Grammatical Error Detection

CS 221 Spring 2021 Project Report

Aditi Jain
ajain4@stanford.edu

Chenhao Li
chenhaol@stanford.edu

I. Abstract

In our final project, we apply AI to the task of identifying grammar errors in text. Our work is unique as it compares and contrasts probabilistic language models with learned neural networks. In the former, we find that these probabilistic models perform well at flagging local grammar errors such as spelling errors and subject-verb agreement but fail to identify larger context grammar errors like noun form agreement. In the latter, we focus our work on the word embeddings which power the learned grammar error detection task. While most models in this space focus on decoder architecture and choose BERT as their encoder, they do not discuss how and why BERT outperforms other embedding methods like GPT and ELMo.

II. Problem Statement

Grammatical error correction (GEC) is the task of correcting text with errors in spelling, punctuation, grammar, or word choice. GEC is popular in industry, with software such as Grammarly, Google Docs, and Gmail helping us compose grammatically correct content. Grammatical Error Correction is also well-researched; CoNLL (Conference on Computational Natural Language Learning) and BEA (Building Educational Applications) have made shared task GEC datasets available for training models to correct grammar errors.

For our project, we simplify the problem of correction to the problem of detection: Rather than providing the exact word additions/deletions/modifications, can we simply identify whether or not a sentence has a grammar error? We hypothesize that this is an easier problem to solve while still providing significant value.

III. Literature Review

This webpage chronicles the progress in the Grammatical Error Correction subfield of NLP. We will briefly touch on a few approaches to GEC so readers can get a feel for the methods. Note that the methods we implemented in this paper are geared towards grammar error detection so while we share datasets with the following approaches, the final evaluation metric is different.

Mentioned above, the CoNLL-2014 shared task test set [1] is referenced in almost all Grammar Error Correction research. It contains 1,312 English sentences (1176 with errors, 136 without) with error annotations by 2 expert annotators. For example, the input sentence might be "She see Tom is catched by policeman in park at last night." and the corresponding annotated output would be "She saw Tom caught by a policeman in the park last night." In GEC literature, researchers use a metric called called MaxMatch, or M^2, first defined in "Better Evaluation for Grammatical Error Correction" [2]. At a high level, MaxMatch efficiently computes a minimum sequence of phrase-level edits between the proposed sentence and expert-annotated sentence, and scores the sequence using $F_{0.5}$ measure (more weight on precision, less on recall).

"Reaching Human-Level Performance In Automatic Grammatical Error Correction: An Empirical Study" [3] composes "fluency boosting learning" with a standard seq2seq framework. By forcing the model to learn not only error-correcting but also learn how to boost a sentence's fluency with more training data, they achieved state of the art performance on the M^2 metric on the CoNLL-2014 test set ($F_{0.5}$ = 75.72).

"GECToR – Grammatical Error Correction: Tag, Not Rewrite" [4] uses a Transformer encoder pre-trained on synthetic data. The authors fine-tuned the model in two stages, first on an errorful corpora and then on a mixed errorful error-free corpora, achieving $F_{0.5}$ of 65.3.

In "Grammatical Error Detection using Error-and Grammaticality-Specific Word Embeddings" [5], the authors constructed a specialized word embedding for the GEC task by training a model that considers the grammaticality of the target word, based on error tags from a GEC corpus.

In our approaches, we borrow ideas from these papers, specifically synthetic training datasets, encoder-decoder architecture, and custom embeddings, but finally report on the fraction of sentences classified correctly, rather than the MaxMatch score over word sequences.

IV. Intro to the Two Approaches in this Paper

For the remainder of the paper, we will deep dive two different approaches to the Grammar Error Detection problem and individually detail their training datasets, baselines, algorithms, and results. Aditi Jain design and implemented the Bayesian language model, while Chenhao Li studied how encoder embeddings inform the accuracy in a neural network setting.

V. Grammatical Error Detection via Bayesian Language Models

A. Main Approach

At a high level, given a sentence, we will compute a score based on bigram probabilities. If this score falls below some threshold, we will classify the sentence as having a grammar error, else the sentence is valid. This approach is implemented in `Bayesian/model.py` in our GitHub repository.

B. Dataset

A large language dataset is crucial to the Bayesian approach. If, at inference time, a word was unseen while building the language model, the model will simply output a zero probability of the 2-sequence of tokens. While we could get around this by introducing Lagrange Smoothing with $\lambda = 1$, we opted not to go this route as we want our approach to identify spelling errors. We will describe this tradeoff in more detail later on.

To compute the bigram model, we leveraged the following datasets, building a singular language model in the form of conditional frequency distributions mapping a word to all possible subsequent words and their frequencies.

We first tried using the Brown [6] corpus available via NLTK. It contains a total of 57k sentences and 1M words across 500 sources of text. The topic include news, reviews, religion, fiction, and more.

Upon realizing that most of the errors were coming from unseen words, we added two more datasets. The Reuters news document corpus [6], also available via NLTK, adds 100k more sentences and 1.3M words. This was also not enough; words like "illustration" and "computational" were not showing up during training. Thus, we decided to inject 1M sentences from the PIE [7] repository. We had found this paper, "Parallel Iterative Edit Models for Local Sequence Transduction" (PIE), while doing background research.

C. Baseline

The first version, or baseline, simply required all bigrams in the inference sentence to have been seeing during training. This, as expected, led to predicting that every single sentence has a grammar error.

D. Algorithm Details

In this section, we share the full details of the Bayesian approach.

1) Training: As described above, we construct a conditional frequency distribution from three large text corpora. This distribution stores the frequencies of each `nextWord` given the `currWord` and updates this distribution for each bigram observed in the corpus. We add a start of sentence character: "⟨s⟩" to ensure that the probability of ("⟨s⟩", "The") is high for example while ("⟨s⟩", "me") is near zero.

2) Inference: During inference, we convert the sentence to a list of bigrams, again appending the start-of-sentence character. For each bigram, we convert the frequency distribution associated with the `prevWord` to a Maximum Likelihood Estimate, modeling the probability P(`nextWord`|`prevWord`). We then threshold this probability; if a sentence contains a bigram with MLE score below the threshold, we predict that the sentence has a grammar error. The threshold we used was 10^{-7} and was decided through manual observation. This threshold can't be too high that we classify (correct) rarely occurring bigrams as being an error or too low that we classify bigrams that are rarely correct (e.g. "consists in" vs "consists of") as not having an error.

E. Results and Error Analysis

We will walk through the results first qualitatively with examples and then quantitatively via test set accuracy. It will be most meaningful to describe the most common types of grammar errors and then discuss the efficacy of the model for each type.

1) Spelling Mistakes: Good

We observe, and it is intuitive, that the model can catch spelling mistakes, because misspelled words have near 0 probability of occurring in the training corpora. For example:

She was frrightened.
(⟨s⟩, she) → 0.00796
(she, was) → 0.11944
(was, frrightened) → 0.0

This approach can also catch missing punctuation, assuming the resulting word is misspelled:

Shes not happy.
(⟨s⟩, shes) → 0.0
(shes, not) → 0.0
(not, happy) → 0.00093

As mentioned earlier, we catch incorrectly spelled words at the cost of marking rare words / bigrams as having a grammar error:

scientific notation
(⟨s⟩, scientific) → 2.06754e-5
(scientific, notation) → 0.0

Because "scientific notation" was never seen in the training corpus, we mark it as having a grammar error.

2) Subject Verb Disagreement: Good
Subject-Verb Agreement refers to the relationship between a subject and its verb. In correct grammar, the forms of the should be either both singular or both plural. We assess that the Bayesian model does well on these cases, since they can generally be identified within the context of a bigram:

butterflies is flying
($\langle$s$\rangle$, butterflies) $\rightarrow$ 8.6147e-7
(butterflies, is) $\rightarrow$ 0.0
(is, flying) $\rightarrow$ 9.7424e-5

This sentence is correctly classified as having a grammar error (it has a probability less than 1e-7 bigram).
The correct version of the sentences is classified has not having an error:

butterflies are flying
($\langle$s$\rangle$, butterflies) $\rightarrow$ 8.6147e-7
(butterflies, are) $\rightarrow$ 0.01785
(is, flying) $\rightarrow$ 0.00024

Here is another simple example:

The Earth revolve ...
($\langle$s$\rangle$, the) $\rightarrow$ 0.13868
(the, earth) $\rightarrow$ 0.00034
(earth, revolve) $\rightarrow$ 0.0

And the correct version:

The Earth revolves ...
($\langle$s$\rangle$, the) $\rightarrow$ 0.13868
(the, earth) $\rightarrow$ 0.00034
(earth, revolves) $\rightarrow$ 0.00062

Unlike (earth, revolve), (earth, revolves) is associated with a greater-than-threshold probability bigram, so the sentence is marked as having correct grammar.

As always, there are error cases, and in the following sample error, the root cause is attributed to incorrect grammar in the training corpus:

Cats is sleeping
($\langle$s$\rangle$, cats) $\rightarrow$ 4.3073e-6
(cats, is) $\rightarrow$ 0.00333
(is, sleeping) $\rightarrow$ 4.3841e-5

The probability of P(is | cats) is higher than one may expect. The probability of P(are | cats) is luckily higher, at 0.0533, but former probability still surpasses the threshold.

The next three types of grammar errors require much larger sentence context. We will briefly go through what they are, why the model doesn't do well on them, and how we could improve the model to do well on these cases.

3) Wrong Tense or Verb Form: Bad
We illustrate with an example of incorrect verb form: **He later realized that he will forget to do his homework**. In this sentence, "he will forget" should be replaced with "he had forgotten". For completeness, here are the corresponding bigram probabilities:

($\langle$s$\rangle$, he): 0.02887
(he, later): 0.001558
(later, realized): 0.00020
(realized, that): 0.22375
(that, he): 0.04015
(he, will): 0.01749
(will, forget): 4.9767e-05
(forget, to): 0.06394
(to, do): 0.01019
(do, his): 0.00267
(his, homework): 9.2440e-05

Even though the tense is wrong, all bigram probabilities pass the 1e-7 threshold.

4) Noun Form Agreement: Bad
Noun form agreement refers to singular/plural agreement between nouns. For example, consider the sentence, "Tigers are one of the fastest animal in the jungle.". Noun form agreement requires "animal" to be "animals", and it should be intuitive at this point why our model doesn't catch this error.

5) Unclear Pronoun Reference: Bad
Finally, unclear pronounce reference refers to cases where the antecedent of a pronoun is ambiguous. Again we will illustrate with an example: Barbara and Harriet waved goodbye to her children. It is unclear whether "her" refers to Barbara or Harriet, or someone else entirely. The probabilistic model will never learn this case as it stretches across $n > 2$ words.

The past few error cases require larger sentence context and well-annotated (e.g. with exact type of error) data for training. Further, this feels like a task for deep learning with a large corpus which the model could generalize from, rather than current Bayesian approach which is somewhat of a "memorized", local context approach.

Finally, the next few grammar error classes are cases where the model gets lucky in most cases.

6) Incorrect Word Form: Reasonable
Incorrect word form includes cases where the adjective form of a word was used instead of a noun, noun instead of a verb, etc. Here is an example that our model correctly classifies as having an error:

A successfully leader always ...
($\langle$s$\rangle$, a) $\rightarrow$ 0.02199
(a, successfully) $\rightarrow$ 1.6915e-6
(successfully, leader) $\rightarrow$ 0.0
(leader, always) $\rightarrow$ 0.00012

If we replace ”successfully” with ”successful”, the corresponding bigram (successful, leader) has probability 0.0009, and the sentence is now classified as having correct grammar.

7) Missing Prepositions: Reasonable

While there are many more cases, the last type of incorrect grammar we will look at is missing prepositions, for example, dropping ’of’, ’on’, ’in’, etc. We find that in some cases, our model can learn when the preposition is missing. Here are two such cases:

He dined the restaurant
She was under the influence alcohol

The first sentence is missing ”in” and second is missing ”of”. The Bayesian model correctly catches both of these errors (and classifies the fixed versions of the sentences as correct).

The model fails, however, when, based on context elsewhere in the sentence, the grammar is incorrect. For example, consider: **The influence her math teacher shaped her passion for numbers.** Technically, all of these bigrams are valid in the probabilistic sense, but ”influence” in this context is the subject of the sentence rather than the direct object. This could be a case where expanding to 3-grams, training on a larger corpus, and using part of speech may help.

F. Performance on CONLL-14 Test Set

Unfortunately, the sentences in the GEC Shared Task Test set were much more difficult than the cases we have shown above. The majority of grammar errors required sentence context, rather than local word context. That said, the biggest issue was actually that most words were unseen in the training corpora, and the Bayesian model simply cannot generalize to cases where it does not know about the word. Here are a few examples of sentences with grammar errors from the test set:
1. Genetic risk refers more to your chance of inheriting a disorder or disease.
2. When people around us know that we got certain disease, their altitudes will be easily changed, whether caring us too much or keeping away from us.
3. However for some rare diseases , people who have certain gene changes are guaranteed to develop the disease.
Due to unseen bigrams, rather than catching the actual grammar error, the Bayesian model predicted ”has grammar error” on every sentence.

VI. Grammatical Error Detection via Neural Networks

A. Main Approach

Besides N-Gram Bayesian Model, we proposed a neural network method. The neural network consists three parts: tokenization, encoder, and decoder.

We use an LSTM with one dense layer as the decoder because we have sequence inputs and LSTMs can well handle sequence information. Meanwhile, we target at testing whether different embedding methods influence the quality of detecting a grammar error, so it is not important to use a very complex decoder.

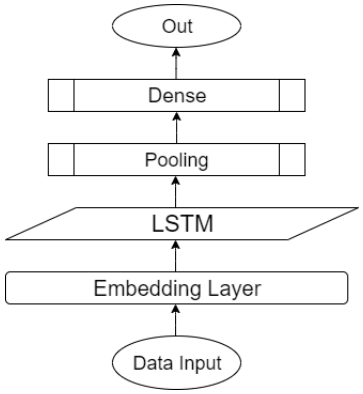

Fig. 1. Network architecture: consists one embedding layer which can be BERT, GPT2, or simple embedding, a single LSTM layer, a pooling layer, and a dense layer

For the choice of tokenizer, since there are errors that are related to punctuation in the dataset, we choose a tokenizer that will tokenize both words and punctuation, and make them all lower cases to avoid bias.

We first implemented a simple embedding method. It will learn all vocabulary based on training dataset and convert tokens to integers regardless of their frequencies. Also we apply <UNK> tokens to those words not learned when doing inference. The encoder is a simple embedding layer that is built in Keras library. It takes in sequences of integers generated by embedding method. The sequences are padded at the end so that each sequence will have the same length as the maximum-length sentence.

Meanwhile, we use pre-trained BERT and GPT2 layer as additional choices. The pre-trained models come from huggingface Transformers library[1].

We use binary cross entropy loss function since we are doing True/False classification, and this loss function can handle it well. Meanwhile, we apply Adam optimizer with learning rate at 0.001 for training purpose. We do a 20% split on validation set, and use binary accuracy as evaluation metric to determine the best model we can get.

B. Dataset

The dataset is converted from Grammar Error Correction dataset we referenced above. The original training dataset has two parts, one is error sentence and another is correct sentence. We generate the classification dataset as combination of these two. For each pair of correct and incorrect sentence, we take either of them with probability. For example, in the original dataset, the error sentence is “She see Tom is catched by policeman in park at last night”, and the corresponding correct sentence is “She saw Tom caught by a policeman in the park

[1] https://huggingface.co/transformers/pretrained_models.html

TABLE I
ACCURACY, FALSE POSITIVE, AND FALSE NEGATIVE ANALYSIS

Embedding Name	***Accuracy***	***False Negative*** $P(\hat{Y}=0\|Y=1)$	***False Positive*** $P(\hat{Y}=1\|Y=0)$
Simple	58.7%	39.5%	57.4%
BERT	61.4%	36.1%	59.6%
GPT2	48.9%	52.8%	55.9%

last night.". When generating our own dataset, we take either of them.

C. Results

We first look at ROC curves for different embedding method at Fig. 2. For all embedding methods, the ROC curve is bad, and AUC scores are very close to 0.5. Within three different methods, comparing with GPT2 performance, BERT and simple embedding have better results and very close to each other.

Then we compare the precision-recall curves for these embedding method at Fig. 3. We can see that all curves are very flat and stay at a high precision while recall is low. Comparing with BERT and simple embedding, GPT2 is slightly worse.

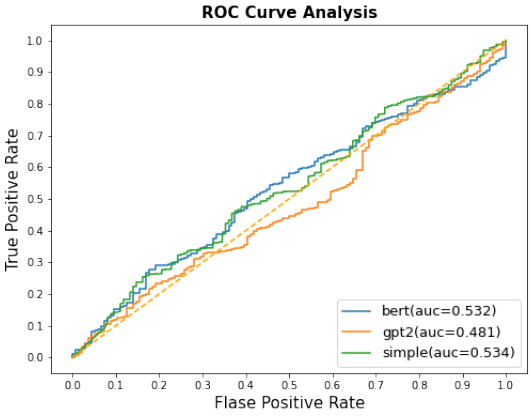

Fig. 2. ROC Curve with AUC score

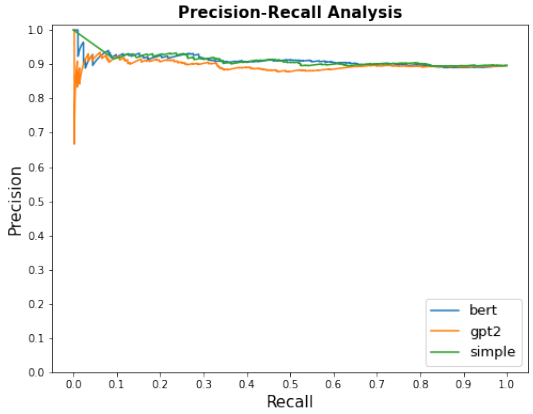

Fig. 3. Precision-Recall Curve

D. Analysis

Since we are doing a classification problem, we also calculate the binary accuracy of different methods on the test set in Table I. We can also see that GPT2 has less than 50% accuracy while BERT and simple embedding gives around 60% accuracy. This 10% difference may due to different architectures of BERT and GPT2. Though both BERT and GPT2 are Transformer based, GPT2 is unidirectional while BERT is bidirectional. This implies when doing the embedding, BERT not only takes the information from previous but also afterwards. In contrast, GPT2 only get forward information. In order to have a better performance in detecting grammar error, it is important to have whole context of sentence instead of one direction. The explanation to why simple embedding also does as well as BERT is simple embedding will convert each token directly to LSTM layer. This is equivalent to that the simple embedding considers the whole sentence but not one direction since it has no architecture that has one direction information feeding.

The high precision and low recall shows that all three methods do good job in detecting positive example but poorly in detecting negatives. This is also revealed by the ROC curve that AUC score is very close to 0.5, which is not ideal. We can see from the false negative rate and false positive rate from Table I. All models have high false positive rates, so the ROC curve reflects that. They have low false negative rates which give them a high precision. It is interesting to notice that GPT2 performance does not do well in either detecting negative as negative or positive as positive (more than 50% for both false negative and false positive rates). This implies that bidirectional embedding method performs better than unidirectional methods in grammar error checking.

Meanwhile, a biased result may caused by the imbalanced test dataset. There are about 86% sentences are positive while few of them are negative. For those negatives, the test set may have syntax difference. It is because CONLL-14 Shared Task Test Set is used as the test set for Grammar Error Correction. We convert it to be used as test set for checker by checking whether there is an error correction marker or not. Also, it is worth noting that as stated in the Bayesian language model, there are lots of words are unseen in the training set. This causes most words are converted to <UNK> tokens, and this increases the difficulty for a language model to successfully classify the sentences

VII. Conclusion and Future Work

In this paper, we showed that probabilistic language models can adequately identify local grammar errors, particularly spelling errors and subject-verb agreement. To extend this approach to do well on more complex grammar errors, one should train on large, representative corpora which can cover the many edge cases of the English language. A solid Grammar Error Detection model needs to have the context of the entire sentence, similar to the finding in the Neural Networks research.

For neural networks, it is not necessary to have a complex model for grammar error checking but in order to boost performance, it is essential that we capture both forward and backward context of the sentence. A bidirectional embedding method is required to yield a good result. Meanwhile, to reduce the vagueness of test set tokens, it is important that we learn those words and language patterns during training. Thus, we either need to have more training data so that it can have word dictionary as large as possible, or make the training data more representative, such as masking words that have less frequencies. This means that we do <UNK> token conversion in both training and inference.

VIII. Code

The code can be found at: https://github.com/aditij113/GrammaticalErrorCorrection.

References

[1] H. T. Ng, S. M. Wu, T. Briscoe, C. Hadiwinoto, R. H. Susanto, and C. Bryant, "The CoNLL-2014 shared task on grammatical error correction," in *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, (Baltimore, Maryland), pp. 1–14, Association for Computational Linguistics, June 2014.

[2] D. Dahlmeier and H. T. Ng, "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*, (Montréal, Canada), pp. 568–572, Association for Computational Linguistics, June 2012.

[3] T. Ge, F. Wei, and M. Zhou, "Reaching human-level performance in automatic grammatical error correction: An empirical study," *CoRR*, vol. abs/1807.01270, 2018.

[4] K. Omelianchuk, V. Atrasevych, A. Chernodub, and O. Skurzhanskyi, "GECToR – grammatical error correction: Tag, not rewrite," in *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, (Seattle, WA, USA â†' Online), pp. 163–170, Association for Computational Linguistics, July 2020.

[5] M. Kaneko, Y. Sakaizawa, and M. Komachi, "Grammatical error detection using error- and grammaticality-specific word embeddings," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, (Taipei, Taiwan), pp. 40–48, Asian Federation of Natural Language Processing, Nov. 2017.

[6] N. L. T. NLTK, "Accessing text corpora and lexical resources." https://www.nltk.org/book/ch02.html, Sept. 2019.

[7] A. Awasthi, S. Sarawagi, R. Goyal, S. Ghosh, and V. Piratla, "Parallel iterative edit models for local sequence transduction," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, (Hong Kong, China), pp. 4259–4269, Association for Computational Linguistics, Nov. 2019.