Text Summarization using Reinforcement Learning

SI-II - MP1

Beatriz Costa
Department of Electronics, Telecommunications, and Informatics.
University of Aveiro
N.Mec: 109657

Luís Silva
Department of Electronics, Telecommunications, and Informatics.
University of Aveiro
N.Mec: 88888

Pedro Carvalho
Department of Electronics, Telecommunications, and Informatics.
University of Aveiro
N.Mec: 84670

Tiago Pinho
Department of Electronics, Telecommunications, and Informatics.
University of Aveiro
N.Mec: 92938

***Abstract*—Alongside the idea of internet there is a limitless amount of data available online. Text summarization is necessary when we think of getting the most precise and useful information from a document and eliminating the irrelevant parts of it.**

Keywords: text summarization, reinforcement learning

I. Introduction

Reinforcement learning is about taking actions in order to maximize reward in a specific situation. Facing a task, the reinforcement learning differs from other types of learning (like supervised learning) in a way that there is no correct answer. The algorithm itself is bound to learn from its experience and choose for itself.

In reinforcement learning we have an initial state and many possible outputs, as there are multiple solutions for a particular problem.

With the evolution of AI research comes new techniques and applications for reinforcement learning. Our focus is to study how RL works and how to apply it focusing on a type of possible application for it. There's where text summarization appears.

Text summarization produces concise and fluent summaries while preserving key information content and overall meaning. It can be done using time-consuming manual methods or through machine algorithms and Artificial Intelligence. The two broad categories of approaches to automatic text summarization are extraction and abstraction.

For this reasons, in this document we aimed on using Reinforcement Learning and exploring the advantages of the different methods for automatic text summarization.

II. Methodology I

As mentioned above there are two different procedures to automatic text summarization:

1) Extractive Summarization - Identifies the relevant sentences or phrases from the original text and extracts and groups them together in order to form a concise summary.
2) Abstractive Summarization - Focuses on the vital information of the original sentences and generates a new set of sentences for the summary, being that this sentences might not be present in the original sentences.

In this work, we choose the abstractive method because we think it is more interesting since it enables the machine to understand the semantic of the text and to create new sentences using Natural Language Processing (NLP), while the other method focuses only on the text itself and forms an non-intelligent machine, leading to an easier implementation, making it more widely used than the abstractive strategy.

A. Dataset

In order to create and train an abstractive approach we decided to use one of kaggle's available datasets "Amazon Fine Food Reviews". This dataset consists of more than 500,000 reviews of foods from the amazon site, which include product and user information, ratings, and, for the text summarization, a plain text review (where we focused). Some examples of the dataset can be found on Fig. 1.

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

Figure 1: Some examples of the dataset used.

B. Pre-Processing of the Data

After the selection of the dataset to be analyzed, the next step was to pre-process this data, since not all of the data is relevant and there is a lot of information that is not the focus of our investigation. Pre-processing also characterizes the process of converting data to something the computer can understand.

So, first, after we uploaded a part of the dataset (200000 examples), we dropped the duplicates and NA values that existed on the database.

Secondly, we did a text and summary cleaning, this means, we converted everything to lowercase, then we remove any existing HTML tags, any "'s", any text inside parentheses, punctuation and special characters. It also used a contraction

map to know the contractions and eliminates stopwords. In NLP, useless words(data), are referred as stopwords, some examples are the words ”the”, ”a”, ”an”, etc...
After doing all that, we added the **START** and **END** special tokens at the beginning and at the end of the summaries. This special tokens are going to be explained in a later phase.
The results of this initial pre-processing can be seen in Fig.2.

```
Review: bought several vitality canned dog food products found good quality product looks like stew processed m
eat smells better labrador finicky appreciates product better
Summary: _START_ good quality dog food  _END_

Review: product arrived labeled jumbo salted peanuts peanuts actually small sized unsalted sure error vendor in
tended represent product jumbo
Summary: _START_ not as advertised  _END_

Review: confection around centuries light pillowy citrus gelatin nuts case filberts cut tiny squares liberally
coated powdered sugar tiny mouthful heaven chewy flavorful highly recommend yummy treat familiar story lewis li
on witch wardrobe treat seduces edmund selling brother sisters witch
Summary: _START_ delight says it all  _END_

Review: looking secret ingredient robitussin believe found got addition root beer extract ordered made cherry s
oda flavor medicinal
Summary: _START_ cough medicine  _END_

Review: great taffy great price wide assortment yummy taffy delivery quick taffy lover deal
Summary: _START_ great taffy  _END_
```

Figure 2: Some examples of some reviews and their summary after the initial stage of pre-processing.

Following that, we analysed the distribution of lengths of the texts and summaries.

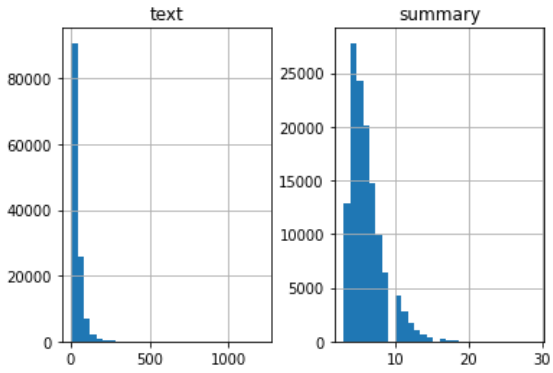

Figure 3: Analysis of the length distribution of the texts and summaries.

Observing the plots present in Fig. 3 we can induce that the majority of reviews have a max size of 100 words and that the summaries have a maximum size 8. This helped us to limit the max number of words to be analysed and reduce the computation time cost.

The final step of pre-processing was preparing the tokenizer, but before that, we split the data in 80% for training and 20% for validation. Relatively, to the tokenizer, tokenizers converts word sequence to an integer sequence.

The function **fit_on_texts** creates the vocabulary index based on word frequency, like for examples, if we have a sentence like ”The cat sat on the mat”, it will create a dictionary like word_index[”The”] = 1, word_index[”cat”] = 2, ..., so every word gets a unique integer value and the lower the integer the more frequent that word appears.

The function **texts_to_sequences** transforms each text to a sequence of integers, so basically takes each word in the text and replaces it with its corresponding integer value from the dictionary referenced above.

After that, we were all ready to build the model and train our network.

C. Architecture

There are a few components to the Sequence-to-Sequence (Seq2Seq) architecture. This model takes in a stream of sentences as input and produces as output a different stream of sentences.

1) Long-Short-Term Memory (LSTM) - Both the encoder and decoder structures, discussed below, are based on this structure. LSTM is a Recursive Neural Network (RNN) that is capable of learning long-term dependencies. At each timestep the LSTM produces values for the hidden-state and cell-state and these are returned as output.

2) Encoder - The encoder reads the input sequence one token at a time, in this case the words that compose the reviews, and in each timestep it captures the contextual information present in the input. In this model we create a stack of three sequentially interchained LSTM structures.

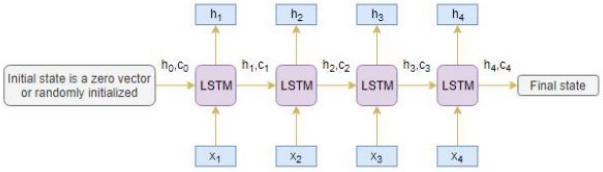

Figure 4: Encoder Structure

3) Decoder - The decoder is an LSTM whose input is set to the resulting internal state of the encoder. Given this initial state the decoder starts generating the output sequence one word at a time. With each new addition the decoder takes the whole sequence into consideration when generating the next word.

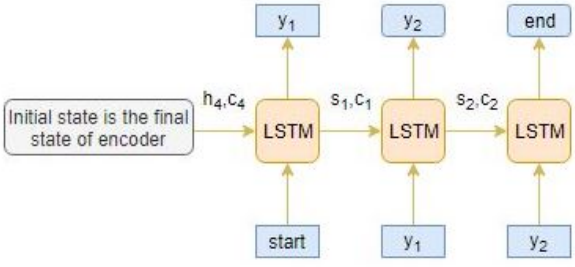

Figure 5: Decoder Structure

4) Attention Layer - The attention layer is a mechanism that helps a neural network to memorize long sequences of information. As discussed before the encoder compresses the sequential input and processes the input in the form of a context vector. Introducing an attention layer creates a shortcut between the entire input and the context vector. The attention layer allows the context vector have the information about the encoder hidden states, decoder hidden states and alignment between source and target.
It represents the quantity of attention that we need to pay to every word in the input sequence for generating a word at timestep t, this means, instead of looking at all the words in the sequence, we can increase the importance of a specific part of the original sequence that results in the target sequence. We can have global or local attention, and the differences between them are that, one pays attention to all source positions, so all hidden states of the encoder are considered for deriving the attended context vector, and the other, only a few of the hidden states are considered, respectively.

5) Inference Phase - After the training, the model was tested with new data, so new source sequences for which the target sequence is unknown, so we built a inference architecture to decode a test sequence. First, we encoded the entire input sequence and initialize the decoder with internal states of the encoder. We then pass the **START** token as an input of the decoder and run it for one timestep, with the internal states. This gave us an output that corresponds to the probability for the next word and the word with the highest probability was selected. After that, we give that selected word as input of the next decoder in the next timestep and update the internal states, and repeat the previous process until we generate the word **END** or hit the maximum length of the target summary.

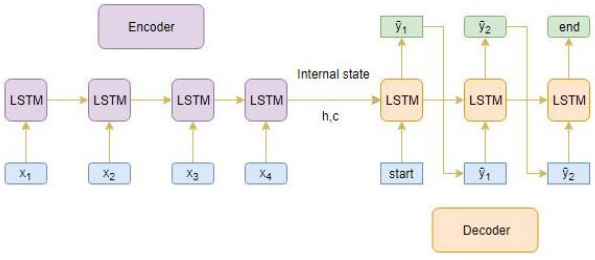

Figure 6: Inference Phase Structure

D. Implementation of the strategy

In Fig.7 is a graphic representation of the architecture implemented.

Is worth noting that we have 3 layers beyond those that constitute the encoder and decoder layers, explained above, that are:

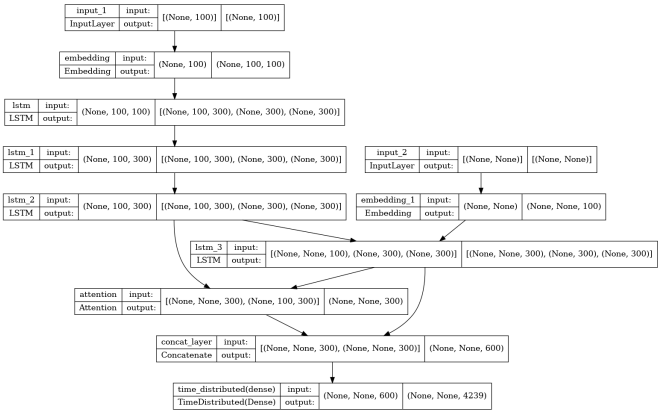

Figure 7: Block diagram of the architecture used.

- Embedding layer - Enables conversion of each word into a fixed length vector of a previously defined size. Works like a lookup table.
- Concatenate layer - As the name suggests, it concatenates a list of inputs.
- TimeDistributed layer - This wrapper allows to apply a layer to every temporal slice of an input.

After defining the model, we chose 2 different optimizers to compare the results, **rmsprop**, since it converts the integer sequence to a one-hot vector on the fly, and **Adam**, which is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments and we the same loss function for both, **sparse categorical cross-entropy**.
It is worth mention that we used early stopping in all runs, which is used to stop training the neural network at the right time by monitoring a user specified metric.
Then we set the inference model to obtain the results present in the next section.

E. Results Analysis

At first glance the results are satisfactory, the summaries generated appear to be more than direct extractions of information present in the reviews, sometimes even matching the expected summary, but upon further inspection patterns become noticeable and it becomes clear that the generated sequences are repetitive and really general.

```
Review: best sugar cubes use office combine brown sugar cubes get used time aesthetically pleasing
Original summary: fave sugar cubes
Predicted summary:  great product


Review: cashews hoping large fresh met expectations recommend koeze anyone looking purchase especially gift d
isappointed
Original summary: great
Predicted summary:  great product


Review: local grocery store quit carrying couple years ago daughters crushed found amazon december bought cas
e christmas absolutely speechless hero hope betty crocker never quits making understand stores carry call dis
tributors make stink wanted order amazon available daughter wants birthday please amazon get make little girl
happy
Original summary: my kids love them
Predicted summary:  great product
```

Figure 8: Repetition in classification

In Fig. 8 we present examples of the repetitiveness and unspecific results. It is clear that the model falls back on ”great

product" as a general response to most scenarios. Although it is not wrong per se, this is not the behaviour that expected from the model.

```
Review: use nespresso roma wife uses nespresso bought cremosa espressivo possible substitutes lower per capsu
le price free shipping amazon prime use recycle like coffee others may prefer sure us weak like flavor cheap
maybe picky use time trying cremosa espressivo wife none one life little pleasures especially good coffee sub
stitute
Original summary: too weak vs nespresso and
Predicted summary:  not worth the money


Review: would think rotten reviews would turn away peanut butter thought looked like peanut butter kind smell
s like peanut butter dipped knife taste prepared kinds bad really good stick roof mouth thick like peanut but
ter spreadable little sweeter full calorie versions used tried without jam crackers could eat pb problems tha
nk walden farms gonna save calories month help
Original summary: there must be something wrong with me
Predicted summary:  not so good
```

Figure 9: Good results in classification

In Fig. 9 we present examples of satisfactory results. Although these generated summaries differ from the expected ones, an interpretation of the context is evident.

```
Review: love clif kid bars decided try based great reviews bought strawberry one house likes kids picky alway
s try treats see even worth trying eat healthy mind healthy snacks even care going try another flavor strawbe
rry cut
Original summary: not what expected
Predicted summary:  great for the price
```

Figure 10: Error in classification

In Fig. 10 we present examples of an error. Although rare, these instances of misinterpretation do exist.

To boost these results the next step is to implement reinforcement learning. Upon investigation based on available articles and thesis on the topic of Reinforcement Learning, in order to apply an optimization to our algorithm we came to conclude that the level of difficulty in the implementation of RL exceeds our knowledge in this area. We did although come across a solution that uses the foundation that we setup but is successful in implementing RL to the model. Even though the computational requirements exceed what we have available, we still set it in our goals to explore, comprehend and present the findings and implementation of this next stage, and so, we will now dive into the full implementation of Text Summarization with Reinforcement Learning.
For a matter of example, the next methodology we are going to present takes approximately 20 days to run in a machine with better PC setup than the setups used.

III. Related Work

Text summarization has been vastly studied in latest years. We now introduce the related works of abstractive approaches and reinforcement learning for optimization of an abstractive summarization. In the section IV we will focus on presenting and explaining a specific approach to this strategy.

The first to apply the model based on seq2seq to abstractive text summarization [1] alongside with other new methods (e.g., [2]), have achieved effectively results.

The problem with implementation of abstractive models, despite it being concise on generating new words, is the loss of information during training and high computational cost, due to the large inputs that are needed, this means that it is created, most of the time, grounded-truth summaries.

Taking this problem into consideration, the models [3], [4] combined the advantages of both of the automatic text summarization categories and proposed a hybrid extractive-abstractive architecture. The method used goes through an extractive network to extract the sentences with obvious semantics from the input, to then the abstractive network summarize those sentences and compute the final text summary.

Previous studies ([5], [2], [3], [6]) have used reinforcement learning [7] in order to eliminate the existing problem of exposure bias in these models which are built on seq2seq framework. Specifically, reference [5] used reinforcement learning policy gradient methods for abstractive summarization, while [2] used actor-critic policy methods. In other hand, reference [6] combined agents with an independent and/or cooperative behaviour to form a training model by reinforcement learning.

IV. Methodology II

A. Dataset

Gigaword is presently the largest static compilation of English news documents available. The most recent edition, *English Gigaword Fifth Edition (Parker et al., 2011)* [8], contains nearly 10-million documents, with a total of more than 4-billion words. This dataset is one of the most used in studies globally due to its richness in semantics and vocabulary. Some examples of the corpus present in this dataset can be found on Fig. 11.

Source	#Files	GB	Words	#Docs
AFE	44	1.2	170,969,000	656269
APW	91	3.6	539,665,000	1477466
NYT	96	5.9	914,159,000	1298498
XIE	83	0.9	131,711,000	679007

Figure 11: Some statistics of the documents available on the dataset Gigaword.

B. Pre-Processing of the Data

In this methodology, the pre-processing done to the data, was basically the same explained in II but instead of using a database present in a csv file, the dataset files were in a txt format and had to be converted to a bin format because the model only accepts data in that format. The author provides a python file, called make_data_files.py that does this conversion, but when we ran it, it gave some errors and we found out that we had to make some adjustments, in particular, on lines 1 and 2 from the code in Listing 1.

```
1 tf_example.features.feature["article"].bytes_list.
      value.extend([b"article"])
2 tf_example.features.feature["abstract"].bytes_list.
      value.extend([b"abstract"])
```

Listing 1: Adjusts to the file provided in order to run the file.

C. ROUGE

Machine Learning projects based on Natural Language Processing struggle to define how to measure accuracy, so how do we measure the accuracy when dealing with language summarization? There's where ROUGE comes in.
But before we start to explain the different types we have to explain some concepts like recall, precision and F1 - Score.

1) Recall: This metric counts the number of overlapping n-grams (grouping of tokens/words) found in both the model output and reference and divide it by the total number of n-grams in the reference.
This ensure us if the model is capturing all the information contained in the reference or not. An example of the calculus of this metric can be found on Fig.12.

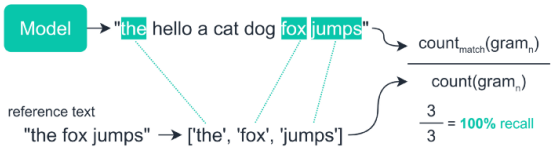

Figure 12: Recall metric calculus example.

2) Precision: This metric helps covering one issue that we get by only analysing the recall parameter, that is that we can't guarantee just with the recall metric that our model isn't just pushing out a huge number of words.
The calculus of this parameter is similar to recall but instead of dividing by the reference n-gram count, we divide by the model n-gram count. We can observe an example of the measurement of this parameter on Fig.13

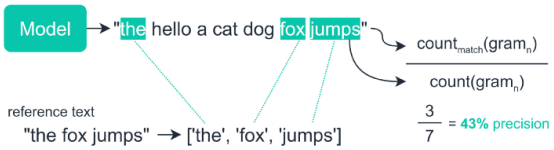

Figure 13: Precision metric calculus example.

3) F1 - Score: The calculus of this parameter uses both the parameters mention before, by multiplying 2 by the multiplication of the two parameters and dividing it by the sum of the parameters.
This measurement show us how good was the performance of our model. In Fig.14 we can see an example of the calculus of this parameter.

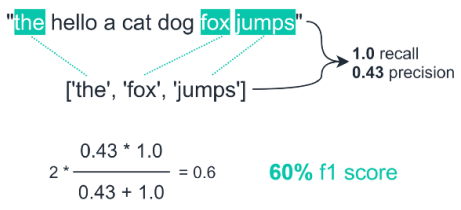

Figure 14: F1 - Score metric calculus example.

4) Types of ROUGE: ROUGE or Recall-Oriented Understudy for Gisting Evaluation to be more explicit, it's a set of metrics that simplify the problem presented before.
There are a lot of of ROUGE types.

- ROUGE - N
- ROUGE - L
- ROUGE - S

ROUGE - N measures the number of matching n-grams between our model generated text and a reference

ROUGE - L measures the longest common subsequence (**LCS**) between our model output and reference, which means we count the longest sequence of tokens that is shared between both. We can view on Fig.15 an example.

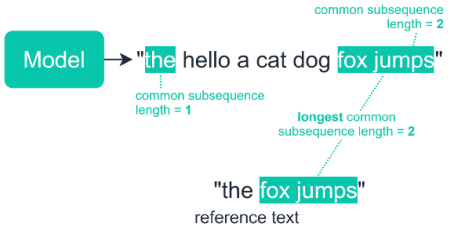

Figure 15: ROUGE - L example.

We can apply our recall and precision calculations as before but we replace the match with **LCS**. The idea is that a longer shared sequence would indicate more similarity between the two sequences.

Finally, the ROUGE - S metric, also called, skip-gram concurrence metric, allow us to search for consecutive words from the reference text, that appear in the model output but are separated by one or more words. We can take as example the Figs.16 and 17. So ROUGE - S allows us to add a degree of leniency to our n-gram matching.

5) Cons of ROUGE metric: ROUGE only measures syntactical matches rather than semantics, so if we had two sequences with the same meaning but express with different words they would have a low ROUGE score.
Rouge also as low consideration for fluency and readability and high ROUGE scores summaries usually obtain low human appealing.

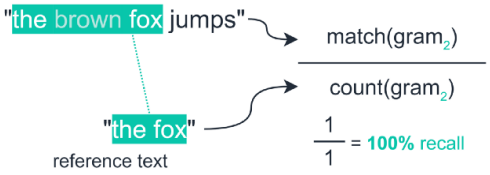

Figure 16: ROUGE S - recall calculus example.

D. Architecture

This beginning of this method goes in line with the method explained before, but as said before it uses RF, but the question here is how and where it uses?
This methodology follows the principles of the articles [5] and [9], so what it differs from methodology 1 is that it maximizes

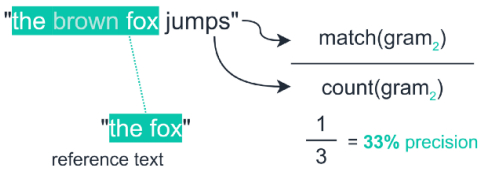

Figure 17: ROUGE S - precision calculus example.

a specific discrete metric, ROUGE, instead of, minimizing the maximum-likelihood loss, using a self-critical policy gradient training algorithm. In this algorithm we produce two separate output sequences at each training iteration, one obtained by sampling from the probability distribution at each decoding time step, and other obtained by maximizing the output probability distribution at each time step, performing a greedy search. Then we define a reward function and compare it with the ground truth sequence with the evaluation metric of our choice.

One potential issue of this model is that optimizing for a specific discrete metric like ROUGE does not guarantee an increase in the summaries quality and readability.

E. Results

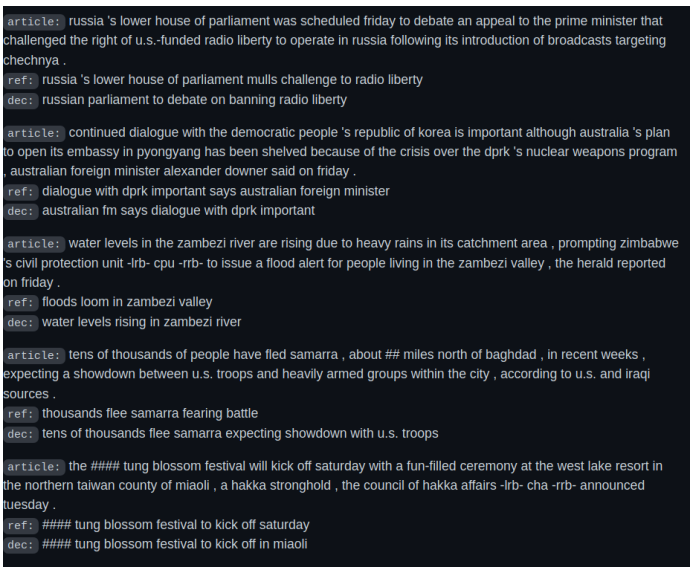

Figure 18: Results obtained by the author with methodology 2.

V. Performance comparison

A. Method I vs Method II

To have a clear image of the effects produced by the integration of RL in text summarization, we ran our developed work over the dataset used by second methodology.

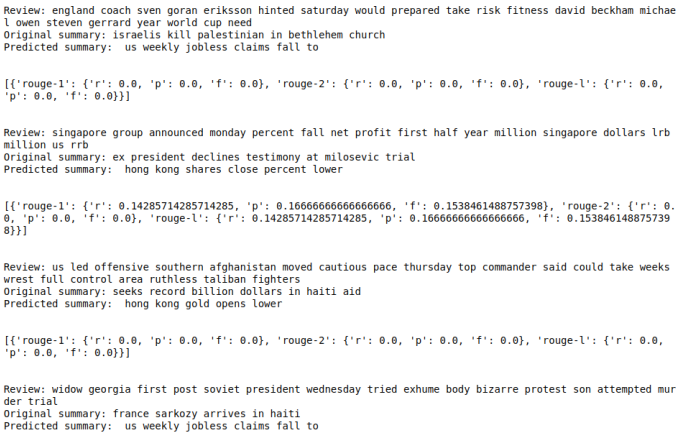

Figure 19: Results obtained by running our work over the second methodology's dataset.

Comparing the results obtained in Fig. 18 and Fig. 19, a distinction is apparent. The summaries generated when using RL are highly specialized and adapt very well to change in subject, this is in stark contrast with the summaries generated by applying only the Seq2Seq architecture where, as discussed previously, the results are repetitive and general.

These results also allow for another very interesting analysis which is the impact of the dataset on the results, namely the ones from the Seq2Seq architecture. When using the Amazon dataset even though the results were not the desired quality wise, they were at least sensible but upon changing our dataset the results, apart from the their characteristic repetitiveness and generality, they are nonsensical most of the time and it is here that we see the effects of different datasets. In the first instance, the dataset portrayed reviews of products where, despite the level of detail employed by the user, there is always an underlying appraisal that can be summarised in summaries like "not so good" or "great product" depending on whether or not the reviewer took a liking to the product. In the second instance the dataset portrayed news articles that are by nature unopinionated and on the most varied subjects from market crashes to floodings and matters of political affairs, for this data it is impossible to produce a general, one-fits-all response to be used as a blanket statement and so no satisfactory results could be achieved without the use of RL.

VI. Conclusion

In conclusion, as you could see the second methodology presents far better results, most because of the use of reinforcement learning. It is possible enhance the quality of abstractive summarization.

For better results, in a future approach, we could try to use different metrics since ROUGE as the limitations mentioned before.

References

[1] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," *arXiv preprint arXiv:1509.00685*, 2015.

[2] P. Li, L. Bing, and W. Lam, "Actor-critic based training framework for abstractive summarization," *arXiv preprint arXiv:1803.11070*, 2018.
[3] Y.-C. Chen and M. Bansal, "Fast abstractive summarization with reinforce-selected sentence rewriting," *arXiv preprint arXiv:1805.11080*, 2018.
[4] W.-T. Hsu, C.-K. Lin, M.-Y. Lee, K. Min, J. Tang, and M. Sun, "A unified model for extractive and abstractive summarization using inconsistency loss," *arXiv preprint arXiv:1805.06266*, 2018.
[5] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," *arXiv preprint arXiv:1705.04304*, 2017.
[6] A. Celikyilmaz, A. Bosselut, X. He, and Y. Choi, "Deep communicating agents for abstractive summarization," *arXiv preprint arXiv:1803.10357*, 2018.
[7] R. Sutton and A. Barto, "Introduction to reinforcement learning. cambridge, ma," 1998.
[8] D. G. Robert Parker, *English Gigaword Fifth Edition*. Linguistic Data Consortium, 2011.
[9] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," 2017.