Correcting Autocorrect: Recognizing Word Impostors

Daniel Stekol
University of Pennsylvania
dstekol@seas.upenn.edu

Jordan Lei
University of Pennsylvania
haochuan@seas.upenn.edu

Abstract

Current grammar correcting algorithms often fail to take contextual information into account. While a word may be grammatically accurate in a sentence, it may still be erroneous in contextual meaning. In this paper, we propose Autocorrect-Squared (AC^2), a model trained to recognize words that may technically be grammatically correct, but are semantically incorrect in their specific context.

1 Introduction

Many modern devices contain auto-correct or grammar-correcting algorithms capable of detecting erroneous words or phrases. Most, if not all, of these grammar correcting methods are capable of detecting with extreme accuracy when a given part of speech is wrong. However, most autocorrect methods do not take into account the contextual information in which the incorrect word or phrase exists. For example, in the sentence "The accomplices were summoned to court in order to [collaborate / corroborate] the evidence." It is clear that the word *corroborate* is contextually correct, and the word *collaborate* is incorrect. A word processor would be unlikely to flag this as an error, given that the word collaborate has the same part-of-speech as the word collaborate. We also note that the words collaborate and corroborate are lexically similar, and could conceivably be mixed up or mistakenly autocorrected when typing in a hurry.
In this paper, we propose a Deep-Learning processor for context-based autocorrect. The objective of our model is to recognize abnormalities in sentences, even when the substituted word is lexically similar and may be correct in terms of part-of-speech. Our paper offers the following contributions:

- We propose a model for context-based autocorrect using a recurrent neural architecture.
- We introduce a dataset, the Context-Sensitive Indicator (CSI) Dataset, which contains sentences drawn from the Brown Corpus (W. N. Francis and H. Kucera, 1965) and corrupted with grammatically identical but contextually incorrect replacement words.

In Section 2, we will introduce our generated dataset and the inputs to our models. In Section 3, we will discuss our methods, including the different training modes of our models. In Section 4, We explain our models, including our baseline logistic regression model and our Deep Learning models. In Section 5 and 6, we will discuss our findings and results. We will conclude with Section 7 and introduce some brief notes on future work.

2 Data

Since as far as we aware, no datasets exist for our particular task, our training and test data was generated artificially by deriving sentences from the Brown Corpus (W. N. Francis and H. Kucera, 1965). Our generated dataset included a two subsets: one that was filtered by part-of-speech, and one that was unfiltered (the precise mechanisms of this filtering are described below). The procedure for generating each sentence in the dataset was as follows:

2.1 Sentence Perturbation

First, a sentence was selected randomly from the corpus, and one of its tokens was selected randomly to be replaced. For the filtered dataset, only particular parts of speech were allowed to be replaced (nouns, adjectives, verbs), since it was felt that replacing other parts of speech (for instance, a conjunction) could drastically change the structure of the sentence. For the unfiltered dataset, the only restriction was that the token to be replaced must be an actual word (rather than a symbol or punctuation mark). Part-of-Speech tags for the original sentences were drawn directly from the annotations of the Brown corpus.

2.2 Generating Replacement Candidates

Candidates for the replacement word were generated by searching for words within a fixed edit-distance of the original word. The underlying assumption is that typed words are most likely to be accidentally replaced by a word that is lexically similar to the original. An edit-distance of 3 was used as the threshold, since this cutoff was found to allow for a sufficiently large variety of reasonable replacement words while excluding large numbers of outlandish suggestions. Suggestions were generated using the edit-search query feature of the SymSpell library (Wolf Garbe, 2019).

2.3 Selecting Word Replacement from Candidates

Since the edit-distance query returns suggestions in order of decreasing word frequency, only the top 100 candidates were considered; this was done to both decrease generation time, and to exclude extremely rare words which were unlikely to actually appear as "impostors". For the filtered dataset, we also used the NLTK library (Steven Bird, Edward Loper, and Ewan Klein, 2001) to tag the suggested words and discarded the words which did not have the same part-of-speech as the original word, thus ensuring that the perturbed sentence would have an identical structure to the original.
Once the list of replacement candidates had been narrowed down as described, a single replacement word was randomly selected from the list, with probabilities weighted by the frequencies of the words. If no suitable candidates existed in the list, no substitution was made, and the sentence was added to the dataset unchanged (with all of its words labelled as correct).
Thus, no sentence had more than one incorrect word, since it was determined that changing too many words would make the sentence unrecognizable, but some of the sentences had no incorrect words, both because the model could use these sentences to learn what good sentences looked like, and because a good model for the given problem should not attempt to find an error in a sentence where there isn't one.

2.4 Postprocessing

In order to convert the data into a format which could be processed by the models, each word was converted to a 300-dimensional vector using PyMagnitude embeddings (Ajay Patel, Alexander Sands, Chris Callison-Burch, and Marianna Apidianaki, 2018). The label for each sentence consisted of a vector of length equal to the sentence, where each element was a per-word binary flag, with 1 indicating that the corresponding word was unchanged and 0 indicating that the word had been substituted.
The complete filtered dataset consists of approximately 120,000 sentences, 20,000 of which were held out for testing, and the unfiltered dataset consists of approximately 80,000 sentences, 4,000 of which were held out for testing.

Below are some randomly selected examples from the filtered dataset:

- Then a porter from Manning's Fish House would trot in with a tray on his *heart* [head].
- There was Wright's, for one, lost amongst trees, its wide *brands* [verandas] strewn with rockers.
- The Sequoia Grove presents another unique aspect of Yosemite, for these ancient giant *times* [trees] are a sight never to be forgotten.
- They answered him in monosyllables, nods, occasionally muttering in Greek to one another, awaiting the word from Papa, who restlessly cracked his knuckles, anxious to stuff himself into his white Cadillac and burst off to the *relay* [freeway].

Note that the sentences are grammatically identical to the originals, and in some cases, it is not obvious which word is out of place.

Below are some randomly selected examples from the unfiltered dataset:

- He *work* [would] have had ample time to go into a blind elsewhere and wait his prey.
- Not unless they *make* [have] to.
- Circumstances gave *the* [her] almost undisputed sway over child-rearing, money-handling, and home countenance.

Note that the unconstrained substitutions can make the sentences quite difficult to comprehend.

3 Methods

For training, our models used an Adam optimizer with a learning rate of 0.00001. Each model was trained on one example at a time (batch-size=1) in order to minimize the number of epochs needed for the model to converge and thus prevent the risk of overfitting.

Initially, we trained each of our models on the filtered and unfiltered datasets (separately), but found that the models were quite biased toward predicting that the entire sentence was correct. This was unsurprising, both because the overall number of correct words was very high compared to the number of incorrect words, and because the initial datasets contained relatively few entirely correct sentences, meaning that the model did not have many chances to see what a "good" sentence looked like, and how it was different from a "corrupted" sentence.

We endeavoured to fix this with two additional changes: we added approximately 5,000 unperturbed sentences to each training set, and we modified the loss to weight the prediction on the incorrect word more heavily: each correct word was given a weight of 1, and incorrect word was given a weight equal to a quarter of the length of the sentence (note that all sentences had length at least 5, so this guaranteed that accurately identifying the incorrect word would affect the loss more than identifying any of the correct words).

Thus, for each of our 3 models (1 baseline, 2 neural) architectures, we had a version which was trained on filtered data with unweighted loss, a version which was trained on unfiltered data with unweighted loss, a version trained on filtered data (augmented with more correct sentences) with weighted loss, and a version trained on unfiltered data (also augmented with more correct sentences) with weighted loss, resulting in a total of 12 models.

4 Model Architectures

Our models were trained to take a sentence (represented as a list of word vectors) as an input and output a scalar between 0 and 1 for each word in the sentence depending on if the word belonged in the sentence (1 corresponding to an appropriate word, and 0 corresponding to an out-of-place word). In the following subsections we will describe our baseline model and the deep learning models, as well as the inductive hypotheses corresponding to each.

4.1 Baseline Model

Our baseline model consisted of taking the average word embedding in the sentence and concatenating it with the current word to obtain a vector of length 600 (300 from the average-word embedding and 300 from the current-word embedding). This vector was passed through a logistic regression, which predicted either 0 or 1 depending on whether the word was contextually inappropriate or appropriate, respectively. This model was trained for 2 epochs, by which point the loss had stopped decreasing.

The logistic regression model has an inductive bias that the model doesn't depend on any sequential information, since the input sentence has been reduced to an average vector. It also assumes that the contextual appropriateness of a word can be determined by the averaged sentence embedding and the current word alone. As is shown in the results, these inductive biases are not particularly suitable for this problem domain.

4.2 Deep Learning Models

Our deep learning models consisted of an LSTM and a bidirectional LSTM, with the latter being our advanced deep learning implementation. The parameters for the LSTM and bi-LSTM were as follows. Both models had an input size of 300, hidden size of 400, and 3 layers. The input size was based on the length of the word-embeddings. The hidden size was chosen to be approximately equal to, but slightly larger than the input size in order to allow for some additional features to be learned, and we chose 3 layers in order for some higher-order features to be captured by the networks.

The LSTM model has the inductive bias that sequential ordering matters in the dataset; that is, contextual understanding requires some features that are based on the position in the sentence. The LSTM implementations also assume that both long and short-term patterns are indicative of whether a given word belongs or does not belong in a given context. The bi-directional LSTM has the added inductive bias that the contextual belonging of a given word in the sentence depends not only on the words that come before it, but also the words that come after it.

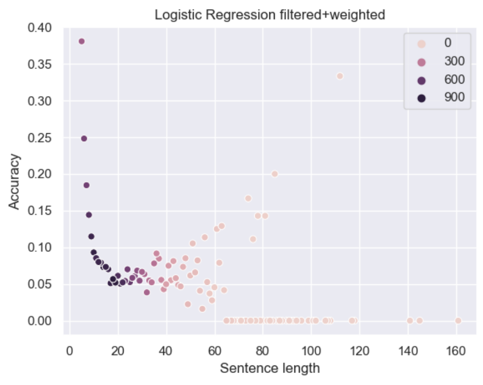

Figure 1: Logistic Regression Performance across different sentence lengths for filtered dataset, using Weighted cross entropy.

5 Results

Our evaluation takes into account the per-sentence accuracy of the models, as well as the per-word precision, recall, and F1 score. For calculating accuracy, a sentence was only counted as correct if the model had labelled the sentence completely correctly (above 0.5 for every fitting word, and below 0.5 for every out-of-place word). Per-word accuracy was not evaluated, since the data is heavily unbalanced (there are far more correct words than incorrect words).

An additional metric used to evaluate our models was best-guess accuracy: for each sentence that contained an incorrect word, we checked whether the lowest-scoring word (as outputted by the model) actually corresponded to the incorrect word.

It should also be noted that on every metric used to evaluate the models, the training score was only a fraction of a percent higher than the test score, meaning that the model did not overfit on the training data (which was originally a concern, since 100,000 training examples is relatively small compared to most NLP datasets).

5.1 Baseline: Logistic Regression

Our baseline model was used to assess the difficulty of the problem. We observed that the baseline model made uninformative predictions on the dataset when using plain Binary Cross Entropy, always predicting that every word belonged where it was. The accuracy of the baseline model was 8% (corresponding to the percentage of the sentences in the data which had not been modified). The recall was negligible and the F1 score was uninformative. After using the weighted loss and modified training data descibed in the Methods section, the model became better at identifying incorrect words in the filtered dataset (Figure 1). However, the model was still irreparably biased on the unfiltered dataset.

More promisingly, however, the best-guess accuracy of the unweighted model was 24% on the filtered data and 14.2% on the unfiltered data, while the best-guess accuracy of the weighted model was 25.1% on the filtered data and 14.8% on the unfiltered data.

5.2 LSTM

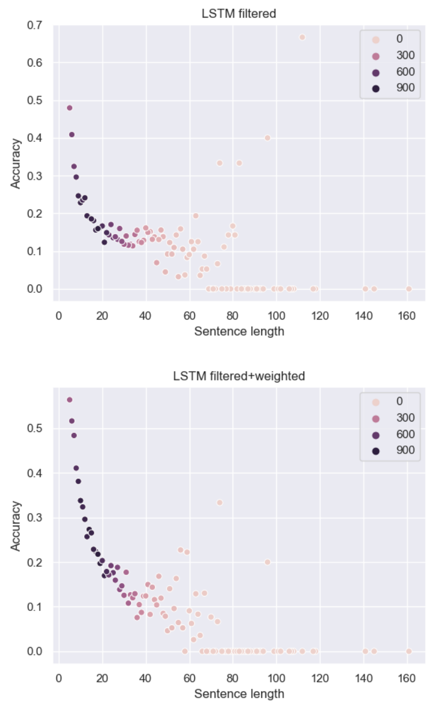

Figure 2: LSTM Performance across different sentence lengths for filtered dataset using Unweighted BCE (top) and Weighted BCE (bottom).

Our LSTM performed significantly better than the baseline, with a per-sentence accuracy of 18% on the filtered dataset (Figure 2). The per-word precision and recall were 69% and 11%, respectively, and the F1 score was 0.20. We attribute the low F1 score to the low recall. The model performed worse on the unfiltered dataset, with an accuracy of 9% and an F1 score of 0.14.

After training the vanilla LSTM with the augmented dataset and weighted loss, we further improved our LSTM accuracy to 22%. The precision and recall were 31% and 34%, respectively, leading to an F1 score of 0.32.

As with the baseline model, the best-guess accuracy of the LSTM was significantly higher than the plain accuracy, with the unweighted model scoring 36.3% on the filtered data and 22.5% on the unfiltered data, while the weighted model scored 36.5% on the filtered data and 24.4% on the unfiltered data.

5.3 Bidirectional LSTM

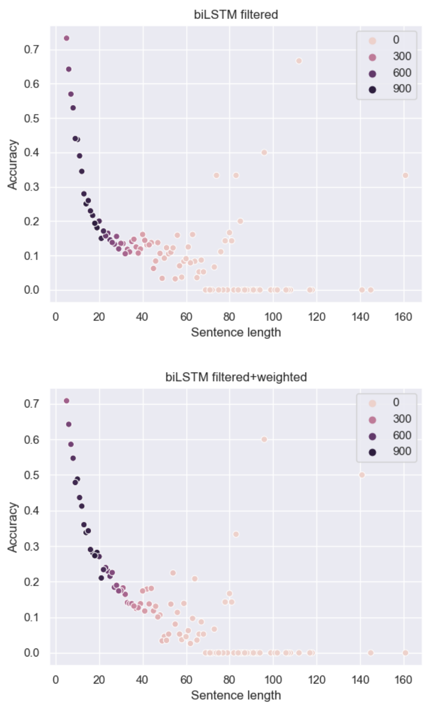

Figure 3: Bi-LSTM Performance across different sentence lengths for filtered dataset using Unweighted BCE (top) and Weighted BCE (bottom).

The bidirectional LSTM outperformed both the baseline and the LSTM, with a per-sentence accuracy of 24% on the filtered dataset (Figure 3). The per-word precision was 71% and the recall was 19%, resulting in an F1 score of 0.30. As before, the model performed worse on the unfiltered dataset, with a per-sentence accuracy of 7% and per-word F1 score of 0.11.

After training the bi-LSTM with the augmented dataset and weighted loss, the model improved significantly, achieving an accuracy of 29% on the filtered dataset, with precision of 33%, recall of 45% and F1 score of 0.38.

As with the other models, the best-guess accuracy of the biLSTM was much higher than its plain accuracy: the unweighted biLSTM scored 39.8% on the filtered dataset and 24.9% on the unfiltered dataset, while the weighted biLSTM scored an impressive 41.6% on the filtered dataset and 26.5% on the unfiltered dataset.

The performance of the various models on the filtered and unfiltered datasets are shown below:

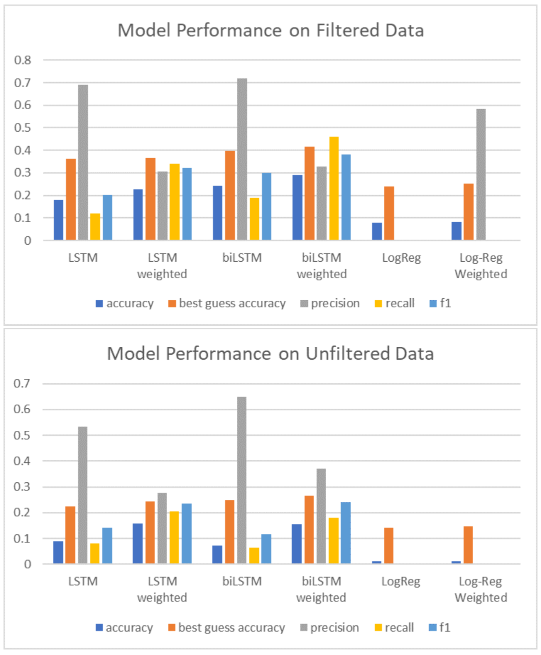

Figure 4: Model Performance Summary

6 Discussion

Although it was originally expected that the models would perform better on the unfiltered dataset since the part-of-speech incongruities could provide clues as to where the out-of-place word is (since certain p.o.s. sequences are less likely than others), the results unequivocally show that the models perform significantly better on the filtered dataset, where the parts-of-speech and grammatical structure are preserved. Upon inspection of the data itself, it can be seen that the unfiltered dataset is likely harder simply because it is more confusing: when a word is replaced with a word that does not play the same role in the sentence, it can become difficult to make head or tail of the sentence. For instance, consider the following sentence from the unfiltered dataset: ”I’m not steering you log”. One could easily assume that the sentence should be ”I’m not steering your log”, when in fact, the sentence is ”I’m not steering you long”. The fact that ”long”, which was playing the role of an adverb, was replaced by ”log”, which plays the role of a noun, obscures the original meaning of the sentence.

As can be seen in Figure 4, the models performed far better on the best-guess accuracy metric than on the plain (.5 threshold) accuracy metric, showing that although the models were actually fairly good at recognizing incorrect words, they were not “confident” enough to assign them scores below 0.5, instead simply giving them slightly lower scores than the other words. This suggests that if a separate model could be trained to accurately predict whether a mistake had been made anywhere in the sentence, it could be combined with our existing models to accurately pinpoint the mistake.

Figure 4 also shows that whereas the models trained with unweighted loss and the unaugmented dataset have much higher precision than recall, the introduction of weighted loss and additional correct sentences brought the precision and recall closer together and raised the overall f1 score. Note, however, that the best-guess accuracy changes very little between the weighted and unweighted models, suggesting that changes in training do not necessarily make the model better at identifying the incorrect word, but simply make it more confident in its predictions. Logistic Regression performs significantly worse than the deep-learning models in all cases (although the introduction of weighted loss and the augmented dataset do somewhat help its performance on the filtered dataset), demonstrating that the problem is far from linearly separable and is much better suited to neural approaches. This also suggests that the sequential information available to the LSTMs is helpful in identifying the substituted words, which is quite unsurprising, since even a human would likely perform quite poorly in this task if shown an unordered set of words rather than an actual sentence.

Figures 1, 2, and 3 demonstrate that the models' performance decays as the sentence length increases: this is likely at least partly due to the fact that the probability of any given word being the impostor decreases as the sentence length increases, but can also be explained by the fact that LSTMs tend to perform poorly on long sequences. However, LSTMs' tendency to forget cannot account entirely for the decline in accuracy, since the performance of the weighted logistic regression model on the filtered dataset (the only instance where logistic regression did not simply resort to always predicting 1s) declined even more quickly as sentence length increased, as shown in Figure 1.

Overall, this problem is clearly a difficult one, as it requires a nontrivial understanding of both sentence structure and meaning. Although none of the models achieve particularly high performance, the most promising results are shown by the biLSTM when trained with weighted loss on the augmented dataset - this supports the idea that bidirectional sequential information is relevant in identifying out-of-context words, and that recurrent neural architectures are better suited to this problem than linear algorithms like Logistic Regression.

7 Conclusion

This paper has introduced a viable Deep Learning model for detecting contextually erroneous words in English sentences. We proposed a modified contextual autocorrect dataset with corrupted words generated from sentences in the Brown corpus.

While our models are not quite up to industry standards, they are a promising first pass at possible implementations of context-based autocorrect systems and word processors. Future work may include the use of transformers and other Deep Learning models for this task, as well as others. It would be interesting to see this applied to contextual autocorrect in other languages, or even perhaps in music.

We hope that our current work will be used to motivate improvements in autocorrect / grammar correcting algorithms in the future.

References

W. N. Francis and H. Kucera. 1965. *A Standard Corpus of Present-Day Edited American English, for use with Digital Computers*, *College English*, 26(4):267.

Steven Bird, Edward Loper, and Ewan Klein 2001. *Natural Language ToolKit (NTLK)*, https://github.com/nltk/nltk

Ajay Patel, Alexander Sands, Chris Callison-Burch, and Marianna Apidianaki 2018. *Magnitude: a fast, simple vector embedding utility library*, https://github.com/plasticityai/magnitude

Wolf Garbe 2019. *SymSpell*, https://github.com/wolfgarbe/SymSpell