Abstractive Text Summarization with Local Attention Mechanism

Recep Inanc
Data Science - Artificial Neural Networks
Coventry University
Coventry, UK
inancr@uni.coventry.ac.uk

***Abstract*— In this paper abstractive text summarization is done using Recurrent Neural Networks and local attention an Abstractive text summarization is a way of summarizing any given text. As opposed to Extractive Summarization, Abstractive summarization's results are not limited to the words used in the original text. In this paper, RNNs and Local attention mechanism are used to the summary generation. This approach uses, forward and backward LSTMs with RRA for encoder-decoder architecture and local attention mechanism for summary scoring and generation. I analyzed the hyperparameters and their effect on the results and discussed the results.**

Keywords: Text summarization; LSTM; attention mechanism; RRA; RNN

I. Introduction

The data on the internet is getting more and more every day and this data is mostly in the form of text. The problem with this is that the meaningful part of the data is actually a small portion of that text and the rest is just clutter. The way to clear this clutter is to summarize the texts.

By the use of computers, we can make automatic summarization and there are 2 main methods to do automatic summarization. First one is Extractive summarization, which is mostly used because it is easier. It is guaranteed that gives you a summary consisting of sentences that are grammatically correct and readable since they are the same sentences that were given in the original text. On the other hand, the extractive approach is too restrictive to produce human-like summaries – especially for longer, more complex text. Because creating a summary of a novel is not the same thing as taking some of the sentences of a novel as they are and putting them in a correct order [1]. The second approach is Abstractive summarization, which is computationally more expensive, but it produces summaries by paraphrasing the given text that means it may contain words that are not in the original text.

This task can be considered as a sequence to sequence problem, where we get the actual text word by word as a sequence and output another text again word by word as a sequence. In the past, there have been many studies on sequence-to-sequence models, and they have been very successful. Sequence-to-sequence models are applied to any problem where you can consider the data as a sequence, rather than a fixed length one-piece input. Such as speech recognition [2], video captioning [3] and machine translation [4]. The state-of-the-art work in this field states that long short-term memory (LSTM) is the most successful option for this task since they are easier to train than vanilla RNNs [5]. This paper uses many previous studies as inspiration, the main source is this work [5], that implements several models to solve critical problems in abstractive text summarization. The other most inspiring work is this one [6], that inspired this work a lot in the implementation phase.

In this paper, we use the state-of-the-art approach and use LSTMs instead of vanilla RNNs. We implement the architecture by using a bi-directional encoder which consists of two LSTMs with recurrent residual attention (RRA) mechanism. The uni-directional decoder on the other side of the architecture is also an LSTM that uses the encoded sequence of input with the help of RRA in between.

II. The Methodology

As we mentioned earlier in this paper, abstractive text summarization is a sequential problem, meaning that the input used, and the output generated are both sequences of values [7, fig.1], and it is proven that using a sequence-to-sequence model would result in a better performance for such problem [8].

So, based on these studies, in this paper, abstractive text summarization method is implemented with the help of Recurrent Neural Networks (RNN) since Deep Neural Networks (DNN) are proven to be not as effective as RNNs because not having a fixed-length output is one of the main issues for DNNs [5].

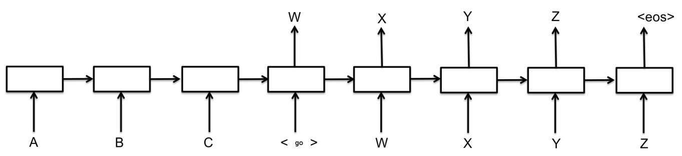


Figure 1- Basic sequence to sequence model. In this figure, the text ABC is used as input and converted into WXYZ as the output. Each letter is taken at a new timestep making this a sequential model.

Attention mechanism is used to increase the performance of this basic model. The attention mechanism aims to solve the main issue of what part of the encoded data to focus on when decoding [9]. Because not all part of the encoded data is relevant to the part that is currently being decoded. The attention model comes between the encoder and the decoder and helps the decoder to pick only the encoded inputs that are important for each step of the decoding process [9, fig. 2].

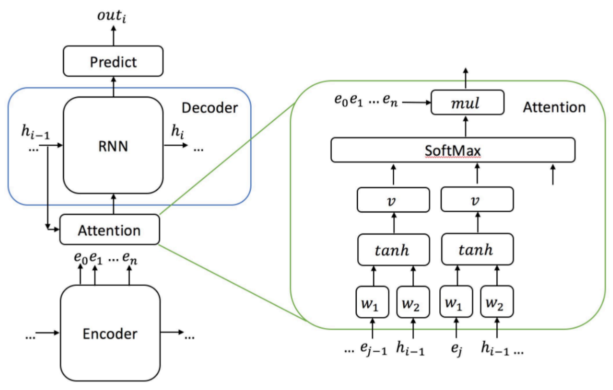


Figure 2-Attention Mechanism. Attention Model is positioned between encoder and decoder.

Usage of RRA in this work is based on the paper [10]. RRA is used for creating connections between timesteps so that after some K number of inputs, the current RRA cell will be able to reach the context of the previous cells so that it can control the contributions of previously hidden states. As it can be seen in [10, fig.3] the usage of recurrent residual attention helps the model to understand the context with the help of easy access to previous states. In the implementation we use K=8 as our RRA connection range.

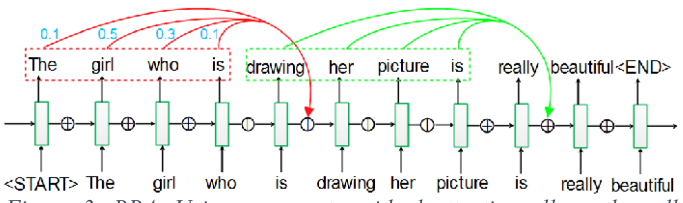
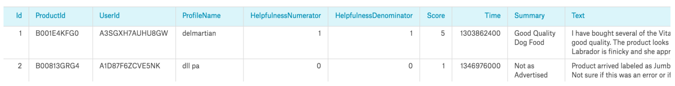


Figure 3- RRA. Using recurrent residual attention allows the cell before "her" to find out that the subject used was "girl" in the previous timesteps.

The encoder-decoder architecture used in this model is based on the architecture described in the paper [11]. The bidirectional RNN encoder reads the source word sequence forward and backward. The forward RNN reads the word sequence in its original order and generates a hidden state *fhi* at each time step. Similarly, the backward RNN reads the word sequence in its reverse order and generates a sequence of hidden states (bhT , ..., bh1). The final encoder hidden state, hi, at each time step i is a concatenation of the forward state fhi and backward state bhi, i.e. hi = [fhi, bhi].

III. Dataset

The dataset used in this paper is a publicly available “Amazon Fine Food Reviews” dataset that. The dataset consists of reviews/comments about Amazon fine food services and their summaries up to October 2016 with some additional fields, such as date, id etc. The size of the dataset is 568,454.



Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vita good quality. The product looks Labrador is finicky and she appr
2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumb Not sure if this was an error or if

Figure 4-Amazon Fine Food Reviews dataset. Columns: Id, ProductId, UserId, ProfileName, HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary, Text

A. Accessing to dataset

Since the dataset is publicly available, both SQL and CSV formats can be found at https://www.kaggle.com/snap/amazon-fine-food-reviews/data. The dataset size is 118.65 MB for csv version.

B. Pre-processing the Dataset

Like in any other machine learning/deep learning model before using the data, some operations should be performed on the dataset to make the dataset more usable, and the model more solid. The methods applied here are almost the same as the ones that are used in any other NLP models.

The first thing we do is to remove columns that are not going to be used, so except *text* and *summary,* we remove all other tows. Then the second thing is to remove the rows that contain null or NA value. Dealing with missing values for NLP it is good to remove the rows that contain any missing text value since it cannot be replaced like the way we do it for numerical values. And since the dataset we are using is populated enough, deleting the rows that contain NA values will not do us any harm. Next, we remove all punctuation from the dataset since I have experimented it that punctuation marks are confusing the system so much that all predictions turn into a bunch of punctuation marks. Then we apply the 1.5IQR rule, which is a statistics-based outlier elimination rule, we apply this rule to all dataset (column-wise) to remove the rows that have any value whose length is below or above this range. Then we lowercase all the characters in the dataset,

C. Word Embedding

To be able to use the dataset we have now, we need to find a way to represent the text data in our model, so that the relationship between each word can also be preserved. First, we convert each word in the set to their corresponding indexes using a dictionary.

For example, a sentence like “Deep learning is fun” can be mapped as “123 22 512 78”. Since the word indexes do not contain any semantic information RNN cannot make use of this, so we map each word to a vector in a higher dimensional space. This process is called word2vec encoding, each word is represented as word vectors, and their relation, similarity, the difference is calculated by the cosine angle between them. So, for example, the words "Germany" and "Berlin" should have the same similarity between "France" and "Paris". To convert this word indexes into vectors, we use a word embedding lookup table. We download GloVe [12] pre-defined word embedding to be able to map our indexes into vectors easily.

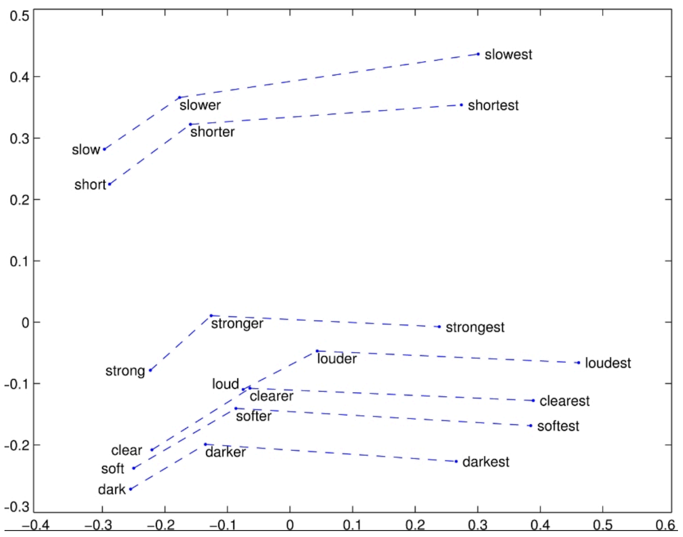


Figure 5- Word Vectors representation

D. *Tokenization*

This step is pretty straightforward, by using the NLTK library we tokenize all the text data in the set. A sentence like "Deep learning is fun" will become an array in the form of ["Deep", "learning", "is", "fun"]. After tokenizing all the data, we add EOS (end of sentence), SOS (start of sentence) and UNK (unknown) special tokens to the appropriate places. UNK is replaced with a token that whose vector representation cannot be found in our word embedding lookup table. After this step, our data is ready to be used.

E. *Further Considerations*

In addition to these steps, we could apply stopwords filtering, we could replace the words that are used with less than a certain frequency.

IV. TRAINING

The training part of this model is the one that took most of the time, since the data we are using has a huge variance, and our aim was to find the connection between all the words that we might face when predicting the summaries. Having over 500k data on this work was a great opportunity, whereas the computational power required to process all these data could not be found. In order to maximize the model accuracy a lot of different combinations of training dataset size, hidden layer size, training/validation/test splits, learning rate and other hyperparameters have been tried.

The dataset is first limited to the size of 100 samples just to see how the other parameters effects are. After some research hidden layer size is set to 175, based on the three rules [13]:

The number of hidden neurons should be:

- Between the size of the input layer and the size of the output layer
- 2/3 the size of the input layer, plus the size of the output layer
- Less than the twice the size of the input layer

The number of epochs used in this training set changes between 1 and 100, but the values over 20 are the ones that give the best results. For the learning rate, the value of 0.03 is empirically found to be the best option for our dataset.

The actual training is done on a dataset of 1000 samples, with training set ratio 0.8, validation set ratio 0.1 and test set ratio 0.1. Normally, recurrent neural networks are hard to train and on top of that we add more complexity to the system by adding the attention mechanism and the bi-directional encoder, which makes it harder to train.

In this model, instead of feeding the training set at once, we use mini batching for couple of reasons. Let's consider using all dataset at once two train, so 1 iteration per epoch. If we do this, our model will not be wandering around to find the best path to global minimum but instead since it is processing all the data we have it will find the way in fewer steps in terms of gradient descent steps. However, this approach is computationally very expensive, considering the dataset and the resources we have. Let's think about the other extreme which is taking a batch size of 1 and having 1 iteration per sample for training will cause in lots of wandering around for the global minimum, eventually it will also take us to the global minimum, making path choices for every sample may help the model the "bounce" out of local minima. A better approach is the one that's in the middle of these two, it is to use mini-batches. Instead of taking the whole dataset or taking them one by one we take the data in batches of 32, 64, 128 etc. and iterate over all batches at each epoch.

After employing mini-batch to our model, I realized that the model is converging too fast at high loss values. This is an indicator of a possible local minima that prevents us from reaching the global optima. There are many studies on how to avoid local minima. The main reason we face with a local minimum is the problem of non-convexity. Which is likely to happen if the training dataset is not changing overall iterations. We employ shuffling the dataset before each epoch so that every epoch batch are generated from random sets. Having batches in random at every epoch enables the model to bounce out of local minima because every time it is directed towards a local minima, a new randomized set of data will have effect on the new hidden states which will prevent any mis-leading gradient descent step.

One of other the most suggested way is to add some momentum and keep learning rate changing through epochs. This might help the model to get out of the local minimum it stuck.

Among many different configurations, the best results for training is as follows:

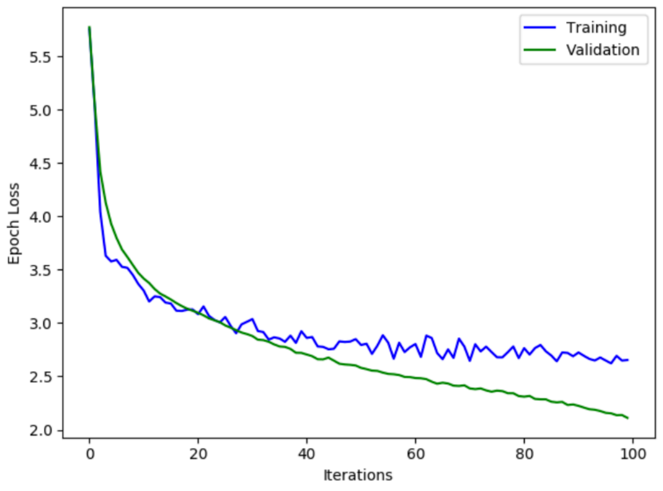


Figure 6- Training/Validation Loss Plot. 100 epochs; 0.003 learning rate; 640 training set size, 160 validation set size; 175 hidden layer size;

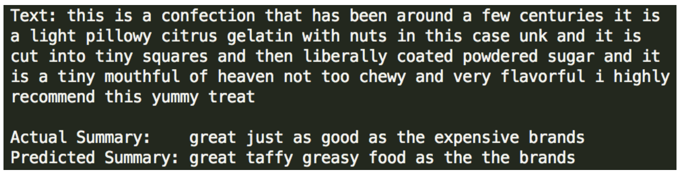
The results show lower loss values, and higher training loss values, the most likely reason this is the diversity among the data used both in training and validation sets.

V. TESTING

The testing is done on a dataset of 100 reviews. The tests are also run on the same computer with the specs 2.7 Ghz Intel Core-i5 MacbookPro.

VI. RESULTS

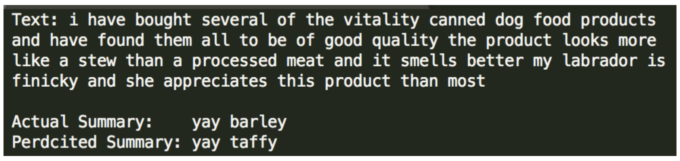
After the training, model is ready to perform individual predictions. Sample results can be seen below:



```
Text: this is a confection that has been around a few centuries it is
a light pillowy citrus gelatin with nuts in this case unk and it is
cut into tiny squares and then liberally coated powdered sugar and it
is a tiny mouthful of heaven not too chewy and very flavorful i highly
recommend this yummy treat

Actual Summary:    great just as good as the expensive brands
Predicted Summary: great taffy greasy food as the the brands
```

Figure 7- Prediction 1



```
Text: i have bought several of the vitality canned dog food products
and have found them all to be of good quality the product looks more
like a stew than a processed meat and it smells better my labrador is
finicky and she appreciates this product than most

Actual Summary:    yay barley
Perdcited Summary: yay taffy
```

Figure 8-Prediction 2

VII. EVALUATION

To evaluate the results there are two main score methods to use BLEU and ROGUE. The Bilingual Evaluation Understudy Score, or BLEU for short, is a metric for evaluating a generated sentence to a reference sentence. The score ranges between 0.0-1.0. The method Works by counting all the matching n-grams in the reference text to n-grams in the generated text. This matching does not consider word order. ROGUE is another scoring method that works the same way BLEU does. The difference between them is explained in this [14]:

"**Bleu measures precision**: how much the words (and/or n-grams) in the *machine generated summaries* appeared in the human reference summaries.
Rouge measures recall: how much the words (and/or n-grams) in the *human reference summaries*appeared in the machine generated summaries."

VIII. ETHICAL CONSIDERATION

The dataset used in this paper is a publicly available dataset that consists of over 500k user reviews on Amazon Fine Food services. The anonymity of the users is guaranteed by dropping the lines in the dataset that contains any information except than the actual review and its summary. The users that shared their reviews will not face any type of harm due to usage of their reviews in this work. This research is independent and does not share any harmful information with the outside world. No additional modifications, or biased summary generation is applied to prevent presenting mis-leading information.

IX. CONCLUSION

Thanks to this project a lot of new methodologies are used. The training process was really good on small amount of data, but the whole data we have is so huge and variant, compared to the training dataset, that the training we done on that small amount of data could not cover all the word relationships in the rest of the dataset. Besides having less computational power to train LSTM models with a complex structure was highly time consuming. The code for the model can be found in: https://github.com/recepinanc/coursework2/blob/master/main.py

X. REFERENCES

[1] E. Jobson and A. Gutiérrez, *Abstractive Text Summarization using Attentive Sequence-to-Sequence RNNs*. California, p. 8.

[2] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel and Y. Bengio, *END-TO-END ATTENTION-BASED LARGE VOCABULARY SPEECH RECOGNITION*. 2016.

[3] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney and K. Saenko, *Translating Videos to Natural Language Using Deep Recurrent Neural Networks*. Denver, Colorado, 2015.

[4] D. Bahdanau, K. Cho and Y. Bengio, *NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE*. 2016.

[5] R. Nallapati, B. Zhou, C. Santos, Ç. Gülçehre and B. Xiang, *Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond*. 2016.

[6] W. Zeng, W. Luo, S. Fidler and R. Urtasun, *EFFICIENT SUMMARIZATION WITH READ-AGAIN AND COPY MECHANISM*. 2017.

[7] Tensorflow, *Sequence-to-Sequence Models*.

[8] I. Sutskever, O. Vinyals and Q. Le, *Sequence to Sequence Learning with Neural Networks*. 2014, p. 1.

[9] T. Baumel, "Neural Attention Mechanism", *Talbaumel.github.io*, 2018. [Online]. Available: https://talbaumel.github.io/blog/attention/. [Accessed: 24- Apr- 2018].

[10] C. Wang, *RRA: Recurrent Residual Attention for Sequence Learning*. 2017

[11] B. Liu and I. Lane, *Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling*. San Francisco, 2016.

[12] J. Pennington, R. Socher and C. Manning, *GloVe: Global Vectors for Word Representation*. 2014.

[13] “How to decide the number of hidden layers and nodes in a hidden layer?”, ResearchGate.net, 2013. [Online]. Available: https://www.researchgate.net/post/How_to_decide_the_number_of_hidden_layers_and_nodes_in_a_hidden_layer