Assessing Quick Update Methods of Statistical Translation Models

Shachar Mirkin[1], *Nicola Cancedda*[2]

[1] Laboratoire d'Informatique de Grenoble, [2] Xerox Research Centre Europe

shachar.mirkin@imag.fr, nicola.cancedda@xrce.xerox.com

Abstract

The ability to quickly incorporate incoming training data into a running translation system is critical in a number of applications. Mechanisms based on incremental model update and the online EM algorithm hold the promise of achieving this objective in a principled way. Still, efficient tools for incremental training are yet to be available. In this paper we experiment with simple alternative solutions for interim model updates, within the popular Moses system. Short of updating the model in real time, such updates can execute in short timeframes even when operating on large models, and achieve a performance level close to, and in some cases exceeding, that of batch retraining.

1. Introduction

Statistical Machine Translation (SMT) systems largely depend on the availability of parallel corpora for training and tuning. Even more crucial is the availability of sufficient in-domain training data. That is, parallel corpora from the same domain the system will be used for. Methods for dealing with insufficient in-domain data typically fall within the *domain adaptation* line of research. Such methods strive to make the best possible use of the in-domain data or to obtain additional bilingual data that is similar to the target domain. Generally speaking, the more in-domain data is available, the better is the translation.

New training data can become available as more translations are being produced (e.g. in the case of the European Parliament proceedings), through focused data collection, or as the result of user feedback to the translation system. Specifically, *post-editing*, i.e. manual correction of automatic translations, is a useful source for training data.

An additional challenge beyond obtaining sufficient in-domain training data (or any training data), is feeding the new data into an existing up-and-running translation system. A standard way to comprehensively update an SMT model based on new data is to re-train the model with the entire data that is available at a given time. This kind of training is often referred to as *batch* (re-)training. Such a process is time consuming and intensive in computational resources, especially when large datasets are involved. In consequence, it may not be feasible to run it often enough, resulting with long lags between two model batch updates, in which the running system is not up-to-date with the newest possible model.

Incremental training algorithms address this issue by enabling an SMT model update based on the new data rather than retraining the model from scratch. This is performed, for instance, by using online versions of the Expectation Maximization algorithm, that is employed in the alignment step of the SMT model construction. Incremental training for SMT models is a relatively new line of research, and mature tools to perform the required updates efficiently are still largely missing.

Still, as we show in this paper, other configurations of the SMT system are also providing means for utilizing new data in between batch updates. We compare several such configurations, where in-domain data is based on spoken language transcriptions, to assess which methods are practically useful for quickly updating the model, especially when the new data belongs to the target domain.

Consider the following setting of an automatic translation system that is either a standalone translator or as part of a larger software system. The system is deployed and is being used, as more training data is becoming available constantly, e.g. through users who provide corrections to the system's translations. To use this data, two kinds of update cycles are employed: (i) a long cycle (e.g., a week), at which end we can perform a *slow update*, that can include re-training, tuning and any other time-consuming tasks; (ii) a short cycle (a day, for instance) in which we wish to carry out a *quick update* consisting of only light-weight tasks that are guaranteed to complete in a timely manner. In these short cycles the model is updated with the newly obtained data. The goal is to improve the model with respect to the previous slow update, and reflect the received feedback; we do not necessarily expect to obtain as good a performance as the following slow update, but hope to be in the same ballpark. The focus of this work is in identifying the most appropriate setting for quick updates, both in terms of translation quality and of time. That, with tools that are currently available.

In the remaining sections we provide (Section 2) a short background about incremental training and domain adaptation techniques, and discuss the effort of each of the steps in building a phrase-based SMT model; we present the configurations for quick updates that we assessed (Section 3), and describe the experimental setting (Section 4). Section 5 presents the experiments we conducted and their results, and Section 6 summarizes the practical takeaways of this study.

2. Background

2.1. Incremental training for SMT

Incremental training methods provide a principled way for updating an SMT model when more data is received, without re-generating the model from scratch. In addition to efficiency, such methods hold the promise to reflect updates immediately, without work interruption, and are therefore of major importance in many scenarios.

Incremental training for MT often makes use of an online version of the Expectation Maximization (EM) algorithm [1]. EM is used for the purpose of aligning the bilingual corpus while computing translation probabilities [2]. In *Online EM*, the model parameters are updated after each example or a small set of examples (*mini-batch*), and not for the entire dataset at once. Naturally, online EM is faster than *batch EM*, but may be less stable.[1]

Ortiz-Matrínez et al. [4] use incremental online EM [5] to update a standard log-linear model. They apply it in the context of Interactive Machine Translation, where conveying to the user the impression of a highly adaptive system is particularly important. A method for incrementally updating SMT models was also proposed within the SMART project [6]. A large set of features, on top of the standard translation features, is extracted from (simulated) post-edited translations. While the weights of the standard features are tuned offline and remain stable, the weights of the new ones are updated after each source-translation pair. Levenberg et al. [7, 8] use *stepwise EM* for updating the translation model parameters. They use IBM Model 1 [2] with HMM alignments [9], collecting counts for translations and alignments and updating them by interpolating the statistics of the old and the new data. We employ and assess an implementation of this algorithm within Moses (see Section 5.7).

2.2. Domain adaptation

Domain adaptation is the task of adapting a statistical model that was trained on a certain domain to perform well on another domain. Generally, *domain* refers to the distribution of the (training or test) instances; in language-based tasks this term may refer to any of topic, style, dialect, genre or a combination of thereof [10].

Domain adaptation is of major importance for SMT, and in particular for spoken language translation, where bilingual training data is often scarce, and models are thus heavily relying on *out-of-domain* corpora for training. Some methods aim to optimize the use of available corpora through data selection – using only the part of the training data that is more similar to the target domain, or by instance-weighting, i.e. giving each example a weight that corresponds to its similarity to the target domain [11, 12, 13]. In [14], such adaptation in performed on-the-fly without assuming the target domain is known in advance. Other methods apply focused domain-specific data acquisition, e.g. by web crawling [15].

In many scenarios, though, little or no *in-domain* data is accessible in advance. It may be attained at a later stage, e.g. via user feedback to the translation, in the form of post-editing. When such data becomes available, it is desirable to update the model with this data without much delay.

[16] start with an in-domain phrase table, which is then filled-up with new entries from other corpora. In- and out-of-domain entries are distinguished with an additional feature. A more explicit separation of domains is found in the mixture models approach. Training data is divided into components according to the different domains. A model (either a translation model or a language model) is trained for each component separately and the models are then weighted and combined to form a complete model [17, 18]. We use this approach in some of the configurations we assess. However, our goal is different: we focus on the capability to perform the updates quickly. Fortunately, as our results show, these considerations often go hand in hand, and methods that work well for domain adaptation are useful also for quick updates.

2.3. SMT model generation

For completeness, we briefly describe the main steps in generating a basic phrase-based SMT model, from a parallel corpus to a tuned model. Our description corresponds to the steps as done in Moses [19], but is typical to most phrase-based SMT systems.

- **Preprocessing**: The model generation process starts with preprocessing of the bilingual parallel (sentence-aligned) corpus, including tokenization, lower-casing, and removal of sentences that are, e.g., very long.
- **Alignment**: Following some file preparation steps, GIZA++ [20], an implementation of the IBM Models, is applied in two directions (source-target and target-source) to produce word alignment within each source-target sentence pair. A symmetrization of the GIZA bi-directional word alignments follows.
- **Phrase table construction**: Based on word alignments, a *translation model* is generated: lexical (word) translation probabilities are computed and phrases are extracted, scored and stored in a phrase table (PT).
- A **reordering table** is constructed to model position change of phrases between the source and the target.[2]
- A **language model** (LM) is generated from the target side of the parallel corpus and possibly additional target language monolingual data.
- Lastly, **tuning** takes place in order to optimize the weights of individual scores (features) within the complete model.

[1] See [3] for a detailed discussion about the variants of online EM.

[2] The abovementioned phrase extraction is also needed for this step; we chose to include it within the translation model generation step since, as explained later, we do not update the reordering model in this work.

Large phrase tables, reordering tables and language models that cannot fit into memory are often binarized for quick loading and access at translation time. Yet, binarization is not feasible when very large tables are concerned. Reducing the size of the tables through filtering based on a given test dataset is not practical in real world scenarios, and is slow to process as well, as it also depends on the size of the tables.

Of the above, alignment is the most time-consuming step; phrase table construction may also requires a substantial amount of time, especially when binarization is performed; tuning involves multiple iterations (typically over 20) in which a development set is translated and evaluated, and is therefore a highly time-consuming task. Indeed, some of the steps can be parallelized, yet not all. For instance, in MGIZA [21], a multi-threaded version of GIZA++, sentences-pairs are aligned in parallel, saving a substantial amount of time; still, parameter estimation is based on counts that are accumulated from all aligned sentences, and is not parallelized. What is often referred to as *batch training* consists of all the above steps applied to the entire data.

Figure 1 shows the relative time required to complete each task, based on an experiment we conducted, with 1 million sentence-pairs for training and 1,000 sentence-pairs for tuning. Both datasets were taken from the Italian-English corpus of Europarl version 7 [22]. As elapsed time depends on the specific machine and its load at the time of measurement, we use the Unix `time` command for obtaining duration information. We look at the accumulated CPU time, which is roughly equivalent to running on a single CPU. For intuition, the alignment task used up approximately 21 CPU hours, which corresponded to about 6 actual hours when running MGIZA with 4 cores. For comparison, under the same machine configuration, alignment of 2,000 sentences took 2.5 CPU minutes, and 10,000 sentences required less than 13 minutes. This experiment was performed on a 64 bit Linux machine, with four 2.67GHz cores and 50GB of RAM.

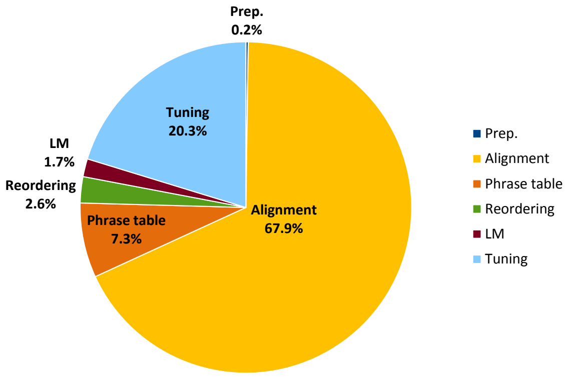

Figure 1: Percentage of the time required by each task of the phrase-based model generation. The times shown here include the binarization of the corresponding model.

3. Quick update configurations

Let's recall the scenario we take interest in: An SMT system is trained based on a large out-of-domain corpus and is meant to be used on a different type of dataset, namely spoken-language texts. The translation service is made available and gradually in-domain data is flowing in. With this data we wish to update the system in the most efficient and effective way. We expect best translations to be produced when we use all the data we have at our disposal; at the same time, we do not wish to carry out intensive processes unnecessarily. We therefore carry out batch updates periodically (long-cycles), and in the interim perform quick, short-cycle updates using the newly obtained data. We wish to identify the most useful configuration – in terms of time and translation quality – for performing such short cycle updates. For that purpose, we examined the following configurations for quick model updates. Each has its pros and cons, as discussed below. Figure 2 depicts their phrase table settings.

1. **OLD-NEW**: In this configuration we use two phrase tables. We maintain all previously obtained ("old") training data, both in-domain and out-of-domain, in one phrase table and the newly obtained data ("new") in a second table. To update the model, we only need to preprocess and align the new data on its own and generate a phrase table from it. This is therefore a very quick way to perform updates.

2. **IN-OUT**: This setting uses two phrase tables as well, but now the out-of-domain data is maintained in one table and the in-domain data in another table. The idea is to allow better model tuning by letting the tuning algorithm give preference to the in-domain table. The drawback is that all in-domain data needs to be processed at every short-cycle update, implying a longer process. As long as in-domain data is limited, this is not an issue. On the contrary, it can contribute to improved alignment quality and phrase table statistics.

3. **3-TABLES**: When in-domain data accumulates, the IN-OUT setting may become too slow. We therefore assess another setting that can potentially combine the benefits of the two above configurations. Here, we use three phrase tables: one for out-of-domain data and two for in-domain. The first among the in-domain tables is used for all previously obtained in-domain data, and the second for the newly obtained data. This way we achieve both separation of in- and out-of-domain data and a quick processing of the new data.

4. **BATCH**: This is a standard setting for phrase-based SMT model generation, used for comparison. The entire training data is concatenated and used together, and a single phrase table is produced. One potential advantage is, as above, an improved alignment quality.

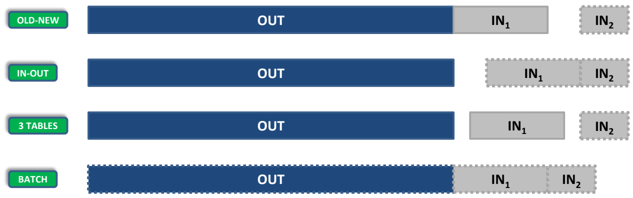

Figure 2: Phrase tables in the different configurations. OUT denotes out-of-domain data; IN$_1$ is in-domain data previously obtained, and IN$_2$ is the in-domain data we have just received and wish to use to update the system with. The dashed lines designate the data that needs to be processed in each update cycle.

In addition to the above, we have experimented with incremental updates via Moses' dynamic suffix array. We describe that in Section 5.7.

Required effort Table 1 details which task needs to be performed in each configuration, and the amount of work that has to be done. We explain the required effort of each configuration through an example, whose timeline is presented in Figure 3: We consider a specific point in time of an operational translation system. This system was trained with 1 million out-of-domain sentence-pairs before any in-domain data was available (S_1 in the figure); over time, 30,000 in-domain bi-sentences were received and the system has been already updated with them in a slow update cycle (S_2). Between the previous slow update and the current point in time, 10,000 in-domain sentences have been obtained, and fed into the system (q_{21}); now we receive 5,000 more, and wish to carry out quick update q_{22}.

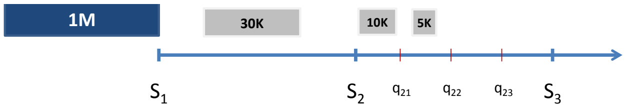

Figure 3: Updates timeline, as described in the example in Section 3. S_i denotes a slow update and q_{ij} a quick update. Boxes represent the available data: dark-shading for out-of-domain and light-shading for in-domain.

Config./ Task	Prep.	Alignment	PT	LM
OLD-NEW	5K	15K	15K	15K
IN-OUT	5K	45K	45K	45K
3-TABLES	5K	15K	15K	15K
BATCH	5K	1,045K	1,045K	1,045K

Table 1: An example of the required effort of each configuration. We consider as "new" all data received since the last slow update.

Here we assume that all data received in between slow updates is small and can be processed together. Preprocessing need not be repeated, but the other steps may perform better given more data. As seen in the table, both OLD-NEW and 3-TABLES require minimal processing. The difference between them is the way the previously-obtained data is stored; IN-OUT requires a more substantial amount of processing, and BATCH requires all the data to be processed from scratch.

So far, our discussion focused on phrase tables. Concerning the LM, Table 1 assumes a setting where the LMs configuration is equivalent to that of the phrase tables. Our experiments showed that – at least for the language pairs we assessed – the reordering model does not significantly affect translation performance; thus, we do not update it in any of the quick update settings. Tuning is discussed in Section 5.6.

4. Setting

4.1. Datasets

We used three datasets in our experiment, two spoken-language parallel corpora, transLectures (TL) and WIT^3 (WIT3 below), and Europarl that represents a large out-of-domain corpus, of non-spoken-language.

transLectures
The first spoken-language dataset was obtained via the transLectures project that is addressing the transcription and translation of scientific video lectures.[3] Translating from English to French, we used the entire datasets that were available at the time of the experiments. These consisted of merely several thousand sentence pairs, that were produced through manual post-editing of the automatic transcription, followed by post-edition of the automatic translation of the transcriptions. The continuous text of the lectures was split into sentences based on long silences in the speech and with a maximal sentence-length constraint. This dataset and its production represent a typical scenario were in-domain spoken language data is scarce, hard to collect and slow to arrive.

- Training set: ∼4,000 English-French sentence pairs.
- Development set: 1,000 bi-sentences, used for tuning.
- Test set: 1,360 sentences-pairs.

WIT3
Our first round of experiments was conducted on the TL data. To confirm the validity of the results across datasets and language pairs, and to allow reproducing our results through a freely available resource, we used another spoken-language dataset, WIT3 [23]. WIT3 (Web Inventory of Transcribed and Translated Talks) is a parallel corpus created from transcription and translation of TED talks.[4] We used a different language pair, Italian to English, and 10-times as much training data as available in the TL dataset.

- Training set: 40,000 sentence-pairs from the Italian-English WIT3 corpus.[5]
- Development set: 1,000 bi-sentences of that corpus.
- Test set: 1,000 bi-sentences from the above corpus.

[3] http://www.translectures.eu/
[4] http://www.ted.com
[5] Downloaded from https://wit3.fbk.eu

Europarl
For each language-pair we used, as part of the training set of most configurations, 1 million Europarl v. 7 bi-sentences.

4.2. Experimental setup

Phrase-based SMT Moses [19] was the translation system used for our experiments. When more than one phrase table was employed, we used the *either* option, meaning that translation options are searched for in either table with no preference to one table over the other, and while not expecting every translation option to be present in both tables.

Alignment Some experiments assessed the use of incremental training and of dynamic suffix arrays. For fair comparison, we used Incremental GIZA [7] in all our experiments rather than GIZA++. However (with the exception of the experiments described in Section 5.7), we did not use its incremental capability.

Language Model We trained 5-gram language models on the target side of the training set(s) using SRILM [24], with modified Kneser-Ney discounting [25].

Tuning Model weights were tuned with batch MIRA [26].

Evaluation We use Smooth (sentence-level) BLEU [27], and report the average score over the test set sentences. All our evaluations were performed on lower case, tokenized texts, using the standard Moses tools for preprocessing.

5. Experiments and results

In this section we present experiments conducted with the TL and WIT3 datasets, and their results.

5.1. Batch updates

We start by providing the results of "regular" batch updates, where the entire training set is used as a single corpus. The first row of each dataset in Table 2 shows the baseline, when no new data is used. This is the starting point of a system that was trained on a large amount of out-of-domain data; in the second row we show the result when 4K (TL) or 40K (WIT3) bi-sentence are used to update the phrase table (i.e. the translation model), but not the LM or the reordering table; the third row shows results of updating all three.

Dataset	Configuration	BLEU
transLectures	Baseline	23.9
	Batch, PT only	27.9
	Batch, complete	28.3
WIT3	Baseline	29.4
	Batch, PT only	30.9
	Batch, complete	30.7

Table 2: Results of batch updates.

Unsurprisingly, the addition of the new in-domain data to the phrase table greatly improves the translation quality; updating the LM and reordering tables adds a bit more on top of that for transLectures. As mentioned, initial experiments showed that reordering had insignificant impact on results, and improvements may thus be mostly attributed to the LM update; we therefore assessed the performance of all following models without updating the reordering table.

5.2. Quick updates

We now evaluate the performance of quick update models. In these experiments we assume we are about to perform an update equivalent to q_{21} in Figure 3. That is, we have received some in-domain data earlier, performed a slow update since, and now receive additional in-domain data, which we use to quickly update the model. Table 3 shows the results of the three configurations where only the phrase table is being updated with the new data, i.e. the language model and the reordering model are not updated at all. While using the same amount of data as for the batch updates in Table 2, and even with this partial model update, each of these configurations outperforms the batch update, over the two datasets. This result is consistent with prior work on domain adaptation (e.g. [17, 18]), but the important aspect that we are concerned with is that this update is **much** faster. Instead of processing over a million sentence pairs, only up to 4,000 (TL) or 40,000 (WIT3) need to be handled.

Dataset	Configuration	BLEU
transLectures	Old-New	29.4
	In-out	29.7
	3-tables	30.2
WIT3	Old-New	31.2
	In-out	31.7
	3-tables	31.2

Table 3: Quick updates, where only phrase tables are updated.

5.3. Quick updates of the language model

Next, we evaluate the performance when the LM is also updated. We use multiple LMs, separated the same way as the phrase tables: Old-New and In-out use two LMs, and 3-tables, uses three. This allows quick update of this model as well. Table 4 shows the results of this set of experiments.

Dataset	Configuration	BLEU
transLectures	Old-New	31.2
	In-out	31.8
	3-tables	31.6
WIT3	Old-New	32.3
	In-out	33.1
	3-tables	32.3

Table 4: Quick update results, with matching LM and phrase table configurations.

In all cases, results are improved relative to updating only the phrase-table (Table 3). Updating the LM was expected to help, yet here we experimentally see that even a quick LM update achieves significant improvements, and is useful for our goal. The best configuration is IN-OUT for both datasets. This is the slowest of the three configurations; hence, depending on the data size, the other options may also be considered, and in particular the 3-TABLES option.

We have seen that quick LM update on top of the phrase table helps; we now wish to verify that updating the LM alone is not sufficient. Table 5 shows two such experiments on the WIT3 dataset. In the first, the target side of the WIT3 training corpus was added to the Europarl corpus to generate a single LM; in the second, the same WIT3 data was used to produce a separate LM. Note that the first among these is not a quick update per-se. Yet, LM generation is much faster than phrase table construction; if the performance is competitive, this can also be an option to consider.

As it turns out, training of a single LM with the additional data did not improve results relative to the baseline. Possibly, in-domain data (consisting of less than 4% or the training data in this case) is diluted in the entire set. More importantly, we see that the quicker update where the LMs are separated, is better. The performance is similar to the configuration where only the phrase table is updated but is inferior to all configurations where both models are updated.

Configuration	**BLEU**
Single LM	29.4
Separate LMs	31.4

Table 5: WIT3, updating only the language model.

5.4. Separating the LMs for batch training

Following the above results where LM separation helps, we assess this option with batch updates as well. Here we maintain a single phrase table, and separate only the LMs. This setting is still slow, yet somewhat quicker than a complete batch update since the previous LM need not be generated, just the new one. The more time-consuming steps of alignment and phrase table construction are still necessary.

Dataset	Configuration	BLEU
TL	BATCH, single LM	28.3
	BATCH, separate LMs	31.6
WIT3	BATCH, single LM	30.7
	BATCH, separate LMs	32.6

Table 6: Comparison of batch configurations, with and without separating the LMs for in/out-of domain data. The single-LM configurations are the same ones shown in Table 2.

Table 6 shows that LM separation significantly improves results also when the PT is batch-trained, and while not considered quick, it is useful to separate the LMs between domains also in this case. The results are still inferior to those obtained by a complete (quick) in-out separation, and are just slightly better than other quick configurations in Table 4.

5.5. No-adaptation

So far our results included two types of datasets. We also wish to understand the effect of the different configurations when only a single domain is concerned. In this setting, IN-OUT and 3-TABLES are not relevant, only OLD-NEW is, with or without phrase table and LM separation. The TL data is too small for this experiment, and we use only WIT3, training a model with $30K$ sentence-pairs and updating it with additional $10K$. The results are shown in Table 7. The first row shows the baseline result before the $10K$ dataset is used, and the second shows the result where all data is trained together in a batch setting. The next two rows show quicker updates: the first – and the quickest – where both phrase table and LMs are separated between old and new data, and the second, where only the phrase tables are separated.

Configuration	BLEU
Baseline	28.2
BATCH	29.2
OLD-NEW	28.5
OLD-NEW, single LM	28.9

Table 7: WIT3 results, where only in-domain data is used.

Now that domain adaptation is no longer a factor, BATCH achieves the best result. Here, we can see the benefit of generating models using the entire data. Quick updates are not far behind, and are faster to carry out. In this setting, separating LMs of the same domain is not useful, and a better model is obtained when more data is used. Notice, though, that these scores are inferior to those obtained in the previous experiments. Out-of-domain data is very useful, and as this is case, quick update methods should still be considered.

5.6. Tuning

Each of the above models was tuned individually before being evaluated. Still, separate experiments show that tuning is not strictly required for every update. Tuning is likely necessary when a configuration is changing, e.g. in terms of components, the data split between them, or the balance between the datasets. When these remain relatively fixed, and a small amount of data is added, tuning may be skipped. Two examples are shown in Table 8. In each, the first row shows the result of a model trained with the Europarl corpus and with partial TL data. The next two models (rows 2 & 3 for each experiment) use additional 1,000 bi-sentences and differ only in the tuning – while the first was re-tuned, the second was not, and used instead the tuned weights of the baseline model. We see that by re-tuning we obtain a small gain in performance; yet, we greatly lose in terms of time. In many cases, then, tuning can be skipped for intermediate updates, and reserved only for slow updates.

Setting	Configuration	BLEU
TL, 2K; IN-OUT	Baseline	27.76
	Re-tuned	28.45
	Not re-tuned	28.31
TL, 4K; OLD-NEW	Baseline	28.51
	Re-tuned	29.37
	Not re-tuned	29.19

Table 8: Tuning with all available data vs. using a model with the same configuration tuned with a smaller amount of data.

So far we have seen several options for model updates that can be applied very quickly. Using the Moses server, once an updated model is ready, it can be loaded into memory practically instantaneously, replacing a previous instance of the server that was loaded with a previous model. That is, as long as all large models are binarized. We can assume binarizing is done during slow updates, and that small models can be loaded quickly and fit into memory easily. With IN-OUT we run into the risk that in-domain data also becomes large; this is not an issue for the 3-TABLES configuration, where the processed data always remains small.

5.7. Incremental training and dynamic suffix arrays

We have extensively experimented with incremental GIZA, and with updates through the dynamic suffix array in Moses.[6] Suffix arrays constitute an alternative to phrase tables, where the entire training data is maintained in memory rather than in a phrase table [28]. Dynamic suffix arrays [7] further enable inserting or deleting training instances, thus updating the translation model without retraining. Although very efficient in comparison to batch training, the process of incrementally updating a model with these tools is not as fast as one would expect. Apart from preprocessing and alignment of the new data (which are required in any case), it requires, prior to the alignment, updating the vocabulary and cooccurrence files, as well as the HMM probabilities. These statistic updates, which operate over the respective files of the entire data, need to be done independently of the size of the new data. It is therefore not efficient to run it per sentence, but rather per mini-batch. Once the new data has been aligned, inserting each bi-sentence into the suffix array is needed to have the translation system updated. Apparently, this is a time consuming process and cannot be considered a real-time update. Creating a phrase table for the new data, and loading another instance of the Moses server, is significantly faster.

We have run multiple comparative experiments with phrase tables vs. suffix arrays, and with combinations of them both, and observed a significant drop in results whenever the suffix array was used, with or without Incremental GIZA. For instance, for the same setting in Table 2, row 1, the BLEU score dropped from 23.9 to 20.8 when the phrase table was replaced with a suffix array. A possible reason is the fact that the inverse translation probabilities are missing in this data structure. Moreover, when an update takes place, the translation server becomes unusable, maintaining the suffix array in memory takes up a large amount of memory and the updated model cannot be saved into disk, but needs to be reconstructed later. Further, updates to the LM are not supported, although this issue was addressed in [8]. All these make this data-structure currently difficult to use or rely on.[7]

The potential advantage of principled incremental training is obvious. Taking into account the previously accumulated data is expected to produce better statistics; doing so while maintaining the system live and constantly updated is a highly sought-after goal. Yet, aligning all data, regardless of the domain, is not always beneficial. Thus, once such tools are stable and efficient, quick updates may be used in conjunction with incremental training and suffix arrays. For instance, out-of-domain data can be maintained in a phrase table, while in-domain data that needs updating is loaded into a suffix array. Preparations for alignment are longer, but the advantage in comparison to IN-OUT is that only alignment of the new data is necessary, but not phrase-table generation.

6. Conclusions

This work focused on identifying simple configurations of phrase-based SMT systems, which allow updating the underlying model quickly when new training data becomes available. We have emphasized the applicability to domain adaption, which is particularly relevant for spoken language applications, where seed in-domain parallel resources are typically scarce or altogether absent. Still, we have shown that this type of updates is suitable also for single-domain settings. We assessed multiple configurations, some of which are based on proven methods from domain adaptation research, to highlight the preferred ones both in terms of translation quality and of processing speed. We described how quick updates can be integrated into the lifecycle of an operational SMT system, enabling efficiently maintaining translation quality while keeping the system up and up-to-date.

Our results show that quick updates are competitive with batch retraining on corpus concatenation, a strong baseline, while being orders of magnitude faster. We have seen that a complete separation of in- and out-of-domain data usually results with best translation quality; yet, this option may become slow over time. The 3-TABLES configuration we proposed solves this issue, albeit at the price of some drop in performance. A potential improvement for this configuration, that we intend to investigate, is to reserve some, moderate size in-domain data for training together with the new data, benefiting from the potential improved alignment, while still keeping the update fast.

[6] We thank Abby Levenberg for his support at this part of the study.

[7] In summer 2013, a new implementation of the dynamic suffix array has been introduced in Moses, where all standard 5 features are computed. Some of the above issues may have been handled. To the best of our knowledge this is still work-in-progress and we have not experimented with it so far.

7. Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 287755.

8. References

[1] O. Cappé and E. Moulines, "On-line expectation–maximization algorithm for latent data models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 71, no. 3, pp. 593–613, 2009.

[2] P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer, "The mathematics of statistical machine translation: parameter estimation," *Comput. Linguist.*, vol. 19, no. 2, pp. 263–311, June 1993.

[3] P. Liang and D. Klein, "Online em for unsupervised models," in *Proc. of NAACL*, 2009.

[4] D. Ortiz-Martínez, I. García-Varea, and F. Casacuberta, "Online learning for interactive statistical machine translation," in *Proc. of HLT*, 2010.

[5] R. Neal and G. Hinton, "A view of the EM algorithm that justifies incremental, sparse, and other variants," in *Learning in Graphical Models*, M. I. Jordan, Ed. MIT Press, 1999, pp. 355–368.

[6] N. Cesa-Bianchi, G. Reverberi, and S. Szedmak, "Online learning algorithms for computer-assisted translation." in *Deliverable D4.2, EU Project SMART*, 2008.

[7] A. Levenberg, C. Callison-Burch, and M. Osborne, "Stream-based translation models for statistical machine translation," in *Proc. of HLT-NAACL*, 2010.

[8] A. Levenberg, "Stream-based statistical machine translation," Ph.D. dissertation, University of Edinburgh, 2011.

[9] S. Vogel, H. Ney, and C. Tillmann, "Hmm-based word alignment in statistical translation," in *Proc. of COLING*, 1996.

[10] B. Chen, R. Kuhn, and G. Foster, "Vector space model for adaptation in statistical machine translation," in *Proc. of ACL*, 2013.

[11] Y. Lu, J. Huang, and Q. Liu, "Improving statistical machine translation performance by training data selection and optimization," in *Proc. of EMNLP-CoNLL*, 2007.

[12] G. F. Foster, C. Goutte, and R. Kuhn, "Discriminative instance weighting for domain adaptation in statistical machine translation," in *EMNLP*, 2010.

[13] A. Axelrod, X. He, and J. Gao, "Domain adaptation via pseudo in-domain data selection," in *Proc. of EMNLP*, 2011.

[14] L. Gong, A. Max, and F. Yvon, "Towards contextual adaptation for any-text translation," in *Proc. of IWSLT*, 2012.

[15] P. Pecina, A. Toral, V. Papavassiliou, P. Prokopidis, and J. van Genabith, "Domain Adaptation of Statistical Machine Translation using Web-Crawled Resources: A Case Study," in *Proc. of EAMT*, 2012.

[16] A. Bisazza, N. Ruiz, and M. Federico, "Fill-up versus interpolation methods for phrase-based SMT adaptation," in *Proc. of IWSLT*, 2011.

[17] G. Foster and R. Kuhn, "Mixture-model adaptation for smt," in *Proc. of WMT*, 2007.

[18] P. Koehn and J. Schroeder, "Experiments in domain adaptation for statistical machine translation," in *Proc. of WMT*, 2007.

[19] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst, "Moses: Open source toolkit for statistical machine translation," in *Proc. of ACL Demo and Poster Sessions*, 2007.

[20] F. J. Och and H. Ney, "Improved statistical alignment models," in *Proc. of ACL*, 2000.

[21] Q. Gao and S. Vogel, "Parallel implementations of word alignment tool," in *Proc. of the ACL Software Engineering, Testing, and Quality Assurance Workshop*, 2008.

[22] P. Koehn, "Europarl: A parallel corpus for statistical machine translation," in *Proc. of MT Summit*, 2005.

[23] M. Cettolo, C. Girardi, and M. Federico, "Wit^3: Web inventory of transcribed and translated talks," in *Proc. of EAMT*, 2012.

[24] A. Stolcke, "SRILM - an extensible language modeling toolkit," in *Proc. of Interspeech*, 2002.

[25] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," in *Proc. of ACL*, 1996.

[26] C. Cherry and G. Foster, "Batch tuning strategies for statistical machine translation," in *Proc. of NAACL-HLT*, 2012.

[27] D. Cer, M. Galley, D. Jurafsky, and C. D. Manning, "Phrasal: A statistical machine translation toolkit for exploring new model features," in *Proc. of the NAACL HLT Demonstration Session*, 2010.

[28] C. Callison-Burch and C. Bannard, "A compact data structure for searchable translation memories," in *Proc. of EAMT*, 2005.