Boosting Query-based Summarization Performance by Exploiting Query Relations

https://github.com/LeonDong1993/Query-Based-Summarization-with-History

Hailiang Dong* Yangxiao Lu Priyanshi Shah Adit Shah
The University of Texas at Dallas
{hxd180000, yxl180116, pds200000, axs190336}@utdallas.edu

1 Introduction

Text summarization is the process of condensing large volumes of text into shorter, more manageable summaries, while still retaining the most important information and main ideas. Unlike standard text summarization that provide a general overview of the text, query-based summarization aims to generate a summary that is tailored to a particular query. The goal is to identify the most relevant sentences or phrases that answer the query, and then use them to generate a summary that provides a focused and concise representation of the information that the user is looking for. Meeting summarization can be treated as a special case of query-based summarization, where each meeting can have multiple queries (questions) associated with it. In this project, our goal is to boosting the performance for a meeting summarization task [4] by exploring the relations between the current query and previous query histories.

In standard query-based summarization, the input consists of a query Q and the source text T. The goal is to train a model that can extract important information from T that is related to Q and then summarize these information as the output. We propose a new method for query-based summarization by not only feeding the model with input Q and T, but also a highly related query Qr and its corresponding summary Qs. One intuition behind this idea is that the additional input Qr and Qs can be treated as a few-shot example in this case and therefore benefit the model from generating summary/answer for query Q. In addition, just like working on a hard math problem that have multiple sub questions, knowing the answer for some of the sub questions can greatly help you working on the others.

*Corresponding Author

To verify the effectiveness of our approach, we conducted several experiments on a meeting summarization task using QMSum [4] dataset. Specifically, we trained two different BART [2] models where one of them only uses Q and T as input and another one is fed with extra input Qr and Qs. We observed that, compared with the performance reported by the authors from original QMSum paper [4], the BART model we trained achieve significant better results with about 7 rouge score increase on average. In addition, the BART model with extra input Qr and Qs can achieve similar performance to our baseline BART model when the Qs is generate by the model itself on the fly (low quality query history case). If the model is fed with golden Qs for Qr, we can achieve about 1 rouge score improvement on test set, which demonstrate the effectiveness of our approach when high quality query history are available.

2 Task

In this project, we aim to boost the performance of meeting summarization by making use of query histories. Specifically, our input consists of multiple meetings where each meeting is associated with a set of queries, and objective is to generate the summary/answer for these queries of all meetings. For each query Q, we will use the notation T to represent the corresponding source text associated with it. In addition, we will denote the most relevant query to Q as Qr, and summary/answer for Qr as Qs. Our task can be formulated as a standard sequence to sequence problem, where the all of the Q, T, Qr, Qs are combined and encoded together as one long text input using separation token. We will use BART architecture to build our model and it is trained to generate the tar-

Datasets	# Meetings	# Turns	# Len. of Meet.	# Len. of Sum.	# Speakers	# Queries	# Pairs
Product	137	535.6	6007.7	70.5	4.0	7.2	690 / 145 / 151
Academic	59	819.0	13317.3	53.7	6.3	6.3	259 / 54 / 56
Committee	36	207.7	13761.9	80.5	34.1	12.6	308 / 73 / 72
All	232	556.8	9069.8	69.6	9.2	7.8	1,257 / 272 / 279

Figure 1: Statistics of the QMSum meeting datasets.

get summary/answer which are already included in the training dataset.

3 Data

We will use the QMSum dataset [4] for training and evaluating our models. QMSum is a new human-annotated benchmark for query-based multi-domain meeting summarization task, which consists of 1,808 query-summary pairs over 232 meetings in three domains as shown in Figure. 1 (the last column show the training/validation/testing instances in the dataset).

Each meeting in the dataset is associated with a set of queries of two types: general query and specific query. For general queries (e.g. summarize the whole meeting), the input source text is assumed to be the whole meeting transcript. As for specific queries (e.g. what is the option of the manager towards the plan), the source is usually only part of the meetings and the golden span is provided as set of intervals. For example, $\{[1,3],[7,10]\}$ means that only transcripts in turns 1 to 3 and 7 to 10 is related to the current query. In our following work, we will assume that the golden span is always given. Otherwise, a locator model can be use to retrieve relavant part of text from the entire source [4].

4 Methodology

4.1 Existing Approach

In query-based summarization, the input consists of a query Q and the source text T, and the objective is to generate a summary that addresses the query while ignoring other irrelevant information in the main text. To solve this task, a typical way is to feed a model with both the query and main text as input, separated by special tokens. In other words, given the query Q and the main text T, the input format is:

$$<s> Q </s> T </s>$$

The model is trained to generate a summary that addresses the query by leveraging the attention mechanism.

4.2 Proposed Method

In this project, we try to improve the performance of query-based summarization by leveraging the relationships between the current query Q and query histories. Our approach involves identifying a most relevant query Q_r and its corresponding summary S_r of Q from the history of previously summarized queries within the same meeting. We then feed the model with input in the following format[1]:

$$<s> Q_r </s> S_r </s> Q </s> T </s>$$

Here, we place the current query in the middle, which is closer to both the main text and the relevant query and its summary. We have found that this format is more efficient and enables model to generate a better summary compared with putting Qr and Qs at the end.

4.3 Query Relations

To find the most relevant query Qr for Q, we need to develop algorithms that evaluate the relevance between two arbitray queries Q_i and Q_j. In this work, we make use of modern deep learning NLP models to directly compare the semantic meaning of two query texts. Specially, we will use BERT [1] model to encode the queries Q_i into a embedding vector V_i where V_i represents the semantic meaning of the query. After that, we compute the cosine similarities between Q_i and Q_j and use the one achieve maximum similarly as the relavant query for Q_i.

[1]In our experiment, we removed the separation token between Qr and Qs, but the performance is similar to this format.

4.4 Ordering of Queries

During training time, we can simply use above method to find the most relevant query for all queries in the training set because we have the golden summary/answer for each queries. However, during testing/validation time, as the golden summary/answer is not available, we cannot generate output for a query Q unless its relevant query Qr has already been processed. In other words, we have to determine the evaluation order for all queries in the test set, and we also need to guarantee that no query can be evaluated before its related query is evaluated.

To solve above problem we have developed an heuristic algorithm that computes the best evaluation order as well as the related queries during the testing/validation time. The algorithm works as follows (we will give a code analysis for this algorithm in the next section):

1. We first compute the relevance between any pair of query and form a relevance matrix S using the method described in the previous section.

2. We then compute the sums of columns of matrix S and form a vector W, now W_i means the total relevance the Q_i to all other queries. If W_i is large, it means other queries are highly related to Q_i. And therefore, evaluate Q_i first will benefit the later queries.

3. We order the queries based on there weight W_i, queries with large weight will be evaluated first.

4. At last, since the order are determined, to determine the related query for query Q, we first find which queries are evaluated before Q and use the one with maximum relevance to Q as its related query.

4.5 Training and Inference

We used BART [2] as our model architecture and modified the positional embedding layers to support up to 2048 tokens. Note that the weight of newly added position (from 1024 to 2047, 0 indexed) is initialized from existing position (0 to 1023). During training time, teacher forcing is used to trained the model and the loss is evaluated by cross entropy function of model's output (a distribution of next token) and target distribution (usually a 1-hot distribution applied with label smoothing) in every generation step. During testing/inference time, we use standard beam search with beam size 4. In addition, we limit our maximum generation length to 150. Note that we have to do summarization one by one due to the evaluation order issue at testing/validation time.

5 Implementations

In this project, we have used the following key libraries and packages:

1. json: used for loading the dataset (the QMSum is provided in json format)

2. pytorch: for deep learning training and evaluation

3. nltk: to prepocess and tokenized the input

4. transformer, datasets and evaluate: the hugging face libraries for training and evaluate the BART model.

5. numpy and pandas: for general dataset manipulation

Our implementation contains several important algorithm and functions. This including (1) the algorithm used to solve the ordering problem describe before in section 4.4; (2) The evaluation function for test/validation set that generate the summaries and answer one by one and dynamically form the input of future queries on the fly; (3) the code piece for modifying the BART architecture to support 2048 tokens[2]. Here, I will give detailed analysis of the algorithm used to solve the ordering problem. The detailed implementation is shown in Listing 1.

In this function, it has three inputs: (1) a list of BERT embedding vectors of queries `query feature`; (2) a similarity function (cosine similarity in our case) `sim func`; and (3) a flag indicate if we are working on the training set `train`. we first use two nested loops to compute the relevance (similarity) between any two pair of queries. After that, we compute the weight of each query by conducting a column sum. As larger weight should be evaluated first, we put a negative sign in the weight and use argsort function provided in numpy to compute the evaluation order.

Once the evaluation order is determined, we compute the related queries. We defined a list called related where $related[i] = j$ would means that Q_j is the related query for query Q_i. We fill this array based on our evaluation order, for the

[2]More details can be found in our github repo

```
def get_related_query(query_feature, sim_func, train = False):
    n = len(query_feature)
    similarity = np.zeros(shape = (n,n))
    for i in range(n):
        for j in range(i+1, n):
            v = sim_func(query_feature[i], query_feature[j])
            similarity[i,j] = v
            similarity[j,i] = similarity[i,j]

    weights = -np.sum(similarity, axis = 0)
    order =  np.argsort(weights)
    related = [0] * n

    for i, k in enumerate(order):
        if not train and i == 0:
            related[k] = None
            continue

        if train:
            selected = order
        else:
            selected = order[0:i]

        j = np.argmax(similarity[k][selected])

        if k == selected[j]:
            related[k] = None
        else:
            related[k] = selected[j]
    return order, related
```

Listing 1: The algorithm for computing evaluation order and related query in test/validation time.

i^{th} item in order, we only consider the previous query from $order[0:i]$, and therefore guarantees there is no future reference.

6 Experiments and Results

6.1 Data Pre-processing

We pre-process the dataset by remove all noisy text generated by speech recognition (as the one did by the original authors). After that we use the tokenizer from nltk to clean the text and form the source text, query text, target text, related query and related summary/answer (for training only) and orgranize them into hugging face dataset format.

6.2 Evaluation

To evaluate the performance of the model on the test dataset from QMSum [4], we employ the ROUGE score metric.ROUGE score is a collection of metrics utilized to measure the quality of generated summaries via comparison with their respective reference summaries. In this work, we will only focus on the ROUGE-1, ROUGE-2 and ROUGE-L F1 performance.

6.3 Training Details

Although Zhong et al. have the results of BART Model on QMSum dataset ,to ensure fairness in our comparison of model performance, we adopt the same experimental setup to fine-tune two BART models with and without query history.

Table 1: The initial fine-tuning results of BART [2] on QMSum dataset [4]. ROUGE F-1 score is employed to evaluate models. During inference, the related queries and their predicted summaries are included as an input component with the present query.

Models	R-1	R-2	R-L
BART (QMSum [4])	32.18	8.48	28.56
BART (Ours)	**40.20**	**14.60**	**35.74**
BART + query history (Ours)	39.34	13.56	34.58

Table 2: The updated fine-tuning results of BART [2] on QMSum dataset [4]. We use ROUGE F-1 score to assess the performance of models. During inference, the related queries and their predicted or ground truth summaries are fed into the model with the current query as input.

Models	R-1	R-2	R-L
BART (QMSum)	32.18	8.48	28.56
BART (Ours)	39.41	14.43	34.30
BART + query history (Ours)	39.24	14.73	34.24
BART + ground truth query history (Ours)	**40.18**	**15.30**	**35.44**

We set the maximum token length to 2048 as QMSum. We trained and evaluated our model using a four GPU system consists of 3x NVIDIA V100 32GB and 1x NVIDIA A100 40GB GPUs. It takes roughly 1 hour for 12 training epochs. During testing, for our configuration of 4 beams, the model can complete the generation of summarization in about 0.55 secs per instance .

6.4 Initial Results

We use the AdamW [3] optimizer with batch size 2 and learning rate 1e-5 (weight decay 1e-4) and trained BART model for 80 epochs on the QMSum dataset. The experimental results are presented in Table 1. There are serveral observations from the results.

1. The BART model we trained (2nd row) outperforms Zhong et al.'s method [4] (1st row), thereby indicating the effectiveness of our training strategy.

2. The BART model with extra related query history (3rd row) did not surpass the model without query history (2nd row).

This results are surprising to us at the beginning we have identified the possible reason in the next section.

6.5 Performance Analysis

There are several potential explanations that why the model with extra query history is performed worse than the baseline model. First, the general queries employ all meeting transcripts as input, which is likely to surpass the token limit. This issue is exacerbated in our cases as we add more content at the beginning of the input. To alleviate this problem, we remove the general queries from the dataset and lost like 15% of the data. Second, the model might be overfitting as QMSum dataset is relatively small. We fix this problem by tuning hyperparameters, especially weigh decay and the number of epoches. Lastly, during the testing phase, the model generates summaries for related queries on the fly. If the generated summary is not of high quality or deviated from the ground truth, then the inclusion of such summary may hurt the performance of future summaries. This problem cannot be fixed in practice, because we don't have ground truth summary at test time. However, in our case, we can at least verify if this is the root cause of the above poor performance.

6.6 Final Results

Based on above discussion, we redesigned the experiment. First, we do not include the general query as input during training/testing. We employ the AdamW [3] optimizer with batch size 3 and learning rate 2e-5 (weight decay 5e-3) and trained the model for 40 epochs. In addition, we also added an experiment where we use golen summaries for related query during test time. The new set of experimental results are presented in Table 2. With predicted queries and their corresponding summaries, the model yields performance comparable to that of its counterpart without query history after we removed the general query and updated our hyper-parameters. Notably, when provided with prior queries and their high-quality, ground truth summaries, the model generates superior output, highlighting the utility of query history for future queries. The performance difference between the input of predicted and ground truth summaries provides evidence for the effectiveness of query history and supports our assumption.

7 Conclusion

In this work, we have proposed a new method for query-based summarization by utilizing the related queries and its corresponding summary in query history. We conducted experiments by fine tuning BART models for the meeting summarization task on QMSum dataset. Our final experiment shows that, with low quality query history, our new method can achieve similar performance compared to baseline method that do not use query history. In addition, once we have high quality query history (e.g. the ground truth), we can further improve over the baseline by about 1 ROUGE score. This provides the evidences that our method is effective.

For future works, we can explore more on exploiting query histories. For example, we can just feed the model with related query but not its corresponding summary. Also, we can compute the related query not only from the current meeting, but also the entire training or tesing data.

8 Acknowledgement

We would like to express our sincere gratitude to Professor Du, Xinya for his guidance and support throughout this project. His expertise and insights were invaluable in helping us navigate the challenges of this research and improve the quality of this work.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[3] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[4] Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, et al. Qmsum: A new benchmark for query-based multi-domain meeting summarization. *arXiv preprint arXiv:2104.05938*, 2021.