Extractive Text Summarization With a LSTM

Gururaj Desai[1], Laura Oberscheid[2]

ABSTRACT

Text summarizing can be used to automatically create short summaries of longer text. In the age of the internet, we are constantly bombarded with a lot of information where you can lose the overview quickly. Extractive and abstractive text summarization are of great help to come up with short summaries of the given text. The extractive approach focuses on the relevant sentences in the document according to some certain criteria. The abstractive approach tries to increase the coherence among the sentences through the elimination of redundancies and clarification of the content of the sentences. This paper describes the modelling of an LSTM for extractive text summarization.

1 Introduction

The aim of extractive text summarizing is according to Zhang et al. (2016), who will be cited for this paragraph, to generate a text summary by ranking the sentences in the text. Whereby the performance heavily relies on the over all quality of the sentence features. Due to extensive research in the natural language processing (NLP) area, handmade features for sentence representation are things of the past. By using deep learning methods in combination with pre-trained word embedding technique, impressive performances for different NLP tasks have been achieved in recent years. With the help of Deep Neural Networks one do not has to deal with intensive time consuming feature engineering because those will features will be learnt implicitly. Word embeddings are a representation of words with a dense vector. They help us especially to solve the problem course of dimension as well as sparsity conventional bag-of words method (Bengio et al., 2003; Mikolov et al., 2013). Having said so using a LSTM for extractive text summarization is solid approach, however recently transformer is producing even more promising results in the area of text summarization.

2 Related Work

According to Zhang et al. (2016), that is the underlying source for this paragraph, there has been great efforts during the last decades in the area of text summarization. Those achievements can be split into two categories, namely supervised and unsupervised methods. Under the supervised methods, we have supervised bigram-based integer linear programming or “Improving the Estimation of Word Importance for News Multi-Document Summarization” (Hong and Nenkova, 2014). Under the supervised methods we can list Latent Semantic Analysis (LSA), submodularity-based methods, Markov Random Walk (MRW), and graph-based methods such as LexRank.

Recently a lot of research went into the field of deep learning methods for summarizing documents. In the paper of PadmaPriya and Duraiswamy (2014) they used a restricted Boltzmann Machine (RBM) to generate generic summaries. However, the RBM only extracts in total four linguistic features of sentences, that are fed in the RBM and performs overall not very well. In contrast, the approach of Cao et al. (2015), to rank the sentences with the help of RNN leads to a better performance compared to traditional methods. Their idea is to transform the sentence ranking task to a regression problem. For the sentence representation, they use hand-crafted features and the RNN is building via a tree structure. This tree construction is said to be time-consuming, thus not optimal. The idea of Denil et al. (2014) was to summarize documents by using a hierarchical CNN. For this, they employed a CNN for extracting sentence features from the words and on top of this another CNN to obtain document features from the learned sentence features.

Or as Zhang et al. (2016) proposed, using a CNN architecture with a little bit of parameter tuning in order to achieve satisfactory results. For this proposed framework, the summarization task is translated into a sentence regression task and solved by a CNN model.

3 Data

3.1 Data description

The dataset used for the text summarization task is called "WikiHow". This knowledgebase consists of online articles describing procedural tasks about different topics with multiple methods or steps (Koupaee and Wang, 2018). From entertainment and food to hobbies and education, the topics covered are covering many different areas. The structure of the articles is always similar. Each article starts off with the title “How to...” followed by a short description of the full article. The next part of the article consists of dedicated steps with headlines for each step.

The original data set consists of 214294 rows and 3 columns: headline, title and text and can be downloaded at : https://ucsb.app.box.com/s/ap23l8gafpezf4tq3wapr6u8241zz358. The title is the overall title for the article, the headline is an introduction for each paragraph and the text is the overall text of the article.

3.2 Data Preprocessing

In order to be able to feed the data into our LSTM, several preprocessing steps needed to be conducted.
Starting with the removal of stop words, parsing the HTML tags, removing special characters as well as double-quotes. We then used the contraction mapping in order to get rid of contractions and then filtered only the text from the articles.

[1] Master of Science: Management & Data Science, Universitat Luneburg, E-mail: Gururaj.Desai@student-leuphana.de, Matriculation Number: 3037041
[2] Master of Science: Management & Data Science, Universitat Luneburg, E-mail: Laura.Oberscheid@student-leuphana.de, Matriculation Number: 3035564

After the preprocessing step we looked at the word length ratio of the title and the text. As our model should learn to make the long text shorter, we deleted those rows that, after the data preprocessing had a longer title then their corresponding text.

4 Model

RNN based models such as Seq2seq have performed significantly well in generating text compared to conventional methods like n-gram model or Markov chain based models. Seq2seq models have been successfully applied to different types of NLP tasks, such as machine translation, text summarization and speech recognition, etc. We have chosen a Seq2seq model to do extractive text summarization. The model was build using the Keras library. Our model is based on the state of the art Encoder-Decoder architecture with three layers in both Encoder and Decoder parts. In the Encoder, the first layer is the embedding layer, this embedding layer takes input tokens and converts them to word embeddings based on the pre-trained word embedding matrix provided by Keras. We have chosen 350 as the embedding vector dimension, meaning, each input word will be ultimately converted to a vector of size 350. After the embedding layer, 2 LSTM layers are used inside the Encoder. Similar to Encoder, our Decoder has embedding layer as the first layer followed by two LSTM dense layers. We have used time distributed layer at the end of the Decoder with the shape being the vocabulary size of our summaries, with this, our model outputs the vector of size equivalent to the vocabulary a size of our summary.

4.1 Optimization and Training

For optimization, we have used advanced gradient descent optimizer "Adam" and trained our model with "Adam" as the loss function. We decided to take "Sparse Categorical Cross Entropy" instead of "Categorical Cross Entropy" because of faster training. "Sparse Categorical Cross Entropy" does not calculate the dot product of the true and predicted labels but instead directly takes the predicted labels probabilities according to the index of the true labels. Therefore it reduces the number of operations while calculating the loss and hence it increases training performance. To tackle overfitting, we have used the famous "Early stopping" mechanism. Early stopping stops the model training when the accuracy of the model starts to decrease and thus loss starts to increase. In most cases deep neural networks are used for large data sets. If the training stops because of extraordinary circumstances, then the trained weights will get lost. To save the trained weights after each epoch, we have used the "ModelCheckpoint" functionality given by Keras.

4.2 Prediction

Predicting texts from the trained model is not straight forward. First, we reconstructed Encoder and Decoder with the trained hidden states. For word predicting, we feed text to get encoded into the Encoder model. The encoded text will be inserted to the Decoder to predict the summary. To start predicting the summary, we need to pass over the context vector from Encoder to the Decoder along with "START" token. Decoder will predict the Softmax probabilities for all the words in the vocabulary at each time step. The predicted word will be chosen with the word having a maximum probability. This predicted word will be an input to the Decoder in the next time step to predict the next word. The process will continue until the Decoder will predict the "END" token. All these predicted words will be concatenated with space(" ") to finalize the summary.

4.3 Training Environment

We have trained our model with 90% of our data and tested the model performance with 10% of the data. So overall we had 200k for training and 20k for testing. For our training, we have used Linux OS having 256GB RAM, 2 CPU, and 14 cores /CPU capacity. Our model is trained with 32 epochs overall with a batch size of 256. We have used Keras 2.3.1 and Tensorflow 1.15.0 library for our project.

5 Evaluation

5.1 Evaluation Metrics

As described in PadmaPriya and Duraiswamy (2014) there are three basic evaluation criteria in the area of text summarization.

5.2 Recall

According to PadmaPriya and Duraiswamy (2014) recall is the overall number of the retrieved sentences to the number of the relevant sentences. This measure is used in order to measure the reliability of a text summarization method.

5.2.1 ROUGE

Rouge-N is an n-gram recall between a predicted summary and actual labeled summary (Lin and Hovy, 2002). Rouge-N can be formalized in the following way,

$$R_n = \frac{\sum_{S \,\epsilon\, \{RF\}} \sum_{gram_n \,\epsilon\, S} Count_{match}(gram_n)}{\sum_{S \,\epsilon\, \{RF\}} \sum_{gram_n \,\epsilon\, S} Count\,(gram_n)} \tag{1}$$

Where n stands for the length of the n-gram, RF means reference summaries. $Count_{match}(gram_n)$ is the maximum number of n-grams co-occurring in a candidate summary and a set of reference summaries. For our evaluation, we have used the ROUGE-1 score. In general ROUGE-N gives us the recall of the predictions. We have used py-rouge python library to compute the rouge scores.py-rouge library provides us the precision, recall, and f1-measure of the rouge scores. py-rouge calculates these measures in the following way,

$$Pn = \frac{OC}{EC} \tag{2}$$

$$Rn = \frac{OC}{RC} \tag{3}$$

Here, OC (Overlapping Count) means a number of n-grams which are present in the predicted summary, EC (Evaluated Count) is used to get the count of n-grams present in predicted summary. Similarly, RC (Reference Count) is the count of n-grams present in the actual labeled summary. Using these, we can calculate P_n ($Precision_{ngram}$) and R_n ($Recall_{ngram}$).

5.3 Precision

Precision is the fraction of retrieved sentences that are relevant in the document (PadmaPriya and Duraiswamy, 2014).

5.3.1 BLEU

The motive behind the development of BLEU is that the human evaluation of predicted summaries is more expensive and time-consuming when it comes to evaluating the quality of the predictions. BLEU is a precision-based measure with ranging values from 0 to 1, with a higher score meaning good match between predicted and actual summary. Although it is a good measure to find the quality (precision) of the predictions, when it comes to text summarization, it fails to evaluate the context between predicted summary and the input text. However, BLEU is out of scope for this paper.

5.4 F-Measure

With the help of the precision and recall values, the F-measure can be expressed as a balance between precision and recall for the whole dataset (PadmaPriya and Duraiswamy, 2014).

$$F1 = \frac{Pn * Rn}{(1 - \alpha * Pn) + \alpha * Rn} \tag{4}$$

F1 measure can be calculated by taking both precision and recall in the above-mentioned manner. Here α is used to give weight to the Pn($Precision_{ngram}$) and Rn($Recall_{ngram}$).

5.5 Baseline

As recommended in the work of Eskici and Perez (2017) a naive approach for the baseline, based on word frequencies will be applied.
The prepossessing step for the baseline is almost identically to the one for the model, except the deletion of the punctuation. After the prepossessing steps with the help of Malik (2018), the number of occurrences for each word and each article are counted. Next, we used a mapping of words to frequencies in order to get the sentence scores. It is the sum of scores of all occurring words in a sentence, normalized by the individual length of the sentence. We then use the sentence with the highest score as a summary of the article. From a logical perspective, we only looked at the largest sentence score of sentences with fewer than 5 words. This constrains was used as most of the titles we want to predict were around 5 words.

The two tables below illustrate the performance of the applied LSTM. Although the model does not perform overall extraordinary well in percentage, some of the predicted sentences do still make sense from a human perspective.

Table 1. Evaluation scores

	Precision	Recall	F1-measure
Baseline	0.0630	0.0676	0.0628
LSTM model	0.0158	0.0108	0.0124

Table 2. Preselected results

Actual Labelled Summaries	LSTM Predictions
horse breeding farm	raise horses
online business	find things sell
computer programming	compile program linux
computer safe mode	mount iso image
dog training	teach dog

5.6 Limitations

We have used unidirectional LSTM for our Seq2seq model, using this we are preserving the information of the past and leaving out the information of the future importance of the words in sequence. Using Bidirectional LSTM, we can have past and future information preserved while training the model. Bidirectional LSTM trains the model with 2 hidden states to store the context of past seen words and future upcoming words. A Encoder-Decoder Seq2seq model processes each word in the input sample sequentially and produces one hidden state vector and this hidden state vector will be given to Decoder.With this approach, we won't be able to learn what all input words are actually relevant to predict the output word of the whole output sentence. Attention mechanism can be used to map each output word to all the input words and learn the importance of the input words to predict the output word. We have used Seq2seq model without Attention, but the Attention mechanism helps us to improve the model and learn the context better to predict the summaries in our case. The Attention mechanisms was not added right from the beginning, as we first tried to come up with a less complex model.

While training our LSTM model we realized that training a deep neural network always takes a lot of computational time and resources. Also, when it comes to language models, with an increasing number of model parameters makes it even harder to train the model with a limited amount of time and resources. When we look into our standard LSTM Encoder-Decoder architecture, Seq2seq models traverse sequentially through all the words of the given input sequence to compute the hidden states. These hidden states are used to compute the output sequence sequentially. The state of the art Transformer solves the issue of computational time when it comes to training the language models. Transformer architecture use Attention-based Encoder-Decoder architecture and get rid of the RNN completely (Vaswani et al., 2017). Hence the parallelization of the Transformer helps us reduce the training time. However as we first trained our LSTM model, changing everything to a Transformer and training this model from scratch would have taken too much time.

6 Discussion and Conclusion

In the paper at hand, we introduced the implementation of Seq2seq model based on a state of the art Encoder-Decoder architecture to produce extractive text summarization. Our model is able to summarize large text into a short summary. We used a publicly available data set and evaluated our model with commonly applied metrics such as ROUGE score.
The model was unfortunately not able to beat the baseline due to our constrains. If we would attempt text

summarization again we would go for a Transformer architecture right from the beginning. Additionally, as mentioned in the limitation part, we would train the model much larger in order to improve our accuracy.
The both authors contributed with equal share to this project and extended their knowledge about text summarization using LSTM. This project helped the authors to deepen their understanding of applying Deep Neural Networks at large scale as well as for real-world problem settings.

References

Bengio, Y., Ducharme, R., Vincent, P. and Jauvin, C. (2003). A neural probabilistic language model, *Journal of machine learning research* **3**(Feb): 1137–1155.

Cao, Z., Wei, F., Dong, L., Li, S. and Zhou, M. (2015). Ranking with recursive neural networks and its application to multi-document summarization, *Twenty-ninth AAAI conference on artificial intelligence*.

Denil, M., Demiraj, A., Kalchbrenner, N., Blunsom, P. and de Freitas, N. (2014). Modelling, visualising and summarising documents with a single convolutional neural network, *arXiv preprint arXiv:1406.3830* .

Eskici, K. and Perez, L. A. (2017). Multi-document text summarization.

Hong, K. and Nenkova, A. (2014). Improving the estimation of word importance for news multi-document summarization, *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, Association for Computational Linguistics, Gothenburg, Sweden, pp. 712–721.
URL: *https://www.aclweb.org/anthology/E14-1075*

Koupaee, M. and Wang, W. Y. (2018). Wikihow: A large scale text summarization dataset.

Lin, C.-Y. and Hovy, E. (2002). Manual and automatic evaluation of summaries, *Proceedings of the ACL-02 Workshop on Automatic Summarization-Volume 4*, Association for Computational Linguistics, pp. 45–51.

Malik, U. (2018). Text summarization with nltk in python.
URL: *https://stackabuse.com/text-summarization-with-nltk-in-python/*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. (2013). Distributed representations of words and phrases and their compositionality, *Advances in neural information processing systems*, pp. 3111–3119.

PadmaPriya, G. and Duraiswamy, K. (2014). An approach for text summarization using deep learning algorithm, *JCS* **10**.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I. (2017). Attention is all you need, *Advances in neural information processing systems*, pp. 5998–6008.

Zhang, Y., Er, M. J. and Pratama, M. (2016). Extractive document summarization based on convolutional neural networks, *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society* pp. 918–922.