Seq2Edits: Sequence Transduction Using Span-level Edit Operations

Felix Stahlberg and Shankar Kumar
Google Research
{fstahlberg,shankarkumar}@google.com

Abstract

We propose Seq2Edits, an open-vocabulary approach to sequence editing for natural language processing (NLP) tasks with a high degree of overlap between input and output texts. In this approach, each sequence-to-sequence transduction is represented as a sequence of edit operations, where each operation either replaces an entire source span with target tokens or keeps it unchanged. We evaluate our method on five NLP tasks (text normalization, sentence fusion, sentence splitting & rephrasing, text simplification, and grammatical error correction) and report competitive results across the board. For grammatical error correction, our method speeds up inference by up to 5.2x compared to full sequence models because inference time depends on the number of edits rather than the number of target tokens. For text normalization, sentence fusion, and grammatical error correction, our approach improves explainability by associating each edit operation with a human-readable tag.

1 Introduction

Neural models that generate a target sequence conditioned on a source sequence were initially proposed for machine translation (MT) (Sutskever et al., 2014; Kalchbrenner and Blunsom, 2013; Bahdanau et al., 2015; Vaswani et al., 2017), but are now used widely as a central component of a variety of NLP systems (e.g. Tan et al. (2017); Chollampatt and Ng (2018)). Raffel et al. (2019) argue that even problems that are traditionally not viewed from a sequence transduction perspective can benefit from massive pre-training when framed as a text-to-text problem. However, for many NLP tasks such as correcting grammatical errors in a sentence, the input and output sequence may overlap significantly. Employing a full sequence model in these cases is often wasteful as most tokens are simply copied over from the input to the output. Another disadvantage of a full sequence model is that it does not provide an explanation for why it proposes a particular target sequence.

In this work, inspired by a recent increased interest in text-editing (Dong et al., 2019; Malmi et al., 2019; Mallinson et al., 2020; Awasthi et al., 2019), we propose *Seq2Edits*, a sequence *editing* model which is tailored towards problems that require only small changes to the input. Rather than generating the target sentence as a series of tokens, our model predicts a sequence of edit operations that, when applied to the source sentence, yields the target sentence. Each edit operates on a span in the source sentence and either copies, deletes, or replaces it with one or more target tokens. Edits are generated auto-regressively from left to right using a modified Transformer (Vaswani et al., 2017) architecture to facilitate learning of long-range dependencies. We apply our edit operation based model to five NLP tasks: text normalization, sentence fusion, sentence splitting & rephrasing, text simplification, and grammatical error correction (GEC). Our model is competitive across all of these tasks, and improves the state-of-the-art on text normalization (Sproat and Jaitly, 2016), sentence splitting & rephrasing (Botha et al., 2018), and the JFLEG test set (Napoles et al., 2017) for GEC.

Our model is often much faster than a full sequence model for these tasks because its runtime depends on the number of edits rather than the target sentence length. For instance, we report speed-ups of $>$5x on GEC for native English in initial experiments. If applicable, we also predict a task-specific edit-type class ("tag") along with each edit which explains why that edit was proposed. For example in GEC, the correction of a misspelled word would be labelled with a `SPELL` (spelling error) whereas changing a word from say first person to third person would be associated with a tag such as `SVA` (subject-verb agreement error).

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 5147–5159,
November 16–20, 2020.

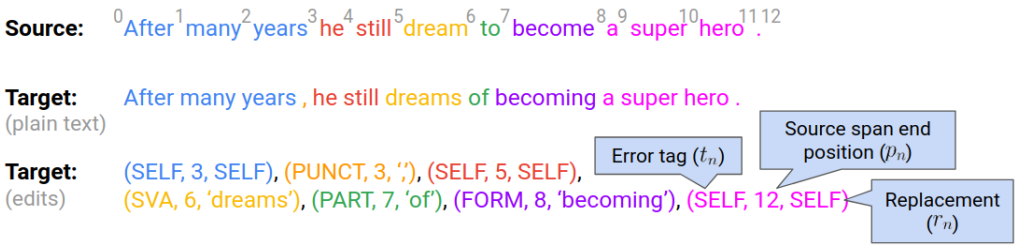

Figure 1: Representing grammatical error correction as a sequence of span-based edit operations. The implicit start position for a source span is the end position of the previous edit operation. SELF indicates spans that are copied over from the source sentence ($\mathbf{x}$). The probability of the first two edits is given by: $P(\text{After many years ,}|\mathbf{x}) = P(t_1 = \text{SELF}|\mathbf{x}) \cdot P(p_1 = 3|\text{SELF}, \mathbf{x}) \cdot P(r_1 = \text{SELF}|\text{SELF}, 3, \mathbf{x}) \cdot P(t_2 = \text{PUNCT}|\text{SELF}, 3, \text{SELF}, \mathbf{x}) \cdot P(p_2 = 3|\text{SELF}, 3, \text{SELF}, \text{PUNCT}, \mathbf{x}) \cdot P(r_2 = ,|\text{SELF}, 3, \text{SELF}, \text{PUNCT}, 3, \mathbf{x})$.

2 Edit-based Sequence Transduction

2.1 Representation

A vanilla sequence-to-sequence (seq2seq) model generates a plain target sequence $\mathbf{y} = y_1^J = y_1, y_2, \ldots, y_J \in V^J$ of length J given a source sequence $\mathbf{x} = x_1^I = x_1, x_2, \ldots, x_I \in V^I$ of length I over a vocabulary V of tokens (e.g. subword units (Sennrich et al., 2016)). For example, in our running grammar correction example in Fig. 1, the source sequence is the ungrammatical sentence $\mathbf{x}$ ="*After many years he still dream to become a super hero* ." and the target sequence is the corrected sentence $\mathbf{y}$ ="*After many years , he still dreams of becoming a super hero* .". The probability $P(\mathbf{y}|\mathbf{x})$ is factorized using the chain rule:

$$P(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^{J} P(y_j|y_1^{j-1}, \mathbf{x}). \quad (1)$$

Instead of predicting the target sequence $\mathbf{y}$ directly, the *Seq2Edits* model predicts a sequence of N edit operations. Each edit operation $(t_n, p_n, r_n) \in T \times \mathbb{N}_0 \times V$ is a 3-tuple that represents the action of replacing the span from positions p_{n-1} to p_n in the source sentence with the replacement token r_n associated with an explainable tag t_n (T is the tag vocabulary).[1] $t_n = r_n = \texttt{SELF}$ indicates that the source span is kept as-is. Insertions are modelled with $p_n = p_{n-1}$ that corresponds to an empty source span (see the insertion of "," in Fig. 1), deletions are represented with a special token $r_n = \texttt{DEL}$. The edit operation sequence for our running example is shown in Fig. 1. The target sequence $\mathbf{y}$ can be obtained from the edit operation sequence using Algorithm 1.

[1] In our ablation experiments without tags, each edit operation is represented by a 2-tuple (p_n, r_n) instead.

Algorithm 1 ApplyEdits()

```
p_0 ← 0 {First span starts at 0.}
y ← ϵ {Initialize y with the empty string.}
for n ← 1 to N do
    if t_n = SELF then
        y ← concat(y, x_{p_{n-1}}^{p_n})
    else if r_n ≠ DEL then
        y ← concat(y, r_n)
    end if
end for
return y
```

Our motivation behind using span-level edits rather than token-level edits is that the representations are much more compact and easier to learn since local dependencies (within the span) are easier to capture. For some of the tasks it is also more natural to approach the problem on the span-level: a grammatical error is often fixed with more than one (sub)word, and span-level edits retain the language modelling aspect within a span.

Our representation is flexible as it can represent any sequence pair. As an example, a trivial (but not practical) way to construct an edit sequence for any pair $(\mathbf{x}, \mathbf{y})$ is to start with a deletion for the entire source sentence $\mathbf{x}$ ($p_1 = I$, $r_1 = \texttt{DEL}$) and then insert the tokens in $\mathbf{y}$ ($p_{j+1} = I$, $r_{j+1} = y_j$ for $j \in [1, J]$).

Edit sequences are valid iff. spans are in a monotonic left-to-right order and the final span ends at the end of the source sequence, i.e.:

$$p_N = I \wedge \forall n \in [1, N) : p_n \leq p_{n+1} \quad (2)$$

None of our models produced invalid sequences at inference time even though we did not implement any feasibility constraints.

2.2 Inference

The output of the edit operation model is a sequence of 3-tuples rather than a sequence of tokens. The probability of the output is computed as:

$$P(\mathbf{y}|\mathbf{x}) = P(t_1^N, p_1^N, r_1^N|\mathbf{x}) = \prod_{n=1}^{N} P(t_n, p_n, r_n|t_1^{n-1}, p_1^{n-1}, r_1^{n-1}, \mathbf{x}). \quad (3)$$

For inference, we factorize the conditional probabilities further as:

$$\begin{aligned} &P(t_n, p_n, r_n|t_1^{n-1}, p_1^{n-1}, r_1^{n-1}, \mathbf{x}) \\ =&P(t_n|t_1^{n-1}, p_1^{n-1}, r_1^{n-1}, \mathbf{x}) \\ &\cdot P(p_n|t_1^{n}, p_1^{n-1}, r_1^{n-1}, \mathbf{x}) \\ &\cdot P(r_n|t_1^{n}, p_1^{n}, r_1^{n-1}, \mathbf{x}). \end{aligned} \quad (4)$$

The decoding problem can thus be written as a flat product of conditional probabilities that correspond to tag, span and replacement predictions, that are interleaved:

$$\begin{aligned} &\arg\max_{N, t_1^N, p_1^N, r_1^N} P(t_1|\mathbf{x}) \cdot P(p_1|t_1, \mathbf{x}) \cdot P(r_1|t_1, p_1, \mathbf{x}) \\ &\cdot P(t_2|t_1, p_1, r_1, \mathbf{x}) \cdots P(r_N|t_1^N, p_1^N, r_1^{N-1}, \mathbf{x}). \end{aligned} \quad (5)$$

At inference time we perform beam decoding over this flat factorization to search for the most likely edit operation sequence. In practice, we scale the scores for the different target features:

$$\begin{aligned} \arg\max_{N, t_1^N, p_1^N, r_1^N} \sum_{n=1}^{N} &\lambda_t \log P(t_n|t_1^{n-1}, p_1^{n-1}, r_1^{n-1}, \mathbf{x}) \\ &+ \lambda_p \log P(p_n|t_1^{n}, p_1^{n-1}, r_1^{n-1}, \mathbf{x}) \\ &+ \lambda_r \log P(r_n|t_1^{n}, p_1^{n}, r_1^{n-1}, \mathbf{x}). \end{aligned} \quad (6)$$

where the three scaling factors λ_t, λ_p, λ_r are optimized on the respective development set.

2.3 Neural Architecture

Our neural model (illustrated in Fig. 2) is a generalization of the original Transformer architecture of Vaswani et al. (2017). Similarly to the standard Transformer we feed back the predictions of the previous time step into the Transformer decoder (A). The feedback loop at time step n is implemented as the concatenation of an embedding of t_{n-1}, the p_{n-1}-th encoder state, and an embedding of r_{n-1}. The Transformer decoder A is followed by a cascade of tag prediction and span end position prediction. We follow the idea of pointer networks (Vinyals et al., 2015) to predict the source span end position using the attention weights over encoder states as probabilities. The input to the pointer network are the encoder states (keys and values) and the output of the previous decoder layer (queries). The span end position prediction module is a Transformer-style (Vaswani et al., 2017) single-head attention ("scaled dot-product") layer over the encoder states:

$$P(p_n|t_1^{n}, p_1^{n-1}, r_1^{n-1}, \mathbf{x}) = \text{softmax}(\frac{QK^T}{\sqrt{d}}), \quad (7)$$

where Q is a d-dimensional linear transform of the previous decoder layer output at time step n, $K \in \mathbb{R}^{I \times d}$ is a linear transform of the encoder states, and d is the number of hidden units.

A 6-dimensional embedding of the predicted tag t_n and the encoder state corresponding to the source span end position p_n are fed into yet another Transformer decoder (B) that predicts the replacement token r_n. Alternatively, one can view A and

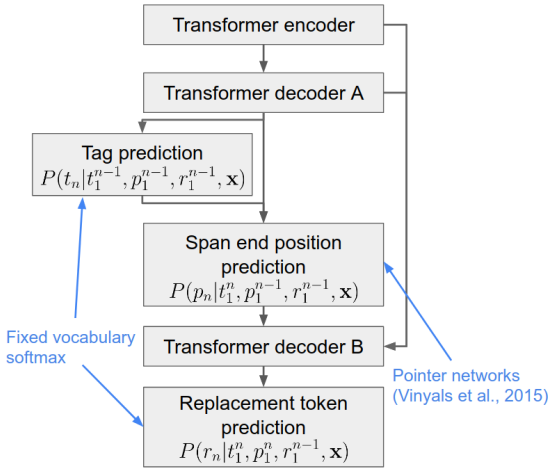

Figure 2: *Seq2Edits* consists of a Transformer (Vaswani et al., 2017) encoder and a Transformer decoder that is divided horizontally into two parts (A and B). The tag and span predictions are located in the middle of the decoder layer stack between both parts. A single step of prediction is shown.

Hyper-parameter	Base	Big
Hidden units	512	1,024
Encoder layers	6	6
Decoder A layers	3	4
Decoder B layers	3	4
No. of parameters (w/o embeddings)	53M	246M

Table 1: The "Base" and "Big" configurations.

	Text normalization		Sentence fusion	Sentence splitting	Simplification	Grammar correction
	English	Russian				
Training data	Wikipedia		DiscoFuse	WikiSplit	WikiLarge	Lang-8, FCE, W&I
Number of sentences	881K	816K	4.5M	990K	296K	2M
Task-specific tags	Semiotic class		Type	-	-	Error tag
Task-specific tag vocabulary size	18	15	15	-	-	28
Source tokenization	Characters		Subwords	Subwords	Subwords	Subwords
Target tokenization	Words		Subwords	Subwords	Subwords	Subwords
Fraction of changed tokens per sentence	9.9%	15.7%	13.9%	8.7%	52.0%	10.5%
Average source length in tokens (I)	64.9	82.4	36.9	39.8	31.7	14.4
Average target length in tokens (J)	57.3	69.4	31.8	36.5	15.8	13.0
Average number of span-level edits (N)	9.6	14.2	7.4	11.8	13.1	4.7

Table 2: Statistics for the task-specific training data. The I, J, and N variables are introduced in Sec. 2.1. Our subword-based systems use the implementation available in Tensor2Tensor (Vaswani et al., 2018) with a vocabulary size of 32K. The pre-training data is described in the text. See Appendix A for the full tag vocabularies.

B as a single Transformer decoder layer stack, in which we added the tag prediction and the span end position prediction as additional layers in the middle of that stack. We connect the decoders A and B with residual connections (He et al., 2016) to facilitate learning. The network is trained by optimizing the sum of three cross-entropies that correspond to tag prediction, span prediction, and replacement token prediction, respectively.[2] In our experiments without tags we leave out the tag prediction layer and the loss computed from it. We experiment with two different model sizes: "Base" and "Big". The hyper-parameters for both configurations are summarized in Table 1. Hyper-parameters not listed here are borrowed from the *transformer_clean_base* and *transformer_clean_big* configurations in the Tensor2Tensor toolkit (Vaswani et al., 2018).

3 Experiments

We evaluate our edit model on five NLP tasks:[3]

- Text normalization for speech applications (Sproat and Jaitly, 2016) – converting number expressions such as "*123*" to their verbalizations (e.g. "*one two three*" or "*one hundred twenty three*", etc.) depending on the context.

- Sentence fusion (Geva et al., 2019) – merging two independent sentences to a single coherent one, e.g. "*I need his spirit to be free. I can leave my body.*" → "*I need his spirit to be free so that I can leave my body.*"

- Sentence splitting & rephrasing (Botha et al., 2018) – splitting a long sentence into two fluent sentences, e.g. "*Bo Saris was born in Venlo, Netherlands, and now resides in London, England.*" → "*Bo Saris was born in Venlo , Netherlands. He currently resides in London, England.*"

- Text simplification (Zhang and Lapata, 2017) – reducing the linguistic complexity of text, e.g. "*The family name is derived from the genus Vitis.*" → "*The family name comes from the genus Vitis.*"

- Grammatical error correction (Ng et al., 2014; Napoles et al., 2017; Bryant et al., 2019) – correcting grammatical errors in written text, e.g. "*In a such situaction*" → "*In such a situation*".

Our models are trained on packed examples (maximum length: 256) with Adafactor (Shazeer and Stern, 2018) using the Tensor2Tensor (Vaswani et al., 2018) library. We report results both with and without pre-training. Our pre-trained models for all tasks are trained for 1M iterations on 170M sentences extracted from English Wikipedia revisions and 176M sentences from English Wikipedia round-trip translated via German (Lichtarge et al., 2019). For grammatical error correction we use ERRANT (Bryant et al., 2017; Felice et al., 2016) to derive span-level edits from the parallel text. On other tasks we use a minimum edit distance heuristic to find a token-level edit sequence and convert it to span-level edits by merging neighboring edits.

The task-specific data is described in Table 2. The number of iterations in task-specific training is set empirically based on the performance on the development set (between 20K-75K for fine-tuning,

[2] We use an unweighted sum as we did not observe gains from weighting the three losses during training.

[3] Citations in this bullet list refer to the test sets we used and do not necessarily point to the pioneering works.

Task	Feature weights (λ)	Iterative refinement	Length normalization	Identity penalty
Text normalization	✓			
Sentence fusion	✓			
Sentence splitting	✓			
Simplification	✓	✓	✓	✓
Grammar correction	✓	✓		✓

Table 3: Decoding parameters that are tuned on the respective development sets. Feature weight tuning refers to the λ-parameters in Sec. 2.2. We use the length normalization scheme from Wu et al. (2016) with the parameter α.

	Model size	Tags	Tuning	Pre-training	Text norm. (SER↓) English	Text norm. (SER↓) Russian	Fusion (SARI↑)	Splitting (SARI↑)	Simpl. (SARI↑)	Grammar (BEA-dev) P↑	Grammar (BEA-dev) R↑	Grammar (BEA-dev) $F_{0.5}$ ↑
a	Base				1.40	4.13	87.15	58.9	31.87	23.3	11.0	19.0
b	Base	✓			1.36	3.95	87.33	58.9	32.10	22.5	13.3	19.8
c	Big			✓	-	-	88.77	63.5	33.01	50.1	34.4	45.9
d	Big	✓		✓	-	-	88.67	63.6	34.54	53.7	35.3	48.6
e	Big		✓	✓	-	-	88.73	63.6	37.16	49.0	38.6	46.5
f	Big	✓	✓	✓	-	-	88.72	63.6	36.30	50.9	39.1	48.0

Table 4: Single model results. For metrics marked with "↑" (SARI, P(recision), R(ecall), $F_{0.5}$) high scores are favorable, whereas the sentence error rate (SER) is marked with "↓" to indicate the preference for low values. Tuning refers to optimizing the decoding parameters listed in Table 3 on the development sets.

between 100K-300K for training from scratch). Fine-tuning is performed with a reduced learning rate of 3×10^{-5}. For each task we tune a different set of decoding parameters (Table 3) such as the λ-parameters from Sec. 2.2, on the respective development sets. For text simplification and grammatical error correction, we perform multiple beam search passes (between two and four) by feeding back the output of beam search as input to the next beam search pass. This is very similar to the iterative decoding strategies used by Lichtarge et al. (2019); Awasthi et al. (2019); Ge et al. (2018); Grundkiewicz et al. (2019) with the difference that we pass through n-best lists between beam search passes rather than only the single best hypothesis. During iterative refinement we follow Lichtarge et al. (2019) and multiply the score of the identity mapping by a tunable identity penalty to better control the trade-off between precision and recall. We use a beam size of 12 in our rescoring experiments in Table 10 and a beam size of 4 otherwise.

Table 4 gives an overview of our results on all tasks. The tag set consists of semiotic class labels (Sproat and Jaitly, 2016) for text normalization, discourse type (Geva et al., 2019) for sentence fusion, and ERRANT (Bryant et al., 2017; Felice et al., 2016) error tags for grammatical error correction.[4] For the other tasks we use a trivial tag set: `SELF`, `NON_SELF`, and `EOS` (end of sequence). We report sentence error rates (SERs↓) for text normalization, SARI↑ scores (Xu et al., 2016) for sentence fusion, splitting, and simplification, and ERRANT (Bryant et al., 2017) span-based P(recision)↑, R(ecall)↑, and $F_{0.5}$-scores on the BEA development set for grammar correction.[5]

Text normalization is not amenable to our form of pre-training as it does not use subword units and it aims to generate vocalizations rather than text like in our pre-training data. All other tasks, however, benefit greatly from pre-training (compare rows **a** & **b** with rows **c** & **d** in Table 4). Pre-training yields large gains for grammar correction as the pre-training data was specifically collected for this task (Lichtarge et al., 2019). Tuning the decoding parameters (listed in Table 3) gives improvements for tasks like simplification, but is less crucial on sentence fusion or splitting (compare rows **c** & **d** with rows **e** & **f** in Table 4). Using tags is especially effective if non-trivial tags are available (compare rows **a** with **b**, **c** with **d**, and **e** with **f** for text normalization and grammar correction).

We next situate our best results (big models with pre-training in rows **e** and **f** of Table 4) in the context of related work.

3.1 Text Normalization

Natural sounding speech synthesis requires the correct pronunciation of numbers based on context e.g. whether the string *123* should be spoken as *one hundred twenty three* or *one two three*. Text normalization converts the textual representation of numbers or other semiotic classes such as abbre-

[4] Appendix A lists the full task-specific tag vocabularies.

[5] Metrics are marked with "↑" if high values are preferred and with "↓" if low values are preferred.

System	SER↓	
	English	Russian
Mansfield et al. (2019)*	2.77	-
Zhang et al. (2019)	1.80	4.20
This work (semiotic tags)	**1.36**	**3.95**

Table 5: Sentence error rates on the English and Russian text normalization test sets of Sproat and Jaitly (2016). *: best system from Mansfield et al. (2019) without access to gold semiotic class labels.

System	Exact↑	SARI↑
Malmi et al. (2019)	53.80	85.45
Mallinson et al. (2020)	61.31	88.78
Rothe et al. (2019)	**63.90**	**89.52**
This work (no tags)	61.71	88.73

Table 6: Sentence fusion results on the DiscoFuse (Geva et al., 2019) test set.

viations to their spoken form while both conveying meaning and morphology (Zhang et al., 2019). The problem is highly context-dependent as abbreviations and numbers often have different feasible vocalizations. Context-dependence is even more pronounced in languages like Russian in which the number words need to be inflected to preserve agreement with context words.

We trained our models on the English and Russian data provided by Sproat and Jaitly (2016). Similarly to others (Zhang et al., 2019; Sproat and Jaitly, 2016) we use characters on the input but full words on the output side. Table 5 shows that our models perform favourably when compared to other systems from the literature on both English and Russian. Note that most existing neural text normalization models (Zhang et al., 2019; Sproat and Jaitly, 2016) require pre-tokenized input[6] whereas our edit model operates on the untokenized input character sequence. This makes our method attractive for low resource languages where high-quality tokenizers may not be available.

3.2 Sentence Fusion

Sentence fusion is the task of merging two independent sentences into a single coherent text and has applications in several NLP areas such as dialogue systems and question answering (Geva et al., 2019). Our model is on par with the FELIX tagger (Mallinson et al., 2020) on the DiscoFuse dataset (Geva et al., 2019) but worse than the BERT2BERT model of Rothe et al. (2019) (Table 6). We hypothesize that BERT2BERT's strategy of making use of target-side pre-training under a language model objective via BERT (Devlin et al., 2019) is particularly useful for sentence fusion.

[6]Each token in this context is a full semiotic class instance, for example a full date or money expression.

System	Exact↑	SARI↑
Botha et al. (2018)	14.6	60.6
Malmi et al. (2019)	15.2	61.7
Malmi et al. (2019) - SEQ2SEQ$_{\text{BERT}}$	15.1	62.3
This work (no tags)	**17.0**	**63.6**

Table 7: Sentence splitting results (Botha et al., 2018).

System	SARI↑
Malmi et al. (2019)	32.31
Dong et al. (2019)	34.94
Xu et al. (2016)	37.94
Mallinson et al. (2020)	**38.13**
This work (no tags)	37.16

Table 8: Text simplification results.[7]

3.3 Sentence Splitting & Rephrasing

Sentence splitting is the inverse task of sentence fusion: Split a long sentence into two fluent shorter sentences. Our models are trained on the WikiSplit dataset (Botha et al., 2018) extracted from the edit history of Wikipedia articles. In addition to SARI scores we report the number of exact matches in Table 7. Our edit-based model achieves a higher number of exact matches and a higher SARI score compared to prior work on sentence splitting.

3.4 Text Simplification

Our text simplification training set (the WikiLarge corpus (Zhang and Lapata, 2017)) consists of 296K examples, the smallest amongst all our training corpora. Table 8 shows that our model is competitive, demonstrating that it can benefit from even limited quantities of training data. However, it does not improve the state of the art on this test set.

3.5 Grammatical Error Correction

For grammatical error correction we follow a multi-stage fine-tuning setup.[8] After training on the common pre-training data described above, we fine-tune for 30K steps on the public Lang-8 (Mizumoto et al., 2012) corpus, followed by 500 steps on the FCE (Yannakoudakis et al., 2011) and W&I (Bryant et al., 2019) corpora. To improve comparability with related work across the different corpora we use ERRANT (Bryant et al.,

[7]We report SARI scores as (re)computed by Mallinson et al. (2020) for all systems in Table 8 to ensure comparability.

[8]Multi-stage fine-tuning has been proven effective for other sequence tasks such as machine translation (Khan et al., 2018; Saunders et al., 2019).

System	BEA-dev			CoNLL-14			JFLEG
	P↑	R↑	$F_{0.5}$ ↑	P↑	R↑	$F_{0.5}$ ↑	GLEU↑
Lichtarge et al. (2019)	-	-	-	65.5	37.1	56.8	61.6
Awasthi et al. (2019)	-	-	-	66.1	43.0	59.7	60.3
Zhao et al. (2019)	-	-	-	67.7	40.6	59.8	-
Choe et al. (2019)	54.4	32.2	47.8	-	-	-	-
Grundkiewicz et al. (2019)	**56**.1	34.8	**50.0**	-	-	-	-
Kiyono et al. (2019)	-	-	-	**67.9**	44.1	**61.3**	59.7
This work (ERRANT tags)	50.9	**39.1**	48.0	63.0	**45.6**	58.6	**62.7**

Table 9: Single model results for grammatical error correction.

System	BEA-test			CoNLL-14			JFLEG
	P↑	R↑	$F_{0.5}$ ↑	P↑	R↑	$F_{0.5}$ ↑	GLEU↑
Lichtarge et al. (2019)	-	-	-	66.7	43.9	60.4	63.3
Awasthi et al. (2019)	-	-	-	68.3	43.2	61.2	61.0
Zhao et al. (2019)	-	-	-	71.6	38.7	61.2	61.0
Ge et al. (2018)	-	-	-	74.1	36.3	61.3	61.4
Choe et al. (2019)	**76.2**	50.3	69.1	**74.8**	34.1	60.3	-
Grundkiewicz et al. (2019)	72.3	60.1	69.5	-	-	64.2	61.2
Kiyono et al. (2019)	74.7	56.7	70.2	72.4	**46.1**	**65.0**	61.4
This work (ERRANT tags)							
5-Ensemble	68.8	**63.4**	67.7	72.0	39.4	61.7	64.2
+ Full sequence rescoring	72.7	62.9	**70.5**	69.9	44.4	62.7	**64.3**

Table 10: Ensemble results for grammatical error correction. Our full sequence baseline achieves 68.2 $F_{0.5}$ on BEA-test, 63.8 $F_{0.5}$ on CoNLL-14, and 62.4 GLEU on JFLEG-test.

2017; Felice et al., 2016) to compute span-based P(recision)↑, R(ecall)↑, and $F_{0.5}$-scores on the BEA development and test sets (Bryant et al., 2019), the M2 scorer (Dahlmeier and Ng, 2012) on the CoNLL-2014 (Ng et al., 2014) test set, and GLEU↑ on the JFLEG test set (Napoles et al., 2017). However, the training data used in the literature varies vastly from system to system which limits comparability as (synthetic) data significantly impacts the system performance (Grundkiewicz et al., 2019).

Table 9 compares our approach with the best single model results reported in the literature. Our model tends to have a lower precision but higher recall than other systems. We are able to achieve the highest GLEU score on the JFLEG test set.

To further improve performance, we applied two techniques commonly used for grammatical error correction (Grundkiewicz et al., 2019, inter alia): ensembling and rescoring. Table 10 compares our 5-ensemble with other ensembles in the literature. Rescoring the n-best list from the edit model with a big full sequence Transformer model yields significant gains, outperforming all other systems in Table 10 on BEA-test and JFLEG.[9]

One of our initial goals was to avoid the wasteful computation of full sequence models when applied to tasks like grammatical error correction with a high degree of copying. Table 11 summarizes CPU decoding times on an Intel® Xeon® W-2135 Processor (12-core, 3.7 GHz).[10] We break down the measurements by English proficiency level. The full sequence baseline slows down for higher proficiency levels as sentences tend to be longer (second column of Table 11). In contrast, our edit operation based approach is faster because it does not depend on the target sequence length but instead on the number of edits which is usually small for advanced and native English speakers. We report speed-ups of 4.7-4.8x in these cases without using tags. When using tags, we implemented the following simple heuristics to improve the runtime ("shortcuts"): 1) If the tag $t_n = \texttt{SELF}$ is predicted, directly set $r_n = \texttt{SELF}$ and skip the replacement token prediction. 2) If the tag $t_n = \texttt{EOS}$ is predicted, set $p_n = I$ and $r_n = \texttt{EOS}$ and skip the span end position and the replacement token predictions. These shortcuts do not affect the results in practice but provide speed-ups of 5.2x for advanced and native English compared to a full sequence model.

The speed-ups of our approach are mainly due to the shorter target sequence length compared to a full sequence model. However, our inference scheme in Sec. 2.2 still needs three times more

[9] This resembles the inverse setup of Chollampatt and Ng (2018) who used edit features to rescore a full sequence model.

[10] We note that our experimental decoder implementation is not optimized for speed, and that absolute runtimes may differ with more efficient implementations.

English proficiency	Avg. source length (I)	Avg. number of edits (N)	Sentences per second↑			
			Full sequence	Edit operation based		
				ERRANT tags	No tags	ERRANT tags (with shortcuts)
Beginner (CEFR-A)	20.4	7.8	0.34	0.55 (1.6x)	0.69 (2.0x)	0.69 (2.0x)
Intermediate (CEFR-B)	25.9	6.0	0.34	0.83 (2.4x)	1.04 (3.0x)	1.07 (3.1x)
Advanced (CEFR-C)	23.0	4.2	0.31	1.21 (3.9x)	1.46 (4.7x)	1.59 (**5.2x**)
Native	26.6	4.0	0.26	1.03 (4.0x)	1.24 (4.8x)	1.34 (**5.2x**)

Table 11: CPU decoding speeds without iterative refinement on BEA-dev averaged over three runs. Speed-ups compared to the full sequence baseline are in parentheses.

Oracle constraints		Text normalization (SER↓)		Grammar correction (BEA-dev)		
Tag (t_n)	Span end position (p_n)	English	Russian	P↑	R↑	$F_{0.5}$ ↑
		1.36	3.95	55.2	36.2	50.0
✓		0.36	3.63	58.6	53.5	57.5
	✓	0.48	3.58	64.9	65.5	65.0
✓	✓	0.24	3.58	71.9	71.9	71.9

Table 12: Partially constraining the decoder with oracle tags and/or span positions (no iterative refinement).

time steps than the number of edits, i.e. around $4.0 \times 3 = 12$ for native English (last row in Table 11). The observed speed-ups of 4.0x-5.2x are even larger than we would expect based on an average source length of $I = 26.6$. One reason for the larger speed-ups is that the decoding runtime complexity under the Transformer is quadratic in length, not linear. Another reason is that although the three elements are predicted sequentially, not each prediction step is equally expensive: the softmax for the tag and span predictions is computed over only a couple of elements, not over the full 32K subword vocabulary. Furthermore, efficient decoder implementations could reuse most of the computation done for the tag prediction for the span position.

3.6 Oracle Experiments

Our model can be viewed from a multi-task perspective since it tries to predict three different features (tag, span position, and replacement token). To better understand the contributions of these different components we partially constrained them using the gold references for both text normalization and grammatical error correction tasks. We avoid constraining the *number* of edits (N) in these oracle experiments by giving the constrained decoder the option to repeat labels in the reference. Table 12 shows that having access to the gold tags and/or span positions greatly improves performance. We hypothesize that these gains can be largely attributed to the resolution of confusion between self and non-self. An interesting outlier is text normalization on Russian which benefits less from oracle constraints. This suggests that the challenges for Russian text normalization are largely in predicting the replacement tokens, possibly due to the morphological complexity of Russian.

System	Tagging accuracy		
	P↑	R↑	$F_{0.5}$ ↑
Lasertagger	54.9	33.7	48.8
This work (unconstrained)	67.9	25.8	51.2
This work (span-constrained)	94.7	52.4	81.5

Table 13: Tagging accuracy on BEA-dev (no iterative refinement).

Since the motivation for using tags was to improve explainability we also evaluated the accuracy of the tag prediction on grammatical error correction. For comparison, we trained a baseline Lasertagger (Malmi et al., 2019) on a subset of the BEA training set (30.4K examples) to predict the ERRANT tags. Insertions are represented as composite tags together with the subsequent tag such that the total Lasertagger vocabulary size is 213. The model was initialized from a pre-trained BERT (Devlin et al., 2019) checkpoint. Decoding was performed using an autoregressive strategy with a Transformer decoder. We used the default hyper-parameters without any task-specific optimization. Table 13 shows that the tag prediction of our unconstrained model is more accurate than the Lasertagger baseline. Errors in this unconstrained setup are either due to predicting the wrong tag or predicting the wrong span. To tease apart these error sources we also report the accuracy under oracle span constraints. Our span constrained model achieves a recall of 52.4%, i.e. more than half of the non-self tags are classified correctly (28 tags).

4 Related Work

A popular way to tackle NLP problems with overlap between input and output is to equip seq2seq models with a copying mechanism (Jia and Liang, 2016; Zhao et al., 2019; Chen and Bansal, 2018; Nallapati et al., 2016; Gulcehre et al., 2016; See et al., 2017; Gu et al., 2016), usually borrowing ideas from pointer networks (Vinyals et al., 2015) to point to single tokens in the source sequence. In contrast, we use pointer networks to identify entire spans that are to be copied which results in a much more compact representation and faster decoding. The idea of using span-level edits has also been explored for morphological learning in the form of symbolic span-level rules, but not in combination with neural models (Elsner et al., 2019).

Our work is related to neural multi-task learning for NLP (Collobert and Weston, 2008; Dong et al., 2015; Luong et al., 2015; Søgaard and Goldberg, 2016). Unlike multi-task learning which typically solves separate problems (e.g. POS tagging and named entity recognition (Collobert and Weston, 2008) or translation into different languages (Luong et al., 2015; Dong et al., 2015)) with the same model, our three output features (tag, source span, and replacement) represent the same output sequence (Algorithm 1). Thus, it resembles the stack-propagation approach of Zhang and Weiss (2016) who use POS tags to improve parsing performance.

A more recent line of research frames sequence editing as a labelling problem using labels such as `ADD`, `KEEP`, and `DELETE` (Ribeiro et al., 2018; Dong et al., 2019; Mallinson et al., 2020; Malmi et al., 2019; Awasthi et al., 2019), often relying heavily on BERT (Devlin et al., 2019) pre-training. Similar operations such as insertions and deletions have also been used for machine translation (Gu et al., 2019b; Stern et al., 2019; Gu et al., 2019a; Östling and Tiedemann, 2017; Stahlberg et al., 2018). We showed in Sec. 3 that our model often performs similarly or better than those approaches, with the added advantage of providing explanations for its predictions.

5 Discussion

We have presented a neural model that represents sequence transduction using span-based edit operations. We reported competitive results on five different NLP problems, improving the state of the art on text normalization, sentence splitting, and the JFLEG test set for grammatical error correction. We showed that our approach is 2.0-5.2 times faster than a full sequence model for grammatical error correction. Our model can predict labels that explain each edit to improve the interpretability for the end-user. However, we do not make any claim that *Seq2Edits* can provide insights into the internal mechanics of the neural model. The underlying neural model in *Seq2Edits* is as much of a black-box as a regular full sequence model.

While our model is advantageous in terms of speed and explainability, it does have some weaknesses. Notably, the model uses a tailored architecture (Figure 2) that would require some engineering effort to implement efficiently. Second, the output of the model tends to be less fluent than a regular full sequence model, as can be seen from the examples in Table 19. This is not an issue for localized edit tasks such as text normalization but may be a drawback for tasks involving substantial rewrites (e.g. GEC for non-native speakers).

Even though our approach is open-vocabulary, future work will explore task specific restrictions. For example, in a model for dialog applications, we may want to restrict the set of response tokens to a predefined list. Alternatively, it may be useful to explore generation in a non left-to-right order to improve the efficiency of inference.

Another line of future work is to extend our model to sequence rewriting tasks, such as Machine Translation post-editing, that do not have existing error-tag dictionaries. This research would require induction of error tag inventories using either linguistic insights or unsupervised methods.

Acknowledgments

We thank Chris Alberti and Jared Lichtarge as well as the anonymous reviewers for their helpful comments.

References

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla. 2019. Parallel iterative edit models for local sequence transduction. 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 4260–4270, Hong Kong, China. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by

jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

Jan A. Botha, Manaal Faruqui, John Alex, Jason Baldridge, and Dipanjan Das. 2018. Learning to split and rephrase from Wikipedia edit history. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 732–737, Brussels, Belgium. Association for Computational Linguistics.

Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. 2019. The BEA-2019 shared task on grammatical error correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 52–75, Florence, Italy. Association for Computational Linguistics.

Christopher Bryant, Mariano Felice, and Ted Briscoe. 2017. Automatic annotation and evaluation of error types for grammatical error correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 793–805, Vancouver, Canada. Association for Computational Linguistics.

Yen-Chun Chen and Mohit Bansal. 2018. Fast abstractive summarization with reinforce-selected sentence rewriting. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 675–686, Melbourne, Australia. Association for Computational Linguistics.

Yo Joong Choe, Jiyeon Ham, Kyubyong Park, and Yeoil Yoon. 2019. A neural grammatical error correction system built on better pre-training and sequential transfer learning. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 213–227, Florence, Italy. Association for Computational Linguistics.

Shamil Chollampatt and Hwee Tou Ng. 2018. A multilayer convolutional encoder-decoder neural network for grammatical error correction. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML 08, page 160167, New York, NY, USA. Association for Computing Machinery.

Daniel Dahlmeier and Hwee Tou Ng. 2012. Better evaluation for grammatical error correction. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572, Montréal, Canada. Association for Computational Linguistics.

Jacob Devlin, 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. Association for Computational Linguistics.

Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. 2015. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1723–1732, Beijing, China. Association for Computational Linguistics.

Yue Dong, Zichao Li, Mehdi Rezagholizadeh, and Jackie Chi Kit Cheung. 2019. EditNTS: An neural programmer-interpreter model for sentence simplification through explicit editing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3393–3402, Florence, Italy. Association for Computational Linguistics.

Micha Elsner, Andrea D Sims, Alexander Erdmann, Antonio Hernandez, Evan Jaffe, Lifeng Jin, Martha Booker Johnson, Shuan Karim, David L King, Luana Lamberti Nunes, et al. 2019. Modeling morphological learning, typology, and change: What can the neural sequence-to-sequence framework contribute? *Journal of Language Modelling*, 7(1):53–98.

Mariano Felice, Christopher Bryant, and Ted Briscoe. 2016. Automatic extraction of learner errors in ESL sentences using linguistically enhanced alignments. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 825–835, Osaka, Japan. The COLING 2016 Organizing Committee.

Tao Ge, Furu Wei, and Ming Zhou. 2018. Reaching human-level performance in automatic grammatical error correction: An empirical study. *arXiv preprint arXiv:1807.01270*.

Mor Geva, Eric Malmi, Idan Szpektor, and Jonathan Berant. 2019. DiscoFuse: A large-scale dataset for discourse-based sentence fusion. 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 3443–3455, Minneapolis, Minnesota. Association for Computational Linguistics.

Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. 2019. Neural grammatical error correction systems with unsupervised pre-training on synthetic data. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 252–263, Florence, Italy. Association for Computational Linguistics.

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

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019b. Levenshtein transformer. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11181–11191. Curran Associates, Inc.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 140–149, Berlin, Germany. Association for Computational Linguistics.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington, USA. Association for Computational Linguistics.

Abdul Khan, Subhadarshi Panda, Jia Xu, and Lampros Flokas. 2018. Hunter NMT system for WMT18 biomedical translation task: Transfer learning in neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 655–661, Belgium, Brussels. Association for Computational Linguistics.

Shun Kiyono, Jun Suzuki, Masato Mita, Tomoya Mizumoto, and Kentaro Inui. 2019. An empirical study of incorporating pseudo data into grammatical error correction. 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 1236–1242, Hong Kong, China. Association for Computational Linguistics.

Jared Lichtarge, Chris Alberti, Shankar Kumar, Noam Shazeer, Niki Parmar, and Simon Tong. 2019. Corpora generation for grammatical error correction. 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 3291–3301, Minneapolis, Minnesota. Association for Computational Linguistics.

Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2015. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*.

Jonathan Mallinson, Aliaksei Severyn, Eric Malmi, and Guillermo Garrido. 2020. Felix: Flexible text editing through tagging and insertion. *arXiv preprint arXiv:2003.10687*.

Eric Malmi, Sebastian Krause, Sascha Rothe, Daniil Mirylenka, and Aliaksei Severyn. 2019. Encode, tag, realize: High-precision text editing. 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 5054–5065, Hong Kong, China. Association for Computational Linguistics.

Courtney Mansfield, Ming Sun, Yuzong Liu, Ankur Gandhe, and Björn Hoffmeister. 2019. Neural text normalization with subword units. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, pages 190–196, Minneapolis - Minnesota. Association for Computational Linguistics.

Tomoya Mizumoto, Yuta Hayashibe, Mamoru Komachi, Masaaki Nagata, and Yuji Matsumoto. 2012. The effect of learner corpus size in grammatical error correction of ESL writings. In *Proceedings of COLING 2012: Posters*, pages 863–872, Mumbai, India. The COLING 2012 Organizing Committee.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

Courtney Napoles, Keisuke Sakaguchi, and Joel Tetreault. 2017. JFLEG: A fluency corpus and benchmark for grammatical error correction. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 229–234, Valencia, Spain. Association for Computational Linguistics.

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. 2014. The CoNLL-2014 shared task on

grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14, Baltimore, Maryland. Association for Computational Linguistics.

Robert Östling and Jörg Tiedemann. 2017. Neural machine translation for low-resource languages. *arXiv preprint arXiv:1708.05729*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Joana Ribeiro, Shashi Narayan, Shay B. Cohen, and Xavier Carreras. 2018. Local string transduction as sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1360–1371, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. 2019. Leveraging pre-trained checkpoints for sequence generation tasks. *arXiv preprint arXiv:1907.12461*.

Danielle Saunders, Felix Stahlberg, and Bill Byrne. 2019. UCAM biomedical translation at WMT19: Transfer learning multi-domain ensembles. In *Proceedings of the Fourth Conference on Machine Translation (Volume 3: Shared Task Papers, Day 2)*, pages 169–174, Florence, Italy. Association for Computational Linguistics.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4596–4604, Stockholmsmssan, Stockholm Sweden. PMLR.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 231–235, Berlin, Germany. Association for Computational Linguistics.

Richard Sproat and Navdeep Jaitly. 2016. RNN approaches to text normalization: A challenge. *arXiv preprint arXiv:1611.00068*.

Felix Stahlberg, Danielle Saunders, and Bill Byrne. 2018. An operation sequence model for explainable neural machine translation. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 175–186, Brussels, Belgium. Association for Computational Linguistics.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5976–5985, Long Beach, California, USA. PMLR.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Jiwei Tan, Xiaojun Wan, and Jianguo Xiao. 2017. Abstractive document summarization with a graph-based attentional neural model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1171–1181, Vancouver, Canada. Association for Computational Linguistics.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. 2018. Tensor2Tensor for neural machine translation. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Papers)*, pages 193–199, Boston, MA. Association for Machine Translation in the Americas.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus

Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. 2016. Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415.

Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. 2011. A new dataset and method for automatically grading ESOL texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 180–189, Portland, Oregon, USA. Association for Computational Linguistics.

Hao Zhang, Richard Sproat, Axel H. Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman, and Brian Roark. 2019. Neural models of text normalization for speech applications. *Computational Linguistics*, 45(2):293–337.

Xingxing Zhang and Mirella Lapata. 2017. Sentence simplification with deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 584–594, Copenhagen, Denmark. Association for Computational Linguistics.

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1566, Berlin, Germany. Association for Computational Linguistics.

Wei Zhao, Liang Wang, Kewei Shen, Ruoyu Jia, and Jingming Liu. 2019. Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data. 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 156–165, Minneapolis, Minnesota. Association for Computational Linguistics.