Improved grammatical error correction by ranking elementary edits

Anonymous ACL submission

Abstract

We offer a rescoring method for grammatical error correction which is based on two-stage procedure: the first stage model extracts local edits and the second classiifies them as correct or false. We show how to use an encoder-decoder or sequence labeling approach as the first stage of our model. We achieve state-of-the-art quality on BEA 2019 English dataset even with a weak BERT-GEC basic model. When using a state-of-the-art GECToR edit generator and the combined scorer, our model beats GECToR on BEA 2019 by $2-3\%$. Our model also beats previous state-of-the-art on Russian, despite using smaller models and less data than the previous approaches.

1 Introduction

Grammatical error correction (GEC) is a task of converting the source to text to its clean version with no orthographic, punctuation, lexical or other errors. As any sequence-to-sequence task, it is often solved using machine translation methods mostly using Transformer architecture(Vaswani et al., 2017) or its variants. One of the few successful exceptions is the GECToR model (Omelianchuk et al., 2020), which reduces GEC to sequence labeling, however, it exists only for English. In sequence-to-sequence models decoding is usually done using beam search, which has two serious drawbacks. The first is exposure bias: the model was never exposed to its errors in training time which complicates the recovery from errors during decoding. The second is left-to-right nature of decoding: the model can make a wrong decision not observing the future context. This does not hold for a reranker model, since it explores entire corrected sequences and thus may utilize richer context. Reranking might also be helpful when the correct edit is not ranked as topmost one, but still appears it the n-top list of hypotheses. Due to these reasons, reranking was heavily used in machine translation both in statistical (Och et al., 2004) and neural (Yee et al., 2019) era.

In contrast to machine translation, sequence editing in GEC can be decomposed to elementary edits such as modifying a single word or a consecutive group of words. In this paper we propose to score elementary edits produced by the basic model and classify them as positive or negative on the second stage of the pipeline. Than the calculated probabilities can be either used directly or combined with the scores from the first stage. Since the additional ranking stage utilizes not only the topmost hypothesis but the k-best list, it may help to recover from errors made by the basic model and increase both precision and recall.

We show that even the scoring model alone achieves state-of-the-art performance on BEA2019 dataset for two variants of the first stage model. Its combination with GECToR model outperforms the models of same size by about 2 points F0.5 score. We also beat current SOTA on Russian with two variants of the basic edit generator.

2 Edit generation

As proposed in Alikaniotis and Raheja (2019), probably the simplest approach to grammatical error correction is to generate possible edits using a rule-based model and then extract those that increase the sentence probability by a sufficient margin. The straightforward way to estimate sentence probability is to use a Transformer language model, such as GPT(Radford et al., 2019) or BERT (Devlin et al., 2019). This approach requires no training data, only a development set for tuning the hyperparameters. As a reverse side of its simplicity, this algorithm has two main limitations:

- Recall is limited to errors that can be specified by the rules.
- The probability estimators are imperfect, especially when the edit changes sequence length.

Therefore the main idea of our paper is to replace the scorer by a more powerful trainable model. Another key detail is that we apply the scorer not to the full corrections, but to the elementary edits. Namely, given the erroneous sentence **The boy fall on floor* and its correction *The boy fell on the floor*, our model should return **True** for sentences *The boy fell on floor* and *The boy fall on the floor* and **False** for other elementary corrections, for example, **The boy falls on the floor*.

So, our model includes three main stages, described subsequently:

1. Extracting elementary edits from the basic model.
2. Classifying these edits as positive or negative.
3. Applying the positively classified edits to the source sentence.

The first part in described in this section and the remaining two in Section 3. A schematic description of our algorithm is given in Figure 1

2.1 Rule-based edit generator

We start with describing edits extraction based on linguistically motivated rule-based model. It may be considered as our reimplementation of Alikaniotis and Raheja (2019). Our edit generation module takes as input the dependency tree of a sentence and applies rule-based edits corresponding to the most frequent errors, such as missing or incorrect determiners, commas and prepositions or wrong choice of word form. The exact list of applied rules is given in Appendix A.1.

These operations produce a fairly large number of possible corrections. To reduce computational burden we apply two-stage filtering. First, for every hypothesis $\mathbf{u}$ we calculate the gain $\log p(u_{\pi+1}|w_1...w_\pi) - \log p(w_{\pi+1}|w_1...w_\pi)$, where π is the length of longest common prefix of u and source sequence $\mathbf{w}$[1]. We choose best K_{del} deletions, K_{ins} insertions and K_{sub} replacement edits according to this score. Then for the selected hypotheses we calculate their full log-probability and pick K best variants provided their score exceeds $p(\mathbf{w}) - \theta$[2].

2.2 Sequence-to-sequence edit generator

To generate edits using a sequence-to-sequence basic model we run standard beam search, align all the produced hypotheses with the source sentence and extract non-trivial parts of such alignments. The score of edit $\mathbf{e}$ equals $\log p(\mathbf{u}|\mathbf{w}) - \log p(\mathbf{v}|\mathbf{w})$, where $\mathbf{u}$ denotes the most probable hypothesis containing $\mathbf{e}$ and $\mathbf{v}$ is the most probable hypothesis that changes nothing in the span of $\mathbf{e}$. If there is no such hypothesis, we set the score to $\log p(\mathbf{u}|\mathbf{w}) - \log p(\mathbf{v}|\mathbf{w}) + 1$, where $\mathbf{v}$ is the last hypothesis in the beam. We experimented with restricting beam search only to hypotheses with one elementary edit and diverse beam search, however, that makes the implementation more complicated without any performance gains.

2.3 Sequence labeling generator

In contrast to other methods, the recent GECToR model (Omelianchuk et al., 2020) reduces grammar error correction to sequence tagging. We give an example of such reduction in Table 1 and refer the reader to Sections 3 and 5 of the original paper to better understand their approach. GECToR operations naturally correspond to elementary edits in our terminology. For each position i we extract all the tags $\mathbf{t}$ such that

$$\log p(t_i = \mathbf{t}) \geq \log p(t_i = \text{KEEP}) - \theta,$$

where θ is the predefined margin. For example, if on the first step of the example in Table 1 we have $p(t_3 = \text{VBD}) = 0.5,\ p(t_3 = \text{VBZ}) = 0.3,\ p(t_3 = \text{KEEP}) = 0.1$, then the VBZ transformation *fall* $\rightarrow$ *falls* will also be extracted. Again, we keep top K edits according to the difference between logarithmic probabilities of the edit and the the default "do nothing" operation (the KEEP tag).

For all the extraction methods we label as positive all edits that appear in the .m2 description of the dataset or may be partitioned to such edits. We also add the "do nothing" edit that returns the source sentence. It is treated as positive if the sentence is already correct.

3 Model description

3.1 Edits classification

Given numerous successes of Transformer models in NLP, we decide to use Roberta(Liu et al., 2019) for edit classification. It takes as input the sequence

$$\mathbf{x} = \langle \text{BOS} \rangle \text{SOURCE} \langle \text{SEP} \rangle \text{EDITED\_SOURCE} \langle \text{EOS} \rangle$$

and outputs the probability of the edited source to be a plausible correction. Consider the sequence

[1] This scoring is performed in one pass of left-to-right LM.
[2] We set $K_{del} = 10, K_{ins} = 10, K_{sub} = 30, K = 15, \theta = 3.0$.

Source	Edit generator	Model score	Stage 1	Stage 2	Stage 3	Stage 4	Target
The boy fall on floor	$(0, 1, \textit{boys})$	0.53	?	?	?	×	*The boy fell on the floor*
	$(1, 2, \textit{falls})$	0.7	?	×	×	×	
	$(1, 2, \textit{fell})$	0.83	?	✓	✓	✓	
	$(3, 3, \textit{the})$	0.9	✓	✓	✓	✓	
	$(-1, -1, \text{None})$	0.57	?	?	?	✓ (terminate)	

Figure 1: The pipeline of our algorithm. On each decoding stage, the most probable (in red) remaining action is selected. It also eliminates other edits with intersecting spans (in blue). In the end all the selected operations are applied in parallel.

Iter.	Source	Edits	Result
1	CLS Boy fall the floor	APPEND_The LOWER VBD KEEP KEEP	The boy fell the floor
2	CLS The boy fell the floor	KEEP KEEP KEEP APPEND_on KEEP KEEP	The boy fell on the floor

Table 1: An example of GECToR labeling and corresponding sentence edits.

$\mathbf{x} = \text{BOS}\ x_1 \dots x_L\ \text{SEP}$ $x'_1 \dots x'_{L+\delta}\ \text{EOS}$ and let $x_i \dots x_j$ and $x'_i \dots x'_{j+\delta}$ be the source and the target of the edit, respectively. Then our classification model M can be decomposed as

$$M(\mathbf{x}) = g(f(\text{READOUT}(\text{ENCODER}(\mathbf{x})))),$$

where

- ENCODER is the Transformer encoder that produces the embedding[3] sequence $\mathbf{h} = h_{\text{BOS}} h_1 \dots h_L h_{\text{SEP}} h'_1 \dots h'_{L+\delta} h_{\text{EOS}}$.
- READOUT is the readout function that converts a sequence of embeddings to the vectorization of the whole input. We use the first embedding of the target span and consider other variants during ablation in Appendix E.
- f is a multilayer perceptron and g is the final classification layer with sigmoid activation.

3.2 Decoding

After classifying the edit we cannot simply apply all edits classified as positive as they may conflict each other (e.g., the edits *fall* → *fell* and *fall* → *falls* for the sentence *The boy fall on the floor*). The conflicts may also happen between adjacent edits (*boy* → *boys* and *fall* → *falls*) thus we consider as contradicting any two edits whose source spans either intersect or are adjacent and non-empty. We test two decoding strategies:

1. (offline, faster) Pick the edits whose probability is greater than the maximum of predefined threshold and "do nothing" edit score. Keep those that do not contradict any edits with higher scores.
2. (online, giving higher scores) If the most probable edit is "do nothing" or its probability is below threshold, stop. Otherwise select the most probable edit, apply it to the current input sentence and remove all the edits with intersecting spans. Repeat this until reaching the maximal number of iterations.

[3] Through all the paper ‘embeddings‘ means the decoder output for current subtoken.

The optimal threshold is model-dependent, we optimize it on development set.

4 Data and experiments

4.1 Data

We apply our model to English data using the BEA 2019 Shared Task data (Bryant et al., 2019). We use the same training data as in the previous works: Write&Improve and LOCNESS corpus (Bryant et al., 2019), First Certificate of English (FCE) (Yannakoudakis et al., 2011), National University of Singapore Corpus of Learner English (NUCLE) (Dahlmeier et al., 2013), Lang-8 Corpus of Learner English(Tajiri et al., 2012) and synthetic data(Awasthi et al., 2019). We test our models on BEA 2019 development and test sets and CoNLL 2014(Ng et al., 2014) test data.

For additional experiments we also use cLang8 (Rothe et al., 2021) – the cleaned and extended version of Lang8 corpus. The characteristics of datasets are given in Table 2.

4.2 Model architecture and training

We initialize the transformer using the weights of pretrained roberta-base. We take the encoding of

Dataset	Size	Usage
W&I+LOCNESS	34308	Train, finetune
FCE	28350	Train
NUCLE	57151	Train
Lang8	1037561	Train
PIE synthetic	9000000	Pretrain
BEA 2019 dev	4384	Development
BEA 2019 test	4477	Test
CoNLL14	1312	Test
cLang8	2372119	Train

Table 2: Training data for English GEC experiments.

the leftmost word in the target span as sequence representation and process it by a 1-layer perceptron with output dimension 768 and ReLU activation. The output of this perceptron is passed to the final linear layer with sigmoid activation. We implement our models using PyTorch and use HuggingFace roberta-base implementation[4].

We follow the training procedure described in (Omelianchuk et al., 2020). Namely, after pretraining on synthetic data only we perform the main training on full BEA 2019 train set which is the concatenation of W&I+LOCNESS, FCE, NUCLE and Lang8 and afterwards finetune the model on W&I+LOCNESS. When using cLang8 instead of Lang8 we do not apply pretraining.

The model is trained using total batch size of 3500 subtokens to fit into 32GB GPU memory. All the examples for a single sentence are placed to the same batch. Since the number of proposed negative edits is much larger than the number of positive ones, we independently average the loss for positive and negative examples inside each batch. We optimize the model with AdamW optimizer using default hyperparameters.

4.3 Edit generation

We test three models of edits generation: the first is a rule-based baseline, BERT-GEC(Kaneko et al., 2020) is a sequence-to-sequence model[5] and GECToR is a sequence labeling model of state-of-the-art quality. We apply beam search (Subsection 2.2) with BERT-GEC[6] and the extension described in Subsection 2.3 with GECToR[7]. In all the variants we extract at most 15 hypotheses such that their score is greater than -3.0[8]. Recalls are given in Table 3.

Dataset	Rule-based	BERT-GEC	GECToR
BEA 2019 dev	45.8	55.5	54.9
W&I train	46.7	61.0	66.3
FCE	40.4	60.7	56.6
NUCLE	39.6	48.3	45.0
Lang8	33.0	50.2	43.3
BEA dev F0,5	< 40	48.8	54.1

Table 3: Recall of different edit extraction methods for English. W&I is W&I+LOCNESS.

We observe that BERT-GEC and GECToR has similar recall on BEA data, while on other datasets BERT-GEC coverage is better despite its lower quality. The coverage of rule-based model is low because it cannot handle free rewriting in principle.

4.4 Main results

In this section we perform two experiments: in the first we select the best data selection based on results only after main training (without pretraining on synthetic data) without taking the scores of the basic model into account. In the second we compare the best performing model trained in full mode described in Subsection 4.2. Following the standard practice, we compare the models by F0.5 score using ERRANT (Bryant et al., 2019) for BEA development set and M2Scorer (Dahlmeier et al., 2013) for other datasets.

Results in Table 4 show that BERT-GEC and GECToR model hace comparable performance, while the rule-based model is behind them due to poor recall.

In our second experiment we evaluate the models trained on full data in two settings: using the scorer probabilities only ('no base model' row) and combining them with the score of the edit generator[9]. Precisely, we set the hypothesis score equal to $\log p_{\text{scorer}}(\mathbf{e}) + \alpha \cdot \text{score}_{\text{gen}}(\mathbf{e})$, where α is

[4]Our code is available on https://www.dropbox.com/s/ubcblvy63ynsfs7/edit_scorer.tar.gz

[5]It is the only seq2seq model with weights available online.

[6]https://github.com/kanekomasahiro/bert-gec

[7]We use the roberta-base GECToR model, which is available from https://github.com/grammarly/gector. Our edit generator code is available on https://www.dropbox.com/s/ncxcjyhbw3q845d/gector.tar.gz

[8]The threshold was tuned on development set.

[9]We do not provide the combined scores for BERT-GEC model as they do not show much improvement over the scorer due to basic model weakness.

Edit generation model	Precision	Recall	F0.5
Rule-based	59.8	22.7	45.1
+finetuning	**63.3**	28.1	50.6
BERT-GEC	65.9	23.7	48.6
+finetuning	62.1	33.9	**53.2**
GECToR	59.5	27.9	48.5
+finetuning	60.4	**34.1**	52.5

Table 4: Comparison of different edit generation schemes on BEA 2019 data. All the models are trained on full BEA 2019 train set and evaluated on the BEA 2019 development data. +finetuning rows refer to further finetuning on W&I-LOCNESS training data. The best results for each metric are in bold.

the tuned parameter[10]. In addition to the models trained on the same data as GECToR, we also evaluate here the version of our model trained on larger cLang8 corpora (Rothe et al., 2021).

As shown in Table 5, our basic model slightly outperforms SOTA GECToR model on BEA 2019 dev and test. Combining its scores with GECToR edit scores, we improve the performance by additional $1.5 - 2\%$. If we do not restrict the training data, on BEA 2019 test our model looses only to T5-XXL model, which is almost 2 orders of magnitude larger ($11B$ parameters instead of $200M$ of roberta-base). On CoNLL-2014 test set our models also show state-of-the-art performance, definitely loosing only to models that has larger size (Sun et al., 2021) and/or were pretrained with significantly more synthetic data.

5 Additional experiments

5.1 Model parameters and objectives

Since other approaches to reranking often use ranking loss, we experiment with adding the margin loss between correct and false edits ('+soft') or between pairs of the form 'correct edit'-'no edit' and 'no edit'-'incorrect edit' ('+contrast'). We also tested representing the hypothesis with mean embedding of the output span, not its first token ('mean'), and using the CLS token representation ('CLS'). We also verified the effect of replacing Roberta-base with either Electra(Clark et al., 2020) or Roberta-large(Liu et al., 2019).

Results in Table 6 show that additional losses help on small dataset but have negative impact on the full one. Taking the target span vector as edit representation is crucial, however, using mean vector of target span does not improve performance. Electra[16] and Roberta-large models significantly improve over the baseline, however, their effect is much smaller on full training dataset.

5.2 Decoding ablation

In this part of the paper we investigate how decoding procedure described in Subsection 3.2 affects the model performance. In the first experiment we vary the decoding algorithm and the decision threshold. In Table 7 we provide the scores for the model trained with GECToR edit generation on full training data before and after finetuning on W&I-LOCNESS training data. Another notable pattern is that before finetuning the best F0.5-score is achieved at threshold $0.6 - 0.7$, while afterwards the optimal threshold is $0.8 - 0.9$. These values are stable across datasets, so setting the threshold to 0.7 before finetuning and to 0.9 after it is nearly optimal, thus threshold tuning is unnecessary.

In Table 8 we also analyze how the quality of the model depends on the maximal number of edits allowed. We observe that recall and F0.5 score are improved up to 8 edits per example. The difference between offline and online algorithms is about $0.5 - 0.7$ F0.5 score. It follows the experience of (Omelianchuk et al., 2020), where iterative rewriting (the analogue of our online decoding) improved performance even more significantly.

5.3 Joining generators

A natural question about our method is whether it is merely a technique that exploits the data more effectively or a general model capable to classify edits as plausible or implausible. We address this question by defining a 'joint' edit generator that simply returns the union of BERT-GEC and GECToR edits. We apply to this data the classifiers trained with only edit type (either GECToR or BERT-GEC). Both the models are pretrained on synthetic data and trained on full BEA2019 train set in standard setting. Then we finetune the models on W&I-LOCNESS either using the same edit generator as

[10] In all the experiments optimal value was $\alpha = 0.1$.
[11] Our evaluation.
[12] Uses cLang8 training data.
[13] Model ensemble.
[14] Uses more than 9M synthetic data samples.
[15] Uses larger language models than roberta-base .

[16] Electra is pretrained on discriminating between real and fake words in context, so its pretraining objective is very similar to the downstream task we solve.

Model	BEA 2019 dev			BEA 2019 test			CoNLL 2014		
	P	R	F0.5	P	R	F0.5	P	R	F0.5
BERT-GEC edits, no base model	62.1	33.9	53.2	80.0	49.1	71.0	70.2	38.0	60.0
GECToR edits, no base model	60.4	34.1	52.5	76.1	52.4	69.8	73.6	34.9	60.2
BERT-GEC edits, pretraining, no base model	68.4	30.4	55.1	82.4	51.1	73.4	71.2	39.4	61.3
GECToR edits, pretraining, no base model	*69.1*	30.9	55.4	82.2	49.7	72.7	72.9	39.1	62.1
GECToR edits, pretraining, combined	68.4	34.5	*57.2*	*82.4*	54.5	*74.7*	*79.1*	38.3	65.2
GECToR, roberta(Omelianchuk et al., 2020)[6]	62.3	**35.6**	54.2	77.1	**55.3**	71.4	72.8	*40.9*	63.0
GECToR, XLNet(Omelianchuk et al., 2020)	66.0	33.8	55.5	79.2	53.9	72.4	77.5	40.2	65.3
GECToR+BIFI(Yasunaga et al., 2021)	NA	NA	NA	79.4	55.0	72.9	78.0	40.6	*65.8*
GECToR edits, cLang8, no base model[11]	**70.2**	32.9	57.2	**82.8**	52.4	74.2	72.6	39.5	63.9
GECToR edits, cLang8, combined[11]	69.3	35.5	**58.2**	82.5	55.1	75.1	**79.6**	36.2	66.0
(Kiyono et al., 2019)[12,13]	NA	NA	NA	74.7	56.7	70.2	73.3	44.2	64.7
(Sun et al., 2021)[13,14]	NA	NA	NA	NA	NA	NA	71.0	**52.8**	66.4
T5-XXL, cLang8 (Rothe et al., 2021)[13,14]	NA	NA	NA	NA	NA	**75.9**	NA	NA	**68.9**

Table 5: Results of different GEC models on three GEC datasets. The first block includes the models trained on BEA train data only, the second one contains the ones that additionally use 9M synthetic samples from (Awasthi et al., 2019), while the last block includes the models that either use ensembles[12], larger Transformer models[14], cLang8 training dataset[11] or more synthetic data[13]. Bold denotes overall **best results** and italic stands for *best results among models of roberta-base size*.

Model	W&I+FCE			BEA 2019 train+finetune		
	P	R	F0.5	P	R	F0.5
Basic	55.5	26.7	46.1(+0.0)	60.4	34.1	52.5(+0.0)
+soft	55.2	30.8	47.6(+1.5)	58.2	35.3	51.6(−0.9)
+contrast	55.1	31.1	47.7(+1.6)	60.9	30.1	50.5(−2.0)
CLS	57.7	22.0	43.5(−2.6)	NA	NA	NA
mean	58.0	27.0	47.2(+1.1)	61.6	31.6	51.8(−0.7)
Electra	60.2	30.1	50.2(+4.1)	60.4	34.1	52.5(+0.0)
Roberta-large	60.8	31.4	51.2(+5.0)	63.5	34.8	54.5(+2.0)

Table 6: Comparison of different architecture modifications on small(W&I+FCE, 60K sentences) and large(BEA2019 train, 1.1M) datasets. The number in brackets is the F0.5 gain over the 'Basic' model. See Appendix E for a complete description.

Threshold	Decoding	Before finetuning			After finetuning		
		P	R	F0.5	P	R	F0.5
0.5	Online	59.2	30.7	49.9	57.1	39.8	52.6
0.6	Online	60.5	29.8	50.2	58.6	38.9	53.2
0.7	Online	63.1	27.7	50.2	60.7	37.9	54.2
0.8	Online	68.8	22.7	48.9	63.1	35.9	54.8
0.9	Online	79.9	10.7	34.8	69.2	30.9	55.4

Table 7: Precision, recall and F0.5 score on BEA 2019 development set with different decision thresholds with/without finetuning. Models are trained on synthetic data and BEA 2019 full train set and finetuned on W&I-LOCNESS train set with GECToR edit generator.

in training or the joint one.

Despite significant increase of edit coverage ('joint' edit generator has recall 63% vs 55% of individual generators) the models show either marginal or even negative change in terms of overall F0.5 score due to lower precision. Finetuning on joint edits data also has little effect which implies the complete training on joint edit set is required. These results also show that GECToR model adapts to edits of other type more easily than the BERT-

		1	2	3	4	5	6	7	8
Offline	Precision	72.9	70.6	69.6	69.5	69.4	69.4	69.4	69.4
	Recall	18.8	25.7	28.0	29.0	29.3	29.5	29.5	29.5
	F0.5 score	46.2	52.4	53.7	54.3	54.5	54.6	54.6	54.6
Online	Precision	72.9	71.0	70.1	69.4	69.2	69.1	69.0	69.0
	Recall	18.8	25.9	30.4	28.8	29.9	30.5	30.9	31.0
	F0.5 score	46.2	52.6	54.5	54.9	55.2	55.3	55.4	55.4
	(F0.5 gain)	(+0.00)	(+0.2)	(+0.8)	(+0.6)	(+0.7)	(+0.7)	(+0.8)	(+0.8)

Table 8: Dependence of model performance from the maximal allowed number of edits. The last row is the difference between online and offline decoding algorithms.

Train	Scorer	Same finetune and test			Same finetune, joint test			Joint finetune and test		
		P	R	F0.5	P	R	F0.5	P	R	F0.5
GECToR	base model	69.1	30.9	55.4	67.6	33.0	55.9(+0.5)	64.8	35.5	55.7(+0.3)
	combined	68.4	34.5	57.2	68.3	34.9	57.3(+0.1)	67.3	36.6	57.6(+0.4)
BERT-GEC	base model	69.1	30.9	55.4	63.4	34.2	54.2(−1.2)	64.2	34.3	54.6(−0.8)

Table 9: Results of models trained on BEA 2019 full train data with BERT-GEC or GECToR edits when tested on BEA 2019 development set with either the same or joint edit generator.

GEC one, showing that it has more perspectives for usage in real-world setting.

5.4 Experiments on Russian

To prove that our approach is not limited to English, we test in on Russian. In contrast to English, it has more complex nominal morphology which extends the space of possible errors even for the rule-based generator. In case of Russian we have much less training data, its characteristics are given in Table 10. Synthetic data is generated using rule-based operations, such as comma / preposition insertion/deletion/replacement or changing the word to another form of the same lexeme[17].

We compare two edit generation models, the first is again the rule-based one[17] and the second is simply the finetuned ruGPT-large[18], their coverage statistics are given in Table 10. We train the scorers[19] on the concatenation of synthetic data and RULEC-GEC train set and then finetune them on real data only. The results are given in Table 11.

We observe that reranking the edits of finetuned ruGPT-large slightly outperforms the edit generator itself. The combined model beats this baseline by a margin of 1.7%. We also note that previous SOTA models had larger size and were trained with significantly more synthetic data. Contrastive to English experiments, scoring the rule-based edits provides even better scores than the model-based ones. We explain this by two reasons: first, the difference between rule-based and model-based edits coverage is smaller for Russian than for English, second, the RULEC-GEC dataset is of much lower quality with a lot of errors uncorrected. Thus it does not contain enough complex edits that cannot be captured by the rules and for which the benefits of model-based generator are more clear.

6 Related work

The task of grammatical error correction has a long history. The main paradigm of recent years is to treat it as low-resource machine translation (Felice et al., 2014; Junczys-Dowmunt et al., 2018) using extensive pretraining on synthetic data (Grundkiewicz et al., 2019). Synthetic data is usually generated using random replacement, deletion, insertion, spelling errors and perturbations (Grundkiewicz et al., 2019; Kiyono et al., 2019; Náplava and Straka, 2019), other approaches include training on Wikipedia edits (Lichtarge et al., 2019) and backtranslation (Kiyono et al., 2019). Another trend is incorporating pretrained Transformer language models either as a part of system architecture (Kaneko et al., 2020) or for the initialization of model weights (Omelianchuk et al., 2020). The extreme case of the latter approach is the "brute force" when one simply uses large encoder-decoder

[17]The full list of operations is given in Appendix A.2

[18]https://huggingface.co/sberbank-ai/ruGPT3large_based_on_gpt2

[19]Since there is no Roberta-base for Russian, we use Roberta-large https://huggingface.co/sberbank-ai/ruRoberta-large.

Dataset	Size		Coverage	
	Sentences	Errors	Rule-based	ruGPT-based
RULEC-GEC train (Rozovskaya and Roth, 2019)	4980	4383	54.4	81.5
RULEC-GEC dev (Rozovskaya and Roth, 2019)	2500	2182	55.5	59.3
RULEC-GEC test (Rozovskaya and Roth, 2019)	5000	5301	46.4	54.3
Synthetic data	213965	187122	78.0	95.8

Table 10: Data used for experiments on Russian GEC and coverage of edit generators on this data.

Model	Training data	P	R	F0.5
Transformer (Náplava and Straka, 2019)	10M synthetic + RULEC-GEC train + dev	63.3	27.5	50.2
mT5-XXL (Rothe et al., 2021)	mC4 synthetic + RULEC-GEC train	NA	NA	51.6
ruGPT-large finetune (strong baseline)	200K synthetic + RULEC-GEC train	65.7	27.4	51.3
rule-based edits	200K synthetic + RULEC-GEC train	69.4	25.9	51.9
ruGPT-large edits, no base model	200K synthetic + RULEC-GEC train	68.2	27.1	51.6
ruGPT-large edits, combined	200K synthetic + RULEC-GEC train	74.4	24.6	53.0

Table 11: Results for Russian on RULEC-GEC dataset. The upper block contains the baselines, current work results are in the lower one.

Transformer that potentially is able to solve any text-to-text task (Rothe et al., 2021).

Another paradigm in GEC is to reduce grammar correction to sequence labeling (Omelianchuk et al., 2020). However, it requires constructing a linguistically meaningful set of tags that could be hard to design for languages with complex morphology. Our work mainly follows the third approach that considers GEC as two-stage process including edit generation as the first stage and their ranking or classification as the second. Edits were usually generated by manually written rules and their scoring was performed by linear classifiers (Rozovskaya et al., 2014) or later by a pretrained language model (Alikaniotis and Raheja, 2019). A recent work of Yasunaga et al. (2021) generates edits using separate sequence-to-sequence Transformer and then filters them using a language model.

Our approach can be seen as a special case of **reranking**. Feature-based reranking was common in statistical machine translation before the advent of neural networks(Och et al., 2004), in grammatical error correction it is mostly performed by a language model R2L scorer (Grundkiewicz et al., 2019). However, recent studies on machine translation (Lee et al., 2021) and summarization (Liu and Liu, 2021) benefit from transformer-based rescoring. Our work is partially inspired by theirs, the key difference is that we use classification loss instead of ranking and rerank individual edits, not complete sentences. As far as we know, the only example of trainable reranking for grammatical error correction is Liu et al. (2021), but it uses a more complex architecture and focuses more on error detection than correction.

7 Conclusion

We have developed an edit classifier for grammatical error correction that achieves state-of-the-art performance even without using the edit generator scores and improves over SOTA models of comparable size after combination with basic model. Our scorer can be combined with different types of architectures, both encoder-decoder and sequence labelers, showing similar performance. The same approach also beats state-of-the-art results on low-resource grammatical error correction for Russian, which is morphologically more complex than English.

We show that additional losses are not helpful yet, however, using better and larger Transformer models looks promising. Since our method works independently of the edit generator, it may be applied in setups where one has to correct errors of a particular type (e.g., verb tense), such as second language learning. In the future work we plan to address this question in more details and test the applicability of our approach on additional languages, such as German or Czech. Last but not the least, the main idea of ranking individual edits can be applied not only to GEC, but to any task where the concept of elementary edit has meaning, for example, machine translation post-editing.

References

Dimitris Alikaniotis and Vipul Raheja. 2019. The unreasonable effectiveness of transformer language models in grammatical error correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 127–133, Florence, Italy. Association for Computational Linguistics.

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.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

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, Atlanta, Georgia. 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.

Mariano Felice, Zheng Yuan, Øistein E. Andersen, Helen Yannakoudakis, and Ekaterina Kochmar. 2014. Grammatical error correction using hybrid systems and type filtering. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 15–24, Baltimore, Maryland. 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.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Shubha Guha, and Kenneth Heafield. 2018. Approaching neural grammatical error correction as a low-resource machine translation task. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 595–606, New Orleans, Louisiana. Association for Computational Linguistics.

Masahiro Kaneko, Masato Mita, Shun Kiyono, Jun Suzuki, and Kentaro Inui. 2020. Encoder-decoder models can benefit from pre-trained masked language models in grammatical error correction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4248–4254, Online. 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.

Ann Lee, Michael Auli, and Marc'Aurelio Ranzato. 2021. Discriminative reranking for neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7250–7264, Online. 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.

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*.

Yixin Liu and Pengfei Liu. 2021. SimCLS: A simple framework for contrastive learning of abstractive summarization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 1065–1072, Online. Association for Computational Linguistics.

Zhenghao Liu, Xiaoyuan Yi, Maosong Sun, Liner Yang, and Tat-Seng Chua. 2021. Neural quality estimation with multiple hypotheses for grammatical error correction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5441–5452, 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, 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, Baltimore, Maryland. Association for Computational Linguistics.

Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. 2004. A smorgasbord of features for statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 161–168, Boston, Massachusetts, USA. 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.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners.

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

Alla Rozovskaya, Kai-Wei Chang, Mark Sammons, Dan Roth, and Nizar Habash. 2014. The Illinois-Columbia system in the CoNLL-2014 shared task. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 34–42, Baltimore, Maryland. Association for Computational Linguistics.

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.

Xin Sun, Tao Ge, Furu Wei, and Houfeng Wang. 2021. Instantaneous grammatical error correction with shallow aggressive decoding. *arXiv preprint arXiv:2106.04970*.

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 (Volume 2: Short Papers)*, pages 198–202, Jeju Island, Korea. 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.

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.

Michihiro Yasunaga, Jure Leskovec, and Percy Liang. 2021. Lm-critic: Language models for unsupervised grammatical error correction. *arXiv preprint arXiv:2109.06822*.

Kyra Yee, Yann Dauphin, and Michael Auli. 2019. Simple and effective noisy channel modeling for neural machine translation. 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 5696–5701, Hong Kong, China. Association for Computational Linguistics.

A Rule-based transformations used for edit generation

A.1 English

Rule-based edit generator includes the following operations:

- Comma insertion and deletion.
- Preposition insertion, deletion and substitution. Insertion is allowed only before the first token of a noun group.
- Determiner insertion, deletion and substitution. Insertion is allowed only before the first token of a noun group.
- *to* insertion before infinitives.
- Spelling correction for OOV words using Hunspell[20].
- Substitution a word with all its inflected forms, inflection is performed using Lemminflect[21].
- Capitalization switching.
- Replacement of comma by period and capitalizing the subsequent word (*I have a dog, it is cute.* → *I have a dog. It is cute.*).

The rules take as input sentence dependency trees, parsing is done using Spacy[22].

A.2 Russian

Rule-based edit generator for Russian includes the following operations:

- Comma insertion and deletion.
- Preposition insertion, deletion and substitution. Insertion is allowed only before the first token of a noun group.
- Conjunction substitution.
- Spelling correction for OOV words using Hunspell[23].
- Joining of consecutive words using Hunspell (e.g. *ne bol'shoj* 'no+big' ↦ *nebol'shoj* 'small').
- Substitution a word with all its inflected forms, inflection is performed using PyMorphy[24].
- Joint noun group inflection (e.g. *bol'shoj dom* 'large house' ↦ *bol'shikh domov* 'large+GEN+PL houses+GEN')
- Capitalization switching.
- Switching the order of consecutive words.

The rules take as input sentence dependency trees, parsing is done using DeepPavlov[25].

B Data sources

English

- W&I-LOCNESS train, dev and test https://www.cl.cam.ac.uk/research/nl/bea2019st/data/wi+locness_v2.1.bea19.tar.gz.
- FCE https://www.cl.cam.ac.uk/research/nl/bea2019st/data/fce_v2.1.bea19.tar.gz.
- NUCLE https://sterling8.d2.comp.nus.edu.sg/nucle_download/nucle.php.
- Lang8 https://docs.google.com/forms/d/e/1FAIpQLSflRX3h5QYxegivjHN7SJ194OxZ4XN_7Rt0cNpR2YbmNV-7Ag/viewform.
- CLang8 https://github.com/google-research-datasets/clang8.
- Conll14 https://www.comp.nus.edu.sg/~nlp/conll14st/conll14st-test-data.tar.gz.
- PIE synthetic data https://drive.google.com/open?id=1bl5reJ-XhPEfEaPjvO45M7w0yN-0XGOA.

Russian

- RULEC-GEC https://github.com/arozovskaya/RULEC-GEC.
- Synthetic data: not available yet.

[20] https://github.com/MSeal/cython_hunspell
[21] https://github.com/bjascob/LemmInflect/
[22] spacy.io
[23] https://github.com/MSeal/cython_hunspell

Source Correct	*Until the dawn all of them go out , so they sacred until they find a refuge .* *By dawn all of them had got out , so they sacred until they found a refuge .*		
Edit	Target	Gain	Label
	Rule-based edit generator		
(1, 2, the) → _	Until dawn all of them go out , so they sacred until they find a refuge .	1.33	True
(11, 11, _) → are	Until the dawn all of them go out , so they are sacred until they find a refuge .	0.95	False
(3, 3, _) → ,	Until the dawn , all of them go out , so they sacred until they find a refuge .	0.95	False
(11, 11, _) → were	Until the dawn all of them go out , so they were sacred until they find a refuge .	−1.73	False
(-1, -1, _) → _	Until the dawn all of them go out , so they sacred until they find a refuge .	0.00	False
	BERT-GEC edit generator		
(11, 11, _) → are	Until the dawn all of them go out , so they are sacred until they find a refuge .	0.06	False
(1, 2, the) → _	Until dawn all of them go out , so they sacred until they find a refuge .	−0.06	True
(11, 11, _) → stay	Until the dawn all of them go out , so they stay sacred until they find a refuge .	−0.24	False
(0, 2, Until the) → Before	Before dawn all of them go out , so they sacred until they find a refuge .	−0.79	False
(12, 12, _) → themselves	Until the dawn all of them go out , so they sacred themselves until they find a refuge .	−2.95	False
(0, 2, Until the) → Up until	Up until the dawn all of them go out , so they sacred until they find a refuge .	−2.99	False
(-1, -1, _) → _	Until the dawn all of them go out , so they sacred until they find a refuge .	0.00	False
	GECToR edit generator		
(0, 1, Until) → In	In the dawn all of them go out , so they sacred until they find a refuge .	5.35	False
(1, 2, the) → _	Until dawn all of them go out , so they sacred until they find a refuge .	4.59	True
(0, 1, Until) → _	The dawn all of them go out , so they sacred until they find a refuge .	4.01	False
(0, 1, Until) → As	As the dawn all of them go out , so they sacred until they find a refuge .	2.86	False
(12, 13, until) → _	Until the dawn all of them go out , so they sacred they find a refuge .	1.21	False
(15, 16, a) → _	Until the dawn all of them go out , so they sacred until they find refuge .	1.01	False
(7, 8, out) → _	Until the dawn all of them go , so they sacred until they find a refuge .	0.72	False
(0, 1, Until) → By	By the dawn all of them go out , so they sacred until they find a refuge .	0.71	True
(3, 3, _) → ,	Until the dawn , all of them go out , so they sacred until they find a refuge .	0.65	False
(8, 10, ,_so) → . So	Until the dawn all of them go out . So they sacred until they find a refuge .	0.48	False
(6, 7, go) → went	Until the dawn all of them went out , so they sacred until they find a refuge .	−0.55	False
(8, 9, ‘,’) → _	Until the dawn all of them go out so they sacred until they find a refuge .	−0.81	False
(12, 12, _) → ,	Until the dawn all of them go out , so they sacred , until they find a refuge .	−1.18	False
(14, 15, find) → found	Until the dawn all of them go out , so they sacred until they found a refuge .	−3.76	True
(-1, -1, _) → _	Until the dawn all of them go out , so they sacred until they find a refuge .	0.00	False

Table 12: Output of different edit generators for the sentence *Until the dawn all of them go out , so they sacred until they find a refuge .* Gain column contains the first stage score.

C Examples of elementary edits

D Model hyperparameters

In Table 13 we summarize the information required to replicate the training procedure. The exact values may vary slightly. In all the experiments we did finetuning for 5 epochs, but generally later checkpoints demonstrated severe overfitting.

Parameter	Value
Batch size	3500 tokens
Optimizer	wAdam
Learning rate	$1e-5$
Weight decay	0.01
Warmup (base models)	0
Warmup (large models)	2000
Pretraining epochs	1
Lang8 training epochs	3
Finetuning epochs	2
Joint training epochs (Russian)	1
Finetuning epochs (Russian)	2

Table 13: Training hyperparameters.

E Ablation studies

The choice of model architecture and training parameters may seem arbitrary. Therefore in this section we study other possible variants of modern architecture. The architecture used in main experiments has the following key components:

1. The model is trained with cross-entropy classification loss without any additional objectives.
2. The loss is normalized separately for positive and negative instances.
3. The encoding of the first token in the output span is used as edit representation.
4. The classification module contains a single hidden layer.
5. Except for the classification module, no additional layers are added on the top of main Transformer encoder.

[24] https://github.com/kmike/pymorphy2/
[25] http://docs.deeppavlov.ai/en/0.14.1/

6. Roberta-base is used as the encoder.

We test the following architecture modifications:

1. Adding an additional ranking objective. We do it adding standard margin loss:

$$L(x^+, x^-) = \max\left(g(x^-) - g(x^+) + \theta, 0\right),$$

$$L = L_{CE} + \alpha \frac{\sum_{(x^+, x^-) \in P} L(x^+, x^-)}{|P|}.$$

   Here g is the logit of positive class before sigmoid, P is the set of contrastive pairs of batch elements, θ is a margin hyperparameter and α is the additional loss weight [26]. We investigate 3 variants of defining P:

   - All pairs of positive and negative instances (+*soft*),
   - Only pairs of positive and negative instances whose spans intersect(+*hard*),
   - All pairs of the form $(\mathbf{e}^+, \mathbf{e}_0)$ and $(\mathbf{e}^0, \mathbf{e}^-)$, where $\mathbf{e}^+, \mathbf{e}^-$ and $\mathbf{e}_0$ are positive, negative and "do nothing" edits, respectively(+*contrast*).

2. Removal of class normalization (*no_norm*).

3. Using the CLS token (*cls*), mean representation of output span (*mean*) and concatenation of output and source span (*origin*) as edit encodings.

4. Adding one more hidden layer in the classification block (*'2 layers'*).

5. Adding an additional Transformer layer between all the edit representations for the same sentence (+*attention*). That allows to potentially use information from other hypotheses.

6. Use other Transformer variants, in particular Electra(Clark et al., 2020) and Roberta-large(Liu et al., 2019).

We run all ablation experiments on the concatenation of W&I+LOCNESS train and FCE datasets using GECToR edit generator, results are given in Table 14. For all the models we select the best performing checkpoint and threshold according to the F0.5 score and perform online decoding. For those models that improve over the basic one on the small dataset, we run additional testing on full BEA train data without finetuning.

We observe that additional losses that are helpful in low-resource setting even decrease performance for larger data. The more promising approach is to use either more suitable to text correction (Electra) or larger (Roberta-large) language models. However, with more data the gap between them and the roberta-base model also becomes smaller.

[26] We set $\alpha = 0.25, \theta = 2.0$.

Model	W&I+FCE			BEA 2019 train+finetune		
	P	R	F0.5	P	R	F0.5
Basic	55.5	26.7	46.1(+0.0)	60.4	34.1	52.5(+0.0)
+hard	55.1	26.4	45.8(−0.3)	NA	NA	NA
+soft	55.2	30.8	47.6(+1.5)	58.2	35.3	51.6(−0.9)
+contrast	55.1	31.1	47.7(+1.6)	60.9	30.1	50.5(−2.0)
no_norm	55.8	27.4	46.2(+0.1)	NA	NA	NA
CLS	57.7	22.0	43.5(−2.6)	NA	NA	NA
+mean	58.0	27.0	47.2(+1.1)	61.6	31.6	51.8(−0.7)
+origin	57.4	26.2	46.4(+0.3)	NA	NA	NA
2layers	55.6	27.7	46.3(+0.2)	NA	NA	NA
+attention	52.8	31.4	46.4(+0.3)	NA	NA	NA
Electra	60.2	30.1	50.2(+4.1)	60.4	34.1	52.5(+0.0)
Roberta-large	60.8	31.4	51.2(+5.0)	63.5	34.8	54.5(+2.0)

Table 14: Comparison of different architecture modifications, the number in brackets is the difference with the 'Basic' model. See the list above for a complete description.