表示我们这个训练任务是tag,不是做一个文本生成任务.
显然这么做会好很多.因为文本生成的字典要大.而tag字典只
有5000. tag的上下文信息也更适合.

GECToR – Grammatical Error Correction: Tag, Not Rewrite

Kostiantyn Omelianchuk **Vitaliy Atrasevych*** **Artem Chernodub*** **Oleksandr Skurzhanskyi***
Grammarly
{firstname.lastname}@grammarly.com

Abstract

In this paper, we present a simple and efficient GEC sequence tagger using a Transformer encoder. Our system is pre-trained on synthetic data and then fine-tuned in two stages: first on errorful corpora, and second on a combination of errorful and error-free parallel corpora. We design custom token-level transformations to map input tokens to target corrections. Our best single-model/ensemble GEC tagger achieves an $F_{0.5}$ of 65.3/66.5 on CoNLL-2014 (test) and $F_{0.5}$ of 72.4/73.6 on BEA-2019 (test). Its inference speed is up to 10 times as fast as a Transformer-based seq2seq GEC system. The code and trained models are publicly available[1].

1 Introduction

Neural Machine Translation (NMT)-based approaches (Sennrich et al., 2016a) have become the preferred method for the task of Grammatical Error Correction (GEC)[2]. In this formulation, errorful sentences correspond to the source language, and error-free sentences correspond to the target language. Recently, Transformer-based (Vaswani et al., 2017) sequence-to-sequence (seq2seq) models have achieved state-of-the-art performance on standard GEC benchmarks (Bryant et al., 2019). Now the focus of research has shifted more towards generating synthetic data for pretraining the Transformer-NMT-based GEC systems (Grundkiewicz et al., 2019; Kiyono et al., 2019). NMT-based GEC systems suffer from several issues which make them inconvenient for real world deployment: (i) slow inference speed, (ii) demand for large amounts of training data and (iii) interpretability and explainability; they require additional functionality to explain corrections, e.g., grammatical error type classification (Bryant et al., 2017).

In this paper, we deal with the aforementioned issues by simplifying the task from sequence generation to sequence tagging. Our GEC sequence tagging system consists of three training stages: pretraining on synthetic data, fine-tuning on an errorful parallel corpus, and finally, fine-tuning on a combination of errorful and error-free parallel corpora.

Related work. LaserTagger (Malmi et al., 2019) 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 our system, the decoder is a softmax layer. PIE (Awasthi et al., 2019) is an iterative sequence tagging GEC system that predicts token-level edit operations. While their approach is the most similar to ours, our work differs from theirs as described in our contributions below:

1. We develop custom g-transformations: token-level edits to perform (g)rammatical error corrections. Predicting g-transformations instead of regular tokens improves the generalization of our GEC sequence tagging system.

2. We decompose the fine-tuning stage into two stages: fine-tuning on errorful-only sentences and further fine-tuning on a small, high-quality dataset containing both errorful and error-free sentences.

3. We achieve superior performance by incorporating a pre-trained Transformer encoder in our GEC sequence tagging system. In our experiments, encoders from XLNet and RoBERTa outperform three other cutting-edge Transformer encoders (ALBERT, BERT, and GPT-2).

*Authors contributed equally to this work, names are given in an alphabetical order.

[1] https://github.com/grammarly/gector

[2] http://nlpprogress.com/english/grammatical_error_correction.html (Accessed 1 April 2020).

Dataset	# sentences	% errorful sentences	Training stage
PIE-synthetic	9,000,000	100.0%	I
Lang-8	947,344	52.5%	II
NUCLE	56,958	38.0%	II
FCE	34,490	62.4%	II
W&I+LOCNESS	34,304	67.3%	II, III

Table 1: Training datasets. Training stage I is pretraining on synthetic data. Training stages II and III are for fine-tuning.

2 Datasets

Table 1 describes the finer details of datasets used for different training stages.

Synthetic data. For pretraining stage I, we use 9M parallel sentences with synthetically generated grammatical errors (Awasthi et al., 2019)[3].

Training data. We use the following datasets for fine-tuning stages II and III: National University of Singapore Corpus of Learner English (NUCLE)[4] (Dahlmeier et al., 2013), Lang-8 Corpus of Learner English (Lang-8)[5] (Tajiri et al., 2012), FCE dataset[6] (Yannakoudakis et al., 2011), the publicly available part of the Cambridge Learner Corpus (Nicholls, 2003) and Write & Improve + LOCNESS Corpus (Bryant et al., 2019)[7].

Evaluation data. We report results on CoNLL-2014 test set (Ng et al., 2014) evaluated by official M^2 scorer (Dahlmeier and Ng, 2012), and on BEA-2019 dev and test sets evaluated by ERRANT (Bryant et al., 2017).

3 Token-level transformations

We developed custom token-level transformations $T(x_i)$ to recover the target text by applying them to the source tokens $(x_1 \dots x_N)$. Transformations increase the coverage of grammatical error corrections for limited output vocabulary size for the most common grammatical errors, such as *Spelling*, *Noun Number*, *Subject-Verb Agreement* and *Verb Form* (Yuan, 2017, p. 28).

The edit space which corresponds to our default tag vocabulary size = 5000 consists of 4971

[3]https://github.com/awasthiabhijeet/PIE/tree/master/errorify

[4]https://www.comp.nus.edu.sg/~nlp/corpora.html

[5]https://sites.google.com/site/naistlang8corpora

[6]https://ilexir.co.uk/datasets/index.html

[7]https://www.cl.cam.ac.uk/research/nl/bea2019st/data/wi+locness_v2.1.bea19.tar.gz

tag 的类别正好是5千

basic transformations (token-independent KEEP, DELETE and 1167 token-dependent APPEND, 3802 REPLACE) and 29 token-independent *g-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 t_1 next to the current token x_i (tag *$APPEND_*$t_1$) or replace the current token x_i with another token t_2 (tag *$REPLACE_*$t_2$).

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. For instance, these transformations include conversion of singular nouns to plurals and vice versa or even change the form of regular/irregular verbs to express a different number or tense.

复数名词

To obtain the transformation suffix for the *VERB_FORM* tag, we use the verb conjugation dictionary[8]. For convenience, it was converted into the following format: $token_0\_token_1 : tag_0\_tag_1$ (e.g., *go_goes* : VB_VBZ). This means that there is a transition from $word_0$ and $word_1$ to the respective tags. The transition is unidirectional, so if there exists a reverse transition, it is presented separately.

The experimental comparison of covering capabilities for our token-level transformations is in Table 2. All transformation types with examples are listed in Appendix, Table 9.

Preprocessing. To approach the task as a sequence tagging problem we need to convert each target sentence from training/evaluation sets into a sequence of tags where each tag is mapped to a single source token. Below is a brief description of our 3-step preprocessing algorithm for color-coded sentence pair from Table 3:

Step 1). Map each token from source sentence to subsequence of tokens from target sentence. [A $\mapsto$ A], [ten $\mapsto$ ten, -], [years $\mapsto$ year, -], [old $\mapsto$ old], [go $\mapsto$ goes, to], [school $\mapsto$ school, .].

[8]https://github.com/gutfeeling/word_forms/blob/master/word_forms/en-verbs.txt

每一个[]里面表示2个token

Tag vocab. size	Transformations	
	Basic transf.	All transf.
100	60.4%	79.7%
1000	76.4%	92.9%
5000	89.5%	98.1%
10000	93.5%	100.0%

Table 2: Share of covered grammatical errors in CoNLL-2014 for basic transformations only (KEEP, DELETE, APPEND, REPLACE) and for all transformations w.r.t. tag vocabulary's size. In our work, we set the default tag vocabulary size = 5000 as a heuristical compromise between coverage and model size.

为了得到上面例子这个结果.,

For this purpose, we first detect the minimal spans of tokens which define differences between source tokens $(x_1 \ldots x_N)$ and target tokens $(y_1 \ldots y_M)$. Thus, such a span is a pair of selected source tokens and corresponding target tokens. We can't use these span-based alignments, because we need to get tags on the token level. So then, for each source token x_i, $1 \le i \le N$ we search for best-fitting subsequence $\Upsilon_i = (y_{j_1} \ldots y_{j_2})$, $1 \le j_1 \le j_2 \le M$ of target tokens by minimizing the modified Levenshtein distance (which takes into account that successful g-transformation is equal to zero distance).

Step 2). For each mapping in the list, find token-level transformations which convert source token to the target subsequence: [A $\mapsto$ A]: \$KEEP, [ten $\mapsto$ ten, -]: \$KEEP, \$MERGE_HYPHEN, [years $\mapsto$ year, -]: \$NOUN_NUMBER_SINGULAR, \$MERGE_HYPHEN], [old $\mapsto$ old]: \$KEEP, [go $\mapsto$ goes, to]: \$VERB_FORM_VB_VBZ, \$APPEND_to, [school $\mapsto$ school, .]: \$KEEP, \$APPEND_{.}].

Step 3). Leave only one transformation for each source token: A $\Leftrightarrow$ \$KEEP, ten $\Leftrightarrow$ \$MERGE_HYPHEN, years $\Leftrightarrow$ \$NOUN_NUMBER_SINGULAR, old $\Leftrightarrow$ \$KEEP, go $\Leftrightarrow$ \$VERB_FORM_VB_VBZ, school $\Leftrightarrow$ \$APPEND_{.}.

每迭代一次
对每一个位置
打一个tag

The iterative sequence tagging approach adds a constraint because we can use only a single tag for each token. In case of multiple transformations we take the first transformation that is not a \$KEEP tag. For more details, please, see the preprocessing script in our repository[9].

4 Tagging model architecture

Our GEC sequence tagging model is an encoder made up of pretrained BERT-like transformer

[9] https://github.com/grammarly/gector

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.

这个纠错问题,我们使用带区分大小写的bert模型.

stacked with two linear layers with softmax layers on the top. We always use cased pretrained transformers in their Base configurations. Tokenization depends on the particular transformer's design: BPE (Sennrich et al., 2016b) is used in RoBERTa, WordPiece (Schuster and Nakajima, 2012) in BERT and SentencePiece (Kudo and Richardson, 2018) in XLNet. To process the information at the token-level, we take the first subword per token from the encoders representation, which is then forwarded to subsequent linear layers, which are responsible for error detection and error tagging, respectively.

5 Iterative sequence tagging approach

To correct the text, for each input token x_i, $1 \le i \le N$ from the source sequence $(x_1 \ldots x_N)$, we predict the tag-encoded token-level transformation $T(x_i)$ described in Section 3. These predicted tag-encoded transformations are then applied to the sentence to get the modified sentence.

Since some corrections in a sentence may depend on others, applying GEC sequence tagger only once may not be enough to fully correct the sentence. Therefore, we use the iterative correction approach from (Awasthi et al., 2019): we 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 (see an example in Table 3). Usually, the number of corrections decreases with each successive iteration, and most of the corrections are done during the first two iterations (Table 4). Limiting the number of iterations speeds up the overall pipeline while trading off qualitative performance.

Iteration #	P	R	$F_{0.5}$	# corr.
Iteration 1	72.3	38.6	61.5	787
Iteration 2	73.7	41.1	63.6	934
Iteration 3	74.0	41.5	64.0	956
Iteration 4	73.9	41.5	64.0	958

Table 4: Cumulative number of corrections and corresponding scores on CoNLL-2014 (test) w.r.t. number of iterations for our best single model.

Training stage #	CoNLL-2014 (test)			BEA-2019 (dev)		
	P	R	$F_{0.5}$	P	R	$F_{0.5}$
Stage I.	55.4	35.9	49.9	37.0	23.6	33.2
Stage II.	64.4	46.3	59.7	46.4	37.9	44.4
Stage III.	66.7	**49.9**	62.5	52.6	**43.0**	50.3
Inf. tweaks	**77.5**	40.2	**65.3**	**66.0**	33.8	**55.5**

Table 5: Performance of GECToR (XLNet) after each training stage and inference tweaks.

6 Experiments

Training stages. We have 3 training stages (details of data usage are in Table 1):

I Pre-training on synthetic errorful sentences as in (Awasthi et al., 2019).

II Fine-tuning on errorful-only sentences.

III Fine-tuning on subset of errorful and error-free sentences as in (Kiyono et al., 2019).

We found that having two fine-tuning stages with and without error-free sentences is crucial for performance (Table 5).

All our models were trained by Adam optimizer (Kingma and Ba, 2015) with default hyperparameters. Early stopping was used; stopping criteria was 3 epochs of 10K updates each without improvement. We set batch size=256 for pre-training stage I (20 epochs) and batch size=128 for fine-tuning stages II and III (2-3 epochs each). We also observed that freezing the encoder's weights for the first 2 epochs on training stages I-II and using a batch size greater than 64 improves the convergence and leads to better GEC performance.

Encoders from pretrained transformers. We fine-tuned BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), GPT-2 (Radford et al., 2019), XLNet (Yang et al., 2019), and ALBERT (Lan et al., 2019) with the same hyperparameters setup. We also added LSTM with randomly initialized embeddings ($dim = 300$) as a baseline. As follows from Table 6, encoders from fine-tuned Transformers significantly outperform LSTMs. BERT, RoBERTa and XLNet encoders perform better than GPT-2 and ALBERT, so we used them only in our next experiments. All models were trained out-of-the-box[10] which seems to not work well for GPT-2. We hypothesize that encoders from Transformers which were pretrained as a part of the entire encoder-decoder pipeline are less useful for GECToR.

Encoder	CoNLL-2014 (test)			BEA-2019 (dev)		
	P	R	$F_{0.5}$	P	R	$F_{0.5}$
LSTM	51.6	15.3	35.0	-	-	-
ALBERT	59.5	31.0	50.3	43.8	22.3	36.7
BERT	65.6	36.9	56.8	48.3	29.0	42.6
GPT-2	61.0	6.3	22.2	44.5	5.0	17.2
RoBERTa	**67.5**	38.3	**58.6**	**50.3**	30.5	**44.5**
XLNet	64.6	**42.6**	58.5	47.1	**34.2**	43.8

Table 6: Varying encoders from pretrained Transformers in our sequence labeling system. Training was done on data from training stage II only.

Tweaking the inference. We forced the model to perform more precise corrections by introducing two inference hyperparameters (see Appendix, Table 11), hyperparameter values were found by random search on BEA-dev.

First, we added a permanent positive *confidence bias* to the probability of $KEEP tag which is responsible for not changing the source token. Second, we added a sentence-level *minimum error probability* threshold for the output of the error detection layer. This increased precision by trading off recall and achieved better $F_{0.5}$ scores (Table 5).

Finally, our best single-model, GECToR (XLNet) achieves $F_{0.5}$ = 65.3 on CoNLL-2014 (test) and $F_{0.5}$ = 72.4 on BEA-2019 (test). Best ensemble model, GECToR (BERT + RoBERTa + XLNet) where we simply average output probabilities from 3 single models achieves $F_{0.5}$ = 66.5 on CoNLL-2014 (test) and $F_{0.5}$ = 73.6 on BEA-2019 (test), correspondingly (Table 7).

Speed comparison. We measured the models average inference time on NVIDIA Tesla V100 on batch size 128. For sequence tagging we don't need to predict corrections one-by-one as in autoregressive transformer decoders, so inference is naturally parallelizable and therefore runs many times faster. Our sequence tagger's inference speed is up to 10 times as fast as the state-of-the-art Transformer from Zhao et al. (2019), beam size=12 (Table 8).

[10] https://huggingface.co/transformers/

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.

GEC system	Time (sec)
Transformer-NMT, beam size = 12	4.35
Transformer-NMT, beam size = 4	1.25
Transformer-NMT, beam size = 1	0.71
GECToR (XLNet), 5 iterations	0.40
GECToR (XLNet), 1 iteration	0.20

Table 8: Inference time for NVIDIA Tesla V100 on CoNLL-2014 (test), single model, batch size=128.

7 Conclusions

We show that a faster, simpler, and more efficient GEC system can be developed using a sequence tagging approach, an encoder from a pretrained Transformer, custom transformations and 3-stage training.

Our best single-model/ensemble GEC tagger achieves an $F_{0.5}$ of 65.3/66.5 on CoNLL-2014 (test) and $F_{0.5}$ of 72.4/73.6 on BEA-2019 (test). We achieve state-of-the-art results for the GEC task with an inference speed up to 10 times as fast as Transformer-based seq2seq systems.

8 Acknowledgements

This research was supported by Grammarly. We thank our colleagues Vipul Raheja, Oleksiy Syvokon, Andrey Gryshchuk and our ex-colleague Maria Nadejde who provided insight and expertise that greatly helped to make this paper better. We would also like to show our gratitude to Abhijeet Awasthi and Roman Grundkiewicz for their support in providing data and answering related questions. We also thank 3 anonymous reviewers for their contribution.

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.

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.

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. Association for Computational Linguistics.

Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. 2013. Building a large annotated corpus of learner english: The nus corpus of learner english. In *Proceedings of the eighth workshop on innovative use of NLP for building educational applications*, pages 22–31.

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.

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.

Yoav Kantor, Yoav Katz, Leshem Choshen, Edo Cohen-Karlik, Naftali Liberman, Assaf Toledo, Amir Menczel, and Noam Slonim. 2019. Learning to combine grammatical error corrections. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 139–148, Florence, Italy. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2015. Adam (2014), a method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR), arXiv preprint arXiv*, volume 1412.

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.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

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.

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.

Diane Nicholls. 2003. The cambridge learner corpus: Error coding and analysis for lexicography and elt. In *Proceedings of the Corpus Linguistics 2003 conference*, volume 16, pages 572–581.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. IEEE.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Edinburgh neural machine translation systems for WMT 16. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 371–376, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. 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.

Toshikazu Tajiri, Mamoru Komachi, and Yuji Matsumoto. 2012. Tense and aspect error correction for esl learners using global context. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 198–202. Association for Computational Linguistics.

Ashish Vaswani, 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.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764.

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-Volume 1*, pages 180–189. Association for Computational Linguistics.

Zheng Yuan. 2017. Grammatical error correction in non-native english. Technical report, University of Cambridge, Computer Laboratory.

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.

A Appendix

id	Core transformation	Transformation suffix	Tag	Example
basic-1	KEEP	∅	$KEEP	. . . many people want to travel during the summer . . .
basic-2	DELETE	∅	$DELETE	. . . not sure if you are {**you** ⇒ ∅} gifting . . .
basic-3	REPLACE	a	$REPLACE_a	. . . the bride wears {**the** ⇒ **a**} white dress . . .
. . .	. . .	. . .	. . .	. . .
basic-3804	REPLACE	cause	$REPLACE_cause	. . . hope it does not {**make** ⇒ **cause**} any trouble . . .
basic-3805	APPEND	for	$APPEND_for	. . . he is {**waiting** ⇒ **waiting for**} your reply . . .
. . .	. . .	. . .	. . .	. . .
basic-4971	APPEND	know	$APPEND_know	. . . I {**don't** ⇒ **don't know**} which to choose. . .
g-1	CASE	CAPITAL	$CASE_CAPITAL	. . . surveillance is on the {**internet** ⇒ **Internet**} . . .
g-2	CASE	CAPITAL_1	$CASE_CAPITAL_1	. . . I want to buy an {**iphone** ⇒ **iPhone**} . . .
g-3	CASE	LOWER	$CASE_LOWER	. . . advancement in {**Medical** ⇒ **medical**} technology . . .
g-4	CASE	UPPER	$CASE_UPPER	. . . the {**it** ⇒ **IT**} department is concerned that. . .
g-5	MERGE	SPACE	$MERGE_SPACE	. . . insert a special kind of gene {**in to** ⇒ **into**} the cell . . .
g-6	MERGE	HYPHEN	$MERGE_HYPHEN	. . . and needs {**in depth** ⇒ **in-depth**} search . . .
g-7	SPLIT	HYPHEN	$SPLIT_HYPHEN	. . . support us for a {**long-run** ⇒ **long run**} . . .
g-8	NOUN_NUMBER	SINGULAR	$NOUN_NUMBER_SINGULAR	. . . a place to live for their {**citizen** ⇒ **citizens**}
g-9	NOUN_NUMBER	PLURAL	$NOUN_NUMBER_PLURAL	. . . carrier of this {**diseases** ⇒ **disease**} . . .
g-10	VERB FORM	VB_VBZ	$VERB_FORM_VB_VBZ	. . . going through this {**make** ⇒ **makes**} me feel . . .
g-11	VERB FORM	VB_VBN	$VERB_FORM_VB_VBN	. . . to discuss what {**happen** ⇒ **happened**} in fall . . .
g-12	VERB FORM	VB_VBD	$VERB_FORM_VB_VBD	. . . she sighed and {**draw** ⇒ **drew**} her . . .
g-13	VERB FORM	VB_VBG	$VERB_FORM_VB_VBG	. . . shown success in {**prevent** ⇒ **preventing**} such . . .
g-14	VERB FORM	VB_VBZ	$VERB_FORM_VB_VBZ	. . . a small percentage of people {**goes** ⇒ **go**} by bike . . .
g-15	VERB FORM	VBZ_VBN	$VERB_FORM_VBZ_VBN	. . . development has {**pushes** ⇒ **pushed**} countries to . . .
g-16	VERB FORM	VBZ_VBD	$VERB_FORM_VBZ_VBD	. . . he {**drinks** ⇒ **drank**} a lot of beer last night . . .
g-17	VERB FORM	VBZ_VBG	$VERB_FORM_VBZ_VBG	. . . couldn't stop {**thinks** ⇒ **thinking**} about it . . .
g-18	VERB FORM	VBN_VB	$VERB_FORM_VBN_VB	. . . going to {**depended** ⇒ **depend**} on who is hiring . . .
g-19	VERB FORM	VBN_VBZ	$VERB_FORM_VBN_VBZ	. . . yet he goes and {**eaten** ⇒ **eats**} more melons . . .
g-20	VERB FORM	VBN_VBD	$VERB_FORM_VBN_VBD	. . . he {**driven** ⇒ **drove**} to the bus stop and . . .
g-21	VERB FORM	VBN_VBG	$VERB_FORM_VBN_VBG	. . . don't want you fainting and {**broken** ⇒ **breaking**} . . .
g-22	VERB FORM	VBD_VB	$VERB_FORM_VBD_VB	. . . each of these items will {**fell** ⇒ **fall**} in price . . .
g-23	VERB FORM	VBD_VBZ	$VERB_FORM_VBD_VBZ	. . . the lake {**froze** ⇒ **freezes**} every year . . .
g-24	VERB FORM	VBD_VBN	$VERB_FORM_VBD_VBN	. . . he has been went {**went** ⇒ **gone**} since last week . . .
g-25	VERB FORM	VBD_VBG	$VERB_FORM_VBD_VBG	. . . talked her into {**gave** ⇒ **giving**} me the whole day . . .
g-26	VERB FORM	VBG_VB	$VERB_FORM_VBG_VB	. . . free time, I just {**enjoying** ⇒ **enjoy**} being outdoors . . .
g-27	VERB FORM	VBG_VBZ	$VERB_FORM_VBG_VBZ	. . . there still {**existing** ⇒ **exists**} many inevitable factors . . .
g-28	VERB FORM	VBG_VBN	$VERB_FORM_VBG_VBN	. . . people are afraid of being {**tracking** ⇒ **tracked**} . . .
g-29	VERB FORM	VBG_VBD	$VERB_FORM_VBG_VBD	. . . there was no {**mistook** ⇒ **mistaking**} his sincerity . . .

Table 9: List of token-level transformations (section 3). We denote a tag which defines a token-level transformation as concatenation of two parts: a *core transformation* and a *transformation suffix*.

Training stage #	CoNLL-2014 (test)			BEA-2019 (dev)		
	P	**R**	$\mathbf{F_{0.5}}$	**P**	**R**	$\mathbf{F_{0.5}}$
Stage I.	57.8	33.0	50.2	40.8	22.1	34.9
Stage II.	68.1	42.6	60.8	51.6	33.8	46.7
Stage III.	68.8	**47.1**	63.0	54.2	**41.0**	50.9
Inf. tweaks	**73.9**	41.5	**64.0**	**62.3**	35.1	**54.0**

Table 10: Performance of GECToR (RoBERTa) after each training stage and inference tweaks. Results are given in addition to results for our best single model, GECToR (XLNet) which are given in Table 5.

System name	Confidence bias	Minimum error probability
GECToR (BERT)	0.10	0.41
GECToR (RoBERTa)	0.20	0.50
GECToR (XLNet)	0.35	0.66
GECToR (RoBERTa + XLNet)	0.24	0.45
GECToR (BERT + RoBERTa + XLNet)	0.16	0.40

Table 11: Inference tweaking values which were found by random search on BEA-dev.