Preordering has proven useful in improving the translation quality of statistical machine translation (SMT), especially for language pairs with different syntax. The top-down bracketing transduction grammar (BTG)-based preordering method (Nakagawa 2015) has achieved a state-of-the-art performance since it relies on aligned parallel text only and does not require any linguistic annotations. Although this online learning algorithm adopted is efficient and effective, it is very susceptible to alignment errors. In a production environment, in particular, such a preorderer is commonly trained on noisy word alignments obtained using an automatic word aligner, resulting in a worse performance compared to those trained on manually annotated datasets. In order to achieve better preordering using automatically aligned datasets, this paper seeks to improve the top-down BTG-based preordering method using various parameter mixing techniques to increase the accuracy of the preorderer and speed up training via parallelisation. The parameters mixing methods and the original online training method (Nakagawa 2015) were empirically compared, and the experimental results show that such parallel parameter averaging methods can dramatically reduce the training time and improve the quality of preordering.

**Key Words:** Preordering, Machine Translation, Parallelisation

1 Introduction

Preordering is an essential preprocessing step in statistical machine translation (Koehn, Och, and Marcu 2003), especially for language pairs with different syntax. When translating from a subject-verb-object (SVO) language such as English to a subject-object-verb (SOV) language such as Japanese, the English verb in a sentence often moves to the end of the Japanese sentence/clause. Phrase-based SMT (PB-SMT) exhibits lower BLEU scores on distant language pairs, like English–Japanese, because the lexical reordering component used in PB-SMT is incapable of performing long-distance reorderings (Sudoh, Duh, Tsukada, Hirao, and Nagata 2010) owing to different word orders.

To cope with this problem, previous studies have investigated reordering before word align-
ment as it is in the preprocessing phase of SMT, called preordering. Preordering has been proven to improve the translation of language pairs with different word orders (Xia and McCord 2004; Collins, Koehn, and Kučerová 2005) as it reduces the complexity of decoding due to the source words being permuted in the word order of the target language. The majority of those works tend to employ a syntactic parser for the source language, and other methods only rely on word-aligned parallel text. DeNero and Uszkoreit (2011) showed that it is possible to learn some reordering rules from source sentences, after which Neubig, Watanabe, and Mori (2012) proposed to learn a discriminative parser (also called preorderer) directly for preordering by treating hidden synchronous parse trees as latent variables. The training time of the parser substantially increases if the data contain many long sentences because of the high complexity \(O(n^5)\) of the used algorithm, i.e., the bottom-up CYK algorithm. Nakagawa (2015) adopted the top-down parsing algorithm to improve the parser efficiency \(O(kn^2)\) for a beam width of \(k\) and sentence length \(n\), but training such a preorderer on large datasets is still very time-consuming.

Regarding the quality of the training data, Nakagawa’s top-down BTG-based preordering method learns the reordering model using the general framework of large-margin online structured prediction (Crammer, Dekel, Keshet, Shalev-Shwartz, and Singer 2006; Watanabe, Suzuki, Tsukada, and Isozaki 2007). To train an accurate preorderer, high quality word alignments are essential but given the expensive cost of manual annotation, production environment often produce word alignments automatically using unsupervised word aligners. The top-down BTG-based preordering method, which is using such an online training algorithm, suffers from small mistakes in training examples. The resulting preorderer trained on large automatically aligned datasets exhibits lower reordering scores.

The online pairwise updating strategy used in the top-down BTG-based preordering method is not suitable for training on automatically aligned datasets, given its inability to deal with massive, noisy alignment errors. A large body of research on batch learning and ensemble learning has been empirically observed to be helpful in handling noisy training data for many NLP tasks (Breiman 1996; Freund and Schapire 1999; Džeroski and Ženko 2004; Rokach 2010; Li, Zhang, Chen, and Smola 2014). These methods mainly fall into two categories: inner parameter mixing methods and outer parameter mixing methods. The inner parameter mixing methods perform parameter mixing on small batches, called mini-batch training in machine learning (Freund and Schapire 1999; Li et al. 2014). The outer parameter mixing methods are part of the framework of distributed training in machine learning (McDonald, Hall, and Mann 2010), or ensemble learning in neural network research (Naftaly, Intrator, and Horn 1997; Dietterich 2000). A small number of studies have concentrated on online to parallel conversions, despite the simplicity of using
parameter mixing techniques to improve the top-down BTG-based preordering method.

This paper investigates the training of a top-down BTG-based preorderer using parameter averaging techniques. The training algorithm was parallelised using bootstrap aggregating with several techniques (mini-batch, distributed, iterative distributed and k-best list), and these methods were empirically compared with the original online method used by Nakagawa (2015). The experimental results in Section 6 show that the proposed methods, which were more robust and stable, surpassed that of (Nakagawa 2015) when using automatically aligned datasets.

2 Preordering

2.1 Background

Previous research has shown that preordering can improve SMT (Xia and McCord 2004; Neubig et al. 2012; Lerner and Petrov 2013; Goto, Utiyama, Sumita, and Kurohashi 2015; Nakagawa 2015) because preordering introduces a preprocessing step in which source words are permuted in a target-like order before the standard SMT pipeline. SMT with preordering involves two phases: reordering and translation. In the first phase (i.e., reordering), $E$ is transformed into $E'$ in the same word order of the target sentence $J$. In the second phase (i.e., translation), the permuted source sentence is translated into the target language using an SMT system (see Fig. 1). Preordering is always performed ahead of other processes in SMT pipelines, e.g., word alignment, tuning and decoding. The final goal of preordering is to reduce the computational complexity of the phrase-based decoding.

Most preordering methods depend on the source-side syntactic parser to perform a series of tree transformations. For example, Isozaki, Sudoh, Tsukada, and Duh (2010b) finalised the head of an English dependency parse tree to the last node so that the order of source words came to resemble a head-final target language (in this case, Japanese). Similary, Cai, Utiyama, Sumita, and Zhang (2014) employed a dependency parser to perform preordering for Chinese–English.

![Fig. 1 Preordering an English sentence into a Japanese word order.](image-url)
Other preordering methods need no syntactic information. For example, DeNero and Uszkoreit (2011) proposed to learn a joint model from a word-aligned parallel corpus for both reordering and translation using the source-side syntactic information to reorder the words. This method resembles the work of bracketing transduction grammar Wu (1997).

In the recent dominant paradigm, preordering has also aroused interest within neural machine translation (NMT) studies (Cho, van Merrienboer, Gulcehre, Bahdanau, Bougares, Schwenk, and Bengio 2014; Bahdanau, Cho, and Bengio 2014). For example, although (Miceli Barone and Attardi 2015) have proposed an RNN-based preordering model for SMT, (Du and Way 2017) found that preordering deteriorates the performance of NMT, but using preordering features such as reordered indices in word embedding has benefited NMT. Hence, studying the learning methods for preordering is still essential.

2.2 Top-down BTG-based Preordering

Inspired by DeNero and Uszkoreit (2011) and Neubig et al. (2012), Nakagawa (2015) proposed preordering using a top-down parser utilizing BTG (Wu 1997). BTG is a simplified case of synchronous context-free grammar. According to the definition of BTG, both the source and target sentences should share the same underlying structures, i.e., BTG parse trees, and the order of constituents may be different. A BTG parse tree may have two types of non-terminal nodes, straight ($S$) and inverted ($I$), and only one type of leaves, terminal ($T$). If $X$ and $Y$ denote the non-terminal and terminal symbols, respectively, the BTG rules that generate a parse tree are the following:

$$X \rightarrow \begin{cases} [X_1X_2], & \text{straight} \\ \langle X_1X_2 \rangle, & \text{inverted} \\ Y_E/Y_J, & \text{terminal}. \end{cases} \quad (1)$$

Straight represents the case where the order of the child nodes is kept. In inverted, the child nodes are permuted in reverse. Terminal stands for the case of termination.\(^1\) BTG provides a simple and effective way to represent permutations and perform word reordering.\(^2\) For example, the source and target sentences in Fig. 1 can be represented as a BTG parse tree in Fig. 2.\(^3\)

Consider the case of preordering, where the BTG parse tree becomes incomplete because of lack of target words. Such a partial BTG parse tree is sufficient to perform accurate reorderings

\(^1\) $Y_J$ can also be a phrase, or even $\varepsilon$.

\(^2\) There exist some permutations that BTG cannot represent, e.g., $X_2X_4X_1X_3$ to $X_1X_2X_3X_4$, see (Wu 1997).

\(^3\) Multiple BTG parse trees may lead to same word orders.

In Fig. 2, the non-terminals upon terminals were omitted for simplicity.
Fig. 2 Example of BTG parsing and two parser derivations: $D'$, system derivation with the highest model score; $D^*$, oracle derivation with the highest model score from the valid subset.

as it is interchangeable with a parser derivation in left-to-right top-down parsing. If $D$ denotes the parser derivation, which is a latent variable, in a BTG-based preordering framework (Neubig et al. 2012; Nakagawa 2015), the inference procedure is defined as follows:

$$
\tilde{E}' = \arg \max_{E' \in \hat{E}} P(E', D | E) \\
= \arg \max_{D \in \hat{D}} P(E'|D, E)P(D | E). 
$$

(2) (3)

where $\hat{E}$ stands for all possible permutations and $\tilde{E}'$ is the best one given the target word order. Only one deterministic derivation for a particular BTG tree exists, thus, the first part of formula 3 can be eliminated. Therefore, the task of preordering can be solved as monolingual parsing with the extension of reordering operations. For source sentence $E$, finding the best reordering is equivalent to finding an underlying parser derivation $D$ licensed by BTG. Nakagawa (2015) assigned a score for a particular derivation with the score function in a linear form:

$$
P(D|E, w) = \text{Score}(D|E) \\
= w^T \cdot \Phi(D, E) \\
= \sum_{d \in D} w^T \cdot \Phi(d, E).
$$

(4) (5) (6)
where $\Phi$ is the feature function and $\mathbf{w}$ is the weight vector. The parser/preorderer makes greedy selections at each parsing step so that $D$ is a sequence of the atomic derivations. Each atomic derivation $d$ is a triple $\langle E[p,q], r, o \rangle$, where $E[p,q]$ is a parse span, e.g., $E[0,7]$, denoted by $[p,q)$, covers the source words $E_p, \ldots, E_{q-1}$; $r$ is the splitting point; and $o$ is the type of non-terminal nodes (straight or inverted). To represent the atomic derivation, Neubig et al. (2012) factored $d$ as a set of local features (e.g., POS tags/word classes of the first/last words in the former/latter parse spans) intersected with the node type label $S$ or $I$. The score for $d$ is the sum of the feature weights. The top-down parser starts with the whole source sentence ($E[0,7]$, i.e., the initial span) and then splits the span dichotomously and recursively until it terminates. Finally, the derivation with the highest score is picked up to perform reordering.

3 Learning Preordering Models

3.1 Online Learning

Syntactic parsers are trained on tree-annotated datasets. The BTG-based preordering method offers a substantial advantage over the conventional methods by relying on word alignments.

This section will briefly introduce the online learning algorithm passive-aggressive-I (PA-I) (Crammer et al. 2006) used in (Nakagawa 2015). Given a training example $(E, a)$, the trainer runs as follows:

1. Produce $k$-best parser derivations using beam search;
2. Select a pair of derivations $\langle D', D^* \rangle$: one derivation with the highest model score from all candidates (called system derivation, notes $D'$) and another with the highest model score from the valid subset; (called oracle derivation, notes $D^*$)
3. If these two derivations diverge or the valid derivation falls off the beam, the weights are updated.

The above procedure allows the oracle derivation to vary from one example to another during training. Nakagawa (2015) extended the atomic derivation $d$ to the four tuple $\langle E[p,q], r, o, v \rangle$, where the extra $v \in \{true, false\}$ records the validity of the current atomic derivation. $\Lambda(\cdot)$, the validation function (see Fig. 3), returns true if and only if the following property holds:

$$\max(\{y_i | i \in [p, r), y_i \neq -1\}) \leq \min(\{y_j | j \in [r, q), y_j \neq -1\}) \quad \text{if } o = \text{straight} \quad (7)$$

$$\max(\{y_j | j \in [r, q), y_j \neq -1\}) \leq \min(\{y_i | i \in [p, r), y_i \neq -1\}) \quad \text{if } o = \text{inverted}. \quad (8)$$

where $y = \{y_0, \ldots, y_{|x-1|}\}$ is the reordered ranks converted from $a$. The above formula checks
the crossings of word alignments. The parser starts with an empty derivation $D$ (valid). During parsing, the following condition is checked to validate the current derivation:

$$\forall d \in D, \Lambda(y, d) = true$$  \hspace{1cm} (9)

Let $z^n_0 = \{z_1, \ldots, z_n\} = (D', D^*)_0^n$ denote the training examples. The trainer updates the feature weights $w$ towards the solution to the following projection problem:

$$w_{t+1} = \arg\min_{w \in \mathbb{R}^n} \frac{1}{2}||w - w_t||^2 \hspace{1cm} s.t. \hspace{1cm} \ell(w; z_t) = 0$$  \hspace{1cm} (10)

where $\ell(\cdot)$ is a structured hinge-loss function, which separates $D'$ from $D^*$ by a margin that is proportional to the square root of the errors. $w_t/w_{t+1}$ are the previous and new weight vectors respectively. According to the different case of passive or aggressive, the following update rules are derived for optimisation:\footnote{For details, see the mathematical proof with the Lagrange multipliers in (Crammer et al. 2006).}

$$w_{t+1} = \begin{cases} w_t & \text{if } \ell(w; z_t) = 0, \text{ passive} \\ w_t + \eta_t \cdot \Phi(z_t) & \text{if } \ell(w; z_t) \neq 0, \text{ aggressive} \end{cases}$$  \hspace{1cm} (11)

Namely, $\ell(\cdot)$ computes the errors, $E$ returns the error cost and $\gamma$ is the margin:

$$\ell(w; z) = \begin{cases} 0 & \text{if } E(z) = 0 \\ E(z) - \gamma(w; z) & \text{otherwise} \end{cases}$$  \hspace{1cm} (12)

$$E(z) = ||D^* - D'||^\frac{1}{2}$$  \hspace{1cm} (13)

$$\gamma(w; z) = w^\top \cdot \Phi(D^*) - w^\top \cdot \Phi(D')$$  \hspace{1cm} (14)

$\eta_t$ is the aggressiveness parameter controlled with an upper bound $C$. During training, $\eta_t$ changes.

Fig. 3 An example of derivation validation.
with epochs and examples.\(^5\)

\[
\eta_t = \min \left\{ C, \frac{\ell(w_t; z_t)}{||\Phi(z_t)||^2} \right\}
\] (15)

### 3.2 Existing Problem: Gradient Noise

Nakagawa (2015) applied pairwise updating of parameters if two parses (one: *system* derivation; another: *oracle* derivation) diverged. The accuracy of the prerodering model substantially relies on the quality of word alignments. Nakagawa (2015) suggested using the symmetrised bidirectional word alignments with the INTERSECTION heuristic\(^6\) heuristic using GIZA++.

However, using the intersected word alignment has two problems:

1. Alignment errors exist in word alignments obtained from GIZA++, which will heavily affect the performance of MT systems.
2. The intersected word alignment contains less alignment points, resulting in less constraint on determining the oracle derivation.

Both of the above problems lead to gradient noise.

- **Alignment Errors**

  The first problem is easy to understand. For example, Fig. 3 shows a particular case of a mistake (assuming that the alignment point “4-4” underlined is incorrect). When using the online algorithm, incorrect updates will frequently cause a higher variance (Klasner and Simon 1995), making the resulting preorderer less accurate. Considering that a noisy gradient signal will be observed,\(^7\) this paper proposes to adopt parallel training approaches to reduce gradient noise.

- **Missing Alignment**

  Fig. 4 illustrates the second case. On the one hand, if the underlined alignment point “4-4” is removed from the example, both derivations have chances to be the “oracle”; thus, an oracle derivation is probably not the best. On the other hand, micro-reordering (coloured in Fig. 4) is meaningless for phrase-based SMT (see Fig. 1). As a consequence, as observed by Nakagawa (2015), a preorderer trained on automatically aligned datasets underperforms compared to one trained on small manual datasets. For this reason, the authors propose to utilise the remains of the derivations in the beam, i.e., the \(k\)-best list, instead of only two derivations.

---

\(^5\) \(C\) is equal to 1.

\(^6\) Other heuristics: *union*, *grow-diag-final*, and *grow-diag-final-and*.

\(^7\) This does not mean that the online PA-I algorithm is not good; on the contrary, it works very well on manual datasets.
Wang and Lepage Improved BTG-based Preordering via Parallel Parameter Averaging

Fig. 4 A particular case where two derivations tend to be valid, resulting from a weak constraint with fewer alignments. However, pairwise updating considers only one derivation as the oracle.

4 Solutions to Reduce Noise

4.1 Parallel Averaging Strategies

The main focus of this paper is the parallel training of a top-down BTG-based preorderer. Previous studies have shown that parameter averaging in model learning step can promote the learnt model (Breiman 1996; Freund and Schapire 1999; Džeroski and Ženko 2004; Rokach 2010; McDonald et al. 2010; Broderick, Boyd, Wibisono, Wilson, and Jordan 2013; Li et al. 2014). The first aim was to distribute the training algorithm similarly to McDonald et al. (2010), Broderick et al. (2013). In this case, parameter mixing was performed outside the course of data processing and did not change the training algorithm. Two distributed parameter mixing strategies we adopted: distributed averaging and iterative distributed averaging. The second aim was to modify the inside algorithm to perform parameter averaging on mini-batches, called mini-batch averaging. Though these methods fall into different categories of the learning frameworks, the idea behind them is the same: computing the margins at different levels or using training examples of different sizes reduces the variance of the learnt models. A visual comparison can be found in Fig. 5. The expectation is that the preorderer using parallel training algorithms will be more stable and robust against errors when trained on automatically aligned datasets.

4.1.1 Distributed Averaging

Distributed averaging is the most straightforward application: it trains the model to convergence separately and then performs parameter mixing. The training examples $\mathcal{T}$ are divided into
several disjoint shards as \{T_1, \ldots, T_m\} and then train on each shard $T_i$ in parallel as distributed systems.\(^8\) McDonald et al. (2010) found that, compared to a standard online perceptron, the distributed perceptron algorithm with parameter averaging yields comparable or better performance in the tasks of named entity recognition (NER) and dependency parsing.\(^9\) Such a distributed strategy is also capable of other online algorithms because during the whole period of training parameters are updated without any commutations among the child processes. The final model parameters are calculated by taking the average of all submodels.

### 4.1.2 Iterative Distributed Averaging

Iterative distributed averaging also divides the training data into several shards. Iterative distributed averaging differs from distributed averaging in that it combines the parameters of submodels after each epoch, i.e., ‘iterative’, and then, resends the averaged weights to child processes and updates the weights of submodels. The main shortcoming of iterative distributed averaging is that it needs extra memory to cache the submodels. McDonald et al. (2010) showed that iterative distributed averaging outperforms distributed averaging or the standard online method in the task of dependency parsing.

### 4.1.3 Mini-batch Averaging

In the PA-I algorithm, some parameters are not linearly separable. For instance, $\eta$ is a parameter that needs to be computed dynamically during training, hence simple batch processing is impractical. To adapt batch learning on the PA algorithm, it needs several tricks. Theoretically, it is possible to estimate $\eta$ approximately on mini-batch instead of examples. Concretely, $\beta \in \mathbb{R}$ is a small positive number, $\bar{z} = \{z_1, \ldots, z_m\}$ denotes the fixed training set (i.e., one mini-batch),

\(^8\) Here, while other techniques of bootstrap aggregating or boosting are also suitable for this situation, we only test the most straightforward solution.

\(^9\) The proof of the convergence can be found in McDonald et al. (2010).
denotes the initial weights before the current mini-batch, and \( w_t \) denotes the weights after training. If all examples are iterated in \( \hat{z} \) and weights are updated using \( \ell(w_t; z) > \beta \), then the loss for any \( z \in \hat{z} \) at time \( t \) will be less than \( \beta \), i.e., \( \ell(w_t; z) \leq \beta \). Hence, the decrease in the loss function guarantees its convergence, which makes the PA-I algorithm compatible with batch learning.\(^{10}\)

In the current implementation, all errors were collected \( \Delta E(\hat{z}) \) in a mini-batch, and the aggressiveness parameter \( \Delta \eta_t \) was computed using the following formula:

\[
\Delta \eta_t = \min \left\{ C, \frac{\sum_{i=1}^{m} \ell(w_t; z_i)}{||\Phi(\hat{z})||^2} \right\}
\]

(16)

The variance of the gradient for the mini-batch was reduced by a factor of \( m \) (i.e., batch size) compared to the original online PA-I algorithm.

4.2 Using \( k \)-best List

Weight updating can also be carried out on multiple derivations (the so-called \( k \)-best-based). For example, in SMT, batch MIRA tuning generates multiple translation hypotheses at each time (Cherry and Foster 2012). The decoder is tuned using two lists of \( k \)-best hypotheses in parallel (one: \textit{hope}, another: \textit{fear}). Inspired by this idea, this paper proposes to minimize the sum of hinge-losses, taking all derivations in the \( k \)-best list into consideration.

\[
\hat{D}_{\text{hope}} = \{ D | D \in \hat{D}_{\text{top-}k} \wedge \forall d \in D, \Lambda(d) = \text{true} \}
\]

(17)

\[
\hat{D}_{\text{fear}} = \{ D | D \in \hat{D}_{\text{top-}k} \wedge D \notin \hat{D}_{\text{hope}} \}
\]

(18)

Rather than using only two derivations \( D' \) and \( D^* \) to update the weights in a pairwise style, the current research makes use of two lists (\( \hat{D}_{\text{hope}} \) and \( \hat{D}_{\text{fear}} \)) extracted from the top-\( k \) derivations, where:

\[
\mathcal{E}(z) = ||\hat{D}_{\text{hope}} - \hat{D}_{\text{fear}}||^\frac{1}{2}
\]

(19)

\[
\gamma(w; z) = w^\top \cdot \Phi(\hat{D}_{\text{hope}}) - w^\top \cdot \Phi(\hat{D}_{\text{fear}})
\]

(20)

The resulting parser is more accurate when training on large datasets and the training was faster to converge.

\(^{10}\) Proof of the convergence (see (Crammer et al. 2006), p. 6).
5 Experimental Setup

5.1 Data

To evaluate the proposed methods, the KFTT Corpus\textsuperscript{11} (English–Japanese) was used to conduct experiments. The total training set was made up of around 330,000 sentences. For the preordering experiments, four training sets were prepared: one small, dataset aligned by human annotators, called Manual-653, which was initially provided in KFTT corpus; one small dataset aligned using GIZA++\textsuperscript{12}; one large set called EM-10K contains 100,000 sentence pairs and another called EM-100K contains 100,000 sentence pairs with word alignments using GIZA++\textsuperscript{13}. Word alignments were extracted using the standard training regimen up to IBM Model 4), and then bidirectional word alignments were symmetrised with the intersection heuristic. The test set provided in the KFTT corpus was used for evaluation. For the translation experiments, the official training, tuning and test sets were used.

5.2 Features

In order to have a fair comparison, the same features as Nakagawa (2015) were used, e.g., the head and tail of the current parser span (unigram, bigram and trigram), the prefix and suffix of the splitting point, and the parent BTG tree type of the current parser span. Each word was factored with its: lexical form, part-of-speech (POS) tag and word class (Brown, Desouza, Mercer, Pietra, and Lai 1992).

5.3 Experimental Settings

In the present experiments, the parallel averaging methods (without the $k$-best list) were compared with the online training method in Nakagawa (2015). The TD-BTG-Preorderer\textsuperscript{14} was used for the baseline systems and the HieraParser\textsuperscript{15} was used for parallel implementation. For the SMT experiments, the Moses toolkit\textsuperscript{16} was used. Standard phrase-based SMT systems (Koehn, Hoang, Birch, Callison-Burch, Federico, Bertoldi, Cowan, Shen, Moran, Zens, et al. 2007) were built with lexical reordering (Koehn, Axelrod, Birch, Callison-Burch, Osborne, Talbot, and White 2005), Minimum Error Rate Training (Och 2003), and 5-gram KenLM language model.

\textsuperscript{11} http://www.phontron.com/kftt/index-ja.html
\textsuperscript{12} We ran GIZA++ on the whole training corpus.
\textsuperscript{13} We randomly selected the sentences from the training set.
\textsuperscript{14} https://github.com/google/topdown-btg-preordering
\textsuperscript{15} https://github.com/wang-h/HieraParser
\textsuperscript{16} https://github.com/moses-smt/mosesdecoder
For POS taggers, the Stanford Part-Of-Speech Tagger\textsuperscript{18} was used for English (Toutanova, Klein, Manning, and Singer 2003) and KyTea\textsuperscript{19} was used for Japanese (Neubig, Nakata, and Mori 2011). To obtain word class tags, the same implementation\textsuperscript{20} as Liang (2005) was used. The class number was fixed to 256. Results were compared in both cases: 1) no basic lexical reordering permitted (distortion limit = 0) and 2) combining the default lexical reordering model with preordering (distortion limit = 6).

For all experiments, the preorderers for 20 and 40 epochs were trained. For distributed methods, the training dataset was divided into eight shards linearly and eight threads for multi-threading processing were used. For both the TD-BTG-Preorderer and the HieraParser, a beam of size 20 was employed. For the $k$-best list, the top-five derivations were selected. Preordering for SMT experiments were performed using the model-trained 20 epochs. The experiments reported in this paper were measured on the same machine with an Intel Core i7-960 3.20 GHz CPU (four cores, and eight threads) with 32 GB RAM.

6 Experimental Results

6.1 Intrinsic Evaluation

6.1.1 Intrinsic Metrics

To evaluate preordering accuracy, Fuzzy Reordering Score (FRS), Normalized Kendall’s $\tau$ (NKT) and Complete Matching Score (CMS) were measured using the manual dataset provided in KFTT corpus.

The FRS (Talbot, Kazawa, Ichikawa, Katz-Brown, Seno, and Och 2011) is a widely-used evaluation metric which measures the quality of reordering and measures the continuity of the output order against the reference. Given the reference permutation $E_r$, FRS is calculated as the precision of word bigrams:

$$\text{FRS}(E_r, E) = \frac{B(E_r, E)}{M(E) + 1}$$

where $B$ stands for the count of overlap bigrams that appear both in the reference sentence and the reordered source sentence. $M$ stands for the total number of words in the source sentence.

\textsuperscript{17} https://kheafield.com/code/kenlm/
\textsuperscript{18} https://nlp.stanford.edu/software/tagger.shtml
\textsuperscript{19} http://www.phontron.com/kytea/index-en.html
\textsuperscript{20} https://github.com/percyliang/brown-cluster
Kendall’s $\tau$ (Kendall 1938) is another simple metric to measure reordering:

$$\tau = 2 \times \frac{\text{number of increasing pairs}}{\text{the total number of pairs}} - 1$$  \hspace{1cm} (22)

Kendall’s $\tau$ may be a negative value; hence, this metric is justified with normalization similarly to Isozaki, Hirao, Duh, Sudoh, and Tsukada (2010a). Normalized Kendall’s $\tau$ is defined as the following:

$$\text{NKT} = 1 - \frac{C}{M \times (M - 1)/2}$$  \hspace{1cm} (23)

where $C$ is the count of errors that pairs are not increasing.

The CMS measures the percentage of complete matches (the one where $\tau = 1.0$) against the total number of testing examples. Assuming that the test set contains $N$ sentences, the complete
matching score is computed as:

\[
CMS = \frac{\text{# of complete matches}}{N}
\]  

(24)

6.1.2 Reordering Results

The scores of reordering can be found in Table 1 and Fig. 6. Compared with the online training method of Nakagawa (2015), all parallel averaging strategies for training the preorderer were effective in reducing the training time. In addition, these methods significantly improved the accuracy of reorderings in terms of the CMS, FRS and NKT when trained on automatically-aligned datasets, although they underperformed on the small hand-aligned dataset. Distributed averaging is not so effective on the small dataset, but it substantially upgraded with the size of the training dataset. Iterative distributed averaging and mini-batch averaging had a competitive performance. Using the \(k\)-best list improved the results on the EM-100K and no significant difference was found on the EM-10K, resulting in slight changes in the training time. The \(k\)-best mini-batch averaging showed an outstanding result on the small dataset (Manual-653). The results also found that increasing the training data improved the overall reordering performance.

6.2 Extrinsic Evaluation

6.2.1 Extrinsic Metrics

For translation evaluation, standard automatic evaluation metrics were used in all experiments, including BLEU (Papineni, Roukos, Ward, and Zhu 2002) and RIBES (Isozaki et al. 2010a). BLEU was used because it is the most widely used metric, but because it does not take into account word order, especially for distant language pairs such as English–Japanese, other metrics were used like RIBES, which is an automatic evaluation metric based on word order correlation coefficients between reference sentences and MT output.

6.2.2 Translation Results

Table 2 shows the translation scores. All parallel averaging methods had comparable performances in the SMT experiments either using EM-10K or EM-100K. It is worth noting that automatically word-aligned datasets were noisy (contained alignment errors). Compared with Nakagawa (2015), the proposed methods achieved comparable results on the large datasets (EM-10K and EM-100K). In other words, training using parallel averaging methods was more stable near the convergence. Among these methods, distributed averaging underperformed on Manual-653 because of the unbalanced aggregation of the submodels. While using the larger training
Table 1  Reordering scores and training times (in minutes) for English–Japanese

<table>
<thead>
<tr>
<th></th>
<th>epoch = 20</th>
<th></th>
<th>epoch = 40</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FRS NKT CMS Times</td>
<td>FRS NKT CMS Times</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual-653</td>
<td></td>
<td>75.54 87.57 35.19 2.30</td>
<td>75.86 87.83 35.77 5.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>online (Nakagawa 2015)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>greedy</td>
<td>distributed</td>
<td>68.96 83.13 26.73 0.70</td>
<td>70.19 82.18 31.15 1.97</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iterative distributed</td>
<td>74.38 87.47 34.23 0.73</td>
<td>75.24 87.72 35.19 1.87</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mini-batch</td>
<td>74.38 88.04 34.23 0.86</td>
<td>75.39 88.30 34.81 2.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td>68.78 83.31 28.08 0.74</td>
<td>69.41 82.44 28.08 1.71</td>
<td></td>
</tr>
<tr>
<td>k-best</td>
<td>iterative distributed</td>
<td>74.13 88.47 33.65 0.77</td>
<td>75.03 88.50 35.87 1.97</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mini-batch</td>
<td>74.77 87.81 35.19 0.83</td>
<td>76.08 88.43 35.19 1.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EM-653</td>
<td>69.67 85.20 32.12 3.77</td>
<td>69.62 85.32 30.57 8.39</td>
<td></td>
</tr>
<tr>
<td></td>
<td>online (Nakagawa 2015)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>greedy</td>
<td>distributed</td>
<td>64.10 80.85 25.19 1.07</td>
<td>66.76 82.00 27.12 2.55</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iterative distributed</td>
<td>68.94 84.57 30.00 1.41</td>
<td>69.52 84.93 30.38 3.47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mini-batch</td>
<td>69.36 85.47 29.42 1.52</td>
<td>70.02 85.74 30.38 3.79</td>
<td></td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td>61.42 79.16 23.65 0.98</td>
<td>62.46 79.73 24.81 2.34</td>
<td></td>
</tr>
<tr>
<td>k-best</td>
<td>iterative distributed</td>
<td>67.87 85.17 29.62 1.25</td>
<td>68.74 85.54 30.77 3.16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mini-batch</td>
<td>67.83 84.32 30.00 1.62</td>
<td>68.53 84.83 30.19 3.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EM-10K</td>
<td>73.16 87.05 35.00 36.44</td>
<td>74.26 87.63 35.96 76.72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>online (Nakagawa 2015)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>greedy</td>
<td>distributed</td>
<td>73.49 86.74 33.38 13.54</td>
<td>73.86 87.21 34.04 32.34</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iterative distributed</td>
<td>74.52 87.38 36.15 14.45</td>
<td>75.22 87.78 36.54 33.23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mini-batch</td>
<td>74.89 87.32 36.54 19.05</td>
<td>75.05 87.39 36.92 38.33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td>73.30 86.58 33.84 12.77</td>
<td>73.38 86.70 34.46 30.54</td>
<td></td>
</tr>
<tr>
<td>k-best</td>
<td>iterative distributed</td>
<td>75.15 87.45 36.15 11.34</td>
<td>75.25 88.01 36.35 25.33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mini-batch</td>
<td>74.83 87.31 35.87 17.85</td>
<td>75.35 87.35 36.15 38.08</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EM-100K</td>
<td>74.66 87.58 36.35 394.23</td>
<td>76.62 88.63 37.68 811.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>online (Nakagawa 2015)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>greedy</td>
<td>distributed</td>
<td>75.73 87.48 35.77 129.34</td>
<td>75.80 87.72 36.15 266.21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iterative distributed</td>
<td>75.53 87.68 36.15 151.25</td>
<td>77.29 88.40 37.97 334.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mini-batch</td>
<td>75.14 88.02 36.97 215.55</td>
<td>76.99 88.40 38.45 454.26</td>
<td></td>
</tr>
<tr>
<td>k-best</td>
<td>distributed</td>
<td>76.13 87.74 36.15 184.92</td>
<td>76.33 87.93 36.35 406.75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iterative distributed</td>
<td>76.31 87.79 37.12 154.11</td>
<td>77.17 88.45 38.07 353.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mini-batch</td>
<td>76.66 87.75 38.07 231.26</td>
<td>77.27 88.25 38.65 476.05</td>
<td></td>
</tr>
</tbody>
</table>

Bold numbers indicate statistically significant difference with the baseline system (bootstrap resampling \( p < 0.05 \)).
Table 2 Translation scores for each system

<table>
<thead>
<tr>
<th></th>
<th>en-ja</th>
<th>ja-en</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DL = 0</td>
<td>DL = 6</td>
</tr>
<tr>
<td>BLEU</td>
<td>RIBES</td>
<td>BLEU</td>
</tr>
<tr>
<td>No pre-reordering</td>
<td>19.79</td>
<td>65.91</td>
</tr>
<tr>
<td>Manual-653</td>
<td>21.24</td>
<td>68.01</td>
</tr>
<tr>
<td>baseline</td>
<td>22.19</td>
<td>70.42</td>
</tr>
<tr>
<td>greedy distributed</td>
<td>19.47</td>
<td>66.92</td>
</tr>
<tr>
<td>iterative distributed</td>
<td>20.76</td>
<td>68.31</td>
</tr>
<tr>
<td>mini-batch</td>
<td>22.04</td>
<td>69.57</td>
</tr>
<tr>
<td>k-best distributed</td>
<td>20.78</td>
<td>67.79</td>
</tr>
<tr>
<td>iterative distributed</td>
<td>21.00</td>
<td>69.34</td>
</tr>
<tr>
<td>mini-batch</td>
<td>21.05</td>
<td>68.99</td>
</tr>
<tr>
<td>EM-10k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>baseline</td>
<td>22.77</td>
<td>71.37</td>
</tr>
<tr>
<td>greedy distributed</td>
<td>22.17</td>
<td>70.07</td>
</tr>
<tr>
<td>iterative distributed</td>
<td>22.56</td>
<td>70.97</td>
</tr>
<tr>
<td>mini-batch</td>
<td>22.99</td>
<td>71.45</td>
</tr>
<tr>
<td>k-best distributed</td>
<td>22.06</td>
<td>70.12</td>
</tr>
<tr>
<td>iterative distributed</td>
<td>22.83</td>
<td>71.61</td>
</tr>
<tr>
<td>mini-batch</td>
<td>22.85</td>
<td>71.15</td>
</tr>
<tr>
<td>EM-100k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>baseline</td>
<td>23.15</td>
<td>71.99</td>
</tr>
<tr>
<td>greedy distributed</td>
<td>23.28</td>
<td>71.55</td>
</tr>
<tr>
<td>iterative distributed</td>
<td>23.24</td>
<td>71.63</td>
</tr>
<tr>
<td>mini-batch</td>
<td>23.58††</td>
<td>72.05</td>
</tr>
<tr>
<td>k-best distributed</td>
<td>22.92</td>
<td>71.76</td>
</tr>
<tr>
<td>iterative distributed</td>
<td>23.01</td>
<td>71.93</td>
</tr>
<tr>
<td>mini-batch</td>
<td>23.51†</td>
<td>71.89</td>
</tr>
</tbody>
</table>

Bold numbers indicate no statistically significant difference with the best system. †/††: significantly better than the baseline system ($p < 0.05/p < 0.01$).
dataset (EM-100K), it performed as well as the other methods. The $k$-best mini-batch averaging had the best performance.

Comparing Table 1 and Table 2, it became clear that the higher quality preordering does not necessarily lead to better translations. There are some reasons that can explain this phenomenon. On the one hand, for phrase-based SMT, it is well known that the performance of SMT systems is not strongly correlated with the quality of the latent variables, e.g., word alignment. For example, a lower alignment error rate (AER) does not necessarily entail better translation scores. On the other hand, micro-reorderings between two leaves in a parse tree often lead to a higher reordering score. However, if the correspondence between two words in the source language and two words in the target language is captured in the phrase table, the translation accuracy may not change.

7 Conclusion

In this paper, the training algorithm of Nakagawa (2015) was improved using bootstrap aggregating with several learning techniques (mini-batch, distributed, iterative distributed and $k$-best list). These parallel training methods were capable of dealing with alignment errors that existed in the training examples. The conducted experiments demonstrated that the current proposal is more effective and efficient when trained on automatically aligned datasets, making fast training a reality. As a possible direction for future work, the author plans to apply neural models to build more accurate parsers. This paper shows that such parameter mixing techniques may benefit other NLP tasks when learning from noisy datasets. The resulting C++ implementation for BTG-based preordering described in this paper is available via Github at the following address: https://github.com/wang-h/HieraParser.

Acknowledgement

This research was supported in part by China Scholarship Council (CSC) under the CSC Grant No. 201406890026. The authors wish to thank the anonymous reviewers for their insightful comments.

Reference


California, Berkeley.


**Hao Wang:** Hao Wang received his M.E. from Waseda University and M.Sc from Shanghai University in 2014. He is currently a PhD candidate at Graduate School of Information, Production and Systems, Waseda University, supported by Oversea Graduate Student Project of the China Scholarship Council. His research interests include natural language processing and machine learning, especially machine translation.

**Yves Lepage:** Yves Lepage received his Ph.D. degree from GETA, Grenoble university, France. He worked for ATR labs, Japan, as an invited researcher and a senior researcher until 2006. He joined Waseda University, Graduate School of Information, Production and Systems in April 2010. He is a member of the Information Processing Society of Japan, the Japanese Natural Language Processing Association, and the French Natural Language Processing Association, ATALA. He was editor-in-chief of the French journal on Natural Language Processing, TAL, from 2008 to 2016.
Wang and Lepage

Improved BTG-based Preordering via Parallel Parameter Averaging

(Received February 23, 2018)
(Revised July 26, 2018)
(Accepted August 30, 2018)