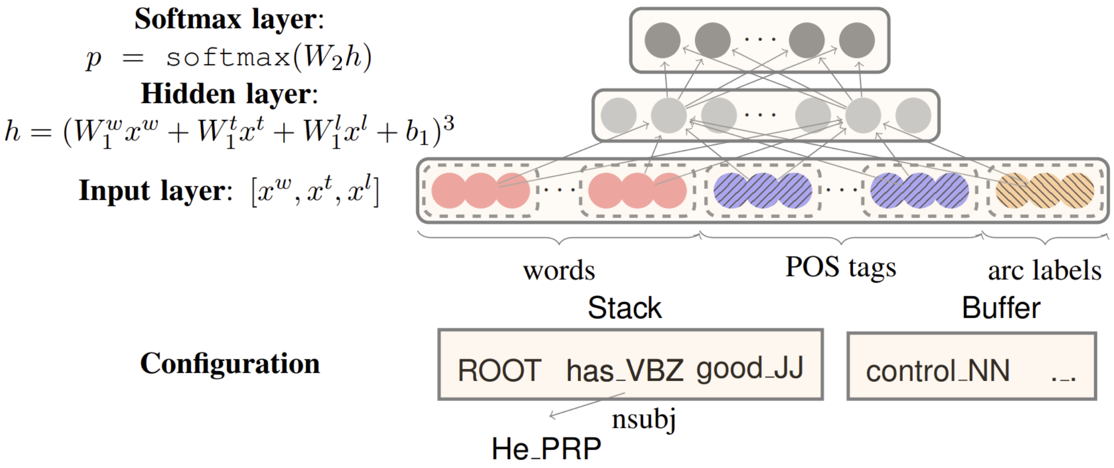
- Sparsity

Features	UAS
All features in Table 1	88.0
single-word & word-pair features	82.7
only single-word features	76.9
excluding all lexicalized features	81.5

- Incompleteness
  - Not include the conjunction of every useful word combination
- Expensive feature computation
  - More than 95% of the time is consumed by feature computation during the parsing process

Goal

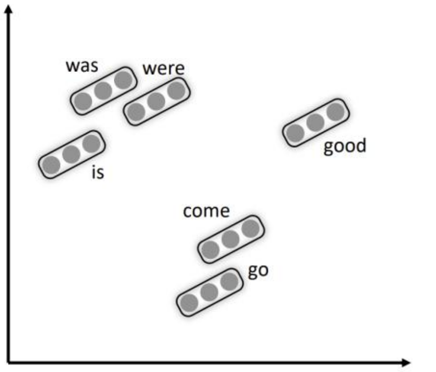
- Propose a novel way of learning a neural network classifier for use in a greedy, transition-based dependency parser
  - Show the usefulness of dense representations
  - Develop a neural network architecture that gives good accuracy and speed
  - Introduce a novel activation function

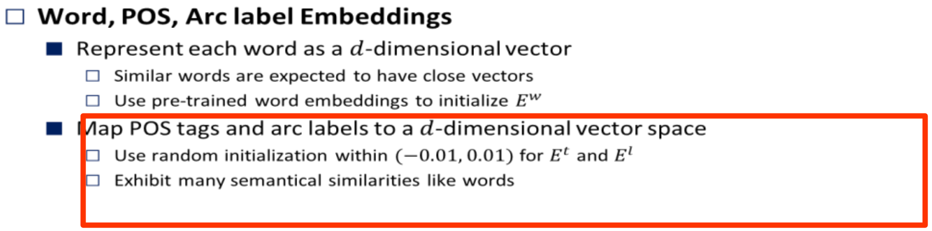


Neural Network Based Parser

☐ Word, POS, Arc label Embeddings

- Represent each word as a d-dimensional vector
  - ☐ Similar words are expected to have close vectors
  - ☐ Use pre-trained word embeddings to initialize E^w
- Map POS tags and arc labels to a d-dimensional vector space
  - ☐ Use random initialization within $(-0.01, 0.01)$ for E^t and E^l
  - ☐ Exhibit many semantical similarities like words




☐ **Word, POS, Arc label Embeddings**

- Represent each word as a d-dimensional vector
  - ☐ Similar words are expected to have close vectors
  - ☐ Use pre-trained word embeddings to initialize E^w
- Map POS tags and arc labels to a d-dimensional vector space
  - ☐ Use random initialization within $(-0.01, 0.01)$ for E^t and E^l
  - ☐ Exhibit many semantical similarities like words

Neural Network Based Parser

☐ **Choose a set of elements,** $i.e., S^w, S^t, S^l$

Feature Template

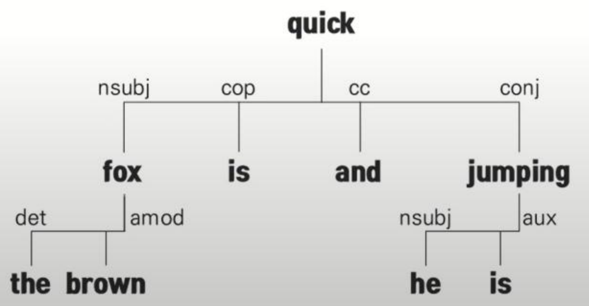
- STACK과 BUFFER의 top 3 단어 (6개) $s_1,\ s_2,\ s_3,\ b_1,\ b_2,\ b_3$
  [over, quick, ROOT, the, lazy, dog]
- STACK top 1, 2 단어의 1st and 2nd left and right child 단어 (8개)
  $lc_1(s_1),\ rc_1(s_1),\ lc_2(s_1),\ rc_2(s_1)$ $lc_1(s_2),\ rc_1(s_2),\ lc_2(s_2),\ rc_2(s_2)$
  [Null, Null, Null, Null] [fox, jumping, is, and]
- STACK top 1, 2 단어의 (left of left) and (right of right) child 단어 (4개)
  $lc_1(lc_1(s_1)),\ rc_1(rc_1(s_1)),\ lc_1(lc_1(s_2)),\ rc_1(rc_1(s_2))$
  [Null, Null] [the, Null]
- 선택된 word feature에 해당하는 POS Tag (18개)
  [IN(전치사), JJ(형용사), ROOT, DT(한정사), …, Null, DT(한정사), Null]
- STACK과 BUFFER의 6개 단어를 제외하고 선택된 word에 달린 arc-label (12개)
  [Null, Null, …, nsubj(주어), conj(접속사), cop(연결사), cc(등위), …, Null]

Example State (c)

Input Sentence

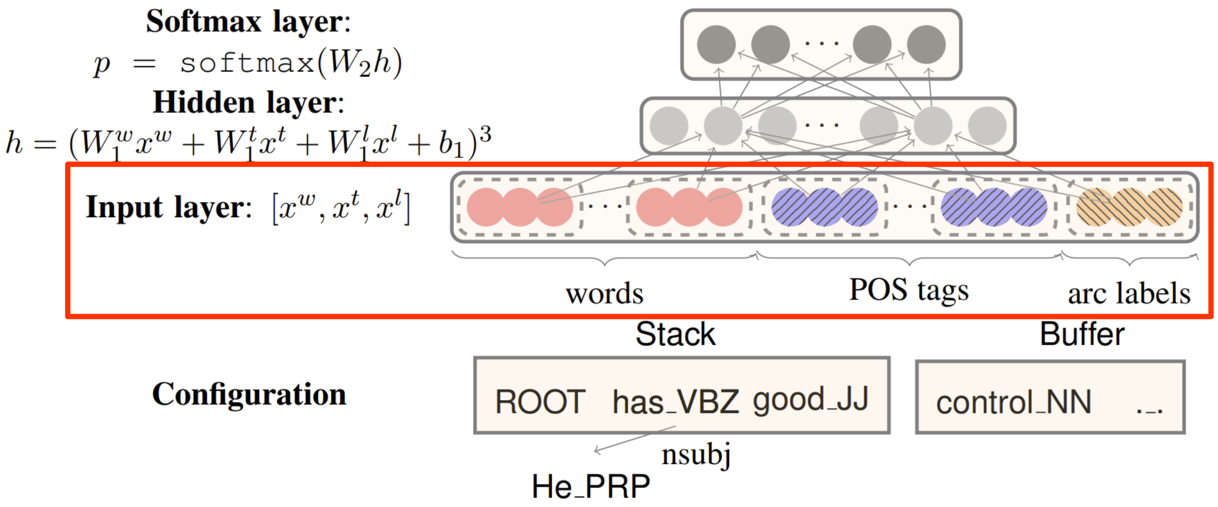
"The brown fox is quick and he is jumping over the lazy dog"

STACK (σ)	BUFFER (β)
ROOT **quick over**	**the lazy dog**



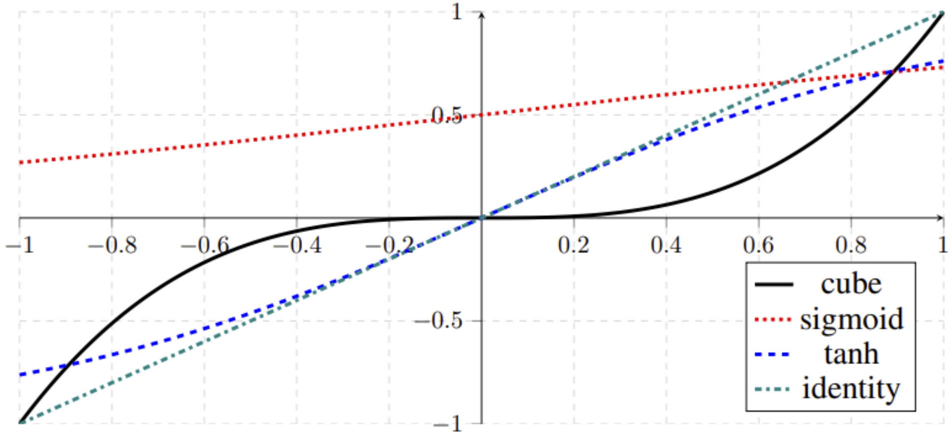
Neural Network Based Parser

- Build a standard neural network with one hidden layer
  - Add $x^w = [e^w_{w_1}; e^w_{w_2}; \ldots e^w_{w_{nw}}], x^t, x^l$ to the input layer



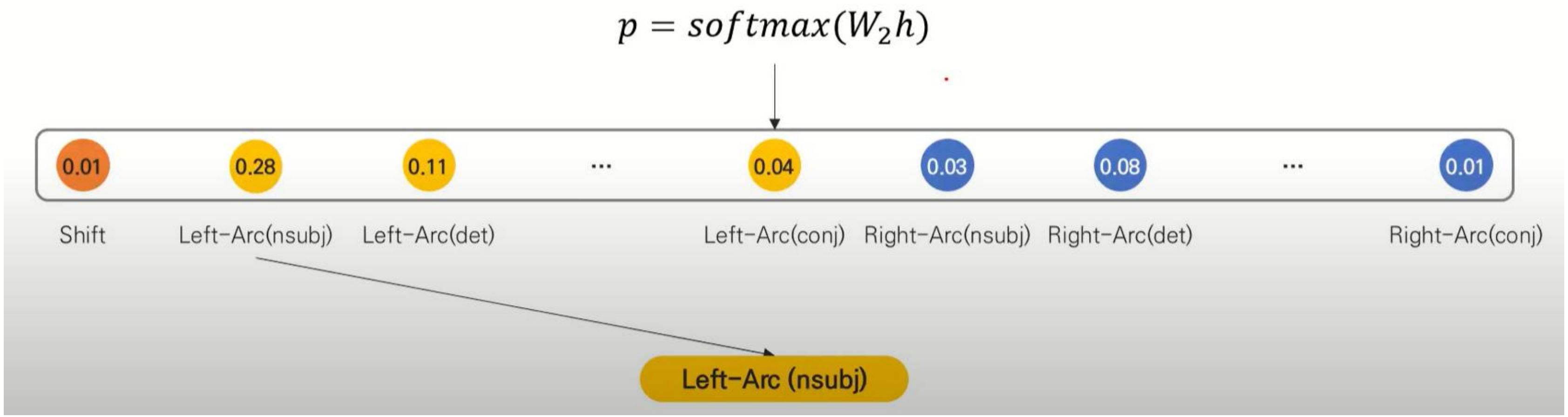
Neural Network Based Parser

- **Build a standard neural network with one hidden layer**
  - Map the input layer to a hidden layer through a cube activation function
  - $h = \left(W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1\right)^3$



Neural Network Based Parser

- **Build a standard neural network with one hidden layer**
  - Model multi-class probabilities $p = softmax(W_2 h)$, where $W_2 \in R^{|\mathcal{T}| \times d_h}$



Neural Network Based Parser

- **Training**
  - Generate training examples $\{(c_i, t_i)\}_{i=1}^{m}$ from the training sentences
  - Use a “shortest stack” oracle which always prefers $\mathrm{LEFT}-ARC_l$ over SHIFT
  - Minimize the cross-entropy loss
    - $L(\theta) = -\sum_i \log p_{t_i} + \frac{\lambda}{2}||\theta||^2$
    - $\theta = \{W_1^w, W_1^t, W_1^l, b_1, W_2, E^w, E^t, E^l\}$

- **Parsing**
  - Pick the transition with the highest score
  - $t = argmax_{t\text{ is feasible}} \mathrm{W}_2(\mathrm{t},\cdot)h(c)$
  - $c \rightarrow t(c)$

Neural Network Based Parser

- **Pre-computation trick**
  - Pre-compute matrix multiplications for most top frequent $10,000$ words
  - Pre-compute matrix computations for all positions and all POS tags and arc labels

    $\rightarrow$ Increases the speed of our parser 8 $\sim$ 10 times

Experiments

☐ Datasets

Dataset	#Train	#Dev	#Test	#words (N_w)	#POS (N_t)	#labels (N_l)	projective (%)
PTB: CD	39,832	1,700	2,416	44,352	45	17	99.4
PTB: SD	39,832	1,700	2,416	44,389	45	45	99.9
CTB	16,091	803	1,910	34,577	35	12	100.0

Table 3: Data Statistics. "Projective" is the percentage of projective trees on the training set.

Experiments

- Outperform other parsing methods

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	89.9	88.7	89.7	88.3	51
eager	90.3	89.2	89.9	88.6	63
Malt:sp	90.0	88.8	89.9	88.5	560
Malt:eager	90.1	88.9	90.1	88.7	535
MSTParser	92.1	90.8	**92.0**	90.5	12
Our parser	**92.2**	**91.0**	**92.0**	**90.7**	**1013**

Table 4: Accuracy and parsing speed on PTB + CoNLL dependencies.

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	90.2	87.8	89.4	87.3	26
eager	89.8	87.4	89.6	87.4	34
Malt:sp	89.8	87.2	89.3	86.9	469
Malt:eager	89.6	86.9	89.4	86.8	448
MSTParser	91.4	88.1	90.7	87.6	10
Our parser	**92.0**	**89.7**	**91.8**	**89.6**	**654**

Table 5: Accuracy and parsing speed on PTB + Stanford dependencies.

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	82.4	80.9	82.7	81.2	72
eager	81.1	79.7	80.3	78.7	80
Malt:sp	82.4	80.5	82.4	80.6	420
Malt:eager	81.2	79.3	80.2	78.4	393
MSTParser	**84.0**	82.1	83.0	81.2	6
Our parser	**84.0**	**82.4**	**83.9**	**82.4**	**936**

Table 6: Accuracy and parsing speed on CTB.

Experiments

□ **Effects of different parser components**

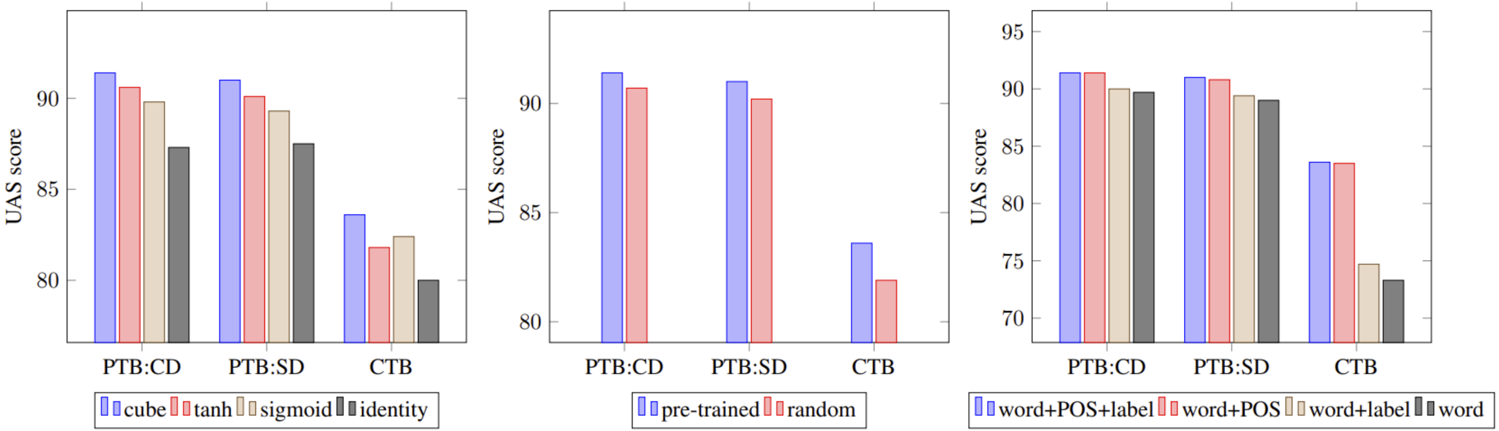
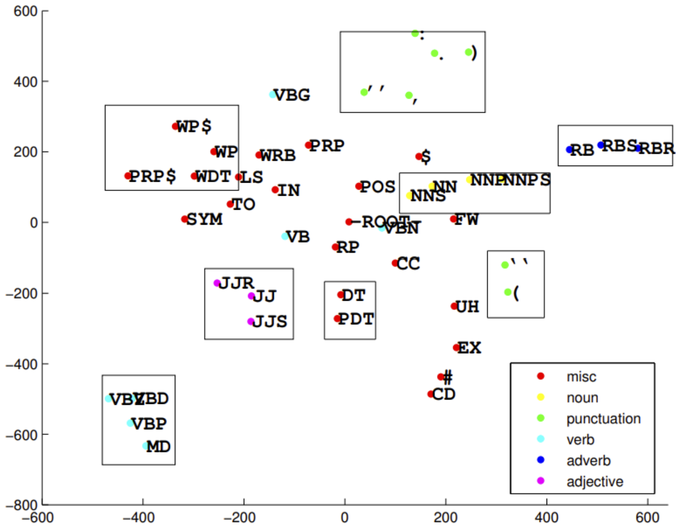
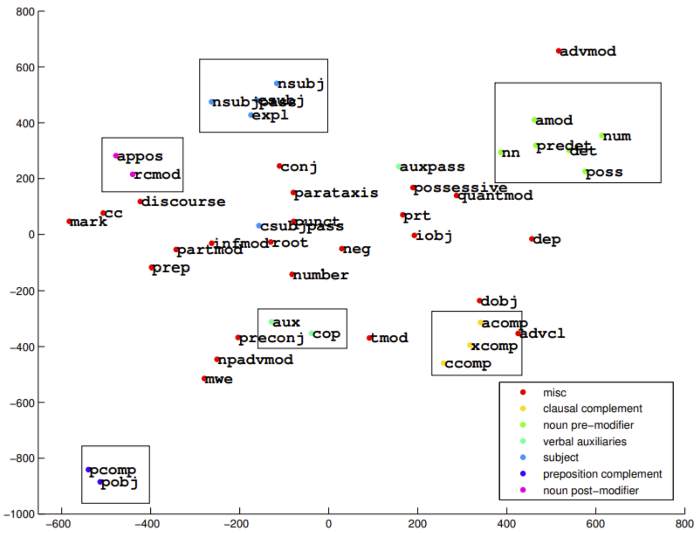


Figure 4: Effects of different parser components. Left: comparison of different activation functions. Middle: comparison of pre-trained word vectors and random initialization. Right: effects of POS and label embeddings.

Experiments

- t-SNE visualization of POS and label embeddings
  - Embeddings effectively exhibit the similarities between POS tags or arc labels.





Experiments

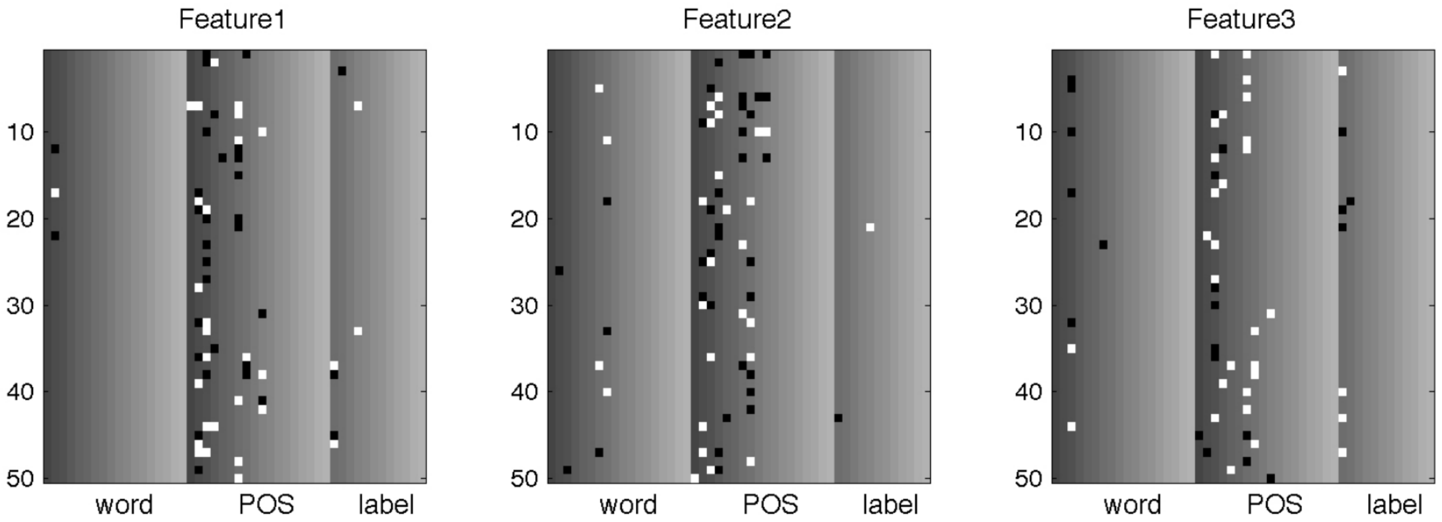


Figure 6: Three sampled features. In each feature, each row denotes a dimension of embeddings and each column denotes a chosen element, e.g., $s_1.t$ or $lc(s_1).w$, and the parameters are divided into 3 zones, corresponding to $W_1^w(k,:)$ (left), $W_1^t(k,:)$ (middle) and $W_1^l(k,:)$ (right). White and black dots denote the most positive weights and most negative weights respectively.

Conclusion

- Dependency Parsing
  - Make model understand sentence structure to be able to interpret language correctly
- Transition-based Dependency Parsing
  - Sparse, Incomplete, Expensive feature extraction

- Neural Network Based Parsing
  - Word, POS, Arc label Embeddings
  - Use a cube activation function

- Experiments
  - Outperform other parsing methods
  - Encode the semantic regularities