Paper Reading

Lingyun Feng

2021-07-08

Local Sequence Transduction (LST)

- In local sequence transduction (LST) an input sequence $x_1, \ldots, x_n$ needs to be mapped to an output sequence $y_1, \ldots, y_m$ where the x and y sequences differ only in a few positions, m is close to n, and x_i , y_j come from the same vocabulary Σ.
- Applications:
  - Grammatical error correction (GEC)
  - Sentence fusion/splitting
- The general sequence transduction task is cast as sequence to sequence (seq2seq) learning and modeled popularly using an attentional encoder-decoder (ED) model. The ED model auto-regressively produces each token y_t in the output sequence conditioned on all previous tokens $y_1, \ldots, y_{t-1}$.

- sequence tagging approaches that cast text generation as a text editing task.

- Why?
  - In some text generation tasks, such as the recently introduced sentence splitting and sentence fusion tasks, output texts highly overlap with inputs. In this setting, learning a seq2seq model to generate the output text from scratch seems intuitively wasteful.
  - Copy mechanisms allow for choosing between copying source tokens and generating arbitrary tokens, but although such hybrid models help with out-of-vocabulary words, they still require large training sets as they depend on output vocabularies as large as those used by the standard seq2seq approaches.

- How?

Encode, Tag, Realize: High-Precision Text Editing

Eric Malmi, Sebastian Krause, Sascha Rothe, Daniil Mirylenka, Aliaksei Severyn

Google Research

- LASERTAGGER—a sequence tagging-based model for text editing.

- Two versions of the tagging model:
  - LASERTAGGER$_{AR}$:

  the tagging model with an autoregressive decoder
  - LASERTAGGER$_{FF}$

  the model with feedforward decoder

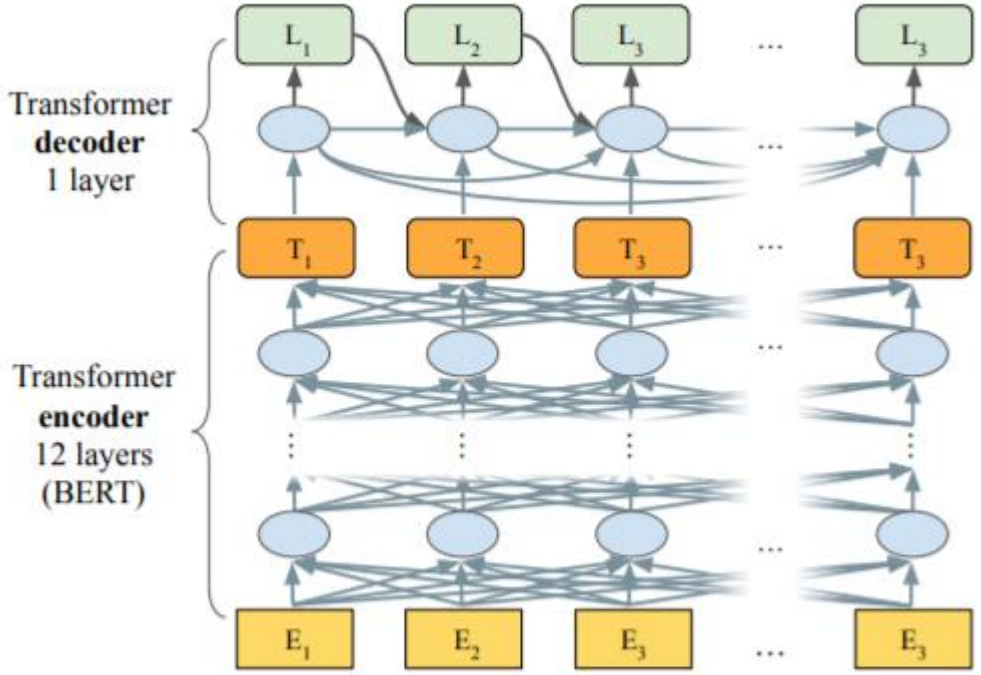


Figure 3: The architecture of LASERTAGGER$_{AR}$.

- Target texts are reconstructed from the inputs using three main edit operations: keeping a token, deleting it, and adding a phrase before the token.

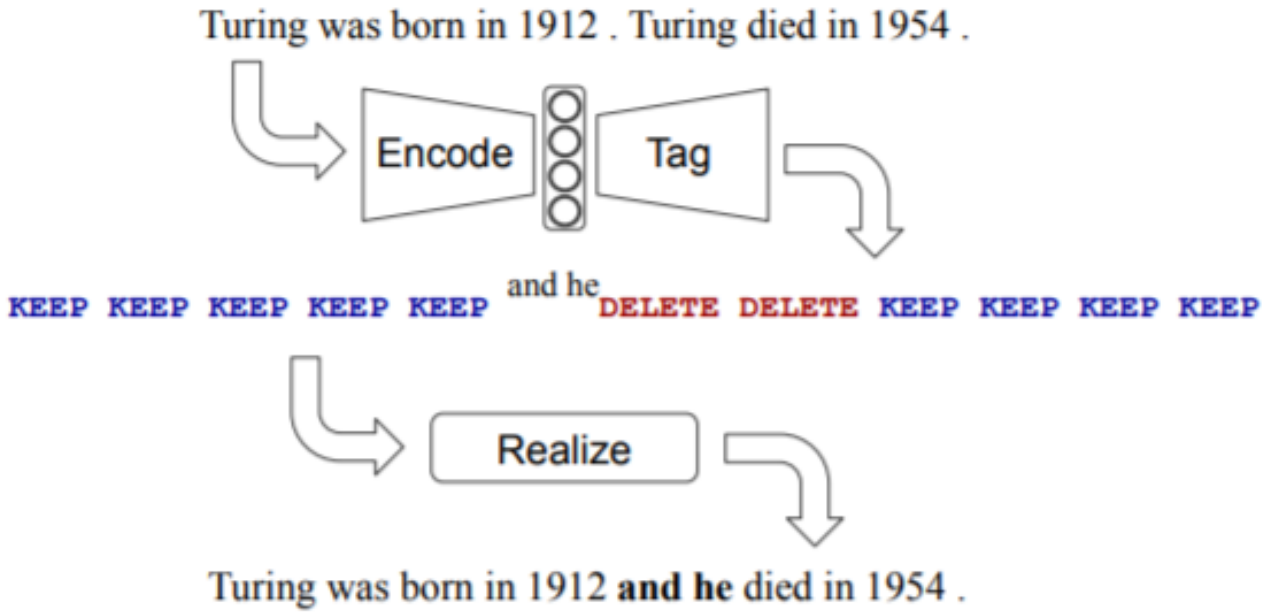


Figure 1: LASERTAGGER applied to sentence fusion.

Left: combining a BERT encoder with an autoregressive transformer decoder to predict the edit operations.
The realization step is to convert tags into the final output text after obtaining a predicted tag sequence.

Experiments

- Sentence Fusion
  - fuses sentences into a single coherent sentence

Model	Exact	SARI
Transformer (Geva et al., 2019)	51.1	84.5
SEQ2SEQ$_{BERT}$	53.6	85.3
LASERTAGGER$_{AR}$ (no SWAP)	46.4	80.4
LASERTAGGER$_{FF}$	52.2	84.1
LASERTAGGER$_{AR}$	**53.8**	**85.5**

Table 2: Sentence fusion results on DfWiki.

- Split and Rephrase
  - requires rewriting a long sentence into two or more coherent short sentences.

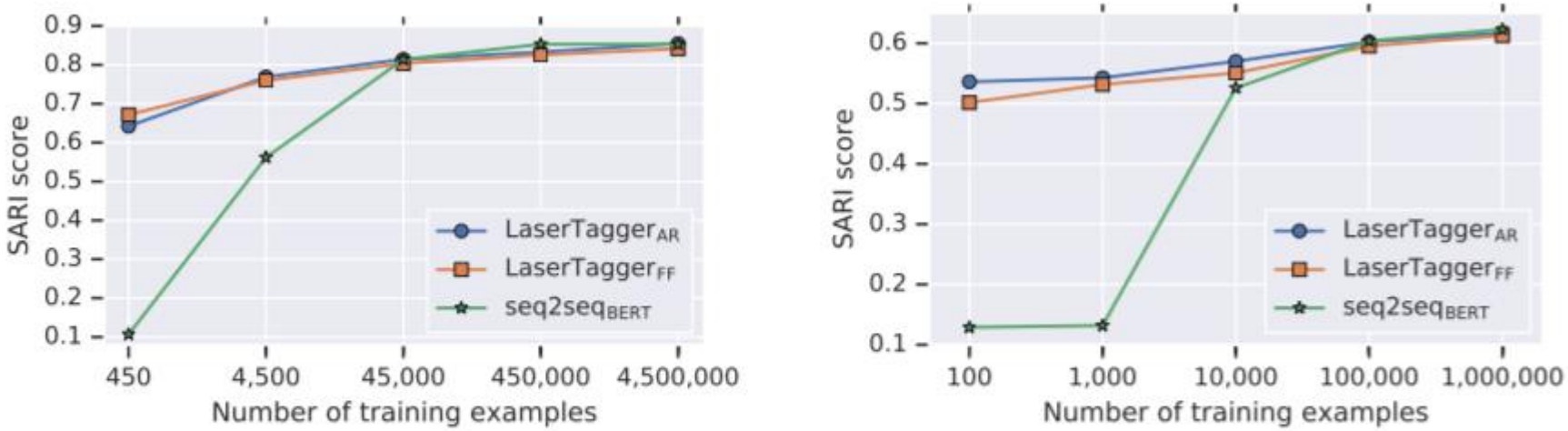
Model	BLEU	Exact	SARI
seq2seq (Botha et al., 2018)	76.0	14.6	60.6
SEQ2SEQ$_{BERT}$	**76.7**	15.1	**62.3**
LASERTAGGER$_{FF}$	76.0	14.4	61.3
LASERTAGGER$_{AR}$	76.3	**15.2**	61.7

Table 3: Results on the WikiSplit dataset.

Exact score: the percentage of exactly correctly predicted fusions
SARI: computes the average F1 scores of the added, kept, and deleted n-grams.

Experiments

- The impact of training-data size



(a) Sentence Fusion on DfWiki.

(b) Split and Rephrase on WikiSplit.

Figure 5: SARI score as a function of the training-data size for three models. Unless we have tens of thousands of training examples, the tagging approach clearly outperforms the seq2seq baseline.

LASERTAGGER methods degrade more gracefully when reducing training-data size, and start to outperform the seq2seq baseline once going below circa 10k examples.

Experiments

- Abstractive Summarization
  - reduce the length of a text while preserving its meaning.

Model	BLEU-4	Exact	SARI	ROUGE-L
Filippova et al. (2015)	26.7	0.0	36.2	70.3
Clarke and Lapata (2008)	28.5	0.3	41.5	77.5
Cohn and Lapata (2008)	5.1	0.1	27.4	40.7
Rush et al. (2015)	16.2	0.0	35.6	62.5
SEQ2SEQ$_{BERT}$	8.3	0.1	32.1	52.7
LASERTAGGER$_{FF}$	33.7	1.5	44.2	81.9
LASERTAGGER$_{AR}$	**35.6**	**3.8**	**44.8**	**82.8**

Table 4: Results on summarization.

- Grammatical Error Correction (GEC)
  - identify and fix grammatical errors in a given input text.

Model	P	R	$F_{0.5}$
Grundkiewicz et al. (2019)	*70.19*	*47.99*	*64.24*
SEQ2SEQ$_{BERT}$	6.13	14.14	6.91
LASERTAGGER$_{FF}$	44.17	24.00	37.82
LASERTAGGER$_{AR}$	**47.46**	**25.58**	**40.52**

Table 5: Results on grammatical-error correction. Note that Grundkiewicz et al. (2019) augment the training dataset of 4,384 examples by 100 million synthetic examples and 2 million Wikipedia edits.

ROUGE-L: a recall-oriented measure computed as the longest common sub-sequence between a reference summary and a candidate summary. $F_{0.5}$ metric: weights precision twice as much as recall.

Experiments

- Inference time

batch size	LaserTagger$_{FF}$	LaserTagger$_{AR}$	Seq2Seq$_{BERT}$
1	13	535	1,773
8	47	668	8,279
32	149	1,273	27,305

Table 6: Inference time (in ms) across various batch sizes on GPU (Nvidia Tesla P100) averaged across 100 runs with random inputs.

LASERTAGGER$_{FF}$ is up to 100x faster at inference time with performance comparable to the state-of-the-art seq2seq models.
Furthermore, both LASERTAGGER$_{FF}$ and LASERTAGGER$_{AR}$ require much less training data compared to the seq2seq models.

Summary

- Advantages:
  - Compared to the seq2seq models, the proposed approach results in a simpler sequence-tagging problem with a much smaller output tag vocabulary.
  - LASERTAGGER has comparable performance when trained on medium-to-large datasets, and clearly outperforms a strong seq2seq baseline when the number of training examples is limited.
  - LASERTAGGER speeds up inference by more than two orders of magnitude, making it more attractive for production applications
  - More controllable and interpretable than seq2seq models due to the small vocabulary of edit operations.
  - Less prone to typical seq2seq model errors, such as hallucination.
- Disadvantages:
  - Arbitrary word reordering is not feasible, although limited reordering can be achieved with deletion and insertion operations.

GECToR – Grammatical Error Correction: Tag, Not Rewrite

Kostiantyn Omelianchuk **Vitaliy Atrasevych*** **Artem Chernodub*** **Oleksandr Skurzhanskyi***
Grammarly

LaserTagger combines a BERT encoder with an autoregressive Transformer decoder to predict three main edit operations: keeping a token, deleting a token, and adding a phrase before a token.

In contrast, in GECToR:

- The sequence tagging model is an encoder made up of pretrained BERT-like transformer, the decoder is a softmax layer stacked with two linear layers with softmax layers on the top.
- Token-level transformations
  - **Basic transformations** perform the most common token-level edit operations, such as: keep the current token unchanged (tag $KEEP), delete current token (tag $DELETE), append new token t1 next to the current token xi (tag $APPEND t1) or replace the current token xi with another token t2 (tag $REPLACE t2).
  - **g-transformations** perform task-specific operations such as: change the case of the current token (CASE tags), merge the current token and the next token into a single one (MERGE tags) and split the current token into two new tokens (SPLIT tags). Moreover, tags from NOUN NUMBER and VERB FORM transformations encode grammatical properties for tokens. Predicting g-transformations instead of regular tokens improves the generalization of GEC sequence tagging system.

Experiments

GEC system	Ens.	CoNLL-2014 (test)			BEA-2019 (test)		
		P	R	$F_{0.5}$	P	R	$F_{0.5}$
Zhao et al. (2019)		67.7	40.6	59.8	-	-	-
Awasthi et al. (2019)		66.1	43.0	59.7	-	-	-
Kiyono et al. (2019)		67.9	**44.1**	61.3	65.5	**59.4**	64.2
Zhao et al. (2019)	✓	74.1	36.3	61.3	-	-	-
Awasthi et al. (2019)	✓	68.3	43.2	61.2	-	-	-
Kiyono et al. (2019)	✓	72.4	**46.1**	65.0	74.7	56.7	70.2
Kantor et al. (2019)	✓	-	-	-	78.3	58.0	73.2
GECToR (BERT)		72.1	42.0	63.0	71.5	55.7	67.6
GECToR (RoBERTa)		73.9	41.5	64.0	77.2	55.1	71.5
GECToR (XLNet)		**77.5**	40.1	**65.3**	**79.2**	53.9	**72.4**
GECToR (RoBERTa + XLNet)	✓	76.6	42.3	66.0	**79.4**	57.2	**73.7**
GECToR (BERT + RoBERTa + XLNet)	✓	**78.2**	41.5	**66.5**	78.9	**58.2**	73.6

Table 7: Comparison of single models and ensembles. The M^2 score for CoNLL-2014 (test) and ERRANT for the BEA-2019 (test) are reported. In ensembles we simply average output probabilities from single models.

- Achieve superior performance by incorporating a pre-trained Transformer encoder in the GEC sequence tagging system. Encoders from XLNet and RoBERTa outperform three other cutting-edge Transformer encoders (ALBERT, BERT, and GPT-2)

Experiments

- Iterative sequence tagging approach

Iteration #	Sentence's evolution	# corr.
Orig. sent	A ten years old boy go school	-
Iteration 1	A ten-years old boy **goes** school	2
Iteration 2	A ten-**year**-old boy goes **to** school	5
Iteration 3	A ten-year-old boy goes to school**.**	6

Table 3: Example of iterative correction process where GEC tagging system is sequentially applied at each iteration. Cumulative number of corrections is given for each iteration. Corrections are in bold.

- Some corrections in a sentence may depend on others, applying GEC sequence tagger only once may not be enough to fully correct the sentence.
- Use the GEC sequence tagger to tag the now modified sequence, and apply the corresponding transformations on the new tags, which changes the sentence further. Usually, the number of corrections decreases with each successive iteration, and most of the corrections are done during the first two iterations.

Seq2Edits: Sequence Transduction Using Span-level Edit Operations

Felix Stahlberg and Shankar Kumar
Google Research

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.

Motivations :

- 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.

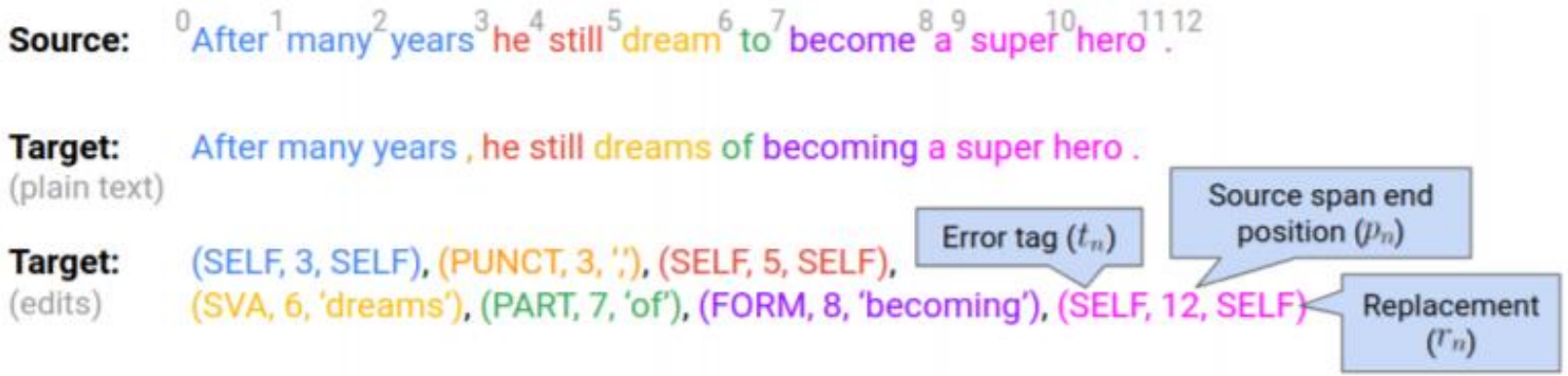


Figure 1: Representing grammatical error correction as a sequence of span-based edit operations. The implicit start position for a source span is the end position of the previous edit operation. SELF indicates spans that are copied over from the source sentence ($\mathbf{x}$). The probability of the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

- Rather than generating the target sentence as a series of tokens, the 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 architecture to facilitate learning of long-range dependencies.
- Much faster than a full sequence model because its runtime depends on the number of edits rather than the target sentence length.
- More explainable by associating each edit operation with a human-readable tag.

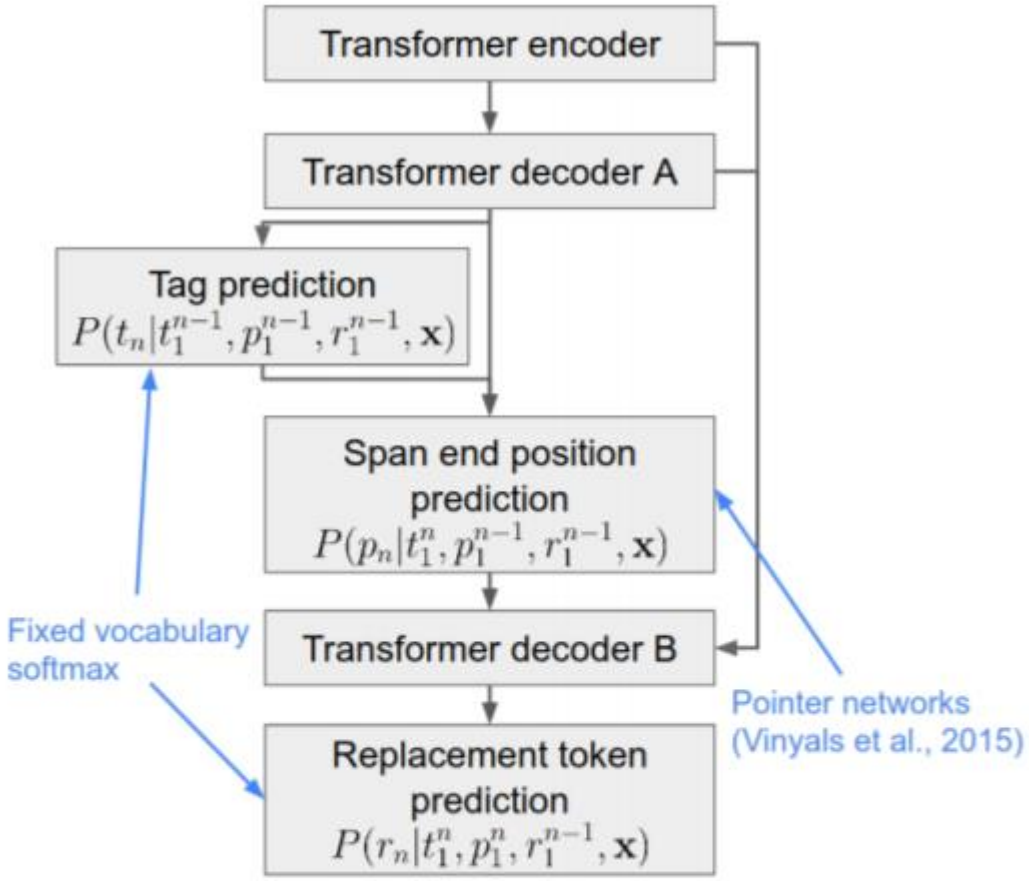


Figure 2: *Seq2Edits* consists of a Transformer (Vaswani et al., 2017) encoder and a Transformer decoder that is divided horizontally into two parts (A and B). The tag and span predictions are located in the middle of the decoder layer stack between both parts. A single step of prediction is shown.

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:

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

For inference, they factorize the conditional probabilities further as:

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

Experiments

- Text normalization for speech application
  - Converting number expressions such as “123” to their verbalizations (e.g. “one two three” or “one hundred twenty three”, etc.) depending on the context.

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.

- Sentence fusion
  - Merging two independent sentences to a single coherent one.

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.

on par with the FELIX tagger on the DiscoFuse dataset but worse than the BERT2BERT
BERT2BERT’s strategy of making use of target-side pre-training under a language model objective is particularly useful for sentence fusion.

Experiments

- Sentence splitting & rephrasing
  - Splitting a long sentence into two fluent sentences.
- Text simplification
  - Reducing the linguistic complexity of text.

System	Exact↑	SARI↑
Botha et al. (2018)	14.6	60.6
Malmi et al. (2019)	15.2	61.7
Malmi et al. (2019) - SEQ2SEQ$_{BERT}$	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]

The 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.

Experiments

- Grammatical error correction.
  - Correct grammatical errors in written text.

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.

- The model tends to have a lower precision but higher recall than other systems.
- Achieve the highest GLEU score on the JFLEG test set.

- Grammatical error correction examples from BEA-dev

Source	$_0$ It $_1$ will $_2$ be $_3$ very $_4$ cool $_5$ to $_6$ see $_7$ the $_8$ **las** $_9$ **part** $_{10}$ **mokingjay** $_{11}$! $_{12}$
Reference	It will be very cool to see the **last part of Mokingjay** !
Full seq.	It will be very cool to see the **last mokingjay** !
Edit model	It will be very cool to see the **last** (SPELL) **part** (SELF) **of** (PREP) **mokingjay** (SELF) ! (SELF)
Edits	(SELF, 8, SELF), (SPELL, 9, 'last'), (SELF, 10, SELF), (PART, 10, 'of'), (SELF, 12, SELF)
Source	$_0$ If $_1$ she $_2$ was $_3$ n't $_4$ awake $_5$, $_6$ why $_7$ **she** $_8$ **could** $_9$ **n't** $_{10}$ remember $_{11}$ anything $_{12}$ after $_{13}$ that $_{14}$? $_{15}$
Reference	If she was n't awake , why **could n't** she remember anything after that ?
Full seq.	If she was n't awake , why **she could n't** remember anything after that ?
Edit model	If she was n't awake , why (SELF) **could n't she** (WO) remember anything after that ? (SELF)
Edits	(SELF, 7, SELF), (WO, 10, 'could n't she'), (SELF, 15, SELF)
Source	$_0$ **Less** $_1$ channels $_2$ means $_3$ **less** $_4$ choices $_5$. $_6$
Reference	**Fewer** channels means **fewer** choices .
Full seq.	**Less** channels means **fewer** choices .
Edit model	**Fewer** (ADJ) channels means (SELF) **fewer** (ADJ) choices . (SELF)
Edits	(ADJ, 1, 'Fewer'), (SELF, 3, SELF), (ADJ, 4, 'fewer'), (SELF, 6, SELF)
Source	$_0$ On $_1$ the $_2$ one $_3$ hand $_4$ travel $_5$ by $_6$ car $_7$ are $_8$ really $_9$ much $_{10}$ more $_{11}$ convenient $_{12}$ as $_{13}$ **give** $_{14}$ the $_{15}$ chance $_{16}$ **to** $_{17}$ **you** $_{18}$ to $_{19}$ be $_{20}$ independent $_{21}$. $_{22}$
Reference	On the one hand , travel by car **is** really much more convenient , as **it gives you** the chance to be independent .
Full seq.	On the one hand , travel by car **is** really much more convenient , as **it gives you** the chance to be independent .
Edit model	On the one hand (SELF) , (PUNCT) travel by car (SELF) **is** (VERB:SVA) really much more convenient (SELF) , (PUNCT) **as give** (SELF) **you** (PRON) the chance (SELF) to be independent . (SELF)
Edits	(SELF, 4, SELF), (PUNCT, 4, ','), (SELF, 7, SELF), (VERB:SVA, 8, 'is'), (SELF, 12, SELF), (PUNCT, 12, ','), (SELF, 14, SELF), (PRON, 14, 'you'), (SELF, 16, SELF), (PRON, 18, DEL), (SELF, 22, SELF)

The second example shows that Seq2Edits is able to handle more complex operations such as word reorderings. However, the model fails to inflect "give" correctly in the last example, suggesting that one weakness of the edit model compared to a full sequence model is a weaker target side language model resulting in less fluent output.

Summary

Advantages:

- 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.
- 2.0-5.2 times faster than a full sequence model for GEC.
- can predict labels that explain each edit to improve the interpretability for the end-user.

Disadvantages:

- The model uses a tailored architecture that would require some engineering effort to implement efficiently.
- The output of the model tends to be less fluent than a regular full sequence model. 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).

Thanks