Towards Surface Realization with CCGs Induced from Dependencies

Michael White
Department of Linguistics, The Ohio State University

SUMMARY

- In the first surface realization (SR) shared task (Belz et al., 2011), no grammar-based systems achieved competitive results, as input conversion turned out to be more difficult than anticipated
- However, a silver lining of the failure of grammar-based systems in the shared task is that it revealed some problems with the data (right)
- Since then, Narayan & Gardent (2012) have shown that grammar-based systems can be substantially improved with error mining techniques
- In this paper, inspired by recent work on converting dependency treebanks (Ambati et al., 2013) and semantic parsing (Kwiatkowski et al., 2010; Artzi & Zettlemoyer, 2013) with Combinatory Categorial Grammar (CCG), we pursue the alternative strategy of inducing a CCG from an enhanced version of the shared task dependencies, with initial experiments showing even better results, competitive with the best systems in the first SR shared task

Algorithm

- In contrast to Ambati et al.'s (2013) approach, the algorithm integrates the proposal of candidate lexical categories with the derivational process, making it possible to derive categories involving unsaturated arguments, such as $\mathsf{s}_{e,\mathsf{dcl}}\backslash\mathsf{np}_x/(\mathsf{s}_{e',\mathsf{to}}\backslash\mathsf{np}_x)$
- It also makes greater use of unary type-changing rules, as with Artzi & Zettlemoyer's (2013) approach; unlike their approach though, it works in a broad coverage setting, and makes use of all the combinators standardly used with CCG, including ones for type-raising
- The algorithm (right) makes use of a seed lexicon that specifies category projection by part of speech, as in (1), as well as a handful of categories for function words, as in (2)
- After lexical instantiation, the algorithm recursively visits each node in the primary dependency tree bottom up (*combineEdges*), using a local chart (*doCombos*) at each step to combine categories for adjacent phrases in all possible ways, along the way creating new categories (*extendCats* and *coordCats*) and unary rules (*applyNewUnary*), as in (3)-(5)

Experiments and Future Work

- We ran the induction algorithm over the standard PTB training sections (02–21), selecting preferred derivations with a simple complexity metric counting steps and slashes, then training a realization ranking model as in previous work (White & Rajkumar, 2012)
- Focusing only on the 80.5% of the sentences for which a complete derivation was found (COMPLETE) yielded a BLEU score of 0.8668 on the devset
- Section 23 results of 0.8260 were much higher than Narayan & Gardent's (2012) score of 0.675 with 38.8% coverage, though still trailing the shared task scores of the top statistical dependency realizers by several points (STUMABA-S at 0.8911 and DCU at 0.8575)
- In future work, we hope to close the gap with the top systems by integrating an improved ranking model into the induction process and resolving the remaining representational issues with problematic constructions

Model	Exact	Complete	BLEU
Sect 00 (dev)			
GEN	2.4	79.5	0.4513
GLOBAL	29.7	79.0	0.7679
NO-BIGLM	29.1	78.2	0.7757
NO-DEPORD	34.3	77.9	0.7956
ALL	35.8	78.4	0.8083
ALL+	36.4	80.5	**0.8210**
COMPLETE	44.4	-	0.8668
NATIVE	48.0	88.7	0.8793
Sect 23 (test)			
GEN	2.8	80.3	0.4560
GLOBAL	31.3	78.5	0.7675
ALL	37.6	77.2	0.8083
ALL+	38.1	80.4	**0.8260**
COMPLETE	47.0	-	0.8743
NATIVE	46.4	86.4	0.8694

Acknowledgements

Thanks to the anonymous reviewers, Richard Johansson and the University of Sydney Schwa Lab for helpful comments and discussion. This work was supported in part by NSF grants IIS-1143635 and IIS-1319318.

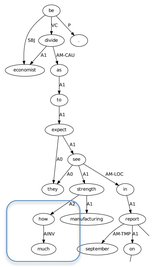
Information Loss in First SR Shared Task Inputs

Free Object Relative Example

Economists are divided as to [how much manufacturing strength]$_i$ they expect to see t_i in September reports on industrial production and capacity utilization , also due tomorrow (wsj_2400.6, "deep" representation)

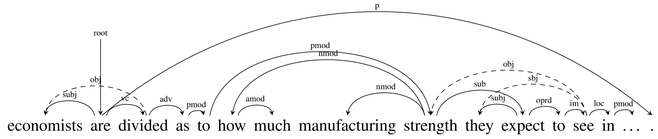
- Here, only one of the two roles played by *how much manufacturing strength* is captured in the deep representation, making it difficult to linearize this phrase correctly
- By contrast, below is an experimental version of the shallow representation intended to capture **all** the syntactic dependencies in the PTB (kudos to Richard Johansson!); doing so makes it feasible to define a compatible CCG

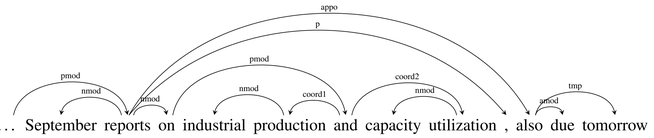
First SR Shared Task Input

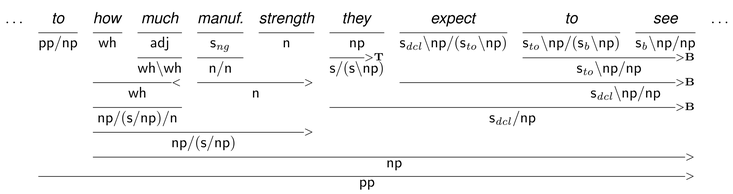


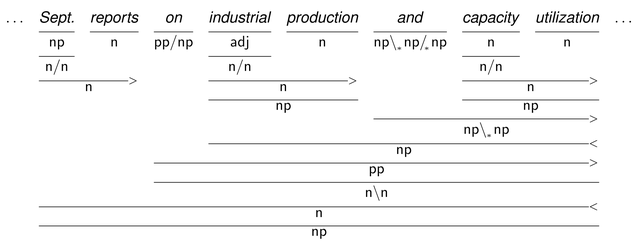
Inducing CCGs from Augmented Syntactic Dependencies

Augmented Syntactic Dependencies with CCG Derivation









Initial Categories

(1) *expect* $\vdash \mathsf{s}_{e,\mathsf{dcl}}$: $@_e(\mathbf{expect} \wedge \langle\text{TENSE}\rangle pres)$

(2) *the* $\vdash \mathsf{np}_x/\mathsf{n}_x$: $@_x(\langle\text{NMOD}\rangle(d \wedge \mathbf{the}))$

Induced Categories

(3) *expect* $\vdash \mathsf{s}_{w_{10},\mathsf{dcl}}\backslash\mathsf{np}_{w_9}/(\mathsf{s}_{w_{11},\mathsf{to}}\backslash\mathsf{np}_{w_9})$: $@_{w_{10}}(\mathbf{expect} \wedge \langle\text{TENSE}\rangle pres \wedge \langle\text{SUBJ}\rangle w_9 \wedge \langle\text{OPRED}\rangle w_{11})$

(4) *and* $\vdash \mathsf{np}_{w_{19}}\backslash_*\mathsf{np}_{w_{18}}/_*\mathsf{np}_{w_{21}}$: $@_{w_{19}}(\mathbf{and} \wedge \langle\text{COORD1}\rangle w_{18} \wedge \langle\text{COORD2}\rangle w_{21})$

(5) $\mathsf{n}_{w_{20}} \Rightarrow \mathsf{n}_{w_{21}}/\mathsf{n}_{w_{21}}$: $@_{w_{21}}(\langle\text{NMOD}\rangle w_{20})$

Grammar Induction

Inputs Training set of sentences with dependencies. Initial lexicon and rules. Argument and modifier relations. Derivation scoring metric. Maximum agenda size.

Definitions *edges* is a map from dependency graph nodes to their edges, where an *edge* is a CCG sign together with a coverage bitset; *agenda* is a priority queue of edges sorted by the scoring metric; *chart* manages equivalence classes of edges; see text for descriptions of auxiliary functions such as *extendCats* and *coordCats* below.

Algorithm
bestDerivs, lexcats, unaryRules ← ∅
For each item in training set:

1. *edges*[*node*] ← *instCats*(*node*), *ruleInsts*[*node*] ← *instRules*(*node*), for *node* in input graph
2. *combineEdges*(*root*), with *root* of input graph
3. *bestEdge* ← unpack(*edges*[*root*]); *bestDerivs* +← *bestEdge.sign*; *lexcats* +← abstractedCats(*bestEdge*), *unaryRules* +← abstractedRules(*bestEdge*), if *bestEdge* complete

def *combineEdges*(*node*):

1. *combineEdges*(*child*) for *child* in *node.kids*
2. *edges*[*node*] +← *coordCats*(*node*) if *node* has coord relations, otherwise *edges*[*node*] ← *extendCats*(*node*,*rels*) for argument *rels*
3. *agenda* ← *edges*[*node*]; *agenda* +← *edges*[*child*] for *child* in *node.kids*; *chart* ← ∅
4. While *agenda* not empty:
   (a) *next* ← *agenda.pop*
   (b) *chart* +← *next*
   (c) *doCombos*(*next*), unless *next* packed into an existing chart item
5. *edges[node]* ← chart edges for *node* filtered for maximal input coverage

def *doCombos*(*next*):

1. *agenda* +← *applyUnary*(*next*), if *next* is for *node*
2. For *item* in *chart*:
   (a) *agenda* +← *applyBinary*(*next*,*item*), if *next* is adjacent to *item*
   (b) *agenda* +← *applyNewUnary*(*next*,*item*), if *next* connected to *item* by a modifier relation

Outputs *bestDerivs, lexcats, unaryRules*