Precept 5: Parsing Howard Chen 03/03/2023

Agenda

- Constituency Parsing
- Dependency Parsing
- Questions

Input

Sam thinks Sandy likes the book



• Probabilistic Context-Free Grammars (PCFGs)

Defines a bunch of production rules.



- Probabilistic Context-Free Grammars (PCFGs)
- MLE for learning

			*******		******
R, q =	:				
5	S	\rightarrow	NP	VP	1.0
	VP	\rightarrow	Vi		0.3
1	VP	\rightarrow	Vt	NP	0.5
1	VP	\rightarrow	VP	PP	0.2
1	NP	\rightarrow	DT	NN	0.8
1	NP	\rightarrow	NP	PP	0.2
I	PP	\rightarrow	IN	NP	1.0
	/i	\rightarrow	sleer	ns	10
	/i /t	\rightarrow \rightarrow	sleep	08	1.0 1.0
N N N	/i /t NN	\rightarrow \rightarrow \rightarrow	sleep saw man)S	1.0 1.0 0.1
\ \ N N	/i /t NN	$ \begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{array} $	sleep saw man wom	os	1.0 1.0 0.1 0.1
N N N N	/i /t JN JN JN	$\begin{array}{c} \uparrow \\ \uparrow \end{array}$	sleep saw man wom teles	an cope	1.0 1.0 0.1 0.1 0.3
N N N N N	/i /t JN JN JN JN	$\begin{array}{c c} \uparrow & \uparrow \\ \uparrow & \uparrow & \uparrow \\ \uparrow & \uparrow & \uparrow \\ \end{array}$	sleep saw man wom teles dog	an cope	1.0 1.0 0.1 0.1 0.3 0.5
N N N N I	/i /t JN JN JN JN JN JN	$\begin{array}{c}\uparrow\\\uparrow\\\uparrow\\\uparrow\\\uparrow\\\uparrow\\\uparrow\\\uparrow\\\uparrow\end{array}$	sleep saw man wom teles dog the	os ian cope	1.0 1.0 0.1 0.3 0.5 1.0
N N N N I I	/i /t /N /N /N /N /N /N /N	1 1 1 1 1	sleep saw man wom teles dog the with	an cope	1.0 1.0 0.1 0.3 0.5 1.0 0.6

- Probabilistic Context-Free Grammars (PCFGs)
- MLE for learning
- Parsing PCFGs: the CKY algorithm

	*****			*****
q =				
S	\rightarrow	NP	VP	1.0
VP	\rightarrow	Vi		0.3
VP	\rightarrow	Vt	NP	0.5
VP	\rightarrow	VP	PP	0.2
NP	\rightarrow	DT	NN	0.8
NP	\rightarrow	NP	PP	0.2
PP	\rightarrow	IN	NP	1.0
Vi	\rightarrow	sleep	os	1.0
Vi Vt	\rightarrow \rightarrow	sleep saw)S	1.0 1.0
Vi Vt NN	\rightarrow \rightarrow \rightarrow	sleep saw man	05	1.0 1.0 0.1
Vi Vt NN NN	$ \begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{array} $	sleep saw man wom	os nan	1.0 1.0 0.1 0.1
Vi Vt NN NN NN	$\begin{array}{c} \uparrow \\ \uparrow \\ \uparrow \\ \uparrow \\ \uparrow \\ \uparrow \\ \uparrow \end{array}$	sleep saw man wom teles	nan cope	1.0 1.0 0.1 0.1 0.3
Vi Vt NN NN NN NN	$\uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow$	sleep saw man wom teles dog	an cope	1.0 1.0 0.1 0.1 0.3 0.5
Vi Vt NN NN NN NN DT	$\uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow$	sleep saw man wom teles dog the	nan cope	1.0 1.0 0.1 0.3 0.5 1.0
Vi Vt NN NN NN DT IN	$\uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow$	sleep saw man wom teles dog the with	an cope	1.0 1.0 0.1 0.1 0.3 0.5 1.0 0.6



 $q(\text{VP} \rightarrow \text{Vt NP}) \times q(\text{NP} \rightarrow \text{NP PP}) = 0.5 \times 0.2 = 0.1$ $q(\text{VP} \rightarrow \text{VP PP}) \times q(\text{VP} \rightarrow \text{Vt NP}) = 0.2 \times 0.5 = 0.1$

Input

I prefer the morning flight through Denver

Output



Arc-standard algorithm

Arc-standard algorithm

How does a dependency parser parse a sentence into a tree?



- Arc-standard algorithm
- Input: a sequence of words
- Output: a parse (i.e., which word points to which word)

- Arc-standard algorithm
- Input: a sequence of words
- Output: a parse (i.e., which word points to which word)
- Configuration: stack (s) + buffer (b) + a set of arcs (A)
- Starting config: $s = [ROOT], b = [w_1, w_2, ..., w_n], A = \emptyset$
- End config: $s = [ROOT], b = \emptyset$

- Arc-standard algorithm
- Input: a sequence of words
- Output: a parse (i.e., which word points to which word)
- Configuration: stack (s) + buffer (b) + a set of arcs (A)
- Starting config: s = [ROOT], b =
- End config: $s = [ROOT], b = \emptyset$
- Actions: SHIFT, LEFT-ARC(r), RIGHT-ARC(r)

$$= [w_1, w_2, ..., w_n], A = \emptyset$$

SHIFT: move word at the top of buffer -> top of stack

Current configuration

After transition



- SHIFT: move word at the top of buffer -> top of stack
- LEFT-ARC(r)

Current configuration

After transition



- SHIFT: move word at the top of buffer -> top of stack
- LEFT-ARC(r)
- RIGHT-ARC(r)

Current configuration



After transition







k	buffer	action	added arc
T] [Book, me, the, morni	ng, flight]	SHIFT	



	stack	buffer	action	added arc
0	[ROOT] [Book, me, the, mo	rning, flight]	SHIFT	
1	[ROOT, Book] [me, the, morning	g, flight]	SHIFT	



	stack	buffer	action	added arc
0	[ROOT] [Book, me, the, mor	ning, flight]	SHIFT	
1	[ROOT. Book] [me, the, morning	flight]	SHIFT	
2	[ROOT, Book, me] [the, morning, flig	ht]	RIGHT-ARC(iobj)	(Book, iobj, 1





	stack	buffer	action	added arc
0	[ROOT] [Book, me, the, mor	ning, flight]	SHIFT	
1	[ROOT, Book] [me, the, morning,	flight]	SHIFT	
2	[ROOT, Book, me] [the, morning, fligh	nt]	RIGHT-ARC(iobj)	(Book, iobj, 1
3	[ROOT, Book] [the, morning, fligh	nt]	SHIFT	



stacl



k	buffer	action	added arc
OT] [Book, me, the, mornin	ıg, flight]	SHIFT	
ok] [me, the, morning, fli	ght]	SHIFT	
ne] [the, morning, flight]		RIGHT-ARC(iobj)	(Book, iobj, 1
ok] [the, morning, flight]		SHIFT	
he] [morning, flight]		SHIFT	
ng] [flight]		SHIFT	



	stack	buffer	action	added arc
0	[ROOT] [Book, me, the,	morning, flight]	SHIFT	
1	[ROOT, Book] [me, the, morn	ing, flight]	SHIFT	
2	[ROOT, Book, me] [the, morning,	flight]	RIGHT-ARC(iobj)	(Book, iobj, 1
3	[ROOT, Book] [the, morning,	flight]	SHIFT	
4	[ROOT, Book, the] [morning, fligh	ıt]	SHIFT	
5	[ROOT, Book, the, morning] [flight]		SHIFT	
6	[ROOT, Book, the,morning,flight] []		LEFT-ARC(nmod)	(flight,nmod,mo





	stack	buffer	action	added arc
0	[ROOT] [Book, me,	the, morning, flight]	SHIFT	
1	[ROOT, Book] [me, the, r	norning, flight]	SHIFT	
2	[ROOT, Book, me] [the, morn	ing, flight]	RIGHT-ARC(iobj)	(Book, iobj, 1
3	[ROOT, Book] [the, morn	ing, flight]	SHIFT	
4	[ROOT, Book, the] [morning,	flight]	SHIFT	
5	[ROOT, Book, the, morning] [flight]		SHIFT	
6	[ROOT, Book, the,morning,flight] []		LEFT-ARC(nmod)	(flight,nmod,mo
7	[ROOT, Book, the, flight] []		LEFT-ARC(det)	(flight,det,tł





	stack	buffer	action	added arc
0	[ROOT] [Book, me, the	, morning, flight]	SHIFT	
1	[ROOT, Book] [me, the, mor	ming, flight]	SHIFT	
2	[ROOT, Book, me] [the, morning	g, flight]	RIGHT-ARC(iobj)	(Book, iobj, 1
3	[ROOT, Book] [the, morning	g, flight]	SHIFT	
4	[ROOT, Book, the] [morning, flig	ght]	SHIFT	
5	[ROOT, Book, the, morning] [flight]		SHIFT	
6	[ROOT, Book, the,morning,flight] []		LEFT-ARC(nmod)	(flight,nmod,mo
7	[ROOT, Book, the, flight] []		LEFT-ARC(det)	(flight,det,tł
8	[ROOT, Book, flight] []		RIGHT-ARC(dobj)	(Book,dobj,fl





	stack	buffer	action	added arc
0	[ROOT] [Book, me, the,	morning, flight]	SHIFT	
1	[ROOT, Book] [me, the, morn	ing, flight]	SHIFT	
2	[ROOT, Book, me] [the, morning,	flight]	RIGHT-ARC(iobj)	(Book, iobj, 1
3	[ROOT, Book] [the, morning,	flight]	SHIFT	
4	[ROOT, Book, the] [morning, fligh	t]	SHIFT	
5	[ROOT, Book, the, morning] [flight]		SHIFT	
6	[ROOT, Book, the,morning,flight] []		LEFT-ARC(nmod)	(flight,nmod,mo
7	[ROOT, Book, the, flight] []		LEFT-ARC(det)	(flight,det,tl
8	[ROOT, Book, flight] []		RIGHT-ARC(dobj)	(Book,dobj,fl
9	[ROOT, Book] []		RIGHT-ARC(root)	(ROOT,root,E





	stack	buffer	action	added arc
0	[ROOT] [Book, me, the	e, morning, flight]	SHIFT	
1	[ROOT, Book] [me, the, mo	rning, flight]	SHIFT	
2	[ROOT, Book, me] [the, morning	g, flight]	RIGHT-ARC(iobj)	(Book, iobj, 1
3	[ROOT, Book] [the, morning	g, flight]	SHIFT	
4	[ROOT, Book, the] [morning, flig	ght]	SHIFT	
5	[ROOT, Book, the, morning] [flight]		SHIFT	
6	[ROOT, Book, the,morning,flight] []		LEFT-ARC(nmod)	(flight,nmod,mo
7	[ROOT, Book, the, flight] []		LEFT-ARC(det)	(flight,det,tl
8	[ROOT, Book, flight] []		RIGHT-ARC(dobj)	(Book,dobj,fl
9	[ROOT, Book] []		RIGHT-ARC(root)	(ROOT,root,E
10	[ROOT] []			



Book me the morning flight





Common confusion: the 'case' label points into the preposition word

• Where does the dependency parser come from?

- Where does the dependency parser come from?
- We need to train it using statistical learning methods!

- Where does the dependency parser come from?
- We need to train it using statistical learning methods!
- Collect datasets $\{(x_i, y_i)\}_{i=1}^N$
- *x*: sentence
- y: dependency parse tree

- Where does the dependency parser come from?
- We need to train it using statistical learning methods!
- Collect datasets $\{(x_i, y_i)\}_{i=1}^N$
- *x*: sentence
- y: dependency parse tree
- How do we use annotated these data to train a parser?

• Model needs to learn: $c \rightarrow a$ (configuration to action mapping)



a = M(c)

- Model needs to learn: $c \rightarrow a$ (configuration to action mapping)
- We can generate this from the annotated $\{(x_i, y_i)\}_{i=1}^N$

- Model needs to learn: $c \rightarrow a$ (configuration to action mapping)
- We can generate this from the annotated $\{(x_i, y_i)\}_{i=1}^N$
- Run arc-standard, we can collect 2n(c, a) pairs from one (x, y) pair, where n is the number of words in x

- Model needs to learn: $c \rightarrow a$ (configuration to action mapping)
- We can generate this from the annotated $\{(x_i, y_i)\}_{i=1}^N$
- Run arc-standard, we can collect 2n(c, a) pairs from one (x, y) pair, where n is the number of words in x
- To complete the process, each word needs to be shifted and merged exactly once

- Model needs to learn: $c \rightarrow a$ (configuration to action mapping)
- We can generate this from the annotated $\{(x_i, y_i)\}_{i=1}^N$
- Run arc-standard, we can collect 2n(c, a) pairs from one (x, y) pair, where *n* is the number of words in *x*
- To complete the process, each word needs to be shifted and merged exactly once
- Number of actions: 2|R| + 1 (+SHIFT)

- **Evaluation**
- Unlabeled attachment score (UAS): only look at the arcs
- Labeled attachment score (LAS): look at both the arcs and the labels



Questions?