Lecture 6: Transforming Dependency to Context-free Grammars

Jelle Zuidema
ILLC, Universiteit van Amsterdam

Unsupervised Language Learning, 2014

Plan for today

Recap

- UDOP
- DG
- CH

Split-head transform

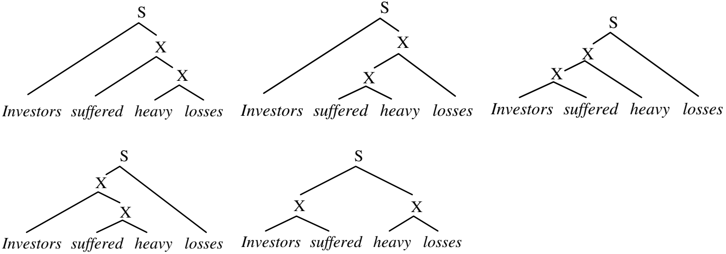
- Projective Bilexical Dependency Grammars
- Split-head encoding
- The fold-unfold transform

(based on slides from Mark Johnson)

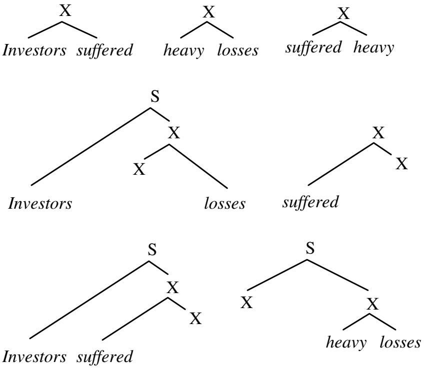
How Does U-DOP Operate?

1. Assign *all* possible binary trees to strings where each root node is labeled *S* and other nodes labeled *X*, and store them in a parse forest

E.g., for WSJ sentence *Investors suffered heavy losses*:



2. Convert the set of all trees into all subtrees. For instance:



=> Note that some subtrees contain discontiguous yields

3. Compute most probable tree among shortest derivations for new string (as in DOP):

Probability of…

a *subtree* t :

$$P(t) = \frac{|t|}{\sum_{t' : root(t')=root(t)} |t'|}$$

a *derivation* $d = t_1 \circ \ldots \circ t_n$:

$$P(t_1 \circ \ldots \circ t_n) = \prod_i P(t_i)$$

a *parse tree* T :

$$P(T) = \sum_d \prod_i P(t_{id})$$

U-DOP compared to other models on WSJ-10

(using 7422 sentences up to **10** words, as in Klein and Manning 2004)

Model	**F-score on WSJ-10**
CCM	71.9
DMV	52.1
DMV+CCM	77.6
U-DOP	**82.7**
U-DOP without discontiguous subtrees	**72.1**

CCM: Klein and Manning (2002) based on all linear (contiguous) contexts *without* holes

DMV: Klein and Manning (2004) using dependency structures

U-DOP: equivalent to CCM *plus* discontiguous contexts *with* holes: *11% improvement in F-score*

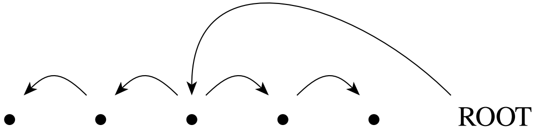
Shortcomings of U-DOP

- Viewed from the statistical inference perspective, the model relies much on heuristics: initialization, training & stopping
- Results with UML-DOP (Bod,06) suggests it is approximately Maximum Likelihood...
- ... but not over the entire PTSG space, as there are exponentially many subtrees, and exponentially many trees for a sentence!
- Implementation must somehow restrict space; efficiency remains the achilles heel.

Definitions

Definitions:

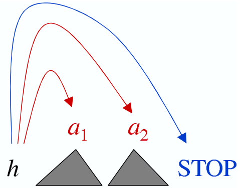
- a dependency d is a pair $\langle h, a \rangle$, where h is the head and a the argument, both are words in sentence s;
- a dependency structure D is set of dependencies which form a planar, acyclic graph rooted in ROOT;
- the skeleton G of a dependency structure specifies the arrows, but not the words; G and s together fully determine the dependency structure.



Dependency Model with Valence

Klein and Manning (2004) propose a model that generates dependencies outwards from the head:

- generate a set of arguments on one side of the head, then a STOP argument to terminate;
- the do the same thing on the other side;
- terminate with probability P_{STOP}, if not STOP then choose another argument with probability P_{CHOOSE}.



Extended Chomsky Hierarchy

language	grammar	rules
$\{a, b, cbabb\}$	Set	$\in$
$(ab)^n$	ngram	$\langle a, b\rangle, \langle b, a\rangle, \langle ab, a\rangle$
$a^n ba^m$	Left-linear	$S \rightarrow AB, B \rightarrow bA$
$a^n b^n$	Context-free	$S \rightarrow aSb, S \rightarrow ab$
$a^n b^n c^n d^n \vert 1 \leq n$	Range Concatenation	$S[abc] \rightarrow A[a, c]B[b]$
	Unrestricted	

Probabilistic Extensions

grammar	probabilistic grammar
Set	Probability distribution
ngram	Markov model
Left-linear	Hidden Markov (HMM)
Context-free	PCFG
Range Concatenation	PLCRS
Unrestricted	

CFG encoding of Dependency Grammars

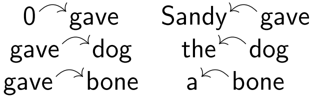
- Given that dependency grammars must be somewhere on the CH, presumable below contextfree, can we reuse the technology we developed for context-free grammars (rule extraction, CYK, Inside algorithm, Inside-outside) for dependency grammars?
- Yes!- at least for some kind of dependency grammars and given the right preprocessing.

Plan for today

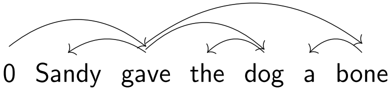
- Recap
  - UDOP
  - DG
  - CH
- Split-head transform
  - Projective Bilexical Dependency Grammars
  - Split-head encoding
  - The fold-unfold transform

Projective Bilexical Dependency Grammars

- Projective Bilexical Dependency Grammar (PBDG)



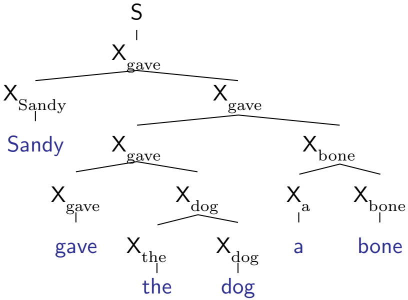
- A dependency parse generated by the PBDG



- Weights can be attached to dependencies (and preserved in CFG transforms)

A naive encoding of PBDGs as CFGs

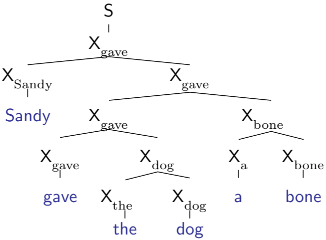
$$\begin{array}{rcll} \mathsf{S} & \rightarrow & \mathsf{X}_u & \text{where } 0 \curvearrowright u \\ \mathsf{X}_u & \rightarrow & u & \\ \mathsf{X}_u & \rightarrow & \mathsf{X}_v\,\mathsf{X}_u & \text{where } v \curvearrowleft u \\ \mathsf{X}_u & \rightarrow & \mathsf{X}_u\,\mathsf{X}_v & \text{where } u \curvearrowright v \end{array}$$

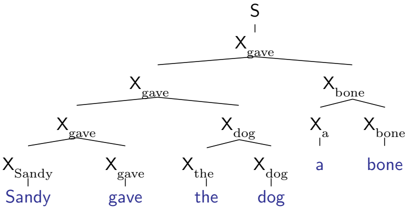


Spurious ambiguity in naive encoding

- Naive encoding allows dependencies on different sides of head to be freely reordered

⇒ Spurious ambiguity in CFG parses (not present in PBDG parses)



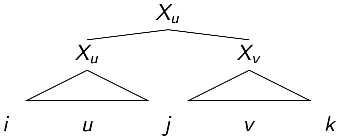


Parsing naive CFG encoding takes $O(n^5)$ time

- A production schema such as

$$\mathrm{X}_u \;\rightarrow\; \mathrm{X}_u\,\mathrm{X}_v$$

has 5 variables, and so can match input in $O(n^5)$ different ways



Plan for today

Recap
  UDOP
  DG
  CH

Split-head transform
  Projective Bilexical Dependency Grammars
  Split-head encoding
  The fold-unfold transform

Simple split-head encoding

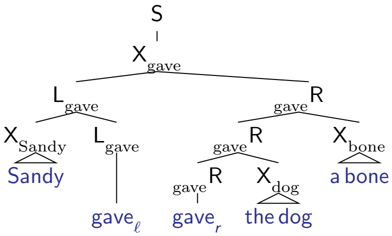
- Replace input word u with a *left variant* u_ℓ and a *right variant* u_r (can be avoided in practice with fancy book-keeping)

Sandy gave the dog a bone

$\Downarrow$

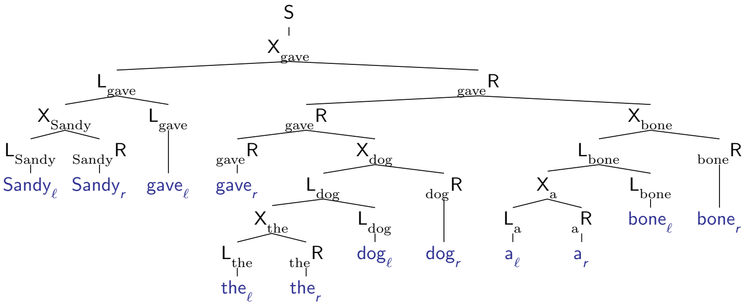
Sandy$_\ell$ Sandy$_r$ gave$_\ell$ gave$_r$ the$_\ell$ the$_r$ dog$_\ell$ dog$_r$ a$_\ell$ a$_r$ bone$_\ell$ bone$_r$

- PCFG separately collects left dependencies and right dependencies



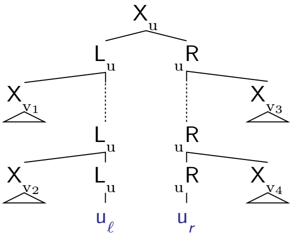
$$\begin{array}{lcll} S & \rightarrow & X_u & \text{where } 0 \curvearrowright u \\ X_u & \rightarrow & L_u \ {}_uR & \text{where } u \in \Sigma \\ L_u & \rightarrow & u_l & \\ L_u & \rightarrow & X_v \ L_u & \text{where } v \curvearrowleft u \\ {}_uR & \rightarrow & u_r & \\ {}_uR & \rightarrow & {}_uR \ X_v & \text{where } u \curvearrowright v \end{array}$$

Simple split-head CFG parse



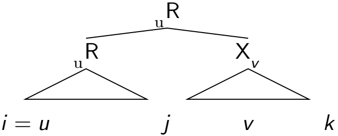
L_u and ${}_uR$ heads are phrase-peripheral ⇒ $O(n^4)$

- Heads of L_u and ${}_uR$ are always at right (left) edge



$$\begin{array}{rcll}
S & \rightarrow & X_u & \text{where } 0 \curvearrowright u \\
X_u & \rightarrow & L_u \ {}_uR & \text{where } u \in \Sigma \\
L_u & \rightarrow & u_l & \\
L_u & \rightarrow & X_v \ L_u & \text{where } v \curvearrowleft u \\
{}_uR & \rightarrow & u_r & \\
{}_uR & \rightarrow & {}_uR \ X_v & \text{where } u \curvearrowright v
\end{array}$$

- $X_u \rightarrow L_u \ {}_uR$ take $O(n^3)$
- ${}_uR \rightarrow {}_uR \ X_v$ take $O(n^4)$



Plan for today

- Recap
  - UDOP
  - DG
  - CH
- Split-head transform
  - Projective Bilexical Dependency Grammars
  - Split-head encoding
  - The fold-unfold transform

The Unfold-Fold transform

- Unfold-fold originally proposed for transforming recursive programs; used here to transform CFGs into new CFGs
- *Unfolding* a nonterminal replaces it with its expansion

$$\begin{array}{l} A \rightarrow \alpha\, B\, \gamma \\ B \rightarrow \beta_1 \\ B \rightarrow \beta_2 \\ \quad \ldots \end{array} \quad \Rightarrow \quad \begin{array}{l} A \rightarrow \alpha\, \beta_1\, \gamma \\ A \rightarrow \alpha\, \beta_2\, \gamma \\ B \rightarrow \beta_1 \\ B \rightarrow \beta_2 \\ \quad \ldots \end{array}$$

- *Folding* is the inverse of unfolding (replace RHS with nonterminal)

$$\begin{array}{l} A \rightarrow \alpha\, \beta\, \gamma \\ B \rightarrow \beta \\ \quad \ldots \end{array} \quad \Rightarrow \quad \begin{array}{l} A \rightarrow \alpha\, B\, \gamma \\ B \rightarrow \beta \\ \quad \ldots \end{array}$$

- Transformed grammar generates same language (Sato 1992)

Unfold-fold converts $O(n^4)$ to $O(n^3)$ grammar

- Unfold X_v responsible for $O(n^4)$ parse time

$$\begin{array}{lcl} \mathrm{L}_u & \rightarrow & u_l \\ \mathrm{L}_u & \rightarrow & \mathrm{X}_v \ \mathrm{L}_u \\ \mathrm{X}_v & \rightarrow & \mathrm{L}_v \ {}_v\mathrm{R} \end{array} \quad \Rightarrow \quad \begin{array}{lcl} \mathrm{L}_u & \rightarrow & u_l \\ \mathrm{L}_u & \rightarrow & \mathrm{L}_v \ {}_v\mathrm{R} \ \mathrm{L}_u \end{array}$$

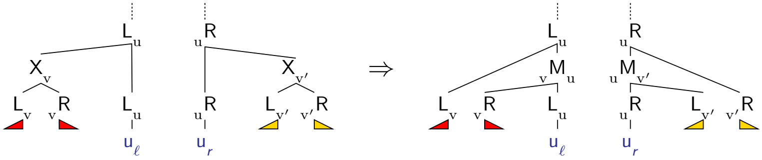
- Introduce new non-terminals ${}_x\mathrm{M}_y$ (doesn't change language)

$${}_x\mathrm{M}_y \rightarrow {}_x\mathrm{R} \ \mathrm{L}_y$$

- Fold two children of L_u into ${}_x\mathrm{M}_y$

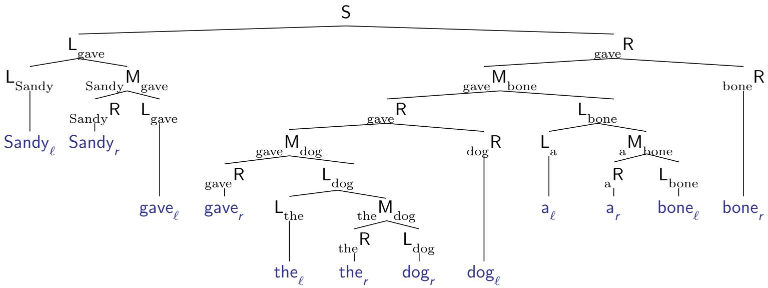
$$\begin{array}{lcl} \mathrm{L}_u & \rightarrow & u_l \\ \mathrm{L}_u & \rightarrow & \mathrm{L}_v \ {}_v\mathrm{R} \ \mathrm{L}_u \\ {}_x\mathrm{M}_y & \rightarrow & {}_x\mathrm{R} \ \mathrm{L}_y \end{array} \quad \Rightarrow \quad \begin{array}{lcl} \mathrm{L}_u & \rightarrow & u_l \\ \mathrm{L}_u & \rightarrow & \mathrm{L}_v \ {}_v\mathrm{M}_u \\ {}_x\mathrm{M}_y & \rightarrow & {}_x\mathrm{R} \ \mathrm{L}_y \end{array}$$

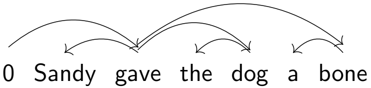
Transformed grammar collects left and right dependencies separately



- X_v constituents (which cause $O(n^4)$ parse time) no longer used
- Head annotations now all phrase peripheral ⇒ $O(n^3)$ parse time
- Dependencies can be recovered from parse tree
- Basically same as Eisner and Satta $O(n^3)$ algorithm
    - explains why Inside-Outside sanity check fails for Eisner/Satta
    - two copies of each terminal ⇒ each terminals' Outside probability is *double* the Inside sentence probability

Parse using $O(n^3)$ transformed split-head grammar





Parsing time of CFG encodings of same PBDG

CFG schemata	sentences parsed / second
Naive $O(n^5)$ CFG	45.4
$O(n^4)$ simple split-head CFG	406.2
$O(n^3)$ transformed split-head CFG	3580.0

- Weighted PBDG; all pairs of heads have some dependency weight
- Dependency weights precomputed before parsing begins
- Timing results on a 3.6GHz Pentium 4 machine parsing section 24 of the PTB
- CKY parsers with grammars hard-coded in C (no rule lookup)
- Dependency accuracy of Viterbi parses $= 0.8918$ for all grammars
- *Feature extraction is much slower than even naive CFG*