Treatment of Markup in Statistical Machine Translation

Mathias Müller
Institute of Computational Linguistics, University of Zurich
mmueller@cl.uzh.ch

Abstract

We present work on handling XML markup in Statistical Machine Translation (SMT). The methods we propose can be used to effectively preserve markup (for instance inline formatting or structure) and to place markup correctly in a machine-translated segment. We evaluate our approaches with parallel data that naturally contains markup or where markup was inserted to create synthetic examples. In our experiments, hybrid reinsertion has proven the most accurate method to handle markup, while alignment masking and alignment reinsertion should be regarded as viable alternatives. We provide implementations of all the methods described and they are freely available as an open-source framework[1].

1 Introduction

It is very common for machine translation to be used in workflows where the source documents contain XML markup. If a document was originally written in Microsoft Word, then in a line like

```
Ich bitte Sie, sich zu einer
Schweigeminute zu erheben.

[Please rise, then, for this
minute's silence.]
```

the inline formatting (**boldface**) will internally be represented as inline XML markup, similar to:

```
Ich bitte Sie, sich zu einer
Schweigeminute zu <b>erheben</b>.
```

[1] https://gitlab.cl.uzh.ch/mt/mtrain

Before translation, such a document would probably be converted to a more flexible and interoperable format that is ubiquitous in the translation industry, XLIFF, which is also an XML standard.

Nevertheless, inline XML elements will remain in the source segments and in theory could actually be sent to a machine translation system. But in practice, standard machine translation systems are unable to properly deal with markup and delegate markup handling to downstream applications like computer-assisted translation (CAT) tools. For instance, the machine translation framework Moses (Koehn et al., 2007) does not have a standard solution for markup handling.

Using a standard, phrase-based SMT system trained with Moses, the translation of markup breaks as early as during tokenization. Standard tokenization is not aware of XML markup and will tear apart XML element tags:

```
Ich bitte Sie , sich zu einer
Schweigeminute zu < b > erheben <
/ b > .
```

No subsequent step during translation will be able to undo the damage and since the XML standard enforces strict rules, the output is very likely a malformed XML fragment. But even if tokenization were aware of XML markup (we provide an implementation of markup-aware tokenization) another problem remains: XML markup does not need to be translated at all since it has clear-cut, language-independent semantics and a statistical system should not be trusted to copy the markup to the target segment unchanged.

So, if a machine translation system is given a source segment that contains inline markup, it should be able to detect the markup and not treat it as text. But simply stripping the markup from the source segment is not satisfactory. If, for instance, a translation system would offer

```
Please rise, then, for this
minute's silence.
```

as a translation, we argue that part of the information present in the source segment (the formatting encoded in the markup tags and) was "lost in translation".

From the point of view of translators, losing the markup during translation has inconvenient consequences. In many translation projects, automatic pre-translation of the source segments is an obligatory step and human translators, instead of translating from scratch, will post-edit the pre-translations. There is reason to believe that wrongly translated markup has an impact on *translator productivity* (OBrien, 2011).

Tezcan and Vandeghinste (2011, 56) argue that an MT system should handle XML markup correctly to avoid inefficient translation workflows. In the same vein, Joanis et al. (2013, 74) say that "post-editing SMT output without the formatting information found in the source may represent a serious loss of productivity". Parra and Arcedillo (2015, 142) state "that inline tags have a big impact on productivity, a fact which is not reflected in any of the known metrics and which has not yet received much attention in research".

We agree with this assessment and would like to work towards the goal of implementing markup handling in standard machine translation frameworks. Several solutions have been put forward, but there is no consensus as to which strategy should be employed in standard use cases. Studies that *compare* different approaches are currently lacking.

In order to facilitate those comparisons, we have implemented different markup handling strategies in the same machine translation framework. We have then carried out experiments to gauge the usefulness of each markup strategy, which we will describe in the remainder of this paper.

2 Related Work

Known methods to handle markup in machine translation belong to one of two general paradigms:

- **reinsertion:** markup is stripped from segments prior to training and translation, and reinserted after translation.
- **masking:** markup is not removed entirely, but replaced with a placeholder (a "mask") before training and translation. After translation, the original content is restored.

Both methods ensure that the actual markup is hidden during training and decoding. In the case of our introductory example that includes two XML element tags and :

```
Ich bitte Sie, sich zu einer
Schweigeminute zu <b>erheben</b>.
```

reinsertion would remove markup from the segment alltogether:

```
Ich bitte Sie, sich zu einer
Schweigeminute zu erheben.
```

while masking would replace the tags with placeholders (appearance of mask token may vary):

```
Ich bitte Sie, sich zu einer
Schweigeminute zu __MASK__ erheben
__MASK__ .
```

Du et al. (2010) present three methods to process TMX markup in an SMT system. The first two methods simply vary the behaviour of the tokenizer with respect to XML markup. The third method, "markup transformation", removes markup before training and translation – and thus is a **reinsertion** strategy. After translation, the markup is restored with the help of phrase segmentation reported by the decoder. They report that XML-aware tokenization yielded the best results, albeit by very small margins.

Zhechev and van Genabith (2010) are the first to describe a **masking** strategy. They are aware that "letting any MT system deal with these tags in a probabilistic manner can easily result in ill-formed, mis-translated and/or out-of-order meta-tags in the translation" (ibid.). To avoid this problem, they replaced XML tags with IDs that act as a placeholder for the actual markup. All IDs were unique on a global level, i.e. throughout the whole corpus. Since markup handling is not the primary goal of this paper, they do not evaluate their approach in any way.

Hudík and Ruopp (2011) further develop the idea of removing the markup before training and translation alltogether. They see their work as a follow-up to Du et al. (2010), trying to improve their **reinsertion** method. They improved the method in the sense that they solved problems

related to reordering and provide an algorithm that reinserts markup into translated segments on the basis of *word alignment* instead of phrase segmentation. Intuitively, reinsertion that uses word alignment will be more precise since reinsertion using phrase segmentation can only insert at phrase boundaries, but no experimental results are presented.

Tezcan and Vandeghinste (2011) experiment with several variants of **masking**. Mainly, what is varied is the specificity of the mask tokens. Mask tokens can be unique identifiers for stretches of markup (resulting in a high number of different mask tokens) or can be more generic (in the extreme case, one single mask token). The main outcome of their experiments is that according to automatic metrics of translation quality, a masking method that assigns masks based on the XML element name performed best.

Finally, Joanis et al. (2013) describe a **reinsertion** strategy that uses both phrase segmentation and word alignment to decide where markup tags should be reinserted. They performed a "mini evaluation" of their approach, manually annotating roughly 1500 segments. The results showed that "most tags are placed correctly" (ibid., 79), because 93 % of TMX tags and 90 % of XLIFF tags were perfect according to the human annotators.

The authors themselves identify an important limitation of their work, namely that they "do not carry out an experimental comparison between the [masking] and [reinsertion] approaches, though this would certainly be a worthwile next step" (ibid., 74). Such an evaluation would indeed be advisable, and the goal of the current work is exactly that: providing reimplementations of different approaches and comparing them to each other in controlled experiments.

3 Data

For our experiments, we have used two data sets with parallel text in German and English:

- **XLIFF:** a real-world collection of XLIFF documents in which inline markup occurs naturally
- **Euromarkup:** a large set of synthetic examples we ourselves have created by inserting inline markup into the Europarl corpus[2]

	English	German
Number of segments	427k	425k
Number of tokens	3.5m	3m
Segments with markup	98k	97k

Table 1: Descriptive statistics of the **XLIFF** data set, markup tags count as 1 token

	English	German
Number of segments	1.7m	1.7m
Number of tokens	52m	50m
Segments with markup	893k	893k

Table 2: Descriptive statistics of the **Euromarkup** data set, markup tags count as 1 token

The documents in the **XLIFF** data set are so-called "introductory checklists" used for parameterization of banking software, similar to software manuals, so the texts are from a very technical domain and were actually post-edited by translators. But although the data set is a real use case and typical of machine translation and industry settings, its suitability for markup handling is questionable.

After performing initial experiments with the XLIFF data set, it became clear that handling the markup in this data is relatively easy: segments are short (8 tokens on average), which means that the translation and additional information like word alignment will be accurate, and there is little reordering that could involve markup tags. In short, there are few hard problems for markup handling methods to tackle in the XLIFF data.

In order to discriminate better between the methods, we introduce a second data set, **Euromarkup**, a blend of Europarl (Koehn, 2005) and markup tags. Because it is a synthetic data set that we built ourselves, it has the following desired properties: longer segments (more than 20 tokens on average) and a lot of reordering. We have introduced markup in a way that is consistent with word alignment and ensured that half of the markup was inserted where reordering takes place.

Tables 1 and 2 show the size of both data sets and, importantly, how much markup they contain. Markup is abundant in both sets and in this respect, both are suitable for testing markup handling approaches.

[2] An implementation of an algorithm that inserts random inline markup into parallel, word-aligned data is available upon request.

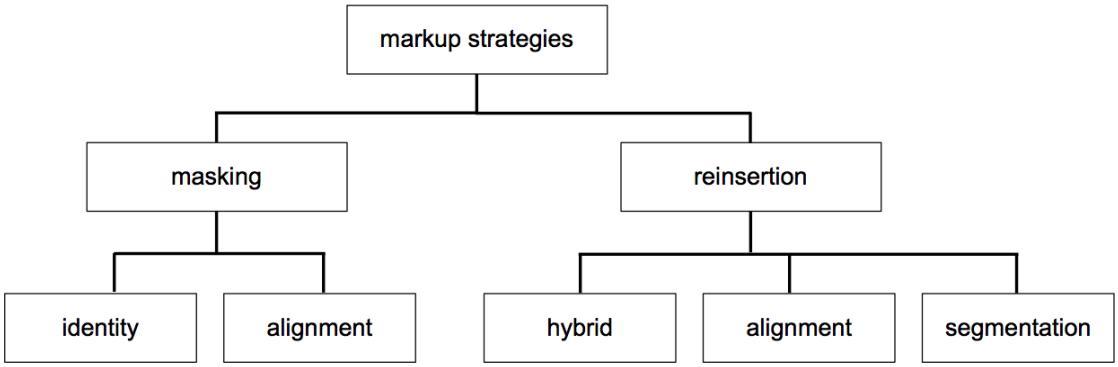

Figure 1: Overview of implemented strategies to process markup

4 Methods

We have implemented five different methods of handling markup in the same machine translation framework, `mtrain`. All methods are described in Section 4.1. Section 4.2 explains the experimental setup and how the results were evaluated.

4.1 Implementation of markup handling methods

Inspired by previous work, we have designed five different ways to treat markup in machine translation (see Figure 1 for an overview). In `mtrain`, two variants of **masking** are available:

- **identity masking:** before training and translation, markup is replaced by mask tokens that are *unique* within the segment. After translation, the original content can be restored without any additional information.
- **alignment masking**: before training and translation, markup is replaced by mask tokens that are *identical* to each other. After translation, word alignment is used to guide the unmasking process.

In all masking approaches, the mapping between the mask tokens and the original markup content must be held in memory until after translation. Stretches of markup are identified by means of a regular expression. Therefore, masking is actually not limited to markup, but is implemented as a general method to mask any string that can be described by a regular expression.

On the other hand, there are three implementations of **reinsertion** that roughly work as follows:

- **segmentation reinsertion:** before training and translation, markup is removed completely from the segments. After translation, the original markup is reinserted into the translation using phrase segmentation.
- **alignment reinsertion:** identical to segmentation reinsertion, except that word alignment is used instead of phrase segmentation.
- **hybrid reinsertion:** both phrase segmentation and word alignment are used for reinsertion, together with a set of rules. A reimplementation of Joanis et al. (2013).

All strategies assume ideal conditions. Masking methods assume that the translation did not necessitate any reordering of mask tokens (this assumption is specific to identity masking) and that the decoder did not omit any mask tokens. Methods that rely on word alignment (alignment masking, alignment reinsertion and hybrid reinsertion) assume ideal, maximally informative word alignment. Methods that rely on phrase segmentation (segmentation reinsertion and hybrid reinsertion) assume that markup only occurs at the boundaries of phrases and that phrase segmentation is available in the first place.

In practice, these assumptions do not always hold. For instance, reordering may take place or the word alignment might be inaccurate and for those cases, the framework offers flexibility. If the placement of a markup tag is uncertain, any method can be instructed to insert the tag anyway at the end of the segment (aggressive behaviour) or not to introduce this markup tag at all (conservative behaviour).

An important difference between masking and reinsertion methods is the nature of the training data: if a masking method is used, then the training

data will contain mask tokens and the system derived from the data will know about mask tokens. If a reinsertion method is used, the training data will not contain any markup. In this regard, reinsertion is more flexible since it can be used with any machine translation system.

4.2 Experiments

We compare the overall performance of all 5 implemented markup handling strategies by training a series of SMT systems. The systems are identical except for their method of markup handling.

What all systems have in common is the data sets, preprocessing (except for markup handling), model training and translation parameters. We have randomly divided the data sets into training (roughly 400k segments for XLIFF data, roughly 1.7m for Euromarkup data), tuning (2000 segments) and testing (1000 segments) sets that were fixed for all systems, the direction of translation is always from German to English.

We train a fairly standard, phrase-based SMT system with Moses: a maximum phrase length of 7, a 5-gram KenLM language model with modified Kneser-Ney smoothing, lexicalized reordering model, standard Moses recasing and standard tokenization. Word alignment and symmetrization is performed by `fast_align` and `atools` (Dyer et al., 2013). The phrase and reordering table are compressed with the `cmph` library. The weights of the model are tuned with MERT (Och, 2003).

For all of the five implemented strategies, such a system was trained, varying only the markup handling. Since our framework allows more fine-grained control over the algorithms, we have used the following settings: if there is uncertainty about where a markup tag should be placed, it must still be inserted into the translation at the very end. The translation of mask tokens is not enforced ("forced decoding"), instead the decision is left to the decoder.

In addition to the five systems above, we have trained the following baseline system:

- **strip:** markup is stripped entirely from the training, tuning and evaluation corpus.

in order to have an estimate of the overall quality of machine translation when no markup is involved.

Finally, we have measured the outcome of our experiments automatically and manually. Automatic metrics should never be used to evaluate the performance of markup handling methods, and we have employed them only to answer a preliminary question: do mask tokens in the training data have an impact on the overall quality of machine translation? It is unclear whether mask tokens affect negatively the overall output of the system and if that were the case, developers should refrain from using masking to handle markup.

We measure the effect of mask tokens by comparing the machine-translated test set with the human reference **after removing markup on both sides**. Then, the `MultEval` tool is used (Clark et al., 2011) to report BLEU (Papineni et al., 2002), METEOR (Lavie and Agarwal, 2007), TER (Snover et al., 2006) and length scores.

While the automatic evaluation translates all of the 1000 segments in the test set, the manual evaluation only looks at the segments where both the source and target reference have tags in them. Markup tags were inspected manually and assigned one of the following categories (inspired by Joanis et al., 2013):

- **good**: correct markup is present, correctly placed,
- **reasonable**: correct markup is present, but needs to be moved,
- **wrong**: markup is broken or not present at all,
- **garbage-in**: the decoder output is unintelligible and there is no proper place for markup.

In general, it is always preferable to transfer markup tags to the target segment, even if the correct position cannot be determined. From the point of view of the post-editor, it is more efficient to move a markup tag instead of going back to the source segment. Therefore, markup tags that are in the wrong place are described as "reasonable". In theory, there are scenarios where markup tags should be dropped entirely (because all tokens related to them have no translation) but in the vast majority of cases, missing markup tags are "wrong".

In this manual evaluation we will focus on evaluating the markup handling, not the performance

	XLIFF				Euromarkup			
	BLEU	**METEOR**	**TER**	**Length**	**BLEU**	**METEOR**	**TER**	**Length**
strip	60.5	46.3	26.6	93.9	32.4	34.2	52.4	98.7
IM	61.0	46.7	26.4	94.3	30.9	33.9	53.8	99.3
AM	60.4	46.3	26.9	94.2	31.4	34.0	54.1	99.8
SR	60.5	46.4	26.8	94.9	32.6	34.5	52.1	98.7
AR	60.5	46.4	26.9	94.9	32.3	34.6	52.1	98.7
HR	60.4	46.3	26.8	94.8	32.2	34.5	52.5	99.0

Table 3: Automatic evaluation of the overall performance of markup handling methods, after markup was removed completely. The metrics reported are BLEU (higher is better), METEOR (higher is better) and TER (lower is better). IM = identity masking, AM = alignment masking, SR = segmentation reinsertion, AR = alignment reinsertion, HR = hybrid reinsertion.

of the systems in general. For each data set, we decided to look at a maximum of 200 parallel segments from the test set that contain markup. In the XLIFF test set, only 176 segments contain markup, so all of them were evaluated, which amounts to a total of 658 tags.

In the Euromarkup test set, we annotated the first 200 segments that contain markup, and they contain 584 tags in total. We only look at the lowercased, tokenized version of the translation output, after processing the reference accordingly.

5 Results

The automatic evaluation in Table 3 shows the overall performance of systems on "normal" text, that is, after markup was stripped from both the machine-translated hypothesis and the human reference. All systems trained on XLIFF data have a performance comparable to the baseline system that did not see any markup at all ("strip"). For instance, the BLEU scores range from 60.4 to 61.0. The systems that use a variant of reinsertion and the "strip" system are expected to produce exactly the same translation since they are trained on the same data, but non-deterministic tuning has caused slight fluctuations in all scores.

For the XLIFF data set, both masking systems perform as good as the baseline system. But in general, the scores for this data set are high and it is clear that the data set is easy to translate. For the Euromarkup data, the behaviour of the reinsertion methods does not change since they are still very close to the baseline. However, using this synthetic data set, masking indeed decreases the overall quality of machine translation in terms of BLEU scores.

Moving on to the manual evaluation, Table 4 shows that for the XLIFF data set, identity masking clearly performs best, because it places correctly 658 out of 658 tags. Alignment masking and alignment reinsertion are not too far behind, both have led to 4 cases of "reasonable" tags (tags are present but in the wrong place). Hybrid reinsertion could not determine the correct position for markup in 34 cases. Even segmentation reinsertion placed markup correctly in 582 out of 658 cases. Using XLIFF data, no tags were omitted ("wrong") and the decoder never produced unusable output ("garbage-in").

Using Euromarkup, the "harder" data set, shifts the picture: hybrid reinsertion performs best on this data set as it placed correctly 437 out of 584 tags. Another 133 were in the wrong place, but all output segments were still well-formed and markup was not broken. Alignment masking and alignment reinsertion still work reasonably well, transferring markup correctly in 412 and 415 cases, respectively. Identity masking on the other hand is now well behind, and segmentation reinsertion performed worst, as expected.

Another striking result is that both masking methods lead to a number of "wrong" tags, i.e. tags that make the whole segment a malformed XML fragment. Malformed content is likely to cause problems, depending on the application that processes the translation output. Finally, the systems trained on Euromarkup data also produced a few cases where the decoder output is unintelligible (i.e. not even a human annotator could have placed markup correctly).

In summary, identity masking solved the task of markup handling perfectly given a corpus of

	XLIFF (tags in total: 658)				Euromarkup (tags in total: 584)			
	good	**reasonable**	**wrong**	**garbage-in**	**good**	**reasonable**	**wrong**	**garbage-in**
IM	**658**	**0**	**0**	**0**	372	167	29	16
AM	654	4	0	0	412	123	39	10
SR	582	76	0	0	252	318	0	14
AR	654	4	0	0	415	148	7	14
HR	624	34	0	0	**437**	**133**	**0**	**14**

Table 4: Manual evaluation of the performance of markup handling methods, by tags. IM = identity masking, AM = alignment masking, SR = segmentation reinsertion, AR = alignment reinsertion, HR = hybrid reinsertion.

short and relatively monotone segments. In that case, both alignment masking and alignment reinsertion are viable alternatives. However, the second, synthetic data set with longer segments and "harder" markup emphasizes better the differences between the methods. Hybrid reinsertion has outperformed all other methods on the second data set. Alignment reinsertion and alignment masking are still viable, but identity masking struggled with the second data set.

6 Discussion

In Section 6.1, we discuss whether masking methods for markup handling have merit. Section 6.2 discusses the performance of all reinsertion methods.

6.1 Masking methods

Mask tokens in the training data can lead to a decrease in overall translation quality and thus "make the translation itself worse" (Joanis et al., 2013, 78). More concretely, Table 3 shows that on the Euromarkup data set, masking systems perform worse (e.g. BLEU score of around 31) than the baseline and the reinsertion systems (e.g. BLEU score of around 32). One possible explanation is that mask tokens in the training data potentially dilute the phrase statistics derived from that corpus. In the training data, a segment like

```
i am delighted to hear that
```

can be interrupted by mask tokens in arbitrary ways:

```
i am __MASK__ delighted __MASK__
__MASK__ to hear __MASK__ that

__MASK__ i __MASK__ am delighted
__MASK__ to hear that __MASK__
```

But at translation time, the same phrase can contain masks in different places:

```
i am __MASK__ __MASK__ delighted to
__MASK__ hear __MASK__ that
```

and since this sequence of words is unseen, the segment will be broken up into smaller phrases, despite the fact that the underlying phrase `i am delighted to hear that` is actually known to the system and could be translated as a single phrase.

This does not hold in general, since we only observed this effect in synthetic data and therefore, this finding does not invalidate masking as a whole. Still, we would only want to tolerate such a degradation in overall translation quality if it comes with superior markup handling performance.

Identity masking worked well on the XLIFF data set, but not on the Euromarkup data. The method is very lean because it does not rely on any kind of auxiliary information (such as phrase segmentation or word alignment), but also it is unable to cope with any amount of reordering on a fundamental level. Unique IDs are assigned to markup tags according to their position in the segment going from left to right, and therefore, reordering is not modelled at all. This means that if translation involves reordering of markup tags, identity masking will fail (see Table 5 for an example).

The reordering problem is overcome by alignment masking, where reordering is explicitly modelled and word alignment is used as a proxy. Handling the markup present in the XLIFF data set did not cause any difficulty for alignment masking and word alignment was sufficient to solve the problem in all but 4 cases. On the Euromarkup data, alignment masking proved to be robust and still placed correctly most tags. Using word alignment

source segment	Leider <i/> war <g/> dies von kurzer Dauer.
target reference	sadly , it <i/> was <g/> short @-@ lived .
identity masking	unfortunately , <i/> this <g/> was a short time .
alignment masking	unfortunately , this <i/> was <g/> a short time .
segmentation reinsertion	<i/> <g/> unfortunately , this was a short time .

Table 5: Examples of markup handling that show 1) the inability of identity masking to deal properly with markup that needs reordering and 2) that segmentation reinsertion can only insert markup at phrase boundaries.

enables the unmasking algorithm to track reordering, at the cost of depending on word alignment.

Both identity masking and alignment masking have led to a number of cases where the placement of tags resulted in the whole segment being malformed XML. On the one hand, this is because the default behaviour of the algorithms is to insert tags at the very end of the segment if the correct place cannot be determined. If, for instance, an opening element tag is placed at the very end in this manner, the whole segment will be malformed. On the other hand, both masking methods do not understand the notion of *tag pairs* (pairs of opening and closing tags) – which is necessary to guarantee that the output will be well-formed XML.

A clear advantage of masking is that it is not limited to markup at all: anything that can be described with a regular expression can be masked and unmasked in our framework[3]. In this respect, masking methods are more versatile than reinsertion methods and for certain use cases, this might outweigh the limitations we have mentioned.

6.2 Reinsertion methods

Looking at the results on the XLIFF data, segmentation reinsertion cannot be said to have failed the task of reinserting markup. Quite on the contrary, it is remarkable that segmentation reinsertion could act on the markup in such a precise way, given that phrase segmentation is imprecise to begin with: it can only insert tags at phrase boundaries, which is bound to lead to errors (see Table 5 for an example). A further analysis of the XLIFF data revealed that markup is frequently present at the very beginning and very end of segments. If there is no reordering, markup at the beginning and end of segments can always be inserted in the right place by segmentation reinsertion, regardless of phrase boundaries.

[3]Incidentally, this is also the explanation for why masking methods do not insert tags in pairs: most strings that can be masked do not come in pairs.

Still, segmentation reinsertion is very limited and the results on the Euromarkup data set confirm that it leads to a very high number of misplaced ("reasonable") tags: 318 out of 584 tags were not placed correctly. In fact, segmentation reinsertion is downright paradoxical: it works better if phrases are short, while longer phrases typically lead to better translations, and by extension, segmentation reinsertion works well if the machine translation system is feeble. If word alignment is available, there is probably no reason to implement or use segmentation reinsertion at all.

The performance of alignment reinsertion is very similar to alignment masking, which is not surprising, given that they make use of the same additional information from the decoder. On the XLIFF data set, alignment reinsertion solves the problem almost perfectly, all scores are identical to alignment masking. On the Euromarkup data set, the number of correctly placed tags ("good" tags) is very similar, but alignment masking is prone to break markup structures, while alignment reinsertion is not. The alignment reinsertion algorithm generally keeps together pairs of tags and actively avoids placements that would break the markup, yet not breaking the markup is not a hard requirement in our implementation.

Turning to the most promising strategy, hybrid reinsertion coped well with both data sets. On the XLIFF data, it placed correctly 624 out of 658 markup tags, but more importantly, it outperformed all other methods on the Euromarkup data. A possible explanation for its superior performance is that, as a hybrid method, it can overcome deficiencies in phrase segmentation with word alignment and vice versa. Similar to the other reinsertion methods, hybrid reinsertion also models pairs of tags explicitly and ensures the well-formedness of the segment.

In addition, our experiments very likely underestimate the method presented in Joanis et al. (2013) since there, "some care is taken to preserve

the source order when multiple tags end up between the same two target language words" (ibid., 78). Our implementation does not guarantee the order of adjacent tags.

The strength of reinsertion in general is that it can be used with any machine translation system, while masking must be used together with a system trained on mask tokens. If masked segments are given to a system that did not see mask tokens during training, the results are quite unpredictable. In the case of phrase-based SMT systems, this would likely lead to all mask tokens being moved to the end of the segment, because language models prefer grouping together unknown words (Fishel and Sennrich, 2014).

Put another way, the decision to use masking as the markup handling method must be made at training time, reinsertion can be introduced at translation time. In both cases, the nature of the decoder is another limiting factor: systems that cannot report phrase segmentation make it impossible to use segmentation reinsertion, but also rule out the best-performing method, hybrid reinsertion. Word alignment, however, can be supplied by an additional tool in case the decoder is unable to report this information. This means that methods relying on word alignment are broadly applicable across machine translation paradigms.

7 Conclusion

We have presented work on handling markup in statistical machine translation. In our experiments we have compared the usefulness of five different markup handling strategies. The main findings are: hybrid reinsertion outperformed all other methods and was found to cope best with the markup in a synthetic data set. Alignment masking and alignment reinsertion also placed correctly two out of three tags and should be regarded as viable alternatives.

However, alignment masking led to more cases of malformed XML and masking methods can only be used with systems that are trained with mask tokens. For new projects that have to decide on a method to handle markup we therefore recommend to use hybrid reinsertion (if phrase segmentation is available) or alignment reinsertion (otherwise).

In recent years, neural approaches have dominated the field of machine translation and it is therefore worth considering whether our results carry over to neural machine translation systems. Encoder-decoder networks with attention (Bahdanau et al., 2014), a popular architecture for translation, do not report phrase segmentation of course, which rules out both segmentation reinsertion and hybrid reinsertion. On the other hand, alignment information can still be derived from attention weights.

Future work could investigate whether alignment masking or alignment reinsertion are feasible in the context of neural machine translation. But neural networks also lend themselves to more innovative experiments: anecdotal evidence suggests that character-level recurrent neural networks (Hochreiter and Schmidhuber, 1997) are capable of generating well-formed markup[4]. This is a remarkable achievement and to our knowledge, this property of neural networks has never been investigated in earnest.

Also, our implementations currently do not properly model two important aspects of the data: whitespace inside and outside of XML elements is not handled properly and our algorithms never regard dropping tags from the translation as a correct action. Addressing those two shortcomings would also be a worthwhile continuation of our work.

Acknowledgments

We thank the anonymous reviewers for their valuable comments and suggestions.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Jonathan H Clark, Chris Dyer, Alon Lavie, and Noah A Smith. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*. Association for Computational Linguistics, pages 176–181.

Jinhua Du, Johann Roturier, and Andy Way. 2010. TMX markup: a challenge when adapting smt to the localisation environment. In *EAMT - 14th Annual Conference of the European Association for Machine Translation*. European Association for Machine Translation.

[4]See http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Atlanta, Georgia, pages 644–648. http://www.aclweb.org/anthology/N13-1073.

Mark Fishel and Rico Sennrich. 2014. Handling technical OOVs in SMT. In *Proceedings of The Seventeenth Annual Conference of the European Association for Machine Translation (EAMT)*. pages 159–162.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Tomáš Hudík and Achim Ruopp. 2011. The integration of Moses into localization industry. In *15th Annual Conference of the EAMT*. pages 47–53.

Eric Joanis, Darlene Stewart, Samuel Larkin, and Roland Kuhn. 2013. Transferring markup tags in statistical machine translation: A two-stream approach. In Sharon O'Brien, Michel Simard, and Lucia Specia, editors, *Proceedings of MT Summit XIV Workshop on Post-editing Technology and Practice*. pages 73–81.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of MT Summit*. volume 5, pages 79–86. http://mt-archive.info/MTS-2005-Koehn.pdf.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*. pages 177–180. http://www.aclweb.org/anthology/P07-2045.pdf.

Alon Lavie and Abhaya Agarwal. 2007. METEOR: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*. pages 228–231.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*. pages 160–167. http://www.aclweb.org/anthology/P03-1021.pdf.

Sharon OBrien. 2011. Towards predicting post-editing productivity. *Machine translation* 25(3):197–215.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pages 311–318.

Carla Parra Escartín and Manuel Arcedillo. 2015. Machine translation evaluation made fuzzier: A study on post-editing productivity and evaluation metrics in commercial settings. In *Proceedings of MT Summit XV*. Association for Machine Translation in the Americas, pages 131–144.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of Association for Machine Translation in the Americas*. pages 223–231.

Arda Tezcan and Vincent Vandeghinste. 2011. SMT-CAT integration in a Technical Domain: Handling XML Markup Using Pre & Post-processing Methods. *Proceedings of EAMT 2011* .

Ventsislav Zhechev and Josef van Genabith. 2010. Seeding statistical machine translation with translation memory output through tree-based structural alignment. In *Proceedings of the 4th Workshop on Syntax and Structure in Statistical Translation*. Coling 2010 Organizing Committee, Beijing, China, pages 43–51. http://www.aclweb.org/anthology/W10-3806.

Appendix

Listings 1, 2 and 3 show how different components of `mtrain` can be used to pre- and post-process markup. Although `mtrain` is a full-fledged wrapper around the Moses framework, its markup handling modules can also be used as standalone components.

Both masking methods are implemented in the module `mtrain.preprocessing.masking`, while three reinsertion methods are available in `mtrain.preprocessing.reinsertion`.

```
>>> from mtrain.preprocessing.masking import Masker
>>> masker = Masker('alignment')
>>> masked_segment = 'Message moi a __email__ ou __xml__ __url__ __xml__'
# after translation
>>> translated_segment = 'Email me at __email__ or __xml__ __url__ __xml__'
>>> mapping = [('__email__', 'an@ribute.com'),
    ('__url__', 'http://www.statmt.org'),
    ('__xml__', '<a>'), ('__xml__', '</a>')]
>>> alignment = {0:[0], 1:[1], 2:[2], 3:[3], 4:[4], 5:[5], 6:[6], 7:[7]}
>>> masker.unmask_segment(masked_segment, translated_segment, mapping, alignment)
'Email me at an@ribute.com or <a> http://www.statmt.org </a>'
```

Listing 1: A case of successful alignment masking and unmasking. The unmasking step crucially depends on alignment information reported by the decoder. Unmasking succeeds in this case because all mask tokens are present in the translation and because the alignment is perfect.

```
>>> from mtrain.preprocessing.reinsertion import Reinserter
>>> reinserter = Reinserter('alignment')
>>> source_segment = 'Hello <g id="1" ctype="x-bold;"> World ! </g>'
# markup removal, then translation...
>>> translated_segment = 'Hallo Welt !'
>>> alignment = {0:[0], 1:[1], 2:[2]}
>>> reinserter._reinsert_markup_alignment(source_segment, translated_segment,
    alignment)
'Hallo <g ctype="x-bold;" id="1"> Welt ! </g>'
```

Listing 2: Alignment reinsertion based on the original source segment that contains markup, the translated segment and, most importantly, the alignment between the source segment without markup and the translation.

```
>>> from mtrain.preprocessing.reinsertion import Reinserter
>>> reinserter = Reinserter('hybrid')
>>> source_segment = 'Hello <g id="1" ctype="x-bold;"> World ! </g>'
# markup removal, then translation...
>>> translated_segment = 'Hallo Welt !'
>>> alignment = {0:[0], 1:[1], 2:[2]}
>>> segmentation = {(0,1):(0,1), (2,2):(2,2)}
>>> reinserter._reinsert_markup_full(source_segment, translated_segment,
    segmentation, alignment)
'Hallo <g ctype="x-bold;" id="1"> Welt ! </g>'
```

Listing 3: Hybrid reinsertion given perfect segmentation and alignment.