A Discriminative Reordering Parser for IWSLT 2013

Hwidong Na and Jong-Hyeok Lee

Department of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH), Republic of Korea
{leona, jhlee}@postech.ac.kr

Abstract

We participated in the IWSLT 2013 Evaluation Campaign for the MT track for two official directions: German↔English. Our system consisted of a reordering module and a statistical machine translation (SMT) module under a pre-ordering SMT framework. We trained the reordering module using three scalable methods in order to utilize training instances as many as possible. The translation quality of our primary submissions were comparable to that of a hierarchical phrase-based SMT, which usually requires a longer time to decode.

1. Introduction

Word reordering is one of the most difficult problems in machine translation. Formally, word reordering refers to arrange the source words into a target-like order, i.e. finding a permutation of the source words. Because searching for all possible permutations is an NP-complete problem, statistical machine translation (SMT) systems have restricted their search space for efficiency. For example, the simple distortion model in phrase-based SMT (PBSMT) prohibit a long distance jump beyond a window size during translation. Therefore, PBSMT suffers from the lack of ability for word reordering at a long distance.

Pre-ordering is one of the most prevailing approaches to overcome this limitation of PBSMT. It is a pre-processing method that reorders the source sentence in advance to the later translation using PBSMT. We categorize previous works into three categories. First, pre-ordering using local information reorders either a (flat) word or chunk sequence [1, 2, 3, 4]. Second, pre-ordering using syntactic information manipulates a syntactic tree so that yield a reordered sentence [5, 6, 7, 8, 9]. Third, pre-ordering using an ad-hoc structure for word reordering induces a discriminative parser trained from a parallel corpus, and apply the parser to obtain a reordered source sentence [10, 11].

Both the second and third approaches work with hierarchical structures of the source sentence. While the second approach requires a syntactic parser which might not be available for resource-poor languages, the third one requires only a small manual word aligned corpus in addition to a large parallel corpus. Hereinafter, therefore, we focus on the third approach. Because of the efficiency, the hierarchical structures of the third approach restrict word reordering within a continuous sequence under a sub-structure, i.e., the hierarchical structures obey Inversion Transduction Grammar (ITG) [12] constraints.

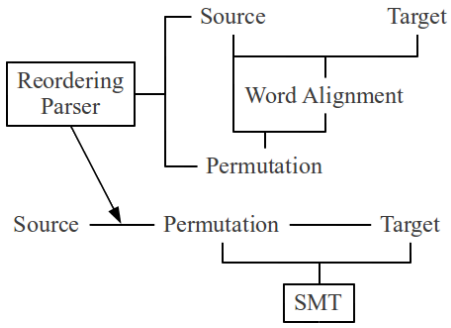

Figure 1: The overall architecture of our system, consisting of a reordering module (reordering parser) and a SMT module

We participated in the IWSLT 2013 Evaluation Campaign for the MT track, and submitted runs for two official directions: German↔English. As German and English have different word orders, we applied a pre-ordering method to resolve the difference requires word reordering.

2. System description

Our system consists of two modules: a reordering module and a SMT module. The reordering module rearranges the words in the source sentence, and the SMT module translates the reordered sentence into the target sentence. The overall architecture of our system is shown in Figure 1. Because we utilized an off-the-shelf SMT system [13] as the SMT module, we focus on the reordering module here.

2.1. Discriminative reordering parser

We briefly summarize the discriminative reordering parser in this section. The most relevant work of this paper was proposed to induce a tree for word reordering produced by a discriminative parser [11]. The goal of their method is to find the best permutation $\hat{\pi}$ for a given source sentence F, according to the following discriminative model.

Algorithm 1: Online learning for a training instance

```
1  procedure UpdateWeight(F, A, w)
2      D ← Parse(F, w)
3      Ḋ ← arg max_{D∈D} Score(D|F, w) + L(D|F, A)
4      D̂ ← arg min_{D∈D} L(D|F, A) − αScore(D|F, w)
5      if L(D̂|F, A) ≠ L(Ḋ|F, A) then
6          w ← β(w + γ(φ(D̂, F) − φ(Ḋ, F)))
7      end
```

$$\begin{aligned} \hat{\pi} &= \arg\max_{\pi} Score(\pi|F) \\ Score(\pi|F) &= Score(D|F, \mathbf{w}) \\ &= \sum_i w_i \phi_i(D, F) \end{aligned} \quad (1)$$

where D is a reordering tree for word reordering which yields π, and w_i and ϕ_i are the ith feature weight and function, respectively.

To learn the weight vector $\mathbf{w}$, they used the loss-driven large-margin training [14] by finding $\dot{D}$ with the highest model score (Eq. 1) and $\hat{D}$ with the smallest loss. A loss function $L(D|F, A)$ is defined by word alignment A, where [11] suggested two kinds of loss functions. Finally, the weight is updated using the difference between the model parse $\dot{D}$ and the oracle parse $\hat{D}$. It is computationally intractable to search over all possible permutations for π. Hence, $\dot{D}$ and $\hat{D}$ are selected among K-best parses encoded in a hyper graph $\mathcal{D}$. To break the tie, $Score(D|F, \mathbf{w})$ and $L(D|F, A)$ are mutually augmented when selecting $\dot{D}$ and $\hat{D}$. The online learning for a training instance $\langle F, A \rangle$ with the current weight vector $\mathbf{w}$ is shown in Algorithm 1 (taken from [11]).

2.2. Scalable training method

We illustrate three methods to scale up the online learning method which iterates several epochs over the training instances. First, we adopted a faster search algorithm known as Cube Growing, and integrated it into a parallel CYK parsing method. Second, the feature generation process run in parallel because it is a major bottleneck of parsing efficiency. Third, the features generated at the first iteration are stored on disk and used in the remaining epochs. In a consequence, our proposed methods enable us to utilize tens of thousands training instances in our experiments.

2.2.1. Cube Growing in parallel CYK parsing

Cube Growing is a dynamic programming algorithm for searching over a hyper graph, proposed by [15]. It produces the kth-best parse on-the-fly, and thus does not enumerate unnecessary hypotheses during the search process. More specifically, two data structures manage the hypothe-

Algorithm 2: Cube Growing in parallel CYK parsing

```
1  procedure Parse
   input : A sentence w_1 ... w_N
   output: A hyper graph with K-best parses
2    for L ∈ [1, N] do
3      for l ∈ [0, N − L]        // in parallel
4      do
5        r ← l + L
6        ModifiedCubeGrowing(l, r)
7      end
8      wait for terminating the cell-level parallelization
9    end
10   root ← The root cell
11   for k ∈ [1, K] do
12     LazyKthBest(root.q, k)
13   end
14
15 procedure ModifiedCubeGrowing
   input : A cell covering [l, r]
   output: A priority queue q with candidates
16   for m ∈ (l, r] do
17     left ← cell [l,m]
18     right ← cell [m,r]
19     L ← peek(left.q)
20     R ← peek(right.q)
21     push(q, Hyp(L, R))        // straight
22     push(q, Hyp(R, L))        // inverted
23   end
24
25 procedure LazyKthBest
   input : A priority queue q and the demanded k
   output: The kth hypothesis in b
26   b ← the list of best hypothesis
27   while size(b) < k + 1 and size(q) > 0 do
28     best ← pop(q)
29     LazyNext(q, best)
30     push(b, best)
31   end
32
33 procedure LazyNext
   input : A priority queue q and the hypothesis best
   output: An extended priority queue q'
     /* best.L and best.R are the left
        and right children of best,
        respectively                    */
34   L ← LazyKthBest(left, rank(best.L)+1)
35   if L exists then
36     push(q, Hyp(L, best.R))   // straight
37     push(q, Hyp(best.R, L))   // inverted
38   end
39   R ← LazyKthBest(right, rank(best.R)+1)
40   if R exists then
41     push(q, Hyp(best.L, R))   // straight
42     push(q, Hyp(R, best.L))   // inverted
43   end
```

ses: a list of best hypotheses and a priority queue of candidates for the next best hypothesis. If the kth-best parse is already produced, it is the kth hypothesis in the best list. Otherwise, Cube Growing enumerates hypotheses by taking the best candidate from the priority queue until the kth hypothesis can be found. Whenever the best candidate is taken from the priority queue, successors of the candidate are pushed on the priority queue, if possible. To obtain the successors, Cube Growing is recursively performed.

[16] proposed that the original CYK parsing algorithm can be parallelized in three levels: sentence-level, cell-level, and grammar-level. Although they reported the grammar-level parallelization achieved the fastest result using thousands of GPUs, we adopted the cell-level parallelization. It is possible to parallelize the original CYK parsing at cell-level because the hypotheses in different chart cells covering same number of words in the sentence do not affect each other. Unfortunately, this property does not hold anymore in Cube Growing because k-best hypotheses are enumerated on demand. Therefore, a race condition arises if we directly apply Cube Growing in the cell-level parallelization.

We modified Cube Growing to fit in the cell-level parallelization. To avoid the race condition, the modified Cube Growing directly accesses to the priority queue for the first best hypothesis. It is postponed to move the first best hypothesis to the best list until the second best hypothesis is requested. From the second best parses, the modified Cube Growing does not run in parallel, which is identical to the original one. Algorithm 2 shows the entire procedures for the cell-level parallelization with the modified Cube Growing.

2.2.2. *Parallel feature generation*

The feature function ϕ in the discriminative model (Eq. 1) is further decomposed into the edge level in a reordering tree.

$$\phi_i(D, F) = \sum_{d \in D} \phi_i(d, F) \quad (2)$$

$$Score(D|F, \mathbf{w}) = \sum_{d \in D} \sum_i w_i \phi_i(d, F) \quad (3)$$

where d is a hyper edge in the hyper graph. Because most of feature functions $\phi_i(d, F)$ involve string operations, the feature generation becomes a major bottleneck of parsing efficiency. In a pilot study of our experiments, the feature generation is the most time-consuming process, which takes over 80% of the total parsing time.

Our proposed method parallelizes the feature generation in advance to produce a reordering tree. For a length-N sentence, there are $N(N-1)/2$ hyper edges in the hyper graph $\mathcal{D}$. For each hyper edge, there are two possible orientations $straight$ and $inverted$. Hence, the feature generation is performed $N(N-1)$ times in total.

With careful design of the feature function, the feature generation can be parallelized: If the feature function is defined only in a single level of the tree, in other words, a feature set generated from the feature function for a hyper edge is independent from that for the other edge. Therefore, two feature sets for two orientations are stored for each hyper edge, and thus $N(N-1)$ feature sets in the hyper graph in total.

Table 1: The statistics of corpora. Figures are the number of sentences. The first column shows the number of parallel sentences, and the second and third column show the numbers of monolingual sentences in German and English, respectively.

Data source	Parallel	German	English
WIT[3] [17]	138,499	146,206	158,641
Newswire	58,908	Not Used	
Europarl	2,399,123	Not Used	
Comman Crawl	1,920,209	Not Used	
News Commentary	178,221	204,276	247,966
News Crawl 2007	0	1,965,298	3,782,548
News Crawl 2008	0	6,690,332	12,954,477
News Crawl 2009	0	6,352,613	14,680,024
News Crawl 2010	0	2,899,914	6,797,225
News Crawl 2011	0	16,037,788	15,437,674
News Crawl 2012	0	20,673,844	14,869,673
Total	4,694,960	54,970,271	68,828,228

2.2.3. *On disk feature*

For each iteration, the feature sets generated by the feature function are identical, and the feature weights are only updated. To avoid redundant feature generation processes, reusing the generated features help the later iteration speed up. As the number of generated features is usually huge, however, it might be impossible store them in memory.

Our proposed method writes the features on disk after the generation instead of keeping them in memory. We simply create a file for each sentence with a identification of the sentence in the file name. At the actual parsing time, the generated features are recovered from the file for each sentence. Then the parser begins to search the best permutation π using the features according to the discriminative model. For each iteration, in other words, we skip the feature generation process and reuse the generated features at the first time.

3. Experimental result

In our experiments, we developed a reordering parser based on [11], `LADER`[1], and utilized a phrase-based SMT system `Moses` [13] for a reordering module and SMT module, respectively. The `tokenize.perl`[2] segmented German and English sentences into words. Word alignment of the segmented sentence pairs was performed using `MGIZA++` [18] for both German↔English directions, and refined using the

[1] `https://github.com/hwidongna/lader`

[2] `http://statmt.org/wmt08/scripts.tgz`

Table 2: The official evaluation results. XYZ in the first column refers the source X, the target Y and the priority of our run, where 1 is the primary and 2 is the contrastive. tst2013* denotes the results are measured on the reference with disfluency.

		Case-sensitive		Case-insensitive	
Run	Data	BLEU	TER	BLEU	TER
DE1	tst2013*	0.2126	0.6760	0.2174	0.6671
DE1	tst2013	0.2117	0.6890	0.2165	0.6804
ED1	tst2011	0.2348	0.5370	0.2406	0.5289
ED1	tst2012	0.2043	0.5913	0.2102	0.5805
ED1	tst2013	0.2243	0.5757	0.2300	0.5657
ED2	tst2011	0.2370	0.5337	0.2432	0.5256
ED2	tst2012	0.2036	0.5892	0.2105	0.5780
ED2	tst2013	0.2237	0.5764	0.2296	0.5665

grow-diag-final-and heuristics. A reordering parser utilized words and their automatically derived classes in the feature function. The training instances of the reordering parser were randomly selected among the word-aligned sentence pairs that licensed under ITG (around 3.5M sentences). For each iteration, the feature weights were updated using 10K instances according to Algorithm 1 and the maximum number of iterations was set to 100. We used the data supplied by the organizers of listed on the IWSLT 2013 Evaluation Campaign site. Table 1 summarizes the data statistics.

We submitted three runs: one for German-to-English (DE1) and two for English-to-German (ED1 and ED2). Our primary runs (DE1 and ED1) were the results of the preordering framework explained in Section 2. ED2 was a contrastive run using a hierarchical phrase-based SMT, which requires a longer time to decode. The decoding time of ED1 is almost half of ED2 excluding the reordering time. Table 2 shows the official results of the evaluation. The results from the other participant can be found in the overview paper [19].

Acknowledgement This work was partly supported by the IT R&D program of MSIP/KEIT (10041807), the CSLi corporation, the BK 21+ Project, and the National Korea Science and Engineering Foundation (KOSEF) (NRF-2009-0075211).

4. References

[1] Y. Zhang, R. Zens, and H. Ney, "Chunk-level reordering of source language sentences with automatically learned rules for statistical machine translation," in *Proceedings of SSST, NAACL-HLT 2007/AMTA Workshop on Syntax and Structure in Statistical Translation*, 2007, pp. 1–8.

[2] R. Tromble and J. Eisner, "Learning linear ordering problems for better translation," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*. Association for Computational Linguistics, 2009, pp. 1007–1016.

[3] K. Visweswariah, R. Rajkumar, A. Gandhe, A. Ramanathan, and J. Navratil, "A word reordering model for improved machine translation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 486–496.

[4] M. M. Khapra, A. Ramanathan, and K. Visweswariah, "Improving reordering performance using higher order and structural features," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 315–324. [Online]. Available: http://www.aclweb.org/anthology/N13-1032

[5] F. Xia and M. McCord, "Improving a statistical mt system with automatically learned rewrite patterns," in *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 508.

[6] M. Collins, P. Koehn, and I. Kučerová, "Clause restructuring for statistical machine translation," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2005, pp. 531–540.

[7] P. Xu, J. Kang, M. Ringgaard, and F. Och, "Using a dependency parser to improve smt for subject-object-verb languages," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009, pp. 245–253.

[8] D. Genzel, "Automatically learning source-side reordering rules for large scale machine translation," in *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 376–384.

[9] H. Isozaki, K. Sudoh, H. Tsukada, and K. Duh, "Head finalization: A simple reordering rule for sov languages," in *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*. Association for Computational Linguistics, 2010, pp. 244–251.

[10] J. DeNero and J. Uszkoreit, "Inducing sentence structure from parallel corpora for reordering," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '11.

Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 193–203. [Online]. Available: http://dl.acm.org/citation.cfm?id=2145432.2145455

[11] G. Neubig, T. Watanabe, and S. Mori, "Inducing a discriminative parser to optimize machine translation reordering," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, ser. EMNLP-CoNLL '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 843–853. [Online]. Available: http://dl.acm.org/citation.cfm?id=2390948.2391039

[12] D. Wu, "Stochastic inversion transduction grammars and bilingual parsing of parallel corpora," *Computational linguistics*, vol. 23, no. 3, pp. 377–403, 1997.

[13] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, *et al.*, "Moses: Open source toolkit for statistical machine translation," in *Annual meeting-association for computational linguistics*, vol. 45, 2007, p. 2.

[14] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *J. Mach. Learn. Res.*, vol. 7, pp. 551–585, Dec. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248566

[15] L. Huang and D. Chiang, "Better k-best parsing," in *Proceedings of the Ninth International Workshop on Parsing Technology*, ser. Parsing '05. Stroudsburg, PA, USA: Association for Computational Linguistics, 2005, pp. 53–64. [Online]. Available: http://dl.acm.org/citation.cfm?id=1654494.1654500

[16] A. Dunlop, N. Bodenstab, and B. Roark, "Efficient matrix-encoded grammars and low latency parallelization strategies for cyk," in *Proceedings of the 12th International Conference on Parsing Technologies*, ser. IWPT '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 163–174. [Online]. Available: http://dl.acm.org/citation.cfm?id=2206329.2206349

[17] M. Cettolo, C. Girardi, and M. Federico, "Wit^3: Web inventory of transcribed and translated talks," in *Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT)*, Trento, Italy, May 2012, pp. 261–268.

[18] Q. Gao and S. Vogel, "Parallel implementations of word alignment tool," in *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, ser. SETQA-NLP '08. Stroudsburg, PA, USA: Association for Computational Linguistics, 2008, pp. 49–57. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622110.1622119

[19] M. Cettolo, J. Niehues, S. Stker, L. Bentivogli, and M. Federico, "Report on the 10th iwslt evaluation campaign," in *Proceedings of the 10th International Workshop on Speech Language Translation*, 2013.