A Hierarchical Structured Self-Attentive Model for Extractive Document Summarization (HSSAS)

Koren Maliniak (ID. 200319572) Or Stern (ID. 305698664)

Submitted as a final project report for Natural Language Processing, IDC, 2019

1 Introduction

Text summarization refers to the technique of shortening long pieces of text. The intention is to create a coherent and fluent summary having only the main points outlined in the document.

We chose an article with innovative approach for extractive text summarization with a complex neural network, so it was a great opportunity for us to get an experience with building network from scratch.

The article: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8344797

1.1 The problem

In these days, we are flooded with information, much more then we can handle. Especially in professional fields where new articles are published on a daily basis. Hence, we have to find a good way to summarize all this information. The model in the article is built exactly for this problem. It summarizes a given document by extracting the most relevant sentences.

1.2 Related Works

- M. Kågebäck, O. Mogren, N. Tahmasebi, and D. Dubhashi, "Extractive Summarization using Continuous Vector Space Models"
- Z. Cao, W. Li, S. Li, and F. Wei, "AttSum: Joint learning of focusing and summarization with neural attention"
- R. Nallapati, B. Zhou, and M. Ma. (Nov. 2016). "Classify or select: Neural architectures for extractive document summarization"

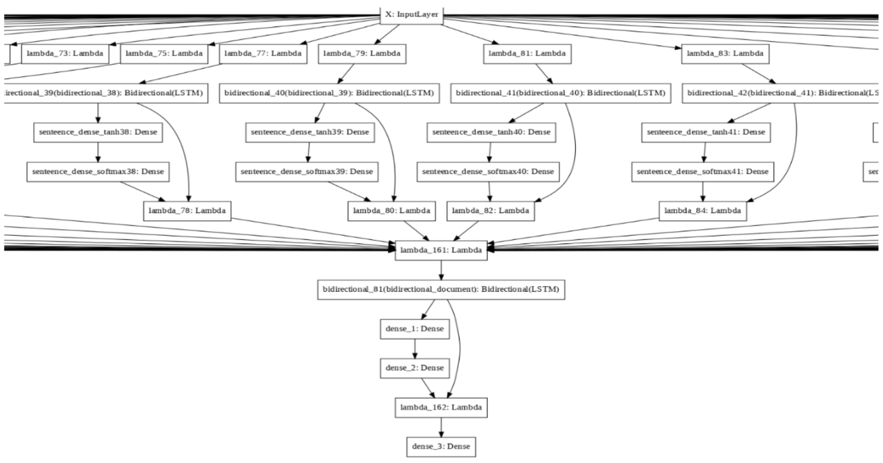

X: InputLayer
lambda_73: Lambda
lambda_75: Lambda
lambda_77: Lambda
lambda_79: Lambda
lambda_81: Lambda
lambda_83: Lambda
irectional_39(bidirectional_38): Bidirectional(LSTM)
bidirectional_40(bidirectional_39): Bidirectional(LSTM)
bidirectional_41(bidirectional_40): Bidirectional(LSTM)
bidirectional_42(bidirectional_41): Bidirectional(LS
senteence_dense_tanh38: Dense
senteence_dense_tanh39: Dense
senteence_dense_tanh40: Dense
senteence_dense_tanh41: Dense
senteence_dense_softmax38: Dense
senteence_dense_softmax39: Dense
senteence_dense_softmax40: Dense
senteence_dense_softmax41: Dense
lambda_78: Lambda
lambda_80: Lambda
lambda_82: Lambda
lambda_84: Lambda
lambda_161: Lambda
bidirectional_81(bidirectional_document): Bidirectional(LSTM)
dense_1: Dense
dense_2: Dense
lambda_162: Lambda
dense_3: Dense

Figure 1: Our Model

2 Solution

2.1 General approach

We first read the article multiple times to understand the proposed network, and what is the meaning of every part in it. Then we started implementing it step by step. The first thing was building the model and running it on some random data - just to see that the network is built properly. After, we approached to the mission of preparing the real data to fit to our model. The last thing was training the model on the real data and analyze the results. You can see a screenshot of our model in figure 1.

2.2 Model Architecture

The model (figure 2) is built from 3 levels hierarchy. It receives as an input a multidimensional matrix representing a document. The matrix built from list of sentences, each one of them is a list of words embeddings.

The first level in the model's hierarchy is built from multiple blocks of LSTM + Attention, each one of them receives a sentence as input (word after word), and the output of the block is a vector representing "sentence embeddings".

The second level in the model's hierarchy is a single block of LSTM + Attention that receives all the sentences embeddings (sentence after sentence), and the output of the block is a vector representing "document embeddings".

The last level in the model's hierarchy is a custom layer that receives as an input the document embeddings, all the sentences embeddings and the return sequence of the LSTM from the second level. This layer makes classification by extracting features that decides the probability of each sentence to be in the summary.

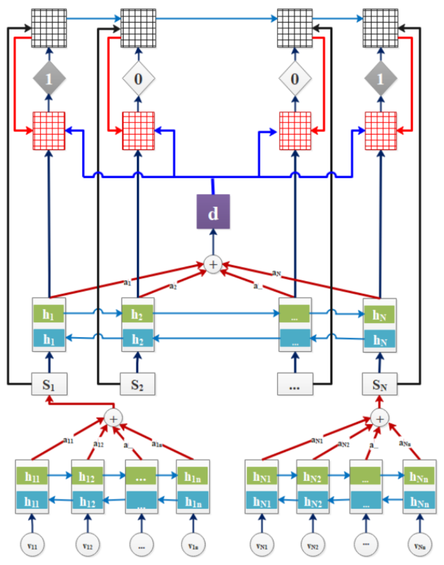


Figure 2: The Model's Architecture

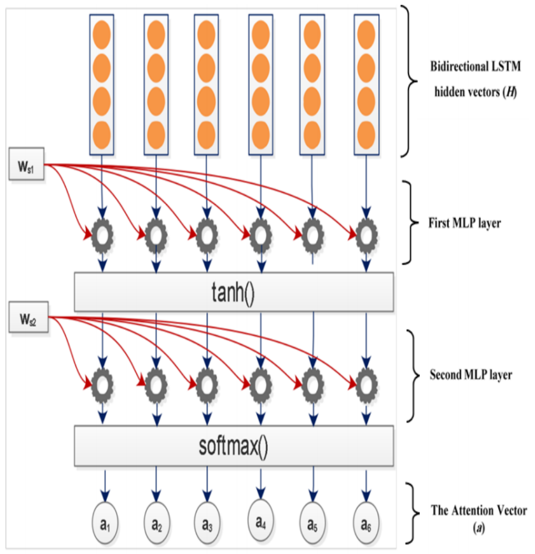


Figure 3: The Attention Layer

2.3 Attention Layer

The attention layer (figure 3) receives as an input the results of each timestamp in the LSTM, and passes into 2 dense layers:

1. tanh - for non-linearity
2. softmax - to get probabilities

Then, the output probabilities are multiplied by the input embeddings, and then all the results are summed into one vector.

2.4 Classification Layer

In the classification layer we extracted features and used them to calculate the sentences' probabilities. The features are:

- Information content - represented as sentence's embeddings
- Salience - represented as multiplication of sentence's embeddings and the document's embeddings
- Novelty - represented as multiplication of sentence's embeddings and another parameter called "The document's state". This parameter is the sum of probabilities of the sentences before the current. The probability decreases when the current sentence is more advanced in the document.

- Position - the result of the final LSTM at the timestamp of the sentence

All these features are learnable. The full description of how to calculate each one of them is written in the article.

2.5 Data Preparation

We used CNN/Daily-Mail dataset. After a long search we found an article (R. Nallapati, F. Zhai, and B. Zhou, "SummaRuNNer: A recurrent neural network based sequence model for extractive summarization of documents") that implemented the data preparation to a convenient structure. We used this data as a starting point, then we still had to separate the sentences and their labels, tokenize the sentences, calculate word embeddings and create input matrix with the structure that fits our model:

- N samples
- 80 sentences per sample
- 50 words per sentence
- Vector of size 300 as word embeddings

3 Challenges

During the work we encountered many challenges, both in understanding some thins and also in the implementation itself. In the end we found the ways to deal with most of them. Here are some of the main challenges.

- We couldn't use the sequential api - so we had to create more basic items (layers) and connect them manually using Keras functional api.
- We learned how to create shared layers outside the model and connect them to the model. These layers are the blocks that handle the words and create the sentence embeddings. The reason these layers are initiated outside the main model and are called from inside the model is because their weights are shared between all samples.
- We learned how to create custom layer with Keras (the classification layer). During the implementation of the layer we found some mistakes in the article's notations and mismatch in sizes of the features. We tried to reach the article's authors but they didn't answer. Because of this, we trained the final model twice - first with the classification layer using the correct size of the features (as we understood it should be) and second with a dense softmax layer instead of this custom layer.
- Because we had to make a classification to each sentence - we couldn't use the regular cross-entropy loss function of Keras, and we had to implement our own loss function.

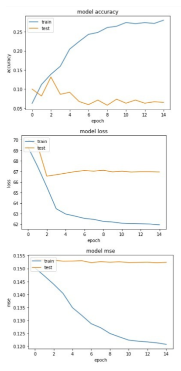
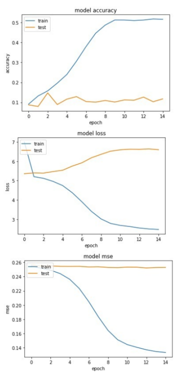
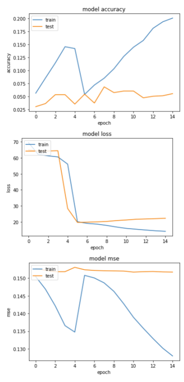
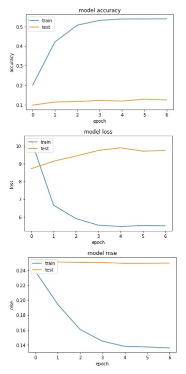
- We faced memory issues in Google Colab so we had to explore and try many parameters to optimize the memory usage while not harming the model. Therefore, we preferred to decrease the size of each sample instead of using less samples. In each sample we reduced the word embeddings size, and the number of sentences per document. We tried to reduce the maximal number of words in each sentence but it did harm the model (because sometimes the last words in the sentence makes it important). By changing number of sentences we basically changed the problem at hand.

4 Experimental results

The proposed method to measure the model in the article is ROUGE (Recall-Oriented Understudy for Gisting Evaluation). ROUGE compares n-grams between the model's summary and the given input summary. The article's authors measured their model with ROUGE-1, ROUGE-2 and ROUGE-L (Longest Common Subsequence). The reason they chose ROUGE method is because they compared their model to other models that are not extractive and therefore doesn't have labels for sentences. They also explain that ROUGE is not necessarily good because it doesn't indicate on readability of the summary.

We didn't implement the ROUGE measurement on our model because we didn't have to compare our results to other abstractive models - we used the labels and tested our model on the test data.

As a result of memory problems with free Google Colab we had to run the model compromising some parameters. Be it the number of examples, the number of sentences per document, or the size of each word embedding. Eventually we chose 4 different configurations as described in the table below. Those configurations maxed some parameters (embedding size, number of examples) and still managed to fit in memory and not crash. As one can see the results did not exceed the 11% accuracy mark. We hope that with better machine to run the model those compromises can be mitigated to achieve better results.

Param \ Conf #	1	2	3	4
Sentences per doc	40	10	40	10
Words per sentence	30	40	30	40
Words embeddings size	50	300	50	300
# of samples	10,000	20,000	10,000	20,000
Classification layer	custom	custom	dense	dense
Accuracy	6%	10%	5%	11%
Results				

Results Conclusions:

- The architecture with the custom layer over-fitted to train from the 4th epoch. It could continue working more but Colab disconnected so we managed to run it 15 epochs.
- The architecture with the dense arrived to early stopping after 6 epochs only, meaning that there was a limit to the over-fit that it could made to the train (make sense, less parameters).
- In every configuration we couldn't reach more then 11% accuracy on the test and validation.
- We could use different probability threshold for classification of 1, and maybe get better results. It could be an idea for future research. We tried it manually and it looked promising but we did not automate it.

5 Discussion

The project was very interesting and challenging. For both of us it was the first time we implemented an article from A to Z completely by ourselves. During the work we faced much more challenges and difficulties then we thought we would, and it made us read and explore more about neural networks in general and especially about implementation details with Keras. In the end of it, we both feel proud that we succeeded to overcome the obstacles and build a working model.

As a future work we think it might be interesting to build some abstractive summarization model and connect it to our model, with lower probability threshold for classification of 1. So basically our model will perform as a filter of the non relevant sentences and shorten the document, and the abstractive model would generate a summarization based on that.

In conclusion, we enjoyed working on the project, it was a great opportunity to learn about building complex neural networks for a real-life NLP problem and we made the most of it.

6 Code

The code is here: https://colab.research.google.com/drive/1fc2GYdHcewXZRS-pU1ZDqntKKjulvFlx.