Split-Emit Process for Natural Language Generation

Ni Lao, 2009-5-7

1 Introduction

There have been plenty of non-statistical Natural Language Generation (NLG) methods in both top-down manner (Dymetman& Isabelle, 1988; Wedekind, 1988) and bottom-up manner (Shieber et al., 1990). As Van Noord (1994) summarized, the main problem of top-down approaches is the possibility of infinite recursion during generation, and the main problems of bottom-up approaches is the severe restriction on possible grammars and rather inefficient processing.

Unlike parsing, for which word order is given, NLG may encounter factorial number of possible word orders (Carroll et al., 1999). This leads to potentially exponential time complexity even with carefully designed algorithm on packed chart. The fact that many arcs generated are never used in a full derivation tree has a significant effect on scalability.

Recent statistical NLG works (Langkilde, 2000; White & Baldridge, 2003; Nakanishi et al., 2005; White et al., 2007) use packed charts to sum out all possible parse trees during training, and to find best parse tree during generation. Miyao & Tsujii (2005) report only 418 out of 500 randomly sampled sentences were successfully generated with a HPSG grammar trained on Penn TreeBank, and some of them take extremely long time to generate. White et al. (2007) report that only 22% of test sentences are successfully generated with a CCG grammar trained from Penn TreeBank, and the search exceeds time limit for 68% of the sentences. Therefore, for most of the sentences, they need to apply post processing to concatenate text fragments into a sentence. These results expose a dilemma of the need of a less permissive grammar to restrict search space to be affordable, and the need of a more permissive grammar to accept more unseen sentences.

In this proposal, we present a fast top-down statistical generation algorithm with permissive CFG grammar. The algorithm, called Split-Emit, can be considered as a reversed shift-reduce algorithm—with two operations split and emit as opposed to reduce and shift. It starts with the whole semantics and split it recursively, and has O(n) generation complexity with respect to size of the input semantics. In order to avoid the possible infinite recursion we add constraint to the split process which forbids the generation of node that already exists. To our knowledge, this is the first statistical top-down generator.

2 The Semantic DAG

Let's first introduce how the semantics of a natural language sentence can be represented as a Directed Acyclic Graph (DAG). Following Nakanishi et al.'s (2005) work, we use Predicate-Argument Structure (PAS) as the semantic representation. For example, the sentence "*He bought the book.*" has the following four PASs:

$$\text{buy}(e) \wedge \text{past}(e) \wedge \text{ARG1}(e,x) \wedge \text{ARG2}(e,y)$$
$$\text{the}(z) \wedge \text{ARG1}(y)$$
$$\text{he}(x)$$
$$\text{book}(y)$$

Each lowercase letter represents an entity, and the semantics of a sentence can be interpreted as a graph with entities as nodes (decorated with labels, e.g. "past tense") and argument relations as edges.

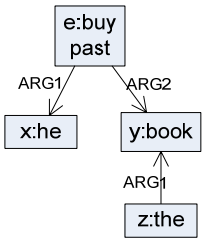

Figure 1 : Graph representation of semantics

For the purpose of text generation, a semantic meaning *M* in PASs representation can be augmented with a CFG tag *L*, and a head pointer *p* pointing to the entity in *M* that is going to be the head of the generated text. For example, if *M* is the same as Figure 1, different text may be generated from different heads or syntactic labels:

<M, S, e>: he bought the book
<M, NP, x>: he who bought the book
<M, NP, y>: the book that he bought

In this way, we maximally separate the semantic aspect of a piece of text (*M*) from its syntactic aspects (*L* and *p*).

This representation is different from the original PASs definition of Nakanishi et al.'s (2005) where each word corresponds to exactly one PAS. In this study, functional words like *to*, *is*,

that are considered syntactic and have no semantic representation.

This representation is also different from HLDS (Hybrid Logic Dependency Semantics) employed by White and Baldridge (2003). In order to generate "the book that he bought" from HLDS, y:book needs to have an argument *GenRel* pointing back to e:buy, where *GenRel* is a general relation as in relative clause modification. In this study, this kind of information is consider syntactic and represented by head pointer p.

For HPSG (Sag et al., 2003) information related to voice, tense etc are specified in syntactic structure. In our representation, this information is specified in semantic graph as labels to the nodes (e.g. passive(x), past(x)).

In the rest of this proposal we assume that semantic structures are representable as DAG. This is ensured by most of the current semantic theories, like Montague Semantics (MS), Discourse Representation Theory (DRT) and Situation Semantics (SS) (Reyle 1988; Halvorsen 1987). As a future work we can also develop heuristic rules to break circles by duplicating some of the nodes.

3 The Split-Emit Generation Process

Here we describe a generative process that transforms a semantic DAG to text. We assume that M is either *atomic* that can be directly transformed into surface text (a word or phrase) or *composite* that can be split into two meanings M_0, M_1, so that M_0 is to be generated before M_1.

Start with the tuple $<M, L, p>$. It is recursively splited until the semantics is atomic, and emits surface words as shown in Figure 1. After M is fully splited we get both a binary syntactic tree, and a piece of text (a sentence, if $L=S$, the "start" non-terminal of sentence).

<table>
<tr><td colspan="5">NP, y
buy(e) ^ past(e) ^ ARG1(e,x) ^ ARG2(e,y)
the(z) ^ ARG1(y); book(y); he(x)</td></tr>
<tr><td colspan="2">NP, y
the(z) ^ ARG1(y)
book(y)</td><td colspan="3">SBAR, e
buy(e) ^ past(e)
^ ARG1(e,x); he(x)</td></tr>
<tr><td colspan="2" rowspan="2"></td><td rowspan="2"></td><td colspan="2">S, e
buy(e) ^ past(e)
^ ARG1(e,x); he(x)</td></tr>
<tr><td>DT, z
the(z)</td><td>NN, y
book (y)</td><td>NP, x
he (x)</td><td>VV, e
buy(e) ^ past(e)</td></tr>
<tr><td>the</td><td>book</td><td>that</td><td>he</td><td>bought</td></tr>
</table>

Figure 2 Text generated by split and emit

This semantic splitting process concord with the Semantics Composition Principle: the semantics of a constituent is the sum of semantics of its daughters (Sag et al., 2003).

Note that sometimes M is splited to an empty semantic and a semantic equal to itself. These kinds of *null splits* permit the generation of functional words not represented in the semantics such as the word "that" in Figure 2.

With the assumption that semantic graph is a DAG, the task of graph splitting is reduced to decide which edge should be broken, the order and syntactic category for each side of the edge. However, with a permissive grammar like CFG (trained from TreeBank), it is very important to choose from possibly huge number of derivation trees for the same tuple $<M, L, p>$. We achieve this goal from two aspects.

First, we reduce number of possible trees generated by exploring several regularities. From the Semantic Inheritance Principle (Sag et al., 2003) we assume that a mother node in derivation tree share the semantic head with one of its daughter. We also assume that at each split step, it is always one of the links around head node that gets broken, and the node on the other side of the broken link becomes head of the other half of semantics. As we know, in HPSG, syntactic head and semantic head are the same except for some head adjunct-daughters (Wilcock & Matsumoto, 1998). This helps justify that when two constituents combined into a new constituent, it is always their heads that are interacting with each other, both syntactically and semantically (Probably need more justification here). This assumption greatly reduces the search scope of links to be broken, and continently help specify the semantic heads.

Second, we can define features and scoring functions to rank the generated sentences. White & Baldridge (2003) and White et al. (2007) score legitimate (HPSG) derivation trees with a variety of factored trigram models over words, POS tags and supertags. Nakanishi et al. (2005) score legitimate (CCG) derivation trees with a log-linear model based on lexicalization, bigram and syntax features. In this work, instead of find the best tree in bottom-up dynamic programming style, we use efficient method in top-down greedy style.

4 Dealing with Infinite Recursion

To propose a top-down approach, we need to deal with the problem of possible infinite recursion during generation (Van Noord, 1994). We

find this problem roots in the inadequacy of CFG labels. For example the sentence "the book was bought by him". Both the constituent spanning "was bought by him" and "bought by him" have the same CFG label VP. This causes a generator to produce potentially infinite number of "was".

We think the best solution is to augment the CFG tag to distinguish such syntactic differences. Probably with tag splitting method of Petrov et al. (2006). In this study, however, we take a simpler solution, by disallowing any split action which generates node that has already been generated (with same semantics and syntactic category).

5 The Split-Emit Algorithm

The Split-Emit Algorithm generally has a symmetric structure to the well known shift-reduce parsing algorithm as the following. It assumes function *split*() that decide which part of the semantics should be realized first and decide new syntactic category for both parts. It also assumes a function *lexicalize*() that translates atomic semantics into surface form, and *decideAction*() to decide which action to take. Same as shift-reduce parsing, it can run in best-first (greedy) mode, or beam search mode. In beam search mode, top K sentences are generated and ranked based on multiplication of probability of all actions (defined in 5.1) taken in the derivative of a sentence. We also compare to a more sophisticated approach, in which ranking is based on log linear models similar to White et al. (2007) and Nakanishi et al. (2005).

function splitEmit(<M,L>)

Input: <semantics M, syntax label L, head p>
Output: sentence w
  S=empty stack []
  while(M!=null or L!=[])
    if M=null
      <M, L, p>=S.pop()
    action=decideAction(<M, L ,p>)
    if action=EMIT
      w+=lexicalize(<M,L ,p >)
      M=null
    else if action=SPLIT
      (<M_0,L_0,p_0>, <M_1,L_1,p_1>)= split(<M,L,p>)
      S.push(<M_1,L_1,p_1>)
      <M,L>=<M_0,L_0,p_0>
return w

Here we give an implementation with PASs semantic representation, and split function based on stacked learning.

5.1 Splitting the Semantics

From assumption in section 3, the edges in consideration are the ones around the head semantic node. For an edge e, we denote the head node v0 and the node on other side of e v1. We extract for each edge the following set of features:

- syntax label of the head L,
- labels in v0, labels in v1,
- argument type (and direction (v0→v1 v.s. v1→v0) of e,
- and possibly other global and local features.

Then train a maximum entropy classifier to predict this edge into the following classes:

- Any combination of D*L_0*L_1,
- NoSplit.

D={forward, backward} decides which side should be realized first. L_0, L_1, are the syntactic categories assigned to each side. NoSplit means this edge should not be broken. During generation, the edge with the highest probability of any D*L_0*L_1 category becomes the split point, and this probability is assigned to this action, which will be used to rank sentences, and restrict the beam width (top k actions sorted by their probabilities go into the beam).

However, this design has the apparent drawback that decisions of edges are made independently. Therefore, we apply stacked learning (Wolpert, 1992; Kou & Cohen, 2007) which permits the use of efficient classifier in place of more computationally-intensive joint inference models. Feature vector of each edge are augmented by the following template (Kou & Cohen, 2007):

- For each category c, sum of the expectation given by classifier of previous level for all the edges in consideration.

Number of stacking can be determined in experiment.

Since the algorithm break one link from the DAG each time (except for the null-splits), about *n-1* splits are needed to break a semantic M of size *n*, and time complexity is about O(n).

5.2 Lexicalization and Decision of Actions

In this proposal, the lexicalization and decision functions are very simple. If the semantic graph contains only one node, then the action is EMIT. Otherwise, the action is SPLIT.

Lexicalization is done using a look up table generated from a training corpus with items like

$$buy(e) \wedge past(e) \rightarrow bought$$

On the left-hand-side is a node observed in the corpus, on the right-hand-side the most probable

word associated with this node in the corpus. If the whole LHS cannot be found in the look up table, then the lexical label ("buy" in the above case) is emitted as a back-off.

6 Evaluation

Training, developing, and testing data are semantic representations derived from Penn TreeBank section 2-21, 22, and 23. There are two possible ways to derive semantic representations (features and predicate-argument relations) and derivation trees for each sentence. First is HPSG derivation trees of Penn TreeBank. Miyao et al. (2004) created more than 250 heuristic rules to transform Penn CFG trees to HPSG trees. Second is CCG derivation trees of Penn TreeBank. White et al. (2007) use about two dozen rules to augment CCG trees with logical forms. Third is LFG derivation trees of Penn TreeBank. Cahill et al. (2004) developed automatic system to annotate f-structure to CFG trees. Since all above tree semantic representations contains predicate-argument structure, it is strait forward to extract semantic DAG from them.

Same as previous works (Table 1), generation accuracy can be measured by BLEU score (Papineni et al., 2001) and exact matches. Since the proposed system does not handle semantic graph that are not DAG, we ignore those during evaluation.

Table 1 Previous works

Work	Semantics Format	Train	Tune	Test	Cvg	BLUE	Exact
White et al. 07	HLDS (CCG)	Sec2~21	Sec22	Sec23	94.5	66.2	14.8
Cahill & Genabith 06	f-structure (LFG)	Sec2~21		Sec23	98.5	66.5	
		Sec2~21		Sec23, len<20	98.7	70.8	
Nakanishi et al. 05	PAS (HPSG)	Sec2~21, len<20	Sec22, len<20	Sec23, len<20	90.8	77.3	
Langkilde-Geary 02 (not corpus based)	feature-value structure	250 million word WSJ text	N/A	Sec23, len<20	82.7	75.7	

References

Bangalore and Rambow. 2000. Exploiting a Probabilistic Hierarchical Model for Generation. COLING.

Aoife Cahill and Josef van Genabith. 2006. Robust PCFG-based generation using automatically acquired LFG approximations. COLING-ACL '06.

Cahill, A., Burke, M., O'Donovan, R., van Genabith, J., and Way, A. (2004). Long-Distance Dependency Resolution in Automatically Acquired Wide-Coverage PCFG-Based LFG Approximations. In Proc.ACL-04, pages 320–327, Barcelona, Spain.

Halvorsen P.-K.: Situation Semantics and Semantic Interpretntlon in Constraint-Based Grammaa~s. CSLI Report No. CSLl-8%lO1, Stanford 1987

Z. Kou and W. W. Cohen. Stacked graphical models for efficient inference in Markov random fields. In Proceedings of the 2007 SIAM Conference on Data Mining (SDM), 2007.

I. Langkilde and K.Knight. 1998. Generation that exploits corpus-based statistical knowledge. In Proc. COLING-ACL'98, pages 704–710.

I. Langkilde. 2000. Forest-based statistical sentence generation. In Proceedings of the NAACL'00.

I. Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In Proceedings of the INLG'02.

Miyao, Yusuke and Jun'ichi Tsujii. Probabilistic disambiguation models for wide-coverage HPSG parsing. In the Proceedings of ACL 2005.

Y. Miyao, T. Ninomiya, and J. Tsujii. 2004. Corpus oriented grammar development for acquiring a Head-Driven Phrase Structure Grammar from the Penn Treebank. In Proceedings of the IJCNLP-04.

Hiroko Nakanishi, Yusuke Miyao, and J. Tsujii. 2005. Probabilistic Models For Disambiguation Of An HPSG-Based Chart Generator. In Proc. IWPT-05.

K. Papineni, S. Roukos, T. Ward, and W. Zhu. 2001. BLEU: a method for automatic evaluation of machine translation. In Proceedings of the ACL'01.

Slav Petrov, Leon Barrett, Romain Thibaux and Dan Klein, "Learning Accurate, Compact, and Interpretable Tree Annotation", in COLING-ACL '06).

Reyle, U.: Compositional Semaaltics for LFG. In: Reyle, U., Bohrcr, C. (eds.): Nabaral Language Parding and Linguistle Theories. Dordrecht 1988

Ivan A. Sag, Thomas Wasow, and Emily M. Bender. Syntactic Theory: A Formal Introduction, 2nd Edition, 608 pages, 1999, 2003

Mike White and Jason Baldridge (2003) Adapting Chart Realization to CCG. In Proc. EWNLG'03.

Michael White, Rajakrishnan Rajkumar and Scott Martin (2007) Towards Broad Coverage Surface Realization with CCG In Proc. of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+MT).

Graham Wilcock , Yuji Matsumoto, Head-driven generation with HPSG, Proc. the 17th international conference on Computational linguistics, p.1393-1397, August 10-14, 1998, Montreal, Quebec, Canada

D. H. Wolpert, Stacked generalization, Neural Networks, vol. 5, pp241-259, 1992.