Word-reordering for Statistical Machine Translation Using Trigram Language Model

Jing He
IIIS, Tsinghua University
Beijing, China
hejing2929@gmail.com

Hongyu Liang
IIIS, Tsinghua University
Beijing, China
sjqxzlhy@gmail.com

Abstract

In this paper we study the word-reordering problem in the decoding part of statistical machine translation, but independently from the target language generating process. In this model, a permuted sentence is given and the goal is to recover the correct order. We introduce a greedy algorithm called *Local-*(k,l)*-Step*, and show that it performs better than the DP-based algorithm. Our word-reordering algorithm can be used in the statistical machine translation process for improving the quality of the translation. Furthermore, motivated by the rank evaluation method, we introduce a novel way for evaluating the results of word-reordering by calculating the inversion pair cardinality.

1 Introduction

Statistical machine translation is a machine translation method based on statistical models, which is in contrast with rule-based machine translation as well as with example-based machine translation. The most commonly used model in statistical machine translation is the source-channel model built by Brown et. al. (1993). In their proposal of translation between English and French, English strings are generated according to some stochastic process and then transformed stochastically into French strings. Therefore, to translate French to English, it is needed to search for an English source string that is most likely according to the English language model (Ponte and Croft, 1998) and the channel model. This process of translation is called decoding. Usually, a decoding process in statistical machine translation is combined with two sub-processes (Chang and Toutanova, 2007; Koehn, Och and Marcu, 2003): generating the words or phrases of the target language, and deciding the correct order of the words or phrases to get a desired target language sentence. For some language pairs, such as English and Chinese, the word-reordering problem is really hard to solve, as the target word order differs a lot from the source word order and little information about the target word order is obtainable from the source sentence. This is because the grammars of English and Chinese differ from one another significantly.

Language model has been successfully applied in the word-reordering process of statistical machine translation. Generally speaking, a language model assigns a probability to a sequence of words according to some probability distribution. A sentence then gets a score under the language model by means of standard conditional probability. Knight (1999) studied this abstract problem and proved that the word-reordering (also called word-replacement) problem under bigram language model is **NP**-hard.

In this article, we study the word-reordering problem under trigram language model in the decoding part of statistical machine translation, but independently from the target language generating process. More precisely, suppose we want to translate a sentence from one language to another, and some methods have been applied to generate all the target words, whereas the words need reordering because two languages may have totally different grammars (like English and Chinese). Thus, our goal is to recover the correct, or reasonable, word order of the target sentence.

We reduce this problem to the traveling salesman problem (TSP) in 3-uniform hypergraphs. We show that by some modification, the dynamic-programming (DP) based algorithm for TSP (Held and Karp, 1962) can be used in solving this generalized problem. Nevertheless, the time complexity of this DP-based algorithm is exponential in the number of words in the sentence, and is thus unreasonable in practice. We design a class of greedy algorithms called **Local-(k, l)-Step**, parameterized by k and l. Roughly speaking, the **Local-(k, l)-Step** algorithm finds, in each step, k words which maximize the language model score, and then add the first l words to the partial sentence. For small k, say $k < 5$, this algorithm runs much faster then the DP-based algorithm.

We also propose a novel way to evaluate the results of word-reordering, motivated by the rank (ordering) evaluation measures in information retrieval. Standard retrieval evaluation measures, such as Mean Average Precision (MAP), are used for evaluating the results of retrieval systems. The reordering or ranking of items is an important task in many real-world applications, and it is needed to compare two different orderings of the same item list. One commonly-used measure is the Kendall's tau coefficient (Kendall, 1938) which uses the notion of *inversion pairs*.

Motivated by these existing methods, we design the following evaluation process for word-reordering: Given a sentence outputted by the algorithm, we regard it as a permutation of the correct sentence, and count the the number of inversion pairs in it, which can be seen as the distance between the result sentence and the correct one. The notion of inversion pairs is often used to measure the distance between permutations. Due to the special structure of our model, it is also a proper measurement of the quality of word-reordering results. Note that to calculate the number of inversion pairs, the correct sentence, or a "standard answer", should be given as well. This can be done by the following design of experiments: We choose 1500 English sentences from http://www.nlp.org.cn, and randomly permute each sentence. Then, the permuted sentences are given as the input to the word-reordering problem, while the original sentences are just "standard answers" which will be used in the evaluation.

We implement both the DP-based algorithm and the **Local-(k, l)-Step** algorithm, and evaluate their results according to their performance on the chosen sentences. From the comparison, it is shown that for well chosen parameters k and l, the **Local-(k, l)-Step** algorithm performs even better than the DP-based algorithm. This seems to contradict with the fact that the latter solves the problem exactly while the former only obtains an "approximate" solution. This, however, is not a real problem since the score under the language model is not always compatible with that under the evaluation using inversion pairs. In fact, neither of the two evaluation methods can accurately measure the "quality" or "correctness" of sentences, because the human language itself comprises many other perspectives that cannot be qualified or characterized exactly by current techniques. It is possible that a sentence that makes no sense obtains a higher language model score than that of a normal sentence in real world.

Finally let us mention that, although throughout the paper we talk about the word-based reordering model, our algorithms can be easily modified to be applicable in the re-ordering process of statistical machine translation whose working unit is phrase (Koehn, Och and Marcu, 2003).

This paper is organized as follows: Section 2 defines the word-reordering model rigorously, and introduces the dynamic programming algorithm and the greedy **Local-(k, l)-step** algorithm. In Section 3 we show the experimental results as well as the evaluation based on the inversion pair cardinality. The last section concludes the whole article with some remarks and future work.

2 The Word-reordering Model

In the word-reordering model, a disordered English sentence is given and the goal is to find the most reasonable order. For example, given the sentence "overrate to is importance their it easy", the best answer should be "It is easy to overrate their importance."

It is hard to establish a standard criterion for evaluating the "quality" of a sentence. In practice, language model is used as a statistical tool to help people find approximate solutions. For our purpose, we adopt the commonly-used trigram source model, given by $lm(w_0|w_1, w_2)$ for all possible English words w_0, w_1 and w_2. Given a disordered sentence, we want to rearrange its words in order to get a maximum score according to the trigram source model. This can be theoretically formalized as the following search problem.

Word-reordering Problem
Input:

1. A dictionary $D = \{d_i \mid 1 \leq i \leq m\}$;
2. A trigram source model $\{lm(w_i \mid w_j, w_k) : 1 \leq i, j, k \leq m\}$;
3. A set of words $S = \{s_1, s_2, \ldots, s_n\} \subseteq D$.

Output: A sentence $s_{i_1} s_{i_2} \ldots s_{i_n}$, where $(i_1, i_2, \ldots, i_n)$ is a permutation of $\{1, 2, \ldots, n\}$, such that $\prod_{j=3}^{n} lm(s_{i_j} \mid s_{i_{j-1}}, s_{i_{j-2}})$ is maximized.

A similar problem with bigram source model $\{lm(w_i|w_j)\}$ is proved to be NP-hard by Knight (1999) using a reduction from the famous HAMILTONIAN PATH problem. The NP-hardness of this problem with trigram source model can be proved analogously.

Despite the hardness result, we can still cope with many instances in practice, since a sentence is usually not so long. The trivial way is to enumerate all $n!$ permutations and find the one achieving maximum score, which takes $O(n \cdot n!)$ time. We will describe better ways for solving it.

2.1 Dynamic Programming

An instance of the TSP problem consists of a directed graph $G = (V, E)$ and a cost function $c : E \to \mathbf{R}$, and the goal is to find a path of minimum cost which visits each vertex in V exactly once (that is, a Hamiltonian path). Note that the word-reordering problem can be reduced to TRAVELING SALESMAN PROBLEM (TSP) with a slight modification that each edge in the graph is a triple instead of a pair (i.e., TSP in 3-uniform hypergraphs). The reduction is as follows: Construct a vertex v_i for each word $s_i \in S$, and then add an hyperedge (v_i, v_j, v_k) with cost $-\ln(lm(w_k|w_i, w_j))$ for every triple (v_i, v_j, v_k). Then, finding a sentence s with maximum score is equivalent to finding a minimum cost Hamiltonian path in this hypergraph, where the cost of a path is defined to be the sum of costs of all triples containing three consecutive nodes in the path. We can assume that the graph is complete (i.e., all possible edges (v_i, v_j, v_k) exist) by adding dummy edges with sufficiently large costs.

Held and Karp (1962) designed a dynamic-programming based (DP-based) algorithm for the original TSP which runs in time $O(2^n \cdot n^2)$, n being the number of vertices in the graph. Their algorithm runs iteratively on all subsets of vertices and finds a minimum cost tour in that subset, with the start and end points specified. We will describe a similar algorithm for solving TSP on 3-uniform hypergraphs, which can be directly applied to the word-reordering problem.

DP-based algorithm for TSP in 3-uniform hypergraphs (3-uni-DP)

Input: A 3-uniform hypergraph $G = (V, E)$; a cost function $c : E \to \mathbf{R}$.

Output: A minimum cost Hamiltonian path.

Algorithm:

```
• Return $fail$ if $|V| < 3$.
• For each triple $(v_i, v_j, v_k)$ where all three vertices are pairwise distinct, let
  $C(\{v_i, v_j, v_k\}, v_i, v_j, v_k) = c(v_i, v_j, v_k)$.
• For $m = 4$ to $|V|$ do
    – For each tuple $(V', v_1, v_2, v_3)$ where $\{v_1, v_2, v_3\} \subseteq V' \subset V$ and $|V'| = m$, compute:
      $C(V', v_1, v_2, v_3) = \min_{v_4 \in V'}\{C(V' \setminus \{v_3\}, v_1, v_4, v_2) + c(v_4, v_2, v_3)\}$
• Find the vertices $v_1, v_2, v_3$ for which $C(V, v_1, v_2, v_3)$ is minimized. Trace back to find the corresponding path.
```

The algorithm **3-uni-DP** runs in time $O(2^n \cdot n^4)$, where $n = |V|$. In each step it computes $C(V', v_1, v_2, v_3)$, which stands for the minimum cost Hamiltonian path in V' that starts with v_1 and ends at (v_2, v_3), by enumerating the third vertex on the path from end and concatenating the shorter path and the last edge. It is not hard to see that this algorithm correctly computes the minimum cost Hamiltonian path in 3-uniform hypergraphs.

2.2 Local-(k, l)-Step Algorithm

The DP-based algorithm **3-uni-DP** solves the word-reordering problem exactly but runs in exponential time, which is unaffordable for long sentences. An idea for reducing the running time is to consider the problem "locally". In each step, we look for a fixed number of unvisited points and try to minimize the cost of the "local path" in which these points are involved. More specifically, we seek for k unvisited points $v_2, v_3, \ldots, v_{k+1}$ minimizing $\sum_{i=0}^{k-1} c(v_i, v_{i+1}, v_{i+2})$, where v_0 and v_1 are the last two nodes in the current partial path. Then, we add the first l points to our partial path, for some $l \leq k$. We give a more formal description as follows.

Algorithm Local-(k, l)-Step

Input: A 3-uniform hypergraph $G = (V, E)$; a cost function $c : E \to \mathbf{R}$.

Output: A minimum cost Hamiltonian path.

Algorithm:

Do the following for all vertex pairs (v_0, v_1) to find the best solution:

```
• $V' \leftarrow V \setminus \{v_0, v_1\}$.
• $PartialPath \leftarrow (v_0, v_1)$.
• While $|V'| \geq k$ do
    – Find $k$ distinct vertices $v_2, v_3, \ldots, v_{k+1}$ in $V'$ which minimizes $\sum_{i=0}^{k-1} c(v_i, v_{i+1}, v_{i+2})$.
    – Add $(v_2, \ldots, v_{l+1})$ to the end of $PartialPath$.
    – $V' \leftarrow V' \setminus \{v_2, v_3, \ldots, v_{l+1}\}$.
• In case $V' \neq \emptyset$, perform an exhaustive search to find the best order to visit the remaining vertices in $V'$, and add this to $PartialPath$.
```

Return the best $PartialPath$ (with the minimum cost) over all start pairs (v_0, v_1).

The algorithm **Local-(k, l)-Step** runs in time $O(n^2 \cdot n^k \cdot \frac{n}{l}) = O(n^{k+3}/l)$, where n is the number of vertices in the graph. This is efficient in practice if we choose a small k, say, $k < 5$. It should be noticed that this algorithm cannot be a constant-factor approximation algorithm for TSP, since for any constant k it runs in polynomial time, and thus cannot approximate TSP within any constant ratio unless **P** $=$ **NP**. However, it may perform well on real-world instances of the word-reordering problem.

In fact we will show that, for some well chosen parameters k and l, the algorithm **Local-(k, l)-Step** performs even better than the algorithm **3-uni-DP**. This seems to contradict with the fact that the latter solves the problem optimally while the former only looks for a reasonable solution. However, this is not a problem because our word-reordering model itself cannot catch accurately the quality of a sentence. Thus, when evaluating the outcomes of our algorithms, it is more proper to use some other measurements, like the *inversion pair* which will be introduced later. The reason why this measurement cannot be adopted in the design of our algorithms is that it can only be calculated if a standard answer to the problem is given.

3 Experiments

3.1 The trigram language model training

In order to build a reasonable trigram language model for the experiment, we download the third version of the Europarl corpus (Koehn, 2005) which is extracted from the proceedings of the European Parliament. This data set is usually used as a base material in statistical machine translation contest or other research projects involving European languages, and the English part can be used for training a trigram English language model. There are about 0.307 million English sentences in the material. Thus, the trigram language model built by SRILM (Stolcke, 2002) can reflect the properties of the English language.

3.2 The reordering experiment

How to evaluate the results of our algorithms is a key problem in our research, as there is no standard for testing the accuracy of word-reordering. Usually, the reordering of words in sentences serves as a subprocess in statistical machine translation, especially in the decoding step. All the existing methods and standards are designed to test the accuracy of the translation results, but not the single process of word-reordering. Therefore, we designed an experiment model and a testing principle for our own purpose, shown as follows.

First, we choose 1500 English-Chinese sentence pairs (from http://www.nlp.org.cn) as the collection of standard answers. We then choose from them all sentences of length less than 10 as our data set. For every sentence in this data set, we generate a disordered sentence by performing a random permutation on the set of words in the original sentence. For example, if the original sentences is “sometimes you are overly frank”, one possible disordered sentence could be “overly are sometimes frank you”. The original sentence is used as a standard answer for later evaluation. We run the **Local-(k, l)-Step** algorithm on all the permuted sentences for all pair of parameters (k, l) such that $1 \leq l \leq k \leq 4$. We also run the **3-uni-DP** algorithm on these sentences for the sake of contrast.

There is another issue considering the start of a sentence. Since the language model includes the possibility of a word being the first word of a sentence, we may add a special “start symbol” to every disordered sentence and force it to be the first word in the output sentence. This is easy to implement in practice, and will make the result more reasonable. In all the experiments we adopt this setting.

Now comes the evaluation part. How to measure the quality of our results, or, the distances between the output sentences and the standard answers? It turns out that any method for determining the distance between permutations also works in our model. The concept of *inversion pairs* is usually used to measure the distance between two permutations, and is thus brought into our experiments. Let σ be a permutation of $\{1, 2, \ldots, n\}$. A pair of indices (i, j), where $1 \leq i < j \leq n$, is called an *inversion pair* of σ if $\sigma(i) > \sigma(j)$. The total number of inversion pairs of σ, also called *inversion pair cardinality* of σ, is a proper measurement of the distance between σ and the identity permutation $(1, 2, \ldots, n)$. Although distances between two arbitrary permutations can be similarly defined, this measurement already suffices for our purpose since we only need to calculate the distance between a permuted sentence and the standard answer.

Take again the sentence "sometimes you are overly frank" as an illustration. We mark this sentence as the identity permutation $(1, 2, 3, 4, 5)$. If the disordered sentence is "sometimes are you frank overly", it should be associated with the permutation $(1, 3, 2, 5, 4)$, and thus has 2 inversion pairs in total (3 comes before 2, and 5 appears before 4).

In this way we can calculate the number of inversion pairs to measure the degree of disordering. For every pair of parameters (k, l) used in the **Local-(k, l)-Step** algorithm, we count the number of output sentences which have smaller inversion pair cardinality than the corresponding randomly permuted sentences. Call this number $Better(k, l)$. We then choose the pair (k, l) which maximizes $Better(k, l)$, denoted by (k_0, l_0), and adopt it as the proper parameter for our algorithm. We also calculate $Better(DP)$, which is the number of sentences outputted by the **3-uni-DP** algorithm having smaller inversion pair cardinality than the corresponding disordered ones.

All the experiments are conducted repeatedly for 5 times, each time generating different randomly disordered sentences. The final result $Better(DP)$ and $Better(k, l)$ are the average of the results of 5 independent experiments. These results are shown in Tables 1 and 2.

	1	2	3	4	5	average
(1,1)	347	354	329	360	349	347.8
(2,1)	376	389	370	386	388	381.8
(2,2)	383	390	369	395	388	385
(3,1)	392	396	386	399	399	394.4
(3,2)	392	397	383	388	388	389.6
(3,3)	396	406	380	401	387	394
(4,1)	399	413	390	405	407	402.8
(4,2)	400	406	398	407	403	402.8
(4,3)	412	414	401	421	416	412.8
(4,4)	406	412	404	421	420	412.6
DP	362	370	364	372	353	364.2

Table 1: $Better(DP)$ and $Better(k, l)$ under all chosen parameters

	average number	percentage
$Better(k_0, l_0)$	412.8	62.55%
$Better(DP)$	364.2	55.18%
All sentences	660	100%

Table 2: The comparison between DP-based and greedy algorithms

3.3 Analysis

The experimental result shows that setting $(k, l) = (4, 3)$ gives the best outcome among all chosen parameters. Thus we let $k_0 = 4$ and $l_0 = 3$. It is a little surprising that $Better(DP)$ is less than $Better(k_0, l_0)$, which indicates that the exact algorithm performs worse than the "approximate" algorithm. As explained before, this is due to our lack of modeling the "quality" or "correctness" of sentences. It is possible that a sentence with high score under trigram language model makes little sense, and a normal sentence appearing in real word obtains a low score under this language model. The algorithm **Local-(k, l)-Step**, on the other hand, makes use of the locality of English sentences, and thus it should be expected to perform well.

4 Conclusions

In this paper, we study the word-reordering problem in the decoding process of statistical machine translation. We adopt the commonly-used trigram language model, and abstract the word-reordering problem as instances of Traveling Salesman Problem in 3-uniform hypergraphs. We show that Held and Karp's dynamic-programming based algorithm for solving TSP (in normal graphs) can be adapted to solve this problem. We also design a greedy algorithm called **Local-(k, l)-Step**, parameterized by k and l, which has a faster running time but gets a non-optimal solution, where by optimal we mean achieving maximum score under the trigram language model.

We implement both algorithms and conduct some experiments. To evaluate the results, we adopt the concept of inversion pairs to measure the distances between the standard answers and the sentences outputted by the algorithms. From the experimental results, we find that setting $k = 4$ and $l = 3$ makes the **Local-(k, l)-Step** algorithm perform best. Moreover, the result obtained by the greedy algorithm is even better than that of the dynamic-programming based algorithm. This is because we are not able to model the quality and correctness of sentences accurately, and thus a sentence with maximum score under trigram language model is not necessarily a best answer to the reordering problem.

Since we study the word-reordering problem independently from the target language generating process, one future direction for our research is to combine the two parts, namely, predicting the collection of words and deciding the correct order of the target sentences, together to design better decoding algorithms.

References

M. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1–2): 81–89, 1938.

P. Brown, S. Della-Pietra, V. Della-Pietra, and R. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2): 263-311, 1993.

P. Chang and K. Toutanova. A Discriminative Syntactic Word Order Model for Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 9-16, Prague, Czech Republic, June 2007.

M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of Society for Industrial and Applied Mathematics*, 10(1):196–210.

K. Knight. Decoding Complexity in Word-Replacement Translation Models. *Computational Linguistics*, 25(4):607–615, 1999.

P. Koehn. Europarl: A Parallel Corpus for Statistical Machine Translation. MT Summit, 2005.

P. Koehn, F. J. Och, and D. Marcu. Statistical Phrases-Based Translation. HLT/NAACL, 2003.

J. M. Ponte and W. B. Croft. A Language Model Approach to Information retrieval. *Research and Development in Information Retrieval*, 275–281, 1998.

A. Stolcke. SRILM – An Extensible Language Modeling Toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, vol. 2, pp. 901–904, Denver, 2002.

A. Aho and J. Ullman. *The Theory of Parsing, Translation and Compiling*, vol. 1. Prentice-Hall, Englewood Cliffs, NJ, 1972.