Incrementally Updating the SMT Reordering Model

Shachar Mirkin
Xerox Research Centre Europe
6 Chemin de Maupertuis, Meylan, France
shachar.mirkin@xrce.xerox.com

Abstract

This work is concerned with incrementally training statistical machine translation (SMT) models when new data becomes available. That, in contrast to re-training new models based on the entire accumulated data. Incremental training provides a way to perform faster, more frequent model updates, enabling keeping the SMT system up-to-date with the most recent data. Specifically, we address incrementally updating the *reordering model* (RM), a component in phrase-based machine translation that models phrase order changes between the source and the target languages, and for which incremental training has not been proposed so far. First, we show that updating the reordering model is helpful for improving translation quality. Second, we present an algorithm for updating the reordering model within the popular Moses SMT system. Our method produces the exact same model as when training the model from scratch, but doing so much faster.

1 Introduction

Parallel data for training statistical machine translation (SMT) models is being constantly generated, both by professional and by casual translators. Typically, large amounts of data are required to produce decent SMT models, yet training a model is an expensive process in terms of time and computational resources. Most often, and in particular when community effort is made to translate new content, it is desirable to keep the system up-to-date with the new data; yet, constant retraining is not feasible. The line of research concerning *incremental training* for SMT has been addressing this problem, aiming at updating the model given new parallel data, rather than retraining it.

Typical phrase-based SMT models use a log-linear combination of various features that mostly represent three sub-models: a *translation model* (TM), responsible for the selection of a target phrase for each source phrase, a *language model* (LM), addressing target language fluency, and a *reordering model* (RM). The reordering model is required since different languages exercise different syntactic ordering. For instance, adjectives in English precede the noun, while they typically follow the noun in French (*the blue sky* vs. *le ciel bleu*); in Modern Standard Arabic the verb precedes the subject, and in Japanese the verb comes last. As a result, source language phrases cannot be translated and placed in the same order in the generated translation in the target language, but phrase movements have to be considered. This is the role of the reordering model. Estimating the exact distance of movement for each phrase is too sparse; therefore, instead, the *lexicalized reordering model* (Koehn, 2009) estimates phrase movements using only a few reordering types, such as a *monotonous* order, where the order is preserved, or a *swap*, when the order of two consecutive source phrases is inverted when their translations are placed in the target side.

Most research on incremental training for SMT addresses parallel corpus alignment, the slowest step of the model training and a prerequisite of many of the following steps, including the reordering model generation. Currently, keeping the reordering model

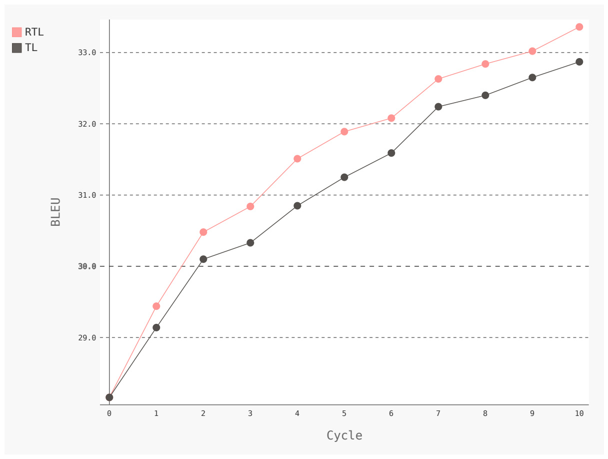

Figure 1: BLEU scores of an SMT system trained with additional data, over 10 cycles, with and without updating the reordering model. *R*, *T* and *L* denote the models that have been updated – Reordering, Translation and Language models. The exact setting of this experiment, as well as additional details, are provided in Section 6.

up-to-date requires retraining. Yet, refraining from updating this model is expected to yield inferior translation performance. An example is shown in Figure 1, comparing results with and without an updated reordering model. While not as important a component as the TM or the LM (see further results in Section 6), updating the RM does improve translation. We therefore seek to allow quick incremental updates of the RM within Moses (Koehn et al., 2007). In this paper we outline several practical options to carry out this update, and describe an implementation of one of them. In a set of experiments we show both that RM updates help improving results and that is can be carried out much quicker than reconstructing the model from scratch.

Next, we describe related work on SMT model updates (Section 2), and provide the details of the Moses reordering model and its relevant data structures (Section 3); we outline and analyze several options to perform RM updates in Section 4, and propose an method in Section 5. Section 6 includes evaluation in terms of translation performance and run-time, and Section 7 summarizes this work and suggests future research directions.

2 SMT model updates

Statistical machine translation systems rely on the availability of large parallel corpora, in particular of the target domain. Such corpora are not always available at the initial stage of the SMT model training, but are sometimes obtained during the lifetime of the system. More parallel data, especially in-domain, may become available, for instance, as users of the system *post-edit* the automatic translations. The source texts and their corrected translations then become new parallel corpora with which the system can be updated. It is then desirable to incorporate the new data into the SMT model as soon as possible. This is particularly a concern for Computer Assisted Translation (CAT) systems, where one wishes to reflect the corrections immediately to avoid repeating translation errors that have already been corrected. The straightforward way to incorporate new data into an SMT model is to retrain the model, i.e. to use all the data accumulated until that point and create the model all over again. However, such retraining may be a lengthy and computationally expensive process, leading to long lags between system updates.

Incremental training provides a principled way to incorporate new data into an existing model without retraining it. For SMT, incremental training research mainly focuses on updating the alignment probabilities from the parallel data. Rightfully so – alignment is the most time-consuming step in SMT model training, which is needed for generating both the translation and the reordering models. Once the alignment model has been updated, and the new data aligned, it is possible to create new data-structures for all sub-models which take into account the entire parallel data. Overall, model update with incremental training is typically a much faster process.

GIZA++[1] (Och and Ney, 2003) is probably the best known alignment tool, and is also the tool used in the Moses translation system. Yet, even with its multi-threaded version, MGIZA++ (Gao and Vogel, 2008), alignment remains the longest step in the SMT model generation. GIZA, like other alignment tools, is using the Expectation Maximization (EM) algorithm (Cappé and Moulines, 2009) to simultaneously learn alignment and translation probabili-

[1]`https://code.google.com/p/giza-pp/`

ties (Brown et al., 1993). Yet, EM relies on having all the data available in advance. When incremental updates to the model are required, *online EM* comes into play. Here, the model parameters may be updated every time a new data point – a sentence-pair, in our case – is introduced. This makes it feasible to perform more frequent updates, thus maintaining the model up-to-date with recent data. Several variants of online EM have been proposed (Liang and Klein, 2009), among which is *stepwise EM* used in (Levenberg et al., 2010; Levenberg, 2011) for updating the parameters of the translation and alignment models. Using IBM Model 1 (Brown et al., 1993) with HMM alignments (Vogel et al., 1996), they collect counts for translations and alignments and update them by interpolating the statistics of the old and the new data. Rather than updating the model for each data point, they do so for a set of bi-sentences, referred to as *mini-batch*. In this work we are using Incremental GIZA++,[2] an implementation of this work, updating the model multiple times with mini-batches of additional parallel data.

Force alignment (Gao et al., 2010) is a technique for aligning new data using an existing model. This enables adding the source and its translation as additional training material. It does not, however, make any updates to the model.[3]

An alternative practical approach to incrementally updating alignments, referred to as *quick updates*, was proposed in (Mirkin and Cancedda, 2013). Instead of updating the existing translation and language models, separate models are generated from smaller amounts of data (e.g. solely the new data) and combined with the previous models through a log-linear combination. This approach allows even faster updates, and in some settings yields comparable results to retraining the model.

Yet, in contrast to the translation and language models, currently Moses supports a single reordering model. Hence, while it is possible to quickly create small TMs and LMs, this is not possible for the reordering model. If its update is ignored, biphrases absent from the reordering model receive a default score, resulting with suboptimal results, as demonstrated in Section 1. Incremental updates of the reordering model have not been addressed yet and the only option currently available is to generate the reordering model from start, which might be a lengthy process. In the following sections we describe our suggestion for incremental and quick updates of this model.

[2]`https://code.google.com/p/inc-giza-pp/`

[3]We have experimentally confronted Incremental GIZA with force alignment and learned that the former method outperforms the latter.

3 The Moses reordering model

3.1 Reordering probability estimation

As we mentioned in Section 1, the reordering model estimates the probability of phrase movements between the source and the target. To deal with sparsity, movement is measured in the lexicalized reordering model in terms of *orientation* types, rather than exact move distance. The default orientations used in Moses are listed below, and are referred to as *msd* (Koehn, 2009):

- *mono* (*monotonous*) – the preceding target phrase is aligned to the preceding source phrase.
- *swap*: the preceding target phrase is aligned to the following source phrase.
- *discontinued* (also called *other*): the phrases did not occur consecutively, but other phrases were inserted between them.

Formally, the probability of each of the above orientation types, o, for a source phrase f and a target phrase e is denoted $p(o|f,e)$. Counting the orientation instances of each phrase pair from the word alignments, in each direction, maximum likelihood is used to estimate this probability:

$$\hat{p}(o|f,e) = \frac{count(o,f,e)}{\sum_{o'} count(o',f,e)} = \frac{count(o,f,e)}{count(f,e)} \tag{1}$$

The estimation can be smoothed by additive (Laplace) smoothing with a factor σ:

$$\hat{p}(o|f,e) = \frac{\sigma + count(o,f,e)}{\sum_{o'} \sigma + count(f,e)} \tag{2}$$

3.2 Data structures

Extracted phrases During the training of a phrase-based Moses model, phrase pairs are extracted from the word-aligned parallel data and used for training both the TM and the RM. Within the phrase extraction step, three files containing the list of phrase pairs are created. Two of them consist of the word alignments within the phrases, one in each direction (source-to-target and target-to-source); the third, the *reordering file*,[4] shows the orientation of each occurrence of the phrase pair, in either direction. Phrase pairs are alphabetically ordered in these files, and repeat if more than one instance of the phrase pair is encountered.

Figure 2 shows a few lines from a reordering file, of an English to French model, built with the *msd* (monotonous-swap-discontinued) orientations (Koehn et al., 2005)[5] Each line in the reordering file contains three parts, separated by '|||': source phrase, target phrase, and 2 indicators of the orientation in which this instance was found, when extracting the phrases from source-to-target and from target-to-source alignments.

Reordering table The *reordering table* (RT), created from the reordering file, is the data structure representing the reordering model. It contains probability estimations for each orientation of a phrase pair in either direction. In contrast to the reordering file, in the RT, each phrase pair appears only once. Figure 3 displays a few lines from a reordering table. In Section 5 we show how these estimations are computed.

4 Updating the reordering model

In this section we describe several options to generate an updated reordering model given new data. We are specifically concerned with a multi-update scenario, where the model needs to be updated with new data repeatedly rather than only once.

4.1 Reordering model generation

Several steps must be performed before a Moses RM can be trained. The necessary steps on which the model generation depends on are listed below.

1. Corpus preparation: tokenization, lowercasing and any other preprocessing.
2. Corpus alignment in both directions, source-to-target and target-to-source.
3. Bidirectional phrase extraction.
4. Creation of the reordering file.

Note that some steps are necessary for other purposes. For instance, Step 1 is necessary for all subsequent steps, including LM training, and Steps 2 and 3 are also necessary for training the TM. In practice, the creation of the reordering file (Step 4) is done within the phrase extraction step.

From the reordering file, the reordering table is created by counting the number of occurrences of each orientation in each direction and normalizing by the total number of occurrences of the phrase pair, as in Equation 2.

4.2 Update options

We now consider several options for updating the reordering model, listing the tasks that need to be performed and analyze their complexity, where the size of a data structure is measured in terms of the number of lines it contains. We can assume that the data that was already used to train the current model (the older data) is significantly larger than the training data which we use for a single update (the newer data). This would typically be the case, for instance, with training data that is based on human feedback, as described earlier. For simplicity, we always refer below to the old data as $\mathcal{A}$ and to the new data as $\mathcal{B}$ without cycle indexes.[6] As we proceed with subsequent update cycles, $\mathcal{A}$ keeps growing, while the size of $\mathcal{B}$ does not depend on prior cycles.

We denote the set of phrase pairs instances generated from the training data – the phrase pairs in the reordering file – as $\mathcal{P}$, with subscript $\mathcal{A}$, $\mathcal{B}$ or $\mathcal{AB}$, marking whether it refers to the old, new or merged (updated) data, respectively. As mentioned, $\mathcal{B}$ is typically much smaller than $\mathcal{A}$: $|\mathcal{P}_\mathcal{B}| \ll |\mathcal{P}_\mathcal{A}|$, and the merged set is at least as large as the old one. That is, $|\mathcal{P}_{\mathcal{AB}}| \geq |\mathcal{P}_\mathcal{A}|$, and $\mathcal{P}_{\mathcal{AB}}$ is strictly larger than $\mathcal{P}_\mathcal{A}$ if any new phrase pairs are found in the new data relative to the older one.

[4]Not to be confused with the reordering *table*.

[5]More precisely, this is the *msd-bidirectional-fe* model, also referred to as *wbe-msd-bidirectional-fe-allff*.

[6]Denoting the initial "old" training data as $\mathcal{A}_0$ and the first new data as $\mathcal{B}_1$, $\mathcal{A}_i = \mathcal{A}_{i-1} \cup \mathcal{B}_i$, where $i = 1, 2, \ldots$ and '$\cup$' denotes the concatenation of the two training datasets.

```
but of course ||| mais bien sûr ||| mono mono
but of course ||| mais bien sûr ||| mono other
but of course ||| mais bien sûr ||| mono other
...
confusion between the ||| confusion entre le ||| other other
confusion between the ||| confusion parmi les ||| other mono
...
emerging ||| naissante ||| mono mono
emerging ||| naissante ||| other mono
emerging ||| naissante ||| other mono
emerging ||| naissante ||| other other
emerging ||| naissante ||| swap other
emerging ||| naissante ||| swap other
emerging ||| naissante ||| swap other
```

Figure 2: Sample lines from a Moses reordering file with *msd* orientations.

```
but of course ||| mais bien sûr ||| 0.78 0.11 0.11 0.33 0.11 0.56
...
confusion between the ||| confusion entre le ||| 0.20 0.20 0.60 0.20 0.20 0.60
confusion between the ||| confusion parmi les ||| 0.20 0.20 0.60 0.60 0.20 0.20
...
emerging ||| naissante ||| 0.18 0.41 0.41 0.41 0.06 0.53
```

Figure 3: Sample lines from a Moses reordering table generated for the *msd* orientations, with 6 feature scores for each phrase pair. The scores are probability estimations, summing to 1 for each direction. For easier display, we round the scores to 2 places after the decimal point.

In contrast to the reordering file, the reordering table contains only unique phrase pairs. We denote the set of unique phrase pairs in each data structure with the superscript (u). For example, the phrase pairs in the new RT are marked as $\mathcal{P}_{\mathcal{B}}^{(u)}$, where $|\mathcal{P}_{\mathcal{B}}^{(u)}| \leq |\mathcal{P}_{\mathcal{B}}|$. To get an intuition of the involved sizes, a reordering file created from 500,000 lines of the tokenized, lowercased Europarl corpus (Koehn, 2005) contains approximately 57M lines of non-unique phrase pairs, and the reordering table contains 33M pairs (58%); the figures for the complete Europarl corpus (1.96M lines after cleaning) are 219M for the reordering file in comparison to 107M lines for the RT (49%).[7]

The update options are listed hereunder. Using Incremental GIZA, all produce the same RT. With respect to complexity, we assume that the old reordering file and the old RT are available at no cost because they were created at previous training iterations. We also assume that phrase extraction of the new data, from which the reordering file is created, is done in any case since it is also needed for the translation model.

[7]The more data we use, especially of the same domain, the fewer new phrase pairs we expect to see; since the RT, but not the reordering file, contains only unique phrase pairs, the ratio of their sizes is expected to decrease with more data.

I. Constructing a reordering table from scratch. This is the non-incremental option to construct the reordering table. Phrase pairs are extracted from the entire data, sorted and a reordering table is constructed. This is obviously the slowest option, and the only one available to-date in Moses. All following options are incremental.

II. Merging reordering files and creating a merged reordering table. Given the reordering file from the new data, $\mathcal{B}$, we can perform a merge of two reordering files in either one of two ways: concatenate $P_{\mathcal{A}}$ and $P_{\mathcal{B}}$ and sort the concatenation, or – since both files are sorted – read the files line-by-line in parallel and merge them to a single file that is already sorted. This can be done in linear time in the size of the two reordering files, $\Theta(|\mathcal{P}_{\mathcal{A}}| + |\mathcal{P}_{\mathcal{B}}|)$. We then create a single reordering table by an additional pass over the merged reordering file. The merge of reordering files and creation of the reordering table can be collapsed into one step, requiring a single pass, but we cannot avoid creating the merged reordering file, since if we follow this option, this file will be required for the next update cycle.

III. Merging a reordering file with an existing reordering table. For this option we need to keep track of the number of occurrences of each phrase pair, since this information is lost during the creation of the reordering table. We pass through the old RT and the new reordering file at the same time, comparing their entries ($\Theta(|\mathcal{P}_{\mathcal{A}}^{(u)}| + |\mathcal{P}_{\mathcal{B}}|)$). Unique entries in the RT are copied as-is to the merged RT, and new entries are created in it for phrase pairs that appear only in the reordering file, using all the lines of the same phrase pair. Whenever we encounter a phrase pair that exists in both, we update the probability estimations of the pair in the RT, based on the accumulated counts from the two data structures.

IV. Merging two reordering tables. This options requires tracking occurrence counts as well. Here, we first create a new RT from the reordering file of the new data in $\Theta(|\mathcal{P}_B|)$, and then merge the old and the new tables. The merge is linear in the size of the two tables, $\Theta(|\mathcal{P}_{\mathcal{A}}^{(u)}| + |\mathcal{P}_{\mathcal{B}}^{(u)}|)$. Starting with two sorted tables, the merged table we end up with is also sorted. As above, entries of unique phrase pairs are copied as-is to the merged RT, and when we encounter two lines with the same phrase pair, we update the pair's probability estimations base on the sum of its counts in the two tables. If we keep occurrence counts in the reordering tables themselves, once the merged table has been created, there is no further need to keep the reordering file. The merged RT will be sufficient for subsequent update cycles.

The fourth option may be slightly slower than the third one since it requires an additional pass through the new RT. However, any processing of $\mathcal{B}$ is fast in terms of actual runtime, due to its small size in the addressed scenario. We chose to implement the fourth option – merging of two reordering tables – due to its simplicity, and describe it in detail in Section 5.

5 Merging reordering tables

In this section we present a simple algorithm for a reordering model update via the merge of two reordering tables. As mentioned in Section 4, this update option requires keeping track of the number of occurrences of each phrase pair. We first present the format and technical details of this extension of the reordering table, and then provide the details of the suggested merge itself.

5.1 Reordering table with counts

To enable updating the table without generating it from scratch we must keep track of the number of occurrences of each phrase pair. To do it without making changes to Moses code, we add the total count of a phrase pair as an additional value following the feature scores in the reordering table. Figure 4 shows several lines of the reordering table shown earlier, now including counts.

Below is a demonstration of calculating the orientations scores in Figure 4 in the source-to-target direction, using Equation 2. In the equations below, $S(\cdot)$ is a scoring function and $C(\cdot)$ is a count function, using counts from the reordering file; f is *'emerging'* and e is *'naissante'* from Figure 4, which occur totally 7 times, out of which, the *mono* orientation occurs once in this direction, and each of *swap* and *other* occur 3 times. Each score is the result of smoothing the counts with a σ factor of 0.5 to avoid 0 probabilities. While demonstrated on the *msd* model, there is nothing that prevents applying the same approach to a different set of orientations.

$$\begin{aligned} &S(mono|f,e) \\ &\quad = \frac{\sigma + C(mono,f,e)}{3\sigma + C(f,e)} = \frac{0.5+1}{1.5+7} = 0.18 \end{aligned} \quad (3)$$

and

$$\begin{aligned} &S(swap|f,e) \\ &\quad = \frac{\sigma + C(swap,f,e)}{3\sigma + C(f,e)} = \frac{0.5+3}{1.5+7} = 0.41 \end{aligned} \quad (4)$$

Hence, recovering from the score the count of a specific orientation (e.g. *mono*) for a given phrase pair:

$$\begin{aligned} &C(mono,f,e) \\ &\quad = S(mono|f,e) \times (3\sigma + C(f,e)) - \sigma \\ &\quad\quad = 0.18 \times (1.5+7) - 0.5 = 1 \end{aligned} \quad (5)$$

```
but of course ||| mais bien sûr ||| 0.78 0.11 0.11 0.33 0.11 0.56 3
...
confusion between the ||| confusion entre le ||| 0.20 0.20 0.60 0.20 0.20 0.60 1
confusion between the ||| confusion parmi les ||| 0.20 0.20 0.60 0.60 0.20 0.20 1
...
emerging ||| naissante ||| 0.18 0.41 0.41 0.41 0.06 0.53 7
```

Figure 4: Sample lines from a reordering table with counts.

To support RT with counts, the configuration (*ini*) file is adjusted to include 7 features instead of 6 (the number of features in the *msd* model), and its weight is set to 0. Figure 5 shows the relevant lines from a tuned configuration file, updated to support counts.

5.2 Merging RTs

Algorithm 1 presents the pseudo code of merging two reordering tables with counts, $R_{\mathcal{A}}$ and $R_{\mathcal{B}}$, into a single one, $R_{\mathcal{AB}}$. The procedure is as follows: We read the reordering tables in parallel, one line at a time, and compare the phrase pair in the old table with the one in the new one. The comparison is alphabetical, using a string made of the source phrase, the delimiter and the target phrase. When the two lines refer to different phrase pairs, we write into the merged table, R_{AB}, the one that alphabetically precedes the other, and read the next line from that table. If they refer to the same phrase pair we merge the lines into a single one, which we write into $R_{\mathcal{AB}}$, and advance in both tables. When one table has been read completely, we write the remainder of the other one into $R_{\mathcal{AB}}$.

Merging two lines into a single one (MERGE_LINES in Algorithm 1) consists of the following steps:

1. Convert the feature scores in each line into counts, as in Equation 5.
2. Sum up the counts for each orientation, as well as the total count.
3. Convert the updated counts of the orientations into scores, as in Equations 3 and 4.

As mentioned in Section 4, the complexity of this algorithm is linear in the length of the tables, i.e. $\Theta(|\mathcal{P}_{\mathcal{A}}^{(u)}| + |\mathcal{P}_{\mathcal{B}}^{(u)}|)$. In terms of memory usage, neither table is fully loaded into memory. Instead, at any given time a single line from each table is read.

Algorithm 1 Merging reordering tables with counts

```
1:  procedure MERGE_R_TABLES($R_{\mathcal{A}}$,$R_{\mathcal{B}}$)
2:      Read first lines of $R_{\mathcal{A}}$ and $R_{\mathcal{B}}$, $R_{\mathcal{A}}^{(1)}$, $R_{\mathcal{B}}^{(1)}$
3:      $i := 1; j := 1$
4:      while $R_{\mathcal{A}}^{(i)} \neq null$ and $R_{\mathcal{B}}^{(j)} \neq null$ do
5:          if $R_{\mathcal{A}}^{(i)} < R_{\mathcal{B}}^{(j)}$ then // Compare bi-phrases
6:              $R_{\mathcal{A}}^{(i)} \rightarrow R_{\mathcal{AB}}$
7:              $i := i + 1$
8:          else if $R_{\mathcal{A}}^{(i)} > R_{\mathcal{B}}^{(j)}$ then
9:              $R_{\mathcal{B}}^{(j)} \rightarrow R_{\mathcal{AB}}$
10:             $j := j + 1$
11:         else // Identical bi-phrases
12:             MERGE_LINES($R_{\mathcal{A}}^{(i)}, R_{\mathcal{B}}^{(j)}$) $\rightarrow R_{\mathcal{AB}}$
13:             $i := i + 1; j := j + 1$
14:         end if
15:     end while

        // Write the rest of the tables:
        //   at least one of them is $EOF$
16:     while $R_{\mathcal{A}}^{(i)} \neq null$ do
17:         $R_{\mathcal{A}}^{(i)} \rightarrow R_{\mathcal{AB}}$
18:         $i := i + 1$
19:     end while
20:     while $R_{\mathcal{B}}^{(j)} \neq null$ do
21:         $R_{\mathcal{B}}^{(j)} \rightarrow R_{\mathcal{AB}}$
22:         $j := j + 1$
23:     end while
24: end procedure
```

```
LexicalReordering name=LexicalReordering0 num-features=7
type=wbe-msd-bidirectional-fe-allff input-factor=0
output-factor=0

LexicalReordering0= 0.0857977 0.0655027 0.0486593 0.115916 -0.0182552 0.0526204 0
```

Figure 5: An example Moses ini file with required changes to support RT counts.

6 Evaluation

In this section we evaluate updating the reordering model from two aspects: (i) translation performance and (ii) run-time. Specifically, we first show that updating this model helps improving translation, as reflected in the BLEU score (Papineni et al., 2002); then we show that the incremental update is faster than the complete one.

6.1 Setting

We used the IWSLT 2013 Evaluation Campaign data, of the English-French MT track.[8] The initial model was trained with 10,000 WIT3 (Cettolo et al., 2012) sentence-pairs; we use 50,000 additional ones to train updated models. The additional data is split into 10 parts of 5,000 bi-sentences, each added to the data used in the prior cycle to generate an updated model. Moses[9] is used as the phrase-based SMT system, with a configuration comprising of a single phrase table and a single LM. 5-gram language models are trained over the target-side of the training data, using SRILM (Stolcke, 2002) with modified Kneser-Ney discounting (Chen and Goodman, 1996). Mean Error Rate Training (MERT) (Och, 2003) is used for tuning the initial model using the development set of the abovementioned campaign, consisting of 887 sentence-pairs, and optimizing towards BLEU. The models are evaluated with BLEU over the campaign's test set of 1,664 bi-sentence. All datasets were tokenized, lowercased and cleaned using the standard Moses tools.

In all our experiments, we use Incremental GIZA that allows updating the alignment and translation models without aligning all the training data at every cycle. With Incremental GIZA, the alignment of the parallel data is identical in both the incremental and the complete RM generation experiments, since even though the alignment probabilities are being updated, only the new data is being aligned, while the older data is left untouched. As a result, we obtain the same phrase pairs from the new data for both RM generation methods. Given that, our algorithm produces the exact same reordering model as its generation from the entire data (up to numerical accuracy).

6.2 Translation performance

First, we demonstrate that updating the reordering table help achieving better translation quality. To that end, we compare all possible combinations of updating the three phrase-based SMT sub-models (reordering, translation and language models, denoted *R*, *T* and *L*, respectively). Figure 6, that includes a detailed view of Figure 1, shows the results of the experiments with each one of these combinations. From the figure we learn that: (i) the reordering model is the least important one of the three. This is consistent with prior work, e.g. (Mirkin and Cancedda, 2013); (ii) updating the reordering model without updating the translation model has practically no impact on results, since new phrase pairs from the new data that are not added to the phrase table cannot be used in the translation. This is reflected in the almost flat line of experiment *R*, and in the very similar results of *RL* in comparison to *L*. The slight improvement in this case may be attributed to more statistics that have been accumulated for the phrase pairs that already existed in the initial data; (iii) when the translation model is updated, adding the reordering model does help, as seen in *RTL* vs. *TL* and *RT* vs. *T*.

6.3 Run-time

We now compare the time necessary to train a reordering model from scratch (complete training) vs. using the suggested incremental update. For this experiment, we used the English-French Europarl corpus, with 1.96 million parallel sentences as $\mathcal{A}$ and 10,000 WIT3 sentence-pairs as $\mathcal{B}$. Other details of the settings did not change.

[8]Downloaded from https://wit3.fbk.eu/mt.php?release=2013-01.

[9]We used the version released on 14/3/2014.

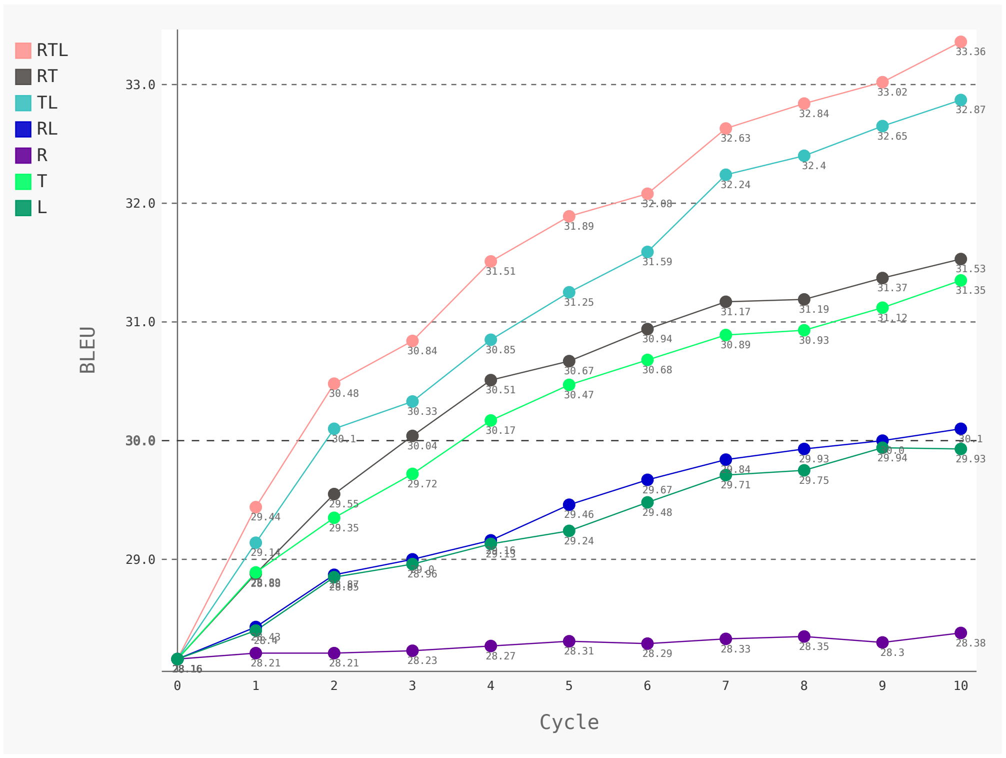

Figure 6: Translation performance (BLEU) when incrementally updating the model with additional data, over 10 update cycles, with different combinations of **R**eordering, **T**ranslation and **L**anguage models.

To objectively measure the run-time of the required steps, regardless of the computer's load at the specific time of experiment, we use the Linux command *time*, summing up the *user* and *sys* times, i.e. the total CPU-time that the process spent in user or in kernel modes. All measurements were conducted on a 64-bit Centos 6.5 Linux server, with 128 GB of RAM and 2 Intel Xeon 6-core 2.50GHz CPUs.

A complete reordering model update, when using Incremental GIZA, consists of of the following two steps:

1. Extracting phrase pairs and creating a reordering file from all the data ($\mathcal{A} \cup \mathcal{B}$)
2. Creating a reordering table from the single reordering file of $\mathcal{A} \cup \mathcal{B}$

In comparison, the incremental update requires the following steps:

1. Extracting phrase pairs and creating a reordering file from the new data ($\mathcal{B}$)
2. Creating a reordering table from the reordering file of $\mathcal{B}$
3. Merging the RTs of $\mathcal{A}$ and $\mathcal{B}$

The time required for generating the complete model in our experiment was 83.6 minutes, in comparison to 17.6 minutes for the incremental one, i.e. 4.75 times faster.

We note that $\mathcal{A}$ represents a corpus of medium size, and often the initial corpus would be much larger.[10] Concerning $\mathcal{B}$, say we plan to perform daily system updates, then a set of 10,000 sentences pairs constitutes a substantial amount of data in terms of what we can expect to obtain in a single day. Hence, the time gain in actual settings may be even larger.

7 Conclusions and future work

This work addressed the incremental update of the reordering model of a phrase-based SMT system. We showed that updating this model is useful for obtaining improved translation, even for a language

[10]For comparison, the rather popular MultiUN corpus (Eisele and Chen, 2010) consists of 13.2M parallel sentence for this language pair (`http://opus.lingfil.uu.se/MultiUN.php`, accessed on 7 August 2014).

pair such as English-French, where phrase movements are not very prominent (in comparison to English-Japanese, for example). We proposed a method for incrementally training this model within the Moses SMT system, which can be done much faster than a complete retrain. It thus supports more frequent SMT model updates to enable quickly benefiting from newly obtained data and user feedback and reflecting it in the system's translation. For future work we wish to investigate using weighted incremental updates of the reordering model, which may enable giving, for instance, more weight to in-domain vs. out-of-domain data or for preferring more recent data. Another extension of this work would be to address updating the binarized version of the reordering table, which enables using the reordering model without loading it into memory.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 287755. We thank Philipp Koehn for a valuable discussion at the beginning of this work.

References

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2):263–311, June.

Olivier Cappé and Eric Moulines. 2009. On-line expectation–maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):593–613.

Mauro Cettolo, Christian Girardi, and Marcello Federico. 2012. WIT^3: Web inventory of transcribed and translated talks. In *Proceedings of EAMT*.

Stanley F. Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of ACL*.

Andreas Eisele and Yu Chen. 2010. MultiUN: A multilingual corpus from united nation documents. In *Proceedings of LREC*.

Qin Gao and Stephan Vogel. 2008. Parallel implementations of word alignment tool. In *Proceedings of Software Engineering, Testing, and Quality Assurance for Natural Language Processing*.

Qin Gao, Nguyen Bach, and Stephan Vogel. 2010. A semi-supervised word alignment algorithm with partial manual alignments. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*.

Philipp Koehn, Amittai Axelrod, Alexandra Birch, Chris Callison-Burch, Miles Osborne, David Talbot, and Michael White. 2005. Edinburgh system description for the 2005 IWSLT speech translation evaluation. In *Proceedings of IWSLT*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL Demo and Poster Sessions*.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of MT Summit*.

Philipp Koehn. 2009. *Statistical machine translation*. Cambridge University Press.

Abby Levenberg, Chris Callison-Burch, and Miles Osborne. 2010. Stream-based translation models for statistical machine translation. In *Proceedings of HLT-NAACL*.

Abby Levenberg. 2011. *Stream-based Statistical Machine Translation*. Ph.D. thesis, University of Edinburgh.

Percy Liang and Dan Klein. 2009. Online EM for unsupervised models. In *Proceedings of NAACL*.

Shachar Mirkin and Nicola Cancedda. 2013. Assessing quick update methods of statistical translation models. In *Proceedings of IWSLT*.

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 *Proceedings of ACL*.

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

Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *Proceedings of Interspeech*.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. HMM-based word alignment in statistical translation. In *Proceedings of COLING*.