Abstractive Text Summarization With Sentence Ranking

W266 Fall 2018 Project Report

Alan Tan, Cynthia Hu, Grace Lin

{alantan, cynthiahu, ylin}@berkeley.edu

Abstract

Neural network based abstractive method for text summarization is popular and effective on the CNN/Daily Mail dataset. We combine extractive and abstractive methods to derive summary for the news articles in this report. We apply the neural sequence-to-sequence model with attention on New York Times corpus after selecting salient sentences. The graph-based sentence ranking algorithm itself achieves high ROUGE score. However, applying the basic sequence-to-sequence model on top of sentence selecting does not improve the score.

1. Introduction

Document summarization is an important and challenging task in natural language processing. It helps people to extract important information from long document quickly and multiple-document summarization is useful in queries.

There are two approaches in summarization. Extractive approach focuses on extracting words or sentences from original document directly by using sentence selection or other compression methods. On the other hand, abstractive approach can rewrite the sentences using new sentences, phrases, or words not just select from existing sentences. The latter method is similar to how human does summarization. Recent work (Rush et al., 2015; Das et al., 2007; Chen and Bansal, 2018; See et al., 2017) has shown great success of applying data-driven abstractive neural model to text summarization.

This project aims to generate abstract from news article[1]. We used New York Times corpus instead of the commonly used preprocessed CNN/daily Mail data in the recent papers, hoping to gain some insights into the level of generalizations of recent works. Inspired by the above work of combining extractive and abstractive methods, we implemented text ranking based model to select salient sentences and then apply a sequence-to-sequence neural network model with attention to rewrite selected sentences to generate summary.

The sentence ranking model achieved slightly lower ROUGE scores than the baseline which is the lead paragraph. However, the abstractive model didn't generate good output. We have seen the common issues of sequence-to-sequence model, such as generating nonsense words or repetition[2].

2. Related Work

Early summarization work focuses on extractive models and linguistic models, such as sentence deletion and compression. (Kupiec et al., 1995; Knight and Marcu, 2002; and many others).

Mihalcea and Tarau (2004) introduced TextRank, a graph-based ranking model for text processing, and showed how this model can be successfully used in natural language applications. They proposed unsupervised methods for keyword and sentence extraction. Our research further develops these methods.

Rush et al. (2015) proposed a fully data-driven approach for abstractive **sentence** summarization. They incorporated the input text into generation by jointly training an encoder model and standard feedforward neural language model. Training on 4 million articles, their model shows great gains on the DUC-2004 shared task.

In recent years, abstractive neural models are popular in summarization which are similar to neural machine learning translation model (Sutskever et al. 2014; Bahdanau et al. 2016). However, the basic sequence-to-sequence model has some issues. For example, it generates incorrect information or repeated words. To solve these issues, See et al. (2017) used a hybrid pointer-generator network. The pointer

[1] Please see our codes here: https://github.com/CynthiaHu/W266TextSummarization
[2] Due to limited resources and capabilities, we didn't fully replicate the complex models in the recent papers but tried to test our own methodologies and MT seq2seq model.

can copy words from original document to keep accurate information and the generator (single-layer LSTM RNN model) will keep the ability to generate new words for the abstractive model. On top of that, their model used coverage to keep track of information used to discourage repetition.

Chen and Bansal (2018) achieved the new state-of-the-art on document summarization based on CNN/Daily mail data. Their model uses an extractor model to select sentences and then to rewrite the selected sentences by an abstractor model. And a policy-based reinforcement learning is used to bring these two together. Multiple-layer LSTM RNN models are used.

3. The Model

Understanding the full breath of data processing including original text is important for properly studying NLP based summarization. We prefer to preprocess the text ourselves instead of using the pre-processed CNN/daily mail data. Further, the CNN data is commonly used in recent works of neural models and we worry about that the whole industry is overfitting the models using the same dataset.

3.1 Preprocessing and Tokenization

We treat analyzing and preprocessing of the original data seriously.

We compared multiple tokenization and sentence segmentation functions. Though the word_tokenize function from NLTK is good, the sentence segmentation didn't work well for our data. For example, if there is no space between two sentences, the NLTK function cannot separate them, it also could not separate sentences by ";". After evaluating our options, we created a customized sentence segmentation and tokenization function by adding several rules, such as checking whether next letter is capitalized and the previous letter is lowercase (so the U.S. is not an end of sentence).

We decided not to perform stemming because the tense in new articles should be preserved as much as we can.

Next, we studied importing pre-trained embedding and sentence embedding. While word2vec could be successfully used to embed our input, it was not able to generate cohesive output by itself. One more challenge with pre-trained embedding is that they are quite slow to embed full sentences with any kind of meaningful weight (either

TF-IDF, or attention). So we chose to train our own model with embedding that is integrated with our seq2seq model.

3.2 Baseline Model

Our process and model is scalable to larger corpus or other data sets. For this project of news article summarization, we use the lead paragraph as our baseline model which is known to be a very high baseline for news summarization.

New York Times provided LEAD_PARAGRAPH as a field in the xml file, indicating this was something they paid extra attention to as well.

3.3 Extractive Model - Sentence Ranking

The essence of graph-based ranking algorithms is to decide the importance of a vertex within a graph, based on global information recursively drawn from the entire graph. To achieve such goals, we would need to identify vertices, the relations that connect such vertices, and calculate scores when the algorithms converge. The score of a vertex is based on the number of votes that are cast for it. The score can be calculated by the following equation:

$$WS(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)} w_{jk}} WS(V_j)$$

where d is a damping factor that can be set between 0 and 1, 1-d referring to the probability of randomly jumping from a vertex to another vertex. The factor d is usually set to 0.85 (Brin and Page,1998), and this is the value we are also using in our implementation.

The steps to use this model include performing statistical parsing and tagging, collecting and normalizing the key phrases from a parsed document, calculating a significance weight for each sentence, using MinHash to approximate a Jaccard distance from key phrases determined by TextRank, and finally summarizing a document based on most significant sentences and key phrases.

We continue to iterate through different edits. When we implemented the model for the first time, we noticed lots of articles starting with financial reports rather than text sentences. This model was able to generate numbers but did not make logical senses to us. We eventually excluded such observations which would be discussed

further in the Model section. It is important to note that TextRank is an unsupervised model which does not require training or tuning.

3.4 Abstractive Model - Sequence-to-Sequence

Our abstractive model is based on re-write of extractive summary. The idea is an abstractive summary usually have its own written style, word choice and idea organization, that could make abstract more informative or easier to consume. We will use a seq2seq model that is popular among translation models.

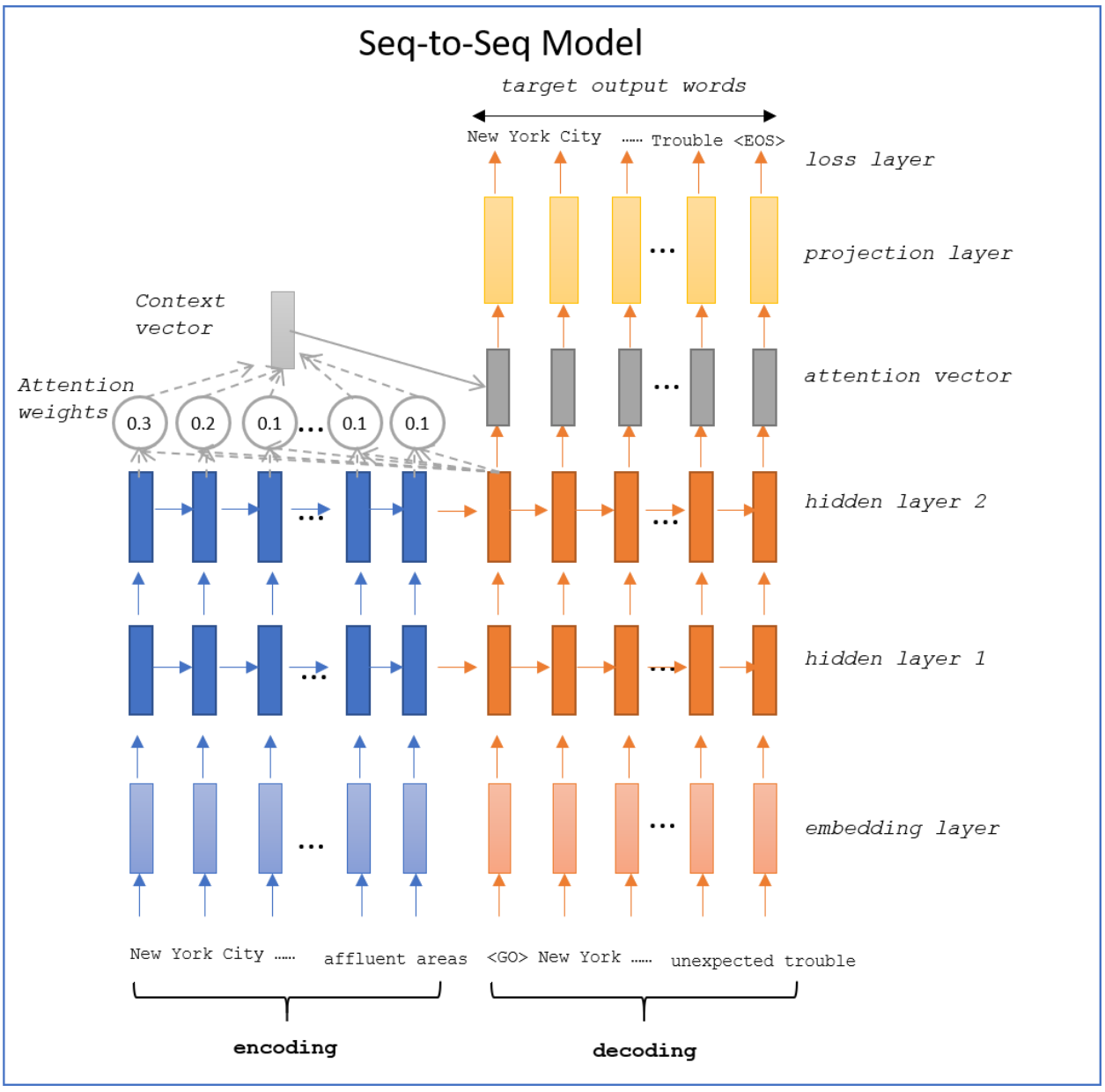

Figure 1: Sequence-to-sequence model with attention

The model includes encoder and decoder parts (see Figure 1). Similar to machine translation, the encoder reads in all sentences from one example and converts the information into one vector for the decoder to absorb. The decoder also take into consideration of previous target output words (abstract words that already chosen), shifting to the left by one time step with an end-of sentence tag appended on the right. We use the reverse source sequence as many recent study popularized. We also added <GO> tag at the beginning of target output to indicate the mark between encoding source and decoding target.

During training, we use the actual target output (manual written abstract) as decoder target; during inference step, we use the RNN output from previous time step as the input for the current input.

Embedding layer

The input words were converted into integers as part of the preprocessing and then convert to H-dimensional (H=300) vector using word embedding model in both encoding and decoding steps. Embedding model is trained on the data set. Because summarization rewrites in the same language, we actually used embedding_lookup during decoding, because there is no need to train two languages. The same model parameters are shared between decoder training step and inference step.

Hidden layer

Multiple-layer LSTM models are used in both encoder and decoder parts and they have the same number of layers. Final state from encoder will be initial state for the decoder RNN cells.

Attention Distribution

We used attention to weigh the words in a sentence. The score function is used to compare the target hidden state with the source hidden state and then the result is normalized to calculate the attention weights.

We used Bahdanau Attention as attention mechanism and wrapped it around the decoding cells. The context vector is the weighted average of the source states. Combining the context vector with the current target hidden state, we will get the attention vector which will be used to derive the softmax logits and loss. The attention vector is fed to the next time step (not shown in the figure).

Attention weights:

$$\alpha_{ts} = \frac{exp(score(h_t, \bar{h}_s)}{\sum_{s'} exp(score(h_t, \bar{h}_{s'})}$$

Context vector:

$$C_t = \sum_s (\alpha_{ts} \bar{h}_s)$$

Attention vector:

$$a_t = f(c_t, h_t) = tanh(W_c[c_t; h_t])$$

Score function:

$$score(h_t, \bar{h}_s) = v_a^T tanh(W_1 h_t + W_2 \bar{h}_s)$$

Projection layer

This output layer turns the top hidden states to logit vectors of dimension V (vocabulary).

Loss

We used weighted softmax cross entropy loss function. We padded target output to the max length of the target sentences in the same batch but we don't want to include padding token in the loss calculation. Thus we use zero weight for those padded positions.

Training/Optimization

Our training goal is to minimize the cost which is calculated as difference between predicted_logit and actual target. To solve the vanishing/exploding gradient issue in recurrent neural networks, especially in our paragraph-to-paragraph model, we use gradient clipping technique during optimization process. For this project, thresholds -1 and 1 are used.

4. Experimental Setup

4.1 Data Set

We used New York Times corpus. It contains over 1.8 million articles published between January 1, 1987 and June 19, 2007 among which over 650,000 articles with abstract written (mainly between 1997 and 2007). Only articles with abstract are used in this project. We split the dataset into training and validation (1 batch in size) data sets and report results on both training and validation data set.

The original text are provided in .xml format, each article resides in its own .xml file and are organized into year/mo/da folders, 1,855,720 in total. However, not all files contain all fields. There are 653,038 files with <abstract> field, which indicate they had human created abstraction, which we will use as our target.

We extracted TITLE, ABSTRACT, LEAD_PARAGRAPH, and FULL_TEXT 4 fields and created a single file with one article on each line, and used year/mo/da/filename.xml as the index.

The decision to take data preparation in our own hands turned out to be paying back. After we experimented with RNN training performance, we found the wide range of length of the sentences in the training set have a profound implication to the training time. The longest LEAD_PARAGRAPH was 16327 words, and longest ABSTRACT was 985 words. To make the training manageable, we decided to only use news that have ABSTRACT_LENGTH <= 100 words and LEAD_PARAGRAPH_LENGTH <= 200 words. This reduced number of records from 653038 to 580454 (retained 88.88% of the records).

The very few very long abstracts and Lead_paragraph caused extremely slow training performance (more than 10 times slower) because of the excessive length of the sequence. And because these long abstracts are rare, they do not provide much value to the training as very few of the sample actually have such long content. The majority of the impact is it made all other smaller sentences being padded to extremely long equal-length sentences and slowed down training.

4.2 Evaluation metrics

Evaluation is based on recall oriented ROUGE metric: ROUGE-1 (unigrams), ROUGE-2 (bigrams), and ROUGE-L (sentences). ROUGE-N refers to the overlap of n-gram between the predicted summary and the actual abstract. ROUGE metric is calculated for each article on the test data set and then it is averaged to get the final score.

We know most of the time abstract model might not have a higher ROUGE-2 score because of the rewrite. So we also perform some manual comparisons.

4.3 Implementation

Below end-to-end process was implemented on a 16 core i7 MacBook Pro with 32GB RAM running latest MacOS and Python 3, Tensorflow 1.18.

We downloaded the 3 GB zipped dataset and wrote a bash script to extract required fields into a consolidated csv file which took about 7 hours with shell parallelism.

We then loaded csv file into a Pandas dataframe, and used the ABSTRACT as target and LEAD_PARAGRAPH as source for seq2seq model. We tokenized source and target changing numbers to “N”, and removed all punctuations. We did not separate sentences because the LEAD_PARAGRAPH and ABSTRACT does not have one to one sentence to sentence relationship; the mapping is only at paragraph level.

We then built joint vocabulary, and mapped words to integer indexes. The graph was built based on section 3.4.

We split data into smaller batches and fed it into the graph for training. Every 100 batches, we peak into training and validation performance with a sample sentence rewritten to output.

5. Results and Discussion

5.1 Summary Statistics

Assuming the lead paragraph is used as the input for the abstractive model, the length of input is right skewed with some extreme outliers. (See Figure 2)

On average there are 145 tokens in the input paragraph and 43 tokens in the actual abstract of the articles. And on average there are 6 sentences in the input and 35 sentences in the full article.

As we pad the sentences to the be the same length in each batch, we chose only input<200 and target<100 articles, so that we remove the outlier what are too long for our machine.

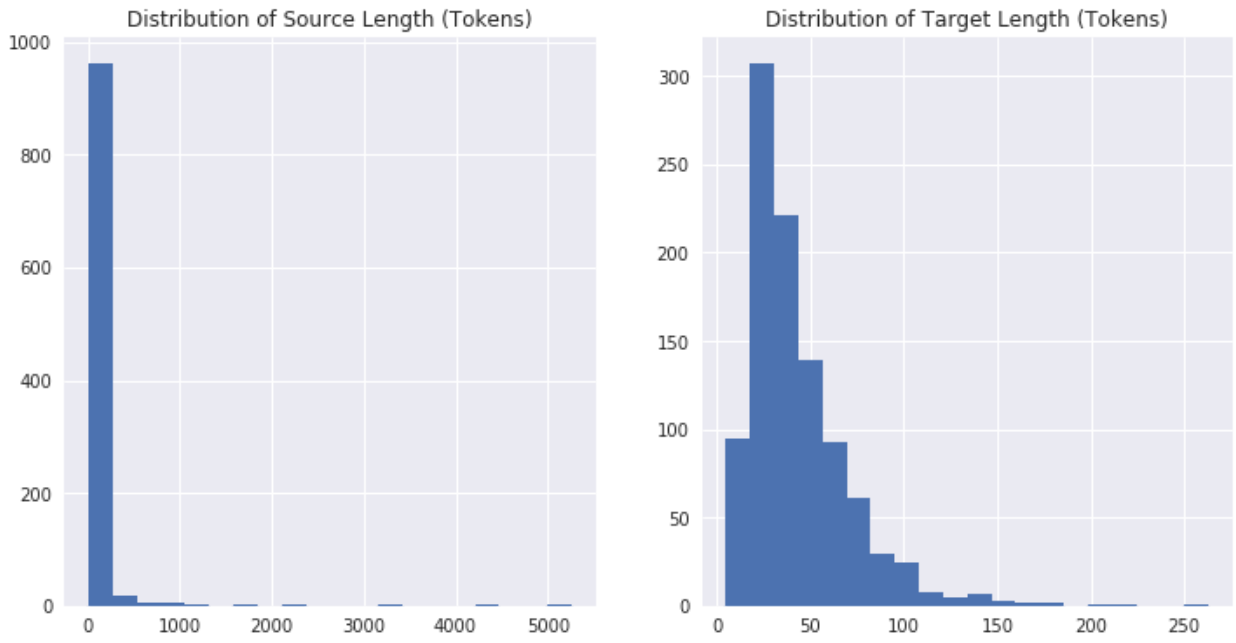

Figure 2 Text length of input and output of the seq2seq model

5.2 Results

For reference, table 1 below is from Chen and Bansal (2018), representing the state-of-the-art document summarization using CNN/Daily Mail data.

Models	ROUGE-1	ROUGE-2	ROUGE-L	METEOR
Extractive Results				
lead-3 (See et al., 2017)	40.34	17.70	36.57	22.21
Narayan et al. (2018) (sentence ranking RL)	40.0	18.2	36.6	-
ff-ext	40.63	18.35	36.82	**22.91**
rnn-ext	40.17	18.11	36.41	22.81
rnn-ext + RL	**41.47**	**18.72**	**37.76**	22.35
Abstractive Results				
See et al. (2017) (w/o coverage)	36.44	15.66	33.42	16.65
See et al. (2017)	39.53	17.28	36.38	18.72
Fan et al. (2017) (controlled)	39.75	17.29	36.54	-
ff-ext + abs	39.30	17.02	36.93	20.05
rnn-ext + abs	38.38	16.12	36.04	19.39
rnn-ext + abs + RL	40.04	17.61	37.59	**21.00**
rnn-ext + abs + RL + rerank	**40.88**	**17.80**	**38.54**	20.38

Table 1: Results on the original, non-anonymized CNN/Daily Mail dataset. Adding RL gives statistically significant improvements for all metrics over non-RL rnn-ext models (and over the state-of-the-art See et al. (2017)) in both extractive and abstractive settings with $p < 0.01$. Adding the extra reranking stage yields statistically significant better results in terms of all ROUGE metrics with $p < 0.01$.

The results of four models using New York Times corpus are in table 2. The baseline has the highest score which is not a surprise for news articles. Our sentence ranking

model achieved scores close to those in the recent papers. However, the seq2seq models only got XX and didn't improve much after adding attention.

Models	ROUGE-1	ROUGE-2	ROUGE-L
Baseline (Lead Paragraph)	45.98	21.97	39.40
Sentence Ranking	38.06	17.53	31.14
Lead + seq2seq			
Lead + seq2seq + attn			

Table 2: Results on the New York Times corpus (score is calculated based on 10,000 samples)

5.3 Observation

When first ran the basic sequence-to-sequence model on a small data set[3], we noticed the issues mentioned in Chen and Bansal (2018). The model generated random and repeated words or didn't generate enough words at all.

To diagnose the model with limited machine power, we chose to use target as source, the idea is this should train a model rewrite literally. We were able to achieve this goal with 113K short sentences without attention in less than 5 Epochs. During training, we were able to observe the generated sentences get closer and closer to the source, and by Epochs 5, it mostly generating exactly the same as source, which is what we expected because our training source and training target were set to identical.

This test indicated our graph will generate expected outcome based on the data we feed to it.

We then move on to feed all actual data (580K) with attention which is quite memory and CPU consuming.

In addition, we tried tuning the values of key parameters, including learning rate (0.0025), the number of epochs (5), the number of layers(3), embedding dimension (300), dropout rate(0.9).

[3] Due to long runtime, we often analyzed and tuned the models on a small data set (1k~4k articles). Thus, the analysis may not hold for the whole data set.

When we found the loss kept decreasing, we know that we haven't reached the minimum loss yet and increased the number of epochs till the loss didn't change much between epochs. When we see the loss oscillating, we know the learning rate is too high. Next, increasing the number of layers generally helps generating more meaningful words, but slows down training. Attention also slows down training dramatically (to the order of dozens)

Defining the vocabulary is a little tricky here. We built the vocabulary based on the input (selected sentences) and output of the seq2seq model, not the whole articles. It's a surprise, when the vocabulary was expanded to cover the full contents, the output was worse. When we saw the very common words generated, we were thinking about removing stop words. However, we don't think we should remove stop words from the source or target as they are useful in generating natural language.

In contrary to the machine translation which performs better if read in the source sentence in the reverse order, the model generates better words if read in the source in the original order.

Example

Here is an example for the model output.

Original Text (truncated): While standing, smooth down as much fabric as possible with both hands. Then cross the knees, bend and sit. Optional precaution: place purse over lap.This peculiar choreography is the daily routine for women wearing miniskirts -- extremely short miniskirts -- as they try to grab a subway seat with all the grace and modesty they can muster.The city can often be an obstacle course for fashionistas, with grates that threaten to trip stilettos at every corner and humidity that can turn any fabric into virtual Saran Wrap.But the chic cannot be ruled by fear. So when hemlines that fall just centimeters below one's bottom became all the rage this summer, young women who dared found a way.The goal is to avoid having your bottom "touch too much of the seat," said Jessica Oser, 24, who sported a lavender polo shirt and khaki mini as she browsed at the Union Square farmers' market this week.Ms. Oser, a student at the Benjamin N. Cardozo School of Law, never wears a miniskirt when she takes a New Jersey Transit train to visit her parents, since crossing her legs properly seems infinitely more difficult on those train seats.The consensus is that not all miniskirts are created equal. Pleated fabric is better because it moves around with the derrière; denim is stiff and more likely to get in the way; cotton is neater to smooth out
Abstract: Women wearing miniskirts describe how to sit on subway seat with modicum of grace and modesty; photos (M)
Baseline (lead paragraph): While standing, smooth down as much fabric as possible with

both hands. Then cross the knees, bend and sit. Optional precaution: place purse over lap.This peculiar choreography is the daily routine for women wearing miniskirts -- extremely short miniskirts -- as they try to grab a subway seat with all the grace and modesty they can muster.
Sentence Ranking: This peculiar choreography is the daily routine for women wearing miniskirts-- extremely short miniskirts-- as they try to grab a subway seat with all the grace and modesty they can muster. The consensus is that not all miniskirts are created equal. You always have to be adjusting," she said, tugging at her cotton black flower print miniskirt as she waited at Union Square for the subway." Those subway grates can blow up miniskirts the same way they got Marilyn Monroe's dress. It is far more likely to stick to the seats or be caught in a compromising position, miniskirt fans say. Choice undergarments for skirts that bare skin?
Seq2Seq[4]: new of reviews of new of new of new of new of new of new of new of new of the and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and and
Seq2Seq + Attention:

Figure 3: Example output from multiple models

6. Conclusion

We apply graph-based text ranking algorithm on the New York Times corpus to select salient sentences for summarizing the article and achieve high ROUGE scores.

The sequence-to-sequence model with attention we implement did not beat the extractive method. What we observe though is for abstractive model to improve its performance, significant computing horsepower is needed -- when we tried to reduce amount of training data to get faster result, the predictive power of the model drops significantly. Using a small portion of the data is not a viable approach in tuning the model.

[4] Here seq2seq model use baseline as input for the encoder.

References

Alexander M. Rush, Sumit Chopra, Jason Weston. 2015. A Neural Attention Model for Abstractive Sentence Summarization.

Dipanjan Das, Andre F.T. Martins. 2007. A Survey on Automatic Text Summarization.

Yen-Chun Chen, Mohit Bansal. 2018. Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting.

Gunes Erkan, Dragomir R. Radev. 2004. LexRank: Graph-based Lexical Centrality as Salience in Text Summarization.

Shashi Narayan, Shay B. Cohen, Mirella Lapata. 2018. Ranking Sentences for Extractive Summarization with Reinforcement Learning.

Abigail See, Peter J. Liu, Christopher D. Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks.

Rada Mihalcea, Paul Tarau. 2004. TextRank: Bringing Order into Texts.

Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio. 2016. Neural Machine Translation by Jointly Learning to Align and Translate.

Ilya Sutskever, Oriol Vinyals, Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks.

Tensorflow Neural Machine Translation (seq2seq) Tutorial https://github.com/tensorflow/nmt, https://towardsdatascience.com/seq2seq-model-in-tensorflow-ec0c557e560f

3 silver bullets of word embeddings in NLP
https://towardsdatascience.com/3-silver-bullets-of-word-embedding-in-nlp-10fa8f50cc5a

TextRank for Text Summarization https://nlpforhackers.io/textrank-text-summarization/

Python implementation for testrank: https://github.com/ceteri/pytextrank

Rouge evaluation: https://github.com/miso-belica/sumy/blob/dev/sumy/evaluation/rouge.py