Explorations in Abstractive Text Summarization

Nilesh Bhoyar, Sue Yang
W266: Natural Language Processing, 2017 Spring
UC Berkeley School of Information
nilesh.bhoyar@ischool.berkeley.edu, suey@ischool.berkeley.edu

Abstract- This work focuses on the exploration of methods to address the challenges of abstractive summarization. The popular Seq2Seq model which was proposed by Sutskever et al. was used. The attentional encoder-decoder recurrent neural network was modified and extended for the problem of Amazon review summarization, and the effectiveness is evaluated by using ROUGE score and human evaluation.

Keywords: text summarization, encoder, decoder, attention

1. INTRODUCTION

The expression “too much information kills information” is so true in today's world. With ever-growing volume of information, it has become impossible for humans to track. Text summarization helps process information more efficiently, in particular:

- In reducing reading time
- In expediting research
- Summary by algorithm is less biased than that of human summarizations.

Our obsession with summarization dates back to 1780’s when Joseph Joubert was summarizing big books in small letters and notebooks. Many computer techniques have been introduced in past 50 years for automatic text summarization. Though we are getting better, many challenges remain unresolved. For instance, how to evaluate the quality of summarization? Can we deem one summary better than the other? Does a “perfect” summary exist for any document?

Text summarization is helpful in scenarios where we form opinions from those of others, such as from the reviews on Amazon or Yelp. However, simply knowing if a review is favorable, or knowing the top popular words may not be meaningful enough at times. Text summarization aims to address if key information can be extracted for the users to skim through the contents quickly and accurately.

2. BACKGROUND

Many works in summarization are of extractive approach, which use word frequency to determine the importance of a word and select sentences. Typical techniques include TextRank, LexRank and TextTeaser etc. But all of these approaches are based on the hypothesis that the core idea of a document can be summed up by a sentence or a few words in a document.

Abstractive methods, however, are more similar in the ways of how human beings generate summaries. The generated summaries are more coherent and concise as well. In the application of Machine Translation and Text Summarization, the encoder-decoder framework and attention mechanism has become especially popular.

3. METHODS

Revisiting some of the basics before diving into recurrent neural networks, RNN is an example of deep neural network. Text summarization is equivalent to finding a target sentence Y that maximizes the conditional probability of Y given a source sentence X. Input sentence X is nothing but the series of inputs and output Y is the prediction. The image below shows the layer of recurrent neurons unrolled through time. Each recurrent neuron has two sets of weights: one for the inputs and the other for the output from the previous step.

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{x}_{(t)}{}^{T} \cdot \mathbf{w}_x + \mathbf{y}_{(t-1)}{}^{T} \cdot \mathbf{w}_y + b\right)$$

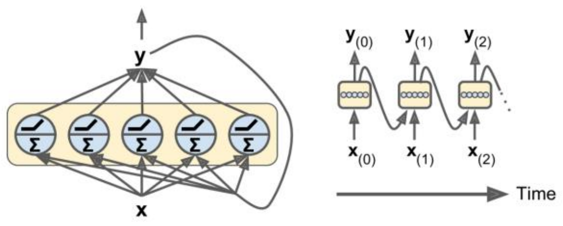


Figure 1. Layer of Recurrent Neurons Unrolled through Time

RNN has two major issues: it does not do well for long term dependencies, and its gradient quickly diminishes when a long sequence is computed. LSTM and its simplified version GRU solve these two issues.
LSTM cell with its key component highlighted is shown in the diagram below. GRU cell is the simplified version of LSTM[5].

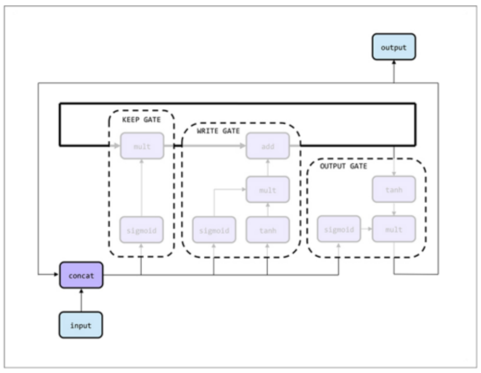


Figure 2. LSTM Cell

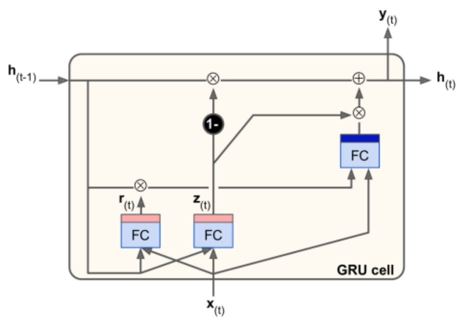


Figure 3. GRU Cell

Our model is inspired by the paper on ABS summarization (Facebook & Harvard)[1] and the machine translation encoder-decoder models. Instead of translating one language to another, we are translating long text to a shorter abstract one.
Our sequence to sequence learner has the architecture below:

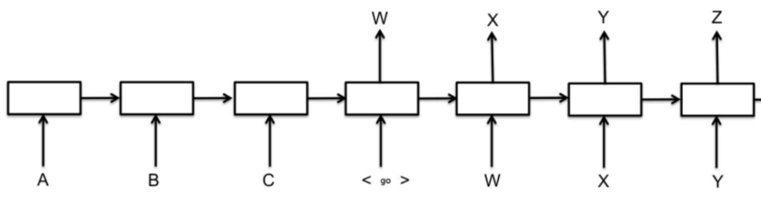


Figure 4. Architecture of sequence-to-sequence model

We take our cells to be LSTM/GRU recurrent units, with dropout between the feed-forward layers. The decode_output is a tensor of shape (batch_size, output_vocab_size). We can run softmax on this to get logit for each letter. The details are in the following section.

3.1 DATA COLLECTION AND PREPROCESSING

Dataset

This dataset consists of reviews of fine foods from Amazon. The data span a period of more than 10 years, including approximately 500,000 reviews up to October 2012. Reviews include product and user information, ratings, and plaintext. The dataset can be downloaded from here. A sample dataset looks like the following:

```
product/productId: B001E4KFG0
review/userId: A3SGXH7AUHU8GW
review/profileName: delmartian
review/helpfulness: 1/1
review/score: 5.0
review/time: 1303862400
review/summary: Good Quality Dog Food
review/text: I have bought several of the Vitality canned dog fo
found them all to be of good quality. The product looks more lik
processed meat and it smells better. My Labrador is finicky and
product better than most.
```

Figure 5. Sample Data (Amazon Review)

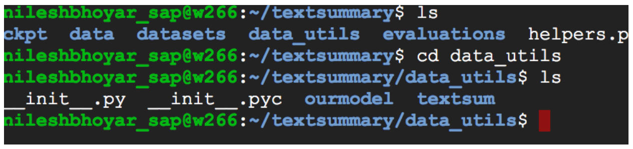
Where:

- product/productId: asin, e.g. amazon.com/dp/B001E4KFG0
- review/userId: id of the user, e.g. A3SGXH7AUHU8GW
- review/profileName: name of the user
- review/helpfulness: fraction of users who found the review helpful
- review/score: rating of the product
- review/time: time of the review (unix time)
- review/summary: review summary
- review/text: text of the review

Data Processing

We have 2 data processing routines. One for our model and the other for producing an output in the format acceptable by Google textsum: https://github.com/tensorflow/models/tree/master/textsum

This is to compare how our model performs against the others.



```
nileshbhoyar_sap@w266:~/textsummary$ ls
ckpt  data  datasets  data_utils  evaluations  helpers.p
nileshbhoyar_sap@w266:~/textsummary$ cd data_utils
nileshbhoyar_sap@w266:~/textsummary/data_utils$ ls
__init__.py  __init__.pyc  ourmodel  textsum
nileshbhoyar_sap@w266:~/textsummary/data_utils$ 
```

Figure 6. Project Structure on Google Cloud Platform

For data processing, we first filter reviews by setting the max and min length of the reviews allowed. We remove stop words using nltk libraries. Once data is cleaned, the vocabulary, word to index and index to word mappings are created, then we pad the records to make them in fixed length.

3.2 SEQ2SEQ AND ATTENTION MODEL

We implement our model using encoder–decoder network mechanism. The encoder reads the source sentence x = (x1, · · ·, xT) and encodes it into a sequence of hidden states h = (h1, · · · , hT).

$$h_t = f(x_t, h_{t-1})$$

Then the decoder, which is also RNN generates a corresponding summarization y = (y1, · · ·, yT0) based on the encoded sequence of hidden states ht.

$$p(y_{i+1}|y_c, x; \theta) = softmax(V h + W enc(x, y_c) + b2)$$
$$y_c = [Ey_{i-c+1}, ..., Ey_i],$$
$$h = tanh(Uy_c + b1)$$

where y_c is the context (previous *C* words), *V, W, U* are weight metrics and *E* is an embedding matrix. *enc* represents an encoder node. For Seq2Seq without attention we use tf.nn.seq2seq.embedding_rnn_seq2seq Tensforflow API.

3.2.1 ATTENTION

Decoder model is supposed to produce output sequence only based on output of the last hidden state from the encoder, i.e., the last hidden state should learn everything from the sequence. This works well if sentences are short, however, in our case reviews are more than 50 words in length. It is unreasonable to assume that we can encode all information in such a long sequence in a single vector. With attention, we allow the decoder to "attend" to different parts of the source sentence at each step of the output generation. The key here is that we let the model **learn** what to attend to base on the input sentence, and what it has produced so far.

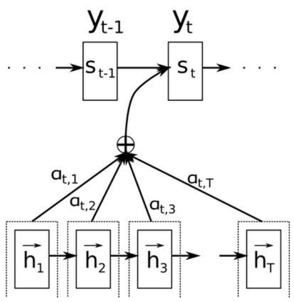


Figure 7. Attention Mechanism in "Neural Machine Translation by Jointly Learning to Align and Translate"

Decoder output word y_t now depends on a weighted combination of all the input states, not just the last state. The attention-based encoder is defined as:

$$\begin{aligned} \mathrm{enc}_3(\mathbf{x}, \mathbf{y}_c) &= \mathbf{p}^\top \bar{\mathbf{x}}, \\ \mathbf{p} &\propto \exp(\tilde{\mathbf{x}} \mathbf{P} \tilde{\mathbf{y}}'_c), \\ \tilde{\mathbf{x}} &= [\mathbf{F}\mathbf{x}_1, \ldots, \mathbf{F}\mathbf{x}_M], \\ \tilde{\mathbf{y}}'_c &= [\mathbf{G}\mathbf{y}_{i-C+1}, \ldots, \mathbf{G}\mathbf{y}_i], \\ \forall i \quad \bar{\mathbf{x}}_i &= \sum_{q=i-Q}^{i+Q} \tilde{\mathbf{x}}_i / Q. \end{aligned}$$

Where *G* is an embedding of the context, *P* is a new weight matrix parameter mapping between the context embedding and input embedding, and *Q* is a smoothing window. For Seq2Seq with attention, we use Tensorflow API tf.nn.seq2seq.embedding_attention_seq2seq.[4]

4. RESULTS AND DISCUSSION

4.1 EVALUATION

Recall

Since standard Perl ROUGE library is hard to wrap in python and it contains may preprocessing we do not need, we write our own ROUGE function to check the recall rate of the words (n-grams) in the reference summaries appearing in the machine generated summaries. ROUGE is an automatic summary evaluation method proposed by Lin and Hovy of ISI, and it is widely used in the summary evaluation task of DUC1 (Document Understanding Conference). ROUGE evaluates the summary based on the co-occurrence information of the n-gram in the summary. The typical n-gram size is 1 to 4. We used ROUGE-1 and ROUGE-2 recall scores in our work to check for the overlap of unigrams and bigrams. ROUGE-N is computed as follows [2]:

$$\frac{\sum_{S \in \{Referemce\ Summaries\}} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in \{Reference\ Summaries\}} \sum_{gram_n \in S} Count(gram_n)}$$

Where n stands for the length of the n-gram, $gram_n$ and $Count_{match}(gram_n)$ is the number of n-grams co-occurring in a candidate summary and a set of reference summaries.

Precision

BLEU is similar to ROUGE score, however it measures the precision instead: how many words (n-grams) in the machine generated summaries appear in the reference summaries.

Accuracy

We use general F1 score to test the accuracy. It is computed as follows based on BLEU and ROUGE score [3]:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\mathrm{recall}} + \frac{1}{\mathrm{precision}}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

4.2 IMPLEMENTATION AND RESULTS

We implemented all models using Tensor Flow and used Adam Optimizer, which computes adaptive learning rates for each parameter. Our model can be executed using run.py in four different ways:

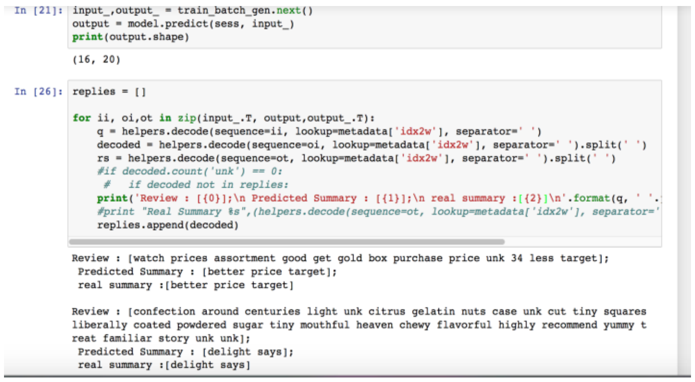
- LSTM without Attention
- LSTM with Attention
- GRU without Attention
- GRU with Attention

Initially we tried to train the model with sequences longer than 100 words, but then quickly realized its limitations. Seq2Seq for large sequences consumes huge amount of memory and terminates without completing the training. Also models with attention are quite slow compared to without attention and they additional memory for attention variables. So we end up using below configuration

Model Name	Input(Review) Length	Output(Summary) Length	Vocab Size
Models without Attention	200 words	30 Words	176K
Models with Attention	30 Words	20 Words	20K

Before we run these models for all records we used memorization technique to make sure they are working i.e. they can memorize small sets. **Run_training.ipynb** notebook can be used to test models for small data and also to learn what model does.

Example of memorization



```
In [21]: input_,output_ = train_batch_gen.next()
         output = model.predict(sess, input_)
         print(output.shape)

         (16, 20)

In [26]: replies = []

         for ii, oi,ot in zip(input_.T, output,output_.T):
             q = helpers.decode(sequence=ii, lookup=metadata['idx2w'], separator=' ')
             decoded = helpers.decode(sequence=oi, lookup=metadata['idx2w'], separator=' ').split(' ')
             rs = helpers.decode(sequence=ot, lookup=metadata['idx2w'], separator=' ').split(' ')
             #if decoded.count('unk') == 0:
              #   if decoded not in replies:
             print('Review : [{0}];\n Predicted Summary : [{1}];\n real summary :[{2}]\n'.format(q, ' '.
             #print "Real Summary %s",(helpers.decode(sequence=ot, lookup=metadata['idx2w'], separator='
             replies.append(decoded)

         Review : [watch prices assortment good get gold box purchase price unk 34 less target];
          Predicted Summary : [better price target];
          real summary :[better price target]

         Review : [confection around centuries light unk citrus gelatin nuts case unk cut tiny squares
         liberally coated powdered sugar tiny mouthful heaven chewy flavorful highly recommend yummy t
         reat familiar story unk unk];
          Predicted Summary : [delight says];
          real summary :[delight says]
```

Final Training time and hyper parameters were used as below

Model Name	Training Time	Epochs	Number of hidden Layers	Embedding Size	Batch Size	Learning Rate
GRU without Attention	> 24 hrs	60000	3	64	16	0.005
GRU with Attention	> 38 hrs	6642	3	64	16	0.005
LSTM without Attention	6 hrs	30000	2	64	16	0.005
LSTM with Attention	16 hrs	30000	2	64	16	0.005

Even after running models for so long, our results are not what expected. Abstract summarization is difficult problem and to solve it 500K data sets is certainly not enough. Models with attention are definitely better than without for obvious reasons but their complexity and memory size makes it hard to train them.

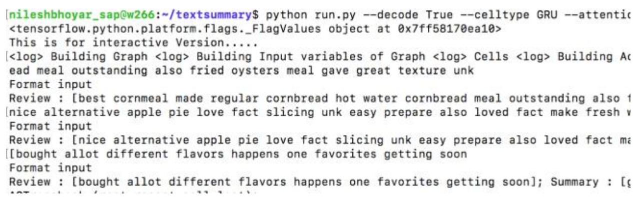
Our final results are as below

Systems	ROUGE-1	Blue-1	Loss
LSTM	0.041	0.026	0.705414
LSTM + Attention	0.047	0.015	0.634441
GRU	0.023	0.032	0.90876
GRU + Attention	0.028	0.015	1.23905

We generated summary of one word in different systems (see 4.3.1 and 4.3.2), hence the evaluation scores will be the same.
Run.py also has the capability to use models in interactive ways, i.e., one can execute run.py in decode mode and enter the review which they want to summarize. Model will reload from the saved session and produce the required decoding. This is very helpful for the qualitative analysis. More information on how to execute this model can be found on github documentation:
https://github.com/nileshbhoyar/textsummary

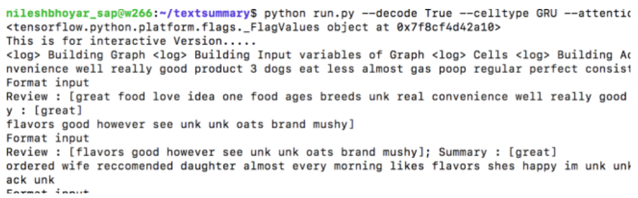
4.3 QUALITATIVE ANALYSIS

Models can be executed in decode mode and user can manually analyze the summaries generated.
For instance to do qualitative analysis of GRU cell without attention mechanism, we will execute run.py as below



```
nileshbhoyar_sap@w266:~/textsummary$ python run.py --decode True --celltype GRU --attentic
<tensorflow.python.platform.flags._FlagValues object at 0x7ff58170ea10>
This is for interactive Version.....
<log> Building Graph <log> Building Input variables of Graph <log> Cells <log> Building Ac
ead meal outstanding also fried oysters meal gave great texture unk
Format input
Review : [best cornmeal made regular cornbread hot water cornbread meal outstanding also f
nice alternative apple pie love fact slicing unk easy prepare also loved fact make fresh w
Format input
Review : [nice alternative apple pie love fact slicing unk easy prepare also loved fact ma
[bought allot different flavors happens one favorites getting soon
Format input
Review : [bought allot different flavors happens one favorites getting soon]; Summary : [g
```

With Attention



```
nileshbhoyar_sap@w266:~/textsummary$ python run.py --decode True --celltype GRU --attentic
<tensorflow.python.platform.flags._FlagValues object at 0x7f8cf4d42a10>
This is for interactive Version.....
<log> Building Graph <log> Building Input variables of Graph <log> Cells <log> Building Ac
nvenience well really good product 3 dogs eat less almost gas poop regular perfect consist
Format input
Review : [great food love idea one food ages breeds unk real convenience well really good
y : [great]
flavors good however see unk unk oats brand mushy]
Format input
Review : [flavors good however see unk unk oats brand mushy]; Summary : [great]
ordered wife reccomended daughter almost every morning likes flavors shes happy im unk unk
ack unk
```

As you can see, though our ROGUE/BLUE scores are not impressive. Models are able to generate the correct summary for most part. It's able to recognize whether review is really great/average or moderate i.e. it's able to capture emotions and deep message within.

We were accepting longer review strings with attention models, but as we could not run model for very long time as we ran out of google cloud credits so we had to stop in between.

5. FURTHER IMPROVEMENTS

Our results can be improved by numerous ways:-

1. Bi-directional Networks- We implemented feed forward DNNs i.e. we are using knowledge of only one side of sentence. Bi-directional networks should improve embeddings and thereby scores
2. More data and training: This NLP application definitely needs more data and training due to inherent nature of problem.
3. Some summaries are longer than 400 words (up to 2500 words) for those we had to truncate strings. This can improved if truncation of string done with some logic by using extractive techniques first and then train the deep network.
4. For data processing, add more buckets to allow variable length of sequences. Also merge extractive technique with deep learning.

Reference

[1] A Neural Attention Model for Sentence Summarization https://www.aclweb.org/anthology/D/D15/D15-1044.pdf
[2] Looking for a Few Good Metrics: ROUGE and its Evaluation http://83.212.103.151/~mkalochristianakis/techNotes/ipromo/rougen5.pdf
[3] F1 Score https://en.wikipedia.org/wiki/F1_score
[4]https://arxiv.org/pdf/1409.0473v7.pdf One of the best paper on Attention
[5] "Long Short-Term Memory," S. Hochreiter and J. Schmidhuber (1997).