L95: Natural Language Syntax and Parsing 3) Dependency Parsing

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

So far we have:

- Used CFGs to define a set of strings through legal structures
- Used the CKY algorithm to find all structures for a given string
- Used PCFGs to assign probabilities to the structures deriving the string
- Used a modified CKY algorithm to find the best structure from all the possibilities

- So far we have: (for statistical parsing more generally we need...)
 - Used CFGs to define a set of strings through legal structures (a grammar)
 - Used the CKY algorithm to find all structures for a given string (a parsing algorithm)
 - Used PCFGs to assign probabilities to the structures deriving the string (a scoring model for parses)
 - Used a modified CKY algorithm to find the best structure from all the possibilities (an algorithm for finding best parse)

Recall that:

$\hat{T}(W) = \underset{trees that yield W}{\operatorname{argmax}} P(T|W)$

• finding Ts that yield W requires a parsing algorithm over a grammar

- knowing P(T|W) requires a probabilistic model over T
- argmax requires an algorithm for finding best parse

More generally:

$$\hat{T}(W) = \operatorname*{argmax}_{trees that yield W} Score(T|W)$$

• Generative models use the product of estimated probabilities of parse pieces to find P(T|W) but we could use other scoring functions...

Recall that:

$$\hat{T}(W) = \underset{trees that yield W}{\operatorname{argmax}} P(T|W)$$

• finding Ts that yield W requires a parsing algorithm over a grammar

- knowing P(T|W) requires a probabilistic model over T
- argmax requires an algorithm for finding best parse

More generally:

$$\hat{T}(W) = \operatorname*{argmax}_{trees \ that \ yield \ W} Score(T|W)$$

• Generative models use the product of estimated probabilities of parse pieces to find P(T|W) but we could use other scoring functions...

Generative models have some issues

Recall that:

 $P(T,W) = \prod_{i=1}^{n} P(A_i \to B_i)$

• But P(T, W) = P(T)P(W|T) and that P(W|T) = 1 so

P(T, W) = P(T) and thus $P(T) = \prod_{i=1}^{n} P(A_i \to B_i)$

- How do we know how to break the parse into pieces?
- Is the independence assumption valid?
- Generative models simultaneously model the tree and the string—discriminative models define P(T|W) directly
- Models are **discriminative** because they compare the correct parse against incorrect parses in training to set parameters... can use machine learning approaches.
- First we're going to learn about a grammar that we will use to define \mathcal{T} within a discriminative model—dependency grammars.

Generative models have some issues

Recall that:

 $P(T,W) = \prod_{i=1}^{n} P(A_i \to B_i)$

• But P(T, W) = P(T)P(W|T) and that P(W|T) = 1 so

P(T, W) = P(T) and thus $P(T) = \prod_{i=1}^{n} P(A_i \to B_i)$

- How do we know how to break the parse into pieces?
- Is the independence assumption valid?
- Generative models simultaneously model the tree and the string—discriminative models define P(T|W) directly
- Models are **discriminative** because they compare the correct parse against incorrect parses in training to set parameters... can use machine learning approaches.
- First we're going to learn about a grammar that we will use to define \mathcal{T} within a discriminative model—dependency grammars.

Generative models have some issues

Recall that:

 $P(T, W) = \prod_{i=1}^{n} P(A_i \to B_i)$

• But P(T, W) = P(T)P(W|T) and that P(W|T) = 1 so

P(T, W) = P(T) and thus $P(T) = \prod_{i=1}^{n} P(A_i \to B_i)$

- How do we know how to break the parse into pieces?
- Is the independence assumption valid?
- Generative models simultaneously model the tree and the string—discriminative models define P(T|W) directly
- Models are **discriminative** because they compare the correct parse against incorrect parses in training to set parameters... can use machine learning approaches.
- First we're going to learn about a grammar that we will use to define *T* within a discriminative model—dependency grammars.

A dependency tree is a directed graph

A **dependency tree** is a directed graph representation of a string—each edge represents a grammatical relationship between the symbols.



A dependency grammar deriving dependency trees

Formally $G_{dep} = (\Sigma, \mathcal{D}, s, \bot, \mathcal{P})$ where:

- Σ is the finite set of alphabet symbols
- D is the set of symbols to indicate whether the dependent symbol (the one on the RHS of the rule) will be located on the left or right of the current item within the string D = {L, R}
- s is the root symbol for the dependency tree (we will use s ∈ Σ but sometimes a special extra symbol is used)
- ullet \perp is a symbol to indicate an allowed endpoint to the graph
- \mathcal{P} is a set of rules for generating dependencies: $\mathcal{P} = \{ (\alpha \to \beta, d) \mid \alpha \in (\Sigma \cup s), \beta \in (\Sigma \cup \bot), d \in \mathcal{D} \}$

In dependency grammars we refer to the term on the LHS of a rule as the **head** and the RHS as the **dependent** (as opposed to *parents* and *children* in phrase structure grammars).

Dependency trees have several representations

Two diagrammatic representations of a dependency tree for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$ where:



The same rules would have been used to generate the string *badfec*. Useful when there is flexibility in the symbol order of grammatical strings.

Recall: shift-reduce parsers and **deterministic** languages

LR(k) Shift-reduce parsers are most useful for recognising the strings of deterministic languages (languages where no string has more than one analysis) which have been described by an unambiguous grammar.

Quick reminder:

- The parsing algorithm has two actions: SHIFT and REDUCE
- Initially the input string is held in the buffer and the stack is empty.
- Symbols are **shifted** from the buffer to the stack
- When the top items of the stack match the RHS of a rule in the grammar then they are **reduced**, that is, they are replaced with the LHS of that rule.
- k refers to the look-ahead.



Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

			STAC	К	BUFFER abcd	ACTION SHIFT
Σ N S P	=	$ \{a, b, c\} \\ \{S, A, B, C, D\} \\ S \\ \{S \rightarrow AB, \\ A \rightarrow a, \\ B \rightarrow bC, \\ C \rightarrow cD, \\ D \rightarrow d\} $				

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	$\{a, b, c\}$	а	bcd	
\mathcal{N}^{-}	=	$\{S, A, B, C, D\}$			
s	=	S			
\mathcal{P}	=	$\{S ightarrow A B,$			
		A ightarrow a,			
		$B ightarrow b \ C$,			
		C ightarrow c D,			
		D ightarrow d			
		2			

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	$\{a, b, c\}$	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$			
s	=	S			
\mathcal{P}	=	$\{S ightarrow A B,$			
		A ightarrow a,			
		B ightarrow b C ,			
		C ightarrow c D,			
		$D ightarrow d\}$			

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	
5	=	S			
${\mathcal P}$	=	$\{S ightarrow A B,$			
		A ightarrow a,			
		B ightarrow b C ,			
		C ightarrow c D,			
		$D ightarrow d\}$			

			CT A CIZ	DUEFED	ACTION
			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
s	=	S			
\mathcal{P}	=	$\{S ightarrow A B,$			
		$A \rightarrow a$,			
		$B \rightarrow b C$			
		$C \rightarrow c D$			
		$D \rightarrow a$			

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
5	=	S	Ab	cd	
\mathcal{P}	=	$\{S \rightarrow A B, A \}$			
		$A \rightarrow a$, $B \rightarrow b C$.			
		$C \rightarrow c D$,			
		$D ightarrow d\}$			

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
5	=	S	Ab	cd	SHIFT
\mathcal{P}	=	$\{S \rightarrow A B,$			
		$A \rightarrow a$,			
		$B \rightarrow bC$,			
		$C \rightarrow C D$, $D \rightarrow d$			
		$D \rightarrow u_f$			

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
5	=	S	Ab	cd	SHIFT
\mathcal{P}	=	$\{S \rightarrow A B,$	Abc	d	
		$A \rightarrow a$, $B \rightarrow b C$			
		$C \rightarrow c D$,			
		$D ightarrow d\}$			

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
5	=	S	Ab	cd	SHIFT
\mathcal{P}	=	$\{S \rightarrow A B,$	Abc	d	SHIFT
		$A \rightarrow a$,			
		B ightarrow b C ,			
		C ightarrow c D ,			
		$D \rightarrow d$ }			
		,			

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
5	=	S	Ab	cd	SHIFT
\mathcal{P}	=	$\{S \rightarrow A B, A B\}$	Abc	d	SHIFT
		$A \rightarrow a, B \rightarrow b C$	Abcd		
		$C \rightarrow c D$,			
		$D ightarrow d\}$			

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> . <i>b</i> . <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
5	=	S	Ab	cd	SHIFT
\mathcal{P}	=	$\{S \rightarrow A B, A B\}$	Abc	d	SHIFT
		$A \rightarrow a$, $B \rightarrow b C$	Abcd		REDUCE
		$C \rightarrow c D$,			
		$D ightarrow d\}$			

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

		STACK	BUFFER	ACTION
			abcd	SHIFT
=	$\{a, b, c\}$	а	bcd	REDUCE
=	$\{S, A, B, C, D\}$	A	bcd	SHIFT
=	S	Ab	cd	SHIFT
=	$\{S \rightarrow AB, A \}$	Abc	d	SHIFT
	$A \rightarrow a, B \rightarrow b C.$	Abcd		REDUCE
	$C \rightarrow c D$,	AbcD		
	$D ightarrow d\}$			
		$= \{a, b, c\}$ $= \{S, A, B, C, D\}$ $= S$ $= \{S \rightarrow AB, A \rightarrow a, B \rightarrow bC, C, C \rightarrow cD, D \rightarrow d\}$	STACK $= \{a, b, c\} = a$ $= \{S, A, B, C, D\} = A$ $= S = Ab$ $= \{S \rightarrow AB, Abc$ $A \rightarrow a, Abc$ $B \rightarrow b C, Abcd$ $C \rightarrow c D, AbcD$ $D \rightarrow d\}$	$\begin{array}{c cccc} & & & & & & \\ & = & \{a,b,c\} & & a & & & bcd \\ & = & \{S,A,B,C,D\} & & A & & bcd \\ & = & S & & Ab & & cd \\ & = & \{S \rightarrow AB, & & Abc & & d \\ & & A \rightarrow a, & & Abc & & d \\ & & B \rightarrow b C, & & Abcd \\ & & C \rightarrow c D, & & AbcD \\ & & D \rightarrow d\} & & & \\ \end{array}$

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
5	=	S	Ab	cd	SHIFT
\mathcal{P}	=	$\{ S \rightarrow A B, A \rightarrow a \}$	Abc	d	SHIFT
		$B \rightarrow b C$,	Abcd		REDUCE
		$C \rightarrow c D$,	AbcD		REDUCE
		$D ightarrow d\}$			

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
5	=	S	Ab	cd	SHIFT
\mathcal{P}	=	$\{S \rightarrow A B, A B\}$	Abc	d	SHIFT
		$A \rightarrow a, B \rightarrow b C.$	Abcd		REDUCE
		$C \rightarrow c D$,	AbcD		REDUCE
		$D ightarrow d\}$	AbC		

Shift-reduce parse for the string *abcd* generated using $G_{cfg} = (\Sigma, \mathcal{N}, s, \mathcal{P})$:

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
S	=	S	Ab	cd	SHIFT
P	=	$\{S \rightarrow A B, A \}$	Abc	d	SHIFT
		$A \rightarrow a, B \rightarrow b C.$	Abcd		REDUCE
		$C \rightarrow c D$,	AbcD		REDUCE
		$D ightarrow d\}$	AbC		REDUCE

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> , <i>b</i> , <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
5	=	S	Ab	cd	SHIFT
\mathcal{P}	=	$\{S \rightarrow A B, A B, A B\}$	Abc	d	SHIFT
		$A \rightarrow a, B \rightarrow b C$	Abcd		REDUCE
		$C \rightarrow c D$,	AbcD		REDUCE
		$D ightarrow d\}$	AbC		REDUCE
			AB		

			STACK	BUFFER	ACTION
				abcd	SHIFT
Σ	=	{ <i>a</i> . <i>b</i> . <i>c</i> }	а	bcd	REDUCE
\mathcal{N}	=	$\{S, A, B, C, D\}$	А	bcd	SHIFT
5	=	S	Ab	cd	SHIFT
\mathcal{P}	=	$\{S \rightarrow A B, A B\}$	Abc	d	SHIFT
		$A \rightarrow a, B \rightarrow b C$	Abcd		REDUCE
		$C \rightarrow c D$,	AbcD		REDUCE
		$D ightarrow d\}$	AbC		REDUCE
			AB		REDUCE

$\Sigma = \mathcal{N} = s = \mathcal{P} =$	$ \{a, b, c\} $ $ \{S, A, B, C, D\} $ S $ \{S \rightarrow A B, $ $ A \rightarrow a, $ $ B \rightarrow b C, $ $ C \rightarrow c D, $ $ D \rightarrow d\} $	a A Ab Abc Abcd AbcD AbC AB S	BUFFER abcd bcd cd d	ACTION SHIFT REDUCE SHIFT SHIFT REDUCE REDUCE REDUCE REDUCE
--	--	---	----------------------------------	---

- A common method for dependency parsing of natural language involves a modification of the LR shift-reduce parser
- The **shift** operator continues to move items of the input string from the buffer to the stack
- The **reduce** operator is replaced with the operations **left-arc** and **right-arc** which *reduce* the top two stack symbols leaving the *head* on the stack

Consider $\mathcal{L}(G_{dep}) \subseteq \Sigma^*$, during parsing the stack may hold γab where $\gamma \in \Sigma^*$ and $a, b \in \Sigma$, and b is at the top of the stack:

- LEFT-ARC reduces the stack to γb and records use of rule $b \rightarrow a$
- RIGHT-ARC reduces the stack to γa and records the use of rule $a \rightarrow b$

$ \begin{array}{c} D \\ s \\ \mathcal{P} \\ \end{array} = \begin{array}{c} P \\ \end{array} $	$egin{array}{l} s \ \{(a o b, \mathcal{L} \mid c \ d \to e, \mathcal{R}) \end{array}$	$c, \mathcal{R} \mid d, \mathcal{R}$)		
h a	$(e ightarrow f, \mathcal{L})$	f			

Σ D s	=	$\{az\}$ $\{\mathcal{L},\mathcal{R}\}$ s	}			STACK	BUFFER bacdfe	ACTION SHIFT	RECORD
Ρ	=	$\{(a \rightarrow (d \rightarrow a)) (a \rightarrow a)\}$	<i>b</i> , <i>L</i> e, <i>R</i>) [€] , <i>L</i>)}	с, К	d, R)				
b	а	С	d	f	e				

Σ D 5 P		$ \{az\} \\ \{\mathcal{L}, \mathcal{R}\} \\ s \\ \{(a \rightarrow (d \rightarrow e)) \\ (e \rightarrow f) \} \} $	b, ℒ e, ℝ) [[] , ℒ)}	c, \mathcal{R}	∣ <i>d</i> , <i>R</i>)	stack b	BUFFER bacdfe acdfe	ACTION SHIFT	RECORD
b	а	с	d	f	e				

Σ D S P		$ \begin{array}{l} \{az\} \\ \{\mathcal{L},\mathcal{R}\} \\ s \\ \{(a \rightarrow (d \rightarrow a)) \\ (e \rightarrow b) \end{array} $	b, <i>L</i> e, <i>R</i>) f, <i>L</i>)}	c,R	∣ <i>d</i> , <i>R</i>)	STACK b	BUFFER bacdfe acdfe	ACTION SHIFT SHIFT	RECORD
b	а	с	d	f	e				

Σ	_	اح د (STACK	BUFFER	ACTION	RECORD
2	_	[α2]	1				bacdfe	SHIFT	
D	=	$\{\mathcal{L},\mathcal{K}\}$	}			b	acdfe	SHIFT	
S	=	S				ba	cdfe		
\mathcal{P}	=	$\{(a ightarrow$	$b, \mathcal{L} \mid$	c, \mathcal{R}	$ d, \mathcal{R})$				
		$(d \rightarrow d)$	$e, \mathcal{R})$						
		$(e \rightarrow i)$	$f, \mathcal{L})\}$						
Ь	2	6	d	f	0				
D	d	C	u	1	C		I	I	11

$\Sigma = \{2, 7\}$ STACK BUFFER ACTION	I RECORD
$L = \{a, b, z\}$ bacdfe SHIFT	
$D = \{L, R\}$ b acdfe SHIFT	
s = s ba cdfe LEFT-AR	$AC \parallel a \rightarrow b$
$\mathcal{P} = \{(a \to b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R})\}$	
$(d ightarrow e, \mathcal{R})$	
$(e ightarrow f, \mathcal{L})\}$	
b a c d f e	

Σ	_	∫a z	ι			STACK	BUFFER	ACTION	RECORD
2	_	[α2]	۲ ۱				bacdfe	SHIFT	
υ	=	$\{\mathcal{L},\mathcal{K}\}$	}			b	acdfe	SHIFT	
S	=	S				ba	cdfe	LEFT-ARC	$a \rightarrow b$
P	=	$\{(a \rightarrow (a \rightarrow))\}$	(b, \mathcal{L})	$ c, \mathcal{K} $	$\mathcal{L} \mid d, \mathcal{R}$	а	cdfe		
		$(a \rightarrow (a \rightarrow a))$	e, \mathcal{K}						
		(0 /	r, 2)}						
C	\neg								
Ļ									
b	а	С	d	f	е				

~		(a -)				STACK	BUFFER	ACTION	RECORD
	=	$\{a,, Z\}$					bacdfe	SHIFT	
υ	=	$\{\mathcal{L},\mathcal{K}\}$	ł			b	acdfe	SHIFT	
5 70	=	S (/ ·	1 0	Т		ba	cdfe	LEFT-ARC	a ightarrow b
Ρ	=	$\{(a \rightarrow a) \}$	$D, L \mid$	c, R	$ a, \kappa)$	а	cdfe	SHIFT	
		$(a \rightarrow)$	e, \mathcal{K}						
		(e → I	, L]}						
C	\neg								
Į									
b	а	С	d	f	е				

~		(a =	ı			STACK	BUFFER	ACTION	RECORD
	=	{aZ	} 1				bacdfe	SHIFT	
υ	=	$\{\mathcal{L},\mathcal{K}\}$	}			b	acdfe	SHIFT	
s D	=	S (/ ·				ba	cdfe	LEFT-ARC	a ightarrow b
Ρ	=	$\{(a \rightarrow$	D, L	c, R	$ a, \kappa $	а	cdfe	SHIFT	
		$(a \rightarrow$	e, K			ас	dfe		
		$(e \rightarrow$	r, L)}						
_	_								
Į									
b	a	с	d	f	е				

~		(a =	n			STACK	BUFFER	ACTION	RECORD
	=	{aZ	ך ר				bacdfe	SHIFT	
D	=	$\{\mathcal{L},\mathcal{K}\}$	}			b	acdfe	SHIFT	
s D	=	S	1 0			ba	cdfe	LEFT-ARC	a ightarrow b
Ρ	=	$\{(a \rightarrow$	b, L	c, \mathcal{R}	$[a, \mathcal{R})$	а	cdfe	SHIFT	
		$(a \rightarrow$	e, κ			ас	dfe	RIGHT-ARC	a ightarrow c
		$(e \rightarrow$	T, L)						
6	_	\frown							
Į		/ \							
b	a	c	d	f	е				

~		[a -]				STACK	BUFFER	ACTION	RECORD
2	=	{az]	ך ר				bacdfe	SHIFT	
D	=	$\{\mathcal{L},\mathcal{K}\}$	}			b	acdfe	SHIFT	
s T	=	S				ba	cdfe	LEFT-ARC	$a \rightarrow b$
P	=	$\{(a \rightarrow$	b, \mathcal{L}	c, \mathcal{R}	$ d,\mathcal{R})$	а	cdfe	SHIFT	
		$(d \rightarrow$	e, \mathcal{R}			ас	dfe	RIGHT-ARC	$a \rightarrow c$
		$(e \rightarrow)$	$f, \mathcal{L})\}$			а	dfe		
_	_	_							
ſ		\bigcap							
b	a	, * C	d	f	е				

$\Sigma = \{az\}$ $D = \{\mathcal{L}, \mathcal{R}\}$ $s = s$ $\mathcal{P} = \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R})$ $(d \rightarrow e, \mathcal{R})$ $(e \rightarrow f, \mathcal{L})\}$	STACK b ba a ac a	BUFFER bacdfe acdfe cdfe cdfe dfe dfe	ACTION SHIFT SHIFT LEFT-ARC SHIFT RIGHT-ARC SHIFT	$\begin{vmatrix} \text{RECORD} \\ a \rightarrow b \\ a \rightarrow c \end{vmatrix}$
$\int_{b} \int_{a} \int_{c} d f e$				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

~		(a –	n			STACK	BUFFER	ACTION	RECORD
2	=	{az	} \				bacdfe	SHIFT	
D	=	$\{\mathcal{L},\mathcal{K}\}$	}			b	acdfe	SHIFT	
s T	=	S				ba	cdfe	LEFT-ARC	$a \rightarrow b$
P	=	$\{(a \rightarrow$	$b, \mathcal{L} \mid$	c, \mathcal{R}	$ d,\mathcal{R})$	а	cdfe	SHIFT	
		$(d \rightarrow$	e, \mathcal{R}			ас	dfe	RIGHT-ARC	a ightarrow c
		$(e \rightarrow$	$f, \mathcal{L})\}$			а	dfe	SHIFT	
						ad	fe		
	_	_							
\int		\bigcap							
b	a	c v	d	f	e				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

$\Sigma = \{az\}$ $D = \{\mathcal{L}, \mathcal{R}\}$ $s = s$ $\mathcal{P} = \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R})$ $(d \rightarrow e, \mathcal{R})$ $(e \rightarrow f, \mathcal{L})\}$	STACK b ba a ac a a ad	BUFFER bacdfe acdfe cdfe dfe dfe fe	ACTION SHIFT SHIFT LEFT-ARC SHIFT RIGHT-ARC SHIFT SHIFT	$\begin{vmatrix} \text{RECORD} \\ a \rightarrow b \\ a \rightarrow c \end{vmatrix}$
b a c d f e				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

$\Sigma = \{az\}$ $D = \{\mathcal{L}, \mathcal{R}\}$ $s = s$ $\mathcal{P} = \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R})$ $(d \rightarrow e, \mathcal{R})$ $(e \rightarrow f, \mathcal{L})\}$	STACK b a ac a ad adf	BUFFER bacdfe acdfe cdfe dfe dfe fe e	ACTION SHIFT SHIFT LEFT-ARC SHIFT RIGHT-ARC SHIFT SHIFT	$\begin{vmatrix} \text{RECORD} \\ a \rightarrow b \\ a \rightarrow c \end{vmatrix}$
b a c d f e				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

$\Sigma = \{az \\ D = \{\mathcal{L}, \mathcal{R} \\ s = s \\ \mathcal{P} = \{(a \rightarrow (d \rightarrow (e \rightarrow (e \rightarrow (a \rightarrow (a \rightarrow (a \rightarrow (a \rightarrow (a \rightarrow (a$	$ \left. \begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\$	STACK b ba a ac a ad adf	BUFFER bacdfe acdfe cdfe dfe dfe fe e	ACTION SHIFT SHIFT LEFT-ARC SHIFT RIGHT-ARC SHIFT SHIFT SHIFT	$\begin{array}{c} \text{RECORD} \\ a \rightarrow b \\ a \rightarrow c \end{array}$
b a c	d f e				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

$\Sigma = \{az\}$ $D = \{\mathcal{L}, \mathcal{R}\}$ $s = s$ $\mathcal{P} = \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R})$ $(d \rightarrow e, \mathcal{R})$ $(e \rightarrow f, \mathcal{L})\}$	STACK b ba a ac ad adf adfe	BUFFER bacdfe acdfe cdfe cdfe dfe dfe fe e	ACTION SHIFT SHIFT LEFT-ARC SHIFT RIGHT-ARC SHIFT SHIFT SHIFT	$\begin{vmatrix} \text{RECORD} \\ a \rightarrow b \\ a \rightarrow c \end{vmatrix}$
b a c d f e				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

Σ	_	{a z}	STACK	BUFFER	ACTION	RECORD
5		[α2] [C D]		bacdfe	SHIFT	
D	=	$\{\mathcal{L},\mathcal{K}\}$	b	acdfe	SHIFT	
S D	=	$\int \left(a + b \right) + c + a + a + a + a + a + a + a + a + a$	ba	cdfe	LEFT-ARC	$a \rightarrow b$
Ρ	=	$\{(a \to b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \mid (d \to c, \mathcal{R})\}$	а	cdfe	SHIFT	
		$(u \rightarrow e, \mathcal{R})$	ас	dfe	RIGHT-ARC	a ightarrow c
		$(e \rightarrow I, \mathcal{L})$	а	dfe	SHIFT	
			ad	fe	SHIFT	
			adf	e	SHIFT	
			adfe		LEFT-ARC	$e \rightarrow f$
∫ b	a	$ \begin{array}{ccc} & & & \\ & & \\ c & d & f & e \end{array} $				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

~		()				STACK	BUFFER	ACTION	RECORD
2	=	$\{az\}$					bacdfe	SHIFT	
D	=	$\{\mathcal{L},\mathcal{K}\}$				b	acdfe	SHIFT	
s D	=	S				ba	cdfe	LEFT-ARC	$a \rightarrow b$
Ρ	=	$\{(a \rightarrow a) \}$	$b, L \mid$	$c, \mathcal{R} \mid$	$d, \mathcal{R})$	а	cdfe	SHIFT	
		$(d \rightarrow e, \mathcal{R})$	ас	dfe	RIGHT-ARC	a ightarrow c			
		$(e \rightarrow t$	$(e \rightarrow f, \mathcal{L})$ }				dfe	SHIFT	
						ad	fe	SHIFT	
						adf	е	SHIFT	
						adfe		LEFT-ARC	$e \rightarrow f$
	_	_		_		ade			
ſ		\bigcap		\square					
b	à	/ * C	d	f \	е				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

~		STACK	BUFFER	ACTION	RECORD
2 =	$\{az\}$		bacdfe	SHIFT	
<i>D</i> =	$\{\mathcal{L},\mathcal{K}\}$	b	acdfe	SHIFT	
s =	s	ba	cdfe	LEFT-ARC	$a \rightarrow b$
P =	$\{(a \to b, \mathcal{L} \mid c, \mathcal{K} \mid d, \mathcal{K})\}$	а	cdfe	SHIFT	
	$(a \rightarrow e, \mathcal{K})$	ас	dfe	RIGHT-ARC	a ightarrow c
	$(e \rightarrow r, \mathcal{L})$	а	dfe	SHIFT	
		ad	fe	SHIFT	
		adf	e	SHIFT	
		adfe		LEFT-ARC	$e \rightarrow f$
_		ade		RIGHT-ARC	d ightarrow e
\int					
b a	c d f e				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

						ı		
~	_		1		STACK	BUFFER	ACTION	RECORD
	_	{d2}			bacdfe	SHIFT		
υ	=	$\{\mathcal{L},\mathcal{K}\}$	}		b	acdfe	SHIFT	
5 10	=	S (/ .		$\mathcal{D} \vdash \mathcal{L} \mathcal{D}$	ba	cdfe	LEFT-ARC	a ightarrow b
Ρ	$= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \mid (c, \mathcal{R} \mid d, \mathcal{R})\}$				а	cdfe	SHIFT	
		$(a \rightarrow e, \mathcal{K})$			ac	dfe	RIGHT-ARC	a ightarrow c
		$(e \rightarrow r, \mathcal{L})$				dfe	SHIFT	
					ad	fe	SHIFT	
					adf	e	SHIFT	
					adfe		LEFT-ARC	$e \rightarrow f$
_	_	_			ade		RIGHT-ARC	d ightarrow e
ſ					ad			
b	a	с	d	fe				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

~		()	STACK	BUFFER	ACTION	RECORD
2	=	$\{az\}$		bacdfe	SHIFT	
D	=	$\{\mathcal{L},\mathcal{R}\}$	b	acdfe	SHIFT	
s D	=	s	ba	cdfe	LEFT-ARC	$a \rightarrow b$
P	=	$\{(a \to b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{R}) \mid (d, \mathcal{R}) \in \mathcal{D}\}$	а	cdfe	SHIFT	
		$(a \rightarrow e, \mathcal{K})$	ac	dfe	RIGHT-ARC	$a \rightarrow c$
		$(e \rightarrow f, \mathcal{L})$	а	dfe	SHIFT	
			ad	fe	SHIFT	
			adf	е	SHIFT	
		\frown	adfe		LEFT-ARC	$e \rightarrow f$
_	_		ade		RIGHT-ARC	$d \rightarrow e$
ſ			ad		RIGHT-ARC	a ightarrow d
b	а	c d f e				

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

~		(a =	า			STACK	BUFFER	ACTION	RECORD
2	=	{az	}				bacdfe	SHIFT	
D	=	$\{\mathcal{L},\mathcal{K}\}$				b	acdfe	SHIFT	
s D	=	S		D		ba	cdfe	LEFT-ARC	a ightarrow b
Ρ	=	$= \{(a \rightarrow b, \mathcal{L} \mid c, \mathcal{R} \mid d, \mathcal{K} \mid d, \mathcal{K} \mid c, \mathcal{R} \mid d, \mathcal{K} \mid d, K$			a, R)	а	cdfe	SHIFT	
		$(a \rightarrow e, R)$	e, K			ас	dfe	RIGHT-ARC	a ightarrow c
	$(e \rightarrow r, \mathcal{L})$					а	dfe	SHIFT	
						ad	fe	SHIFT	
						adf	e	SHIFT	
		\frown				adfe		LEFT-ARC	$e \rightarrow f$
	_	$ \land \land$		_		ade		RIGHT-ARC	d ightarrow e
ſ			ad		RIGHT-ARC	a ightarrow d			
b	a	, c	ď	ŕ	e	а			

Example of shift-reduce parse for the string *bacdfe* generated using $G_{dep} = (\Sigma, D, s, \bot, P)$

~		(ı			STACK	BUFFER	ACTION	RECORD
2	=	{az	}				bacdfe	SHIFT	
D	=	$= \{\mathcal{L}, \mathcal{R}\}$				b	acdfe	SHIFT	
s D	=	S		Ð		ba	cdfe	LEFT-ARC	a ightarrow b
Ρ	=	{(<i>a</i> →	$(\mathcal{D}, \mathcal{L} \mid \mathcal{D})$	c, κ	(a, R)	а	cdfe	SHIFT	
	(d ightarrow e					ас	dfe	RIGHT-ARC	a ightarrow c
	$(e \rightarrow r, \mathcal{L})$					а	dfe	SHIFT	
	1					ad	fe	SHIFT	
						adf	e	SHIFT	
			\neg		<u>\</u>	adfe		LEFT-ARC	e ightarrow f
\int			.	ade		RIGHT-ARC	d ightarrow e		
		$\left(\right)$		ad		RIGHT-ARC	a ightarrow d		
b	a	c t	ď	ŕ	e	а		TERMINATE	$ $ root \rightarrow a

- For natural language there would be considerable effort in manually defining \mathcal{P} —this would involve determining the dependencies between all possible words in the language (although note that e.g. RASP uses a similar transition-based approach with a manually defined PSG)
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items in the **configuration** as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

- For natural language there would be considerable effort in manually defining \mathcal{P} —this would involve determining the dependencies between all possible words in the language (although note that e.g. RASP uses a similar transition-based approach with a manually defined PSG)
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items in the **configuration** as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

- For natural language there would be considerable effort in manually defining \mathcal{P} —this would involve determining the dependencies between all possible words in the language (although note that e.g. RASP uses a similar transition-based approach with a manually defined PSG)
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items in the **configuration** as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

- For natural language there would be considerable effort in manually defining \mathcal{P} —this would involve determining the dependencies between all possible words in the language (although note that e.g. RASP uses a similar transition-based approach with a manually defined PSG)
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items in the **configuration** as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

- For natural language there would be considerable effort in manually defining \mathcal{P} —this would involve determining the dependencies between all possible words in the language (although note that e.g. RASP uses a similar transition-based approach with a manually defined PSG)
- Creating a deterministic grammar would be impossible (natural language is inherently ambiguous).
- Natural language dependency parsing can achieved deterministically by **selecting parsing actions** using a machine learning **classifier**.
- The **features** for the classifier include the items in the **configuration** as well as properties of those items (including **word-embeddings** for the items).
- Training is performed on **dependency banks** (that is, sentences that have been manually annotated with their correct dependencies).
- It is said that the parsing is **grammarless**—since no grammar is designed ahead of training.

We can use feature templates to analyse a configuration

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)

$$s_{2}.t \circ s_{1}.t \circ b_{1}.t; s_{2}.t \circ s_{1}.t \circ lc_{1}(s_{1}).t; \\s_{2}.t \circ s_{1}.t \circ rc_{1}(s_{1}).t; s_{2}.t \circ s_{1}.t \circ lc_{1}(s_{2}).t; \\s_{2}.t \circ s_{1}.t \circ rc_{1}(s_{2}).t; s_{2}.t \circ s_{1}.w \circ rc_{1}(s_{2}).t; \\s_{2}.t \circ s_{1}.w \circ lc_{1}(s_{1}).t; s_{2}.t \circ s_{1}.w \circ b_{1}.t$$

Chen and Manning, ACL, 2014

There are problems with the features

- The features are indispensable but highly sparse.
- The feature templates are **incomplete**.
- The features are **expensive to compute**.

Chen and Manning present NN dependency parser in 2014



Consider sets S^w, S^t, S^t : e.g. $S^t = \{lc_1(s_2), t, s_2, t, rc_1(s_2), t, s_1, t\} \rightarrow PRP, VBZ, NULL, JJ$ and w^t is the concatenation of the tag embeddings.

Chen and Manning present NN dependency parser in 2014



Consider sets S^w, S^t, S^l : e.g. $S^t = \{lc_1(s_2).t, s_2.t, rc_1(s_2).t, s_1.t\} \rightarrow PRP, VBZ, NULL, JJ$ and w^t is the concatenation of the tag embeddings.

Chen and Manning use a rich feature sets

- S^w contains 18 feature templates:
 - $1\;$ The top 3 words on the stack and buffer

 $s_1, s_2, s_3, b_1, b_2, b_3$

2 The first and second leftmost and rightmost children of the top two words on the stack

 $lc_1(s_i), rc_1(s_i), lc_2(s_i), rc_2(s_i)$ where i = 1, 2

3 The leftmost of leftmost and rightmost of rightmost children of the top two words on the stack.

 $lc_1(lc_1(s_i)), rc_1(rc_1(s_i))$ where i = 1, 2

 S^T is the corresponding tags

S' the corresponding arc labels (excludes category 1 above)

Chen and Manning use a rich feature sets

- S^w contains 18 feature templates:
 - $1\;$ The top 3 words on the stack and buffer

 $s_1, s_2, s_3, b_1, b_2, b_3$

2 The first and second leftmost and rightmost children of the top two words on the stack

 $lc_1(s_i), rc_1(s_i), lc_2(s_i), rc_2(s_i)$ where i = 1, 2

3 The leftmost of leftmost and rightmost of rightmost children of the top two words on the stack.

 $lc_1(lc_1(s_i)), rc_1(rc_1(s_i))$ where i = 1, 2

- S^{T} is the corresponding tags
- S' the corresponding arc labels (excludes category 1 above)

Details of the original Chen and Manning

- Generate training examples (pairing real configurations with their gold parsing actions) from dependancy bank using a shortest stack oracle
- Training objective is to minimize cross-entropy loss
- Back-propagation to the embeddings during training
- During parsing they use greedy decoding.
- Further improvements in recent years.