Make it clean (P9)

Michele Zenoni (matr. 989482)

Department of Computer Science "Giovanni degli Antoni"
Università degli Studi di Milano
Via Celoria, 18 - 20133 - Milan (MI), Italy.

Contributing authors: michele.zenoni@studenti.unimi.it;

Abstract

The post-OCR correction is a widely studied problem in the Natural Language Processing (NLP) reasearch field; for this project I analyze it as a spelling correction task to investigate if a transformer model can be suited to enhance the classical (statistical) spelling correction techniques by examining various spelling correction strategies on different evaluation metrics.

1 Introduction

Optical Character Recognition (OCR) is the technique to extract textual information from images (photos or scanned documents) and convert it into machine-encoded text; it constitutes a major computer science research topic, combining computer vision and Natural Language Processing (NLP).

The computer vision and image processing module is the core of an OCR software system, it has to recognize characters' shapes for a given alphabet in the input image and translate them into encoded characters (e.g. ASCII or UTF-8). Input images can be generic pictures, which may or may not contain text, the first goal is, in fact, to determine if some text is present, given it can be rotated, scaled or written in different fonts. After finding all places where some text is present, the second step requires to encode each character keeping the order and considering spaces, punctuation and line separations between them if any.

Despite using some open-source advanced OCR tools like Google's Tesseract[1], the output text is not going to be always correct; this is due to several factors including: image resolution, noise and type of writing (handwritten or printed texts); for this

[1] https://tesseract-ocr.github.io/

purpose a post-OCR correction pipeline should be deployed, possibly exploiting some NLP techniques.

Transformers [3] based architectures turned out to be a good option for post-OCR correction: Nastase V. and Hitschler J. in [5] developed a solution for word-segmentation errors with a precision above 96% (on the ACL test dataset), they achieved that by training a machine translation-based correction model built using the *nematus*[2] system [2] on 2,000,000 input-output sequences from the ACL collection; Nguyen et al. used OpenNMT[3], an open-source toolkit for Neural Machine Translation (NMT), to build a character-level MT based on the Bidirectional Encoder Representations for Transformers (BERT) [7] for the error correction task [9]; finally Todorov K. and Colavizza G. demonstrated how to fine-tune pre-trained language models for post-OCR correction, showing advantages and limitations of their approach specifically for an OCRed historical corpora [10].

Hugging Face[4] is a popular web platform that offers free access to open-source deep learning models, datasets and demo apps either via web API or Python packages. For this project I've used a version of the BART model by Facebook AI (today Meta AI) [8] fine-tuned for grammar and spelling correction for the English language directly available on the Hugging Face platform[5].

BART is a denoising autoencoder for pre-training sequence-to-sequence models, which can be seen as a generalization of BERT for the use of a bidirectional encoder and a Generative Pre-trained Transformer (GPT) [6] with the left-to-right decoder. Beside the architecture, BART is peculiar for its pre-training phase: inputs to the bidirectional encoder are "corrupted" with the following noising transformations as shown in Figure 1:

- **token masking**: random tokens are sampled and replaced with the `[MASK]` token, following the BERT pre-training phase for Masked Language Modeling (MLM) [7];
- **token deletion**: random tokens are deleted from the input;
- **text infilling**: a number of text spans are sampled and replaced with the `[MASK]` token, 0-length spans correspond to the insertion of the `[MASK]` token;
- **document rotation**: a token is chosen uniformly at random and the whole document is rotated so that it begins with that token;
- **sentence permutation/shuffling**: the document is splitted into sentences based on full stop ('.') and these sentences are randomly shuffled;
- **text infilling + sentence permutation/shuffling**: combination of **text infilling** and **sentence permutation/shuffling**;

then, the likelihood of the original document is calculated with the autoregressive decoder, for this reason the decoder outputs need not to be aligned with the encoder inputs. The way it has been used for this project will be explained in the next sections.

The structure of this paper is as follows: Section 2 illustrates the problem formalization, the working hypothesis and assumptions, and the objective of the project; Section 3 shows the achieved results using different evaluation metrics; finally, Section 4

[2] https://github.com/EdinburghNLP/nematus
[3] https://opennmt.net/
[4] https://huggingface.co/
[5] https://huggingface.co/oliverguhr/spelling-correction-english-base

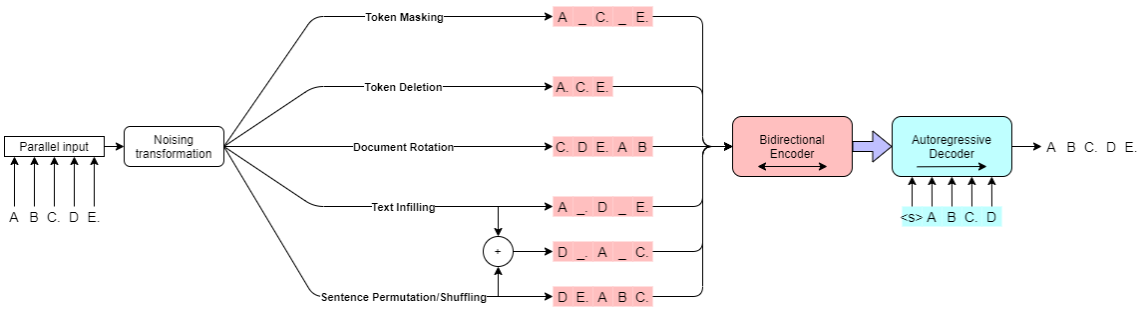


Figure 1: BART pre-training example applying arbitrary noising transformations to the encoder inputs; the decoder outputs need to be shifted.

concludes the paper with a discussion on the work done and on possible future directions.

All code used for this project is available in a dedicated GitHub repository.

2 Research question and methodology

Following the approaches adopted in the literature presented in the previous Section, the post-OCR correction problem can be formalized as a variant of the spelling correction problem. One of the simplest, yet effective, formal definition of the problem was presented by Norvig P. [1]: given a word w and a vocabulary V, w is considered "wrong" or "misspelled" if $w \notin V$; a spelling corrector is a function f such that:

$$\hat{w} = f(w) = \arg\max_{c \in C}(P(c) \cdot P(w|c))$$

where C is the set of candidates words for the correction that, given a distance metric d and a threshold t, can be formalized as: $C = \{c | c \in V \wedge d(w, c) \leq t\}$. A common distance metric between strings is the Levenshtein distance, which computes the minimum number of single character operations (insertions, deletions, substitutions) required to change one string (c) into another (w).

This is known as a statistical definition of the spelling correction problem, a more in-depth analysis can be formulated adopting the noisy channel model [12].

However, taking into account only one single word at a time may work for the *wrong characters in words* type of error, but is not enough to solve *wrong segmentation* errors, in which a word is mistakenly separated into sub-strings possibly attached to the neighboring words. So, the project aims to improve the classical spelling correction approach by considering the sentence context, and this can be done by fine-tuning a sequence-to-sequence (seq2seq) pre-trained transformer. I've expressly deviated from models trained at the character level presented in the previous Section as the state-of-the-art for the spelling correction task, to explore BART as a seq2seq model, exploiting its denoising pre-training objective. More precisely, the model is a fine-tuned version of BART-base, which is composed by a 6 layer bidirectional encoder and a 6 layer autoregressive decoder.

The goal of this project is, therefore, to compare different spelling correction strategies by evaluating them with different metrics on artificially made corrupted sentences, simulating real OCRed English documents (more details on the dataset creation in the next Section). In particular, I've considered five types of spelling correction strategies:

1. **Norvig**$^+$: it corrects every word in the input sentence separately (without taking into consideration the context) using the PySpellChecker[6] module which is based on the Norvig spelling corrector [1]. It uses the Levenshtein distance to find candidate words for the correction and a customisable threshold (default 2). In addition, a function that helps the corrector to find segmentation errors has been developed (this is the reason for the $^+$ in the name of the strategy);
2. **SymSpell**[7]: an open-source spellchecker that uses a Symmetric Delete spelling correction algorithm and a Triangular Matrix approach for word segmentation [4]. It uses the Damerau-Levenshtein distance metric, which differs from the classical Levenshtein distance by including transpositions among the allowed single character operations, I've used a maximum edit distance of 3 to search for possible candidate corrections. For this project I rely on its latest Python implementation[8];
3. **BART**: this strategy implies the use of the fine-tuned version of BART mentioned above, I have selected empirically the model parameters as follows: 128 as the maximum number of new tokens that can be generated, 4 beams for the beam search to find the optimal next token (and best sequence), 0.5 as temperature, which is an hyperparameter to control the determinism of the generation (> 1 to increase the randomness/creativity of the model, < 1 to increase the determinism of the generation);
4. **BART + Norvig**$^+$: the combination of BART and Norvig$^+$, the input text is first processed by Norvig$^+$ and then fed into the BART model;
5. **BART + SymSpell**: the combination of BART and SymSpell, the input text is first processed by SymSpell and then fed into the BART model (this is expected to be the best strategy);

and four different evaluation metrics which will be discussed in the next Section.

[6] https://github.com/barrust/pyspellchecker
[7] https://github.com/wolfgarbe/SymSpell
[8] https://github.com/mammothb/symspellpy

3 Experimental results

I've implemented a Python script (`create_dataset_json.py`) to generate the dataset for this project: given a ground truth text the program modifies it by inserting some errors. More precisely, the script creates a `json` file with the following format:

```
[
  {
    "id": <sample_id>,
    "text": <corrupted_text>,
    "ground_truth": <original_text>,
    "number_of_errors": <number_of_introduced_errors>
  },
  ...
]
```

where the text of the placeholders (between "<" and ">") indicates the role of the entry.

The generation process is customisable: the Boolean flag `USE_WIKI_RANDOM` indicates whether the generation should use the Wikipedia API to get the samples (`USE_WIKI_RANDOM = True`) or extract random sentences from a given text file (`USE_WIKI_RANDOM = False`); in the latter case I've opted to use a publicly available text file[9] which includes 100,000 English sentences transcribed from the European parliament speeches. The `SAMPLES` field is used to control the number of samples to generate.

The modification of the text is also controllable, I have decided to implement four char level edit operations:

1. `MODIFY_CHAR`: substitute the current character with a different one chosen at random;
2. `DELETE_CHAR`: delete the current character, i.e. substitute it with the empty string;
3. `INSERT_CHAR`: insert a randomly selected character before the current character;
4. `INSERT_SPACE`: insert a white space before the current character.

The probability, for each character in the original sentence, to be modified with one of the listed operations is encoded in the `MODIFY_PROB` variable; the choice among the possible modifications is not uniform by default, but it can be specified using a weight (one for each operation).

For this project the four correction strategies have been evaluated on 1,000 samples (`SAMPLES = 1000`) taken from the previously cited text file (`USE_WIKI_RANDOM = False`) with `MODIFY_PROB = 0.02`, meaning that for each character in the original sentence there is a 2% probability to introduce an error, and `MODIFY_CHAR_WEIGHT = 0.30`, `DELETE_CHAR_WEIGHT = 0.15`, `INSERT_CHAR_WEIGHT = 0.15`, `INSERT_SPACE_WEIGHT = 0.40`, meaning that if an error introduction occurs (with 2% probability) then the operation will be `MODIFY_CHAR`, `DELETE_CHAR`, `INSERT_CHAR` or `INSERT_SPACE` respectively with 30%, 15%, 15% and 40% probabilities.

[9] https://github.com/oliverguhr/spelling/blob/main/data/raw/eng-europarlv7.100k.txt

The evaluation takes into consideration four metrics:

1. **accuracy**: given a ground truth sentence and a candidate correction with normalized spaces (each occurrence of multiple spaces is replaced with a single space), convert each of their characters in the respective ASCII code, then if the two sentences are of different lengths, extend the shorter by adding a dummy code (-1) until they become of the same length and, finally, compute the normalized accuracy between the two lists of numeric codes, i.e. $\frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$;
2. **accuracy without punctuation**: same as **accuracy** but without considering punctuation;
3. **Word Error Rate (WER)**: given a ground truth sentence and a candidate correction with normalized spaces, remove punctuation and divide the sentences according to spaces, if the two sentences are of different lengths, extend the shorter by adding a dummy word until they become of the same length; if they are equal (not case sensitive) increase the correct score by 1, otherwise increase the error score by the Levenshtein distance between the two, finally, return $\frac{\text{error score}}{\text{error score+correct score}}$;
4. **average Character Error Rate (CER)**: given a ground truth sentence and a candidate correction with normalized spaces, remove punctuation and divide the sentences according to spaces; compute the CERs between each word of the two sentences (i.e. the case sensitive, normalized Levenshtein distance) and return their sum divided by the number of words in the ground truth sentence.

Table 1 shows some numerical examples of the four evaluation metrics.

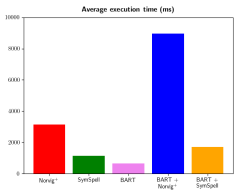
Ground truth	Candidate correction	Accuracy	Accuracy (no punct.)	WER	Average CER
Hello, world	Hello worlds	0.417	0.917	0.500	0.083
Hello, world	helo wa ld	0.167	0.182	1.000	0.700
Hello, world	WORLD	0.000	0.000	1.000	0.500
Hello, world	Hello, world	1.000	1.000	0.000	0.000

Table 1: Examples of the different metrics computed between a ground truth text (reference) and a possible correction (hypothesis).

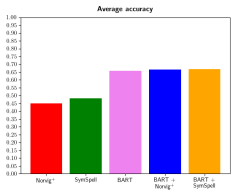
The results of the evaluation with the parameters specified above are summarised in Figure 2 and listed in Table 2, the average execution time per sample has also been reported. The average length of the samples was ≈ 143 characters.

Correction strategy	Execution time (ms)	Accuracy	Accuracy (no punct.)	WER	Average CER
Norvig$^+$	3156	0.450	0.690	0.404	0.239
SymSpell	1153	0.482	0.739	0.253	0.151
BART	**641**	0.658	0.708	0.401	0.255
BART + Norvig$^+$	8972	0.665	0.777	0.286	0.170
BART + SymSpell	1700	**0.667**	**0.780**	**0.234**	**0.137**

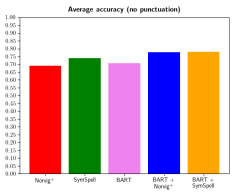
Table 2: Evaluation results as an average of 1,000 samples, best result per column is given in bold.



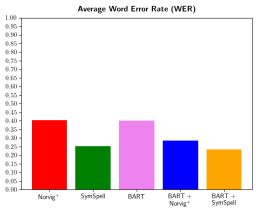
(a) Average execution time per sample (ms)



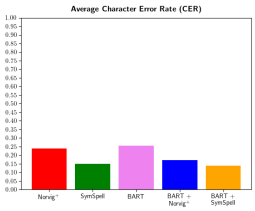
(b) Average accuracy



(c) Average accuracy (no punctuation)



(d) Average Word Error Rate (WER)



(e) Average Character Error Rate (CER)

Figure 2: Evaluation results as an average of 1,000 samples.

4 Concluding remarks

In this project I explored different approaches to tackle the spelling correction problem as a variant of the post-OCR correction problem. The use of a transformer architecture pre-trained on denoising proves to be a valuable support for the classical (statistical) spell checking approach, particularly for word segmentation errors. In fact, the best strategy in all evaluation metrics is **BART + SymSpell**, which was the expected result of the analysis.

However, the evaluation process showed that the model has a considerable room for improvement and needs a more refined fine-tuning phase possibly on a large dataset

of artificially corrupted text, varying the error probabilities or on a dataset of real OCRed documents such as the one presented in [11].

Future work should, alternatively, focus on developing a BART model pre-trained at the character level following the state of the art approaches for the spelling correction task presented in the first Section.

References

[1] P. Norvig. "How to write a spelling corrector." (2016), [Online]. Available: https://norvig.com/spell-correct.html (visited on 06/27/2023).

[2] R. Sennrich, O. Firat, K. Cho, *et al.*, "Nematus: A toolkit for neural machine translation," in *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, Valencia, Spain, Apr. 2017, pp. 65–68.

[3] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, 2017.

[4] W. Garbe. "Fast word segmentation of noisy text." (2018), [Online]. Available: https://seekstorm.com/blog/fast-word-segmentation-noisy-text/ (visited on 06/27/2023).

[5] V. Nastase and J. Hitschler, "Correction of ocr word segmentation errors in articles from the acl collection through neural machine translation methods," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018.

[6] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2019.

[8] M. Lewis, Y. Liu, N. Goyal, *et al.*, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[9] T. T. H. Nguyen, A. Jatowt, N.-V. Nguyen, M. Coustaty, and A. Doucet, "Neural machine translation with bert for post-ocr error detection and correction," in *Proceedings of the ACM/IEEE joint conference on digital libraries in 2020*, Virtual Event, China, Aug. 2020, pp. 333–336.

[10] K. Todorov and G. Colavizza, "Transfer learning for historical corpora: An assessment on post-ocr correction and named entity recognition," in *CHR 2020: Workshop on Computational Humanities Research*, Amsterdam, the Netherlands, Nov. 2020, pp. 310–339.

[11] T. Hegghammer, *Noisy ocr dataset (nod)*, version 1.0.0, Jul. 2021. DOI: 10.5281/zenodo.5068735. [Online]. Available: https://doi.org/10.5281/zenodo.5068735.

[12] D. Jurasfky and J. H. Martin, "Spelling correction and the noisy channel," in *Speech and Language Processing*, 2023.