Grammatical Error Correction at the Character Level

Adam Viola **Rushikesh Dudhat** **Pranjali Parse** **Shubham Shetty**
{aviola, rdudhat, pparse, shubhamshett}@umass.edu

1 Introduction

Grammatical error correction (GEC) is the task of correcting spelling, punctuation, and other grammatical errors in ungrammatical text. Over the past decade, neural machine translation-based approaches to GEC, which generate the grammatical version of a sentence from its ungrammatical form, have dominated the field. However, their slow inference speed, large data requirement, and poor explainability has led to research toward text-edit approaches, which make modifications to ungrammatical sentences to produce their grammatical form.

In our work, we model GEC as a sequential, two-step problem that involves edits on the character level. The first step is deletion, where we fine tune a Reformer, a character-level transformer, to predict whether or not to delete each character. The second step is insertion, where we predict which characters to insert between existing characters to produce a grammatical sentence. These operations can be trained independently, using the longest common subsequence between the ungrammatical and grammatical sentence as an intermediary label. We plan to experiment with different implementations of the deletion and insertion mechanisms, and compare our results to existing approaches on several GEC benchmarks.

2 Related work

There exists a great deal of literature on the topic of grammatical error correction. The literature commonly approaches GEC from two points of view – as a sequence-to-sequence task and a text-edit task.

2.1 Sequence-to-sequence

Under the sequence-to-sequence formulation, an ungrammatical sentence is given as input, and its grammatical counterpart is generated. These approaches borrow models originally created for neural machine translation (NMT) (Chollampatt and Ng, 2018). More recently, transformer-based (Vaswani et al., 2017) sequence-to-sequence models developed for NMT have achieved state-of-the-art GEC performance (Grundkiewicz et al., 2019).

The literature has highlighted several difficulties that make large sequence-to-sequence models difficult to train and deploy for real-world use (Omelianchuk et al., 2020; Wiseman et al., 2018). These models suffer from slow inference speed, poor interpretability and explainability, and require a large amount of training data.

However, recent research in synthetic data generation has been shown to boost the performance of sequence-to-sequence models (Stahlberg and Kumar, 2021); these methods produce ungrammatical sentences that correspond to grammatical sentences, which can be used to train GEC models in a self-supervised fashion.

2.2 Text-edit

Grammatical error correction is alternatively viewed as a text-edit task, where an ungrammatical sentence is given as input, and a sequence of deletions, insertions, or other modifications are predicted to create the corresponding grammatical sentence.

Malmi et al. (2019) propose LaserTagger, a sequence tagging approach that corrects ungrammatical sentences using three edit operations. For each token in the ungrammatical sentence, LaserTagger predicts whether to (1) keep or (2) delete the token as well as whether or not to (3) insert a common phrase (from a predetermined phrase vocabulary) before the token. The authors find that predicting each tag simultaneously (keep/delete and/or insert) performs worse than assigning tags in an autoregressive, left-to-right fashion.

Stahlberg and Kumar (2020) propose Seq2Edits, a text-edit approach that predicts a sequence of span-based edit operations.

Omelianchuk et al. (2020) propose GECToR, a sequence tagging approach that assigns token-level transformations which map each token of the ungrammatical input to its corresponding token in the grammatical target. Since GECToR predicts transformations instead of tokens, each edit is explicitly explainable. GECToR is trained using both synthetic and annotated data, and it achieves state-of-the-art performance on the CoNLL-2014 (Ng et al., 2014) and BEA-2019 (Bryant et al., 2019) benchmarks.

3 Approach

We plan to improve on existing text-edit approaches for GEC through two significant modifications: (1) text-edits at the character level and (2) the division of GEC into two independent sub-tasks.

3.1 Character-level edits

Text-edit approaches to GEC are motivated by the large overlap between ungrammatical sentences and their grammatical counterparts. Taking this argument one step further, we plan to make perform GEC at the character level motivated by the observation that ungrammatical words and their grammatical counterparts often differ by a small number of characters.

3.2 Sub-tasks

Existing text-edit GEC approaches were designed to operate on the word or sub-word level, and they cannot directly transfer for use at the character-level. As a result, we propose to divide GEC into two independent sub-tasks performed sequentially: (1) character deletion and (2) character insertion.

The basis for the sub-task breakdown is the longest common subsequence (LCS), which can be used as an intermediary label to enable independent models for the deletion and insertion tasks. Consider the following ungrammatical (UG) and grammatical (G) sentences along with their LCS:

UG: `Children are played football.`
LCS: `Children are play football.`
G: `Children are playing football.`

To map an ungrammatical sentence to the LCS, characters only need to be deleted. To map the LCS to the grammatical sentence, characters only need to be inserted. The LCS, which can be computed programmatically using dynamic programming or other algorithms such as the Myers difference algorithm (Myers, 1986), can serve as the ground-truth for the character deletion sub-task and the input to the character insertion sub-task.

3.3 Implementation

Our implementation will consist of two networks called the deletion and insertion modules. Each module will have a separate pre-trained character-level transformer and will be fine-tuned for its corresponding sub-task. We will use the Reformer, which is an efficient transformer pre-trained as a character-level language model (Kitaev et al., 2020). Please refer to Figure 1 for an overview of the architecture.

Deletion module: This is a multi-task binary classification module to predict whether to keep/delete each character in a sentence. The ground truth for this module will consist of the longest common subsequence between the ungrammatical sentence and its grammatical form. We will feed the ungrammatical sentence to the Reformer, apply a linear layer to map each character representation to a single value, and use the sigmoid function to determine whether to keep/delete the character. We will train the linear layer from scratch and fine-tune the Reformer. The loss will be averaged over character-wise binary cross-entropy loss. We will experiment with performing deletions in a sequential manner and may make need-based changes to the network during our implementation.

Insertion module: This module will predict characters that can be added between two adjacent characters or at the start/end of an incorrect sentence. The ground truth for this module will be the original grammatically correct sentence. During training, we will feed the ground truth of the deletion module (longest common subseqeuence) as an input to the Reformer. We will concatenate the adjacent outputs from the Reformer, corresponding to adjacent characters, and pass them through the softmax layer, which will predict a probability distribution over the entire character vocabulary with an extra class corresponding to no insertion. We will use an independent pre-trained Reformer,

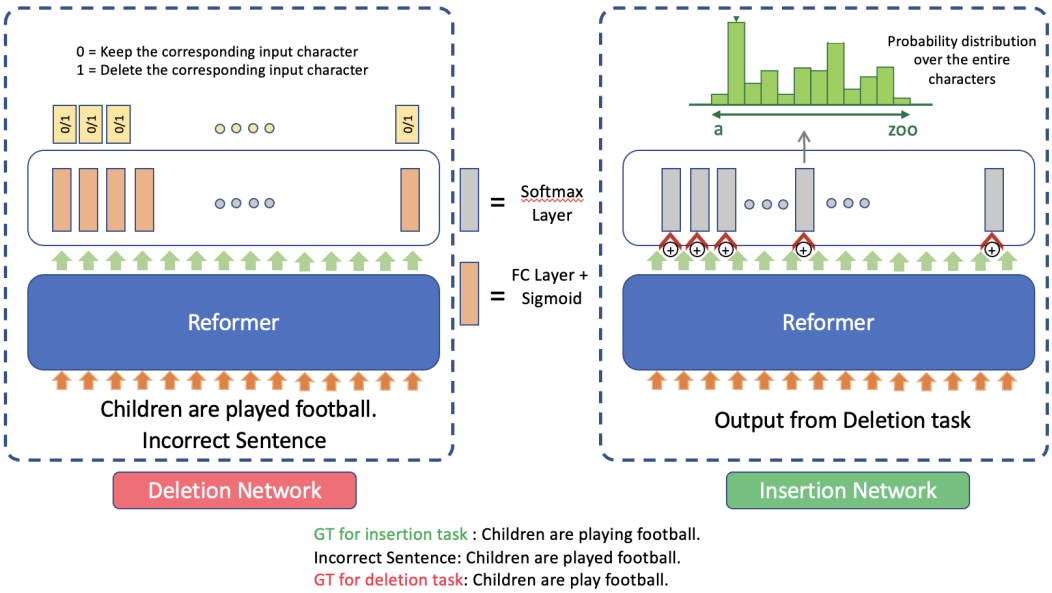

Figure 1: Grammatical Error Correction at the Character Level Architecture Design

which we will fine-tune during training.
We will train insertion and the deletion modules independently. At test time, we will first pass our ungrammatical sentences through the deletion module, and the output will be fed to the insertion module to generate the predicted grammatical sentence. We also plan on experimenting with insertion and deletion modules as a complete network.

What baseline algorithms will you use? Since GEC is a popular area of research, we will compare our approach directly to existing approaches.

Our approach views GEC as a text-edit task, so we compare our approach to the text-edit approaches described in the related work section: LaserTagger (Malmi et al., 2019), Seq2Edits (Stahlberg and Kumar, 2020), and GECToR (Omelianchuk et al., 2020). The published work of each model describes its performance on the public CoNLL-2014 and/or BEA-2019 datasets, enabling direct comparisons to our approach.

3.4 Schedule

Our group will be working collectively on each task. The following broad tasks will be undertaken (refer Figure 2) -

1. Environment setup (1 week)
2. Data exploration and preparation (3 weeks)
3. Model implementation and evaluation (3 - 4 weeks)
4. Error analysis (2-3 weeks)
5. Final report (1-2 weeks)

4 Data

We will use the following publically available datasets to train the model -

1. C4_200M Synthetic Dataset for Grammatical Error Correction (Stahlberg and Kumar, 2021)
2. Lang-8 Corpus (Mizumoto et al., 2011)
3. NUCLE (Dahlmeier et al., 2013)
4. WI-LOCNESS (Bryant et al., 2019)

The final model can be evaluated against several GEC benchmarking datasets such as -

1. CoNLL-2014 Shared Task (Ng et al., 2014)
2. BEA Shared Task - 2019 (Bryant et al., 2019)
3. JFLEG (Napoles et al., 2017)

5 Tools

We will use PyTorch along with the 🤗 Transformers and PyTorch Lightning libraries to implement our models. 🤗 Transformers has an implementation and pretrained version of the Reformer. We will run our code on either our personal GPU or Google Colab.

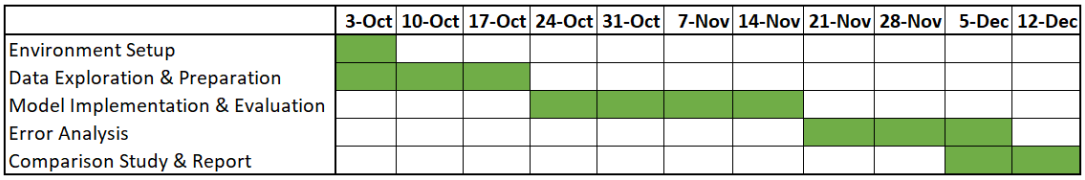

Figure 2: Project Schedule

References

Bryant, C., Felice, M., Andersen, Ø. E., and Briscoe, T. (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.

Chollampatt, S. and Ng, H. T. (2018). A multilayer convolutional encoder-decoder neural network for grammatical error correction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Dahlmeier, D., Ng, H. T., and Wu, S. M. (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. Association for Computational Linguistics.

Grundkiewicz, R., Junczys-Dowmunt, M., and Heafield, K. (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.

Kitaev, N., Kaiser, Ł., and Levskaya, A. (2020). Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Malmi, E., Krause, S., Rothe, S., Mirylenka, D., and Severyn, A. (2019). Encode, tag, realize: High-precision text editing. *arXiv preprint arXiv:1909.01187*.

Mizumoto, T., Komachi, M., Nagata, M., and Matsumoto, Y. (2011). Mining revision log of language learning SNS for automated Japanese error correction of second language learners. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 147–155. Asian Federation of Natural Language Processing.

Myers, E. W. (1986). Ano (nd) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266.

Napoles, C., Sakaguchi, K., and Tetreault, J. (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. Association for Computational Linguistics.

Ng, H. T., Wu, S. M., Briscoe, T., Hadiwinoto, C., Susanto, R. H., and Bryant, C. (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.

Omelianchuk, K., Atrasevych, V., Chernodub, A., and Skurzhanskyi, O. (2020). Gector–grammatical error correction: Tag, not rewrite. *arXiv preprint arXiv:2005.12592*.

Stahlberg, F. and Kumar, S. (2020). Seq2edits: Sequence transduction using span-level edit operations. *arXiv preprint arXiv:2009.11136*.

Stahlberg, F. and Kumar, S. (2021). Synthetic data generation for grammatical error correction with tagged corruption models. In *Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 37–47, Online. Association for Computational Linguistics.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Wiseman, S., Shieber, S. M., and Rush, A. M. (2018). Learning neural templates for text generation. *arXiv preprint arXiv:1808.10122*.