Character Transformations for Non-Autoregressive GEC Tagging

Milan Straka and **Jakub Náplava** and **Jana Straková**
Charles University,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics
{straka,naplava,strakova}@ufal.mff.cuni.cz

Abstract

We propose a character-based non-autoregressive GEC approach, with automatically generated character transformations. Recently, per-word classification of correction edits has proven an efficient, parallelizable alternative to current encoder-decoder GEC systems. We show that word replacement edits may be suboptimal and lead to explosion of rules for spelling, diacritization and errors in morphologically rich languages, and propose a method for generating character transformations from GEC corpus. Finally, we train character transformation models for Czech, German and Russian, reaching solid results and dramatic speedup compared to autoregressive systems. The source code is released at https://github.com/ufal/wnut2021_character_transformations_gec.

1 Introduction

The current state of the art for grammatical error correction (GEC) is achieved with encoder-decoder architectures, leveraging large models with enormous computational demands (Grundkiewicz et al., 2019; Rothe et al., 2021). As such autoregressive approach is slow on inference and is impossible to parallelize, it has recently been suggested to replace autoregressive sequence-to-sequence decoding with per-token tagging to enable parallel decoding, achieving a dramatic speedup by a factor of 10 in NMT (Gu et al., 2018) and very recently, also in GEC (Omelianchuk et al., 2020).

Omelianchuk et al. (2020) approaches GEC as a tagging task, discriminating between a set of word-level transformations. The designed set is efficient for English corpora, which rarely contain spelling errors, and for English language, which does not have diacritization marks and its morphology is very modest compared to morphologically rich languages such as Czech or Russian. Using a set of word-level transformations designed for English,

all character-level corrections would have to be handled by generic word-for-word REPLACE rule, leading to an explosion of rules.

We therefore suggest character transformations on subword level. Moreover, our transformations are automatically inferred from the corpus as opposed to being manually designed. Our approach has the following advantages:

- character-level errors, such as diacritics, spelling and morphology are handled,
- the transformations can be shared between subwords, preventing an explosion of rules,
- the transformations are generated automatically from corpus for each language.

We present an oracle analysis of various transformations sets at different levels, in English and three other languages: Czech, German and Russian. We find that the word-level set of rules may be suboptimal for morphologically rich languages and corpora with spelling errors and diacritics.

Finally, we train models with character transformations for non-autoregressive grammatical error correction in Czech, German and Russian, reaching solid results and dramatic speedup compared to autoregressive systems.

2 Related Work

Awasthi et al. (2019) propose an alternative to popular encoder-decoder architecture for GEC: a sequence-to-edit model which labels words with edits. Its advantage is parallel decoding while keeping competitive results. Mallinson et al. (2020) introduce a framework consisting of two tasks: *tagging*, which chooses and arbitrarily reorders a subset of input tokens to keep, and *insertion*, which in-fills the missing tokens with another pretrained masked language model. Omelianchuk et al. (2020) develop custom, manually designed, per-token *g-transformations*.

We further improve the sequence-to-edit model

Algorithm 1: Extended LCS Alignment

Input: input subwords s, gold characters g

$w[:,:] \leftarrow 0$

for $i \leftarrow |s|$ **to** 1 **do**
    for $j \leftarrow |g|$ **to** 1 **do**
        $w[i,j] \leftarrow w[i+1,j]$
        for $l \leftarrow 1$ **to** $\max(|g|-j, 8+3|s[i]|)$ **do**
            $g \leftarrow g[j:j+l]$
            if *g.isspace()* **then continue**
            $c \leftarrow \begin{cases} 1 & \textbf{if } s[i] = g \\ 0.75 & \textbf{if } s[i].strip() = g.strip() \\ 0.5 \cdot \text{LevenshSimilarity}(s[i], g) \end{cases}$
            $w[i,j] \leftarrow \max(w[i,j], c + w[i,j+1])$
        end
    end
end

return alignment with weight $w[0,0]$, as in LCS

---

with an attempt at non-autoregressive grammatical error correction for languages other than English, with *character* transformations applied at *subwords*, inferred *automatically* from parallel GEC corpus.

3 Transformations

Given that we encode an input sentence using BERT (Devlin et al., 2019), it is natural to represent it using a sequence of *subwords*. We prepend a space to every first subword in a word and use no special marker for other subwords. Note that the subwords might not correspond directly to parts of input, because the *bert uncased* model strip input casing and diacritics.

3.1 Alignment

In order to *automatically* encode the gold data via *character* transformations, we first align the input subwords and the corrected sentence. We compute the alignment with Algorithm 1, which is an extended version of LCS, where each subword is aligned with a *sequence* of gold characters. We ignore casing, diacritics and consider all punctuation equal during the alignment, and bound the maximum length of a correction (number of characters aligned to a single input subword) by $8+3\cdot$*input subword character length* for efficiency.

3.2 Transformations

We consider four kinds of transformations, differing in two dimensions – the granularity of the transformation and the unit it is applied on:

Input	gatherin		leafes	
Correct	**G**athering		leaves	
Subwords	␣gathe	rin	␣lea	fes
string-at-word	␣Gathering		␣leaves	
string-at-subword	␣Gathe	APP. g	KEEP	ves
char-at-word	APPEND g, UPPERC. 2nd		REPL. 3rd *from end with* v	
char-at-subword	UPPERC. 2nd	APP. g	KEEP	REPL. 1st *with* v

Figure 1: Example of the four types of transformations.

- character transformations applied on each subword separately (*char-at-subword*),
- character transformations applied on each complete word (*char-at-word*),
- string transformations applied on each subword (*string-at-subword*),
- string transformations applied on each complete word (*string-at-word*).

In such terminology, the transformations proposed by Awasthi et al. (2019) and Omelianchuk et al. (2020) can be referred to as *string-at-word*. An example of the described transformation types is illustrated in Figure 1.

To apply a transformation on a complete word, we concatenate the corresponding subwords and aim to produce the concatenation of the subwords' corrections.

A string transformation can be one of *keep*, *replace* by given string or *append* a given string before/after.

A character transformation consists of multiple character edits, which we construct as follows:

1. We start by computing the standard smallest *edit script* between an input subword and a correction. The edit script is a sequence of *inserts*, *replaces* and *deletes*, and we index each edit operation either from the beginning of the input subword (if it involves the first half of it) or from the end of it (otherwise). The edit script is computed on lowercased strings, and in case of *bert uncased* models, also on undiacritized strings.

2. Afterwards, the unmodified input subword (i.e., including casing in case of *bert cased* models) is processed by the edit script, obtaining a correction with possibly incorrect casing. If some incorrectly lowercased charac-

ters are indeed present, *uppercase* operations are added, each indexing a single character either from the beginning of the correction (if the character is in the first half of it) or from the end of it (otherwise).

3. Finally, for *bert uncased* models, we still need to handle missing diacritics. We achieve it analogously to step 2, adding missing diacritical marks with operations indexing single characters again either from the beginning of the correction (if the character is in the first half) or from the end of it.

The reason for special handling of casing (and diacritization for *bert uncased* models) is that the proposed rules are more general, allowing to capture for example corrections *go→Going* and *walk→Walking* with a single rule *append "-ing", uppercase first.*

4   Transformations Upper-bound F-score

To assess the effect of number and type of transformations, we compute the potential maximum $F_{0.5}$ score with the MaxMatch M^2 scorer (Dahlmeier and Ng, 2012).

We generate transformation dictionary from the training portion of the following GEC corpora: CoNLL-2014 shared task for English (Ng et al., 2014), AKCES-GEC (Náplava and Straka, 2019) for Czech, FALKO-MERLIN GEC (Boyd, 2018) for German and RULEC-GEC (Rozovskaya and Roth, 2019) for Russian. The sizes of the datasets are quantified in Table 1.

We also generate transformations from synthetic data augmentation used for training (Section 5); to prevent the explosion of the transformation dictionary, we consider only 1000 synthetic sentences, except for Russian, which employs 5000 synthetic sentences because of very small authentic data. Finally, we add a special *uncorrectable error* dictionary element, indicating an error that cannot be corrected by any dictionary transformation.

To encode a gold correction with a transformation, we first try looking it up in the dictionary. If it is not present, we go through all dictionary transformations in random order, accepting the first one producing the correct output. If no transformation match, we resort to the *uncorrectable error* (during prediction, it will be replaced by the input token).

We show all combinations (*character/string* at *words/subwords*), using cased and uncased mBERT, in Figure 2. Clearly, character transforma-

Language	Dataset	Sentences
Czech	AKCES-GEC	47 371
German	Falko-MERLIN	24 077
Russian	RULEC-GEC	12 480

Table 1: GEC datasets used for constructing rules and for evaluation, including their size.

tions applied at subwords (*char-at-subword*, green) have the highest potential in terms of upper-bound $F_{0.5}$ in all four languages. At the same time, word replacements (*string-at-word*, red) do not scale well. This effect is emphasized in morphologically-rich Czech and Russian, for which the upper-bound string replacement $F_{0.5}$ (*string-at-word*, red) falls below the current GEC systems state-of-the-art $F_{0.5}$ (horizontal dotted line).

5   Experiments

We train the character subword GEC tagging model using the *char-at-subword* transformation, which have achieved the best upper-bound score.[1] We train the character subword GEC tagging model (*char-at-subword*) for Czech, German and Russian, in two stages: First, models are trained on a large synthetic corpus, generated by a reimplementation of Náplava and Straka (2019). Then, the models are finetuned on a mixture of synthetic and authentic data in ratio 1:2. The authentic data used in the second stage are AKCES-GEC (Náplava and Straka, 2019) for Czech, FALKO-MERLIN GEC (Boyd, 2018) for German and RULEC-GEC (Rozovskaya and Roth, 2019) for Russian.

The model is based on a pretrained BERT encoder (Devlin et al., 2019), specifically *bert-base-multilingual* (*uncased* for Czech and German, *cased* for Russian). After encoding the tokens, we add a simple softmax classifier that projects embeddings for each subword into a distribution over a set of transformations (Section 3) generated from authentic data with a limited addition from synthetic data (Section 4). We generate 7.7k transformations for Czech, 4.3k transformations for German and 3.1k transformations for Russian.

GEC models based on Transformer and BERT-encoder were shown to perform better when applied iteratively (Lichtarge et al., 2018;

---

[1]We also performed preliminary experiments with *char-at-word* GEC tagging model, and it performed worse than using the *char-at-subword* transformations.
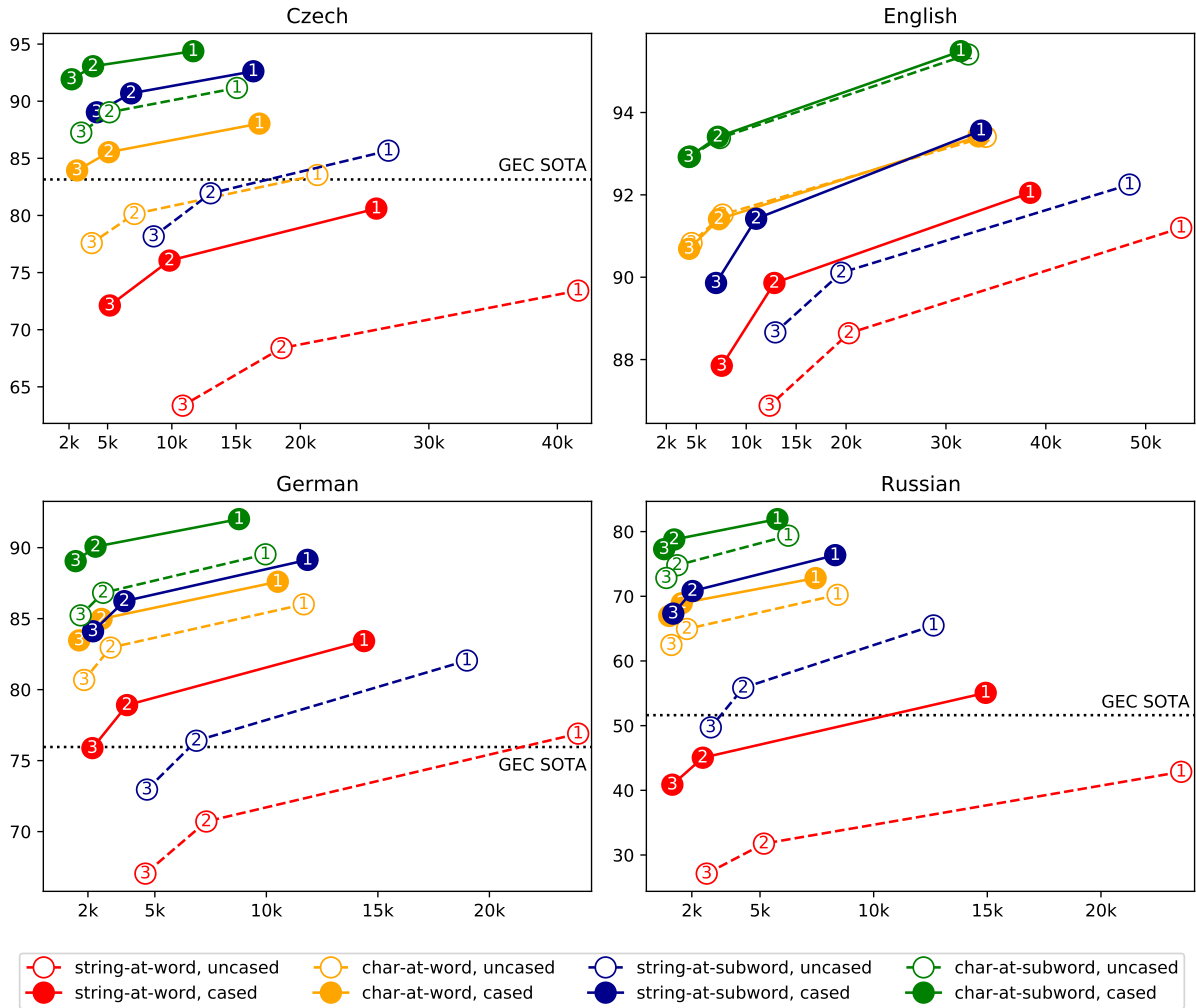
Figure 2: $F_{0.5}$ depending on number and type of transformations, if all transformations were correctly predicted (upper-bound). Up and right is better (higher $F_{0.5}$, fewer rules), down and left is worse (lower $F_{0.5}$, more rules). Circled numbers ①, ② and ③ denote that we kept transformations present at least once, twice or three times in the training data, respectively (larger means less transformations).

Omelianchuk et al., 2020). Therefore, we experiment with multiple iterations and report results both for single iteration and four iteration phases after which we did not observe significant improvements.

We train both the fully-connected network and BERT with AdamW optimizer (Loshchilov and Hutter, 2019) which minimizes the negative log-likelihood. Both for pretraining and finetuning, the learning rate linearly increases from 0 to $5 \cdot 10^{-5}$ over the first 10000 steps and linearly decreases to 0 over 20 epochs. We use the batch size of 2048 sentences and clip each training sentence to 128 tokens. We pretrain each model for circa 14 days and finetune it for circa 2 days on Nvidia P5000 GPU and select the best checkpoint according to development set.

We experimented with weighting all classes different from the KEEP instruction by a factor of 3. It turned out effective only for pretraining Russian.

We present the results of our models in Table 2. Compared to autoregressive models of similar size (Náplava and Straka, 2019), our models achieve solid results with large speedup due to the non-autoregressive tagging approach. Obviously, the inflation of model size (Rothe et al., 2021) to enormous size (13B parameters) leads to further improvements at the cost of increased computational demands.

5.1 Runtime Performance

To evaluate the speed-up of the non-autoregressive decoding, we compare the runtime performance of our system to the autoregressive Transformer

Model	Params	$F_{0.5}$
Richter et al. (2012)		58.54
Náplava and Straka (2019)^{synt}	210M	66.59
Náplava and Straka (2019)^{fine}	210M	80.17
Rothe et al. (2021) *base*	580M	71.88
Rothe et al. (2021) *xxl*	13B	83.15
Ours synthetic	172M	64.29
Ours finetuned	172M	72.86
Ours finetuned 4 iterations	172M	75.06

(a) Czech

Model	Params	$F_{0.5}$
Boyd (2018)		45.22
Náplava and Straka (2019)^{synt}	210M	51.41
Náplava and Straka (2019)^{fine}	210M	73.71
Rothe et al. (2021) *base*	580M	69.21
Rothe et al. (2021) *xxl*	13B	75.96
Ours synthetic	170M	44.29
Ours finetuned	170M	62.92
Ours finetuned 4 iterations	170M	65.95

(b) German

Model	Params	$F_{0.5}$
Rozovskaya and Roth (2019)		21.00
Náplava and Straka (2019)^{synt}	210M	40.96
Náplava and Straka (2019)^{fine}	210M	50.20
Rothe et al. (2021) *base*	580M	26.24
Rothe et al. (2021) *xxl*	13B	51.62
Ours synthetic	180M	25.36
Ours finetuned	180M	36.62
Ours finetuned 4 iterations	180M	38.68

(c) Russian

Table 2: Model results

Model	Time Per Sentence
T2T	162.34
BERT-GEC	41.26

(a) CPU decoding on a 32-core Intel Xeon

Model	Time Per Sentence
T2T	22.36
BERT-GEC	5.09

(b) GPU decoding on Nvidia Quadro P5000

Table 3: Average time in milliseconds required to process a single sentence in the Czech test set, measured using both (a) CPU decoding and (b) GPU decoding.

6 Conclusion And Future Work

We proposed a character-based method to generate target transformation instructions for GEC tagging models, as an alternative to autoregressive models. We compared the character transformations to previously used word-level transformation instructions and have shown that character-based rules have better coverage and scale better in Czech, German and Russian. Moreover, we trained character-based GEC tagging models for these languages. The source code is available at `https://github.com/ufal/wnut2021_character_transformations_gec`.

For future work, we propose to investigate ways to generate synthetic data to achieve better coverage of the target transformation set, since the current process for generating synthetic errors is well suited for encoder-decoder models, but may fail to cover certain transformations.

Acknowledgements

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*

encoder-decoder architecture from Náplava and Straka (2019), which is of comparable size. The measurements are performed using both a CPU-only decoding (performed on a dedicated 32-core Intel Xeon E5-2630) and GPU decoding (measured on an Nvidia Quadro P5000). The results presented in Table 3 show that the non-autoregressive system is four times faster.

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.

Adriane Boyd. 2018. Using Wikipedia edits in low resource grammatical error correction. In *Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text*, pages 79–84, Brussels, Belgium. 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, 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.

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

Jared Lichtarge, Christopher Alberti, Shankar Kumar, Noam Shazeer, and Niki Parmar. 2018. Weakly supervised grammatical error correction using iterative decoding. *arXiv preprint arXiv:1811.01710*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Jonathan Mallinson, Aliaksei Severyn, Eric Malmi, and Guillermo Garrido. 2020. FELIX: Flexible text editing through tagging and insertion. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1244–1255, Online. Association for Computational Linguistics.

Jakub Náplava and Milan Straka. 2019. Grammatical error correction in low-resource scenarios. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 346–356.

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.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. GECToR – grammatical error correction: Tag, not rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 163–170, Seattle, WA, USA → Online. Association for Computational Linguistics.

Michal Richter, Pavel Straňák, and Alexandr Rosen. 2012. Korektor–a system for contextual spell-checking and diacritics completion. In *Proceedings of COLING 2012: Posters*, pages 1019–1028.

Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. 2021. A simple recipe for multilingual grammatical error correction. *CoRR*, abs/2106.03830.

Alla Rozovskaya and Dan Roth. 2019. Grammar error correction in morphologically rich languages: The case of russian. *Transactions of the Association for Computational Linguistics*, 7:1–17.