Learning Non-Monotonic Automatic Post-Editing of Translations from Human Orderings

António Góis*
Unbabel
Lisbon, Portugal
antoniogois@gmail.com

Kyunghyun Cho
New York University
Facebook AI
New York, USA
kyunghyun.cho@nyu.edu

André Martins
Unbabel
Instituto de Telecomunicações
Lisbon, Portugal
andre.martins@unbabel.com

Abstract

Recent research in neural machine translation has explored flexible generation orders, as an alternative to left-to-right generation. However, training non-monotonic models brings a new complication: how to search for a good ordering when there is a combinatorial explosion of orderings arriving at the same final result? Also, how do these automatic orderings compare with the actual behaviour of human translators? Current models rely on manually built biases or are left to explore all possibilities on their own. In this paper, we analyze the orderings produced by human post-editors and use them to train an automatic post-editing system. We compare the resulting system with those trained with left-to-right and random post-editing orderings. We observe that humans tend to follow a nearly left-to-right order, but with interesting deviations, such as preferring to start by correcting punctuation or verbs.

1 Introduction

Neural sequence generation models have been widely adopted for tasks such as machine translation (MT) (Bahdanau et al., 2015; Vaswani et al., 2017) and automatic post-editing of translations (Junczys-Dowmunt and Grundkiewicz, 2016; Chatterjee et al., 2016; Correia and Martins, 2019; Lopes et al., 2019). These models typically generate one word at a time, and rely on a factorization that imposes a left-to-right generation ordering. Recent alternatives allow for different generation orderings (Welleck et al., 2019; Stern et al., 2019; Gu et al., 2019a), or even for parallel generation of multiple tokens (Gu et al., 2018; Stern et al., 2019; Gu et al., 2019b; Zhou et al., 2020), which allows exploiting dependencies among non-consecutive tokens. One potential difficulty when training non-monotonic models is how to learn a good generation ordering. There are exponentially many valid orderings to generate a given sequence, and a model should prefer those that lead to accurate translations and can be efficiently learned. In previous work, to guide the search for a good ordering, oracle policies have been provided (Welleck et al., 2019), or another kind of inductive bias such as a loss function tailored to promote certain orderings (Stern et al., 2019). However, no supervision has been used with orderings that go beyond simple patterns, such as left-to-right, random ordering with a uniform distribution, or a balanced binary tree.

While prior work has focused on learning generation orderings in an unsupervised manner, in this paper we ask the question of whether human generation orderings can be a useful source of supervision. One such possible source lies in the keystrokes of humans typing. It is known that

0 :	Die LMS geöffnet *ist* .	[I:2:**ist**]
1 :	Die LMS **ist** geöffnet *ist* .	[D:4:*ist*]
2 :	Die LMS **ist** geöffnet .	

Table 1: Example of a small post-edit from the training set. Each action is represented by three features: its type (I for insert and D for delete), its position in the sentence and the token to insert/delete. In this example, the token marked *in red* needs to be removed since it is incorrectly placed. The **blue** token is inserted to obtain the correct pe.

*Work partly done during a research visit at New York University.

edit operations performed by human translators are not arbitrary (Góis and Martins, 2019). But it is not known how the orderings preferred by humans look like, or how they compare to orders learned by models.

To investigate this question, we propose a model that learns generation orderings in a supervised manner from human keystrokes. Since a human is free to move back and forth arbitrarily while editing text, the chosen order of operations can be used as an additional learning signal. More specifically, we do this in the context of **automatic post-editing (APE)** (Simard et al., 2007). APE consists in improving the output of a blackbox MT system by automatically fixing its mistakes. The act of post-editing text can be fully specified as a sequence of delete (DEL) and insert (INS) **actions** in given positions. Furthermore, if we do not include redundant actions in a sequence, that sequence can be arbitrarily reordered while still producing the same output. For instance, in Table 1, we can switch the order of the two actions, as long as we rectify to delete position 3 instead of position 4.

We compare a model trained with human orderings to others trained with left-to-right and random orderings. We show that the resulting non-monotonic APE system learned from human orderings outperforms systems learned on random orderings and performs comparably or slightly better than a system learned with left-to-right orderings.

2 Dataset

2.1 WMT data and keystrokes

The dataset used in this paper is the keystrokes dataset introduced by Specia et al. (2017) in the scope of the QT21 project. This dataset consists of triplets required to train an APE system: *source* sentences (src), *machine-translation* outputs (mt) and *human post-edits* (pe). Features about the post-editing process are also provided, including the keystroke logging. In particular, we focus on the language pair English to German (En-De) in the Information Technology (IT) domain, translated with a Phrase-Based Statistical MT system (PBSMT) – this dataset has a large intersection with the data used in the WMT 2016-18 APE shared tasks (Chatterjee et al., 2018). This allows for comparison with systems previously submitted to the shared task by using the exact same development and test sets, while augmenting the training set with keystroke logging information.

	size	mt=pe	min-edit	human-edit
train with keystrokes	16,068	18.2%	6.6	14.48
full train	23,000	14.6%	11.8	—
dev '16	1,000	6.0%	11.3	—

Table 2: WMT-APE datasets: Original training set and development set from the WMT-APE shared task, and subset of the training set also found in the dataset from Specia et al. (2017). mt=pe is the percentage of samples where the mt output is already correct. min-edit is the average count of actions (DEL ans INS) required to change mt into pe, computed from Levenshtein distance. human-edit is the average count of actions computed from human keystrokes. Both average action counts exclude samples with zero actions.

Out of 23,000 training samples provided by the WMT 2016-17 shared tasks, 16,068 are also present in the dataset from Specia et al. (2017). This intersection is obtained by requiring the same triplet (src, mt, pe) to be present in both datasets. Since the WMT dataset comes already pre-processed, the following pre-processing is applied to the dataset containing keystrokes, to increase their intersection: using tools from Moses (Koehn et al., 2007), we apply En punctuation-normalization to the whole triplet, followed by tokenization of the corresponding language (either En or De). Additionally, we preprocess the raw keystrokes to obtain word-level DEL and INS actions (detailed in §2.2).

Table 2 shows statistics from WMT's original data and training set after intersecting with the keystrokes dataset from Specia et al. (2017). We denote by min-edit the average count of DEL and INS obtained from the Levenshtein distance. Average count of human actions (human-edit) is only available for the subset of the training data found in the keystrokes dataset. Also note that keystrokes will not be required during inference, only for training. Once a model is already trained, the only input required is a (src, mt) pair in order to predict a full sequence of actions and produce the final pe. This allows to use the exact same development and test sets as in the shared task, without losing any samples.

2.2 Preprocessing raw keystrokes

The original keystrokes logging provides character-level changes made by the human editor. Since this information is too fine-grained for our model, we preprocess the raw keystokes to obtain word-level DEL and INS. Our starting

src	When you decrease opacity , the underlying artwork becomes visible through the surface of the object , stroke , fill , or text .
mt	Wenn Sie die Deckkraft verringern , wird das zugrunde liegende Bildmaterial durch die Oberfläche des Objekts , Kontur , Fläche oder Text angezeigt .
pe	Wenn Sie die Deckkraft verringern , wird das darunterliegende Bildmaterial durch die Oberfläche des Objekts , **der** Kontur , **der** Fläche bzw. des Textes sichtbar .
l2r	D:8:zugrunde D:8:liegende I:8:darunterliegende I:16:**der** I:19:**der** D:21:oder D:21:Text D:21:angezeigt I:21:bzw. I:22:des I:23:Textes I:24:sichtbar STOP
shuff	D:20:oder I:22:bzw. D:20:Text I:22:des I:10:darunterliegende D:8:zugrunde I:23:sichtbar I:19:**der** I:24:Textes I:17:**der** D:22:angezeigt D:8:liegende STOP
h-ord	I:17:**der** I:20:**der** D:22:oder I:24:bzw. I:25:des D:22:Text I:25:Textes D:8:zugrunde D:8:liegende I:8:darunterliegende D:21:angezeigt I:24:sichtbar STOP
human	I:17:**der** I:20:**der** D:22:oder I:22:bzw. I:23:des D:24:Text I:24:Textes D:8:zugrunde D:8:liegende I:8:darunterliegende D:25:. D:24:angezeigt I:24:sichtbar I:25:. STOP

Table 3: Example of a sentence and its minimum-edit actions ordered in three different ways: left-to-right (*l2r*), randomly shuffled (*shuff*) and following human order (*h-ord*). The unfiltered human actions are also presented (*human*). We can see that the human chose to first insert the two words marked **in blue**, later moving back in the sentence to edit the leftmost mistakes.

point is the sequence of strings containing the mt state after each keystroke. We track which word is currently being edited and store an action to summarize the change. Replacements are represented as a DEL followed by INS. Multiple words may be changed simultaneous, either by selecting and deleting a block of words or by pasting text. Block changes assume a left-to-right sequence of actions.

Table 3 contains an example of a preprocessed sequence of keystrokes in the last line (*human*). When applied to the mt, the sentence is converted to pe. Note that this kind of action sequences may be impossible to re-order due to redundant actions — for a token t not present in mt nor pe, the actions INS:0:t DEL:0:t cannot be switched.

We perform an additional step to eliminate redundant actions performed by human post-editors. Editors may take paths significantly longer than the minimum edit distance. During experiments these longer paths proved harmful for the model, so we designed a way to filter all actions which are not relevant. First, an optimal action sequence that minimizes edit (Levenshtein) distance is obtained with dynamic programming. Since by definition this sequence does not contain redundant actions, these actions can be reordered to produce the same output. This provides a chance to experiment with different orders, such as left-to-right or human order. To align the unfiltered human actions with the minimum-edit actions, we first match human actions to machine actions that insert/delete the same token. Then, we break ties by aligning each machine action to the human action applied to the closest position in the sentence. Note that in some cases this alignment is not possible – for instance in Table 1, if the editor had moved the word *geöffnet* instead of *ist*, the alignment would have failed. However, in practice this only happens in around 1% of the samples. In such cases, we simply keep the unfiltered human order.

For reproducibility purposes, we provide the dataset containing the (src, mt, pe) triplets, together with the four kinds of action sequences seen in Table 3, in https://github.com/antoniogois/keystrokes_ape.

2.3 Analysis of action sequences

Given the actions provided by the minimum-edit distance between mt and pe it is possible to reorder them arbitrarily, as explained in the previous section. In Figure 1 we visualize three different kinds of orderings: the one produced by human post-editors, a random ordering, and the ordering obtained by processing the sentence left-to-right.

We show, in the vertical axis of Figure 1, in which position of the sentence an action is applied, relatively to the other actions of the same sample. On the left-hand plot we display two samples. Sample A contains 3 actions, applying the leftmost action followed by the rightmost and finally an action applied in a sentence position somewhere in between the first two. This could be generated from random shuffling or human-order, but never from the artificial left-to-right order. On the other hand, Sample B contains 5 actions which could have been ordered by any of the three methods. In practice, 2.0% of the human-ordered samples fol-

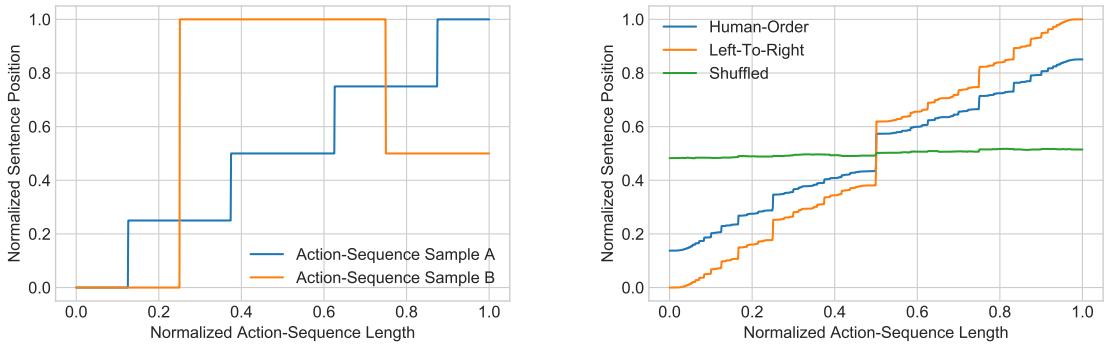

Figure 1: Relative positions of actions in a sequence. Each action-sequence length is normalized to 1, and we show the order by which actions were applied. For instance, sample A in the left-hand plot contains 3 actions, applied in a non-monotonic order — first the leftmost action, then the rightmost and finally the middle-action (e.g. corresponding to actions in positions 3, 8 and 5 of a sentence). Samples A and B represent respectively 2.0% and 4.2% of the human action-sequences. The right-hand plot shows the average of all sequences, in human order, left-to-right and shuffled. Human-order tends to follow a left-to-right order, but not as strict as the artificial left-to-right. The shuffled sequences do not follow any pattern, as expected.

low sample A, and 4.2% follow sample B.

On the right-hand plot we visualize the average line for each of the three orders. We can see that human actions tend to follow a left-to-right order, but not as strictly as the artificial left-to-right. Shuffled sequences are equally likely to start on the far left or far right of the sentence.

Relative orderings displayed in Figure 1 can be represented as a permutation of the sequence (*0, 1, ..., #actions*), i.e. Sample A would be (*0, 2, 1*) and Sample B (*0, 1, 2, 3, 4*). This way it is possible to use Kendall's τ distance (Kendall, 1938) to measure how far we are from a pure left-to-right order. We show this in Table 4, together with the percentage of actions which are a jump-back (applied to a position in the sentence to the left of the previous action) requiring a jump of at least 1 or 4 tokens. We confirm that the human-order is nearly left-to-right, but with some deviations.

	Jump-Back	JB≥4	Kendall's τ
l2r	0.00%	0.00%	0.00
shuff	39.43%	21.34%	0.48
h-ord	14.87%	4.26%	0.16

Table 4: Statistics of action orders following left-to-right, random shuffle and human-order. Jump-Back counts actions applied to any position before the previous action, whereas JB≥4 requires a jump of at least 4 tokens. Kendall's τ distance is measured between the sequence (*0, 1, ..., #actions*) and its permutation matching the order of the actions in the sentence.

In Figure 2 we visualize the words preferred by human editors as first action. We count how many times each word is picked as the first action (without discriminating `DEL` and `INS`), both for human order and left-to-right order. We subtract human occurrences by left-to-right occurrences, keeping only words with a difference of at least 5, and group them by part-of-speech tags. We can see that humans prefer to begin with punctuation, whereas a left-to-right order favours determinants, which tend to appear in the beginning of the sentence.

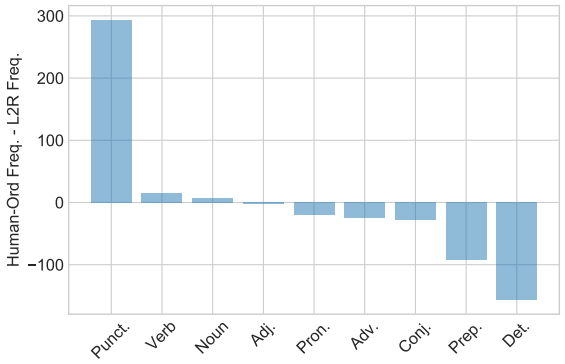

Figure 2: Part-of-speech tags preferred by humans as a first action — positive values indicate tags preferred by humans, negative values indicate tags preferred by left-to-right order. We count occurrences of each word as first action in both human action sequences and left-to-right sequences. We subtract humans counts by left-to-right counts, discard words with a difference lower than 5, and group results by part-of-speech tag.

3 Model

Inspired by recent work in non-monotonic generation (Stern et al., 2019; Gu et al., 2019a; Emelianenko et al., 2019), we propose a model that receives a `src`, `mt` pair and outputs one action at

a time. When a new action is predicted, there is no explicit memory of previous time-steps. The model can only observe the current state of the `mt`, which may have been changed by previous actions of the model.

This model is based on a Transformer-Encoder pre-trained with BERT (Devlin et al., 2019). After producing one hidden state for each token, a linear transformation outputs two values per token: the logit of deleting that token and of inserting a word to its right. Out of all possible `DEL` or `INS` positions, the most likely operation is selected. A special operation is reserved to represent End-of-Decoding. If an `INS` operation is chosen, we still need to choose which token to insert. Another linear transformation is applied to the hidden state of the chosen position. We obtain a distribution over the vocabulary and select the most likely token. Figure 3 illustrates this procedure.

After a `DEL` or `INS` is applied, we repeat this procedure using the updated `mt`. Decoding can end in three different ways:

- When the `STOP` action is predicted;
- When the model enters a loop;
- When a limit of 50 actions is reached.

Once decoding ends, the model outputs the final post-edited `mt`.

Model details. We use BERT's implementation from Wolf et al. (2019) together with OpenNMT (Klein et al., 2017), both based on PyTorch (Paszke et al., 2019). The pretrained BERT-Encoder contains 12 layers, embedding size and hidden size of 768.

We begin with an input sequence x. This sequence is the concatenation of:

`src + [SEP] + <S> + mt + <T>`

where `<S>` and `<T>` are auxiliary tokens used to allow `INS` in position 0 and to represent End-of-Decoding. Tokens before and after `[SEP]` have a different segment embedding, to help differentiate between `src` and `mt` tokens. Let N be the length of x after applying a BERT pre-trained tokenizer (Wolf et al., 2019). This sequence is the input of the BERT-Encoder with hidden dimension $h = 768$. The output is a matrix $H \in \mathbb{R}^{N \times h}$. We call each possible `DEL` position and each `INS` position an **edit-operation**. Note that this does not yet include the choice of a token from the vocabulary. For each token in the partially generated

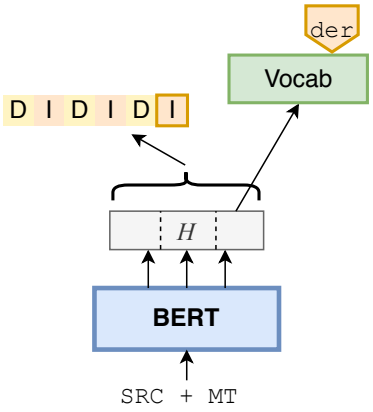

Figure 3: Our proposed model for automatic post-editing. A BERT-Encoder receives as input `src` and `mt` to produce a hidden representation H. We apply a linear transformation to the full H, obtaining a probability distribution over all possible actions. If the chosen action is an `INS`, we obtain a distribution over the vocabulary by applying another linear transformation to H's row of the chosen action position.

sentence, we obtain the logit of `INS` (on the position to its right) and `DEL` (of the token itself) using a learnable matrix $W \in \mathbb{R}^{h \times 2}$. The distribution probability over all possible edit-operations is defined as:

$$p(\text{edit\_op}) = \text{softmax}(\text{flatten}(HW)) \quad (1)$$

To represent the End-of-Decoding operation, we use the action `DEL<T>`. All unavailable actions are masked: `DEL<S>`, `INS-after<T>`, and edit-operations on `src`. When the model predicts an `INS`, a token is then predicted for that position. Let i be the chosen position, $h_i \in \mathbb{R}^h$ the i$^{\text{th}}$ row in H, and $V \in \mathbb{R}^{|v| \times h}$ the matrix mapping to all tokens in a vocabulary of size v:

$$p(\text{token} \mid \text{edit\_op}) = \text{softmax}(Vh_i) \quad (2)$$

The predicted action is applied and we repeat this procedure using the updated x. Since no history of previous actions is kept, this opens the possibility of entering a loop. To handle loops, when we re-visit a state x we stop decoding. Alternatively, we tried applying the N^{th} most likely action on the N^{th} visit to a given x, but this degraded performance slightly.

4 Training

During training, the model may have several correct actions to choose from, even if we only consider actions following a minimum edit distance

	dev 2016		test 2016		test 2017		test 2018	
	TER ↓	BLEU ↑	TER ↓	BLEU ↑	TER ↓	BLEU ↑	TER ↓	BLEU ↑
MT baseline (uncorrected)	24.81	62.92	24.76	62.11	24.48	62.49	24.24	62.99
l2r	22.33 (±0.13)	67.04 (±0.11)	**22.53 (±0.26)**	**66.23 (±0.26)**	**22.63 (±0.3)**	**65.84 (±0.29)**	22.97 (±0.20)	65.49 (±0.26)
shuff	22.47 (±0.15)	66.74 (±0.22)	22.87 (±0.23)	65.89 (±0.28)	23.24 (±0.25)	65.14 (±0.24)	22.94 (±0.12)	65.39 (±0.18)
h-ord	**22.15 (±0.23)**	**67.19 (±0.15)**	22.65 (±0.16)	66.15 (±0.19)	22.75 (±0.08)	65.63 (±0.04)	**22.70 (±0.15)**	**65.72 (±0.22)**
Correia and Martins (2019) (seq2seq BERT)	—	—	18.05	72.39	18.07	71.90	18.91	70.94
Bérard et al. (2017) (actions)	23.07	—	22.89	—	23.08	65.57	—	—

Table 5: Results on development set and test sets used in WMT 2018 APE shared task. We show our system's performance trained by each of the three proposed orderings, and two other models for comparison. Correia and Martins (2019) is a monotonic model following the sequence-to-sequence architecture and pre-trained on BERT (seq2seq BERT). Bérard et al. (2017) predict a sequence of actions in a left-to-right order.

path. We compare different ground-truth action sequences based on minimum edit actions, all arriving at the same `pe`:

- Left-to-right (*l2r*);
- Randomly shuffled (*shuff*);
- Human-ordered (*h-ord*).

Minimum edit actions are generated using the dynamic programming algorithm to compute Levenshtein distance. The algorithm is set to output left-to-right actions, but since its output contains no redundant actions, they can be arbitrarily reordered. One simple way to re-order the actions is by randomly shuffling them. A more sophisticated alternative consists in matching each minimum-edit action to a human action, as described in §2.2.

We also experimented with unfiltered human actions. However this resulted in significantly inferior performance, possibly due to the hesitations made by humans typing, who may add and delete words unnecessary for the final `pe`.

Training details. We train the model by maximizing the likelihood of the action sequences provided as ground truth. Following Correia and Martins (2019) we use Adam (Kingma and Ba, 2015) with a triangular schedule, increasing linearly for the first 5,000 steps until 5×10^{-5}, applying a linear decay afterwards. BERT components have ℓ_2 weight decay of 10^{-4}. We apply dropout (Srivastava et al., 2014) with $p_{drop} = 0.1$ and, for the loss of vocabulary distribution, label-smoothing with $\epsilon = 0.1$ (Pereyra et al., 2017). We use batch size of 512 tokens and save checkpoints every 10,000 steps.

5 Experiments

We compare the effect of using different action orders on the development set and test sets of WMT 2018 APE shared task (Chatterjee et al., 2018).

By using only training data overlapping with WMT's training sets (as described in §2.1), we are able to use WMT's development and test sets for evaluation. This allows to compare the performance of our model with that of previous submissions. Note however that our systems are in disadvantage, due to being trained on fewer data: out of the original 23,000 training samples we only found 16,068 in Specia et al. (2017).

5.1 Performance of models

We explore three different ways to order the actions provided by minimum edit distance: *l2r*, *shuff* and *h-ord*. For each run, we pick the best checkpoint measured by TER in the development set, and evaluate on 3 test sets. Table 5 shows the average and standard deviation of 5 runs. Depending on the dataset chosen, the best performance is achieved by either *l2r* or *h-ord*, with small variations between the two. Random shuffling is consistently worse than the alternatives, by a margin of around 0.3 TER. All three alternatives significantly improve the uncorrected MT baseline.

To compare with existing results, we choose two models. Correia and Martins (2019) use an archi-

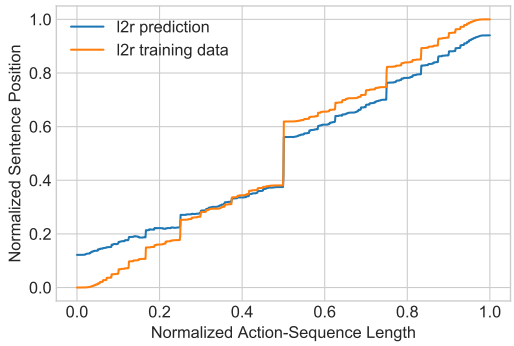

Figure 4: Relative positions of actions in a sequence, as explained in Figure 1. Comparing the original full left-to-right training data curve (orange) with the model predictions (blue), we see that the model became slightly non-monotonic.

tecture based on a monotonic autoregressive decoder (factorized in a left-to-right order). They propose a strategy that leverages on the pretrained BERT transformer (Devlin et al., 2019), achieving performance gains with it. Monotonic autoregressive models typically achieve a superior performance than their non-monotonic and non-autoregressive counterparts (Zhou et al., 2020). Bérard et al. (2017) propose a model that predicts a sequence of actions, which is closer to our approach, although they impose a left-to-right order. As expected, we do not outperform the results of the monotonic autoregressive model. However, we beat Bérard's action-based model, even though we use a smaller training set due to requiring keystrokes (16,086 samples instead of 23,000). This gain could be due to the pretraining of the BERT encoder used in our model, but also because of a largely different architeture (e.g. we use a Transformer encoder instead of a LSTM).

5.2 Learned orderings

Regarding the orderings learned by our model, they largely resemble the behaviour of the training data. Similar values for Kendall's τ distance indicate a similar amount of non-monotonicity in each of the three scenarios, as seen in Table 6. The only exception is the left-to-right model which, unlike the training data, becomes slightly non-monotonic during inference time. This is shown by the increase in Kendall's τ distance and illustrated by Figure 4. This imprecision in the decoding order may be expected since the model does not have an explicit memory of what has already been done.

	Kendall's τ	Δ K's τ	%loops	%do-noth
l2r	0.04	+0.04	15.6	10.5
shuff	0.50	+0.02	12.9	11.2
h-ord	0.16	0.00	16.0	9.8

Table 6: Statistics on the actions predicted by the 3 different models, measured on a single run of the development set. Δ K's τ refers to the difference between Kendall's τ distance of the model's output and of the corresponding training data. %loops counts samples that entered a loop, and %do-noth counts samples where the first predicted action was STOP.

6 Related Work

Non-monotonic models. Recent work explored alternatives for neural sequence generation that do not impose a left-to-right generation order. On the one hand, this allows for bidirectional attention to both left and right context of the token being generated. On the other hand, it is a more challenging task since it implies learning a generation order from a number of possibilities that grows exponentially. Generation order is usually treated as a latent variable, and our work differs in that we use supervision from human post-editors.

Gu et al. (2019a) propose an insertion-based model which avoids re-encoding by using relative attention, and has two ways of learning order: one using pre-defined orders, the other searching for orders that maximize the sequence likelihood, given the current model parameters. Emelianenko et al. (2019) train using sampled orders instead, to better escape local optima. They also drop the relative attention mechanism together with its better theoretical bound on time complexity – showing that, in practice, inference remains feasible.

Welleck et al. (2019) propose a model that generates text as a binary tree. They learn order from a uniform distribution that slowly shifts to search among the model's own preferences, or alternatively using a deterministic left-to-right oracle.

Lawrence et al. (2019) use placeholders to represent yet-to-insert tokens, allowing for bidirectional attention without exposing future tokens. Decoding is either done left-to-right or by picking the most certain prediction. Alternatively all tokens can be decided in parallel, but with significant loss in performance.

Non-autoregressive models. Another class of models focuses on parallel decoding of multiple tokens, moving away from the traditional autoregressive paradigm. This unlocks faster inference, but brings the difficult challenge of learning dependencies between tokens (Gu et al., 2018). Stern et

al. (2019) explore both non-monotonic autoregressive and non-autoregressive decoding with the Insertion Transformer. They use loss functions that promote either left-to-right order, a uniform distribution or a balanced binary tree for maximal parallelization.

The recently proposed Levenshtein Transformer (Gu et al., 2019b) introduces a Delete operation, and can generate or refine text by iterating between parallel insertions and parallel deletions — allowing to tackle the task of MT and also APE. Ruis et al. (2020) add a Delete operation to the Insertion Transformer and evaluate on artificial tasks. Our work differs in that we keep our model autoregressive, tackle the non-monotonicity by providing supervision to the order, analyze learned orders and focus on the APE task.

In general, this class of models is difficult to train and relies on several tricks. Knowledge distillation can bring improvements (Zhou et al., 2020), recently allowing Levenshtein Transformer to close the gap in translation quality between autoregressive monotonic and non-autoregressive models. In our setup, the tools proposed by Zhou et al. (2020) to measure data complexity could be used, for instance, for filtering out samples which are too complex.

Automatic post-editing. APE was initially proposed to combine rule-based translation systems with statistical phrase-based post-editing (Simard et al., 2007). As the quality of MT systems improves, there is less benefit in post-editing its mistakes, in particular if the MT system is trained on in-domain data. Current neural MT systems tend to generate very fluent output, therefore to fix their mistakes it is not enough to look at the `mt` output, but more deeply seek information from the `src` to fix adequacy errors. Top-performing systems for post-editing currently rely on tricks such as round-trip translation (Junczys-Dowmunt and Grundkiewicz, 2016) to increase dataset size, leveraging on pre-trained models (Correia and Martins, 2019) and using conservativeness penalties (Lopes et al., 2019) to avoid over-editing. Bérard et al. (2017) post-edit by predicting a sequence of actions with an imposed left-to-right order. Another recent work directly models edits, without including order information but allowing to re-use edits in unseen contexts (Yin et al., 2019).

Human post-editing. Previous work has explored keystrokes to understand the behavior of human editors. O'Brien (2006) investigates the relationship between pauses and cognitive effort, while later research (Lacruz et al., 2012; Lacruz and Shreve, 2014) examines keystroke logs for the same effect. Specia et al. (2017) introduce a dataset of human post-edits, containing information on keytrokes. Recently it was shown that detailed information from post-editing, such as sequences of edit-operations combined with mouseclicks and waiting times, contain structured information (Góis and Martins, 2019). The same work provides evidence that this kind of information allows to identify and profile editors, and may be helpful in downstream tasks.

7 Conclusions

In this work we explored different ways to order the edit operations necessary to fix mistakes in a translated sentence. In particular, we studied which orderings are produced by humans, and whether they can be used to guide the training of a non-monotonic post-editing system.

We found that humans tend to use nearly left-to-right order, although with exceptions, such as preferring to fix punctuation and verbs first. We then proposed a Transformer-based model pre-trained with BERT that learns to automatically post-edit translations in a flexible order. We learned this model in three different ways: by supervising it with orderings performed by humans, by using a left-to-right ordering, or with random orderings. In all three settings, the model outperformed the uncorrected machine translation baseline and a previous system also designed to predict actions (Bérard et al., 2017).

Training the model with human orderings achieved performance equivalent to left-to-right, or even superior. The random order consistently yielded slightly lower results. The model learned to mimic the proposed orders in all three cases.

Acknowledgments

We would like to thank Iacer Calixto, António Lopes, Fábio Kepler, Sean Welleck, Nikita Nangia, and the anonymous reviewers for their insightful comments. This work was partially supported by the EU/FEDER programme under PT2020 (contracts 027767 and 038510) and by the European Research Council (ERC StG DeepSPIN 758969).

References

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

Bérard, Alexandre, Laurent Besacier, and Olivier Pietquin. 2017. LIG-CRIStAL submission for the WMT 2017 automatic post-editing task. In *Proceedings of the Second Conference on Machine Translation*, pages 623–629, Copenhagen, Denmark, September. Association for Computational Linguistics.

Chatterjee, Rajen, José GC de Souza, Matteo Negri, and Marco Turchi. 2016. The fbk participation in the wmt 2016 automatic post-editing shared task. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 745–750.

Chatterjee, Rajen, Matteo Negri, Raphael Rubino, and Marco Turchi. 2018. Findings of the wmt 2018 shared task on automatic post-editing. In *Proceedings of the Third Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 723–738, Belgium, Brussels, October. Association for Computational Linguistics.

Correia, Gonçalo M and André FT Martins. 2019. A simple and effective approach to automatic post-editing with transfer learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3050–3056.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Emelianenko, Dmitrii, Elena Voita, and Pavel Serdyukov. 2019. Sequence modeling with unconstrained generation order. In *Advances in Neural Information Processing Systems*, pages 7698–7709.

Góis, António and André FT Martins. 2019. Translator2vec: Understanding and representing human post-editors. In *Proceedings of Machine Translation Summit XVII Volume 1: Research Track*, pages 43–54.

Gu, Jiatao, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

Gu, Jiatao, Qi Liu, and Kyunghyun Cho. 2019a. Insertion-based decoding with automatically inferred generation order. *Transactions of the Association for Computational Linguistics*, 7:661–676.

Gu, Jiatao, Changhan Wang, and Junbo Zhao. 2019b. Levenshtein transformer. In *Advances in Neural Information Processing Systems*, pages 11179–11189.

Junczys-Dowmunt, Marcin and Roman Grundkiewicz. 2016. Log-linear combinations of monolingual and bilingual neural machine translation models for automatic post-editing. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 751–758.

Kendall, Maurice G. 1938. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93.

Kingma, Diederik P. and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Klein, Guillaume, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada, July. Association for Computational Linguistics.

Koehn, Philipp, 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 Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June. Association for Computational Linguistics.

Lacruz, Isabel and Gregory M. Shreve. 2014. Pauses and cognitive effort in post-editing. *Post-editing of machine translation: Processes and applications*, page 246.

Lacruz, Isabel, Gregory M Shreve, and Erik Angelone. 2012. Average pause ratio as an indicator of cognitive effort in post-editing: A case study. In *AMTA 2012 Workshop on Post-Editing Technology and Practice (WPTP 2012)*, pages 21–30. AMTA.

Lawrence, Carolin, Bhushan Kotnis, and Mathias Niepert. 2019. Attending to future tokens for bidirectional sequence generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1–10, Hong Kong, China, November. Association for Computational Linguistics.

Lopes, António V, M Amin Farajian, Gonçalo M Correia, Jonay Trénous, and André FT Martins. 2019. Unbabel's submission to the wmt2019 ape shared task: Bert-based encoder-decoder for automatic

post-editing. In *Proceedings of the Fourth Conference on Machine Translation (Volume 3: Shared Task Papers, Day 2)*, pages 118–123.

O'Brien, Sharon. 2006. Pauses as indicators of cognitive effort in post-editing machine translation output. *Across Languages and Cultures*, 7(1):1–21.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035. Curran Associates, Inc.

Pereyra, Gabriel, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. 2017. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*.

Ruis, Laura, Mitchell Stern, Julia Proskurnia, and William Chan. 2020. Insertion-deletion transformer. *arXiv preprint arXiv:2001.05540*.

Simard, Michel, Nicola Ueffing, Pierre Isabelle, and Roland Kuhn. 2007. Rule-based translation with statistical phrase-based post-editing. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 203–206, Prague, Czech Republic, June. Association for Computational Linguistics.

Specia, Lucia, Kim Harris, Aljoscha Burchardt, Marco Turchi, Matteo Negri, and Inguna Skadina. 2017. Translation quality and productivity: A study on rich morphology languages. In *Machine Translation Summit XVI*, pages 55–71.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Stern, Mitchell, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pages 5976–5985.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Welleck, Sean, Kianté Brantley, Hal Daumé III, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation. In *International Conference on Machine Learning*, pages 6716–6726.

Wolf, Thomas, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Yin, Pengcheng, Graham Neubig, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L. Gaunt. 2019. Learning to represent edits. In *International Conference on Learning Representations*.

Zhou, Chunting, Jiatao Gu, and Graham Neubig. 2020. Understanding knowledge distillation in non-autoregressive machine translation. In *International Conference on Learning Representations*.