Fast Generation of Translation Forest for Large-Scale SMT Discriminative Training

Xinyan Xiao, Yang Liu, Qun Liu, and **Shouxun Lin**
Key Laboratory of Intelligent Information Processing
Institute of Computing Technology
Chinese Academy of Sciences
{xiaoxinyan,yliu,liuqun,sxlin}@ict.ac.cn

Abstract

Although discriminative training guarantees to improve statistical machine translation by incorporating a large amount of overlapping features, it is hard to scale up to large data due to decoding complexity. We propose a new algorithm to generate translation forest of training data in linear time with the help of word alignment. Our algorithm also alleviates the oracle selection problem by ensuring that a forest always contains derivations that exactly yield the reference translation. With millions of features trained on 519K sentences in 0.03 second per sentence, our system achieves significant improvement by 0.84 BLEU over the baseline system on the NIST Chinese-English test sets.

1 Introduction

Discriminative model (Och and Ney, 2002) can easily incorporate non-independent and overlapping features, and has been dominating the research field of statistical machine translation (SMT) in the last decade. Recent work have shown that SMT benefits a lot from exploiting large amount of features (Liang et al., 2006; Tillmann and Zhang, 2006; Watanabe et al., 2007; Blunsom et al., 2008; Chiang et al., 2009). However, the training of the large number of features was always restricted in fairly small data sets. Some systems limit the number of training examples, while others use short sentences to maintain efficiency.

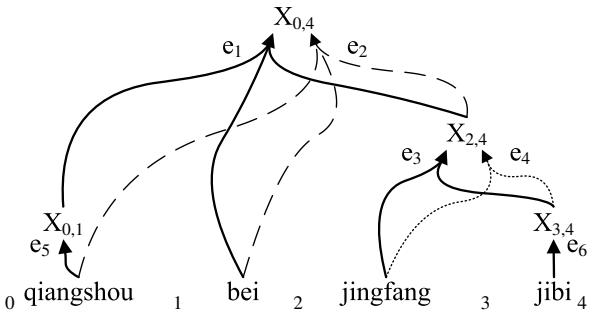
Overfitting problem often comes when training many features on a small data (Watanabe et al., 2007; Chiang et al., 2009). Obviously, using much more data can alleviate such problem. Furthermore, large data also enables us to globally train millions of sparse lexical features which offer accurate clues for SMT. Despite these advantages, to the best of our knowledge, no previous discriminative training paradigms scale up to use a large amount of training data. The main obstacle comes from the complexity of packed forests or n-best lists generation which requires to search through all possible translations of each training example, which is computationally prohibitive in practice for SMT.

To make normalization efficient, contrastive estimation (Smith and Eisner, 2005; Poon et al., 2009) introduce neighborhood for unsupervised log-linear model, and has presented positive results in various tasks. Motivated by these work, we use a *translation forest* (Section 3) which contains both "*reference*" derivations that potentially yield the reference translation and also *neighboring* "*non-reference*" derivations that fail to produce the reference translation.[1] However, the complexity of generating this translation forest is up to $\mathcal{O}(n^6)$, because we still need biparsing to create the reference derivations.

Consequently, we propose a method to fast generate a subset of the forest. The key idea (Section 4) is to initialize a reference derivation tree with maximum score by the help of word alignment, and then traverse the tree to generate the subset forest in linear time. Besides the efficiency improvement, such a forest allows us to train the model without resort-

[1] Exactly, there are no reference derivations, since derivation is a latent variable in SMT. We call them reference derivation just for convenience.

Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, pages 880–888,
Edinburgh, Scotland, UK, July 27–31, 2011.



hyper-edge		rule
e_1	r_1	$X \Rightarrow \langle X_1$ *bei* X_2, X_1 was $X_2 \rangle$
e_2	r_2	$X \Rightarrow \langle$ *qiangshou bei* X_1, the gunman was $X_1 \rangle$
e_3	r_3	$X \Rightarrow \langle$ *jingfang* X_1, X_1 by the police$\rangle$
e_4	r_4	$X \Rightarrow \langle$ *jingfang* X_1, police $X_1 \rangle$
e_5	r_5	$X \Rightarrow \langle$ *qiangshou*, the gunman$\rangle$
e_6	r_6	$X \Rightarrow \langle$ *jibi*, shot dead$\rangle$

Figure 1: A translation forest which is the running example throughout this paper. The reference translation is "*the gunman was killed by the police*". (1) Solid hyperedges denote a "reference" derivation tree t_1 which exactly yields the reference translation. (2) Replacing e_3 in t_1 with e_4 results a competing non-reference derivation t_2, which fails to swap the order of $X_{3,4}$. (3) Removing e_1 and e_5 in t_1 and adding e_2 leads to another reference derivation t_3. Generally, this is done by deleting a node $X_{0,1}$.

ing to constructing the oracle reference (Liang et al., 2006; Watanabe et al., 2007; Chiang et al., 2009), which is non-trivial for SMT and needs to be determined experimentally. Given such forests, we globally learn a log-linear model using stochastic gradient descend (Section 5). Overall, both the generation of forests and the training algorithm are scalable, enabling us to train millions of features on large-scale data.

To show the effect of our framework, we globally train millions of word level context features motivated by word sense disambiguation (Chan et al., 2007) together with the features used in traditional SMT system (Section 6). Training on 519K sentence pairs in 0.03 seconds per sentence, we achieve significantly improvement over the traditional pipeline by 0.84 BLEU.

2 Synchronous Context Free Grammar

We work on synchronous context free grammar (SCFG) (Chiang, 2007) based translation. The elementary structures in an SCFG are rewrite rules of the form:

$$X \Rightarrow \langle \gamma, \alpha \rangle$$

where γ and α are strings of terminals and nonterminals. We call γ and α as the source side and the target side of rule respectively. Here a rule means a phrase translation (Koehn et al., 2003) or a translation pair that contains nonterminals.

We call a sequence of translation steps as a *derivation*. In context of SCFG, a derivation is a sequence of SCFG rules $\{r_i\}$. *Translation forest* (Mi et al., 2008; Li and Eisner, 2009) is a compact representation of all the derivations for a given sentence under an SCFG (see Figure 1). A *tree* t in the forest corresponds to a derivation. In our paper, tree means the same as derivation.

More formally, a **forest** is a pair $\langle V, E \rangle$, where V is the set of **nodes**, E is the set of **hyperedge**. For a given source sentence $\mathbf{f} = f_1^n$, Each node $v \in V$ is in the form $X_{i,j}$, which denotes the recognition of nonterminal X spanning the substring from the i through j (that is $f_{i+1}...f_j$). Each hyperedge $e \in E$ connects a set of antecedent to a single consequent node and corresponds to an SCFG rule $r(e)$.

3 Our Translation Forest

We use a translation forest that contains both "*reference*" derivations that potentially yield the reference translation and also some neighboring "*non-reference*" derivations that fail to produce the reference translation. Therefore, our forest only represents some of the derivations for a sentence given an SCFG rule table. The motivation of using such a forest is efficiency. However, since this space contains both "good" and "bad" translations, it still provides evidences for discriminative training.

First see the example in Figure 1. The derivation tree t_1 represented by solid hyperedges is a reference derivation. We can construct a non-reference derivation by making small change to t_1. By replacing the e_3 of t_1 with e_4, we obtain a non-reference deriva-

tion tree t_2. Considering the rules in each derivation, the difference between t_1 and t_2 lies in r_3 and r_4. Although r_3 has a same source side with r_4, it produces a different translation. While r_3 provides a swapping translation, r_4 generates a monotone translation. Thus, the derivation t_2 fails to move the subject "police" to the behind of verb "shot dead", resulting a wrong translation "the gunman was police shot dead". Given such derivations, we hope that the discriminative model is capable to explain why should use a reordering rule in this context.

Generally, our forest contains all the reference derivations $\mathcal{RT}$ for a sentence given a rule table, and some neighboring non-reference derivations $\mathcal{NT}$, which can be defined from $\mathcal{RT}$.

More formally, we call two hyperedges e^1 and e^2 are **competing hyperedges**, if their corresponding rules $r(e^1) = \langle \gamma^1, \alpha^1 \rangle$ and $r(e^2) = \langle \gamma^2, \alpha^2 \rangle$:

$$\gamma^1 = \gamma^2 \wedge \alpha^1 \neq \alpha^2 \quad (1)$$

This means they give different translations for a same source side. We use $C(e)$ to represent the set of competing hyperedges of e.

Two derivations $t^1 = \langle V^1, E^1 \rangle$ and $t^2 = \langle V^2, E^2 \rangle$ are **competing derivations** if there exists $e^1 \in E^1$ and $e^2 \in E^2$: [2]

$$V^1 = V^2 \wedge E^1 - e^1 = E^2 - e^2 \\ \wedge e^2 \in C(e^1) \quad (2)$$

In other words, derivations t^1 and t^2 only differ in e^1 and e^2, and these two hyperedges are competing hyperedges. We use $C(t)$ to represent the set of competing derivations of tree t, and $C(t,e)$ to represent the set of competing derivations of t if the competition occurs in hyperedge e in t.

Given a rule table, the set of reference derivations $\mathcal{RT}$ for a sentence is determined. Then, the set of non-reference derivations $\mathcal{NT}$ can be defined from $\mathcal{RT}$:

$$\cup_{t \in \mathcal{RT}} C(t) \quad (3)$$

Overall, our forest is the compact representation of $\mathcal{RT}$ and $\mathcal{NT}$.

[2]The definition of derivation tree is similar to forest, except that the tree contains exactly one tree while forest contains exponentially trees. In tree, the hyperedge degrades to edge.

Algorithm 1 Forest Generation

```
1:  procedure GENERATE(t)
2:      list ← t
3:      for v ∈ t in post order do
4:          e ← incoming edge of v
5:          append C(t, e) to list;
6:          for u ∈ child(v) from left to right do
7:              t_n ← OPERATE(t, u)
8:              if t_n ≠ t then
9:                  append t_n to list
10:                 for e' ∈ t_n ∧ e' ∉ t do
11:                     append C(t_n,e') to list
12:                 if SCORE(t) < SCORE(t_n) then
13:                     t ← t_n
14:     return t,list
```

4 Fast Generation

It is still slow to calculate the entire forest defined in Section 3, therefore we use a greedy decoding for fast generating a subset of the forest. Starting form a reference derivation, we try to slightly change the derivation into a new reference derivation. During this process, we collect the competing derivations of reference derivations. We describe the details of local operators for changing a derivation in section 4.1, and then introduce the creation of initial reference derivation with max score in Section 4.2.

For example, given derivation t_1, we delete the node $X_{0,1}$ and the related hyperedge e_1 and e_5. Fixing the other nodes and edges, we try to add a new edge e_2 to create a new reference translation. In this case, if rule r_2 really exists in our rule table, we get a new reference derivation t_3. After constructing t_3, we first collect the new tree and $C(t_3, e_2)$. Then, we will move to t_3, if the score of t_3 is higher than t_2. Notably, if r_2 does not exist in the rule table, we fail to create a new reference derivation. In such case, we keep the origin derivation unchanged.

Algorithm 1 shows the process of generation.[3] The input is a reference derivation t, and the output is a new derivation and the generated derivations.

[3]For simplicity, we list all the trees, and do not compress them into a forest in practice. It is straight to extent the algorithm to get a compact forest for those generated derivations. Actually, instead of storing the derivations, we call the generate function twice to calculate gradient of log-linear model.

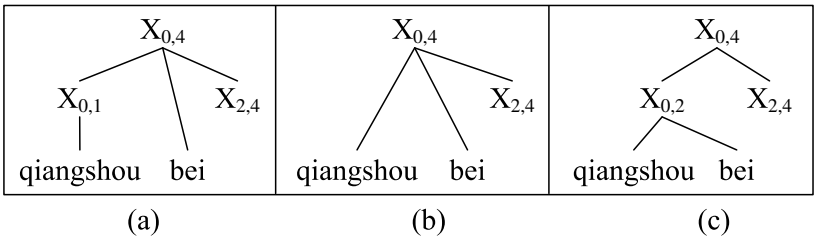


Figure 2: Lexicalize and generalize operators over t_1 (part) in Figure 1. Although here only shows the nodes, we also need to change relative edges actually. (1) Applying lexicalize operator on the non-terminal node $X_{0,1}$ in (a) results a new derivation shown in (b). (2) When visiting *bei* in (b), the generalize operator changes the derivation into (c).

The $list$ used for storing forest is initialized with the input tree (line 2). We visit the nodes in t in post-order (line 3). For each node v, we first append the competing derivations $C(t,e)$ to $list$, where e is incoming edge of v (lines 4-5). Then, we apply operators on the child nodes of v from left to right (lines 6-13). The operators returns a reference derivation t_n (line 7). If it is new (line 8), we collect both the t_n (line 9), and also the competing derivations $C(t_n, e')$ of the new derivation on those edges e' which only occur in the new derivation (lines 10-11). Finally, if the new derivation has a larger score, we will replace the origin derivation with new one (lines 12-13).

Although there is a two-level loop for visiting nodes (line 3 and 6), each node is visited only one time in the inner loops. Thus, the complexity is linear with the number of nodes $\#node$. Considering that the number of source word (also leaf node here) is less than the total number of nodes and is more than $\lceil(\#node+1)/2\rceil$, the time complexity of the process is also linear with the number of source word.

4.1 Lexicalize and Generalize

The function OPERATE in Algorithm 1 uses two operators to change a node: *lexicalize* and *generalize*. Figure 2 shows the effects of the two operators. The **lexicalize** operator works on nonterminal nodes. It moves away a nonterminal node and attaches the children of current node to its parent. In Figure 2(b), the node $X_{0,1}$ is deleted, requiring a more lexicalized rule to be applied to the parent node $X_{0,4}$ (one more terminal in the source side). We constrain the lexicalize operator to apply on pre-terminal nodes whose children are all terminal nodes. In contrast, the **generalize** operator works on terminal nodes and inserts a nonterminal node between current node and its parent node. This operator generalizes over the continuous terminal sibling nodes left to the current node (including the current node). Generalizing the node *bei* in Figure 2(b) results Figure 2(c). A new node $X_{0,2}$ is inserted as the parent of node *qiangshou* and node *bei*.

Notably, there are two steps when apply an operator. Suppose we want to lexicalize the node $X_{0,1}$ in t_1 of Figure 1, we first delete the node $X_{0,1}$ and related edge e_1 and e_5, then we try to add the new edge e_2. Since rule table is fixed, the second step is a process of decoding. Therefore, sometimes we may fail to create a new reference derivation (like r_2 may not exist in the rule table). In such case, we keep the origin derivation unchanged.

The changes made by the two operators are local. Considering the change of rules, the lexicalize operator deletes two rules and adds one new rule, while the generalize operator deletes one rule and adds two new rules. Such local changes provide us with a way to incrementally calculate the scores of new derivations. We use this method motivated by Gibbs Sampler (Blunsom et al., 2009) which has been used for efficiently learning rules. The different lies in that we use the operator for decoding where the rule table is fixing.

4.2 Initialize a Reference Derivation

The generation starts from an initial reference derivation with max score. This requires bi-parsing (Dyer, 2010) over the source sentence $\mathbf{f}$ and the reference translation $\mathbf{e}$. In practice, we may face three problems.

First is efficiency problem. Exhaustive search over the space under SCFG requires $\mathcal{O}(|f|^3|e|^3)$.

To parse quickly, we only visit the tight consistent (Zhang et al., 2008) bi-spans with the help of word alignment $\mathbf{a}$. Only visiting tight consistent spans greatly speeds up bi-parsing. Besides efficiency, adoption of this constraint receives support from the fact that heuristic SCFG rule extraction only extracts tight consistent initial phrases (Chiang, 2007).

Second is degenerate problem. If we only use the features as traditional SCFG systems, the bi-parsing may end with a derivation consists of some giant rules or rules with rare source/target sides, which is called degenerate solution (DeNero et al., 2006). That is because the translation rules with rare source/target sides always receive a very high translation probability. We add a prior score $log(\#rule)$ for each rule, where $\#rule$ is the number of occurrence of a rule, to reward frequent reusable rules and derivations with more rules.

Finally, we may fail to create reference derivations due to the limitation in rule extraction. We create minimum trees for $(\mathbf{f}, \mathbf{e}, \mathbf{a})$ using shift-reduce (Zhang et al., 2008). Some minimum rules in the trees may be illegal according to the definition of Chiang (2007). We also add these rules to the rule table, so as to make sure every sentence is reachable given the rule table. A source sentence is **reachable** given a rule table if reference derivations exists. We refer these rules as **added rules**. However, this may introduce rules with more than two variables and increase the complexity of bi-parsing. To tackle this problem, we initialize the chart with minimum parallel tree from the Zhang et al. (2008) algorithm, ensuring that the bi-parsing has at least one path to create a reference derivation. Then we only need to consider the traditional rules during bi-parsing.

5 Training

We use the forest to train a log-linear model with a latent variable as describe in Blunsom et al.(2008). The probability $p(\mathbf{e}|\mathbf{f})$ is the sum over all possible derivations:

$$p(\mathbf{e}|\mathbf{f}) = \sum_{\mathbf{t} \in \triangle(\mathbf{e},\mathbf{f})} p(\mathbf{t}, \mathbf{e}|\mathbf{f}) \quad (4)$$

where $\triangle(\mathbf{e}, \mathbf{f})$ is the set of all possible derivations that translate $\mathbf{f}$ into $\mathbf{e}$ and $\mathbf{t}$ is one such derivation.[4]

[4]Although the derivation is typically represent as $\mathbf{d}$, we denotes it by t since our paper use tree to represent derivation.

Algorithm 2 Training

procedure TRAIN($\mathcal{S}$)
  Training Data $\mathcal{S} = \{\mathbf{f}^n, \mathbf{e}^n, \mathbf{a}^n\}_{n=1}^N$
  Derivations $\mathcal{T} = \{\}_{n=1}^N$
  for $n = 1$ **to** N **do**
    $t^n \leftarrow$ INITIAL($\mathbf{f}^n, \mathbf{e}^n, \mathbf{a}^n$)
  $i \leftarrow 0$
  for $m = 0$ to M **do**
    for $n = 0$ to N **do**
      $\eta \leftarrow$ LEARNRATE(i)
      $(\Delta L(\mathbf{w}^i, t^n), t^n) \leftarrow$GENERATE($t^n$)
      $\mathbf{w}^i \leftarrow \mathbf{w}^i + \eta \times \Delta L(\mathbf{w}^i, t^n)$
      $i \leftarrow i + 1$
  return $\frac{\sum_{i=1}^{MN} \mathbf{w}^i}{MN}$

This model defines the conditional probability of a derivation $\mathbf{t}$ and the corresponding translation $\mathbf{e}$ given a source sentence $\mathbf{f}$ as:

$$p(\mathbf{t}, \mathbf{e}|\mathbf{f}) = \frac{\exp \sum_i \lambda_i h_i(\mathbf{t}, \mathbf{e}, \mathbf{f})}{Z(\mathbf{f})} \quad (5)$$

where the partition function is

$$Z(\mathbf{f}) = \sum_{\mathbf{e}} \sum_{\mathbf{t} \in \triangle(\mathbf{e},\mathbf{f})} \exp \sum_i \lambda_i h_i(\mathbf{t}, \mathbf{e}, \mathbf{f}) \quad (6)$$

The partition function is approximated by our forest, which is labeled as $\tilde{Z}(\mathbf{f})$, and the derivations that produce reference translation is approximated by reference derivations in $\tilde{Z}(\mathbf{f})$.

We estimate the parameters in log-linear model using maximum a posteriori (MAP) estimator. It maximizes the likelihood of the bilingual corpus $\mathcal{S} = \{\mathbf{f}^n, \mathbf{e}^n\}_{n=1}^N$, penalized using a gaussian prior (L2 norm) with the probability density function $p_0(\lambda_i) \propto \exp(-\lambda_i^2/2\sigma^2)$. We set σ^2 to 1.0 in our experiments. This results in the following gradient:

$$\frac{\partial \mathcal{L}}{\partial \lambda_i} = E_{p(\mathbf{t}|\mathbf{e},\mathbf{f})}[h_i] - E_{p(\mathbf{e}|\mathbf{f})}[h_i] - \frac{\lambda_i}{\sigma^2} \quad (7)$$

We use an online learning algorithm to train the parameters. We implement stochastic gradient descent (SGD) recommended by Bottou.[5] The dynamic learning rate we use is $\frac{N}{(i+i_0)}$, where N is the

[5]http://leon.bottou.org/projects/sgd

number of training example, i is the training iteration, and i_0 is a constant number used to get a initial learning rate, which is determined by calibration.

Algorithm 2 shows the entire process. We first create an initial reference derivation for every training examples using bi-parsing (lines 4-5), and then online learn the parameters using SGD (lines 6-12). We use the GENERATE function to calculate the gradient. In practice, instead of storing all the derivations in a list, we traverse the tree twice. The first time is calculating the partition function, and the second time calculates the gradient normalized by partition function. During training, we also change the derivations (line 10). When training is finished after M epochs, the algorithm returns an averaged weight vector (Collins, 2002) to avoid overfitting (line 13). We use a development set to select total epoch m, which is set as $M = 5$ in our experiments.

6 Experiments

Our method is able to train a large number of features on large data. We use a set of word context features motivated by word sense disambiguation (Chan et al., 2007) to test scalability. A word level context feature is a triple (f, e, f_{+1}), which counts the number of time that f is aligned to e and f_{+1} occurs to the right of f. Triple (f, e, f_{-1}) is similar except that f_{-1} locates to the left of f. We retain word alignment information in the extracted rules to exploit such features. To demonstrate the importance of scaling up the size of training data and the effect of our method, we compare three types of training configurations which differ in the size of features and data.

MERT. We use MERT (Och, 2003) to training 8 features on a small data. The 8 features is the same as Chiang (2007) including 4 rule scores (direct and reverse translation scores; direct and reverse lexical translation scores); 1 target side language model score; 3 penalties for word counts, extracted rules and glue rule. Actually, traditional pipeline often uses such configuration.

Perceptron. We also learn thousands of context word features together with the 8 traditional features on a small data using perceptron. Following (Chiang et al., 2009), we only use 100 most frequent words for word context feature. This setting use CKY decoder to generate n-best lists for training. The complexity of CKY decoding limits the training data into a small size. We fix the 8 traditional feature weights as MERT to get a comparable results as MERT.

	TRAIN	RTRAIN	DEV	TEST
#Sent.	519,359	186,810	878	3,789
#Word	8.6M	1.3M	23K	105K
Avg. Len.	16.5	7.3	26.4	28.0
Lon. Len.	99	95	77	116

Table 1: Corpus statistics of Chinese side, where Sent., Avg., Lon., and Len. are short for sentence, longest, average, and length respectively. RTRAIN denotes the reachable (given rule table without added rules) subset of TRAIN data.

Our Method. Finally, we use our method to train millions of features on large data. The use of large data promises us to use full vocabulary of training data for the context word features, which results millions of fully lexicalized context features. During decoding, when a context feature does not exit, we simply ignore it. The weights of 8 traditional features are fixed the same as MERT also. We fix these weights because the translation feature weights fluctuate intensely during online learning. The main reason may come from the degeneration solution mentioned in Section 4.2, where rare rules with very high translation probability are selected as the reference derivations. Another reason could be the fact that translation features are dense intensify the fluctuation. We leave learning without fixing the 8 feature weights to future work.

6.1 Data

We focus on the Chinese-to-English translation task in this paper. The bilingual corpus we use contains $519,359$ sentence pairs, with an average length of 16.5 in source side and 20.3 in target side, where $186,810$ sentence pairs (36%) are reachable (without added rules in Section 4.2). The monolingual data includes the Xinhua portion of the GIGAWORD corpus, which contains 238M English words. We use the NIST evaluation sets of 2002 (MT02) as our development set, and sets of MT03/MT04/MT05 as test sets. Table 2 shows the statistics of all bilingual corpus.

We use GIZA++ (Och and Ney, 2003) to perform

System	#DATA	#FEAT	MT03	MT04	MT05	ALL
MERT	878	8	33.03	35.12	32.32	33.85
Perceptron	878	2.4K	32.89	34.88	32.55	33.76
Our Method	187K	2.0M	33.64	35.48	32.91*	34.41*
	519K	13.9M	34.19*	35.72*	33.09*	34.69*
Improvement over MERT			+1.16	+0.60	+0.77	+0.84

Table 2: Effect of our method comparing with MERT and perceptron in terms of BLEU. We also compare our fast generation method with different data (only reachable or full data). #Data is the size of data for training the feature weights. * means significantly (Koehn, 2004) better than *MERT* ($p < 0.01$).

word alignment in both directions, and grow-diag-final-and (Koehn et al., 2003) to generate symmetric word alignment. We extract SCFG rules as described in Chiang (2007) and also added rules (Section 4.2). Our algorithm runs on the entire training data, which requires to load all the rules into the memory. To fit within memory, we cut off those composed rules which only happen once in the training data. Here a composed rule is a rule that can be produced by any other extracted rules. A 4-grams language model is trained by the SRILM toolkit (Stolcke, 2002). Case-insensitive NIST BLEU4 (Papineni et al., 2002) is used to measure translation performance.

The training data comes from a subset of the LDC data including LDC2002E18, LDC2003E07, LDC2003E14, Hansards portion of LDC2004T07, LDC2004T08 and LDC2005T06. Since the rule table of the entire data is too large to be loaded to the memory (even drop one-count rules), we remove many sentence pairs to create a much smaller data yet having a comparable performance with the entire data. The intuition lies in that if most of the source words of a sentence need to be translated by the added rules, then the word alignment may be highly crossed and the sentence may be useless. We create minimum rules from a sentence pair, and count the number of source words in those minimum rules that are added rules. For example, suppose the result minimum rules of a sentence contain r_3 which is an added rule, then we count 1 time for the sentence. If the number of such source word is more than 10% of the total number, we will drop the sentence pair.

We compare the performances of MERT setting on three bilingual data: the entire data that contains 42.3M Chinese and 48.2M English words; 519K data that contains 8.6M Chinese and 10.6M English words; FBIS (LDC2003E14) parts that contains 6.9M Chinese and 9.1M English words. They produce 33.11/32.32/30.47 BLEU tested on MT05 respectively. The performance of 519K data is comparable with that of entire data, and much higher than that of FBIS data.

6.2 Result

Table 3 shows the performance of the three different training configurations. The training of MERT and perceptron run on MT02. For our method, we compare two different training sets: one is trained on all 519K sentence pairs, the other only uses 186K reachable sentences.

Although the perceptron system exploits 2.4K features, it fails to produce stable improvements over MERT. The reason may come from overfitting, since the training data for perceptron contains only 878 sentences. However, when use our method to learn the word context feature on the 519K data, we significantly improve the performance by 0.84 points on the entire test sets (ALL). The improvements range from 0.60 to 1.16 points on MT03-05. Because we use the full vocabulary, the number of features increased into 13.9 millions, which is impractical to be trained on the small development set. These results confirm the necessity of exploiting more features and learning the parameters on large data. Meanwhile, such results also demonstrate that we can benefits from the forest generated by our fast method instead of traditional CKY algorithm.

Not surprisingly, the improvements are smaller when only use 186K reachable sentences. Sometimes we even fail to gain significant improvement. This verifies our motivation to guarantee all sentence

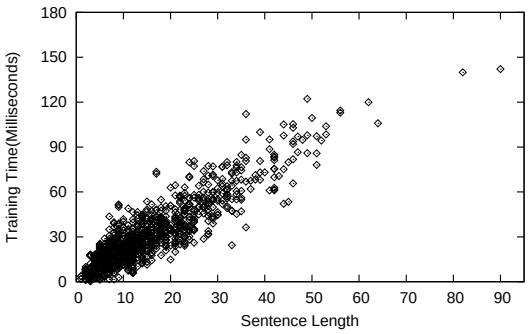


Figure 3: Plot of training times (including forest generation and SGD training) versus sentence length. We randomly select 1000 sentence from the 519K data for plotting.

are reachable, so as to use all training data.

6.3 Speed

How about the speed of our framework? Our method learns in 32 mlliseconds/sentence. Figure 3 shows training times (including forest generation and SGD training) versus sentence length. The plot confirms that our training algorithm scales linearly. If we use n-best lists which generated by CKY decoder as MERT, it takes about 3105 milliseconds/sentence for producing 100-best lists. Our method accelerates the speed about 97 times (even though we search twice to calculate the gradient). This shows the efficiency of our method.

The procedure of training includes two steps. (1) Bi-parsing to initialize a reference derivation with max score. (2) Training procedure which generates a set of derivations to calculate the gradient and update parameters. Step (1) only runs once. The average time of processing a sentence for each step is about 9.5 milliseconds and 30.2 milliseconds respectively.

For simplicity we do not compress the generated derivations into forests, therefore the size of resulting derivations is fairly small, which is about 265.8 for each sentence on average, where 6.1 of them are reference derivations. Furthermore, we use lexicalize operator more often than generalize operator (the ration between them is 1.5 to 1). Lexicalize operator is used more frequently mainly dues to that the reference derivations are initialized with reusable (thus small) rules.

7 Related Work

Minimum error rate training (Och, 2003) is perhaps the most popular discriminative training for SMT. However, it fails to scale to large number of features. Researchers have propose many learning algorithms to train many features: perceptron (Shen et al., 2004; Liang et al., 2006), minimum risk (Smith and Eisner, 2006; Li et al., 2009), MIRA (Watanabe et al., 2007; Chiang et al., 2009), gradient descent (Blunsom et al., 2008; Blunsom and Osborne, 2008). The complexity of n-best lists or packed forests generation hamper these algorithms to scale to a large amount of data.

For efficiency, we only use neighboring derivations for training. Such motivation is same as contrastive estimation (Smith and Eisner, 2005; Poon et al., 2009). The difference lies in that the previous work actually care about their latent variables (pos tags, segmentation, dependency trees, etc), while we are only interested in their marginal distribution. Furthermore, we focus on how to fast generate translation forest for training.

The local operators lexicalize/generalize are use for greedy decoding. The idea is related to "pegging" algorithm (Brown et al., 1993) and greedy decoding (Germann et al., 2001). Such types of local operators are also used in Gibbs sampler for synchronous grammar induction (Blunsom et al., 2009; Cohn and Blunsom, 2009).

8 Conclusion and Future Work

We have presented a fast generation algorithm for translation forest which contains both reference derivations and neighboring non-reference derivations for large-scale SMT discriminative training. We have achieved significantly improvement of 0.84 BLEU by incorporate 13.9M feature trained on 519K data in 0.03 second per sentence.

In this paper, we define the forest based on competing derivations which only differ in one rule. There may be better classes of forest that can produce a better performance. It's interesting to modify the definition of forest, and use more local operators to increase the size of forest. Furthermore, since the generation of forests is quite general, it's straight to

apply our forest on other learning algorithms. Finally, we hope to exploit more features such as reordering features and syntactic features so as to further improve the performance.

Acknowledgement

We would like to thank Yifan He, Xianhua Li, Daqi Zheng, and the anonymous reviewers for their insightful comments. The authors were supported by National Natural Science Foundation of China Contracts 60736014, 60873167, and 60903138.

References

Phil Blunsom and Miles Osborne. 2008. Probabilistic inference for machine translation. In *Proc. of EMNLP 2008*.

Phil Blunsom, Trevor Cohn, and Miles Osborne. 2008. A discriminative latent variable model for statistical machine translation. In *Proc. of ACL-08*.

Phil Blunsom, Trevor Cohn, Chris Dyer, and Miles Osborne. 2009. A gibbs sampler for phrasal synchronous grammar induction. In *Proc. of ACL 2009*.

Peter F. Brown, Vincent J.Della Pietra, Stephen A. Della Pietra, and Robert. L. Mercer. 1993. The mathematics of statistical machine translation. *Computational Linguistics*, 19:263–311.

Yee Seng Chan, Hwee Tou Ng, and David Chiang. 2007. Word sense disambiguation improves statistical machine translation. In *Proc. of ACL 2007*, pages 33–40.

David Chiang, Kevin Knight, and Wei Wang. 2009. 11,001 new features for statistical machine translation. In *Proc. of NAACL 2009*.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

Trevor Cohn and Phil Blunsom. 2009. A Bayesian model of syntax-directed tree to string grammar induction. In *Proc. of EMNLP 2009*.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP 2002*.

John DeNero, Dan Gillick, James Zhang, and Dan Klein. 2006. Why generative phrase models underperform surface heuristics. In *Proc. of the HLT-NAACL 2006 Workshop on SMT*.

Chris Dyer. 2010. Two monolingual parses are better than one (synchronous parse). In *Proc. of NAACL 2010*.

Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *Proc. of ACL 2001*.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proc. of HLT-NAACL 2003*.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proc. of EMNLP 2004*.

Zhifei Li and Jason Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proc. of EMNLP 2009*.

Zhifei Li, Jason Eisner, and Sanjeev Khudanpur. 2009. Variational decoding for statistical machine translation. In *Proc. of ACL 2009*.

Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. 2006. An end-to-end discriminative approach to machine translation. In *Proc. of ACL 2006*.

Haitao Mi, Liang Huang, and Qun Liu. 2008. Forest-based translation. In *Proc. of ACL 2008*.

Franz J. Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proc. of ACL 2002*.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proc. of ACL 2003*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proc. of ACL 2002*.

Hoifung Poon, Colin Cherry, and Kristina Toutanova. 2009. Unsupervised morphological segmentation with log-linear models. In *Proc. of NAACL 2009*.

Libin Shen, Anoop Sarkar, and Franz Josef Och. 2004. Discriminative reranking for machine translation. In *Proc. of NAACL 2004*.

Noah A. Smith and Jason Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *Proc. of ACL 2005*.

David A. Smith and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proc. of COLING/ACL 2006*.

Andreas Stolcke. 2002. Srilm – an extensible language modeling toolkit. In *Proc. of ICSLP 2002*.

Christoph Tillmann and Tong Zhang. 2006. A discriminative global training algorithm for statistical mt. In *Proc. of ACL 2006*.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2007. Online large-margin training for statistical machine translation. In *Proc. of EMNLP-CoNLL 2007*.

Hao Zhang, Daniel Gildea, and David Chiang. 2008. Extracting synchronous grammar rules from word-level alignments in linear time. In *Proc. of Coling 2008*.