FAST DECODING FOR STATISTICAL MACHINE TRANSLATION

Ye-Yi Wang   *Alex Waibel*

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA

ABSTRACT

We investigated an efficient decoding algorithm for statistical machine translation. Compared to the other algorithms, this new algorithm is applicable to different translation models, and it is much faster. Experiments showed that the algorithm achieved an overall performance comparable to the state of the art decoding algorithms.

1. INTRODUCTION

A statistical machine translation system consists of three sub-tasks: the modeling task describes machine translation processes with stochastic models; the learning task estimates the parameters in the models; and the decoding task searches for the translation that has the highest score according to the models. [1, 2, 3] described different translation models and their learning algorithms. [4, 2, 5, 6] introduced different decoding algorithms. However, those decoding algorithms have many limitations. Below is a brief review of these algorithms:

1.1. IBM Stack Decoder

In the IBM Stack Decoder [4], a hypothesis is comprised of a source sentence prefix string and an alignment between the prefix string and the target (input) sentence. Each hypothesis is associated with a model score. For a given target sentence, each subset of the words in that target sentence is associated with a priority queue. A hypothesis is put into one of the priority queues according to the target words that have been accounted for by that hypothesis. Because there are 2^n (n is the target sentence length) subsets of target words that can be accounted for by a hypothesis, the number of priority queues is exponential in the target sentence length. A long target sentence will lead to huge number of priority queues; hence too much memory space will be allocated. When an input target sentence is longer than 15 words, the decoder can allocated more than 1 GB memory. Exponential number of priority queues also implies that exponential number of hypotheses have been generated by the decoder. Therefore the decoder is extremely slow. In our experiments with IBM decoder for Model 3 [1], we let the decoder stop searching and register a failure whenever it allocated more than 750 MB memory. Table 1 shows the number of sentences that IBM decoder had failed, as well as the number of states being extended.

Sentence Length	Total	Failed	Pct	Extended Hypo #
1-4	81	0	0%	16
5-8	128	0	0%	2,705
9-12	101	0	0%	11,660
13-16	41	19	46.3%	63,472
17-20	16	9	56.3%	145,768
All (1-20)	367	28	7.6%	—

Table 1: Performance for the IBM Model 3 Stack Decoder: Input target sentences are grouped according to their lengths. The second column lists the total number of sentences in each group. The third column lists the number of sentences that the decoder failed on. The fourth column lists the failure percentage. The fifth column lists the average numbers of hypotheses being extended by the decoder, which were collected with those successfully decoded sentences.

1.2. Dynamic Programming Decoding Algorithms

[2] and [6] described dynamic programming decoding algorithms for statistical machine translation. While dynamic programming algorithms worked fast, they imposed a strong constraint on the translation model: no crossover was allowed in word-to-word alignments between parallel sentences. This constraint basically requires that the source and target languages have very similar word orders. In case when the word orders are different (like English and German), a preprocessor is required to make the two languages similar.

1.3. A* Decoding Algorithm

[5] described an A* decoding algorithm. While this algorithm was much faster than the IBM decoder, it was only applicable to IBM Model 1 and Model 2, since its scoring mechanism relied on the reinterpretation of a probabilistic equation specific to the models. It is not clear how this algorithm can be used for more complicated models.

In summary, the current decoding algorithms are either too inefficient or too restrictive. And all of them are not generally applicable. For example, they will not work with the more complicated structure-based model [3].

2. FAST STACK DECODER FOR MODEL 1 AND MODEL 2

The high failure rate and the slow speed of the IBM stack decoder were due to the same reason — retaining the alignment between a source sentence prefix and the target sentence in a hypothesis. Because the number of possible alignments is exponential in sentence length, this results in exponential number of priority queues and hypotheses. Therefore the algorithm is too expensive with respect to both time and space complexities.

On the other side, efficient algorithms are available for Model 1 and Model 2 to calculate $P(\mathbf{g} \mid \mathbf{e}) = \sum_A P(\mathbf{g} \mid A, \mathbf{e})$[1], the *a posteriori* probability of a target sentence $\mathbf{g}$ given a source $\mathbf{e}$, over all possible alignments A. Therefore we do not have to make assumption about the alignment between a source sentence prefix and the target sentence. Instead, a hypothesis can be just a prefix string of the source sentence, whose score is the likelihood of the target sentence summed over all possible alignments. In doing so, we greatly reduced the size of hypothesis space and make the decoding algorithm more efficient.

To be specific, here we present a modified fast decoding algorithm for Simplified Model 2 [5]. The decoder for Model 1 can be simply derived from it.

An important issue here is how we score a hypothesis. In Simplified Model 2, the equation for the *a posteriori* likelihood of a target sentence $\mathbf{g}$ given a source sentence $\mathbf{e}$ can be used to assess a hypothesis:

$$
\begin{aligned}
P(\mathbf{g} \mid \mathbf{e}) &= \epsilon \sum_{a_1=0}^{l} \cdots \sum_{a_m=0}^{l} \prod_{j=1}^{m} t(g_j \mid e_{a_j}) a_l(a_j \mid j) \\
&= \epsilon \prod_{j=1}^{m} \sum_{i=0}^{l} t(g_j \mid e_i) a_l(i \mid j) \qquad (1)
\end{aligned}
$$

here $l = |\mathbf{e}|$ and $m = |\mathbf{g}|$.

Although (1) was obtained from the alignment model, it would be easier for us to describe the scoring method if we interpret the last expression in the equation as follows: each word e_i in the hypothesis contributes the amount $\epsilon t(g_j \mid e_i) \times a_l(i \mid j)$ to the probability of the target sentence word g_j.

Given the target sentence $G = g_1 g_2 \cdots g_m$, assume that the source sentence length is l at this moment. The hypothesis $H_l = l : e_1, e_2, \cdots, e_k$ has hypothesized k words as the prefix of the source sentence of length l. Then the probability mass contributed by the source word e_i ($0 \le i \le k$) to the target word g_j is $\epsilon t(g_j \mid e_i) \times a_l(i \mid j)$. For a source position $k < i \le l$, since the source word at that position has not be introduced into the prefix, its contribution to the target word g_j is averaged over all possible source words, which is $\epsilon a_l(i \mid j) \times \sum_{k=0}^{|L|} \Pr(w_k) \times t(g_j \mid w_k)$. Here $|L|$ is the size of the source language lexicon, w_k is the kth

[1] e and g are used here because English and German are the source and target languages in our system.

word in the source lexicon, and $\Pr(w_k)$ is the prior probability of the source word w_k, which can be obtained with the maximum likelihood estimator. Therefore, if we use $\tau_{kl}(j \mid i, H_l)$ to denote the contribution of the ith source position of H_l to the probability mass of the jth target word, we have

$$
\tau_{kl}(j \mid i, H_l) = \qquad (2)
$$
$$
\begin{cases}
\epsilon\, a_l(i \mid j) t(g_j \mid e_i) & 0 \le i \le k \\
\epsilon\, a_l(i \mid j) \sum_{k=0}^{|L|} \Pr(w_k) t(g_j \mid w_k) & k < i \le l
\end{cases}
$$

The translation model score of H_l is therefore

$$
\tau(H_l) = \prod_{j=1}^{m} \sum_{i=0}^{l} \tau_{kl}(j \mid i, H_l) \qquad (3)
$$

In practice, since we do not make any assumption of the source sentence length, the score of a hypothesis $H = e_1, e_2, \cdots, e_k$ has to be averaged over all possible sentence lengths:

$$
\tau(H) = \sum_{i=k}^{L_m} \Pr(k \mid m) \times \tau(H_i) \qquad (4)
$$

here $\Pr(k \mid m)$ is the source sentence length distribution conditioned on the target sentence length, which was modeled with Poisson distributions. L_m is the maximum sentence length allowed.

Because our objective is to maximize $P(\mathbf{e}, \mathbf{g})$, we have to include the ngram language model probability of the prefix string. Therefore the score of H is

$$
S(H) = \tau(H) \times \prod_{i=1}^{k} P(e_i \mid e_{i-N+1} \cdots e_{i-1}). \qquad (5)
$$

Because of the different number of factors in the language model score, hypotheses of different prefix lengths are not comparable. Therefore hypotheses are stored in different priority queues according to their prefix lengths. This results in the following algorithm:

Algorithm 1 Fast Stack Decoder for Simplified Model 2

> **Input:**   target sentence $T = t_1 t_2 \cdots t_n$.
> **Output:** source sentence $S = s_1 s_2 \cdots s_m$.
> **Data Structures:**
>   a set of priority queues $\mathbf{Q}_0 \mathbf{Q}_1 \cdots \mathbf{Q}_{L_m}$
>   for hypotheses.

1. Initialize with a null hypothesis (with prefix string length 0) H_0, compute $S(H_0)$ with (2), (3), (4) and (5).

2. $\mathbf{Q}_0 \leftarrow H_0$

3. For each $\mathbf{Q} \in \{\mathbf{Q}_0 \mathbf{Q}_1 \cdots \mathbf{Q}_{L_m}\}$ and $H \in \mathbf{Q}$

4.   set the threshold for $\mathbf{Q}$

5.   if $S(H) > Threshold(\mathbf{Q})$

6.   for each promising source word s

7. $H' = \text{append}(H, s)$

8.   score H' with (2), (3), (4) and (5).

9. $\mathbf{Q}_{|H'|} \leftarrow H'$

10.   exit the loop if N complete source sentences are available in $\mathbf{Q}$'s.

11. Report the hypothesis with the highest score in $\mathbf{Q}'s$ as the translation of T.

3.   HYPOTHESIS RESHUFFLING

The aforementioned algorithm is only applicable to Model 1 and Model 2. There is no efficient way to compute the likelihood of a target given a source over all possible alignments for more complicated models. To apply the algorithm to those models, we present a hypothesis reshuffling algorithm here. The idea of hypothesis reshuffling was based on the observation that the fast stack decoder often found, in the top N translation candidates, the correct translations, or almost correct translations — translations that had the correct bags of words arranged in wrong orders.

The hypothesis reshuffling algorithm uses the decoder for a simple model (e.g., the fast stack decoder for Simplified Model 2) to find top N hypotheses. It then searches for the translation that is the best according to a more complicated model, in the neighborhood of those candidate hypotheses. We define the following terminology to describe the algorithm:

Definition 1 *Word move*

Two hypotheses $H = e_1 e_2 e_3 ... e_n$ and $H' = e'_1 e'_2 e'_3 ... e'_n$ differ by a word move if there exist $1 \leq i \leq j \leq n$ such that either of the following holds:

$(e_1 ... e_{i-1} = e'_1 ... e'_{i-1}) \wedge (e_i = e'_j) \wedge (e_{i+1} ... e_j = e'_i ... e'_{j-1}) \wedge (e_{j+1} ... e_n = e'_{j+1} ... e'_n)$ or

$(e'_1 ... e'_{i-1} = e_1 ... e_{i-1}) \wedge (e'_i = e_j) \wedge (e'_{i+1} ... e'_j = e_i ... e_{j-1}) \wedge (e'_{j+1} ... e'_n = e_{j+1} ... e_n)$

Definition 2 *Word swap*

Two hypotheses $H = e_1 e_2 e_3 ... e_n$ and $H' = e'_1 e'_2 e'_3 ... e'_n$ differ by a word swap if $e_k = e'_k$ holds for all $1 \leq k \leq n$ except for $1 \leq i \leq j \leq n$, for which we have $(e_i = e'_j) \wedge (e_j = e'_i)$.

Definition 3 *Neighbor hypothesis*

Two hypotheses H and H' are neighbors iff H and H' differ by a word move or a word swap.

The search process can be described with the following algorithm:

Algorithm 2 Decoding with Hypothesis Reshuffling

Input: target sentence $T = t_1 t_2 \cdots t_n$.
Output: source sentence $S = s_1 s_2 \cdots s_m$.
Data Structures:
             a priority queue $\mathbf{Q}$ for hypotheses.
Models: a base model M_1 for candidate hypothesis;
             a model M_2 for rescoring the candidate hypotheses and their neighbors.

1. Using the decoder for M_1, find the top N hypotheses. Score the hypotheses with M_2, and then add these hypotheses to Q.

2. Repeat Step 3-7, until there is no change of the top K hypotheses in Q:

3. For each of the top K hypotheses in $\mathbf{Q}$

4. $\mathcal{N}_H \leftarrow \mathbf{neighbor\_set}(H)$;

5.   for each $H' \in \mathcal{N}_H$

6.   score H' with M_2

7. $Q \leftarrow H'$

8. Report the hypothesis with the highest score in $\mathbf{Q}$ as the translation of T.

The choice of the value N and K is a trade-off between speed and accuracy. A large N and K make the hill-climbing search in the hypothesis neighborhood less likely to stop at a local maximum, while the decoding process takes more time. In experiments reported here, $N = 12$ and $K = 6$ was selected by trial and error.

When we apply the algorithm, we often let M_1 "borrow" the translation parameters from M_2, because in general the translation distribution of a source word in a more advanced model is less ambiguous and more accurate [3].

4.   EVALUATION

Two experiments were conducted to evaluate the performance of the new stack decoder + reshuffling algorithm (henceforth SD+R algorithm). In the first experiment, we compared the performance of the algorithm (with fast stack decoder for the base model, henceforth FSD+R algorithm) with that of the IBM stack decoder for Model 3. In the second experiment, we compared the performance of two different SD+R algorithms for our structure-based model[2] [3]: the first one used IBM Model 1 as the base model and applied the fast stack decoding algorithm to find the hypothesis candidates (FSD+R); the second one used IBM Model 3 as the base model and applied the IBM stack decoder to find the hypothesis candidates (IBM+R). Table 2 compares the performance: among the successfully decoded sentences, the IBM decoder and IBM+R decoder had higher accuracy (Accuracy column). However, the different algorithms performed similarly if the accuracy is

---

[2] A slight modification was made for the reshuffling algorithm for the structure-based model — we introduced phrase move and phrase swap in addition to word move and word swap in defining the neighboring hypotheses.

Model	Decoder	Total	Failed	Corr.	Okay	Incorr.	Accuracy	Accuracy*
Model 3	IBM	367	28	191	68	80	66.4%	61.3%
Model 3	FSD+R	367	2	188	74	103	61.6%	61.3%
SModel	IBM+R	367	28	202	77	60	70.9%	65.5%
SModel	FSD+R	367	2	203	76	86	66.0%	65.7%

Table 2: Performance Comparison. The first row is the performance of IBM Model 3 with the stack decoder. The second row is the performance of IBM Model 3 with fast stack decoder and hypothesis reshuffling. The third row is the performance of the structure-based model with the IBM stack decoder and hypothesis reshuffling, and the fourth row is the performance of the structure-based model with fast stack decoder and hypothesis reshuffling. The "failed" column lists the number of sentences for which the search was aborted. "Accuracy" was calculated with respect to the successfully decoded sentences, and "Accuracy*" was calculated with respect to the total input sentence (367). Here a correct translation gets 1 credit; an okay translation gets 1/2 credit; and an incorrect translation gets 0 credit.

Model	Decoder	Total Errors	$S(\mathbf{e}) > S(\mathbf{e}')$	$S(\mathbf{e}) \leq S(\mathbf{e}')$
Model 3	IBM	148	17 (11.5%)	131 (88.5%)
Model 3	FSD+R	177	26 (14.7%)	151 (85.3%)
SModel	IBM+R	137	18 (13.1%)	119 (86.9%)
SModel	FSD+R	162	28 (17.3%)	134 (82.7%)

Table 3: Reference vs. Machine-Made Translations. $S(\mathbf{e})$ is the score of reference translation, $S(\mathbf{e}')$ is the score of the machine made translation. When $S(\mathbf{e}) > S(\mathbf{e}')$, we know, for sure, that a decoding error has occurred

calculated among all input sentences (Accuracy* column). This is because the IBM decoder failed on more sentences, and usually those sentences were difficult ones and likely to result in errors with the FSD+R algorithm.

For those erroneous (okay and incorrect) translations, Table 3 compares their model scores with that of the reference translations. When a reference translation has a higher score than an erroneous translation, we know for sure that the decoder has made an error. Otherwise the error may be resulted from either the decoder or the model. Here FSD+R had higher known decoding error rate than the IBM decoder (Model 3) or the IBM decoder with reshuffling (structure-based model) had. However, since FSD+R decoded more sentences, and the IBM decoder failed on those extra sentences, it was likely that the FSD+R decoder made more mistakes on these difficult sentences and resulted in higher decoding error rate.

While the new algorithm does not improve the translation accuracy, its biggest advantage is its decoding speed. Using fast stack decoder for the base model, we do not have to differentiate hypotheses with the same prefix string but different alignments. Therefore we reduce the number of hypotheses dramatically. Although we need extra time in the reshuffling phrase, we found the new decoder worked 4-5 times faster than the IBM decoder did for Model 3. The speed advantage was more evident for the structure-based model. Since the IBM decoder was not directly applicable to this model, it had to be coupled with the reshuffling algorithm anyway. Therefore the time reduction resulting from switching from the IBM decoder to the fast stack decoder was fully observed. The IBM+R decoder could spend hours on some long sentences, while the FSD+R decoder normally found a translation within 15 minutes.

Another advantage of the new decoding algorithm is its general applicability. Base model decoding plus reshuffling according to an advanced model provides a general framework for any complicated models.

5. CONCLUSIONS

The base model decoding plus reshuffling algorithm achieved performance comparable to the IBM stack decoder. It works much faster, and it is generally applicable to more complicated models.

6. REFERENCES

1. P. F. Brown, S. A. Della-Pietra, V. J Della-Pietra, and R. L. Mercer. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19(2):263–311, 1993.

2. C. Tillmann, S. Vogel, H. Ney, and A. Zubiaga. A DP-based Search Using Monotone Alignments in Statistical Translation. In *Proceedings of ACL/EACL'97*, pages 313–320, Madrid, Spain, 1997.

3. Y. Wang and A. Waibel. Modeling with Structures in Statistical Machine Translation. In *Proceedings of COLING-ACL '98*, Montréal, Canada, 1997.

4. A. L. Berger, P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, J. R. Gillett, J. D. Lafferty, R. L. Mercer, H. Printz, and L. Ures. *Language Translation Apparatus and Method Using Context-Based Translation Models*. United States Patent No. 5,510,981, 1996.

5. Y. Wang and A. Waibel. Decoding Algorithm in Statistical Machine Translation. In *Proceedings of ACL/EACL'97*, pages 366–372, Madrid, Spain, 1997.

6. S. Niessen, S. Vogel, H. Ney, and C. Tillmann. Modeling with Structures in Statistical Machine Translation. In *Proceedings of COLING-ACL '98*, Montréal, Canada, 1997.