Reproducing Summarizing Long-Form Document with Rich Discourse Information

Giovanni Poggio
Politecnico di Torino
s287775@studenti.polito.it

Giovanni Calleris
Politecnico di Torino
s281262@studenti.polito.it

***Abstract*—This work is about summarizing long-form documents by reproducing the model (*HEROES*) presented in the article *Summarizing Long-Form Document with Rich Discourse Information* (Zhu et al.) [1]. The model is composed by the *Content Ranking* and the *Extractive Summarization* modules. The former creates a digest of sections and sentences. The latter relies on Graph Attention Networks to build heterogeneous graphs from which the most important sentences are selected. Here, we report the steps followed, we propose our variations, and compare the obtained results. In the reference paper [1], the *type-specific transformation matrix* used "to project the features of different types of semantic nodes into the corresponding representation space" is not discussed. To do this step we applied the *attention mechanism*. The result we obtained (Rouge-L-sum: 41.41%) is in line with the one reported by the paper (Rouge-L-sum: 43.33%). We also propose a *non-recursive* model coherent with this approach, further training on the *Content Ranking* module, and the addition of a scheduler.**

The code is available in our GitHub repository[1].

I. Problem statement

The purpose of this project is to summarize long-form documents by replicating the architecture proposed by Zhu et al. [1]. The model, *HEROES* (**H**ierarchy-**E**nhanced Graph for **R**ich-disc**O**urse Document **E**xtractive **S**ummarization), is composed of two main modules: *Content Ranking* and *Extractive Summarization*. The former aims at selecting the most relevant sections and sentences within the selected sections, in our case 4 and 30 respectively. The latter builds a heterogeneous graph from the *digest* (the previously chosen sections and sentences) to highlight the sentences that will compose the final summary.

II. Related Work

To perform extractive summarization, Wang et al. [2] used a graph neural network, which includes nodes of different granularity (words and sentences), to learn cross-sentence relationships. Jia et al. [3] proposed the use of the Graph Attention Networks (*GAT*) [5] layer to select the more relevant sentences to include in the summary.

[1]GitHub: https://github.com/giovannipogg/ourheroes

III. Methods

A. The Dataset

We used the PubMed dataset. It is composed of 119924 documents of different lengths. To have cleaner data and avoid GPU memory issues, we removed those documents whose summary was longer than the body in terms of sentences or based on the number of tokens. We have also excluded documents whose summary or body had no sentences or tokens. We then inspected statistics about sections to filter documents containing very long ones. After having done this, the retained files account for about 70% of the original dataset.

B. Pre-processing and Word Embedding

Our pre-processing pipeline consists in tokenizing the article while leaving untouched the structure of sections and sentences. Then, we used a pre-trained 300-dimensional GloVe [7] to get the word vector representations. We removed the words which could not be vectorized (most of which were LaTeX code present in some articles).

C. Content Ranking Module

We train the *Content Ranking* module to predict the similarity of each section and sentence to the gold summary. The training of this module is done independently from that of the *Extractive Summarization* module.

In this section we describe how this module is structured and trained.

1) Title Representation: A BiLSTM [6] layer is applied to the title to obtain contextualized word representations. Given that not all the words contribute equally to the semantics of the title, upon this contextualized word representation it is applied the *attention* [4] mechanism:

$$u_i = \tanh\left(W_{\text{att}} h_i + b_{\text{att}}\right)$$

$$\alpha_i = \frac{\exp(u_{\text{i}}^{\intercal} u_{\text{att}})}{\sum_i \exp(u_{\text{i}}^{\intercal} u_{\text{att}})}$$

$$v_t = \sum_i \alpha_{\text{i}} h_{\text{i}}$$

where, in this case, h_i is the output of the BiLSTM.

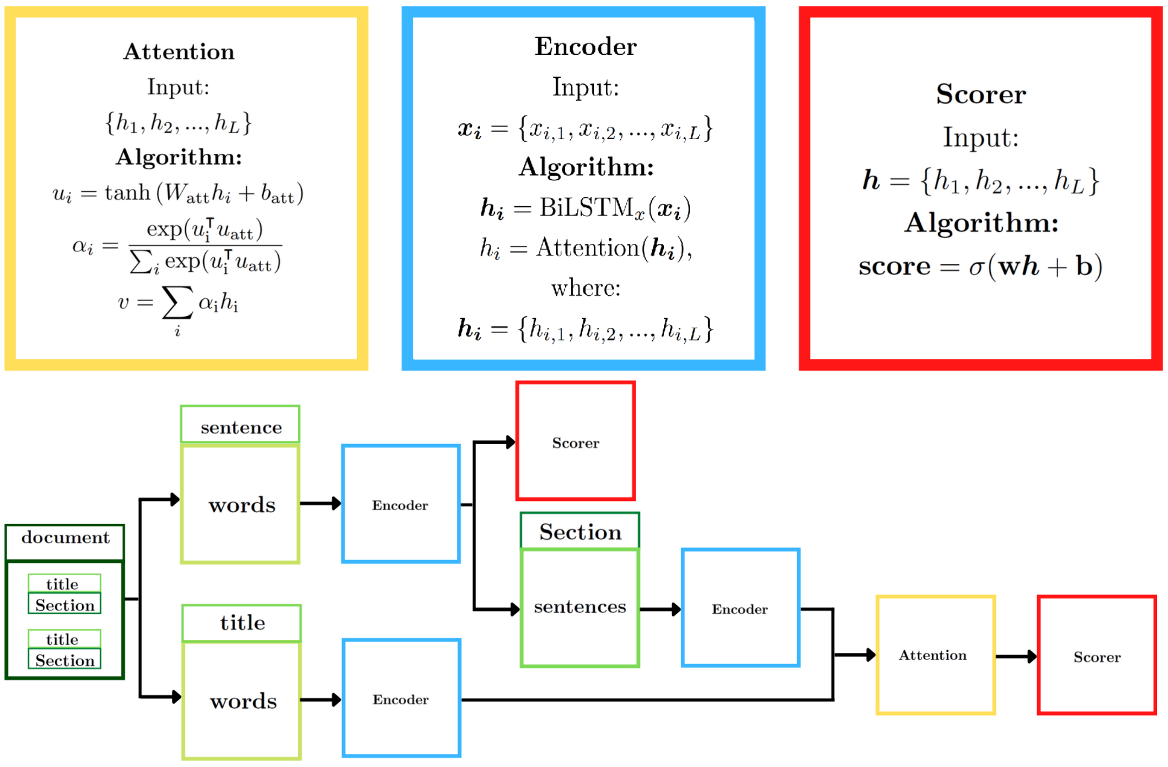
Attention
Input:
$\{h_1, h_2, ..., h_L\}$
Algorithm:
$u_i = \tanh(W_{att} h_i + b_{att})$
$\alpha_i = \frac{\exp(u_i^\intercal u_{att})}{\sum_i \exp(u_i^\intercal u_{att})}$
$v = \sum_i \alpha_i h_i$
Encoder
Input:
$\boldsymbol{x_i} = \{x_{i,1}, x_{i,2}, ..., x_{i,L}\}$
Algorithm:
$\boldsymbol{h_i} = \mathrm{BiLSTM}_x(\boldsymbol{x_i})$
$h_i = \mathrm{Attention}(\boldsymbol{h_i})$,
where:
$\boldsymbol{h_i} = \{h_{i,1}, h_{i,2}, ..., h_{i,L}\}$
Scorer
Input:
$\boldsymbol{h} = \{h_1, h_2, ..., h_L\}$
Algorithm:
$\mathrm{score} = \sigma(\mathbf{w}\boldsymbol{h} + \mathbf{b})$
document
title
Section
title
Section
sentence
words
title
words
Encoder
Encoder
Scorer
Section
sentences
Encoder
Attention
Scorer

Figure 1. The *Content Ranking* module architecture.

2) Content Representation: To get sentence and section representations, we proceed similarly to the previous step: we get sentence representations from the words that compose it. From sentences, we get the section representations for each section.

We apply, on words and sentences, first a BiLSTM and then the attention mechanism to get the higher level representations, sentences and sections respectively:

$$\boldsymbol{h_i} = \mathrm{BiLSTM}_x(\boldsymbol{x_i})$$
$$h_i = \mathrm{Attention}(\boldsymbol{h_i}),$$

where:

$$\boldsymbol{x_i} = \{x_{i,1}, x_{i,2}, ..., x_{i,L}\}$$
$$\boldsymbol{h_i} = \{h_{i,1}, h_{i,2}, ..., h_{i,L}\}$$

Initially, $\boldsymbol{x_i}$ is the sequence of words in a sentence and, then, the sequence of sentences in a section. We obtain the sentence and section representation from these computations. Note that we used different parameters for the previous steps. To get sentence representation, we used:

```
{"input_size": 300;
"hidden_size": 512}
```

300 since it is the embedding size of GloVe.
To get section representation, we used:

```
{"input_size": 1024;
"hidden_size": 512}
```

1024 since the BiLSTM output size is double its `hidden_size` (2×512).

3) Sentence & Section Scoring: The *scorer* is a sub-module composed of a linear layer with a sigmoid activation function. To get the importance scores of each sentence, we passed their representation to the scorer sub-module whose `input_shape` is double the `hidden_size` of the last BiLSTM.
For sections, the attention mechanism was applied to compute sections' importance scores upon stacking title and section representations.

$$\mathrm{output} = \mathrm{Attention}(\{\mathrm{Title, Section}\})$$

Then we applied a dedicated scorer sub-module to obtain the section scores.

4) Training: We trained the *Content Ranking* module for 5 epochs with batch size 32 and learning rate 1e-4. The loss used is binary cross entropy with respect to the ROUGE-2 recall of each section against the gold summary and, for sentences, the averaged ROUGE-1/2/L-sum F1 against sentences in the gold summary.

$$L_{BCE} = -\frac{1}{N}\sum_i y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)$$

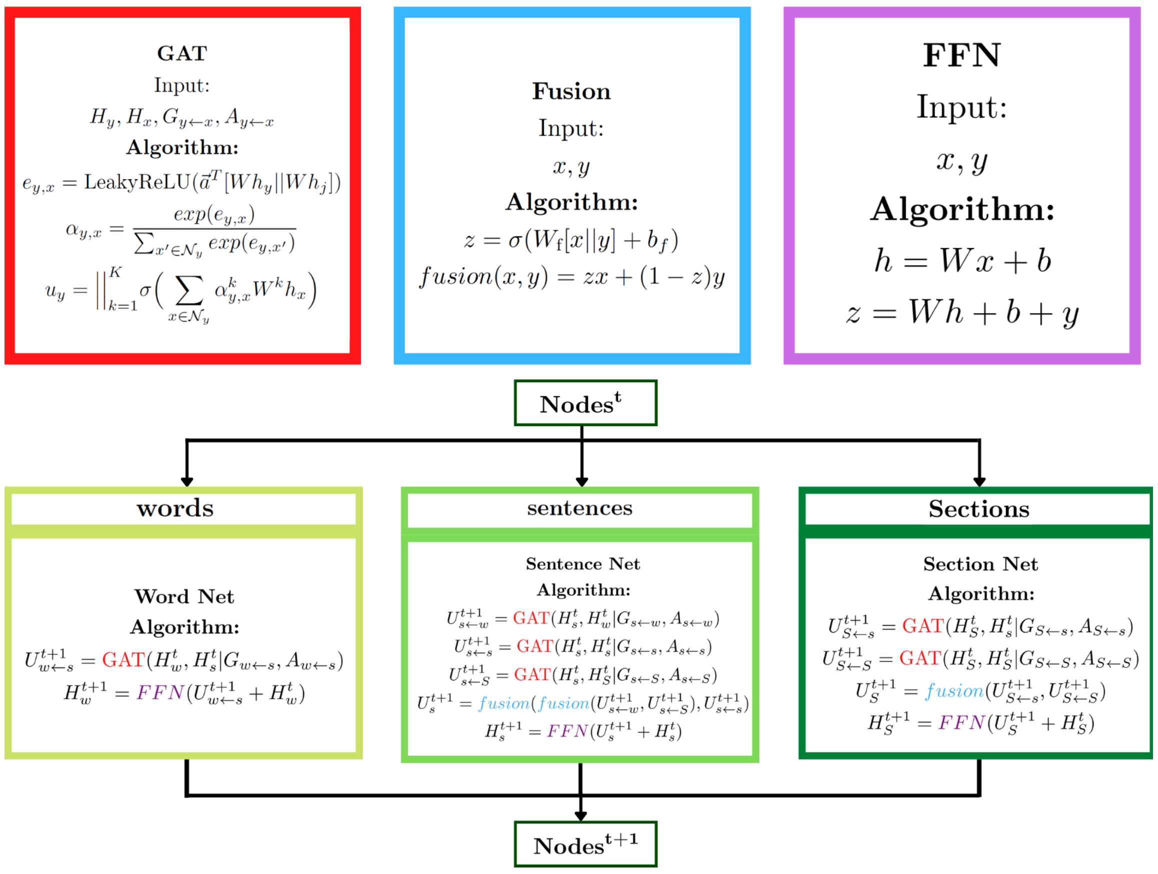
GAT
Input:
$H_y, H_x, G_{y\leftarrow x}, A_{y\leftarrow x}$
Algorithm:
$e_{y,x} = \text{LeakyReLU}(\vec{a}^T[Wh_y||Wh_j])$
$\alpha_{y,x} = \frac{exp(e_{y,x})}{\sum_{x'\in\mathcal{N}_y} exp(e_{y,x'})}$
$u_y = \Big\|_{k=1}^{K} \sigma\Big(\sum_{x\in\mathcal{N}_y} \alpha_{y,x}^k W^k h_x\Big)$
Fusion
Input:
x, y
Algorithm:
$z = \sigma(W_f[x||y] + b_f)$
$fusion(x, y) = zx + (1 - z)y$
FFN
Input:
x, y
Algorithm:
$h = Wx + b$
$z = Wh + b + y$
Nodes^t
words
Word Net
Algorithm:
$U_{w\leftarrow s}^{t+1} = \text{GAT}(H_w^t, H_s^t|G_{w\leftarrow s}, A_{w\leftarrow s})$
$H_w^{t+1} = FFN(U_{w\leftarrow s}^{t+1} + H_w^t)$
sentences
Sentence Net
Algorithm:
$U_{s\leftarrow w}^{t+1} = \text{GAT}(H_s^t, H_w^t|G_{s\leftarrow w}, A_{s\leftarrow w})$
$U_{s\leftarrow s}^{t+1} = \text{GAT}(H_s^t, H_s^t|G_{s\leftarrow s}, A_{s\leftarrow s})$
$U_{s\leftarrow S}^{t+1} = \text{GAT}(H_s^t, H_S^t|G_{s\leftarrow S}, A_{s\leftarrow S})$
$U_s^{t+1} = fusion(fusion(U_{s\leftarrow w}^{t+1}, U_{s\leftarrow S}^{t+1}), U_{s\leftarrow s}^{t+1})$
$H_s^{t+1} = FFN(U_s^{t+1} + H_s^t)$
Sections
Section Net
Algorithm:
$U_{S\leftarrow s}^{t+1} = \text{GAT}(H_S^t, H_s^t|G_{S\leftarrow s}, A_{S\leftarrow s})$
$U_{S\leftarrow S}^{t+1} = \text{GAT}(H_S^t, H_S^t|G_{S\leftarrow S}, A_{S\leftarrow S})$
$U_S^{t+1} = fusion(U_{S\leftarrow s}^{t+1}, U_{S\leftarrow S}^{t+1})$
$H_S^{t+1} = FFN(U_S^{t+1} + H_S^t)$
Nodes^{t+1}

Figure 2. The *Extractive Summarization* module architecture.

D. *Extractive Summarization Module*

The *Extractive Summarization* module receives a *digest* from the previous module. After removing stop-words, it is processed as follows:

- The *word nodes* are the output of GloVe-300, having an embedding dimension equal to 300.
- The 1*st* *step for sentence nodes* is the output of the *word nodes* passed independently through 6 1D *CNN* with different filter sizes to acquire different local n-gram features.

```
{"input_size": 300;
"hidden_size": 50;
"filter_size": [2, 3, 4,
                5, 6, 7]}
```

  At this step, the embedding dimension is $6 \times 50 = 300$.
- The 2*nd* *step for sentence nodes* is the output of the 1*st* *step for sentence nodes* passed through a BiLSTM with `hidden_size` 256. Doing this operation, we have embedding dimensions equal to $(2 \times 256 = 512,\ number\ of\ words)$.
- The 1*st* *step for section nodes* representation is done by applying the attention mechanism and a BiLSTM with `hidden_size` 256 to the sequence of sentence representations belonging to the same section obtained at the 2*nd* *step for sentence nodes*. Doing this operation, we have embedding dimensions equal to $(2 \times 256 = 512,\ number\ of\ sentences)$.
- We compute the *boundary distance embeddings* for sentences (the distance of each sentence from the beginning and the end of its section) by iterating k from 1 to $(\texttt{output\_dimension}//2 + 1)$.

Given

$$f(k) = 10000^{\left(2\times\frac{k}{\text{output\_dim}}\right)},$$

we calculate:

$$\sin\left(\frac{\text{sentence\_number\_from\_begin}}{f(k)}\right),$$

$$\cos\left(\frac{\text{sentence\_number\_from\_begin}}{f(k)}\right),$$

$$\sin\left(\frac{\text{sentence\_number\_from\_end}}{f(k)}\right),$$

$$\cos\left(\frac{\text{sentence\_number\_from\_end}}{f(k)}\right).$$

We used 64 as the `output_dimension`. After having computed these values, we stack them together and obtain the boundary distance embeddings with embedding dimensions equal to $4 \times 32 = 128$.

- The 3rd *step for sentence nodes* are the concatenation of each element in the output of *the* 2nd *step for sentence nodes* with the *boundary distance embeddings*, obtaining $(512 + 128 = 640,\ number\ of\ words)$ as embedding dimensions.
- Finally, to obtain the *sentence nodes* and the *section nodes*, since it is not discussed what the *type-specific transformation matrix* mentioned in [1] is, we apply a type-specific Attention layer to the result of the 3rd *step for sentence nodes* and the 1st *step for section nodes*, respectively. This operation removes the last dimension of the embeddings. Hence, the former will be of dimension 640 and the latter 512.

After this processing, we use these nodes to construct heterogeneous sub-graphs upon which we have applied the *GAT* module (available as *GATConv* from the *PyTorch geometric* library) and different *fusion* layers to merge the output of each sub-graph. At the end of this process, there is a *Sentence Extractor* module from which we obtain the final prediction, i.e. which sentences will compose the final summary (limited to 200 words maximum).

The sub-graphs are the following:

```
{"Sentence nodes as target":
    {Word nodes as the source:
        {
        "input_size": {300};
        "hidden_size": {640}
        }
    };
    {Sentence nodes as the source:
        {
        "input_size": {640};
        "hidden_size": {640}
        }
    };
    {Section nodes as the source:
        {
        "input_size": {512};
        "hidden_size": {640}
        }
    };
"Word nodes as target":
    {Sentence nodes as the source:
        {
        "input_size": {640};
        "hidden_size": {300}
        }
    };
"Section nodes as target":
    {Word nodes as the source:
        {
        "input_size": {300};
        "hidden_size": {512}
        }
    };
    {Sentence nodes as the source:
        {
        "input_size": {640};
        "hidden_size": {512}
        }
    };
}
```

The *GAT* network with word nodes as the target has 6 head attention, while the others have 8 head attention. Therefore, the output dimension will be 6 and 8 times the one of the `hidden_size` respectively. The output of each attention head is stacked together along the embedding dimension.

To make sub-graphs with the same target "communicate" with one another, we have to process their outputs. To this end, we implemented the *fusion* layer as in our reference paper:

$$z = \sigma(W_{\text{f}}[x||y] + b_f)$$
$$fusion(x, y) = zx + (1 - z)y$$

Here σ is the sigmoid activation function and $||$ is the stacking operation along the embedding dimension. After having "fused" these outputs together, there is a feed-forward network (FFN). This layer is a small residual neural network: we pass the output of the *fusion* layer through a new linear layer and we add, to this intermediate result, the original target *node* embeddings of that sub-graph.

In the following: H_x is the node representation, where x can be w (words), s (sentences) or S (sections); $G_{y\leftarrow x}$ is the sub-graph and $A_{y\leftarrow x}$ is the adjacency matrix. Here we describe the *update steps*:

- Sentence as target:

$$U^{t+1}_{s\leftarrow w} = \text{GAT}(H^t_s, H^t_w | G_{s\leftarrow w}, A_{s\leftarrow w})$$
$$U^{t+1}_{s\leftarrow s} = \text{GAT}(H^t_s, H^t_s | G_{s\leftarrow s}, A_{s\leftarrow s})$$
$$U^{t+1}_{s\leftarrow S} = \text{GAT}(H^t_s, H^t_S | G_{s\leftarrow S}, A_{s\leftarrow S})$$
$$U^{t+1}_s = fusion(fusion(U^{t+1}_{s\leftarrow w}, U^{t+1}_{s\leftarrow S}), U^{t+1}_{s\leftarrow s})$$
$$H^{t+1}_s = FFN(U^{t+1}_s + H^t_s)$$

- Word as target:

$$U^{t+1}_{w\leftarrow s} = \text{GAT}(H^t_w, H^t_s | G_{w\leftarrow s}, A_{w\leftarrow s})$$

$$H_w^{t+1} = FFN(U_{w\leftarrow s}^{t+1} + H_w^t)$$

- Section as target:

$$U_{S\leftarrow s}^{t+1} = \text{GAT}(H_S^t, H_s^t | G_{S\leftarrow s}, A_{S\leftarrow s})$$

$$U_{S\leftarrow S}^{t+1} = \text{GAT}(H_S^t, H_S^t | G_{S\leftarrow S}, A_{S\leftarrow S})$$

$$U_S^{t+1} = fusion(U_{S\leftarrow s}^{t+1}, U_{S\leftarrow S}^{t+1})$$

$$H_S^{t+1} = FFN(U_S^{t+1} + H_S^t)$$

We describe two possible approaches:

Paper approach With the *Recursive update*, we can leave the architecture of the network as we have described so far and iterate over the same layers of each block of the *update steps*. We iterate twice in this case.

Extension We propose a different approach: *Non-recursive Extractive Summarization* module. Instead of iterating over the same layers, we initialize new ones for each block of the *update steps*. By choosing this method, we do not bind the network in the making of the embeddings by forcing it to repeat the same transformations at each *update step*. We have chosen 2 as depth in order to be able to compare the results with the other approach.

After this step, we pass H_s^T, where T is the last iteration, to the final classification layer with a sigmoid activation function to compute the final score (*extraction propensity*) for each sentence.

1) Training: We trained the *Extractive Summarization* module for 10 epochs with batch size 8 (we create batches by passing as a single graph 8 independent graphs representing 8 different articles). To train it, for each sentence in the gold summary we label as positive the two sentences in the article having the highest averaged ROUGE-1/2/L-sum F1, and the others as negative (0 and 1 respectively), using binary cross entropy loss.

IV. Experiments and Results

To perform the analysis we relied on the PyTorch and PyTorch Geometric libraries. The most computationally intensive operations were those involving graphs, which require intensive GPU usage: enough memory (depending on the batch size) and for a considerable amount of time. In fact, on our hardware (NVIDIA RTX 3060), each epoch on the *Extractive Summarization* module took around 8 hours (more than 80 hours in total for each attempt).

Between our results and those of the reference paper there is a slight difference. In particular, our replica of the paper obtained Rouge-L-sum: 41.41%, our main extension 41.43%, while the reference paper [1] obtained Rouge-L-sum: 43.33%. As the final score has an intrinsic stochastic component, we ran the algorithm twice. The two results we obtained are consistent with one another. Hence, we deem that the difference with respect to the paper [1] is due to a different implementation of the aforementioned *type-specific transformation matrix*.

Final scores			
	ROUGE F1 (%)		
	1	2	L-Sum
Official	45.68	**20.51**	41.41
Non-recursive	**45.69**	20.49	**41.43**
Official II	44.08	19.34	40.13
Non-recursive II	43.43	18.72	39.49
CR 10 NR	43.47	18.66	39.50
CR 10 Sched NR	43.29	18.53	39.35
Paper results	**48.14**	**21.82**	**43.33**

Table I
Rouge-1/2/L scores for our algorithms and the reference paper [1].

In "Tab. I" and "Fig. 4", we can see the Rouge-1/2/L scores for the reference paper and our models:

- the two reproduction of the paper [Official];
- the two attempts of a *Non-recursive* extension [Non-recursive];
- the attempt of training the *Content Ranker* module for 5 more epochs (10 in total) followed by the *Non-recursive* model [CR 10 NR];
- the attempt of training the *Content Ranker* module for 10 epochs with a scheduler followed by the *Non-recursive* model [CR 10 Sched NR].

Among our models, the one that performed best was our extension for the Rouge-1 and Rouge-L but not for the Rouge-2, where the "official" version is better. However this occurred with a very small margin, so much so that in the second attempt the results are reversed.

V. Conclusions

We can conclude that this is a performing approach to summarizing long-form documents

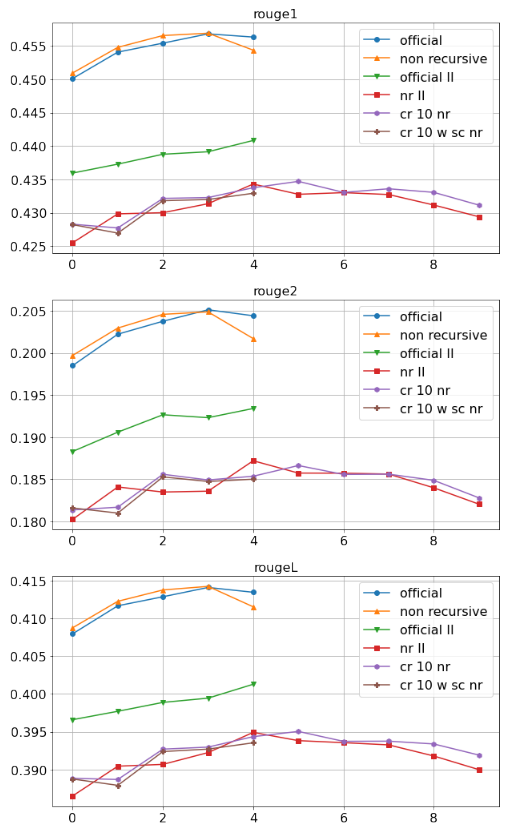

Figure 3. Trend of the Rouge-1/2/L scores for our algorithms.

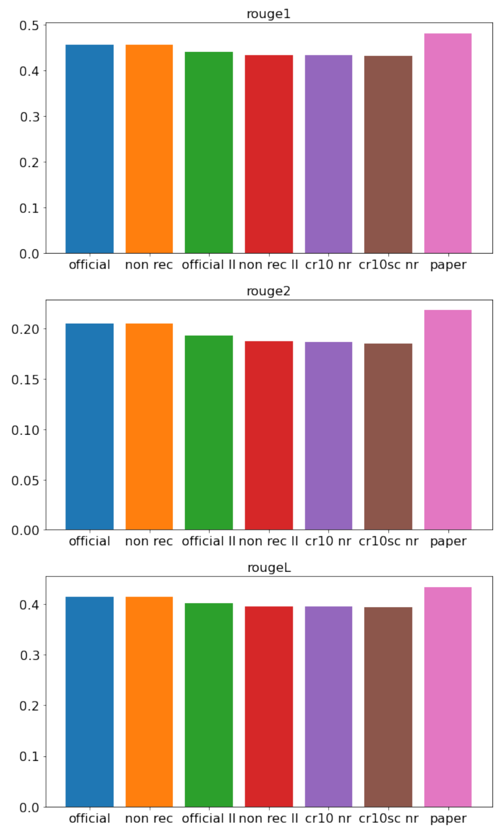

Figure 4. Rouge-1/2/L scores for our algorithms and the reference paper [1].

but that it requires not readily available hardware. The *Content Ranking* module is very useful because it ranks and selects a certain number of sections and sentences within them to create a *digest*. The *Extractive Summarization* module builds heterogeneous graphs of different discourse level. It performs well on highlighting the most relevant sentences from a *digest*. The performance of our reproduction of the model is in line with what obtained in the paper [1]. Although our extension removes the constraints imposed by the iteration over the same layers by making it non-recursive, there is no significant difference in the results.

References

[1] T. Zhu, W. Hua, J. Qu and X. Zhou, “Summarizing Long-Form Document with Rich Discourse Information,” CIKM ́21, 2021.

[2] D. Wang, P. Liu, Y. Zheng, X. Qiu, and X. Huang, “Heterogeneous Graph Neural Networks for Extractive Document Summarization,” Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 6209-–6219.

[3] R. Jia, Y. Cao, H. Tang and F. Fang, C. Cao, and S. Wang, “Neural Extractive Summarization with Hierarchical Attentive Heterogeneous Graph Network,” Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 3622-–3631.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, “Attention Is All You Need,” NIPS’17: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 6000-–6010.

[5] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò and Y. Bengio, “Graph Attention Networks,” 6th International Conference on Learning Representations, 2017.

[6] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, vol. 9.8, pp. 1735–1780, 1997.

[7] J. Pennington, R. Socher and C. Manning, “GloVe: Global Vectors for Word Representation,” Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1532–1543, 1997.