NUANS Miniproject: Extractive Summarization as Text Matching on FairySum

Leonardo Lavalle
1838492
Sapienza, University of Rome
lavalle.1838492@studenti.uniroma1.it

1 Introduction

Extractive Summarization is an NLP process which consists in selecting a subset of sentences from the text to form the summary. In *Homework 2* it's been already pointed out all the difficulties one has to deal with and how it's not trivial at all to perform an extractive summary on our FairySum dataset.

The **FairySum** dataset is the one created by us, the students which followed NUANS course and manually annotated the little corpus given by Prof. Navigli about fairy tales and short stories. The dataset comprises a total of 91 stories. For the project has been divided in train set (75 stories) and test/val set (15 stories). The story "*Rothschild's Violin*" hasn't been considered because of the absence of a ground truth summary. The gold summaries of the texts are not unique: in fact we considered as ground truth the set of each summary generated by the students. Consequently, the evaluation of the generated summary for a given story will not rely on a single gold summary but rather on a set of them. In FairySum dataset, the number of gold summaries for each text is at most three.

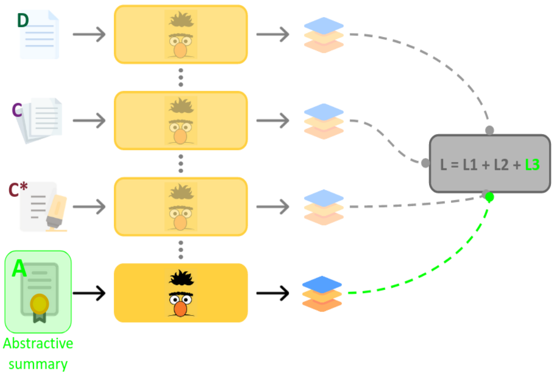

Figure 1: Custom model architecture. It's a *quadruplet* BERT network. Each block receives a different input. **D** is the original text, **C** is the set of summary *candidates*, **C*** is the *gold* summary and **A** is the *abstractive* summary.

During the manual annotation of extractive summaries, two aspects emerged which could be leveraged to produce the best possible summary and that I wanted to dig into in this project: (i) the importance of determining in a quantitative way the *lenght* (Section 3.2) of the output summary which the model has to produce. (ii) the help that can be received from the presence and information contained in the *abstractive* (Section 3.3) summary.

Most of the extractive summarization techniques are based on the concept of sentence salience to identify the most important sentences in a document. For example the baseline model we use for comparison (*Sentence-BERT*) uses LexRank [3] to score the importance. Because of its peculiarity and groundbreaking idea with respect to the field, **MatchSum** [6] immediately caught my attention. First of all the paper stresses that a summary and the original text have to be *semantically* close, which in my opinion is the right starting way to achieve good results on the topic. It then doesn't employ strictly deep learning techniques but introduces other ways to deal with the problem as, for example, the introduction of summary candidates (see Section 2 for more details). I developed indeed a custom solution (Figure 1) based on the idea of MatchSum with the addition of new features to be tested on the FairySum dataset to perform extractive summarization [1].

2 MatchSum

As anticipated, this paper [6] (submitted in 2020) created a paradigm shift with regard to the way people built neural extractive summarization systems. They formulate the extractive summarization task as a semantic text matching problem, in which a source document and candidate summaries are

[1] links to my Project git hub repository NUANS Project and wandb workspace NUANS workspace.

matched in a semantic space (Figure 2).

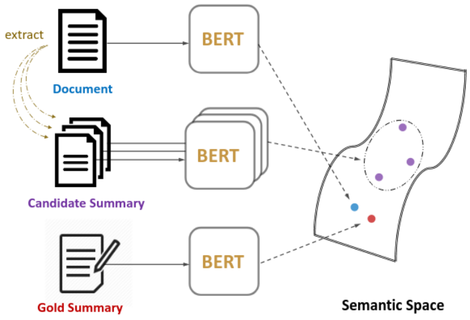

Figure 2: MatchSum explained.

It's basically divided in two phases: **1)** *extraction* phase where a set of possible summary candidates is computed. First, a deep learning method is used to select *ext* number of sentences from the text. Then, the final number of summary candidates is generated by taking all combinations of *sel* sentences subject to the pruned document, producing a total of $\binom{ext}{sel}$ candidates (as it can be evinced, this is very computationally expensive and it's a problem that I tried to address in Section 3). **2)** *matching* phase in which the best summary among all the other candidates is choosen because is the closest one to the original text in the semantic space. *How is the semantic space built?* Thanks to the text embedding generated by BERT and the loss functions.

2.1 Loss functions

We will attempt to provide a brief explanation of how to shape the semantic space of text embeddings, which enables texts with similar semantics to be located closer in the space. Meanwhile let r_D and r_C denote the embeddings of the document D and candidate summary C. Their similarity score is measured by $f(D, C) = cosine(r_D, r_C)$. In order to fine-tune BERT models, we use two margin-based triplet losses[2] to update the weights. Intuitively, we want the embedding of the best summary to be the most similar as possible to the original document one. In addition, the gold summary $C^\star$ should be semantically closest to the source document D:

$$L_1 = max(0, f(D, C) - f(D, C^\star) + \gamma_1), \quad (1)$$

where γ_1 is the margin value. For the second loss all candidate summaries are sorted in descending order of ROUGE [4] scores with the gold summary. The basic idea is to let a better candidate summary to obtain a higher similarity score compared with an unqualified candidate summary:

$$\begin{aligned} L_2 = max(0, f(D, C_j) - f(D, C_i) \\ + (j - i) * \gamma_2)(i < j), \end{aligned} \quad (2)$$

where C_i represents the ranked better candidate and γ_2 is another margin value. The two losses are finally summed up to obtain the final loss which will guide the entire semantic space modeling process.

3 Methods

All the project was implemented by scratch using the *pytorch-lightning* framework and using *wandb* (see 1 to access the workspace) for logging and monitoring the training phase. This because the official repo of [6] was not well documented and written using FastNLP library which I'm not very familiar with. The basic block of my network is the Bert-like Encoder, which is in charge of producing the textual embeddings. I started by using BERT [2] and RoBERTa [5] pretrained architectures from Hugging Face.

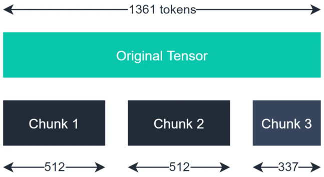

Figure 3: Visualization of chunking operation.

But as soon as I started writing the first lines of code I recognize that the maximum number of tokens which these two types of encoders could handle was absolutely not enough for the task. In fact, the FairySum texts and summaries far exceeded the allowed size of 512. A "chunking" strategy was required, meaning a procedure that breaks down the tokens of a text into smaller segments that can fit within the maximum allowed encoder size. The embeddings of these segments are then combined to produce a single embedding (Figure 3). The chunking method will remain an indispensable component of my project workflow as it is the sole means of meaningfully embedding lengthy texts. Alternatively, architectures that permit longer maximum lengths (4096), such as the LongFormer [1], are also utilized to address this issue and test performances of my solution.

[2] very well explained in the article Triplet Loss Intro.

The custom architecture used throughout the project is the one depicted in Figure 1: a *quadruplet* bert-like encoder (BERT, RoBERTa or LongFormer) network with tied-weights.

3.1 Chunk

As already mentioned, the chunk operation over the tokenized texts is something essential to achieve satisfactory semantic embeddings. To that end, with the help of a well-done article[3], I implemented it. Fundamentally, the *collate* function of the DataLoader class contains the code that divides the tokenized text into smaller, encoder-compatible pieces; and in the *forward* function of the model, a method is implemented to aggregate the embeddings related to the same input text by computing their mean. Unfortunately, due to limited computational resources, it was not possible to keep and process all portions of tokens sequence. It was necessary to introduce a maximum number of chunks for each text (*max_num_chunks_text* for the texts to be summarized and *max_num_chunks* for the candidate and gold summaries). In order to lose as little information as possible, I conducted an analysis about the average dataset chunks length and thus what should have been the ideal numbers for the two hyperparameters. It turned out that if the maximum embedding length is 512 (Bert and RoBerta) the two values are 10 and 4 and if the length is 4096 (Longformer) the values are 2 and 1 respectively (see code in 1 for more details). During training phase it was tried to respect these values as far as possible.

3.2 Candidates Extraction

Maybe the most important and sensitive part of the overall method. In the MatchSum paper [6] not much is said about how to compute the number of extracted sentences (*ext*) and of the final candidates ($\binom{ext}{sel}$) for each text. I therefore decided to pursue a personal approach. The number of extracted sentences (using the Sentence-BERT model [7]) is chosen thanks to the employment of a Support Vector Regression model (SVR). In fact, during the last Homework, I noticed that generally the output length of a summary depends on two quantities: the *original length of the story* and the *concentration of events* in it. It is very likely that there are others, but these are the ones that caught my attention the most. For the calculation of the last quantity I took advantage of the *BookNLP* library which helped me in detecting events within the stories. To measure the concept of "events concentration", all I did was just count the sentences that contained at least one event. In Figure 4 is possible to visualize the fitting curves of these two quantities (a sixth degree polynomial). After the prediction of the *ext* sentences for each text (which is a kind of indication of the length that the final summary should have), it's needed to select the candidate summaries. *How many?* Instead of compute the size of the candidate summary set according to [6], whose method suffers from combinatorial explosion problems, I engaged a self-devised strategy. Always working on sentence indices, starting from the output of the regression model n we select k_range candidate sets. Each i^{th} set includes *hypothetically* all simple combinations of $(n - k\_range + i)$ sentences from a set of n sentences, but for computational reasons we randomly sample from each set only $pick\_random\_n$ summary candidates (see project code linked by 1 for more clues). For instance, if I set $k\_range = 10$ and $pick\_random\_n = 5$ the total number of candidates will be 50.

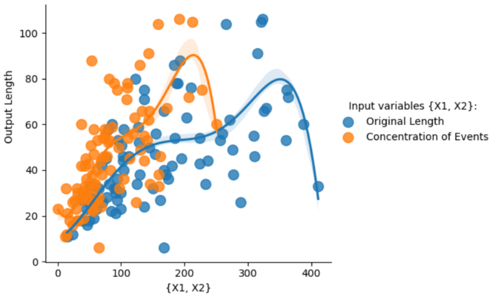

Figure 4: Fitting curves for the prediction of output length summaries.

3.3 Abstractive Summary

The thing that helped me the most during manual FairySum annotation, without any doubts, was the possibility of having a brief **abstractive summary** for each story at hand to help with a selection of the most important happenings within the plot. Hence, my attempt involved creating a module that utilizes the information present in an abstractive summary to generate an extractive summary in its favor. In particular, I decided to use Pegasus [8], a SOTA model for abstractive summaries, as a black box for all the stories of the training set. I held the conviction that the incorporation of a fourth bert-like encoder into the existing architecture (as depicted

[3] How to Apply Transformers to Any Length of Text

in Figure 1) would enhance the modeling of the semantic space of textual embeddings by also including the abstractive summaries as well. I have indeed added a third margin-based triplet loss:

$$L_3 = max(0, f(D,C) - f(D,A) + \gamma_3), \quad (3)$$

where A represents the abstractive summary and γ_3 a margin value. In this way, since the original story and its abstractive summary should be semantically close, we force the model to represent their embeddings as similar as possible. An obvious problem of using *pegasus-large* pretrained model as it is, is that having been trained on news articles that are shorter than our dataset, the output abstractive summaries are certainly not ideal. This can be a research direction to follow for future improvements of the overall method.

3.4 Implementation details

The model has been trained on Google Colab (GPU Tesla T4, 16 GB) from scratch starting from the pretrained architectures available on Hugging Face (*bert-base-uncased*, *roberta-base* and *allenai/longformer-base-4096*). Using Adam as optimizer, Adam epsilon was set at 1e-6 for numerical stability and the learning rate at 0.001. The selected training scheduler is Reduce-On-Plateau, which is quite effective and lowers the learning rate when the performance metric has stopped improving. I have also applied Early Stopping for regulation on ROUGE-L score of the validation set for 20 epochs. The main loss consists of the three ones described by 1, 2 and 3 equations. Depending on the model used for training, the [CLS] and the [SEP] tokens had to be set accordingly: 101 and 102 for Bert, 0 and 2 for RoBerta and LongFormer. The *hidden size* of the encoders layers is left unchanged at 768. Since these type of architectures are quite heavyweight, the only way of fine-tuning them is to freeze some layers. Regarding that, I introduced two different strategies: the first one only unfreezes the 11^{th} (the last) and the pooler layer, and the second one also unfreezes a portion of the 10^{th} layer. These two fine-tuning versions were created to address the lack of high GPU capabilities and, together with the thoughtful choice of batch size, made it possible to train and not go out of memory. In all my experiments, batch size couldn't exceed size 4.

	ROUGE		
Model	**R - 1**	**R - 2**	**R - L**
Baseline	0.684	0.539	0.540
BERT	**0.736**	0.609	0.610
RoBERTa	0.729	**0.616**	**0.620**
LongFormer	0.726	0.607	0.609

Table 1: ROUGE results.

4 Results

For evaluating quantitatively the summary extraction system the **ROUGE** metric [4] is used. In particular ROUGE-1, ROUGE-2 and ROUGE-L. In this case the reference summaries are composed by the ones produced manually by the students of NUANS course. In the cases in which the references summaries to a certain text are more than one, a simple average of the output measure will be computed. We are de facto considering all the students extractive summaries equally.

In Table 1 we can see the results. The *baseline*, as already anticipated, is a SBERT model [7] which produces the top-scoring sentences that make up 30% of the original text. The scores are not that bad but is quite obvious how my models with the custom approach outperform it (+0.08% on ROUGE-L). Talking about my solutions, it is not very clear which one is the best. Perhaps RoBERTa could be considered the best option based solely on ROUGE-L, which typically captures sentence structure with greater accuracy compared to the other two metrics because it takes into account the longest common subsequence. It must be said, however, that ROUGE is an intrinsic evaluation measure, so it doesn't consider the *semantics* of summaries and therefore may not always be completely reliable and its results should be approached with caution.

The outcome it's not quite what I was hoping for and, as it can be observed, the results of the three different encoders are comparable, with no particular one standing out significantly from the others. The limited computing power available is the primary reason for this. I'm confident that by increasing the batch size and training additional encoder layers (instead of only two), I could have significantly enhanced the modeling of the semantic space and improved the overall performance. Additionally, integrating a fine-tuned Pegasus model on Fairysum would have also contributed to the generation of more effective abstractive summaries, along with the possibility of having more summary candidates during both training and inference phases.

The field of extractive summarization is still in its early stages, and current methods have significant limitations. These methods often fail to consider the relationships between adjacent sentences, and it can be challenging to determine the appropriate level of summarization required. In fact, this task is often difficult even for humans, which makes it an exciting and rapidly developing area of research.

References

[1] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pages 4171–4186. Association for Computational Linguistics.

[3] Dragomir R. Radev Gunes Erkan. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization.

[4] Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

[6] Yiran Chen Danqing Wang Xipeng Qiu Xuanjing Huang Ming Zhong, Pengfei Liu. 2020. Extractive summarization as text matching.

[7] Iryna Gurevych Nils Reimers. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks.

[8] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2019. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization.