DASC 7600 Data Science Project

An improved Grammatical Error Correction Editor based on NLP Techniques

group members:
Xie Diangu
Zheng Binfan
Gao Peixin
Huang Haohan

Content

Abstract

The Grammar Error Correction (GEC) system can help non-native language learners have a better understanding of target language, and corresponding researches has a rapid development period currently. In this paper, we implement a composite GEC system which achieves excellent result of F0.5 and GLEU. We use transformer model with copy-augmented architecture as our baseline, which is proved effective in GEC task. Source sentences of text will be checked by a combined spellchecker (pyspellchecker + language model) first, and then be corrected by the main model (Transformer with copy-architecture and re-ranker). The fixed sentences from this main model will be corrected again by rule-based method. Additionally, we also build an interactive interface which can allow users to use our GEC system to correct their sentence or paragraph intuitively.

key words: copy-augmented, transformer, spellchecker, error generation, language model, re-ranker, rule-based method

1.Introduction

1.1 Background

Because of the in-depth globalization, learning foreign language becomes a normal phenomenon in the current world. Texts written by non-native learners are usually full of errors. English, the most important and common language, also has this problem of its learners, so improving the power of error correction tools makes much sense. Grammatical error correction (GEC) is a main stream task of natural language processing (NLP) whose target is creating a system which can automatically corrects errors in texts, especially those written by non-native learners. GEC can help foreign learners correct their sentence by themselves and they can acquire right usage timely. There are a lot of kinds of errors, containing grammatical errors, collocation errors, spelling errors, wrong idiom words and so on. GEC task attempts to correct the errors by algorithms and machines automatically instead of correcting those grammar errors one by one by human annotators. Our project mainly focuses on GEC for English.

1.2 Major Approaches

GEC task can be detected in different aspects. Firstly, we can deal with this problem based on grammars, since grammar mistakes may be the most common problem. Summarizing all situation of grammar errors and providing amendment functions to transform them may be useful. Secondly, comparing wrong sentences with errors and the right version and extracting statistical features to describe these errors can connect this task to statistical region, so we can transplant statistical models, which have many great results before, to this task. Thirdly, a pair of wrong sentences and corresponding right sentences is just like a pair of sentences expressed by one language and corresponding sentences expressed by another language, so powerful neural models of machine translation may also get excellent results in GEC task. According to these three aspects, there are three major approaches for automatic GEC, containing rule-based method, Statistical Machine Translation (written as SMT) and Neural Machine Translation (written as NMT).

Rule-based method is the original method of GEC. At the very beginning, people should analyze the syntactics structure and summary common mistakes so that machine can auto-correct errors based on stated conditions. Daniel Naber developed an open source rule-based checker of style and grammar (Daniel Naber, 2003), which can detect errors by rules summarized before. Specifically, each sentence anis divided into rules, and any word of the input sentence can get its own part-of-speech tag. Then by comparing the input with the pre-defined rules, the error will be detected if a rule is matched. This model is simple while developers should point out all wrong conditions before correction, which seems impossible because strange and new errors are generated all the time. After realizing this shortcoming, automated feedback of rule on text is developed. Mariano Felice (Felice, 2014) uses rules extracted from the Cambridge Learner Corpus (CLC) (Nicholls, 2003) to solve the problem above. Based on n-grams, these rules are noted as wrong at lowest five times and at least 90% of the times they occur to ensure high precision. Additionally, many cases are checked by means of rules extracted from a machine-readable dictionary. This rule-based system can generate a file which produces the suggested corrections list. However, this method still cannot derive complex errors of texts, e.g., spelling error out of the original texts, misusing of similar semantic words and so on.

SMT is still a popular method of GEC (Alla Rozovskaya and Dan Roth. 2016； Shamil

Chollampatt and Hwee Tou Ng, 2017), containing three diverse levels of tokens, i.e., word-based, phrase-based and syntax-based. The first two are more popular. As for word-based SMT of GEC, it uses the statistical property of word-matching to correct errors. The earliest research (Brown, 1993) is corresponding to machine translation, which gives the definition of word-by-word alignment between sentences pairs. It also states a series of five statistical models to translate sentences, and parameter estimating algorithm is also given. This model returns probability to any possible word-by-word alignments if a pair of sentences is given. As for the phrase-based, it uses the statistical property of phrase rather than single word. A few years before 2008, phrase-based SMT systems got advanced and excellent performance, while the shortcoming that SMT is weak when word order changes were still existing. Phrase reordering models typically lack the power to solve long-distance re-orderings problem with syntax-based systems. Michel Galley (Galley, 2008) presents a newly hierarchical phrase reordering model which improves non-local re-orderings successfully by combining a phrase-based system. Lastly, for the third level, the syntax-based SMT, these models want to capture the information of syntax to solve the problems which are difficult to be solve in the first two. Trees' thought is frequently used in this branch (Haitao Mi, 2008; Min Zhang, 2008; Haitao Mi, 2008; Yang Liu, 2009). Models make use of different information based on linguistic theory, containing using none information about grammar, using the information of the target sentence's grammar or using information of the source sentence's grammar. As for machine translation task, the challenge is that progress will be slower on translation model which can learn the relation between the grammars of the source language and the target language. David Chiang (Chiang, 2010) makes a summary about this challenge and points out a simple method to solve this problem. Although the main achievement of SMT are derived from machine translation task, while it can be transplanted to GEC task easily, since the difference is just that the original sentence is a wrong text of GEC task or a text by another language text of machine translate task. In fact, SMT may be used better in GEC because some troublesome problems faced in machine learning task are not problems in GEC. For example, just like what David Chiang focuses on, the grammar difference between the source and target sentence, is nonexistent in GEC since there is just one language in this task. GEC based on SMT does not need philologer and always has higher accuracy (before 2017), but it cannot solve the problem of homophony and the performance is easy to be influenced when the structure of sentence is wrong (although some models are created to solve these problems, they still exist). Additionally, this method cannot perform well when it is used to correct discontinuous phrase and multi-phrases.

GEC based on NMT is the most important method in current years. This model is based on Sequence-to-Sequence model, using RNN as encoder and decoder to transfer wrong sentence to corrected sentence. It uses the information (including predicted words and source sentence) before to predict the next word. This model can output more fluent output which may be accepted more simply by human evaluators, and can capture higher features. More importantly, it does not need philologer and can solve homophony. However, we also cannot ignore the fact that there are not many large-scale datasets and training process is always slow. Our model bases mainly on NMT.

As mentioned above, different approaches have different shortcomings and advantages, and they are complementary in some ways. For example, rule-base system may ignore some complex semantic errors, while NMT can learn it well by deriving corresponding information from the training dataset. Based on this natural thought, some hybrid models are created. Roman Grundkiewicz (Roman Grundkiewicz, 2018) build a GEC system by combining SMT and NMT, which can preserve the accuracy of SMT output and can also generate more fluent sentences just like NMT. Mariano Felice (Felice, 2014) also achieves some state-of-the-art results at that point by combining rule-based method and SMT system which is augmented by a large language model. Shamil Chollampatt (Chollampatt, 2017) develops a GEC system mainly based on SMT approach, which uses tuning and task-specific features. In addition, it is enhanced by the power of neural network joint models. In addition, this model also incorporates a character-level component of SMT to correct misspelled words that the elementary SMT-based system cannot find. This hybrid model can solve the shortcoming of SMT-based system, i.e., being weak in generalizing beyond patterns from training dataset and granularity below the word level. Furthermore, it also gets an improvement of F0.5 over the best previous before. Currently, some systems which are even more complicated are created. Zheng Yuan (Zheng Yuan, 2019) presents a system pipeline which uses error detection model and correction models. Specifically, when a text is inputted, it is firstly corrected by two different neural machine translation systems which are complementary to each other. One of these systems utilizes convolutional networks, which also combines the thought of multi-task learning. Another one utilizes a Transformer-based system. Moreover, these two machine translation systems return the n-best lists. Then, these lists are combined and get scores by a finite state transducer (FST). Then the output of FST will be inputted to the re-rankers who uses lots of features and returns the final answer. This complex general system achieves 66.75% F0.5 of the BEA 2019 shared task.

1.3 Summary of NMT

Our model is based mainly on NMT method. GEC based on NMT has a develops rapidly these years(Zheng Yuan and Ted Briscoe.2016; Schmaltz et al., 2017; Ji et al., 2017; Grundkiewicz and Junczys-Dowmunt, 2018). There are several main motivations of this process, containing special mechanism transferred from machine translation task, error generation and pre-trained models based on the Transformer architecture**.**

Some mechanisms and tricks are prove to be useful in neural machine translation task (MT). Since best results of GEC are always gotten from models based on Transformer architecture which is similar to machine translation task and the basic thought is seq2seq which is same as MT, a natural thought is that we can migrate mechanisms and tricks which are efficient in MT for GEC. Re-ranking is a common thought in MT(Graham Neubig, 2015). Additionally, Copying mechanism is also an effective method in some NLP tasks, such as text summarization (Jiatao Gu and Zhengdong Lu, 2016) and semantic parsing (Robin Jia and Percy Liang, 2016). Copying mechanism can store more original information in output sentence from input sentence. Compared to MT, this mechanism looks more suitable for GEC since the difference between input and output sentence in GEC is always much smaller than that in MT. A copy-augmented architecture is proposed and proves to be effective in GEC (Wei Zhao, 2019). By adding copying distribution and copying scores, the information of the unchanged words from the input sentence and the output sentence and fixed words can get a balance by a balancing factor.

Error generation plays a vital role in the quick development of GEC. There are two mainly thoughts of error generation, i.e., generating sentence with errors directly based on errors extracted from corpus or changing right sentence to wrong sentence by some rules. Phrases directly extracted from training dataset of GEC perform well in BEA 2019 Shared Task on GEC (Maggie Liu, 2019), and appending error+context phrase to the original GEC dataset proves that it can get higher precision. Moreover, choosing the way how the pseudo error should be generated and used is also be detected by experiment methods (Shun Kiyono, 2019). Many choices are detected by extensive experiments and an excellent performance is gotten without any modification of model architecture.

As for the pre-trained models based on the Transformer architecture, the classical transformer architecture is changed or fine-tuning to adapt to scholar's requirements. Some pre-trained models are developed such as GPT (Radfor et al., 2018) and multi-headed architecture based on BERT (Julia Shaptala, 2019). Radfor gets large gains on several NLP tasks, containing document classification, semantic similarity assessment, question answering, and textual entailment, which is realized mainly by generative pre-training of using language to deal with different unlabeled datasets, and also by one-to-one fine-tuning on each specific task. As for Shaptala's research about GEC, BERT model is used to create encoded representation and some other enhancements, i.e., "Heads". "Heads" are some fully connected networks which can detect errors and get recommendation from these networks on handling the highlighted sentence. This model increases system productivity and reduces the necessary process time when it keeps high accuracy.

2.Metrics & Dataset

2.1 F0.5 Metric

2.1.1 F-beta Score

In machine learning, it is necessary to evaluate the training model in order to judge whether the algorithm is good or not. Precision and recall are used to measure the performance of a model. Precision refers to the proportion of true positive samples among all samples judged positive by the model, while Recall refers to the proportion of true positive samples among all samples which are actually positive.

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

These two indicators are usually contradictory and restrict each other in large-scale data sets, so comprehensive consideration is needed. The most common method is F-measure, which is the weighted harmonic average of Precision and Recall. The beta in the formula is used to balance the weight of Precision and Recall in the F-score calculation. If the beta is 1, Precision is just as important as Recall. If the beta is less than 1, Precision is more significant than Recall. If the beta is greater than 1, then Recall is more significant than Precision. Precision is generally considered more important than recall in dealing with Grammatical Error Correction and an F 0.5 score is used when giving twice as much weight to precision.

$$F - Score = (1 + \beta^2) * \frac{Precision * Recall}{(\beta^2 * Precision) + Recall}$$

2.1.2 MaxMatch Algorithm

The precision and recall are calculated by M^2 (MaxMatch) algorithm which is proposed in (Dahlmeier, Daniel & Ng, Hwee., 2012). This algorithm computes the highest overlap between the systems edits which is made by a GEC system and the so called golden edits which is made by human annotators. There are 4 basic edit forms: inserting, substituting, deleting a token or just leave the token unchanged. Under the background of GEC task, the similar modification for an error can have different forms of edits, which will confuse our evaluation method. One way of getting rid of this weakness is using n-gram based evaluator which we will introduced in the next section. Another way is to pre-aligned the edits to golden edits using algorithms like M^2. For example, if someone just say ‘apple’ in a sentence, which is a grammar error, his actual intention is ‘an apple’ or ‘apples’. If human annotator marks this error with a substitution edit (apple → an apple), and the GEC system marks this error with a insertion edit (ϵ→an), these two edits are the same correction of different forms. If the evaluation method just checks how many GEC edits match with reference edits directly, such kind of modification will be count as a wrong correction which doesn’t make any sense. M^2 is an algorithm to adjust a set of edits to make it matches with the set of corresponding reference edits as many as possible. An edit lattice is constructed for transforming the source sentence to the candidate sentence raised by GEC system and showing all possible edit forms on its edges. Very small cost is assigned to the edges of edit corresponding to the golden edits. And If we find the shortest path in the edit lattice, this is the maximum overlap with golden edits. In the above example, edit (ϵ→an) will be adjust to (apple → an apple) through edit combination. Denoting the adjusted system edits as $\{e_1, \dots, e_n\}$ and the golden edits as $\{g_1, \dots, g_n\}$ for n sentences in a corpus, we can calculate the precision and recall using the following formula.

$$P = \frac{\sum_{i=1}^{n} |e_i \cap g_i|}{\sum_{i}^{n} |e_i|}$$
$$R = \frac{\sum_{i=1}^{n} |e_i \cap g_i|}{\sum_{i=1}^{n} |g_i|}$$

Hence, we can calculate the F-beta score of a GEC system by precision and recall.

2.2 GLEU Metric

2.2.1 Shortcoming of F0.5

GLEU (so called Generalized Language Evaluation Understanding metric) (Courtney Napoles et al, 2015) is another common-used standard to evaluate a GEC system's performance. It was developed to overcome some disadvantages of F0.5 metric. There could be two different aspects of evaluating the performance given a GEC system. On the one hand, the output sentence corrected by the system shouldn't be too different from the original wrong sentence, on the other hand, the output sentence should also be such a sentence that is native sounding and idiomatic instead of a non-fluent machine generated sentence. The issue of F0.5 is that it only calculates the F-score by counting of minimal phrase-level edits to convert the original sentence to the hypothesis sentence from GEC system using M^2 (MaxMatch), so it prefers to use as few changes as possible to make a wrong sentence grammatical correct. Sometimes such 'minimal edits' strategy-based metric will lead to a very rigid and stiff correction which is not accepted by users in real applications. Furthermore, if the system adds or deletes a long phase in one place, this modification will be regard as 'one edit' together to calculate F0.5 and this does not make any sense. Besides, for a specific error in source sentence, the F0.5 metric cannot detect the difference if the GEC system ignores the error and just leaves it or the GEC system makes a wrong modification on this error. Hence F0.5 itself is not the golden standard we want in real life GEC situation. That's why we need another 'fluency edits'-based strategy of metric as supplement in our GEC system – the GLEU.

2.2.2 Motivation

Conventionally the calculation of GLEU is only using JFLEG dataset that we are going to introduce in section 2.4.2. Because the form of GEC task can be regarded as a machine translation task of translating the wrong sentence with grammar errors to a grammatically correct sentence, it is natural to borrow machine translation metric BLEU score (Kishore Papineni et al, 2002). BLEU is calculated based on n-grams model to calculate the precision score between source sentence and target sentence, then multiple with a penalty term to control the recall. However, it cannot be used without any modification due to the underlying difference between machine translation task and GEC task. In MT, any of the n-gram in the source sentence should not be considered in metric because they are definitely wrong, hence a good translation should be the one that is very like the target sentences. So, BLEU doesn't include any information from the source sentence. But in GEC task the situation is different. Usually only a few parts of the source sentence will be changed during correction while the other parts of the sentence shouldn't be changed. Only considering the target sentence isn't enough and the precision depends on both the changed part and the unchanged part of the sentence. GLEU is a metric that gives more weight in the correctly changed parts of the source in calculating precision.

2.2.3 Calculation

Given a bag of n-grams denoted by B, we can count the frequency of a specific n-gram by using the following formula.

$$Count_B(n-gram) = \sum_{n-gram' \in B} d(n-gram, n-gram')$$

$$d(n-gram, n-gram') = \begin{cases} 1 & if\ n-gram = n-gram' \\ 0 & otherwise \end{cases}$$

Given the source sentence denoted by S, the candidate output of the GEC system C, and the reference target sentence R, the formula of precision for GLEU is as follows.

$$p_n' = \frac{\sum_{n-gram \in C} Count_{R\setminus S}(n-gram) - \lambda\left(Count_{S\setminus R}(n-gram)\right) + Count_R(n-gram)}{\sum_{n-gram' \in C'} Count_S(n-gram') + \sum_{n-gram \in R\setminus S} Count_{R\setminus S}(n-gram)}$$

Compared with BLEU (Kishore Papineni et al, 2002), the precision calculate process is different because of the natural purpose of GEC task. On the one hand, for those n-grams which appears only in reference sentence but not in source sentence (denote by R\S), usually they are mistakes that need to be corrected by GEC system. If there are some n-grams in candidate sentence that matches with this set of grams (R\S), these matches will be counted twice in the above precision formula, to give reward for the GEC system which correctly recognize these mistakes and repaired them. On the other hand, if the candidate n-grams matches with those n-grams which appears only in source sentence but not in target sentence (denote by S\R), usually they are likely to be mistakes to be corrected but the GEC system cannot detect, and the precision will be penalized when these types of matches appear. The weight λ determines the level of penalty.

Hence, GLEU can be calculate by

$$GLEU(C,R,S) = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n'\right)$$

which is a weighted exponential average of 1-gram precision, 2-gram precision, ⋯, until N-gram precision, where

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-c/r)} & if\ c \leq r \end{cases}$$

is the same brevity penalty as BLEU.

In our experiment, we follow the parameter setting of the original paper (Courtney Napoles et al, 2015), which is also the standard parameters of this metric when comparing different GEC systems. We use N=4, $w_n = \frac{1}{N}$ and $\lambda = 0$. Too long grams (longer than 5 words) will not be counted and we assume different 1-gram to 4-gram are equality important.

2.2.4 Performance

Based on the paper (Courtney Napoles et al, 2015), GLEU metric is closer to human feeling of the severity of grammatical mistake than F0.5. And by combining with n-gram model, it rewards the edits that is both fluent and grammatical correct. By saying 'not fluency' we mean that in calculation of F0.5 score, all types of error correction forms are equally important to this metric, but some of the correction form is actually more preferred by human. For example, if we want to insert a missing token, F0.5 will equally reward the system of inserting by a comma or a verb, but human will prefer using verb instead of comma in most cases. After ranking multiple outputs of different GEC systems by F0.5, GLEU and human volunteers separately, GLEU result have a bigger correlation coefficient with human ranking.

2.3 Training Datasets

One of the biggest problems faced by people who study grammatical error correction problem is the lack of large enough, well annotated corpus of good quality. Some datasets such as CLC (Cambridge Learner Corpus) (Nicholls. 2003) that contains more than 200000 texts is a quite large and well-annotated corpus but cannot be used freely. In our study we investigate different sources of data that is available to us for both model training and evaluation. We combined different datasets together to make our model meet as many types of mistakes as we can during training. Furthermore, we also tried to use error generation model to make our dataset even larger. We studied different combined methods of the datasets and choose the best performance in our final model. Here we list all the train datasets that we used. The datasets are collected mainly from BEA (Building Educational Applications) 2019 Shared Task: Grammatical Error Correction (Christopher Bryant, 2019).

2.3.1 NUCLE

NUCLE corpus (NUS Corpus of Learner English) (Daniel Dahlmeier et.al, 2013) is a large, commonly used dataset for GEC model. It was created between August and December 2009. The component of this dataset is mainly essays written by Asian undergraduate students of similar academic level taking English courses at CELC (Centre for English Language

Communication at National University of Singapore). These essays were annotated by their English instructors. Some but not all sentences are double annotated. There are 59,871 sentences from 1414 documents with totally 46,597 error annotations. In every hundred words there exists 3.82 errors averagely. Errors are divided into 13 categories of 27 different error codes. Compared to other datasets, NUCLE are written by students with relatively good English proficiency and shows a sparse error distribution. The quality of this dataset is very good but the size is not very big.

2.3.2 Lang-8

Lang-8 corpus (Toshikazu Tajiri, 2012) is currently the largest corpus for GEC task that we can obtain. Each sentence in lang-8 dataset could also have multiple ways of corrections from different annotators. The dataset is collected from lang-8 website (http://lang-8.com/) which is used to help foreign language learners to improve their proficiency. People post their essays on this website and some native speakers can help them correct their articles. We only use the English subsection in our study. The total number of sentences in this corpus is 1069549. The users of lang8 come from different countries of the world, so their English abilities are also irregular. The types of grammar errors are much more complicated than NUCLE. Besides, the annotators are just native speakers instead of professional teacher, hence the quality of annotations of lang8 isn't as good as the annotations of NUCLE. But it is still somewhat-clean and very useful in our training process.

2.3.3 FCE

FCE corpus (The First Certificate in English) (Helen Yannakoudakis, 2011) is a subset of the CLC dataset we mentioned above. This is a very small dataset, only containing 1244 texts from English learners' answers of First Certificate in English (FCE) exam. Each article is not as long as the articles in NUCLE dataset, around 200-400 words. Compared to NUCLE dataset, the writers of FCE has a lower level of mastering English and the density of error is higher.

2.3.4 W&I+LOCNESS

In 2019 BEA shared task (Christopher Bryant, 2019), two new datasets of Grammar Error Correction have been released. The first one is W&I (the Cambridge English Write and Improve) corpus. W&I is another online platform just like lang8 to help English learners. Each of the sentence from this corpus has another notation besides the correction edits which denotes the learners' English level: A (beginner), B (intermediate), C (advanced). The second one is LOCNESS corpus (Sylviane Granger, 1998). It is written by native English students hence the mistakes in it are subtle. Just like W&I, the sentences in LOCNESS are annotated with a notation N (Native) to specify the writers' ability. The two datasets can be regarded as one corpus together. We didn't use the learner proficiency information from this dataset because we would like to build a system that is suitable for all levels' users. However, in other studies, some people further make use of this information to improve the performance of their GEC system. The train dataset contains 10493 sentences of level A, 13032 sentences of level B and 10,783 sentences of level C. All sentences of level N are in BEA development set and test set.

2.4. Test Datasets

2.4.1 CoNLL-2014

For testing set, the CoNLL-2014 test set (Hwee Tou Ng et.al, 2013) is frequently used in other paper. This is the convolutional F0.5 metric benchmark to compare different GEC systems. It is very small, only containing 1,312 sentences comes from 50 essays written by 25 undergraduate students in Singapore. Each of the source sentence has been annotated by two different experts. The shortcoming of this test set is that topics of these essays are very limited so it may not be well-generated to other topics. Each test sentence has 2 reference sentences.

2.4.2 JFLEG

Another commonly used corpus for evaluating GEC system is JFLEG (JHU Fluency-Extended GUG corpus) test set (Courtney Napoles, 2017) which has 747 sentences. The sentences are written by candidates from TOEFL exam. This dataset mainly focuses on fluency edits of a sentence. The metric for this dataset is GLEU. The fluency edits correction strategy is different from minimal edits correction strategy represented by F0.5. Each of the source sentence has 4 different references which are corrected by rewriting instead of making modification on the original sentence to make them fluent and native-sounding.

Because we use both fluency edits-based corpus (lang8) and minimal edits-based corpus (NUCLE and FCE) during our training process, we would like to compare the performance of our GEC system in both ways evaluated by F0.5 and GLEU together.

2.4.3 BEA Test Dataset

BEA test set is a subsample of the W&I+LOCNESS corpus. Both the training and testing sets have a similar distribution for sentences of each level. For test set, there are 4477 pairs of sentences with 1107 from level A, 1330 from level B, 1010 from level C and 1030 from level

N. The reference sentences for test are not public available yet, but the evaluation result measured by F0.5 can be download from the BEA website. This is a relatively new dataset compared with ConLL14 and JFLEG, and we only use it as a supplement for those two fundamental test sets.

2.5 M2 Format

There are many kinds of tagging forms for GEC corpus. Some of them are visually intuitive but not that easy to deal with by computer program like marking different types of errors by different colors. Some of them are convenient but have information redundancy such as directly given the source and target sentence for each sample. The BEA shared task organizer has converted all corpus into M2 format which is information-efficient and can be easily transformed into sentence pairs through regular expression.

An example of M2 format is as follows:

```
S This are a sentence .
A 1 2 ||| R:VERB:SVA ||| is ||| -REQUIRED- ||| NONE ||| 0
A 3 3 ||| M:ADJ ||| good ||| -REQUIRED- ||| NONE ||| 0
A 1 2 ||| R:VERB:SVA ||| is ||| -REQUIRED- ||| NONE ||| 1
A -1 -1 ||| noop ||| -NONE- ||| REQUIRED ||| -NONE- ||| 2
```

The first line followed by capital letter S is the tokenized source sentence which may contain some errors. The lines followed by capital letter A is an annotation for the source sentence to correct its error. The first element for each annotation the offset of start and end token for this error, following by the type of this error and the tokenized correct string. We don't need to use the next two elements. The last element denotes the annotator's id for this error. If it is a noop edit, that means the annotator thinks it is not necessary to modify the original sentence.

For the example above there are three references for the original sentence. The first reference is 'This is a good sentence .', the second reference is 'This is a sentence .' and the third reference is the original one 'This are a sentence .'

2.6 Unlabeled Datasets

In our report, we also need some unlabeled datasets to train our model. Those datasets are usually big enough to help us alleviate the problem of insufficient labeled data. Firstly, we use those unlabeled datasets to train a language model. Since the datasets are sufficient, we could build a good inference system that predicts the next word and even get a valid fluency score for a sentence. Then, enough unlabeled datasets can help us pretrain our model. In fact, the labeled datasets like FCE and NUCLE do not have enough sentences. If we start our model training process with random initialization, it will need a lot of epochs and take a long time to train the whole model. So, we decide to select a well-pretrained model and fine tune it. Our pretrained model will make full use of unlabeled datasets and we will turn them into unlabeled datasets by adding noise.

2.6.1 Google Billion Word Benchmark

The One-Billion Word benchmark (Chelba et al., 2013) is a large dataset. It is actually derived from a news-commentary site. It contains a lot of articles and can provide us with many sentences. The dataset includes near 830,000,000 tokens over a vocabulary of about 800,000 words. Besides, sentences in this dataset are shuffled and thus context is limited.

2.6.2 WikiText-103

The WikiText-103 (Merity et al., 2016) corpus contains about 270,000 unique words and each word occurs many (more than 3) times in the dataset. It has about 100M tokens and 260K vocabulary in its dictionary. The dataset consists of shuffled article from Wikipedia where the context is split by sentence.

3. Our Approaches

3.1 Transformer and Attention

3.1.1 Introduction

Transformer (Ashish Vaswani et al. 2017) is a neural network model and achieve state-of-the-art results for machine translation tasks. The fundamental building block of transformer is called self-attention mechanism. Since Transformer can lend itself to parallelization, it is an advantage for it to speed up the process of model training. Compared with traditional neural network model, transformer overcomes some shortcomings. For convolutional neural network (CNN), which uses 'filter' to centralize nearby information of each input unit, if the sequence is very long, CNN needs to stack many filter layers to encode global information. For recurrent neural network (RNN), the calculation process is one by one, so it is hard to parallelize and the training process will be very slow. Compared with Long Short-Term Memory (LSTM), transformer models are different. They consider the entire sentence as a whole unlike LSTMs (or in general RNNs) where the sentence is processed sequentially -- one word per time step. Transformer sees all words simultaneously -- so there is no backpropagation through time.

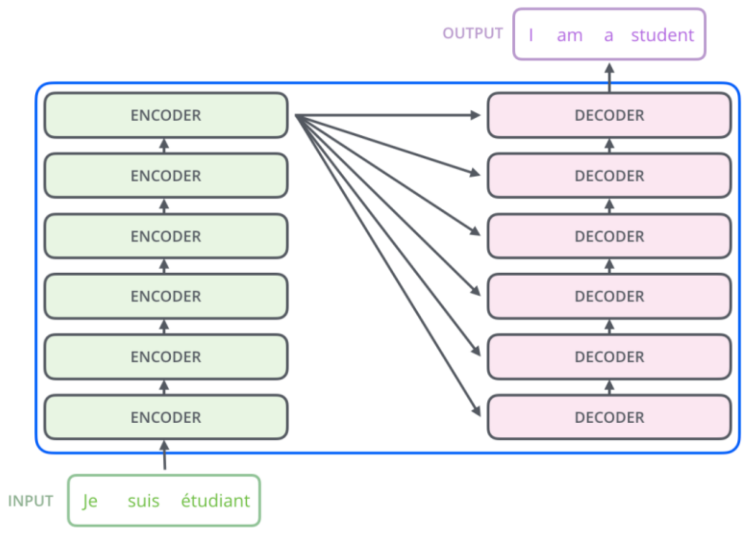


Figure 1
(source: https://jalammar.github.io/illustrated-transformer/)

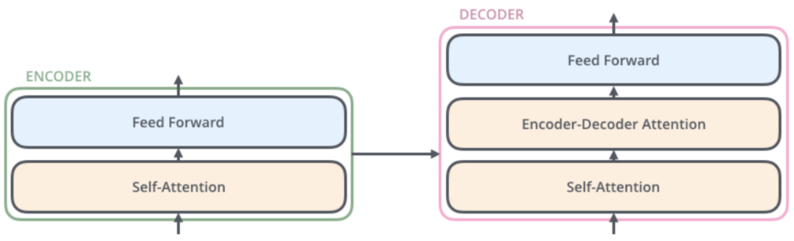


Figure 2
(source: https://jalammar.github.io/illustrated-transformer/)

3.1.2 Transformer

Basically, as illustrated in Figure 1 and Figure 2, transformer is composed of encoder part and decoder part. The encoding component is a stack of N (usually N=6) encoder structures, and each encoder has a multi-head self-attention layer, following by a feed-forward neural network. The decoder is also a stack of N (usually N=6) similar decoder structures. Each of them is similar as the block in the encoder, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence. In addition, transformer uses residual structure and layer normalization techniques in each sub-layer to improve its training speed and accuracy. To sum up, the encoder encodes the input sequence to hidden states, and the decoder decodes the hidden states with the help of previous information.

The sentences entered from the encoder first pass through a layer of self-attention, which helps the encoder focus on the other words in the input sentence when it encodes each word. The output from the attention layer is transferred to the feed-forward neural network. The feedforward neural network corresponding to the words in each position is exactly the same. There is also a feed-forward layer and a self-attention layer in decoder just as the encoder. In addition, there is an attention layer between the two levels, which is used to focus on the relevant parts of the input sentence. Word embedding occurs only in the lowest encoder. All

encoders have the same feature that they receive a list of vectors, each of which has 512 dimensions. Now let's start encoding. As mentioned above, an encoder receives a list of vectors as input, then passes the vectors in the list to the self-attention layer for processing, and then to the feedforward neural network layer, passing the output to the next encoder.
To better understand the order of the input words, transformer adds a position vector for each input word embedding. In this way, the position of each word and the distance between different words in the sequence will be learned by the model. The position vectors are added to the word embedding so that they can better express the distance between the words in the following operations. There is a residual connection around each sublayer (self-attention, feedforward network) in each encoder, followed by a layer-normalization step.

Batch Normalization calculated mean and variance for a minibatch input sample and normalized each case in the input of a certain layer of neural network based on the calculated mean and variance. However, Batch Normalization has two obvious shortcomings. (1) It is highly dependent on the size of mini-batch, which will be constrained in actual use. (2) It is not applicable to normalize operation in RNN network. When Batch Normalization is used, statistical information such as mean value and variance of the mini batch of a certain layer of neural network needs to be calculated and saved. However, for RNN, the length of sequence is inconsistent. In other words, the depth of RNN is not fixed. Different time-steps need to save different statics features. But Layer Normalization can solve these problems effectively. Unlike Batch Normalization, Layer Normalization normalize the input of all neurons in a layer of the deep network. The input of neurons of the same layer in Layer Normalization has the same mean and variance, and different input samples have different mean and variance. In Batch Normalization, mean and variance were calculated according to the input of different neurons, and the input in the same minibatch had the same mean and variance. Therefore, the Layer Normalization does not depend on the size of the mini-batch or the depth of the input sequence.

The input sequences are processed by the encoder workers. The outputs of the top encoders are then converted into attention vectors containing the key vectors and value vectors. The Query vector for this layer is obtained by the previous Decoder layer. These vectors will be used by each decoder for its own encoder-decoder attention layer. The next step repeats the process until it reaches a special termination symbol that indicates that the decoder has finished its output. At the next time step, the output from each step is transmitted to the bottom decoders, and these decoders output their decoder results just as the encoder did before. In addition, just as what we did about the input of the encoder, the positional encodings are added to those decoders to represent the location of each word.

The self-attention layer in those decoders behaves differently from the encoder. In a decoder, the auto attention layer is only allowed to process those positions further forward in the output sequence. Before the SoftMax step, it hides the following locations and sets them to '-inf'. This attention layer creates the query matrix in the layer below and retrieves the key and value matrixes from the encoder's output.

The decoder component will eventually output a vector of real number. We need to convert floating point numbers into words. The linear transformation layer is followed by the SoftMax layer. The linear layer is a fully connected layer that projects the vectors generated by the decoder components into a much larger vector which is called logits. Each unit of the large vector refers to the score of a particular word. Those scores are converted into probabilities in the next soft-max layer.

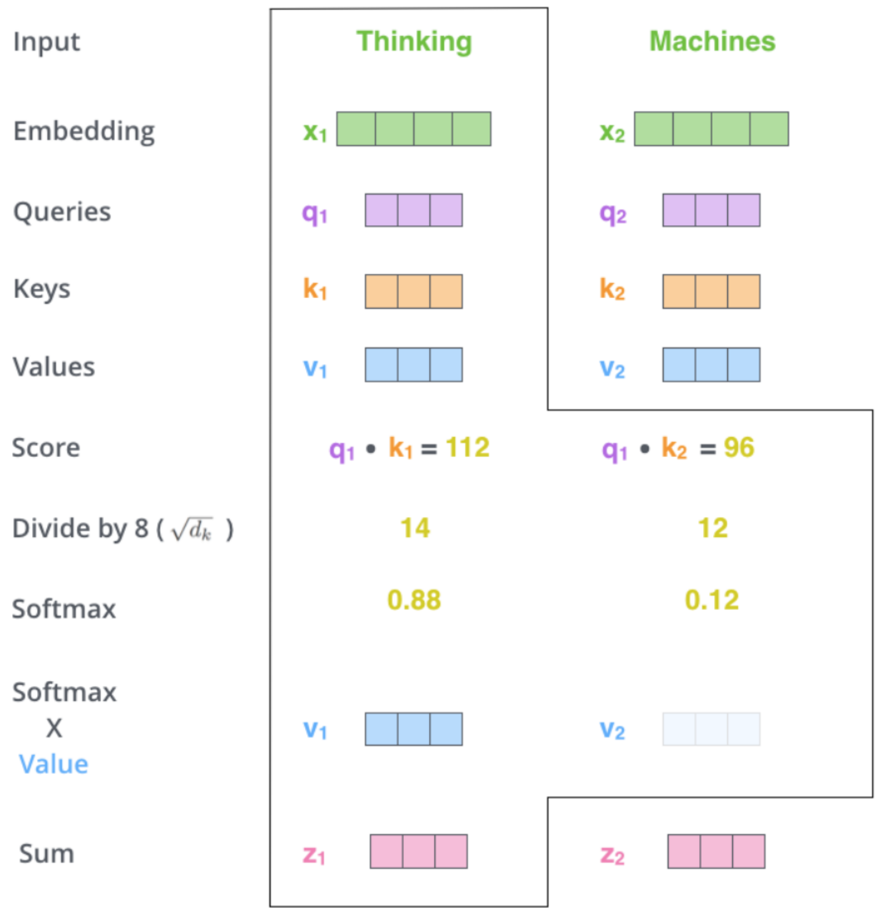


Figure 3
(source: https://jalammar.github.io/illustrated-transformer/)

3.1.3 Self-Attention

Attention is a mechanism used to improve the effect of seq2seq model. It is very widely used in image annotation, machine translation and many other fields. The Attention Mechanism is similar to the way we observe things. When human beings observe things they pay attention to, they usually do not regard things as a whole, but tend to selectively obtain some critical parts. Therefore, attention can assist the model to assign different weights to each part of the input, extract more significant messages, and enable the model to produce more effective results, without incurring more costs on the calculation and storage.

Self-attention layer, as mentioned before, is a very useful structure in transformer models. It allows us to see other position in the input sequence and to search for clues to help better coding the word. In detail, as illustrated in Figure 3, self-attention uses a query vector and a key-value vector pair for each input unit, and every input unit will attend other input unit by dot product of their query vector and key vector and then be scaled by the square root of their dimension. The scaled dot product is activated by soft-max function and used as a coefficient for each value vector, just like Equation (1):

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

The weighted sum of all value vectors is the output sequence.

Besides, the transformer also uses multi-head self-attention to strengthen its effect. The multi-head attention makes concatenation of different heads' output. Different heads may focus on different aspects of the information. For example, some heads may only focus on local information, some heads may only focus on global information, while multi-heads attentions can see both local and global information at the same time. The concatenation of heads' outputs will go through dimension reduction to mix the information.

3.1.4 Other Parts

In addition, transformer also uses positional encoding for each input sequence because the attention itself will ignore the order of the sequence and the order is very important in NLP tasks. To address this, the transformer adds a vector to each input embedding, which helps keep the information for each word's position and the distance between different words in a sentence. The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

The decoder part outputs a vector and then it will be mapped into a word from a dictionary. The linear layer is a simple fully connected neural network that projects the vector produced by the decoder part into a much larger vector. Then we will build a dictionary learned from training dataset. The soft-max layer can help turn those scores into probabilities and the model will chose the word with highest probability. Since the model predicts one word at a time, we can assume that the model is selecting the word with the highest probability from that probability distribution and throwing away the rest. This method is called greedy decoding. However, it is not wise to drop the rest of high probability words. Therefore, another way is to hold on to top N words, then in the next step, run the model for N times. We repeat this for positions 2 and 3 and so on, then find the best combinations. This method is called "beam search".

3.2 Copying Mechanism

3.2.1 Loss Function

For traditional machine translation task, we have its specific loss function. Just like machine translation, our grammar error correction task also uses their loss function.

In order to compute the loss function, we simply compare our model's output probability distributions and correct output probability distributions. It is a sum of the cross-entropy loss of each word during decoding process. The formula is shown in (2):

$$loss = -\sum_{t=1}^{T} \log\left(p_t(y_t)\right) \qquad (2)$$

In this equation, y_i (i=1,···,T) represents a sequence of word tokens and p_i (i=1,···T) is the vocabulary distribution that is calculated by the decoder part.

3.2.2 Copy Mechanism

Copying mechanism is an effective method in some NLP tasks, such as text summarization (Jiatao Gu et al., 2016) and semantic parsing (Robin Jia and Percy Liang., 2016). Jiatao Gu combines encoder-decoder and copying mechanism, which can copy certain segments from the source sentence to target sentence. The innovation point is that decoder uses the probability of two modes rather than original version (just one): generate-mode and copy-mode, and the later extracts words from source sequence. Moreover, the update approach of states is also changed from word before to word-embedding and the position information of the source sequence.

In our GEC task, copying mechanism is also a useful idea just like it in text summarization tasks. Unlike traditional seq2seq model, grammar error correction model is used to correct sentences. In many cases, we do not need to change the tokens or words because it is correct. Then copying mechanism is able to copy tokens or words from the source sentence. It's a natural feeling that it will be very suitable. Besides, our dataset is not very big and cannot contain every word in the dictionary. Therefore, if we correct a sentence, it is common that some unknown words will appear. In order to deal with unknown tokens, we import copying mechanism, which can help copy unknown tokens or words into target sentence.

As illustrated in Figure 4, vocabulary distribution P_t^{gen} is original output of the decoder part, but it is not the final distribution that generates target tokens or words. Copy-augmented network will consider the copy distribution P_t^{copy} which is generated by the source sentence. Basically, the copying mechanism will add a new attention distribution between the encoder's hidden states $H^{src}(h_{1\ldots N}^{src}$ in figure 4) and the decoder's current hidden state h^{trg}. The process that we calculate the copy attention distribution is shown in Equation 3, 4, 5, which is the same as the transformer's encoder-decoder attentions:

$$q_t, K, V = h_t^{trg} W_q^T, H^{src} W_v^T, H_{src} W_v^T \quad (3)$$

$$A_t = q_t^T K \quad (4)$$

$$P_t^{copy}(w) = softmax(A_t) \quad (5)$$

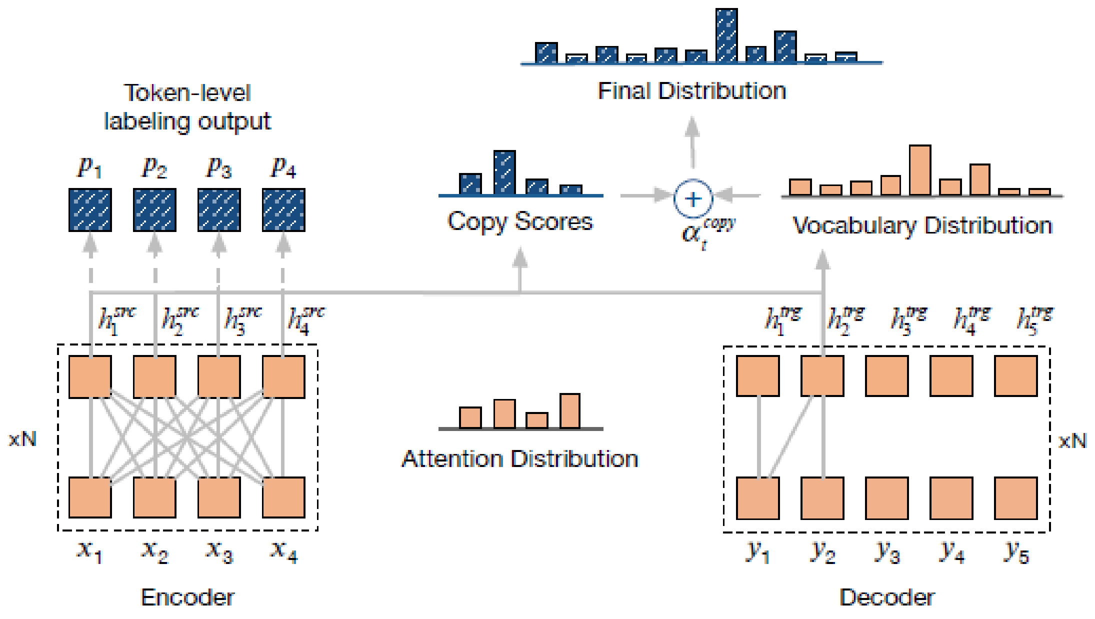
As the Equation 3, 4, 5 shows, query vector (q_t), key vector (K) and value vector (V) are used to calculate the attention distribution. Then the normalized attention distribution will be used as the copy scores just like figure 4. We can also get the balancing factor α_t^{copy} by the copy hidden states, just like Equation 6.

$$\alpha_t^{copy} = sigmoid(W^T \sum (A_t^T \cdot V)) \quad (6)$$

Finally, at each time step t, we can calculate the final distribution by Equation 7:

$$p_t(w) = \left(1 - \alpha_t^{copy}\right) * p_t^{gen}(w) + \left(\alpha_t^{copy}\right) * p_t^{copy}(w) \quad (7)$$

This is the whole process that we add copy-augment network to the original transformer structure. As all the equations show, the copy mechanism makes full use of the information of the original tokens or words and contain them in the final distribution.



Final Distribution
Token-level labeling output
p_1 p_2 p_3 p_4
Copy Scores
α_t^{copy}
Vocabulary Distribution
h_1^{src} h_2^{src} h_3^{src} h_4^{src}
h_1^{trg} h_2^{trg} h_3^{trg} h_4^{trg} h_5^{trg}
xN
Attention Distribution
xN
x_1 x_2 x_3 x_4
Encoder
y_1 y_2 y_3 y_4 y_5
Decoder

Figure 4 (Zhao et al., 2019)

3.3 Pretrain

Pre-training plays a key role in deep learning. Deep neural network has the following disadvantages. (1) The deeper the network, the more training samples are required. If supervised learning is used, a large number of samples need to be labeled, otherwise small samples are easy to cause overfitting. (2) Parameter optimization of multilayer neural network is a problem of high order non-convex optimization, which usually has poor convergence. (3) There is a gradient diffusion problem. The gradient calculated by the back propagation algorithm decreases significantly as the depth advances, resulting in small contribution of the previous network parameters and slow update speed.

When building a network model to complete a specific task, the traditional method needs to randomly initialize the parameters, then start training the network, and keep adjusting until the loss function of the network converges. When the results are satisfactory, the parameters of the training model can be saved so that the trained model can get better results the next time it performs a similar task. This process is pre-training.

The essence of pre-training is that model parameters are no longer randomly initialized, but pre-trained by some tasks such as language model. Pre-training belongs to the category of transfer learning. The pre-training model in this paper mainly refers to unsupervised pre-training task, while the transfer method is mainly model finetune. In natural language processing tasks, large-scale text corpus is used for pre-training and small data sets for specific tasks are fine-tuned to reduce the difficulty of a single NLP task.

Denoising auto-encoders (Pascal Vincent et al., 2008) is used to pre-train deep neural network. This paper mainly describes how to use denoising auto-encoder to learn the intermediate features. Further, this method can initialize the weight of the neural network which is equivalent to an unsupervised method of learning to train neural networks. Unlike traditional auto encoders, in denoising auto-encoders some of the input values are randomly set to 0 to get the corrupted result. Then the denoising auto-encoder map the damaged input values to y and the reconstructed value z is calculated. Finally, the rebuilt value z is compared with the original value x to calculate the loss function and gradient descent is used to update the parameters. Denoising auto encoder can be used for DNN pre-training as normal auto encoder. After the training process of noise addition and noise reduction, the network can be forced to learn stronger robust invariance characteristics and get more effective expression of input.

BERT (Devlin et al., 2018) took an unsupervised approach to train the bi-directional transformer model. BERT mainly focus on the pre-train methods and uses Masked Language Model and Next Sentence Prediction two methods respectively to catch phrases and Sentence level representation. The inputs of previous models are a sequence of text from left to right, or the combination of left-to-right and right-to-left. The experimental results show that the bi-directional language model can understand the context more deeply than the unidirectional language model. The purpose of BERT is to generate language models and only transformer encoder is needed. Transformer's encoder can read an entire text sequence at once, acting as a bi-directional model. There are two pre-training tasks in BERT, Masked Language Model and Next Sentence Prediction. Before entering the sequence of words to BERT, 15% of the words in each sequence are replaced with [MASK] tokens. The model then will begin to predict the masked original word based on the context of other unmasked words in the sequence. There is a trick in how to do the Mask, 10% of the words in [MASK] tokens will be replaced with other real words, 10% of the words will not be replaced, and the remaining 80% will be replaced by [Mask]. During BERT's pre-training, the model receives pairs of sentences as input and predicts whether the second of these sentences will also be a following sentence in the original file. During the training, 50% of the input pairs are contextual in the original document, and the other 50% are randomly composed from the corpus and disconnected from the first sentence.

Inspired by denoising auto-encoders and BERT, copying mechanism model is pre-trained by using One Billion Word Benchmark (Chelba et al., 2013). One Billion Word Benchmark is a big English corpus with almost one billion words of training data. The pre-training data is generated by the following ways. (1) A token is deleted with 10% probability. (2) A token is added with 10% probability. (3) Randomly replace a word in the original sentence with a word in the vocabulary with 10% probability. (4) Shuffle the words by adding a normal distribution deviation to their position, then reorder the words according to corrected results with a standard deviation 0.5.

Through a lot of training data, the transformer-base model is able to learn how to reconstruct the input sentence with noise. What is more, the decoder is initialized by the pre-trained parameters, but other parameters are initialized randomly.

4. Tracks

4.1 Language Model

The language model modeling method can be divided into two categories: statistical language model and rule language model. The latter is a deterministic language model based on Chomsky's formal language, which pays more attention to the analysis of grammatical information in the language. Statistical language model has the advantages of high accuracy, easy training and easy maintenance. Currently, language model is widely used in natural language processing, especially in speech recognition, machine translation, automatic word segmentation, syntactic analysis and other related researches based on statistical model. The most common language model is the n-gram language model, which is simple and straightforward to build but needs smoothing due to the lack of data. In a language model, the probability distribution p(s), usually constructed as the string s, represents the probability of the string s appearing as a sentence.

Statistical language model includes n-gram model, decision tree model, maximum entropy model and so on. There are several major problems in the use of statistical language models. (1) Data sparsity problem: To solve the data sparsity problem of statistical language model, we can adopt the method of enlarging the training corpus or using various smoothing algorithms. (2) Domain dependency problem: Statistical language model is strongly dependent on the data domain. One direct way to solve the dependency problem is to collect corpus of relevant domain, and the other way is to use adaptive method. (3) Model size is too large. (4) Decoding speed is slow.

Statistical language model obtains the probabilistic relationship between words by analyzing and making statistics of the massive corpus collected. In general, statistical language model, the scale of dictionary is relatively large. If the trigram model is adopted, the number of all trigram conforming to the rules will grow exponentially. In fact, the training corpus we can obtain is far less than this order of magnitude, which leads to the fact that some trigram units cannot be observed in the training corpus, resulting in data sparsity. The solution of sparsity problem is usually giving reasonable probability values to those units that do not appear in the training corpus, which is the smoothing problem of statistical language model. The basic idea of data smoothing is to reduce the conditional probability distribution of the occurrence of n-gram so that the non-occurrence of n-gram conditional probability distribution is non-zero. After data smoothing, the probability sum is guaranteed to be 1.

Language model is the task of predicting the next word or tokens in a paragraph, with lots of applications such as speech recognition task (Ebru Arisoy et al., 2012). In recent years, neural methods based on recurrent neural networks (such as LSTMs (Rafal J´ozefowicz et al., 2016), self-attention structure (Rami Al-Rfou et al., 2018)) and convolutional neural networks (Yann N. Dauphin et al., 2017) have been shown to make much progress. Neural networks even outperform classical n-gram language models (Kneser et al., 1995; Chen et al., 1996).

In our grammar error correction task, neural language models are also useful in the whole systems. Since language model can help predict the next word in a sentence, it is also able to calculate a score for a given sentence. The score might come from the probability distribution of a sequence of words, i.e.

$$P(w_0, \dots, w_N) = P(w_0) \prod_{i=1}^{N} P(w_i | w_0, \dots, w_{i-1}), \tag{8}$$

where w_i are discrete word indices in a vocabulary. Therefore, a good language model can help us select and check the result of our models' output. Specifically, we will build a context-aware neural spellchecker and incorporate a neural language model. After achieving some outputs of our copy-augmented network, we will re-rank those outputs based on language model and model scores. Besides, we will do the fluency boost inference where the language model is necessary. Those parts will be included in the following sections.

There are several kinds of language models and we have introduced them in the previous sections. In our project, we mainly use neural language model, since it is a popular approach recently and outperforms lots of traditional methods. We also import adaptive input representations which add the adaptive soft-max into input word representations and output vectors (Alexei Baevski and Michael Auli, 2019). Sometimes, there are plenty of words in our dictionary, thus it is hard to compute every output vocabulary. In order to lower the computational burden, we need to import the adaptive soft-max which assigns more weight parameters to frequent words and fewer weight parameters to rare words. (Edouard Grave et al., 2017) Besides, the method of adaptive input embeddings is also introduced and it gives more priority to frequent words in dictionary.

In detail, adaptive representations are used in word embeddings part. As illustrated in Figure 5, we set some clusters that help partition the frequency ordered vocabulary $V = V_1 \cup$

$V_2, \dots, V_{n-1} \cup V_n$ such that $V_i \cap V_j = \emptyset$ for $\forall i,j$,and $i \neq j$, where V_1 contains the most frequent words and V_n the least frequent words. Also, the capacity of each cluster is reduced by a factor of k (Usually k=4, following (Edouard Grave et al., 2017)). Therefore, V_1 has a dimension of d and V_n has a dimension of $\frac{d}{k^{n-1}}$. Then, the adaptive input embedding layer will map the embeddings from each cluster to dimension d by linear projections $W_1 \in R^{d \times d}, \dots, W_n \in R^{d/k^{n-1} \times d}$. Those vectors with dimension d will be concatenated and fed into the model.

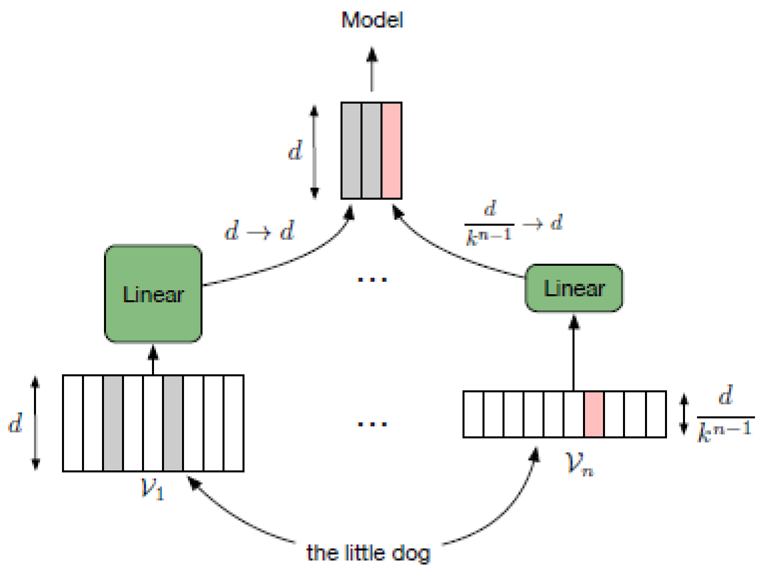


Figure 5(Alexei Baevski and Michael Auli, 2019)

For the output layer, an adaptive softmax with nearly the same partition is also used in our pre-trained model. In general case, the dictionary is partitioned as $V = V_h \cup V_1 \dots V_J, V_i \cap V_j = \emptyset \; if \; i \neq j$. As illustrated in Figure 6, The head sub-dictionary V_h can be accessed at the first level, and the others $(V_1 \dots V_J)$ in the second level. The model will predict the probability distribution from the head sub-dictionary at first. If the result shows that tail parts $(V_1 \dots V_J)$ have the maximum probability value, the model will turn to V_i and use forward neural network to predict. In this way, we can build a much larger dictionary while not improving the computational burden.

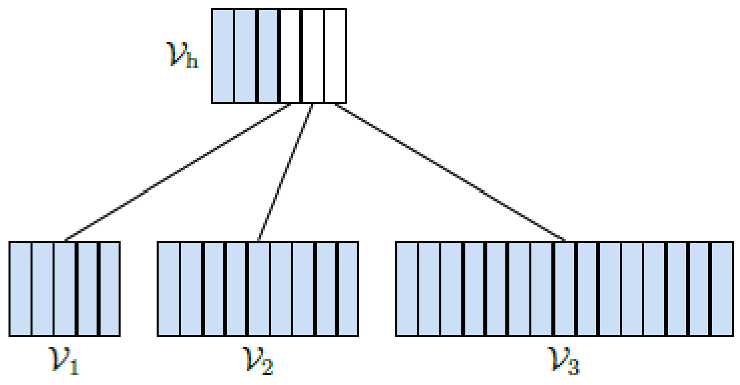


Figure 6(Edouard Grave et al., 2017)

Besides, in order to reduce the number of parameters, both the projections $W_1, \dots, W_n$ and the parameters for the actual words can be shared in our model. Since the model output vectors will be multiplied with the word embeddings in the head sub-dictionary, the head projection is not shared. Then we share the parameters except for the head because the adaptive softmax has $n-1$ additional embeddings for clusters which are not the same as the input.

In conclusion, out language model is a self-attention architecture (Ashish Vaswani et al., 2017), which is also a basic transformer structure mentioned before. Since the experiment resources are limited and the training process is very long, we download the pre-trained model from the FAIRSEQ sequence modeling toolkit (Myle Ott et al, 2019) and use it to calculate the fluency of sentences.

4.2 Spellchecker

4.2.1 Introduction

Spelling mistakes are common in English text especially from non-native learners. Missing, superfluous or wrong letters may cause misunderstanding, and will also reduce the precision of the model. Spellchecker is a powerful tool to detect spelling mistakes of texts, which is widely used in text editor such as Word. There are some common Spellcheckers, such like Enchant, Hunspell, Aspell, Hspell, Zemberek, Voikko, AppleSpell and pySpell. Most of them are open source programs and libraries, which can judge if a word is right, give some words which are corresponding to the original word and determine what is the most possible corresponding words.

4.2.2 Algorithms of spellcheckers

Sometimes outputs suggested by spellcheckers are not so reasonable. For example, when we input a sentence “What a beautifely woman!”, we can find that “beautifely” is a wrong version of word “beautiful”. Spellchecker find this word and give a suggestion dictionary $\mathrm{W} = \{\mathrm{w}_1, \mathrm{w}_2, \dots, \mathrm{w}_n\}$ to correct it (w_i is a word). Finally, it suggests “beautifully” (which is w_1 in suggestion dictionary) as the most possible output, rather than “beautiful”, which is the true word we know. The reason is that the final two letters, “ly”, are same as the final two letters of “beautifully”, so this “beautifully” is more similar with the original word “beautifely”. Because of the shortcoming that spellchecker just finds most similar corresponding word, some GEC developers do not merge spellchecker into their model.

Some corresponding blogs introduce algorithms of spellcheckers, just like the recommended website of pyspellchecker which is from Peter Norvig's blog: https://norvig.com/spell-correct.html. Now let us use some probability theory to describe this situation. The key determining the spelling checking rational or not is finding a function $\mathrm{correct(w)}$ of a word w, which try to choose the best correction for w. However, it is difficult to define which is "best" since a word has different best version in different environments (for example, the wrong word is "lates", and there are several choices "lattes", "latest", "late" or some other words). A useful algorithm is trying to extract the most suitable candidate c of the suggestion dictionary C containing all suggested candidate corrections, in which the highest probability is gotten when c is the final correction when the word w is given, i.e.,

$$\mathrm{argmax}_{\mathrm{c} \in \mathrm{candidates}} P(c|w)$$

According to Bayes' Theorem, it also equals to:

$$\mathrm{argmax}_{\mathrm{c} \in \mathrm{candidates}} P(c)P(w|c)/P(w)$$

For any possible candidate, $\mathrm{P(w)}$ is the same value, so we can throw it out, which means the formula is equal to:

$$\mathrm{argmax}_{\mathrm{c} \in \mathrm{candidates}} P(c)P(w|c)$$

In general, there are four main portions of the expression above:

- **Selection Mechanism:** argmax
  Return the candidate which has the highest probability.
- **Candidate Model:** $\mathrm{c} \in \mathrm{candidates}$
  Return which candidate corrections to consider
- **Probability Model:** $\mathrm{P(c)}$
  The probability of the candidate c appearing in the text as a word. For example, if the occurrences of "beautiful" of the target text makes up about 1%, then $\mathrm{P(beautiful)} = 0.01$.
- **Error Model:** $\mathrm{P(w|c)}$
  The probability that w may be inputted into the text if the writer wants to input c. For example, if the P(beaitif|beautiful) is high for a writer, then P(beautiful| beautiful) would be low.

Rethinking the process before and after the usage of Bayes' Theorem, the original term $\mathrm{P(c|w)}$ becomes complex because it is replaced with an expression involving two different models rather than one. The purpose of this transform is that $\mathrm{P(c|w)}$ is hard to be expressed since it is conflating two different factors, and it is easier to divide these two parts and express them explicitly. For instance, there is a misspelled word $\mathrm{w} =$ "thew" and its suggestion dictionary contains c = "thaw" and c = "the". It's hard to know which has a higher value of $\mathrm{P(c|w)}$. We can say that "thaw" looks like better since it just changes "a" to "e", which is a very small change. We can also say that "the" seems better since this word is so common but adding "w" to the original word seems a bigger change, which means less probability change (maybe just a classical finger slipped off the "e"). This example reflects the fact that it's difficult to estimate $\mathrm{P(c|w)}$ both the probability of c and the probability of the transform from c to w, hence separating these two factors is cleaner. (PS. This example is also from the website: https://norvig.com/spell-correct.html).

Except this simple spelling checking algorithm, spellcheckers use some other algorithms to find the correction. Pyspellchecker (we use it in our model) uses a Levenshtein Distance algorithm to get permutations between the edit distance of 2 from the initial word, then it compares all permutations, which contains deletions, transpositions, insertions and replacements, to known words from the word frequency dictionary. Words found more frequently in the frequency dictionary are more likely to become the correct results. Additionally, Hunspell is another popular spell checker. Hunspell is widely used in some products, such as Mozilla Fire fox, OpenOfiice.org, LibreOffice & Thunderbird. Hunspell is much more complex than Pyspellchecker, which improves suggestion using n-gram similarity, dictionary based pronunciation data and some rules. Moreover, some other tips are also used such as stemming, morphological analysis and generation. Hunspell is the official spellchecker of some services and products, such as Mozilla Firefox 3, thunderbird, Google Chrome and so on. In addition, some software packages also use it, containing macOS, Opera, SDL Trados and so on.

4.2.3 Language Model + Spellchecker

In fact, the most complicated spelling checking algorithm still cannot avoid the wrong spelling since it just focuses on a single word in a phrase or sentence. The answer of the example about "beautiful" can get different answers in different situations. A natural thought is that the choice will become more suitable if we can judge candidates of the misspelled word in the sentence aspect rather than just the single word aspect.

In our model, we combine language models and pyspellchecker (one of the open source spellchecker) to correct spelling errors before input sentence into our main model. As mentioned in 4.1, language model can give a score to each sentence after training. A sentence which is more fluent or proper will get higher score, so using language model to choose words from suggestion dictionary will get highest score after replacing the original word. The specific process is as following.

- Step1: Input a sentence $\mathrm{S} = s_1 s_2 \dots s_{i-1} s_i s_{i+1} s_m$, and check each word is right or not. If a word $\mathrm{s_i}$ is wrong, go to the next step, where $\mathrm{s_i}, \mathrm{i} = 1,2,3,4 \dots, \mathrm{n}$ represents word of sentence.
- Step2: Use Pyspellchecker to get the suggestion dictionary $\mathrm{W} = \{\mathrm{w_1}, \mathrm{w_2}, \dots, \mathrm{w}_n\}$, and replace $\mathrm{w_j}, \mathrm{j} = 1,2 \dots., \mathrm{n}$ with original word $\mathrm{s_i}$. Store all possible checked sentences and the original sentence into $\mathrm{C} = \{\mathrm{C_1}, \mathrm{C_2}, \dots, \mathrm{C_n}, \mathrm{S}\}$, where $\mathrm{C_j} = s_1 s_2 \dots s_{s-1} w_j \mathrm{s_{i+1}} \dots s_m$.
- Step3: Use language model to find the sentence C_{max} with the highest score. Then return to step1 and repeat check process but begin from $\mathrm{s_{i+1}}$ to $\mathrm{s_m}$ now.

It is worth mentioning that in order to avoid the mis-check situation caused by some proper nouns such as "Facebook", which is not recorded by dictionary of Pyspellcheck, we input the original sentence into the sentence set. If this word is a proper noun, language model will recognize and choose it. For example, in JFLEG dataset, before we add this tip, "Facebook" and other newly proper noun are all be fixed into some other words, while they are preserved after adding this tip.

4.3 Boost and Re-rank

For a Grammatical Error Correction seq2seq model, the basic training data is a pair of corrected sentences composed of original sentences and correct sentences. Theoretically, as long as there is a large amount of training data, we can get a relatively perfect Grammatical Error Correction model. In practice, however, the number of such sentence pairs is quite limited. Therefore, the generalization ability of seq2seq model will be affected if the training data is not enough. As a result, even if the input sentences are slightly changed, the model may not be able to correct them completely. At the same time, we also found that single seq2seq inference could not completely correct a sentence with multiple grammatical errors. In this case, we may need to use multiple rounds of seq2seq inference to modify a sentence repeatedly.

Based on the traditional seq2seq model, a new learning and inference mechanism -- fluent boost learning and inference is proposed. The core principle of fluent boost learning is that in the process of training model, the seq2seq model generates multiple results, and then matches the generated sentences with the correct sentences whose fluency is not as good as the target correct sentences, forming a new fluent improved sentence pair, which can be used as the training data of the next round of training. The fluent boost inference utilizes the seq2seq model to modify the sentences several times until the fluency of the sentences is no longer improved. This multi-round modification strategy can eliminate some grammatical errors in the sentence firstly, making the context of the sentence clearer and helping the model to correct the remaining errors.

4.3.1 Fluency Boost Learning

As for Grammatical Error Correction problem, a qualified training sample usually needs to meet two requirements. The semantics of the source sentence and the target sentence should be the same, for we don't want the modified sentence to change the original meaning. The fluency of the target sentence can be improved, which is the actual ultimate goal of Grammatical Error Correction.

The sentence pairs generated by fluency boost learning can well meet the above two conditions. Firstly, since our seq2seq model is trained with error-correction sentence pairs as training data, sentence pairs created by fluency boost learning usually do not change the original sentence. When the model is relatively stable, the sentence in n-best results is usually only 1 to 2 words different from the original sentence, and rarely changes the original sentence. Secondly, the fluency enhancement sentences can guarantee a higher fluency of the target end sentences than the source end sentences.

We are going to start with the concept of fluency score. Generally, native speakers are more likely to write fluent sentences. Just as the formulas below, $f(x)$ is the fluency score (TaoGe, FuruWei, MingZhou , 2018) of sentence x and $H(x)$ is the cross entropy of sentence x:

$$f(x) = \frac{1}{1 + H(x)}$$

$$H(x) = -\frac{\sum_{i=1}^{|x|} \log P(x_i | x_{<i})}{|x|}$$

Given context $x_{<i}$ beforehand, the language model is able to compute the probability of x_i. We can find that the fluency score of a sentence is inversely proportional to the cross entropy of the sentence. Fluency score function is implemented for both boost learning and boost inference and nature sentences tend to have higher fluency score.

Given a training example, fluency boost learning generates multiple sentence pairs which help expand the training set and improve model learning ability.

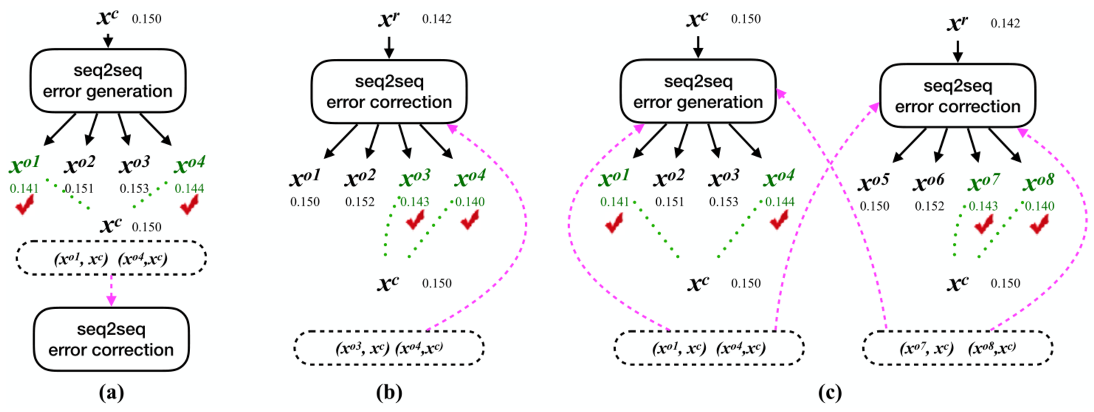


Figure 7(TaoGe, FuruWei, MingZhou , 2018)

To augment training data to provide more examples of common errors, fluency boost sentence pairs are generated from original sentence pairs to benefit the model. There are three methods of boost learning, back-boost, self-boost, and dual boost (TaoGe, FuruWei, MingZhou , 2018). The method of generating fluent sentence pairs through back-translation is called back-boost learning, and the method of generating fluent sentence pairs through correction model itself is called self-boost learning. There is an error generation seq2seq model and an error correction model in back-boost. Correct sentences are input into the error generation model and the less fluent sentences are generated. Then, those sentences are paired with error correction model and are feed into the error correction model. As for self-boost, there is only one error correction model. Sometimes the error correction model may also generate sentences with less fluency score. But they are not useless. They are paired with the correct sentences and put back into the error correction model during training. For self-boost learning, in different training stages, the iterative updating of the model will lead to different n-best candidate sets generated before and after the same sentence, so the sentences generated by the model with errors are more diversified. Since back-boost and self-boost generate smooth sentence pairs from completely different angles, which means that the two approaches complement and reinforce each other. Therefore, we further combined the two methods to produce the final dual-boost learning method.

The last one is dual-boost. Dual-boost is the combination of back-boost and self-boost. The error correction model and error generation model are dynamically updated and mutually promoted by putting the results of one model into the training data of the other model. The influent sentences generated by the error generation model can expand the error correction model’s training set to get a better result, and the influent sentences generated by error correction model can improve the performance of the error correction model. It is worth mentioning that our fluency boost methods can also be applied to a large number of correct texts. Since a correct sentence can be regarded as a correct sentence pair with the same source and target ends, applying fluency boost learning to correct text can help us greatly expand the scale and content diversity of training data.

4.3.2 Fluency Boost Inference

Fluency boost inference takes advantage of the particularity of Grammatical Error Correction task that the input and output are essentially the same. So, we can modify the output sentence as input. Based on the multi-round seq2seq inference, a more effective method round-trip modification is proposed. Round-trip modification uses a reverse (right-to-left) decoder and a forward (left-to-right) decoder alternately to modify a sentence. Because the forward and reverse decoders have their own advantages for different errors, round-trip modification allows the two models to take full advantage of their advantages.

The inference process (TaoGe, FuruWei, MingZhou , 2018) mainly focuses on how to use the trained model to get the corrected sentence given a source sentence. The motivation is that when there are many errors in the source sentence, the context is very complicated, and the model may not be powerful enough to correct all errors at one time. More likely it will first correct the most obvious mistake. And if single round model inference cannot perfectly correct the wrong sentence with multiple errors, we use multi-round boost inference instead. We keep using the model as long as the fluency is increasing. With this incrementally edit technique the model can firstly focus on the main problem and then on the tiny problems, solving them one by one.

Besides, we use round-way decoders to do error correction. This is not difficult to understand because some types of errors are easily detected when we have previous information from the source sentence, while some other types of errors are easily detected when we have later information from the source sentence. In the model words, we train two different order decoders and using them successively in the whole inference process. Left-to-right and right-to-left decoders will well complement each other and using decoders with different orders will get the meaning of source sentence with different orders. In order to reduce the impact of random errors on the boost inference model, we set a threshold and boost results are only adopted when the score increases by more than 10%.

For example, when the input sentence is 'Tomorrow I go to party'. In the first iteration of boost inference, the output sentence becomes 'Tomorrow I am going to a party'. In the second iteration the output result becomes 'I am going to a party tomorrow'. As we can see, the output results after several iterations are likely to be more smoothly and fluently.

4.3.3 Rerank

The disadvantage of Viterbi algorithm is that sometimes it leads to low efficiency when the dictionary is huge. Beam search can be regarded as the greedy form of Viterbi algorithm. Beam search is a method which is used during the test phase for better result and accuracy. It is not used during the training stage. Beam search cannot be guaranteed to find the global optimal solution. Because the search space is too large, a relatively optimal solution is adopted. The Viterbi algorithm can quickly find the global optimal solution when the dictionary size is small. Greedy search considers the probability of the current word every time. However, some commonly used structures in English, such as 'is going', have a high probability of occurrence, which will lead to redundant sentences eventually generated by the model. Greedy search can be considered as the special case of cluster search when beam size is 1.

Encoder-decoder model with attention is basically the core of the current GEC model. But in order to further improve the performance of GEC model, there is an important practical step, reranking. Typical end-to-end Seq2Seq model directly outputs the results with the highest score in the decoded search such as beam search. However, in many cases, due to the mistake of encoder-decoder model, the result with the highest score in decoding may not be the best, so the reranking is an optimization for this problem. The particular way is very simple and straightforward. For example, beam search decoding keeps the score of the top k candidate sentences. Then, on the basis of the decoding score, we re-rank the top k candidate sentences with the probability score of the sentences on the external large language model, and then select the new highest score sentence as the final output.

4.4 Checkpoint-Average

In our experiment, we save checkpoints for each epoch and try to average some Transformer model checkpoints, since some paper have already reported that checkpoint averaging can prevent overfits and get lower variance results (Martin Popel and Ondˇrej Bojar, 2018; Marcin Junczys-Dowmunt et al, 2016). However, not every epoch can contribute to the final averaged checkpoint. Because in the beginning epoch, the model does not perform very well and needs to be trained for more iterations. Therefore, we will do some experiments based on the valid set and select the last n checkpoints.

4.5 Rule Based Method

Rule based method is a traditional way for grammatical error correction task. Some common error types can be defined as specific patterns. Here we use the open source LanguageTool sever (Daniel Naber, 2003) as our rule-based checker engine and build an interface of this sever to our GEC system. The source code is written by java that can be downloaded from github (https://github.com/languagetool-org/languagetool) and we use pylanguagetool package as a python wrapper of this sever. We built the LanguageTool locally on our sever so it can be used offline.

It should be mentioned that this rule-based checker can also be used to correct spell error. The Transformer based models and language models are designed to learn the context information and correct grammar errors. Although they can recognize some spell errors correctly, their performance on dealing with spell error is not as good as professional spell checker or language tool spellchecker. Besides, models only learn the information that it has seen in the training corpus. The training dataset of us can not cover all error situation making by users of our system, so we decide to add the rule-based checker module after our GEC system hoping it can make some compensation. In addition, spellchecker pointed out in 4.2 is also designed to these errors.

Because the performance of the language tool itself is relatively poor compared to our system, so we only select two subsets of rules as compensation. The first subset contains 8 rules to correct punctuation errors and combine separate tokens. For example, after tokenization process the full stop is separate from the sentence by a space, hence we use one rule to make the full stop follow the sentence without this extra space. Another example is after tokenization process the words “can’t”, “doesn’t” is separate to “ca n’t” and “does n’t” for word embedding purpose, hence we use a rule to combine them again to make our system more user friendly. The second subset contains 22 grammar rules. For example, the choice of “a” or “an”, a complete sentence should start by upper case and so on. Besides, we also defined a rule by ourselves to capitalize the first letter of some words of company name such as ‘Facebook’. For evaluating purpose, we only use the second subset of rules because the evaluation is also based on tokenized sentence. In our user interactive mode, we use both sets of rules to further correct the punctuation and other types of errors.

4.6 Error Generation Model

4.6.1 Introduction

We have already mentioned that one of the biggest problems for GEC task’s study is the lack of high-quality large and well annotated corpus. The models based on deep learning can benefit a lot from the increasing of training sample. So a natural idea is using different kinds of data augmentation methods to make some pseudo error and correct sentence pairs.

The fluency boost learning (TaoGe, FuruWei, MingZhou , 2018) process we mentioned above actually has data argumentation effect and requires iteratively generating synthetic corpus during the training process. This kind of error generation is not very match with our current GEC system's training process. The reason is that the fairseq framework we currently use fixes the amount of dataset during training. Besides, along with the growth of dataset during training in boost learning, the training time for each epoch will be longer and longer. In (TaoGe, FuruWei, MingZhou , 2018) they use multiple GPUs to accelerate training, and in our experiment we only have one GPU on our server. So we build a separate Error Generation Model based on CNN to enlarge our dataset before the training process.

Traditionally there are 3 methods for error generation. The first one is adding direct noise to grammatical correct data. Given the corpus such as wiki103 we can randomly add or delete or replace or mask few words in sentence or keep the original words to create the corresponding wrong version. The pre-train model we use comes from direct noise gene GEC ration method. The second one is self-boost method that we have already introduced in section 4.3.1. It uses a semi-finished model trying to repair some wrong sentences in an existed corpus. Of course, the outputs are not well corrected and we can pair the output sentences and the corresponding right answer to create new training data. The third method is backtranslation. Firstly, it reverses the original GEC corpus, using the original target sentences which are grammatical correct as inputs and the original source sentences with grammar mistakes as outputs to train an error generation model. Secondly, it uses this error generation model to generate the wrong version of the natural sentences for a seed corpus. Thirdly, with those pairs of grammatical wrong and correct pseudo sentence pairs, we can train a GEC model by using the pseudo dataset and natural corpus together. Usually the seed corpus for error generation can be very large and covers a wider range of topics than natural GEC corpus annotated by human. For example, SimpleWiki (1369460 sentences), Wikipedia (145883941 sentences) or Gigaword (131864979 sentences).

There are two ways of using those machines generated errors. The first one is joint training. At each training epoch, we use both the natural and the synthetic dataset as a whole. The second one is pre-training. We use large synthetic dataset to pretrain a model, then use natural corpus to further train and fine-tune the pre-trained model. It should be noticed that for joint training setting, we should carefully decide how much synthetic data we should use; as for pre-training setting, more synthetic data can bring better performance. The reason is that the quality of synthetic data is usually worse than natural corpus. In joint training setting if there is too much synthetic data, the teaching signal will be dominate from it and the natural corpus's signal will have less impact.

Because we have already used a pre-trained model trained by direct noise, we choose backtranslation method for error generation and the joint setting for the usage of synthetic dataset. The error generation model implemented by us is based on CNN (Convolutional Neural Network) which is different from the models based on attention such as transformer or the models based on RNN such as LSTM.

4.6.2 CNN

CNN (Convolutional Neural Network) is mainly used for computer vison and image processing domain, but it can also apply to NLP problem. FAIRSEQ is a framework mainly based on CNN (Jonas et. al., 2017). This seq2seq model structure can be illustrate in the following figure.

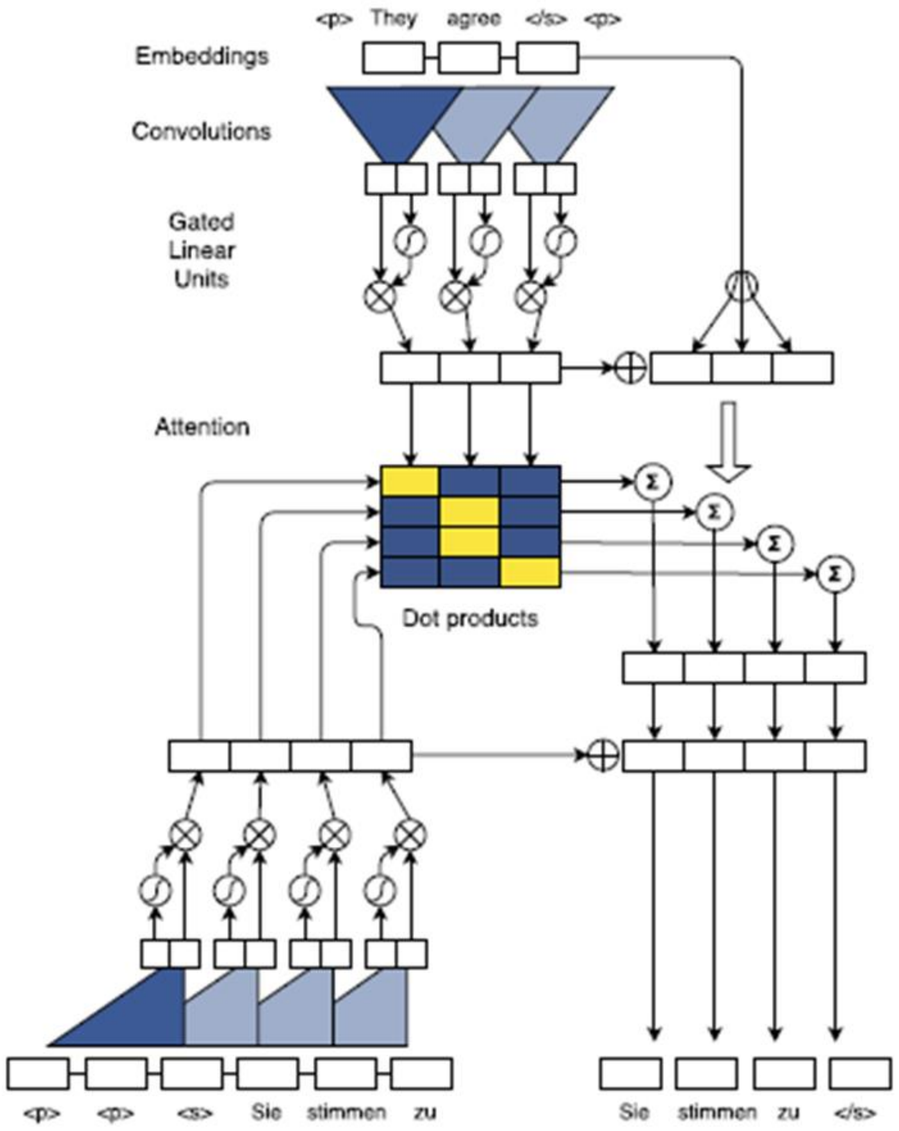


figure 8 (Jonas et. al., 2017)

At the beginning it is the embedding part. The initial word is split by token and word embedding is initialized. Besides word embedding CNN will also add on positional embedding to emphasize the positional information of each word. The encoder part is a stack of N similar encoder structures. Several convolutional filters are used to concentrate the information of the nearby words. After the filters it use a non-linear activation unit. In computer vison they often choose RELU but here fairseq uses GLU (gated linear units) just like LSTM to enhance non-linearity to deal with sequence information. The GLU kernels controls which parts of the outputs from the convolutional kernel are dependent and which parts are independent. Also, residual connection is used to like res-net in CV to reuse the embedding information. The decoder part is a stack of N similar decoder structures. Convolution operation and GLU appear again in decoder part. The model adds attention mechanism for each decoder layer to attend all the decoded words' information when dealing with next word. Finally, target word probabilities are obtained by mapping the outputs from the last layer to the word dictionary. We choose the word with highest probability as our decoded result.

FAIRSEQ is a leading framework of expending the usage of CNN from image processing area to sequence learning such as machine translation. Compared with the traditional sequence structure of RNN, CNN can be easy to compute in parallel instead of dealing with the sequence unit one by one. Besides, it can handle the context information conveniently by stacking filter layers to obtain a large receptive field. Currently CNN based on faireseq framework is better than traditional RNN models in both efficiency and quality.

4.6.3 Error Generation Process

We use our largest dataset lang-8 for error generation purpose. Because there are nearly half sentence pairs that is unchanged in lang-8 (their source and target sentence are the same), we think that it is possible to reuse these sentences. Hence, we decided to use the sentence pairs whose source and target are different to train an error generation model based on CNN structure and use the unchanged sentences as seed corpus for generation purpose.

In building our training dataset, firstly we remove those changed sentence pairs whose length of source and target have a great difference. We only keep the pairs of which the absolute value of length difference between source and target is smaller than 6. Secondly, we remove some pairs whose length of the grammatical correct sentence is too short (shorter than 4 tokens) or too long (longer than 26 tokens). Then we have a corpus with 380 thousand sentence pairs for training and split around 27 thousand sentence pairs for valid. We also randomly choose another 380 thousand unchanged sentences as test set for generation.

After preprocess and binarizing our train, valid and test sets together, we construct a standard CNN model from FAIRSEQ framework for error generation. The dimension of encoder's embedding and decoder's embedding are both 500. There are 7 encoder layers and 7 decoder layers with the size of (1024, 3). And for optimizer setting, we use Nesterov's Accelerated Momentum (NAG) optimizer with parameters momentum 0.99, learning rate 0.25, learning rate shrink 0.1 and drop out rate 0.2. The training process continued for 10 epochs.

Then we use the best checkpoint measured on valid dataset for error generation. The beam size for searching is 5 and we extract the 10 best outputs for each sentence. There is still some post-process work to be done. Because the lang-8 corpus is not much clean, there are some sentences inserted with emoji symbols or irregular punctuation marks. After filtering those sentences, we also remove the too long or too short sentences. For every correct input sentence in the test set, we generate multiple wrong version sentences. We only choose those sentence pairs where the fluency score of the correct sentence divide by the fluency score of the wrong sentence is greater than 1.05 because those sentences have more severe mistakes.

5 Experiments

5.1 Baseline Model

5.1.1 Datasets

As previous introduction, we mainly use four kinds of public dataset (NUCLE, Lang-8, FCE and W&I+LOCNESS) as our parallel training data. For unlabeled dataset, the well-known One Billion Word Benchmark is used while training the pre-trained model. Test set of CoNLL-2014 shared task, test of JFLEG (JHU Fluency-Extended GUG corpus) and test set of Bea-2019 shared task is our test benchmark in order to have a more persuasive result. Besides, we select development set of Bea-2019 shared task and randomly split a big dataset (Lang-8) as our development benchmark. As for our language model, we also use the One Billion Word Benchmark as our training and valid data set.

Just like previous works, we need to do data preprocessing for our dataset. Compared with other dataset, the Lang-8 corpus has too many sentences. Therefore, we remove the unchanged sentence pairs in the Lnag-8 corpus. Also, we train a dictionary with 50,000 words from the Lang-8 data set and the dictionary is our vocabulary set in our training process.

For evaluation part, we will report results on the JFLEG test set (JFLEG), the CoNLL-2014 test set (CoNLL-2014), and the official test set of the BEA-2019 shared task (BEA-2019). All reported results come from checkpoint files which are output of our model training. In our experiment part, we will report the scores measured by ERRANT (Christopher Bryant et.al, 2017; Mariano Felice et.al, 2016) for BEA test set, and CoNLL-2014 data set. Since the reference sentences of BEA test set is not available right now, we could only get evaluation reports of the model outputs by F0.5 the website gives. Since a lot of paper compute results for CoNLL-2014 test set in recent year, we also report F0.5 results measured by the M^2 scorer (Daniel Dahlmeier et.al, 2012) on CoNLL-2014 to compare them with those of previous studies. Besides, we use the GLEU metric for JFLEG test set. Summarily, we mainly focus on F0.5 for CoNLL-2014 test set and GLEU for JFLEG test set.

5.1.2 Model Frame and Training Settings

In this report, our codes are modified from FAIR Sequence-to-Sequence Toolkit (FAIRSEQ) (Myle Ott, 2019). It is a public Transformer implementation frame developed by Facebook AI research and most of its parts are based on PyTorch. It has state-of-the-art implementations of models that have different structures like convolutional neural network, LSTM and Transformer. Also, it offers a lot of pretrained models for different natural language processing tasks, like machine translation, language model and speech recognition. Basically, we use its Python implement as our baseline codes. FAIRSEQ(-py) is distributed with a MIT-licensed. Its source code is available on GitHub at https://github.com/pytorc h/fairseq/.

While training our transformer model, the hidden size of dimension and token embeddings are 512. The transformer structure consists of 6 encoder layers and 6 decoder layers and the number of multi-head attention is 8. A dictionary containing about 50,000 words is used for the input and output tokens. Similar to other paper's setting, we set the dropout to 0.2. In the position-wise feed-forward network, the dimension is 4096.

In optimization part, we mainly use Nesterovs Accelerated Gradient (Yurii E Nesterov, 1983). The learning rate of optimization process is 0.002. For other parameters, the patience is 0, the weight decay is 0.5, the momentum is 0.99 and minimum learning rate is 1e-4. For each epoch, we will evaluate our model's performance based on the development sets.

In decoding part, the beam-size of the final layer is 12 and we will normalize our model scores by length in the end. Also, we will rerank the top 12 outputs based on a combination of language model scores and original model scores. We will talk about this part in the following section.

5.1.3 Baseline Results

All of our models are fitted into 28GB GPU RAM memory across 1 GPU in GPU Farm resources offered by Department of Computer Science in The University of Hong Kong. According to the suggestions of author of FAIRSEQ, we train our model for 9 epochs, starting with the parameters of pre-trained model. Our baseline model is only a Transformer model with a pre-trained copy-augmented architecture model and we did not use language model and other tricks. Finally, our copy-augmented architecture achieves a 58.86 F0.5 score. This is the result without data from BEA-2019 shared task, because when we trained our model in the early stage, the dataset is not available at that time. After adding BEA-2019 training set and development set, our model achieves a 59.76 F0.5 score. For JFLEG testing set, we only achieve a 52.6 score at the beginning. Therefore, this is the baseline results of our model without any tricks. In the following sections, we will report more results with our methods that we have mentioned before.

5.2 Error Generation Experiment Result

After the error generation process we mentioned in section 4.6.3, we got a synthetic dataset accompany with the natural corpus for training our grammar correction model. This synthetic dataset has 527322 pairs of sentences. Through observing with our naked eyes, the quality of this pseudo data is not bad. The mistakes made by our error generation model are similar to those mistakes made by human English learners. In order to present some examples, we list the following table 1 of the first 20 generated sentence pairs. The left column is the source sentences for GEC model whose errors were constructed by our CNN based error generation model. The right column is the target sentences for GEC model which are the unchanged sentences in lang-8 corpus.

	source sentences in synthetic corpus	target sentences in synthetic corpus
1	This song `` For Today `` is one of my favorite song in all her songs .	This song , `` For Today `` is one of my favorite songs in all her songs .
2	I bought it to study english .	I bought it to study English .
3	I can cook 3 different food .	I can cook 3 different styles of food .
4	This year I joined university sports club .	This year I joined my university 's sports club .
5	I went to shopping with my mother , little sisters .	I went shopping with my mother and little sisters .
6	After that , They did sumo match .	After that , they did a sumo match .
7	It is very fun to talk with them and know daily life .	It is very fun to talk with them and know their daily life .
8	We need enjoy company with our colleague ?	Do we need enjoy company with our colleague ?
9	It 's like the difference of two different world .	It 's like the clash of two different worlds .
10	I was wondering why the three of them have to go to restroom together .	I was wondering why the three of them had to go to the restroom together .
11	It is tough , time - consuming , but well worth .	It is tough , time - consuming , but well worth it .
12	I got computer repaired .	I got my computer repaired .
13	I wish I had a more time in my life .	I wish I had more time in my life .
14	I wanna go to America or Europe this summer .	I want to go to America or Europe this summer .
15	I go to friend 's house on Friday .	I go to a friend 's house on Fridays .
16	These days , I 'm so busy .	I 'm so busy these days .
17	I shall try to pass the examination of entrance .	I shall try to pass the entrance examination .
18	I had a cold since last week .	I had a cold last week .

19	It does n't have any window glasses of course air conditioner either .	It does n't have any window glasses and of course air conditioner either .
20	It 's chilly autumn night .	It was a chilly autumn night .

Table 1 : Examples of generated sentence pairs

By joint setting of using the synthetic data, our GEC model reaches a result that 59.28 for F0.5 and 52.15 for GLEU. We analysis the reason that why error generation method doesn't give us the result as good as we expected. The mainly reason is that maybe the choice of seed corpus for error generation is not appropriate. The information of grammar error forms in lang-8 dataset has already been used in the training data. Because we also use part of lang-8 corpus train an error generation model and part of lang-8 corpus as generation seed, the types of errors generated won't go beyond the original scope, thus it causes information redundant. In pre-train setting, model is usually "the more data, the better result", but in joint setting information redundant will cause troubles. In fact, we have also tried to use SimpleWiki as another seed corpus for error generation. The generation result is much poorer than lang-8 seed corpus. There are many words in wiki that do not appear in lang-8 and are replaced by <unk> notation after generation. The model cannot generate words that hasn't been seen during training. So we stop here temporarily for the error generation track.

5.3 Spellchecker+Language Model

As discussed in 4.2, pure spellcheckers can just correct the misspelled words based on itself rather than semantic of the whole sentence, and language model can be used to calculate the fluency or correctness of a sentence. Combination of these two tips can improve the result effectively.

There are several powerful language models with different sizes. Generally speaking, larger language model always has deeper architecture, more parameters and can store more knowledge that how to judge the quality of a sentence, which means it is more likely to perform better. Three kinds of langue models are considered in our experiments, containing wiki103 whose size is around 750MB, adaptive_lm_wiki103_small whose size is around 1.2GB and adaptive_lm_wiki103_big whose size is bigger than 3GB. In view of the memory space (since we still need use other memory to store information of the follow-up steps), the performance of these three models and the computational efficiency, adaptive_lm_wiki103_small is the best. Adaptive_lm_wiki103_small can improve the GLEU of JFLEG from around 52.60 to 60.18 (wiki103 can only improve this value from 52.60 to around 59.9, and adapative_lm_wiki103_big can also improve the value from 52.60 to 60.18 while it need more memory space, which also proves that adaptive_lm_wiki103_small has find almost all misspelling errors, so wiki103 and the bigger one adaptive_lm_wiki103_small is not considered). However, F0.5 of dataset CONLL14 is reduced from 59.76 to 58.84, maybe because some proper nouns are replaced according to the value of langue model which reduce the precision. Since GLEU of JFLEG and F0.5 of CONLL14 are two standards to judge the performance of model in GEC, spellchecker + langue model will be used in GLEU only since it can only improve the performance of GLEU.

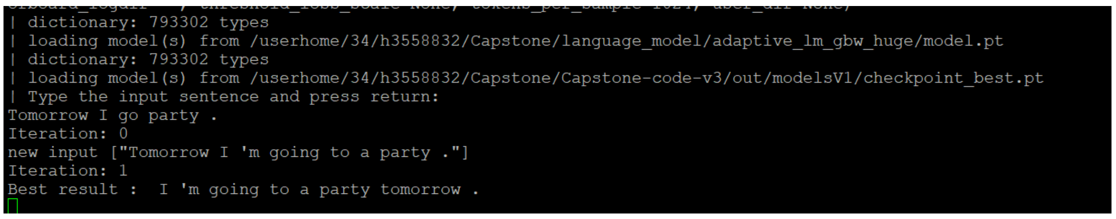
5.3 Re-Rank

Rerank operation combines our model with other language models to improve the effect of Grammatical Error Correction system. When the model calls the parameters that have been trained to predict test data, an n-best candidate set is generated for each input sentence. In the original model, the model outputs the sentences rated highest by the model to calculate F0.5 score and GLEU. But sometimes the model can't find the correct sentence because of some errors. At this point, we need to use the language model to correct the results of the model. We apply the language model to the sort-rerank step. When the model outputs an n-best candidate set of sentences, we retain the scores of those sentences. At the same time, these output sentences are rated by the language model, and the higher the score, the more reasonable and fluent the language model thinks the sentences are. Finally, we weighted the scores calculated by the language model and the results of the model itself to get a new score. Intuitively, there is an order of magnitude difference between what the model calculates and what the language model calculates. Therefore, we set the weights of the scores obtained by the model and the scores obtained by the language model to be 10 to 1.

At first, we download pre-trained WikiText-103 language model and dictionaries to rerank. However, because the language model is not large enough, the result is not very ideal. Scores only improved in the JFLEG GLEU test set. GLEU's score rose to 55.5 from 52.9. But It didn't work for the F0.5 score. The result of F0.5 score decreased from 59.76 to 59.61. In order to get a better effect of the model, we download the pre-trained model from the FAIRSEQ. We combined the scores of the new language model with the scores of the model itself. The score of F0.5 was raised from 59.76 to 60.64.

5.4. Boost

We use the Boost operation in two places, interactive.py and generate.py. First, we applied the boost operation to the interactive interface. We added the second version of the language model to the original interactive python file. Similar to the rerank operation above, we combined the model's calculated score with the language model's score. Each time the user enters a sentence, the model first calculates the scores for the n-best candidate set of sentences. Then the language model starts to load and the score from the model itself is weighted 10 to 1 against the score from the language model. The sentence with the highest score is selected for the next iteration. When the highest score of this iteration is no higher than that of the previous iteration, the iteration ends and the final result is output. To reduce the impact of random errors and avoid too many iterations, we set a threshold for the model. The model will only move to the next iteration if the best result from this iteration is 10% higher than the best result from the previous iteration.



```
| dictionary: 793302 types
| loading model(s) from /userhome/34/h3558832/Capstone/language_model/adaptive_lm_gbw_huge/model.pt
| dictionary: 793302 types
| loading model(s) from /userhome/34/h3558832/Capstone/Capstone-code-v3/out/modelsV1/checkpoint_best.pt
| Type the input sentence and press return:
Tomorrow I go party .
Iteration: 0
new input ["Tomorrow I 'm going to a party ."]
Iteration: 1
Best result :  I 'm going to a party tomorrow .
```

For example, we input the sentence ‘Tomorrow I go party .’ in the interactive interface. After the first iteration, the new input becomes ‘Tomorrow I ‘m going to a party .’ and the final output ‘I ‘m going to a party tomorrow .’ is generated in the second iteration.
We then applied the boost idea to the calculation of F0.5 scores and GLEU scores for the test set. For each sentence in the test set, the model generates an n-best candidate. We boost the n-best candidate set for each sentence. Similar to what we did in the interactive interface, we generated a new list of n-best candidate set for the test set. Boost results in smoother sentences and sometimes changes the order and structure of sentences. But it's not good for the model to calculate the F0.5 and GLEU scores, so the scores calculated by the final model did not improve.

5.5 Rule-based Method Experiment Result

As we mentioned there are two set of rules that we used. Here we just use the second subset for evaluation purpose and leave the first subset only for user interactive mode. The F0.5 score for conll14 rises from 60.64 (reranked by language model track) to 61.13. The GLEU score rises from 60.32 to 60.47. Hence adding appropriate rules is a good compensation for deep learning based model and increase F0.5 and GLEU at the same time because they deal with the problem from different aspects.

It should be mentioned that if we do not use our GEC model and only modify the test set of CoNLL-2014 and JFLEG by rules of language Tool, both F0.5 and GLEU is not good. The offline F0.5 is 28.22 while the offline GLEU is 50.12. That’s a proof that our system is much better than this opensource GEC server evaluated by both F0.5 and GLEU.

5.5 Summary

5.5.1 Result

After adding these tricks and fine-tuning parameters, we improve value of F0.5 and GLEU. Since different standard may be suitable for different tricks, some tricks get good results for one of F0.5 and GLEU while reduce another one. Hence, we list our result on two tables (Table 2 and Table 3) to show respectively useful tricks for these two standards. Details are illustrated above from 5.1 to 5.2. Flowing the tradition of current research in GEC task, the dataset of F0.5 is CoNLL-2014 test set, and the dataset of GLEU is JFLEG test set.

Model	F0.5
Copy-model	58.86
+ Adding BEA-2019 training dataset	59.76
+ Rerank + LM	60.64
+ Rule-based System	61.13

Table 2: F0.5 Result [PS:LM means language model.]

Model	GLEU
Copy-model	52.60
+ rerank + LM	55.50
+ Spellchecker + LM (adaptive_lm_wiki103_small)	60.18
+ Rule-based System	60.47

Table 3: GLEU Result. [PS: LM means language model.]

5.5.2 Example

- Example 1:

Original:

The air quality in an area are affect by many factors such like SO2 level, PM2.5 level, NO2 level and AQI level which is a important index to measure air quality. Air pollution have always been a serious problem in China, especally in the north region. In recent year, the Chinese government has devoted more and more energy to fighting against air pollution and improving air quality. Therefore, it is of great practical signifcance to study the changes of air quality and which factors affect air quality significantly in major Chinese cities in recent years. In our paper, our most important variable is AQI. As the AQI increases, a increasingly large percentage of the population is likely to experience severe adverse health effects. Some investigations shows that the quality of urban air is closely related to the seasons and meteorological conditions. Therefore, in order to explore the influence of temprature, weather and wind force on air quality and to study the changes of air quality over time in different regions, we use python collect daily weather and air quality data from Janurary 1, 2014 to December 6, 2018. We hope our report will gives people a general idea of how China's air quality has change in recent years.

Corrected by our model:

The air quality in an area is affected by many factors such as SO2 level, PM2.5 level, NO2 level and AQI level which is an important index to measure air quality. Air pollution has always been a serious problem in China, especially in the northern region. In recent year, the Chinese government has devoted more and more energy to fighting against air pollution and improving air quality. Therefore, it is of great practical significance to study the changes of air quality and which factors affect air quality significantly in major Chinese cities in recent years. In our paper, our most important variable is AQI. As the AQI increases, an increasingly large percentage of the population is likely to experience severe adverse health effects. Some investigations show that the quality of urban air is closely related to the seasons and meteorological conditions. Therefore, in order to *explore the influence of temprature, weather and wind force on air quality and to study the changes of air quality over time in different regions, we use python to collect daily weather and air quality data from January 1, 2014 to December 6, 2018. We hope our report will give people a general idea of how China's air quality has changed in recent years.*

- Example 2:

Original:
I am glad to get the letter from your. It seem that you has a difficult in learning English, even worse, you want to give up it. I don't think it's good idea. English is one of the most wildly used international language nowdays. It also play a important part in our lives. Business people uses English at international conference and tourists speaks English when they going abroad. What's more, English is used in an large number of movie and books. It's clear that English is becoming more and more importance. so, It's necessary for us to learning English well. I believe you will makes progress as long as you stick to working hard on it. I'm look forward to hearing from you.

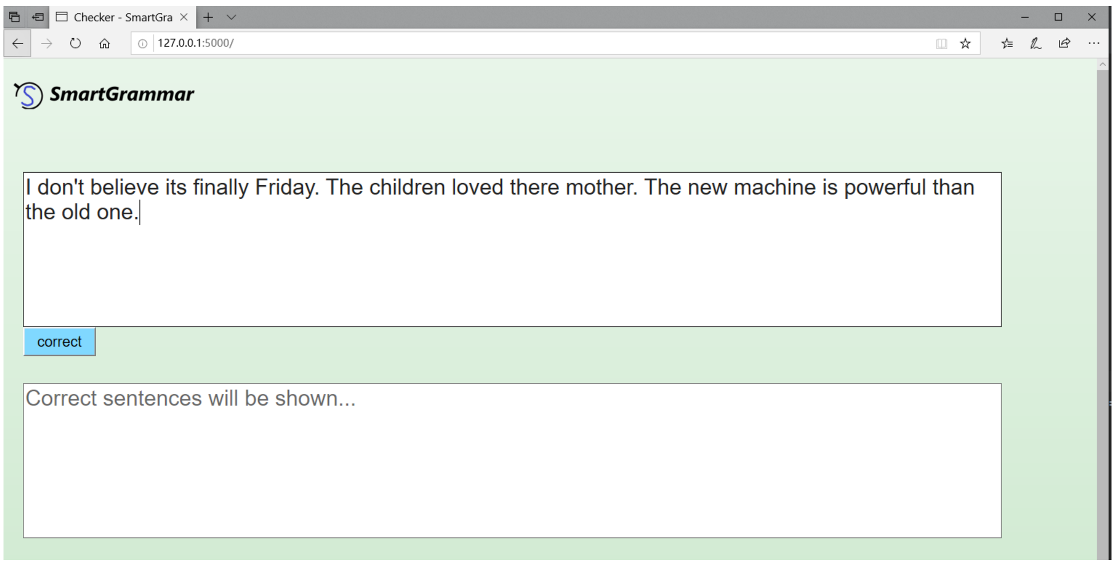
Corrected by our model:
I am glad to get a letter from you. It seems that you have a difficulty in learning English, even worse, you want to give up. I don't think it's a good idea. English is one of the most widely used international languages nowadays. It also plays an important part in our lives. Business people use English at international conferences and tourists speak English when they go abroad. Furthermore, English is used in a large number of movies and books. It 's clear that English is becoming more and more important. So, it 's necessary for us to learn English well. I believe you will make progress as long as you stick to working hard on it. I 'm looking forward to hearing from you.

5.6 Interactive Surface

After finishing training our model, we will create an interactive interface to present what we have done and what our model can do. In fact, FAIRSEQ has provided an interactive python file for us and we will modify it according to our requirement. In this case, we will set some parameters for outputting the correct sentence. We will use the same dictionary to map the words or tokens and check spell for input sentences. Then by setting the number of output sentences to 12, we will achieve a final score by a combination of language model score and model score and select the sentence with the highest score.

Then we build a web page for showing output sentence of our model. Specifically, we create an API for our GEC system, using the lightwork framework Flask. This API will act as an access point for the model across many languages, allowing us to utilize the predictive capabilities through HTTP requests. Also, we have written some HTML codes to prettify our web pages. All of these servers are built on machine from Huawei Cloud Service

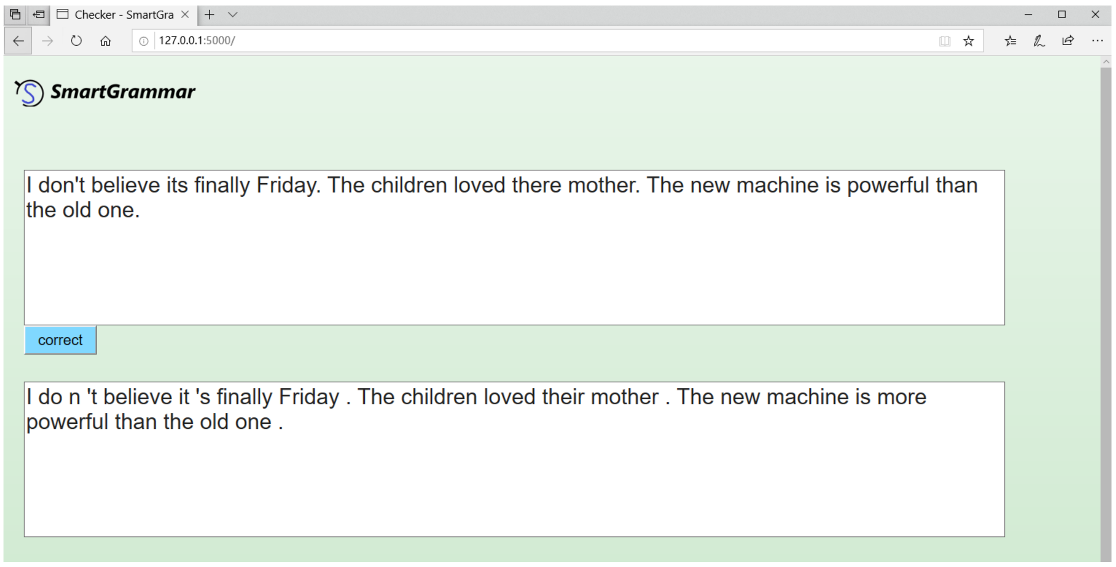
As illustrated in Figure 9, in our interactive interface, users could input a paragraph that need to be corrected. Our interactive python file will split the paragraph into several sentences and correct them one by one. After that, just like Figure 10, the interface will output the correct sentences, merge them into paragraph and print it in the interface. In this way, if users write an article and not sure whether there are sentences with grammar error, they could copy them and paste their paragraph here. Then our model will help them to check the grammar error. It is very convenient for users.



I don't believe its finally Friday. The children loved there mother. The new machine is powerful than the old one.

Correct sentences will be shown...

Figure 9



I don't believe its finally Friday. The children loved there mother. The new machine is powerful than the old one.

I do n 't believe it 's finally Friday . The children loved their mother . The new machine is more powerful than the old one .

Figure 10

6 Conclusion and Future work

In our report, we treat grammar error correction problem as a sequence-to-sequence task and transform it into a natural language process problem. After simplifying our mission, we present a Transformer model with copy-augmented architecture for sequence-to-sequence task. Firstly, we choose Transformer as sequence-to-sequence model because it is popular and perform very well in machine translation task in recent year. Secondly, we add a copy-augmented architecture to our original Transformer model, which is suitable for our GEC problem. The baseline result of this model is proved to be very good. Thirdly, we do a lot of experiments to improve our baseline result, including reranking, adding rule-based method and error generation. In these ways, we achieve an excellent result of F0.5 & GLEU and use well-trained model to solve GEC problem. Finally, we build an interactive interface to present the result of our model. It allows users to try to use our GEC system and help them correct their paragraph.

However, the GEC problem is very complex, there is a long way to go to build a reliable GEC system. If we want our automatic GEC systems to beat humans, it will take a long time. Therefore, there are still several future directions following our systems. The most important thing is to collect more sentence pairs. Because it always takes a long time to invite several experts to correct articles and then make a new dataset. It is very difficult to collect data for GEC task and we have to add unlabeled dataset in our project. Also, trying to extend sentence-level GEC systems to paragraph-level contexts is a good direction. It is easier to deal with complex semantic errors like collocation, if we could include previous sentences. In the W&I+LOCNESS dataset which is actual a collection of multi-paragraph essays, adding multi-sentence contexts may help us improve our GEC model's performance.

Reference:

[1] Daniel Naber, 2003. A Rule-Based Style and Grammar Checker.
[2] Felice, M., Yuan, Z., Andersen, Ø., Yannakoudakis, H., & Kochmar, E. 2014. Grammatical error correction using hybrid systems and type filtering. Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task, 15-24.
[3] Diane Nicholls, 2003. The Cambridge Learner Corpus: Error coding and analysis for lexicography and ELT. In proceedings of the Corpus Linguistics 2003 conference, pages 572-581.
[4] Alla Rozovskaya and Dan Roth. 2016. Grammatical error correction: Machine translation and classifiers. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2205–2215
[5] Shamil Chollampatt and Hwee Tou Ng. 2017. Connecting the dots: towards human-level grammatical error correction. In Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications, pages 327–333
[6] Zheng Yuan, Felix Stahlberg, Marek Rei, Bill Byrne, Helen Yannakoudakis: Neural and FST-based approaches to grammatical error correction. BEA@ACL 2019: 228-239
[7] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Robert L. Mercer, 1993, The Mathematics of Statistical Machine Translation: Parameter Estimation. Computational Linguistics, Volume 19, Number 2, June 1993, Special Issue on Using Large Corpora: II, pages 263-311.
[8] Michel Galley, Christoper D.Manning, 2008, A Simple and Effective Hierarchical Phrase Reordering Model. 2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A

meeting of SIGDAT, a Special Interest Group of the ACL.
[9] Haitao Mi, Liang Huang, and Qun Liu. 2008. Forest-based Translation. In Proceedings of ACL 2008.
[10] Min Zhang, Hongfei Jiang, Aiti Aw, Haizhou Li, Chew Lim Tan, and Sheng Li. 2008. A Tree Sequence Alignment-based Tree-to-Tree Translation Model. In Proceedings of ACL 2008.
[11] Yang Liu, Yajuan Lü, and Qun Liu. 2009. Improving Tree-to-Tree Translation with Packed Forests. In Proceedings of ACL/IJNLP 2009.
[12] David Chiang, 2010, Learning to Translate with Source and Target Syntax. In proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 1443-1452.
[13] Zheng Yuan and Ted Briscoe. 2016. Grammatical error correction using neural machine translation. Published in HLT-NAACL 2016 DOI:10.18653/v1/n16-1042.
[14] Allen Schmaltz, Yoon Kim, Alexander Rush, and Stuart Shieber. 2017. Adapting sequence models for sentence correction. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 2807–2813.
[15] Jianshu Ji, Qinlong Wang, Kristina Toutanova, Yongen Gong, Steven Truong, and Jianfeng Gao. 2017. A nested attention neural hybrid model for grammatical error correction. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 753-762.
[16] Roman Grundkiewicz and Marcin Junczys-Dowmunt, 2018, Near Human-Level Performance in Grammatical Error Correction with Hybrid Machine Translation. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), pages 284–290, New Orleans, Louisiana, June, Association for Computational Linguistics.
[17] Graham Neubig, Makoto Morishita, Satoshi Nakamura, Neural Reranking Improves Subjective Quality of Machine Translation: NAIST at WAT2015. In Proceedings of Proceedings of the 2nd Workshop on Asian Translation (WAT2015), pages 35-41.
[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. arXiv preprint arXiv:1706.03762.
[19] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One-billion word benchmark for measuring progress in statistical language modeling. arXiv preprint arXiv:1312.3005.
[20] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016 Pointer sentinel mixture models. arXiv, preprint arXiv:1609.07843.
[21] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. arXiv preprint arXiv:1603.06393.
[22] Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. arXiv preprint arXiv:1606.03622.
[23] Wei Zhao, Liang Wang, Kewei Shen, Ruoyu Jia, Jingming Liu, 2019, 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.
[24] Mengyang Qiu, Xuejiao Chen, Maggie Liu, Krishna Parvathala, Apurva Patil, Jungyeul Park,

2019, Improving Precision of Grammatical Error Correction with a Cheat Sheet. In proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications, pages 240-245.

[25] Shun Kiyono, Jun Suzuki, Masato Mita, Tomoya Mizumoto, 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.

[26] Alec Radford, Karthik Narasimhan, Tim Salimans, Jakob Uszkoreit, and Ilya Sutskever. 2018. Improving language understanding by generative pretraining. OpenAI Blog.

[27] Julia Shaptala, 2019, Multi-headed Architecture Based on BERT for Grammatical Errors Correction. In proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications, pages 246-251.

[28] Dahlmeier, Daniel & Ng, Hwee. 2012. Better evaluation for grammatical error correction. 568-572.

[29] Courtney Napoles, Keisuke Sakaguchi, Matt Post, Joel Tetreault, 2015, Ground Truth for Grammatical Error Correction Metrics. In proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), pages 588-593.

[30] Kishore Papineni, Salim Roukos, Todd Ward, Wei-Jing Zhu, 2002, BLEU: a Method for Automatic Evaluation of Machine Translation. In proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pages 311-318.

[31] Daniel Dahlmeier, Hwee Tou Ng, 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.

[32] Toshikazu Tajiri, Mamoru Komachi, 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.

[33] Helen Yannakoudakis, Ted Briscoe, 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.

[34] Christopher Bryant, Mariano Felice, Øistein E. Andersen, 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.

[35] Sylviane Granger, 1998, The computer learner corpus: A versatile new source of data for SLA research.

[36] Hwee Tou Ng, Siew Mei Wu, Yuanbin Wu, Christian Hadiwinoto, Joel Tetreault , 2013, The CoNLL-2014 Shared Task on grammatical error correction. In proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task, pages 1-12.

[37] Courtney Napoles, Keisuke Sakaguchi, Joel Tetreault, 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.

[38] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, pages 1096–1103.
[39] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and KristinaToutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
[40] Ebru Arisoy, Tara N. Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. 2012. Deep Neural Network Language Models. In NAACL-HLT Workshop on the Future of Language Modeling for HLT.
[41] Vaswani, A. & Zhao, Y. & Fossum, V. & Chiang, David. 2013. Decoding with large-scale neural language models improves translation. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. 1387-1392.
[42] Baltescu, Paul & Blunsom, Phil. 2014. Pragmatic Neural Language Modelling in Machine Translation. 10.3115/v1/N15-1083.
[43] Rafal J´ozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. arXiv, preprint arXiv: 1602.02410.
[44] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2018. Character-level language modeling with deeper self-attention. arXiv, preprint arXiv:1808.04444.
[45] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. arXiv, preprint arXiv: 1612.08083
[46] Kneser, Reinhard and Ney, Hermann. 1995. Improved backing-off for m-gram language modeling. In Acoustics, Speech, and Signal Processing, ICASSP-95., 1995 International Conference on, volume 1, pp. 181–184.
[47] Chen, Stanley F and Goodman, Joshua. 1996. An empirical study of smoothing techniques for language modeling. In Proceedings of the 34th annual meeting on Association for Computational Linguistics, pp. 310–318.
[48] Alexei Baevski and Michael Auli. 2019. Adaptive Input Representations for Neural Language Modeling. In Proceedings of International Conference on Learning Representations.
[49] Edouard Grave, Armand Joulin, Moustapha Ciss´e, David Grangier, and Herv´e J´egou. 2017. Efficient softmax approximation for gpus. In Proc. of ICML.
[50] Tao Ge, Furu Wei, Ming Zhou, 2018, Reaching Human-level Performance in Automatic Grammatical Error Correction: An Empirical Study, arXiv preprint arXiv:1807.01270
[51] Martin Popel and Ondˇrej Bojar. 2018. Training tips for the transformer model. The Prague Bulletin of Mathematical Linguistics, 110(1):43–70.
[52] Marcin Junczys-Dowmunt, Tomasz Dwojak, and Hieu Hoang. 2016. Is neural machine translation ready for deployment? A case study on 30 translation directions. In International Workshop on Spoken Language Translation IWSLT.
[53] Gehring, Jonas and Auli, Michael and Grangier, David and Yarats, Denis and Dauphin, Yann. (2017). Convolutional Sequence to Sequence Learning. arXiv preprint arXiv: 1705.03122
[54] 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 (ACL 2017), pages 793–805.
[55] Mariano Felice, Christopher Bryant, and Ted Briscoe. 2016. Automatic Extraction of

Learner Errors in ESL Sentences Using Linguistically Enhanced Alignments. In Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 825–835.
[56] 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 (NAACL 2012), pages 568–572.
[57] Myle Ott, Sergey Edunov,Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, andMichael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In Proceedings of NAACL-HLT 2019: Demonstrations.
[58] Yurii E Nesterov. 1983. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In Dokl. Akad. Nauk SSSR, volume 269, pages 543–547.