LIG-CRIStal System for the WMT17 Automatic Post-Editing Task
Alexandre Bérard, Olivier Pietquin, Laurent Besacier

To cite this version:
Alexandre Bérard, Olivier Pietquin, Laurent Besacier. LIG-CRIStal System for the WMT17 Automatic Post-Editing Task. Second conference on machine translation (WMT17) during EMNLP 2017, Sep 2017, Copenhague, Denmark. hal-01580881

HAL Id: hal-01580881
https://hal.archives-ouvertes.fr/hal-01580881
Submitted on 3 Sep 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract

This paper presents the LIG-CRIStAL submission to the shared Automatic Post-Editing task of WMT 2017. We propose two neural post-editing models: a monosource model with a task-specific attention mechanism, which performs particularly well in a low-resource scenario; and a chained architecture which makes use of the source sentence to provide extra context. This latter architecture manages to slightly improve our results when more training data is available. We present and discuss our results on two datasets (en-de and de-en) that are made available for the task.

1 Introduction

It has become quite common for human translators to use machine translation (MT) as a first step, and then to manually post-edit the translation hypothesis. This can result in a significant gain of time, compared to translating from scratch (Green et al., 2013). Such translation workflows can result in the production of new training data, that may be re-injected into the system in order to improve it. Common ways to do so are retraining, incremental training, translation memories, or automatic post-editing (Chatterjee et al., 2015).

In Automatic Post-Editing (APE), the MT system is usually considered as a blackbox: a separate APE system takes as input the outputs of this MT system, and tries to improve them. Statistical Post-Editing (SPE) was first proposed by Simard et al. (2007). It consists in training a Statistical Machine Translation (SMT) system (Koehn et al., 2007), to translate from translation hypotheses to a human post-edited version of those. Béchara et al. (2011) then proposed a way to integrate both the translation hypothesis and the original (source language) sentence. More recent contributions in the same vein are (Chatterjee et al., 2016; Pal et al., 2016).

When too little training data is available, one may resort to using synthetic corpora: with simulated PE (Potet et al., 2012), or round-trip translation (Junczys-Dowmunt and Grundkiewicz, 2016).

Recently, with the success of Neural Machine Translation (NMT) models (Sutskever et al., 2014; Bahdanau et al., 2015), new kinds of APE methods have been proposed that use encoder-decoder approaches (Junczys-Dowmunt and Grundkiewicz, 2016, 2017; Libovický et al., 2016; Pal et al., 2017; Hokamp, 2017), in which a Recurrent Neural Network (RNN) encodes the source sentence into a fixed size representation (encoder), and another RNN uses this representation to output a new sequence. These encoder-decoder models are generally enhanced with an attention mechanism, which learns to look at the entire sequence of encoder states (Bahdanau et al., 2015; Luong et al., 2016).

We present novel neural architectures for automatic post-editing. Our models learn to generate sequences of edit operations, and use a task-specific attention mechanism which gives information about the word being post-edited.

1.1 Predicting Edit Operations

We think that post-editing should be closer to spelling correction than machine translation. Our work is based on Libovický et al. (2016), who train a model to predict edit operations instead of words. We predict 4 types of operations: KEEP, DEL, INS(word), and EOS (the end of sentence marker). This results in a vocabulary with three symbols plus as many symbols as there are possible insertions.
A benefit of this approach is that, even with little training data, it is very straightforward to learn to output the translation hypothesis as is (MT baseline). We want to avoid a scenario where the APE system is weaker than the original MT system and only degrades its output. However, this approach also has shortcomings, that we shall see in the remainder of this work.

**Example** If the MT sequence is "The cats is grey", and the output sequence of edit ops is "KEEP DEL INS(cat) KEEP KEEP INS(.)", this corresponds to doing the following sequence of operations: keep "The", delete "cats", insert "cat.", keep "is", keep "grey", insert ". " The final result is the post-edited sequence "The cat is grey . "

We preprocess the data to extract such edit sequences by following the shortest edit path (similar to a Levenshtein distance, without substitutions, or with a substitution cost of +∞).

### 1.2 Forced Attention

State-of-the-art NMT systems (Bahdanau et al., 2015) learn a global attention model, which helps the decoder look at the relevant part of the input sequence each time it generates a new word. It is defined as follows:

\[
\text{attn}_{\text{global}}(h, s_t) = \sum_{i=1}^{A} a_i^t h_i \quad (1)
\]

\[
a_i^t = \text{softmax}(e_i^t) \quad (2)
\]

\[
e_i^t = v^T \tanh(W_1 h_i + W_2 s_t + b_2) \quad (3)
\]

where \(s_t\) is the current state of the decoder, \(h_i\) is the \(i\)th state of the encoder (corresponding to the \(i\)th input word). \(A\) is the length of the input sequence. \(W_1, W_2\) and \(b_2\) are learned parameters of the model. This attention vector is used to generate the next output symbol \(w_t\) and to compute the next state of the decoder \(s_{t+1}\).

However, we don’t predict words, but edit operations, which means that we can do stronger assumptions as to how the output symbols align with the input. Instead of a soft attention mechanism, which can look at the entire input and uses the current decoder state \(s_t\) to compute soft weights \(a_i\); we use a hard attention mechanism which directly aligns \(t\) with \(i\). The attention vector is then \(\text{attn}_{\text{forced}}(h, s_t) = h_i\).

The \(t \rightarrow i\) alignment is pretty straightforward: \(i\) is the number of KEEP and DEL symbols in the decoder’s past output \((w_1, \ldots, w_{t-1})\) plus one.

### 1.3 Chaining Encoders

The model we proposed does not make any use of the source side \(\text{SRC}\). Making use of this information is not very straightforward in our framework. Indeed, we may consider using a multi-encoder architecture (Zoph and Knight, 2016; Junczys-Dowmunt and Grundkiewicz, 2017), but it does not make much sense to align an edit operation with the source sequence, and such a model struggles to learn a meaningful alignment.

We propose a chained architecture, which combines two encoder-decoder models (see fig. 1). A first model \(\text{SRC} \rightarrow \text{MT}\), with a global attention mechanism, tries to mimic the translation process that produced \(\text{MT}\) from \(\text{SRC}\). The attention vectors of this first model summarize the part of the \(\text{SRC}\) sequence that led to the generation of each \(\text{MT}\) token. A second model \(\text{MT} \rightarrow \text{OP}\) learns to post-edit and uses a forced attention over the \(\text{MT}\) sequence, as well as the attention vectors over \(\text{SRC}\) computed by the first system. Both models are trained jointly, by optimizing a sum of both losses.

<table>
<thead>
<tr>
<th>Task</th>
<th>Train</th>
<th>Dev</th>
<th>Test 2016</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>en-de</td>
<td>23k</td>
<td>1000</td>
<td>2000</td>
<td>500k</td>
</tr>
<tr>
<td></td>
<td>(12k + 11k)</td>
<td></td>
<td></td>
<td>4M</td>
</tr>
<tr>
<td>de-en</td>
<td>24k</td>
<td>1000</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

Table 1: Size of each available corpus (number of \(\text{SRC}, \text{MT}, \text{PE}\) sentence tuples).
2 Experiments

This year’s APE task consists in two sub-tasks: a task on English to German post-editing in the IT domain (en-de), and a task on German to English post-editing in the medical domain (de-en). Table 1 gives the size of each of the corpora available. The goal of both tasks is to minimize the HTER (Snover et al., 2006) between our automatic post-editing output, and the human post-editing output.

The en-de 23k training set is a concatenation of last year’s 12k dataset, and a newly released 11k dataset. A synthetic corpus was built and used by the winner of last year’s edition (Junczys-Dowmunt and Grundkiewicz, 2016), and is available this year as additional data (500k and 4M corpora).

For the en-de task, we limit our use of external data to the 500k corpus. For the de-en task, we built our own synthetic corpus, using a technique similar to (Junczys-Dowmunt and Grundkiewicz, 2016).

2.1 Synthetic Data

Desiderata We used similar data selection techniques as Junczys-Dowmunt and Grundkiewicz (2016), applied to the de-en task. However, we are very reticent about using as much parallel data as the authors did. We think that access to such amounts of parallel data is rarely possible, and the round-trip translation method they used too cumbersome and unrealistic. To show a fair comparison, this paper should show APE scores when translating from scratch with an MT system trained with all this parallel data.

To mitigate this, we decided to limit our use of external data to monolingual English (common-crawl). So, the only parallel data we use is the de-en APE corpus.

PE side Similarly to Junczys-Dowmunt and Grundkiewicz (2016) we first performed a coarse filtering of well-formed sentences of common-crawl. After this filtering step, we obtained about 500M lines. Then, we estimated a trigram language model on the PE side of the APE corpus, and sorted the 500M lines according to their log-score divided by sentence length. We then kept the first 10M lines. This results in sentences that are mostly in the medical domain.

MT and SRC sides Using this English corpus, and assuming its relative closeness to the PE side of the APE corpus, we now need to generate SRC and MT sequences. This is where our approach differs from the original paper.

Instead of training two SMT systems PE → SRC and SRC → MT on huge amounts of parallel data, and doing a round-trip translation of the monolingual data, we train two small PE → SRC and PE → MT Neural Machine Translation systems on the APE data only.

An obvious advantage of this method is that we do not need external parallel data. The NMT systems are also fairly quick to train, and evaluation is very fast. Translating 10M lines with SMT can take a very long time, while NMT can translate dozens of sentences at once on a GPU.

However, there are strong disadvantages: for one, our SRC and MT sequences have a much poorer vocabulary as those obtained with round-trip translation (because we only get words that belong to the APE corpus). Yet, we hope that the richer target (PE) may help our models learn a better language model.

TER filtering Similarly to Junczys-Dowmunt and Grundkiewicz (2016), we also filter the triples to be close to the real PE distribution in terms of TER statistics. We build a corpus of the 500k closest tuples. For each tuple in the real PE corpus,
Table 2: Top 8 edit ops in the target side of the training set for en-de (top), and most generated edit ops by our primary (500k + 23k) system on dev set (bottom).

<table>
<thead>
<tr>
<th>Token</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEEP</td>
<td>326581</td>
<td>66.9%</td>
</tr>
<tr>
<td>DEL</td>
<td>76725</td>
<td>15.7%</td>
</tr>
<tr>
<td>&quot;</td>
<td>5170</td>
<td>1.1%</td>
</tr>
<tr>
<td>,</td>
<td>3249</td>
<td>0.7%</td>
</tr>
<tr>
<td>die</td>
<td>2461</td>
<td>0.5%</td>
</tr>
<tr>
<td>der</td>
<td>1912</td>
<td>0.4%</td>
</tr>
<tr>
<td>zu</td>
<td>1877</td>
<td>0.4%</td>
</tr>
<tr>
<td>werden</td>
<td>1246</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

we select a random subset of 1000 tuples from the synthetic corpus and pick the tuple whose euclidean distance with the real PE tuple is the lowest. This tuple cannot be selected again. We loop over the real PE corpus until we obtain a filtered corpus of desirable size (500k).

2.2 Experimental settings

We trained mono-source forced models, as well as chained models for both APE tasks. We also trained mono-source models with a global attention mechanism, similar to Libovický et al. (2016) as a measure of comparison to our forced models.

For en-de, we trained two sets of models (with the same configuration) on the 12k train set (to compare with 2016 competitors), and on the new (23k) train set.

The encoders are bidirectional LSTMs of size 128. The embeddings have a size of 128. The first state of the decoder is initialized with the last state of the forward encoder (after a non-linear transformation with dropout). Teacher forcing is used during training (instead of feeding the previous generated output to the decoder, we feed the ground truth). Like Bahdanau et al. (2015), there is a max-out layer before the final projection.

We train our models with pure SGD with a batch size of 32, and an initial learning rate of 1.0. We decay the learning rate by 0.8 every epoch for the models trained with real PE data, and by 0.5 every half epoch for the models that use additional synthetic data. The models are evaluated periodically on a dev set, and we save checkpoints for the best TER scores.

We manually stop training when TER scores on the dev set stop decreasing, and use the best checkpoint for evaluation on the test set (after about 50k steps for the small training sets, and 120k steps for the larger ones).

Unlike Junczys-Dowmunt and Grundkiewicz (2016), we do not use subword units, as we found them not to be beneficial when predicting edit operations. For the larger datasets, our vocabularies are limited to the 30,000 most frequent symbols.

Our implementation uses TensorFlow (Abadi et al., 2015), and runs on a single GPU.¹

2.3 Results & Discussion

As shown in table 3, our forced (contrastive 1) system gets good results on the en-de task, in limited data conditions (12k or 23k). It improves over the MT and SPE baselines, and over the global attention baseline (Libovický et al., 2016). The chained model, which also uses the source sentence, is able to harness larger volumes of data, to obtain yet better results (primary model). However, it lags behind large word-based models trained on larger amounts of data (Junczys-Dowmunt and Grundkiewicz, 2016, 2017; Hokamp, 2017).

Figure 2 compares alignments performed by our attention models. We see that the global attention model struggles to learn a meaningful alignment on a small dataset (12k). When more training data is available (23k), it comes closer to our forced alignment.

We see that our good results on en-de do not transfer well to de-en (see table 4). The BLEU scores are already very high (about 16 points above those of the en-de data, and 10 points above the best APE outputs for en-de). This is probably due to the translation direction being reversed (because of its rich morphology, German is a much harder target that English). The results obtained with a vanilla SMT system (SPE) seem to confirm this difficulty.

¹Our source code, and the configurations used in the experiments are available here: https://github.com/eske/seq2seq/tree/APE

²More results are published on the web page of the task: http://statmt.org/wmt17/ape-task.html
The only reason why our de-en systems are able to not deteriorate the baseline, is that they only learned to do nothing, by producing arbitrarily long sequences of KEEP symbols. Furthermore, we see that the best results are obtained very early in training, before the models start to overfit and deteriorate the translation hypotheses on the dev set (see steps column). The difference between our scores on the de-en dataset is not statistically significant, therefore we cannot draw conclusions as to which model is the best. Furthermore, it turns out that our models output almost only KEEP symbols, resulting in sequences almost identical to the MT input, which explains why the scores are so close to those of the baseline (see table 5).

Adding substitutions is not particularly useful as it leads to even more data sparsity: it doubles the vocabulary size, and results in less DEL symbols, and less training feedback for each individual insertion.

**Future work** One major problem when learning to predict edit ops instead of words, is the class imbalance. There are much more KEEP symbols in the training data as any other symbol (see tables 2 and 5). This results in models that are very good at predicting KEEP tokens (do-nothing scenario), but very cautious when producing other symbols. This also results in bad generalization as most symbols appear only a couple of times in the training data.

We are investigating ways to get a broader training signal when predicting KEEP symbols. This can be achieved either by weight sharing, or by multi-task training (Luong et al., 2016).

Another direction that we may investigate, is how we obtain sequences of edit operations (from PE data in another form). Our edit operations are extracted artificially by taking the shortest edit path between MT and PE. Yet, this does not necessarily correspond to a plausible sequence of operations done by a human. One way to obtain more realistic sequences of operations, would be to collect finer-grained data from human post-editors: key strokes, mouse movements and clicks could be used to reconstruct the ‘true’ sequence of edit operations.

Finally, we chose to work at the word level, when a human translator often works at the character level. If a word misses a letter, he won’t delete the entire word and write it back. However, working with characters poses new challenges: longer sequences means longer training time, and more memory usage. Also, it is easier to learn semantics with words (a character embedding does less sense). Yet, using characters means more training data, and less sparse data, which could be very useful in a post-editing scenario.
Table 4: Results on the de-en task. Because the test set was not available before submission, we used a small part (1000 tuples) of the training set as a train-dev set. This set was used for selecting the best models, while the provided dev set was used for final evaluation of our models. The 500k + 24k corpus is a concatenation of our synthetic corpus with the 24k corpus oversampled 10 times.

<table>
<thead>
<tr>
<th>Model</th>
<th>PE attention</th>
<th>Data</th>
<th>train-dev TER</th>
<th>dev TER</th>
<th>test 2017 TER</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>none</td>
<td>24k</td>
<td>16.11</td>
<td>15.58</td>
<td>15.55</td>
<td>79.54</td>
</tr>
<tr>
<td>SPE</td>
<td>24k</td>
<td></td>
<td>15.74</td>
<td>15.58</td>
<td>15.47</td>
<td>79.28</td>
</tr>
<tr>
<td>Best 2017 (FBK)</td>
<td>24k + ?</td>
<td></td>
<td>15.29</td>
<td>15.55</td>
<td>15.47</td>
<td>79.82</td>
</tr>
<tr>
<td>Mono-source</td>
<td>global</td>
<td>24k</td>
<td>16.06</td>
<td>15.57</td>
<td>15.62</td>
<td>79.48</td>
</tr>
<tr>
<td></td>
<td>forced (contr. 1)</td>
<td></td>
<td>16.05</td>
<td>15.63</td>
<td>15.68</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td>forced (contr. 2)</td>
<td></td>
<td>16.02</td>
<td>15.67</td>
<td>15.53</td>
<td>79.46</td>
</tr>
<tr>
<td>Chained</td>
<td>forced (primary)</td>
<td>500k + 24k</td>
<td>15.98</td>
<td>15.67</td>
<td>15.53</td>
<td>79.46</td>
</tr>
</tbody>
</table>

Table 5: Top 8 edit ops in the target side of the training set for de-en (left), and most generated edit ops by our primary system on train-dev (right).
References


