SCFG Latent Annotation for Machine Translation

Tagyoung Chung, Licheng Fang, Daniel Gildea

Department of Computer Science
University of Rochester
Rochester, NY 14627

Abstract

We discuss learning latent annotations for synchronous context-free grammars (SCFG) for the purpose of improving machine translation. We show that learning annotations for nonterminals results in not only more accurate translation, but also faster SCFG decoding.

1. Introduction

Synchronous context-free grammar (SCFG) is an expressive yet computationally expensive model for syntax-based machine translation. Practical variations of SCFG-based models usually extract rules from data under a set of constraints. The hierarchical phrase-based model [1] is a restricted form of synchronous context-free grammar. The grammar has only one nonterminal, and extraction is based on word alignments of sentence pairs. GHKM [2] represents a more general form of synchronous context-free grammar. The grammar is extracted under the constraints of both word alignments of sentence pairs and target-side parse trees. In this paper, we discuss adding annotations to nonterminals in synchronous context-free grammars extracted using the GHKM method [2] to improve machine translation performance.

Nonterminal refinement is a well-studied problem in English parsing, and there is a long, successful history of refining English nonterminals to discover distributional differences. The fact that the granularity of nonterminals in the Penn English Treebank [3] is too coarse for automatic parsing has been widely discussed, and addressing the issue by refining English nonterminals has been shown to improve monolingual parsing performance.

There are many ways to refine the set of nonterminals in a Treebank. Many of the previous attempts essentially try to weaken the strong independence assumptions of probabilistic context-free grammars by annotating nonterminals in different ways. For example, Johnson [4] takes the approach of simply annotating each node with its parent's label. The effect of this is to refine the distribution of each nonterminal over sequences of children according to its position in the tree; for example, a VP beneath an SBAR node will have a different distribution over its children from a VP beneath an S node. This simple technique alone produces a large improvement in English Treebank parsing. Klein and Manning [5] expanded this idea with a series of experiments wherein they manually refined nonterminals to different degrees, which resulted in parsing accuracy rivaling that of bilexicalized parsing models of the time.

More recently, Petrov et al. [6] refined techniques originally proposed by Matsuzaki et al. [7] and Prescher [8] for automatically learning latent annotations, resulting in state-of-the-art parsing performance with cubic-time parsing algorithms. The main idea of these approaches is to learn latent annotation of nonterminals by splitting nonterminals and learning distributional differences of split nonterminals using the EM algorithm [9]. Improvements presented by Petrov et al. [6] include merging back frivolous splits and smoothing the parameters.

There have been efforts to expand the idea of nonterminal annotation to synchronous context-free grammars in order to improve machine translation. Huang and Knight [10] presented various linguistically motivated annotations to the Penn English Treebank nonterminals — some of which were motivated by ideas by Klein and Manning [5]. Their results show that annotating nonterminals in target-side parse trees and then extracting an SCFG grammar from the trees improves machine translation performance. Huang et al. [11] applied the idea by Petrov et al. [6] to learn a latent distribution for the single Hiero nonterminal and used distributional similarities as a soft constraint for rewriting rules. Wang et al. [12] used the same split-merge approach to learn latent annotations of the target-side parse tree before extracting an SCFG grammar. This approach results in improvement in machine translation performance.

However, it is possible that the nonterminal refinements that help parsing may not be optimal for the purposes of machine translation. In particular, since the internal nodes of GHKM translation rules are irrelevant to the decoding process, there is no need to for the translation model to predict them accurately. The decoder must, however, choose which translation rules to combine, and refined nonterminals can help tell us which rule combinations to prefer.

In this paper, we discuss how we can learn latent annotations that are targeted for use in translation. Our work is similar in spirit to the work of Wang et al. [12], but with the important difference that, rather than learning latent annotations on target-side trees before grammar extraction, we learn latent annotations on the extracted SCFG grammar. We show that our approach brings statistically significant improvement

1. Binarize target-side parse trees
2. Mark frontier nodes
3. Binarize rule trees
4. Learn latent annotations for rule trees
5. Annotate rule trees with the Viterbi split
6. Recover SCFG grammar from rule trees

Table 1: SCFG rule extraction

1. Binarize target-side parse trees
2. Learn latent annotations for parse trees
3. Annotate parse trees with the Viterbi split
4. Extract SCFG grammar from parse tree

Table 2: Baseline

over the method presented by Wang et al. [12]. Annotating SCFGs somewhat enlarges the grammar, however, contrary what one might expect, we show that the annotated grammar greatly speeds up decoding. Our contribution can be summarized as follows. First, we present a novel way to annotate SCFG grammar which improves translation accuracy over the baseline approach. Second, we provide detailed analyses of why decoding with the annotated grammar not only increases translation accuracy but also leads to faster decoding.

2. Learning latent annotations for SCFG

In this section, we discuss several issues arising from learning latent annotations for SCFG. We also discuss the exact process of labeling nonterminals with latent annotations. Figure 1 describes the entire process of extracting SCFG rules, learning latent annotations, and generating the final grammar. Table 1 summarizes the process. We will discuss each step in more detail and highlight the difference between our approach and the baseline in the rest of the section. Our baseline is the work by Wang et al. [12], which is summarized in Table 2.

2.1. Parse tree binarization

The issue of binarization comes up multiple times when learning latent annotations for SCFGs. As shown by Wang et al. [13], binarization of target-side parse trees before extracting GHKM rules improves translation quality by breaking larger rules into smaller rules with more generalization power. Target-side tree binarization generates virtual tree nodes that may be frontiers, which relax the constituent boundary constraints imposed by parse trees during rule extraction. In both the baseline and our grammar refining approach, we left-binarize English parse trees. However, for comparison, we also do experiments where we use the original English parse trees without binarization. Parse tree binarization is the first step in Figure 1.

2.2. Marking frontier nodes

To learn latent annotation for SCFG, we need to extract GHKM rules from each sentence and reconstruct them as a rule tree for each sentence. In practice, using parsed English sentences, Chinese terminals and their alignments, we marked frontier nodes as defined in [2] and removed non-frontier nodes. We slightly modify the definition of frontier nodes to avoid extracting unary rules: a frontier node cannot have a parent who covers the same source-side span. The details of this change can be found in [14]. Steps 2 and 3 of Figure 1 show marking frontier nodes and removing non-frontier nodes. There is one exception to the process of removing non-frontier nodes. To aid EM, we do not remove preterminals even when they are not frontier nodes. In Figure 1, VBD and JJ are removed because they are not frontier nodes. However, in actual grammar training, they were not removed.

To differentiate from the original English parse tree, we henceforth refer to a tree with non-frontier nodes removed as a *rule tree* (e.g., the tree after step 3 in Figure 1) and a normal English parse tree as a parse tree.

2.3. Rule tree binarization

Efficient split-merge EM training requires binarized trees. During training, splitting each nonterminal into two introduces 2^{n+1} rules for each rule with n right-hand-side nonterminals. We run the grammar trainer on binarized rule trees in all our experiments to keep training practical.

One difference of our approach from that of the baseline approach also lies in this rule binarization. Figure 2 illustrates this difference. The leftmost tree in Figure 2 is a target-side parse tree fragment with frontier nodes marked. Then, we remove non-frontier nodes to reveal the SCFG rule tree. As mentioned, we need to binarize the rule tree for split-merge grammar training. Each rule in the rule tree is left-binarized which introduces virtual nonterminals. By altering structures inside SCFG rules, we enable the grammar trainer to maximize the likelihood of generating the data, which is achieved by not using the Treebank grammar, but using the SCFG rules which are actually used in translation. In case of the example in Figure 1, all SCFG rules happen to be binary; therefore the rule tree is kept as it is.

2.4. Latent annotation

We follow the method presented in [6] for split-merge-smooth cycle on SCFG grammar with little change. The general *cycle* of one iteration of training follows:

1. Split a nonterminal into two nonterminals and use EM to learn probabilities for the new rules introduced.

2. Merge split nonterminals back if the loss in likelihood for merging is small.

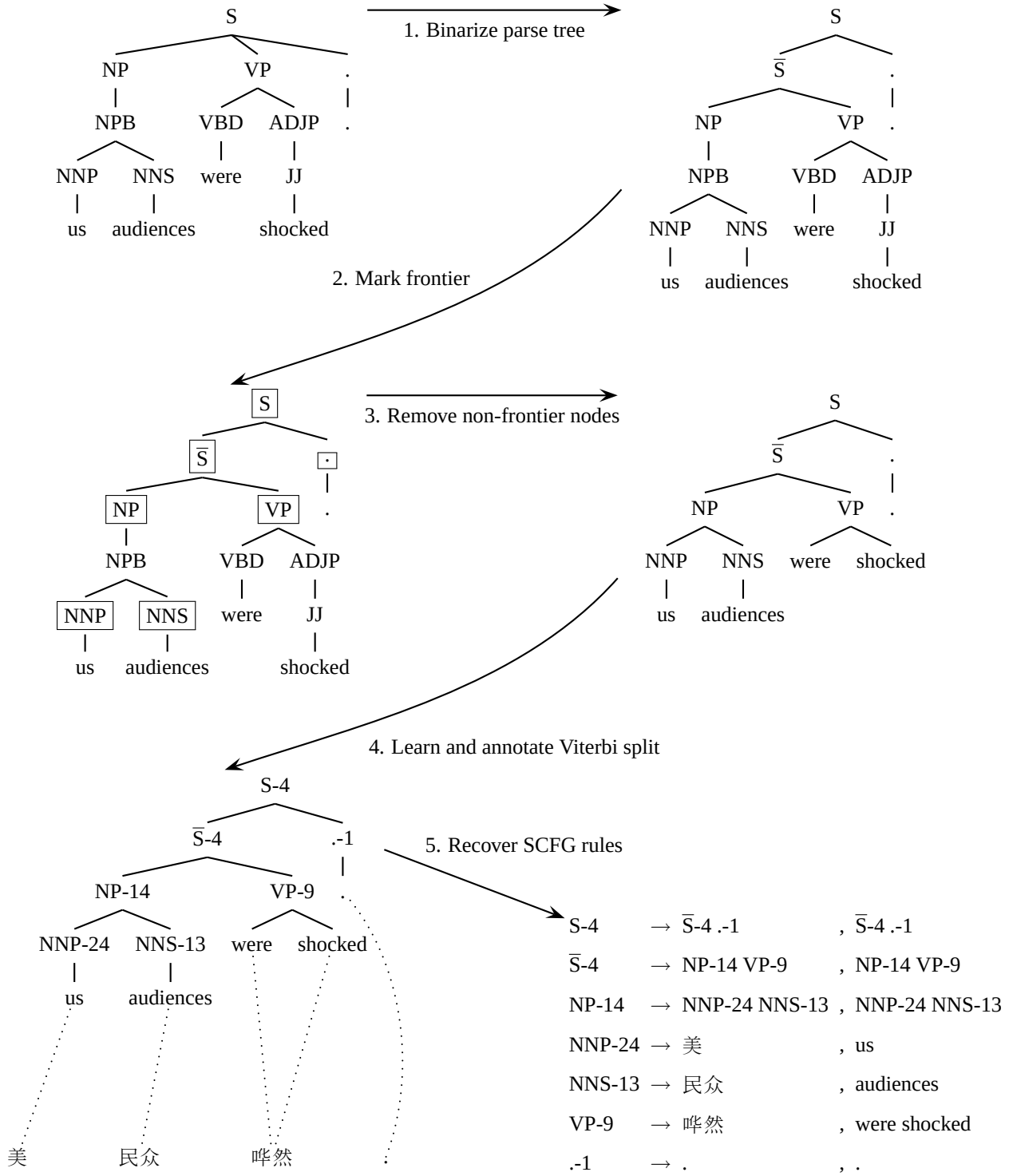

Figure 1: Illustration of Viterbi-split annotated SCFG extraction process. Boxed nodes are frontier nodes. Between step three and four, there is *binarize rule tree* step. However, since the rule tree is already binary, the step has been omitted.

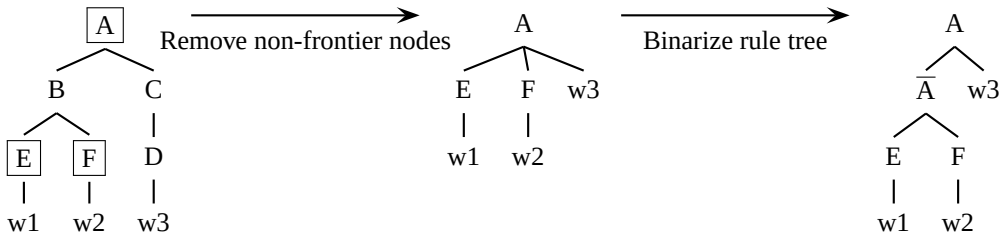

Figure 2: Illustration of rule tree binarization

3. Use additive smoothing so that distributions for split nonterminals may not diverge too much.

Learning latent annotation on reconstructed rule trees rather than target-side English parse trees is the key difference between our work and the baseline. Because we binarize the reconstructed rule trees before learning latent annotation, depending on the structure of a rule, compared to the baseline, we may end up learning latent annotation for very different structure and nonterminals.

After learning the latent annotation for SCFG rules, we annotate rule trees with Viterbi split annotation, rather than using entire split SCFG grammar. This is reasonable because learned split annotations are usually very consistent. This also vastly reduces the number of SCFG rules that the decoder needs to handle. Step 4 in Figure 1 illustrates Viterbi split annotation.

After the Viterbi split annotation, we can finally recover SCFG rules from the rule tree, which is illustrated as Step 5 in Figure 1.

2.5. SCFG terminals

So far, we have only discussed using only English terminals for split-merge training. However, since we are working with SCFG rules, we may choose to use English, Chinese, or both English and Chinese for terminals of the rule trees that we use for learning latent annotation. Although what we have is essentially English syntactic structure, we can replace English terminals with Chinese using word alignment, or we can try to fuse two aligned terminals. Figure 3 shows the three aforementioned configurations. We experiment with all three configurations.

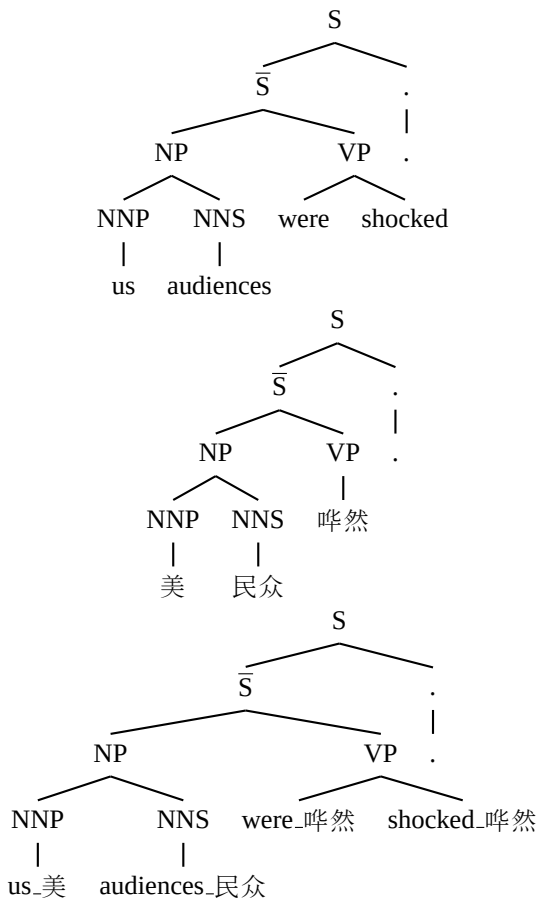

Figure 3: Rule trees with three different terminal configurations

3. Experiments

3.1. Setup

We used a Chinese-English parallel corpus with the English side parsed for our experiments. The corpus consists of 206K sentence pairs, which is 5M words on the English side.[1] Min-

[1]We randomly sampled our data from various different sources (LDC2006E86, LDC2006E93, LDC2002E18, LDC2002L27, LDC2003E07, LDC2003E14, LDC2004T08, LDC2005T06, LDC2005T10, LDC2005T34, LDC2006E26, LDC2005E83, LDC2006E34, LDC2006E85, LDC2006E92, LDC2006E24, LDC2006E92, LDC2006E24)

imum error rate training (MERT) [15] was used to tune parameters. We used a 392-sentence development set with four references for parameter tuning, and a 428-sentence test set with four references for testing. The development set and the test set only had sentences with less than 30 words for decoding speed. All English-side data used in the experiments were lower cased. Our in-house SCFG decoder was used for experiments with a trigram language model. The same settings of cube-pruning [16] were used for all experiments. BLEU [17] was calculated for evaluation. We also performed significance tests for our results using paired bootstrap resampling [18]. The difference is considered statistically significant if $p < 0.05$ based on 1000 iterations of paired bootstrap resampling.

We left-binarized the English parse trees before running split-merge-smoothing grammar training cycles and extracting GHKM grammars for the baseline, where we tried to replicate the work in [12]. We mark the frontier nodes on the same binarized parse trees to segment the parse trees according the source side of SCFG rules. We then convert each parse tree segment to be left-binarized for grammar training. We also extracted an SCFG from unbinarized parse trees for a comparison. All split-merge-smoothing cycles used the same settings: 2-way split and 50% merge-back. We have tried using only English terminals, only Chinese terminals, and fused terminals as SCFG terminals for the grammar training. Synchronous binarization [19] was applied to all resulting grammars.

3.2. Results

3.2.1. Learning splits on SCFG vs. tree

Table 3 summarizes BLEU scores of our test set from each grammar. BLEU scores on the development set, which was used to tune parameters, exhibit the same trend, which is discussed below. The column labeled *tree*, which is our baseline, indicates grammars extracted from the parse trees that are annotated with Viterbi split annotations learned from the parse tree. As previously mentioned, the latent annotations for these grammars are learned on binarized parse trees. The column labeled *SCFG* and *SCFG-un* indicates grammars extracted from rule trees that are annotated with Viterbi split annotations learned from the rule trees. The latent annotation for these grammars are learned on binarized SCFG rules with only English terminals. The difference between *SCFG* and *SCFG-un* is whether original parse trees were binarized before marking frontier nodes.

The best result for our baseline came at the second iteration of grammar training, and the best result for grammars extracted from SCFG rule trees came at the fifth iteration of grammar training, regardless of whether original parse tree was binarized at the time of initial SCFG extraction (marking frontier nodes). The difference between our best result and the best result of the baseline is statistically significant. Incidentally, the second cycle grammar that is extracted from the Viterbi split annotated parse tree is the most comparable to the best result presented in [12] when we compare the nonterminal set size. (Using one iteration of split-merge-smoothing cycle 4-way split and 90% merge-back, the nonterminal set size in their work is 178, while, in our work, it is 237 at the second cycle.)

cycle	tree	SCFG	SCFG-un
0	20.64	20.64	20.05
1	20.57	20.49	20.69
2	**21.94**	21.27	20.73
3	21.22	21.45	20.86
4	21.33	21.87	21.90
5	21.38	**22.98**	**22.52**
6	21.75	22.32	22.02

Table 3: BLEU score results for Viterbi split (learned from the original trees) annotated SCFG grammars that are extracted from the Viterbi annotated original binarized parse tree (baseline), Viterbi split (learned from the rule trees) annotated SCFG that are extracted from binarized parse tree, and Viterbi split (learned from rule trees) annotated SCFG that are originally extracted from regular parse tree (not binarized) for each grammar training cycle

3.2.2. Binarization

Binarizing the parse trees before extracting rules did bring some improvement. On our test set, for non-split grammar, the improvement is from BLEU of 20.05 to 20.64. The difference is statistically significant. The second and third column of Table 3 show the difference between SCFG grammars extracted from binarized parse trees and parse trees that are not binarized. The second column represents results from grammars that are extracted from binarized parse trees, which is consistently better than the results (the third column) from using parse trees that are not binarized for all grammar training cycle. However, at the fifth cycle, when the both grammars peak in terms of BLEU scores on the test set, the difference is only significant at $p < 0.084$. Overall, the result confirms previous works by others [12, 13] and shows that the benefits of binarization extend to our way of learning latent annotations.

3.2.3. Using different terminals

Having only English terminals yielded the best result. Table 4 summarizes the three configurations for terminals. On our test set, having only English terminals during the grammar training yielded BLEU of 22.98 for our test set. Having Chinese terminals yielded 21.12, and having fused terminals yielded 21.26.

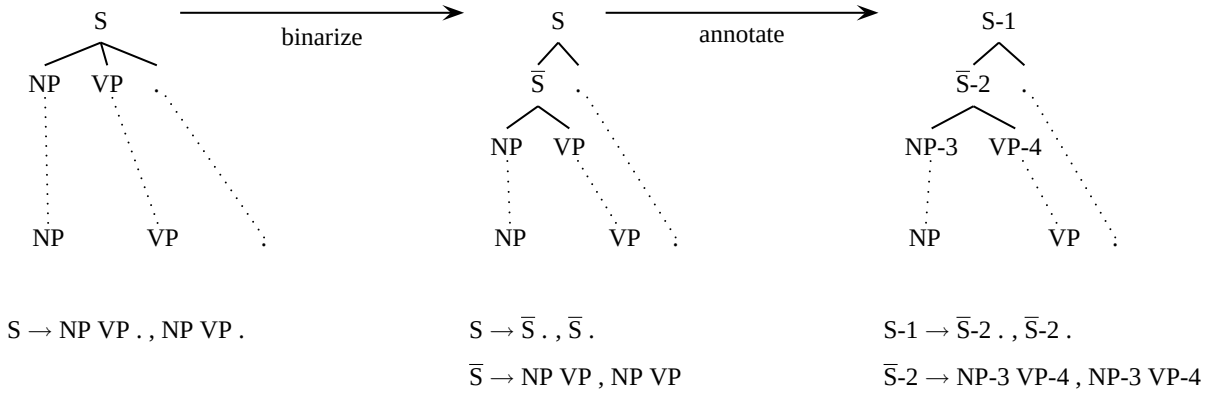

Figure 4: Illustration of how binarization and latent annotation alters extracted SCFG rules

English	Chinese	Fused
22.98	21.12	21.26

Table 4: SCFG with only English terminals, only Chinese terminals, and fused terminals. All grammars are of fifth cycle.

4. Analyses

4.1. What is happening?

There are two major reasons for the success of the nonterminal relabeling strategy presented in this paper. They are binarization and consistent latent annotation.

Figure 4 summarizes our approach to relabeling nonterminals. First, binarization of target-side parse trees enables us to learn a more general grammar. For example, in the example given in Figure 4, we can extract two rules instead of one, which increases the chances of extracting rules that are useful at the test time. The binarization gives us a modest boost of BLEU score. The score for our test set increases from 20.05 to 20.64, which is consistent with works by others [12, 13].

Binarization effectively Markovizes the rules we extract, but, as discussed in Section 3.2, the biggest improvement comes when rules are annotated with learned latent annotations. The annotation restricts rules that can be applied given a context, therefore, we are implicitly learning bigram rule probabilities. The annotation guides the decoder to correctly glue together Markovized rules. This brings us two positive effects: accuracy and speed, which are discussed in the next two sections.

4.2. Why is it more accurate?

Figure 5 shows an example of why the annotation brings about more accurate translation. It shows how a lexical rule ($\overline{\text{PP}}$-27 → 为了, in order to) can be applied at decoding time. The figure includes the three rules with highest probability from the fifth-cycle grammar that can be applied to this rule (rules that have the left-hand side ($\overline{\text{PP}}$-27) of this rule as a child). All three rules have $\overline{\text{VP}}$-2 as a second child. The figure also shows the top three rules that have $\overline{\text{VP}}$-2 on the left-hand side. Two of them have VB as the first child, which stands for base-form verb. There is one exception (the rule that has $\overline{\text{VP}}$-20 as the left child). However, even for this rule, the top three rules that have $\overline{\text{VP}}$-20 on the left-hand side all have ADVP as the first child and VB as a second child (not shown in the figure). This chain of rules restricts the word following *in order to* to be a verb in its base-form, which leads to grammatically correct translation.

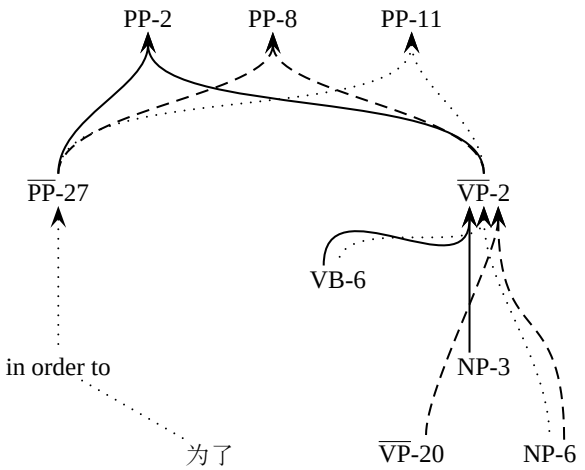

Figure 5: Illustration of how annotation helps the decoder to find more accurate translation

4.3. Why is it faster?

Table 5 shows the total number of rules and total number of nonterminals for each grammar. As one would expect, as the split-merge-smoothing cycle progresses, we increasingly get more nonterminals and grammars become larger. However, the size of grammar does not explode as one might expect. Although the sixth cycle grammar has more than ten times as many nonterminals as the non-split grammar, the grammar is

cycle	rules	nonterminals
0	1.43×10^6	103
1	1.55×10^6	158
2	1.62×10^6	239
3	1.67×10^6	359
4	1.74×10^6	542
5	1.81×10^6	817
6	1.89×10^6	1234

Table 5: number of rules and nonterminals for grammars from each cycle

cycle	cubes	edges	WPM
0	8.45×10^7	3.75×10^6	15.66
1	1.07×10^8	3.64×10^6	12.75
2	9.21×10^7	3.66×10^6	13.59
3	7.83×10^7	3.64×10^6	15.17
4	5.50×10^7	3.62×10^6	19.11
5	3.78×10^7	3.56×10^6	25.83
6	2.25×10^7	3.35×10^6	39.44

Table 6: Number of cubes being built, number of hyperedges proposed, and decoding time (words per minute) for each grammar

only about 30 percent larger because annotations stay very consistent.

One might also expect that, since the grammar becomes larger as the split-merge-smoothing cycle progresses, the speed of decoding with larger grammar would slower as well. However, we have the quite the opposite result. Table 6 summarizes decoding times of the test set for each grammar. The grammar is re-decoded with the same set of parameters to eliminate the influence of other factors (such as the feature for the total number of rules used). The trend is largely the same even when the grammar is decoded with different sets of parameters. It is clear from the table that decoding grammars from later stages of split-merge-smoothing cycles is faster, not slower.

This phenomenon can be explained by examining the number of edges proposed by the decoder in Table 6. The decoder uses cube pruning and resulting hypergraphs have two types of edges: -LM edges, which are the outcome of parsing with the grammar without language model integration and +LM edges, which represent derivations after the intersection of -LM edges with the language model. The cube pruning builds one cube for each -LM edge. Therefore, the number of cubes that are built is effectively the number of -LM edges. Table 6 shows both the number of cubes that are built and the number of +LM edges proposed by the decoder for the entire test set. We can observe that because of pruning, the number of +LM edges stays roughly the same even when decoding with different grammars. We also observe that the decoding time is highly correlated with the number of -LM edges, where no pruning is applied. We can conclude from these observations that the number of cubes being built is the dominating factor in the decoding time. The reason that fewer -LM edges are being built when decoding with the annotated grammar is that the Viterbi split annotations that are learned from the grammar training specialize the rules in the annotated grammar, which results in fewer rules being matched when decoding with the annotated grammar, thus leading to faster decoding.

For example, for the lexical rule in Figure 5, in the non-split grammar, there are 8030 rules that have $\overline{\text{PP}}$ as a child but there are only 860 rules that have $\overline{\text{PP}}$-27 as a child in the fifth-cycle grammar.

5. Conclusion and future work

In this paper, we have discussed how we can learn latent annotations for SCFGs to improve machine translation results. The method we presented here has two-fold benefits. It makes decoding more accurate in terms of BLEU score and it speeds up decoding by a considerable margin. We showed that learning latent annotations on SCFGs directly yields better translation results than learning latent annotations for target-side trees.

There are a number of ways to improve our approach. First, we can employ more sophisticated binarization rather than doing simple left binarization. For example, we have performed synchronous binarization only after extracting grammars, but we could try to binarize the tree in the same manner before extracting grammars. Second, we can investigate further into how we can effectively incorporate source-side terminals. The current method of simply fusing source and target terminals may be too crude to yield any benefits. We need to further analyze the result and understand why the current approach yields no benefits and design a method of including source-side terminals in grammar training in a way that would improve translation result. Finally, we may benefit from more sophisticated grammar training. The current method does not provide a good indication of whether the grammar is over-fitting or under-fitting until we decode with the grammar and observe fluctuation in the translation accuracy. We can use a development set to control the number of training cycles. We may also introduce priors to incorporate our beliefs about what good split nonterminal distributions should be like.

6. References

[1] D. Chiang, "A hierarchical phrase-based model for statistical machine translation," in *Proceedings of the 43rd Annual Conference of the Association for Computational Linguistics (ACL-05)*, Ann Arbor, MI, 2005, pp. 263–270. [Online]. Available: http://www.aclweb.org/anthology/P/P05/P05-1033

[2] M. Galley, M. Hopkins, K. Knight, and D. Marcu, "What's in a translation rule?" in *Proceedings of the*

2004 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-04), Boston, 2004, pp. 273–280.

[3] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, June 1993.

[4] M. Johnson, "PCFG models of linguistic tree representations," *Computational Linguistics*, vol. 24, no. 4, pp. 613–632, 1998.

[5] D. Klein and C. D. Manning, "Accurate unlexicalized parsing," in *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo, Japan: Association for Computational Linguistics, July 2003, pp. 423–430. [Online]. Available: http://www.aclweb.org/anthology/P03-1054

[6] S. Petrov, L. Barrett, R. Thibaux, and D. Klein, "Learning accurate, compact, and interpretable tree annotation," in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. Sydney, Australia: Association for Computational Linguistics, July 2006, pp. 433–440. [Online]. Available: http://www.aclweb.org/anthology/P/P06/P06-1055

[7] T. Matsuzaki, Y. Miyao, and J. Tsujii, "Probabilistic CFG with latent annotations," in *Proceedings of the 43rd Annual Conference of the Association for Computational Linguistics (ACL-05)*, 2005, pp. 75–82.

[8] D. Prescher, "Inducing head-driven PCFGs with latent heads: Refining a tree-bank grammar for parsing," *Machine Learning: ECML 2005*, pp. 292–304, 2005.

[9] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the *EM* algorithm," *Journal of the Royal Statistical Society*, vol. 39, no. 1, pp. 1–21, 1977.

[10] B. Huang and K. Knight, "Relabeling syntax trees to improve syntax-based machine translation quality," in *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. New York City, USA: Association for Computational Linguistics, June 2006, pp. 240–247. [Online]. Available: http://www.aclweb.org/anthology/N/N06/N06-1031

[11] Z. Huang, M. Cmejrek, and B. Zhou, "Soft syntactic constraints for hierarchical phrase-based translation using latent syntactic distributions," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA: Association for Computational Linguistics, October 2010, pp. 138–147. [Online]. Available: http://www.aclweb.org/anthology/D10-1014

[12] W. Wang, J. May, K. Knight, and D. Marcu, "Restructuring, re-labeling, and re-aligning for syntax-based machine translation," *Computational Linguistics*, vol. 36, pp. 247–277, June 2010.

[13] W. Wang, K. Knight, and D. Marcu, "Binarizing syntax trees to improve syntax-based machine translation accuracy," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007, pp. 746–754. [Online]. Available: http://www.aclweb.org/anthology/D/D07/D07-1078

[14] T. Chung, L. Fang, and D. Gildea, "Issues concerning decoding with synchronous context-free grammar," in *Proceedings of the ACL 2011 Conference Short Papers*. Portland, Oregon: Association for Computational Linguistics, 2011. [Online]. Available: http://www.cs.rochester.edu/~gildea/pubs/chung-fang-gildea-acl11.pdf

[15] F. J. Och, "Minimum error rate training for statistical machine translation," in *Proceedings of the 41th Annual Conference of the Association for Computational Linguistics (ACL-03)*, 2003. [Online]. Available: http://www.isi.edu/~och/acl03.pdf

[16] D. Chiang, "Hierarchical phrase-based translation," *Computational Linguistics*, vol. 33, no. 2, pp. 201–228, 2007.

[17] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL-02)*, 2002. [Online]. Available: http://acl.ldc.upenn.edu/P/P02/P02-1040.pdf

[18] P. Koehn, "Statistical significance tests for machine translation evaluation," in *2004 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Barcelona, Spain, July 2004, pp. 388–395.

[19] L. Huang, H. Zhang, D. Gildea, and K. Knight, "Binarization of synchronous context-free grammars," *Computational Linguistics*, vol. 35, no. 4, pp. 559–595, 2009. [Online]. Available: http://www.mitpressjournals.org/doi/pdf/10.1162/coli.2009.35.4.35406