

HULAT-UC3M at SimpleText@CLEF-2022: Scientific text simplification using BART

Adrián Rubio¹, Paloma Martínez¹

¹Universidad Carlos III de Madrid, Computer Science and Engineering Department, Av de la Universidad, 30, 28911, Leganés, Madrid, Spain

Abstract

This paper describes the proposed system developed by HULAT-UC3M research group from Universidad Carlos III de Madrid to solve Task 3 of SimpleText@CLEF-2022 on scientific text simplification. We present an abstractive approach implemented with BART. BART is a sequence-to-sequence model trained as a denoising autoencoder. In order to fulfill this specific task proposed, it has been fine-tuned to simplify text passages provided by the task organizers. The proposed system obtained a loss value of 1.232 and a rouge L value of 0.615 on the validation set.

Keywords

Scientific text simplification, Summarization, Deep Learning, BART

1. Introduction

Science is becoming harder to understand to non-scientists. With the ever increasing use of jargon, long sentences and scientific terms research papers are hermetic to the average person. Not only this issue hinders the ability of non-scientist to understand current breakthroughs, it also impedes researchers advancing with their careers and those moving onto new fields of study. Because the high volume of scientific literature there is also a great interest on developing tools capable of summarizing such papers, so that they allow to process more data in less time.

Summarizing text is an essential process in text simplification together with lexical and sentence simplification. Currently there are two main approaches for text summarization, which are extractive summarization and abstractive summarization [1, 2]. Extractive text summarization involves tokenizing the text into sentences and ranking them using a variety of different methods. The sentences regarded as more relevant or that encapsulate more meaning of the text are ranked higher than those that don't. These sentences are later bundled up together to form the summary.


Abstractive summarization is more complex and more computationally intensive than extractive summarization, however it is the only approach that can provide cohesion and it is more similar to human-like text summarization. Human-like text summarization involves reading the whole text, grow ones understanding of the text, and then rewrite the text focusing on the most paramount aspects of it. Computers do not have such knowledge or language capability, therefore this makes abstractive text summarization a very difficult and non-trivial

CLEF 2022: Conference and Labs of the Evaluation Forum, September 5–8, 2022, Bologna, Italy

✉ 100405991@alumnos.uc3m.es (A. Rubio); pmf@inf.uc3m.es (P. Martínez)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

task. However this approach yields better results as it manages to understand the meaning of the text, thus adding more cohesion and context to the summaries produced [3]. Abstractive summarization is the approach followed in HULAT-UC3M system.

2. Dataset

The dataset provided is divided into two datasets [4]. Firstly there is the training set, which consists of six-hundred and forty seven scientific passages with their corresponding simplified passage. The source passage and the simplified passage are provided in two different CSV files. The first file contains the source passage, identified with an ID, a document ID which provides information on the origin of the passage, and a query text which can be interpreted as the topic of the passage. The query text is identified with an ID as well. The second file contains the simplified version of the passage, identified with its ID as well. In total there are six-hundred and forty seven entries in the training set.

The other part consists of the test set, which has the same attributes as the first file of the training dataset, but includes one-hundred and sixteen thousand seven-hundred and forty two entries.

The training set was used to fine-tune the model to the task, whereas the test set was used to generate the simplified passages which were evaluated.

3. Resources

To develop this system, a Python notebook was developed in Google Colaboratory. Several libraries were necessary for the adequate development of the model. Pandas, in its current version 1.4.2, is a versatile library used to process data. PyTorch 1.11, Transformers 4.19.2 and Blurr 1.0.0 were necessary to use in order to being able to use a pre-trained BART model from Huggingface was used [5]. This model has been pre-trained using the CNN/Daily Mail dataset.

4. Methods and system description

4.1. Data pre-processing

As the training data was provided in two different files, it was necessary to combine them into one file which then could be used for training the model. Furthermore, a key aspect of the data is the query text; it is paramount to include this information in the training set, as it was thought that it would improve the results. However this had to be included somehow in the source passage, as for training it was required that the entries be composed of three attributes: source identification, source passage and simplified passage.

In terms of how to include the query into the source passage, there are several options:

- Option 0: Not including the query.
- Option 1: Add the query at the end of the passage: source passage + query text.

- Option 2: Include the query separated from the source passage by " related to": source passage + "related to" + query text.
- Option 3: Include the query separated from the source passage by ",related to ": source passage + ", related to" + query text.

Data was generated using all the options described to later experiment and observe which yields the best results. The pre-processing of data has been developed in a Python script, using the Pandas library, with the version being Python 3.8. From the first data file, shown as File 1 in Figure 1, we extracted the values of sentence id, source sentence and query text into a dataframe. Then depending on the option it was necessary to add text to the data (for example options 2 and 3), this was done with a simple concatenating operation. Up next we perform the aggregate operation to join source sentence and query text into source sentence. The dataframe is redefined to being just the sentence id and source sentence. From the other data file, shown as File 2 in Figure 1, we extracted sentence id and simplified sentence into another dataframe.

Both dataframes are joined using the sentence id as key into a single dataframe. This dataframe is later converted to a easy to work CSV file, which consists of three attributes. The sentence id, which identifies the sentence, the source sentence itself and the simplified sentence. This output format is used as it is quite easy to work with the data in this format, using the pre-trained model which will be explained up next.

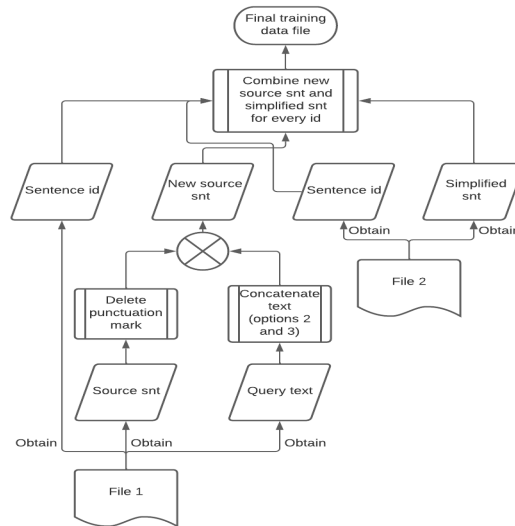


Figure 1: Architecture of data pre-processing

4.2. Deep Learning Model

This model implements a pre-trained BART architecture from Huggingface. As explained in the original paper [6], BART is a sequence-to-sequence model trained as a denoising autoencoder. The main component of BART are transformers, as BART stands for Bidirectional AutoRegressive Transformers. A transformer is a component based in a encoder-decoder architecture and

is sequence-to-sequence. This means that BART that can be fine-tuned to take a text sequence and produce a different text sequence as an output [7]. We can fine-tune this language model to a specific task in NLP. In this case the model was fine-tuned for Conditional Text Generation, which is used for text summarization.

On the first phase we use the output CSV file of the data pre-process and perform supervised training. The inputs to the model are the source sentence and the simplified sentence, they are in a dataframe format. Computers don't understand sentences, therefore the first step is to tokenize each sentence, and obtain the embeddings of the inputs to the transformers.

With word embeddings, the aim is to map every word in the sentence to a point in space where similar words in meaning are physically closer to each other. The space in which they are present is called an embedding space. In this case the pre-trained model used incorporates a pre-trained embedding space [5]. This embedding space maps each word in the sentence to a vector. Furthermore it is important to represent the location of the word within the sentence, as the location of a word in the sentence may result in different meanings, this is where positional encoders are used. It is a vector that has information on the distances between words in the sentence [8, 9]. After passing a word through word embedding and applying positional encoding we obtain the word vectors that have positional information, i.e. context. The architecture for this specific model is detailed in Figure 2. With a transformer encoder there is no need to pass each word individually through the input embedding, all words of the sentence are passed simultaneously and determine the word embeddings simultaneously. We apply this process both to source sentence and simplified sentence, thus obtaining the embedded text (source sentence) and target (simplified sentence), and then pass it in to the transformer.

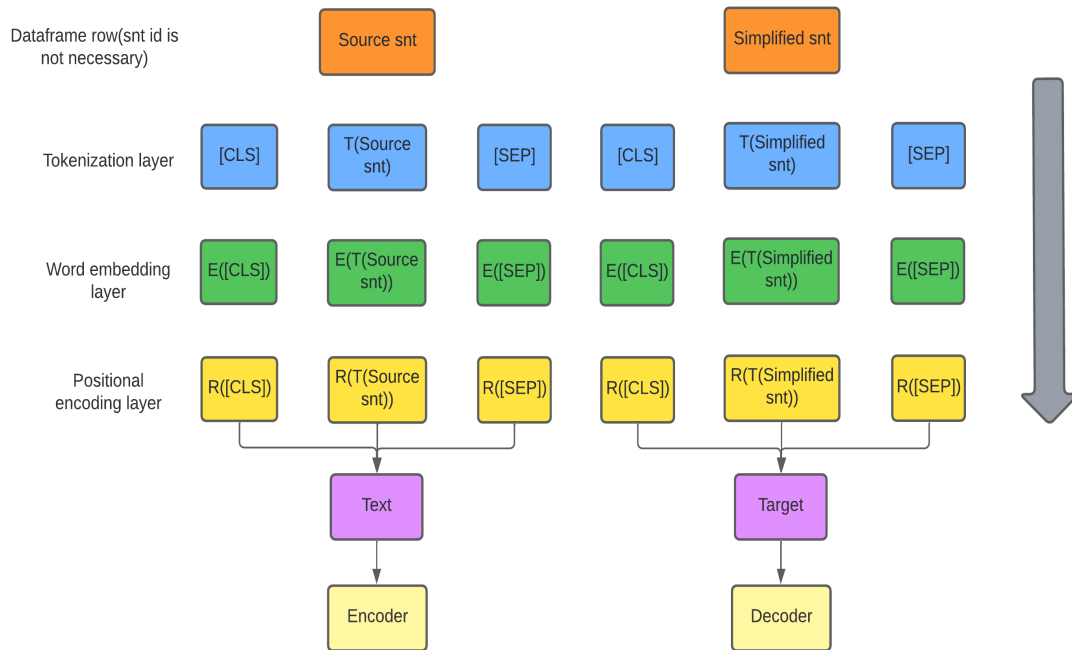


Figure 2: Embedding architecture of the system

5. Experimenting and results

Firstly several experiments were carried out to see which of the data processing options yields better results. Table 1 shows that adding the query information into the training data yields better results. The difference between the losses of Option 0 and Option 1 is substantial. Having acknowledged this, we can observe that introducing this information in a way that would be equivalent to natural language (Option 3), generates better the best results among the described options.

Table 1

Metrics obtained using the different options described (two epochs using training dataset).

| Option | Train loss | Valid loss | Rouge 1 | Rouge 2 | Rouge L |
|----------|------------|------------|---------|---------|---------|
| Option 0 | 2.91 | 2.15 | 0.655 | 0.51 | 0.625 |
| Option 1 | 1.24 | 1.417 | 0.652 | 0.5 | 0.619 |
| Option 2 | 1.179 | 1.346 | 0.652 | 0.5 | 0.62 |
| Option 3 | 1.137 | 1.231 | 0.659 | 0.494 | 0.615 |

Once the text is transformed to input sequences, it can then be used to fine-tune the system for the task. As described in section 2, the training data consists of the sentences and their respective simplifications. The training dataset is further split into training (80%) and validation (20%). The objective of fine-tuning with this data is to create a model capable of generating the most similar simplifications to the simplifications provided. To achieve this purpose several experiments were carried out modifying the value of the hyperparameters in order to obtain the best metrics.

The Blurr library provides a built-in method which slowly ramps-up the value for the learning rate in a log-linear way [10]. The loss is recorded for every iteration. In Figure 3, the values are plotted in order to find the best value for the learning rate. This process is very handy if the model is being developed in a Python Notebook. One cell can be dedicated to find the best learning rate, and the following cell performs the fine-tuning after having obtained the value of the previous cell. Because of this resource there was no need to experiment with the value of the learning rate ourselves. The experimenting was performed modifying the values of other hyperparameters, namely batch size, number of epochs and the length of the output.

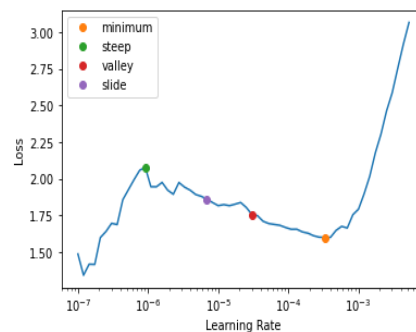


Figure 3: Loss value for a given learning rate

Table 2

Values tested and optimal value for each hyperparameter.

| Hyperparameter | Range | Optimal |
|---------------------------|--------------------------------|---------|
| Batch size | [1,2,3,4,5] | 2 |
| Epochs | [1,2,3,4,5] | 2 |
| Max length and Min length | [(10,30)(15,30)(10,40)(10,50)] | (10,50) |

As table 2 shows, when it comes to the adequacy of the hyperparameters, it was observed that the optimal number of epochs is 2, as with more epochs the model started overfitting to the training data, the loss value on validation incremented at the same time as the loss value on training decreased.

Regarding the batch size, smaller sizes are beneficial for two main reasons. Firstly smaller batch sizes are noisy, offering a regularizing effect and lower generalization error; secondly it makes it easier to fit one batch worth of training data in memory. This is paramount since resources on Google Colab are limited, namely GPU RAM size.

As for the minimum and maximum length of the output simplification, this was more of a eyeballing task. In several experiments it was observed that the output did not have sufficient length to adequately simplify larger passages, thus the decision was made to set the maximum length to 50 words.

The metrics obtained with the optimal hyperparameters shown in table 3:

Table 3

Obtained metrics with optimal hyperparameters.

| Validation loss | Rouge 1 | Rouge 2 | Rouge L | Precision | Recall | F1 |
|-----------------|---------|---------|---------|-----------|--------|-------|
| 1.231 | 0.659 | 0.494 | 0.615 | 0.938 | 0.94 | 0.938 |

After the model had been fine-tuned, the test data file was used to generate the simplifications of the passages.

6. Conclusions

Developing this kind of models is no easy task. Throughout the development process there has been some setbacks and dead ends. Nevertheless the developed model fulfills the task competently. The generated simplified passages have obtained a desired evaluation in the validation set and an direct inspection of several of the simplified passages shows that the passages generated have grammatical correctness and adequately simplify the original sentences. Moreover the hypothesis that adding the query text as a topic marker would improve the results was validated.

However there were some options which could not be implemented or explored due to time constraints. Namely it was devised to use another external set of data to further the scope and variety of the training data. Some sources that could be used are Simply Wikipedia, Turk corpus or Asset corpus. Further work may be carried out with regard to this.

Acknowledgments

This work has been supported by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17) and by the Research Program of the Ministry of Science and Innovation - Government of Spain (ACCESS2MEET project-PID2020-116527RB-I00).

References

- [1] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, K. Kochut, Text summarization techniques: a brief survey, arXiv preprint arXiv:1707.02268 (2017).
- [2] Shrivarneshi, Text summarization approaches for nlp – practical guide with generative examples, <https://www.machinelearningplus.com/nlp/text-summarization-approaches-nlp-example/> (2020).
- [3] P. Sikka, V. Mago, A survey on text simplification, arXiv preprint arXiv:2008.08612 (2020).
- [4] L. Ermakova, P. Bellot, J. Kamps, D. Nurbakova, I. Ovchinnikova, E. SanJuan, E. Mathurin, S. Araújo, R. Hannachi, S. Huet, N. Poinso, Automatic simplification of scientific texts: Simpletext lab at clef-2022, in: M. Hagen, S. Verberne, C. Macdonald, C. Seifert, K. Balog, K. Norvaag, V. Setty (Eds.), *Advances in Information Retrieval*, Springer International Publishing, Cham, 2022, pp. 364–373.
- [5] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, CoRR abs/1910.13461 (2019). URL: <http://arxiv.org/abs/1910.13461>. arXiv:1910.13461.
- [6] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, arXiv preprint arXiv:1910.13461 (2019).
- [7] A. Bajaj, P. Dangati, K. Krishna, P. A. Kumar, R. Uppaal, B. Windsor, E. Brenner, D. Dotterrer, R. Das, A. McCallum, Long document summarization in a low resource setting using pretrained language models, arXiv preprint arXiv:2103.00751 (2021).
- [8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, *Advances in neural information processing systems* 30 (2017).
- [9] J. Alammam, The illustrated transformer, <https://jalammar.github.io/illustrated-transformer/> (2018).
- [10] L. N. Smith, A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay, arXiv preprint arXiv:1803.09820 (2018).

Online Resources

The source code is accessible via

- GitHub