Multi-headed Architecture Based on BERT for Grammatical Errors Correction

Julia Shaptala
WebSpellChecker LLC
julia@webspellchecker.net

Bohdan Didenko
NLP Research, WebSpellChecker LLC
bogdan@webspellchecker.net

Abstract

During last years we have seen tremendous progress in the development of NLP-related solutions and area in general. It happened primarily due to emergence of pre-trained models based on the Transformer (Vaswani et al., 2017) architecture such as GPT (Radford et al., 2018) and BERT (Devlin et al., 2019). Fine-tuned models containing these representations can achieve state-of-the-art results in many NLP-related tasks. Given this, the use of pre-trained models in the Grammatical Error Correction (GEC) task seems reasonable.

In this paper, we describe our approach to GEC using the BERT model for creation of encoded representation and some of our enhancements, namely, "Heads" are fully-connected networks which are used for finding the errors and later receive recommendation from the networks on dealing with a highlighted part of the sentence only. Among the main advantages of our solution is increasing the system productivity and lowering the time of processing while keeping the high accuracy of GEC results.

1 Introduction

Modern state-of-the-art GEC models use the sequence-to-sequence (seq2seq) approach and Transformer Encoder-Decoder architecture (Ge et al., 2018). The core idea of seq2seq approach for GEC is the following: tokens from the source sequence are sent to the model input, and a similar sequence without errors is expected as an output. Transformer Decoder is auto-regressive, meaning that it predicts tokens one by one. Though this approach can represent the following challenges: (i) the sequence is reconstructed entirely, regardless of errors number; (ii) sentences are processed at low speed during inference; (iii) errors tend to accumulate since a failure in prediction of a single token can lead to a rupture of the entire chain in the network.

In this paper, we suggest an alternative approach for GEC with "Multi-headed" architecture that uses BERT as Encoder and specialized "Heads" networks enabling additional text processing based on particular error types. In addition, particular Heads let us discover the error placement and come out with error correction. When we can create an effective dictionary for different types of errors suggested in ERRor ANnotation Toolkit (Bryant et al., 2017), such Heads as Punctuation, Articles and Case will be used. Otherwise, if we cant create an effective dictionary, we are going to use a special **"highlight and decode" technique** in a bundle with Transformer Decoder to suggest a correction.

Also, we used Boosting Approach (Ge et al., 2018) as an auxiliary step to improve the GEC within the framework of this competition.

2 Data and Text Pre-processing

The data sets which we used for the network training were in the m2 format (Dahlmeier and Ng, 2012). This data obviously has its issues; not all the data sets can be considered the perfect ones and may require pre-processing before they can be used for neural networks training. Thus, before using given data sets we performed a number of operations to filter out irrelevant data and improve its quality by simplifying its form. The main problem of such data format is that each edit made is recorded separately, and it is not possible to display the related changes.

The data and text pre-processing phases are described below.

Phase 1. Adjusting form of the information in data sets (by combining related changes). Below is an example of a sentence in m2 format which

Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications, pages 246–251
Florence, Italy, August 2, 2019.

displays our approach to grammatical errors correction:

S I think that you have to bring with you winter clothes because here there is a really cold weather !
A 7 11|||R:OTHER|||winter|||REQUIRED|||-NONE-|||0
A 11 11|||M:OTHER|||clothes with you|||REQUIRED|||-NONE-|||0
A 13 14|||R:OTHER|||it|||REQUIRED|||-NONE-|||0
A 15 16|||U:DET||||||REQUIRED|||-NONE-|||0
A 18 19|||U:NOUN||||||REQUIRED|||-NONE-|||0
Result - I think that you have to bring winter clothes with you because here it is really cold !

As you can see, the related changes in the sentence are divided into a number of edit operations U (Unnecessary), M (Missing) and in some cases R (Replacement), M, and even R, R. To combine related changes, we find $R \cap I$ where R removed tokens, I inserted tokens from all edits. In addition, we have combined edits with a non-zero intersection into one edit. As a result, we get an example with only one edit which is MOVE.

S I think that you have to bring with you winter clothes because here it is really cold !
A 7 11|||MOVE|||winter clothes with you|||REQUIRED|||-NONE-|||0

Phase 2. Using Textual Semantic Similarity (Yang and Tar, 2018) analysis to filter noisy data. For example, to filter noise in the data like this:

S It was very spicy .
A 0 1|||R:OTHER|||Delete|||REQUIRED|||-NONE-|||0
A 1 4|||R:OTHER|||this sentence|||REQUIRED|||-NONE-|||0

Textual Semantic Similarity analysis was used to define the similarity between a source sequence and a sequence after applying corrections and discarded the sentences with the similarity below 0.87.

The original sentence containing a mistake is a vector as well as the meaning of a corrected sentence. Textual Semantic Similarity is calculated using the scalar multiplication of vectors (vector size equals 512), each of them is output of the Universal Sentence Encoder[1]. As a result we have one number ranging from 0 to 1 which is the ratio of semantic similarity of the two sentences. The higher the scalar multiplication number is, the higher Textual Semantic Similarity of the two sentences.

After we have processed 600K sentences from the data sets used for this competition[2], we realised that most part of sentences before the number of 0.87 are not acceptable for usage and change the meaning or not valid at all.

Thus, our assumption is that the sentences that equal 0.87 and above are usable, and we will train our model on it. All the other sentences are filtered as noise as in the example in m2 format above.

Phase 3. Flattening the data by extending the number of sentences for training. Our next step is to enlarge the amount of data for training and convert the sentence with N edits to N sentences with one edit. Conventionally, we called it "flatten m2 blocks".

Example below represents a sentence in m2 format with 2 edits: we replace the verb (R:VERB:SVA) and add missing adjective (M:ADJ). As a result we have two sentences with one edit, one for a replaced verb (R:VERB:SVA) and the second for an added missing adjective (M:ADJ).

Example of the original sentence in m2 format:

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

Result sentence after the first edit:

S This are a sentence .
A 1 2|||R:VERB:SVA|||is|||-REQUIRED-|||NONE|||0
Result - This is a sentence.

Result sentence after the second edit:

S This is a sentence .
A 3 3|||M:ADJ|||good|||-REQUIRED-|||NONE|||0
Result - This is a good sentence.

Our assumption is that one epoch (or the process of training of a neural network) on the "flatten" of data should have a better result than a few epochs on the original data and reduce the effect of network overfitting.

3 The Model

The main architectural advantage of our approach is using trained "Heads". Heads are the fully-connected networks that receive the BERT output

[1] https://tfhub.dev/google/universal-sentence-encoder/2

[2] https://www.cl.cam.ac.uk/research/nl/bea2019st/

result embedding as input and have an output of the Head dictionary size. Each Head is classified by error type given in Errant Error Type Token Tier (Bryant et al., 2017).

We distinguish the following Heads types depending on their usage and based on their context:

- By the type of operation: Replace, Insert, Range Start and Range End;
- By the type of error: Punctuation, Articles, Case, Noun Number, Spelling, Verbs;
- By the type of correction method: ByDictionary (Punctuation, Articles, Case), ByDecoder (Noun Number, Spelling, Verbs). Output of ByDictionary Heads will be a suggestion from the dictionary. Output of ByDecoder Heads which only detect errors positions will be represented as a "Head type mask" (e.g. Spelling Head mask). For example, Punctuation offers suggestions from its dictionary while Verbs points the place of the error to generate a suggestion by Decoder.

Figure 1 below outlines the number of the parameters of each Head. The dark grey color represents the output which is processed by Decoder, and light grey - the results provided from a Head dictionary.

Error Types / Operations	Punctuation	Articles	Case	Noun Number	Spelling	Verbs
Replace	BES * PDS	BES * ADS	BES * CDS	BES * HDS		
Insert	2 * BES * (PDS - 1)	2 * BES * (ADS - 1)	-	-	-	-
Range Start	BES * RDS					
Range End	BES * RDS					

Figure 1: Number of parameters for each Head type.

The following Head dictionary sizes are used: BERT embedding size (BES) 768; Punctuation dictionary size (PDS) 36; Articles dictionary size (ADS) 5; Case dictionary size (CDS) 3; Highlighting dictionary size (HDS) 2; Range dictionary size (RDS) 2. RDS is applicable for Range Start and Range End Hands. The size of the dictionary for both equals 2; one for *skip* and the other for *start position* or *end position* accordingly. Additionally, for the Insert operation, Delete is eliminated action, thus, we use "-1".

Since a BERT output is the encoded representation of each token from the input sequence, Heads analyze each token from the BERT output, detect an error in it and depending on its type, either immediately provide a correction or highlight this error position for further correction by the Decoder as shown in Figure 2 below.

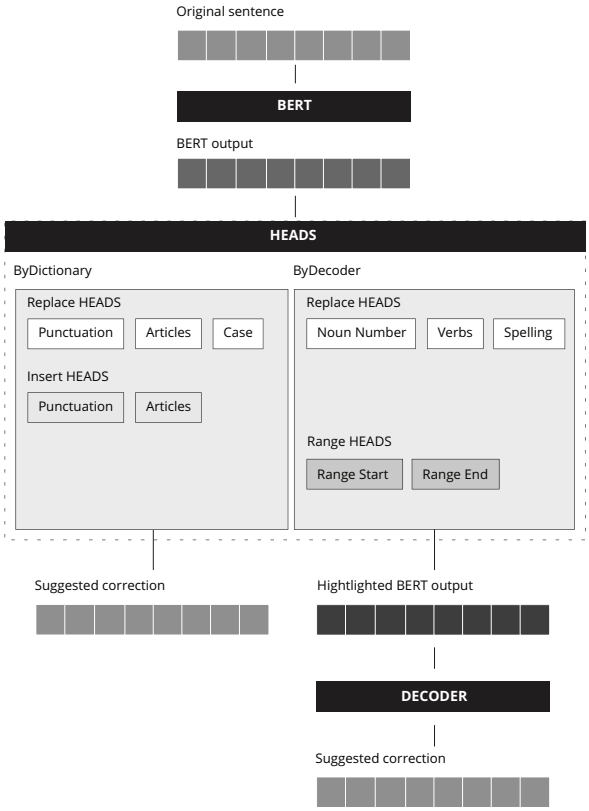

Figure 2: The Multi-headed model architecture.

Also, Heads networks are distinguished by the type of the operation performed such as Replace and Insert. Replace Heads are the Heads performing the Replace operation, and it can either provide a suggestion from its dictionary (ByDictionary), or provide a Head type mask for further processing by the Decoder (ByDecoder) as shown in Figure 3 below.

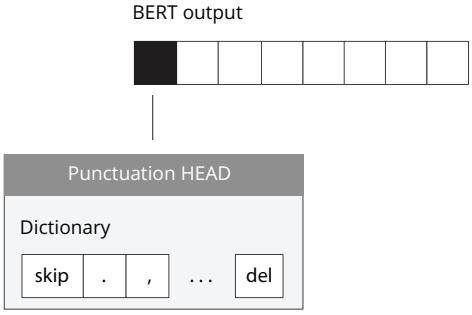

Figure 3: Example of the Replace operation.

During the Insert operation, an Insert Head takes two BERT output embeddings which have

the dimension of 768 located nearby, concatenates to one embedding with dimension 2*768, processes it and outputs the result with the dimension which equals the dictionary size of a particular Head type.

Thus, we have probability distribution of a particular Head. Position with highest probability in a dictionary is what should be inserted. If the probability equals 0, nothing should be done. An example of the Insert operation is shown in Figure 4 below.

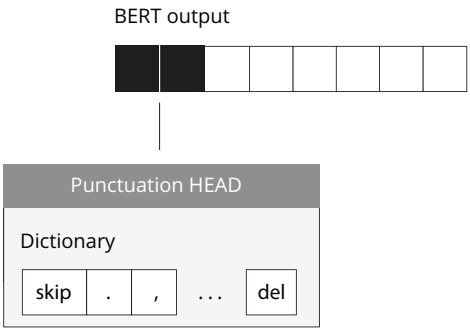

Figure 4: Example of the Insert operation.

Range Heads, Range Start, and Range End are used to define the range (start and end position) of an error for the Decoder. Each Range Head uses an approach similar to the Replace ByDictionary Head, thus, the length of its dictionary equals 2. As an output from two Heads, we receive Range Start mask and Range End mask. Using these masks we receive a resulting Range mask that will be used in the highlight and decode technique as shown in Figure 5 below. Thus, Range Head enables detection of those parts of the sentence which need to be either replaced or paraphrased.

Figure 5: Example of the Range Start and Range End operation.

4 Highlight and Decode Technique

Since there are different types of errors, and it is not possible to compile effective dictionaries as the number of correction options is too large, we used classic Transformer Decoder (Vaswani et al., 2017) and the entire BERT vocabulary. We developed a special "highlight and decode" technique to generate a suggestion for a particular place, determined by one of the Heads, and, thus, managed to avoid the reconstruction of the entire sentence (see Figure 6 below).

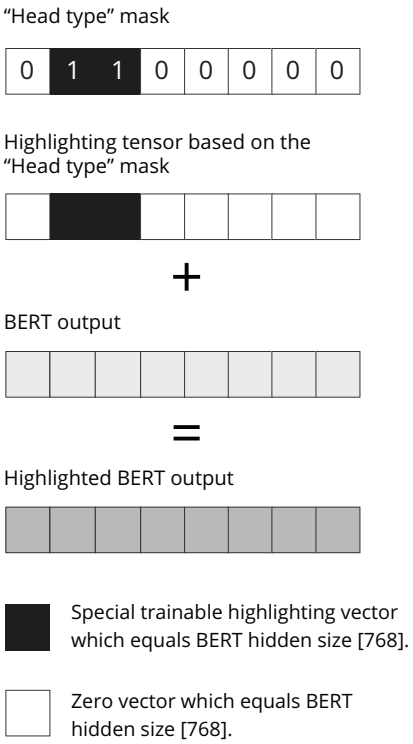

Figure 6: Obtaining of the highlighted BERT output.

The highlighted BERT output, a Decoder input, in Figure 6 above is a summary of the BERT output and the highlighting tensor, consisting of special embeddings (based on Head type mask) in place of errors detected by one of the ByDecoder Heads (such as Spelling), and zero vectors in other places. Such approach allows the Decoder to learn how to predict a suggestion only for the highlighted place in the sentence. The various types of Heads and "highlight and decode" technique let the network find and offer suggestions for any error types.

5 Training Process and Setup

We trained our neural networks using Google Colab TPU resources. A total of 100,000 iterations

were performed on "flatten" data from the Cambridge English Write & Improve (W&I) corpus and the LOCNESS corpus dataset[3]. The learning rate *5e-5* which is recommended in the BERT approach (Devlin et al., 2019) was implemented. However, for the layers of the BERT itself, a layer-by-layer multiplier was used for the learning rate which decreases from the last layers to the first. We calculated the learning rate of a specific layer using the logarithmic formula:

$$lr_i = LR * \log_2\left(1 + \frac{1}{(BL + 1) - i}\right),$$

where BL is number of the BERT layers; LR is model learning rate, e.g.: 5e-5.

It helped us to manage the accuracy of the results adjusting their weights, thus, helping to sort out the errors and improving the results quality by 15% according to our empirical observations.

Also, for each Head of the Replace operation, a special "protection mask" was used to reveal an error only for tokens that can be changed by the given Head. The approach which is shown in Figure 7 below the was used to create a protection mask (for details, see the Spacy library [4]).

Figure 7: The protection masks examples for three Head types, namely Punctuation, Articles, and Verbs.

Unlike the Replace operation, the protection masks are not used for the Insert operation as it is equal to a protection mask with all values equaling 1. Thus, Insert can be done to any place between the tokens.

6 Post Processing and Model Output

At the inference stage, iterative sentences corrections were applied. Each sentence passes through the model, and we get the probability distribution for each Head as an output. During each iteration, the Head with the highest "confidence rate" is chosen from all the Heads as the code below shows:

max_class = argmax(prob) confidence_rate = prob[max_class] if max_class != 0 else 0. # Index 0 means skip in all dictionaries.

Similar to the training stage, the probabilities for the Replace operation are multiplied by the protection mask. The edit proposed by the Head with maximal confidence rate is applied to the sentence, preliminary saving it to the history of previous changes. The process is looped until the following conditions are met: (i) probabilities of edits in all Heads reach zero (0), e.g. all errors have been fixed; (ii) length of the history is more than ten (10) meaning the network tried to improve the original sentence more than 10 times.

Also during each stage, we calculate Textual Semantic Similarity between the current version and the original sentence. This is also a part of our architecture concept. If the similarity is below 0.87, the loop stops, and we use the most recent sentence from the iterations history. Thus, we intended to perform the most effective correction for all grammatical errors in a sentence.

7 Concept Analysis and Roadmap

We have achieved the following results[5] within the framework of BEA 2019 competition. Let us now summarize the main challenges we faced when developing the suggested concept:

- Each Head type has a different learning speed due to different sizes and quality of dictionaries. When some Heads have not been trained yet, others start overfitting. For example, Spelling, Articles, and Punctuation Heads were trained faster than the Range Head and the Decoder itself. Thus, the results have worsened.

[3] https://www.cl.cam.ac.uk/research/nl/bea2019st
[4] https://spacy.io/
[5] https://competitions.codalab.org/my/competition/submission/563950/detailed_results/

- All Heads work independently. This is an issue for sentences where errors depend on each other, for example, in a sentence where the tense of one verb relies on the tense of another one. In the approach proposed in this article, each Head gives the probability of an error without taking into account the probabilities for other Heads in other networks. The same is true for the suggestion prediction. Thus, all results should be revised, and assessment should be made.

- The Decoder learned to predict the "End Of Sequence" (EOS) token as the first one to remove the token. Since EOS is the most frequently encountered token, position of the maximum probability on the Decoder prediction was often EOS. As a result, our solution has mistakenly eliminated tokens from the sentence, thus, lowering the quality of neural network and final output result .

To address the above-mentioned issues, we plan the following changes for our proposal:

- Choosing a unique learning rate for each Head separately. A different approach to consider in our case is to freeze the change in Head weights after it reaches the maximum accuracy for the validation dataset.

- Redesigning the architecture so that the Heads can share information among themselves.

- Using a separate token for deletion, as an option to use one of [unused1-100] tokens from the BERT vocabulary. According to our research and test results, it can improve the accuracy in two times.

References

Christopher Bryant, Mariano Felice, and Ted Briscoe. 2017. Universal sentence encoder. *ERRor ANnotation Toolkit: Automatically extract and classify grammatical errors in parallel original and corrected sentences*.

Daniel Dahlmeier and Hwee Tou Ng. 2012. Better evaluation for grammatical error correction. *In Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805v2 [cs.CL] 24 May 2019*.

Tao Ge, Furu Wei, and Ming Zhou. 2018. Reaching human-level performance in automatic grammatical error correction: An empirical study;. *Microsoft Research Technical Report*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Jakob Uszkoreit, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *OpenAI Blog*.

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:1706.03762v5 [cs.CL] 6 Dec 2017*.

Yinfei Yang and Chris Tar. 2018. Advances in semantic textual similarity. *Google AI Blog*.