Context Specific Text Summarization using seq2seq Model

Motivation

There is an enormous amount of textual material, and it is only growing every single day. The data are unstructured and there is a great need to reduce much of this text data to shorter, focused summaries that capture the salient details, skim the results as well as check whether the larger documents contain the information that we are looking for. Text summarization is the process of producing a briefer version of a document or a set of documents while preserving the most essential information of the original texts. In a text summarization system, a text summary is decoded by a model, which is trained on a specific textual dataset. However, text documents differ by what is here called contexts, which may encompass categories such as genres or document types. A text summarization model may be more suitable (i.e.: produce more faithful summaries) for one context but not others (Ferreira 2014). What contexts is a model suitable for depends on the data it was trained on. This project proposes a text summarization system consisting of multiple models, each trained with a different dataset. The model used by the system is determined on the basis of the input text's similarity to the training data of the models. An architecture such as this is useful in applications where input text of various kinds can be expected. This idea is demonstrated with a web app that has been developed where a user does not have to choose which text summarization model to use.

Approach

In this project, we utilize one implementation of a seq2seq model with an encoder-decoder architecture in Tensorflow for text summarization. To extend it, we added a mechanism to evaluate the similarity of an input text with different models trained using this algorithm. This allows the text to be summarized automatically by the most appropriate model, based on its context.

We also have constructed a web application to allow one user to insert a text, choose one particular pre-trained model or allow the model to be selected automatically, and generate the summary.

Text Summarization Model

The algorithm that we used as base is available in https://github.com/dongjun-Lee/text-summarization-tensorflow (https://github.com/dongjun-Lee/text-summarization-tensorflow).

This code implements a Encoder-Decoder (seq2seq) model with attention mechanism. Its main components include:

Word Embedding: uses Glove pre-trained vectors to initialize word embedding (https://nlp.stanford.edu/projects/glove/ (https://nlp.stanford.edu/projects/glove/)).

Encoder: uses LSTM cells with stack_bidirectional_dynamic_rnn (https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/stack_bidirectional_dynamic_rnn (https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/stack_bidirectional_dynamic_rnn)).

Decoder: uses LSTM BasicDecoder for training, and BeamSearchDecoder for inference (https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/BeamSearchDecoder (https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/BeamSearchDecoder)).

Attention Mechanism: uses BahdanauAttention with weight normalization(https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/BahdanauAttention (https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/BahdanauAttention)).

The diagram below demonstrates how the model is organized.

Text Similarity

To calculate the similarity of the input text with the pre-trained models, we utilize the SpaCy library (https://spacy.io (https://spacy.io)).

For each of the pre-trained models, we have a filtered version of the data used to train them. This data is used to calculate the similarity with the input data to be summarized. Then, the web application selects the model with the highest similarity score to summarize the input text.

Web Application

We have developed a Flask (http://flask.pocoo.org (http://flask.pocoo.org)) application to allow the use of the pre-trained models in a more friendly way. Basically, this application allows the user to input a text and generate its summary. The user can select different models or allow the automatic selection of the model according to the similarity of the input text with the pre-trained models.

More details about the implementation and use of the application are described in the subsequent sections.

Data

The following are the datasets used for training the models in the text summarization system:

- CNN News and DailyMail: these are datasets of news articles from CNN and Daily Mail websites (Nallapati et al. 2016). Each article is paired with a human generated summary that is derived from the summary bullets that accompany the article. This dataset contains 287,226 training pairs, 13,368 validation pairs and 11,490 test pairs. It can be obtained from https://github.com/JafferWilson/Process-Data-of-CNN-DailyMail (https://github.com/JafferWilson/Process-Data-of-CNN-DailyMail). The files of the data preprocessed for this project are in the folders /model/sumdata/CNN and /model/sumdata/DailyMail.
- Gigaword: a corpus of 9.5 million English news articles, where the headline is considered to be the article's summary (Rush, Chopra and Weston 2015). This dataset was obtained from https://github.com/harvardnlp/sent-summary (https://github.com/harvardnlp/sent-summary). The pretrained model on this datatset that is used here was obtained from https://github.com/dongjun-Lee/text-summarization-tensorflow (https://github.com/dongjun-Lee/text-summarization-tensorflow). The files of the data preprocessed for this project are in the folder /model/sumdata/Gigaword.

For data in the two aforementioned files, only the first **three** sentences from the articles and the first **two** sentences from the official summary were used. This was done so as not to exceed the capacity of the computer used for training. Additionally, it was found that irrelevant information is more likely to occur at the end of articles and summaries.

For the CNN News model and the Daily Mail model, training was conducted only for **4** and **9 epochs** respectively, due to computational constraints. Ideally, the models would have been trained for at least 10 epochs.

Code

As mentioned in the **Approach** section, we have used the code available in https://github.com/dongjun-Lee/text-summarization-tensorflow (https://github.com/dongjun-Lee/text-summarization-tensorflow) as a base code.

To improve the results of the summaries, we have changed the architecture of the two Neural Networks (LSTM) used in the Encoder and Decoder layers including/removing hidden layers and the number of neurons. However, the combination with the best results is the one used by the author of the algorithm.

Another customization we have done, but now resulting in better summaries, is the maximum length of the training articles (article_max_len) and summaries (summary_max_len). The original implementation only considered the first 50 characters of the articles and the first 15 characters of the summaries during the training phase. As a result, using this limits caused quality deterioration of the predicted summaries, especially considering that the data used to train the models have longer articles and longer summaries. In our final code, we set the maximum length to 500 characters for the articles and 100 characters for the summaries. The downside of this change is that the time required to train a model got significantly longer.

Besides those changes, we have also made a few modifications in the original code to allow the summarization to be called from the web application.

The code to calculate the similarity of the models with the input text and the code for the web application was developed by our group from scratch.

The files of this project are organized in the following structure:

- **/app/**
  - **app.py (http://app.py)**: Script to run the Flask web application.
  - **model_selector.py**: Calculates the similarity scores for the input text and the models and select the most similar model to be used to summarize the text.
  - **summarize.py (http://summarize.py)**: Calls the summarize_single.py script to summarize the input text inserted in the web application.
- **/model:**
  - **train_specific.py**: Train a new model.
  - **summarize_single.py**: Summarizes the input text inserted in the web application.
  - **model.py (http://model.py)** and **util.py (http://util.py)**: Support methods for training and testing.
  - **sumdata/**: Folder to store the datasets to be used to train models.
  - **saved_model/**: Folder to store the pre-trained models.
- **/text_similarity:** Contains a jupyter notebook **Similarity.ipynb** with some examples of the similarity scores calculation
- **/data_prep:** Contains scripts to clean data used to train models
- **/poster:** Contains the final PowerPoint document and the images used in the project poster

Web Application

image alt

In using the demonstration web application, the user may opt to choose a summarization model from the drop-down list. Alternatively, the user may forego choosing a model, in which case the model will be automatically selected. Next, the input text is to be entered into the text area. After the input is submitted, after a moment the output summary will be displayed. If the summarization model was not explicitly chosen, the name of the model that was automatically chosen will be displayed.

Experimental Setup

For evaluating the system, an experiment has been designed and implemented in /model/experiment.py. The experiment consists of two conditions. In condition 1, the text summarization system uses the Gigaword model exclusively. In condition 2, the system uses multiple models, that is, the design proposed in this project. The testing data come from Google's sentence compression corpus consisting of 10,000 pairs of uncompressed and

uncompressed sentences from news articles (originally obtained from https://github.com/google-research-datasets/sentence-compression (https://github.com/google-research-datasets/sentence-compression)). To access the version preprocessed for this project refer to model/sumdata/DownloadData.txt.

For scoring, the two conditions, the ROUGE scores of all the summaries in each condition are summed and averaged. ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It is a set of metrics for evaluating automatic summarization of texts as well as machine translation. It works by comparing an automatically produced summary or translation against a set of reference summaries (typically human-produced). The following variants of ROUGE scores were used in our project:

ROUGE-N: Overlap of N-grams between the system and reference summaries.

- ROUGE-1 refers to the overlap of 1-gram between the hypothesis and reference summaries.
- ROUGE-2 refers to the overlap of bigrams between the hypothesis and reference summaries.

ROUGE-L: based on the longest common subsequence (LCS) between the hypothesis and reference summaries. This takes into account sentence level structure similarity naturally, and identifies the longest co-occurring in sequence of n-grams automatically.

ROUGE scores also consist of precision and recall components:

Recall in the context of ROUGE refers to how much of the reference summary occurs in the hypothesis summary:

$$\text{Recall} = \frac{number\_of\_overlapping\_words}{total\_words\_in\_reference\_summary}$$

Precision is how much of the hypothesis summary is relevant or needed:

$$\text{Precision} = \frac{number\_of\_overlapping\_words}{total\_words\_in\_system\_summary}$$

For this experiment, two variants of ROUGE scores (ROUGE-N and ROUGE-L) are recorded to get a detailed observation of the results.

The reason for using the Google dataset is that it is independent of the models in the system, thus avoiding a bias for any model to be selected. Although current text summarization research usually uses a dataset such as the Gigaword dataset, this deviation is not necessarily a flaw of the experimental design. To demonstrate the effect of the proposed system design it is sufficient to show that it leads to a performance difference over a control condition. Furthermore, because the models could not be sufficiently trained, due to processing constraints, the system cannot yet be fairly compared to the state-of-the-art for now. Thus, the control condition experiment design was chosen.

However, due to unresolved problems involving a library (TensorFlow), which prevented *iterative* production of summaries, the experiment script could not be ran during the duration of this project. In lieu of this, in the **Result** section details about specific summarizations are given.

Challenges

The main factor that limited the results that could be obtained was computational resources. The models that are in the system were not trained for an ideal number of epochs (see Data section). This is due to the neural network-based learning model used in this project, and it is known that neural network training requires considerable computing power (Edwards 2015). The best resource available for this project was one of the team member's access to a computer with 64GB of RAM, but with no graphics processing unit (GPU). Nevertheless, training the models still consumed a considerable amount of time (6 days).

Another impediment faced by this project was the availability of diverse types datasets. For demonstrating the concept of the project, it is ideal that the system could accommodate a wider number of contexts. Datasets of other textual genres could be found, such as a dataset for summaries of product reviews and a dataset of abstracts of academic articles. However, these datasets often had data points only in the hundreds, which are insufficient to train a neural network. Training by such amount of data would lead to poor decoding performance of the model. This scarcity is understandable because creating a dataset annotated with human-generated summaries of texts requires considerable time and resources. Furthermore, training a greater number of models would take computational resources and time that is beyond the scope of this project.

Results

The table below presents two examples comparing the predictions of the summaries by the models we have trained.

image alt

Analysis of the Results

Overall, we considered the results satisfactory considering the challenges we had in preparing the models.

In some cases, we consider that the predicted summary had a higher quality (from a human perspective) than the official summary. As an example, consider the summary generated by the Gigaword model for the first article in the table above (**Results** section).

Another interesting aspect of the results we achieved is that the models sometimes substitute one word from the original article by a different word, but representing the same type of entity. For instance, we had some examples where the article describes some event with some soccer player or team name and the predicted summary was generated using a different soccer player name or team. In some cases, it can generate some inconsistencies with the real world. The second example of the table above shows two cases where inconsistency occurs. The first is in the output of the CNN News model, which states that it was the team coach who scored two goals. Next, the output of the Daily Mail model describes local-level soccer teams PSG and Getafe playing a match in the World Cup.

Besides that, our final evaluation is that the quality of the predictions would be significantly higher if we had the resources to train the models with more iterations. However, the main target of this project was to develop a mechanism to automatically select the most appropriate model to generate the summary of the input data and this aspect of the project was achieved.

Future Work

One way of extending the proposal being demonstrated in this project is instead of having models that vary by contexts, the models would vary by summary formats. Not only do documents differ by types, but so are summaries. Some of the different genres of summaries are headlines, synopsis, abstracts, and TL;DR notes (“Too long; did not read”) on internet comments. Having models by summary formats would allow the user to customize the summary to publication domains.

Another proposal is to have the system to choose among output summaries of multiple models. This is different from the current design in which the model is chosen based on the input text, and then that model is used to produce the summary. Instead, upon receiving an input text, multiple models would be used to create multiple summaries. Then the system will choose the summary of the highest quality (possibly based on a measure involving the summary and the original). In effect, this revised design suspends the assumption that it is the model trained on texts most similar to the input text that will produce the best summary. However, decoding multiple summaries would take an inordinate amount of time for an

application. Instead, a more sensible design revision is to choose only a subset of the models to produce candidate summaries. What models are in the subset can be determined on the basis of text similarity as in the current design.

A final proposal for revising the current design is to have automatic learning of different summary types. Currently, the categories of summaries are predetermined, that is the developer determines that a particular dataset represents a particular summary genre, and then a model is trained exclusively on that dataset. Instead, it is worthwhile to have a model that is fed with a heterogeneous set of textual data, and then the model would discover what categories of summaries are there. Then individual models would be trained based on those discovered categories. Human-predetermined summary categories may be arbitrary, and in fact, there may be finer categories that are not humanly discoverable.

Running the project files

In order to run the project files, install dependencies of the project by entering the following in a command line:

```
$ pip3 install –r requirements.txt
```

Downloading training files and pre-trained models

Next, download the pre-trained models and the data used to train the models by running the cells below. However, to run the web application it is only necessary to download the pre-trained models.

(this is already described in the project.ipynb file)

Running the web app

To run the demonstration web app, enter the following in a command line:

```
$ python3 /app/app.py
```

After a moment, the user may be prompted to allow Python to accept network connections on the computer, in which case do allow it to. Then open a web browser, and enter the URL http://0.0.0.0:5000/form (http://0.0.0.0:5000/form). If this does not work, try accessing using http://localhost:5000/form (http://localhost:5000/form). The web app will open in the browser.

Training a new model with new data

In order to train a new model with new data, it is necessary to modify the file /model/train_specific.py in the lines 41, 42 and 43 update the following variables:

```
41. model_name = <name of the new model>
42. train_article_path = base_path+'path after /model/ to the .txt file with the
43. train_title_path = base_path+'path after /model/ to the .txt file with the s
```

And run:

```
$ python3 /model/train_specific.py
```

Warning: for large training datasets this process may take days to run if a GPU is not used.

References

- Edwards, Chris. 2015. Growing pains for deep learning. *Communications of the ACM* 58(7), pages 14-16.

- Ferreira, Rafael, Fred Freitas, Luciano Cabral, Rafael Lins, Rinaldo Lima, Gabriel Franca, Steven Simske and Luciano Favaro. 2014. A Context Based Text Summarization System. Proceedings - 11th IAPR International Workshop on Document Analysis Systems,
- Nallapati, Ramesh, Bowen Zhou, Caglar Gulcehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond, *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, pages 280–290.
- Rush, Alexander M., Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379-389.