tl;dr – Abstractive Text Summarization Using Sequence to Sequence Models and an N-Gram Language Model

Sanjay Dorairaj, UC Berkeley, dorairajsanjay@berkeley.edu

Code: https://github.com/dorairajsanjay/tldr
Website: http://tldr.ddnsfree.com/
The **tl;dr** version. Click here for the long version :-)

Abstract

This paper explores the use of sequence to sequence networks [13] for abstractive text summarization tasks. The model overlays an attention mechanism and a beam search decoder on top of a vanilla Seq2Seq network model. Since Seq2Seq models are prone to grammatical errors and repetition, an n-gram based language model is then used to validate top beam search results and pick the best summary. The baseline model consists of a vanilla Seq2Seq model with attention and a greedy search based decoder. The results show reasonable performance for the baseline and demonstrate the working of the language model as a noisy channel validation model. There were some issues overlaying a Beam Search decoder on top of the Bahdanau attention mechanism due to potential implementation issues. It is possible that with more training time and/or tweaks to the code, the beam search model performance complemented with the n-gram based language model will outperform the greedy search approach as expected. Overall however, this paper and project were immensely valuable in enhancing my understanding of Sequence to Sequence networks mechanism for text summarization and the Tensorflow development framework.

Introduction

The success of Seq2Seq networks is especially true in the area of text classification, machine translation (NMT) and abstractive summarization tasks (ABS), outsmarting traditional approaches and becoming the norm for NLP tasks [1], [2]. Their success has been vastly aided by exponential improvements in compute power and the availability of large volumes of data, which are both prerequisites for the success of neural network models. Seq2Seq models have progressed from the lowly recurrent neural network model, to LSTMs and then on to sequence to sequence models. This progression while entertaining at first [4], was not altogether useful.

The introduction of attention-based models[2], pointer-generation networks [1] and the corresponding improvements in accuracy have however transformed these models into the power-horses of current NLP/NLU classification, translation, conversational and summarization systems. Performance of Seq2Seq models have not remained consistent across all domains; some, such as NMT have enjoyed more success when compared to domains such as text summarization. A key distinction between domains that have been relatively more successful when compared to ABS is that most other models have a more or less proportionate size of input and target data. ABS however maps a large set of input features to a limited set of output features, placing it in a veritable hard spot that is more challenging for Seq2Seq models to tackle. Nevertheless, progress in the area of ABS is still relatively faster than other state of the art methods, and is bound to improve, given the growing interest in this field and the acceleration of new ideas and technologies in the area of neural network technologies.

This paper attempts to build on the success of state of the art text summarization techniques using neural networks to build a text summarization model.

Background

This section documents some of the underlying techniques used in this paper and the proposed model.

Sequence to Sequence Models

A sequence to sequence network [4] is also referred to an encoder-decoder model. This model allows us to generate a sequence of text output corresponding to a given input sequence. The model typically uses an enhanced RNN cell, such

as a bi-LSTM in a multi-stacked mode, in order to process input text. A key advantage of RNNs relative to other neural networks is that they allow the model to factor in the order in which words appear in the input text. This is distinctly different from a bag of words model, that does not pay any attention to the order of words in a sequence of text. LSTMs address the issue of vanishing or exploding gradients in vanilla RNNs. The bi-directional LSTM allows the model to leverage both the future and the past to make predictions. The last hidden state of the encoder network, also called the **thought vector**, holds knowledge of the entire input sentence. This
is fed into another LSTM network, known as the decoder network. The decoder network takes as input the thought vector and the previous output word and generates the next word. The decoder stops generating words when it generates a special word that denotes the end of the output sequence.

Attention Models and Beam Search

The problem with vanilla sequence to sequence networks in the context of text summarization is that they have been shown to generate inaccurate details of the underlying input text [1]. The reason for this has been largely attributed to the model's inability to compress all information about the piece of input text into a single hidden vector i.e. even LSTMs tend to forget critical information if they need to process large amount of text.

Attention-based models attempt to fix this issue by allowing the decoder to glimpse the details of encoder processing and factor this information into output generation.

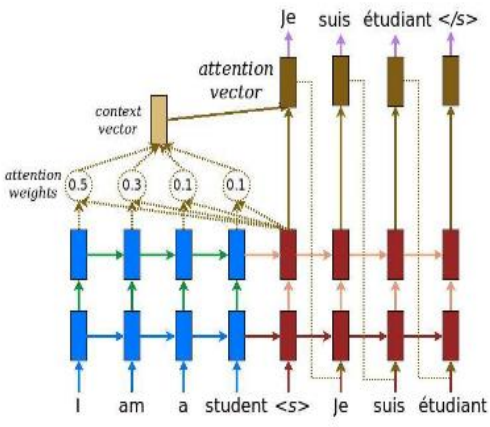


Figure: Sequence to sequence model with attention [14]

The Beam Search algorithm allows us to generate output summaries whose total joint likelihood is higher than that of any other text sequence. The alternative to this approach is the greedy search algorithm, where at each time step we select the decoder output with the highest probability. Conditioning the next output only on the previous output may result in an output sequence whose joint likelihood is lower than another sequence that was unexplored since we failed to pursue other possibilities. Beam search tries different combinations of partial output sentences, each time picking a set of partial output sentences with the highest likelihood. The process repeats until we generate the full sentence. At this point, we repeat our conditional probability computation one last time and pick the winning sentence.

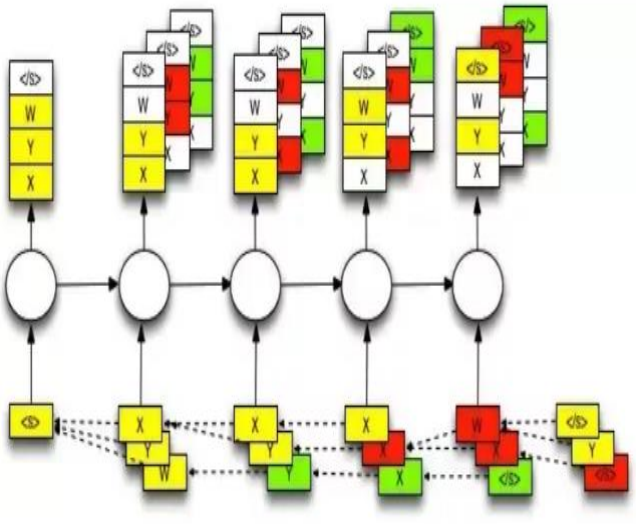


Figure: Beam Search algorithm [15]

Performance of state of the art text summarization models

The most successful model thus far in abstractive text summarization is based on work done by Abigail See et al [1]. In this paper, the authors use a sequence to sequence model with attention and add to it two new features - **pointer generation networks** and the **coverage mechanism**. Pointer generation allows the model to borrow words from the input text in the event the summary vocabulary is insufficient to represent the input text i.e. too many out of vocabulary words or unknowns. The coverage mechanism ensures that the model applies the attention mechanism to all parts of the input text and penalizes the model otherwise. This helps avoid another major drawback with prior summarization techniques namely repetition.

Shown below are the ROUGE and METEOR scores of the method cited above (in bold) relative to that of other popular methods [1]

	ROUGE			METEOR	
	1	2	L	exact match	+ stem/syn/para
abstractive model (Nallapati et al., 2016)*	35.46	13.30	32.65	-	-
seq-to-seq + attn baseline (150k vocab)	30.49	11.17	28.08	11.65	12.86
seq-to-seq + attn baseline (50k vocab)	31.33	11.81	28.83	12.03	13.20
pointer-generator	36.44	15.66	33.42	15.35	16.65
pointer-generator + coverage	**39.53**	**17.28**	**36.38**	17.32	18.72
lead-3 baseline (ours)	40.34	17.70	36.57	20.48	22.21
lead-3 baseline (Nallapati et al., 2017)*	39.2	15.7	35.5	-	-
extractive model (Nallapati et al., 2017)*	39.6	16.2	35.3	-	-

Original Text (truncated): lagos, nigeria (cnn) a day after winning nigeria's presidency, *muhammadu buhari* told cnn's christiane amanpour that he plans to aggressively fight corruption that has long plagued nigeria and go after the root of the nation's unrest. *buhari* said he'll "rapidly give attention" to curbing violence in the northeast part of nigeria, where the terrorist group boko haram operates. by cooperating with neighboring nations chad, cameroon and niger, he said his administration is confident it will be able to thwart criminals and others contributing to nigeria's instability. for the first time in nigeria's history, the opposition defeated the ruling party in democratic elections. *buhari* defeated incumbent goodluck jonathan by about 2 million votes, according to nigeria's independent national electoral commission. the win comes after a long history of military rule, coups and botched attempts at democracy in africa's most populous nation.

Baseline Seq2Seq + Attention: UNK UNK says his administration is confident it will be able to destabilize nigeria's economy. UNK says his administration is confident it will be able to thwart criminals and other nigerians. he says the country has long nigeria and nigeria's economy.

Pointer-Gen: *muhammadu buhari* says he plans to aggressively fight corruption in the northeast part of nigeria. he says he'll "rapidly give attention" to curbing violence in the northeast part of nigeria. he says his administration is confident it will be able to thwart criminals.

Pointer-Gen + Coverage: *muhammadu buhari* says he plans to aggressively fight corruption that has long plagued nigeria. he says his administration is confident it will be able to thwart criminals. the win comes after a long history of military rule, coups and botched attempts at democracy in africa's most populous nation.

Development Environment

Key features of the development environment are listed below

1. Cloud environment - AWS EC2
2. AMI - Deep Learning AMI (Ubuntu)
3. Instance Type - p2.8xlarge
4. Tensorflow 1.6.0,Python 3.6.3

Dataset

The dataset chosen for this model is the CNN dataset [5]. This dataset contains a large corpus of around 90,000 CNN news articles. Each article contains one or more highlights. This project uses the first highlight as the **target or summary** data**.** The CNN dataset, along with the Daily Mail dataset, was originally released to help accelerate the pace of work in automatic question and answering. This dataset has however also found to be very useful in building models for text summarization. The pointer generation model by Abigail See et al [1], also uses this dataset for validation. The dataset is freely available and instructions to process and use this dataset in a format fit for text summarization are readily available online.

The data, which consists of story-summary pairs from the CNN dataset [5] is divided into three sets - **training**, **validation** and the **hold out test data**. The original data exists in a zip file format. The zip file consists of multiple individual **.story** files, with each story file containing an article and multiple highlights. Each highlight is preceded with the **@highlight** keyword.

The code base [6] contains a command line parameter to extract the highlights and stories, clean them, generate separate vocabularies for stories and highlights and store them in a pickle file for later use by the model.

After data processing and cleanup, the following was the distribution of counts of the various data artifacts.

- Training data - 83217
- Validation data - 4623
- Summary Vocabulary - 40,000
- Story Vocabulary - 40,000

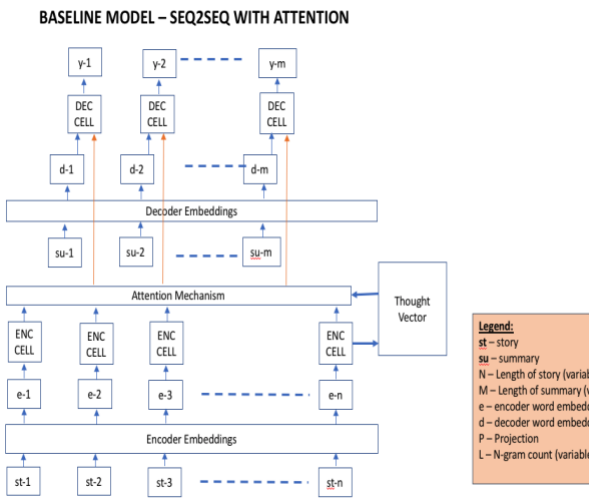
Next, the Brown corpus was used to build a bigram language model that was used to validate output summaries from the sequence to sequence network. The Brown corpus and bigrams and their probability distributions are readily available as part of the NLTK package [7]

Methods

This project uses two models to drive the process of text summarization. The **first model** is a sequence to sequence model with attention and beam search. The **second model** picks top beam search results and runs them through an n-gram based language model to determine the summary that best satisfies the n-gram language model.

Baseline Model

The baseline model is a sequence to sequence model with attention. The attention mechanism used is Bahdanau's attention. Inference is done using the greedy search algorithm.



In the baseline model, we create a vanilla sequence to sequence network and overlay the network with the Bahdanau attention mechanism. The **attention mechanism** allows the decoder to peek into all of the encoder outputs and create an attention vector composed of a weighted sum of encoder output states. This set of all encoder hidden output units is referred to as the attention memory vector. The model learns to predict the underlying hidden units in the encoder that are more correlated with the next decoder input. This method has been shown to improve the quality of predictions in a sequence to sequence network [2][4].

Decoding is done using the greedy search algorithm, where we pick the softmax output with the highest probability score in each time step of the decoding process until we see the end of sentence token or we have exceeded the configured max output time steps.

Proposed Model

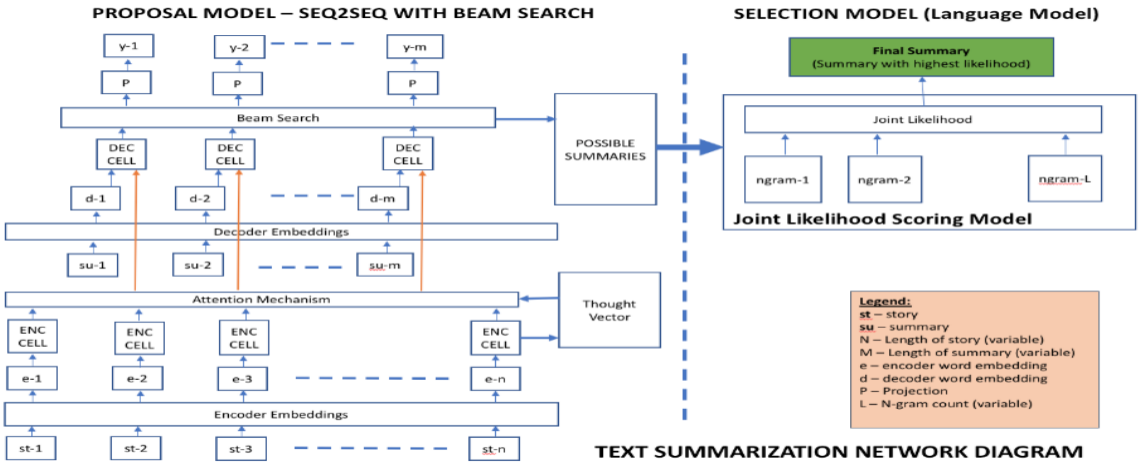
The proposed model depicted below, builds on top of the baseline model by adding beam search

Error!

decoder layer at inference time **and** an additional n-gram language model to further validate results from the beam search decoder.

In the proposed model, the beam search algorithm is used to yield multiple output summaries based on the joint probability score of words emitted at each time step. This method ensures that we are not only looking at the output at each time step but also looking at the total cumulative probability score of words emitted at all time steps. The length of the beam is typically configured to be in the range of 3 to 10. Beam width is truncated to ensure that the search space does not get excessively large as we progress through each time step.

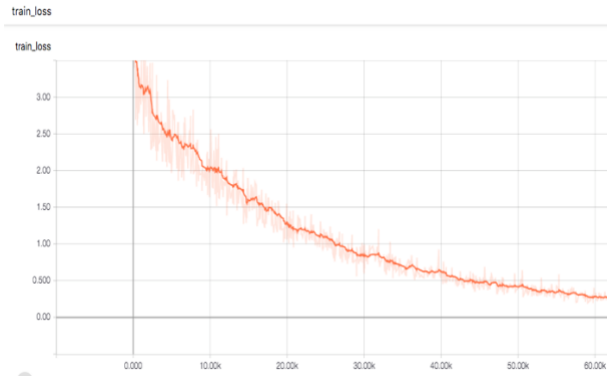
After beam search is completed, the top N beam search results are taken and run through an n-gram based language model to pick the summary whose joint n-gram likelihood is maximum. For the purpose of the project, a **bigram language model** based on the **Brown corpus** was used for validation.



Training

The training step involved feeding batches of story/summary pairs from the training dataset into a sequence to sequence network. Word embeddings for the network were learnt during the training phase. The models were trained for over 50 epochs. The Adam optimizer is used for updating the gradient at the end of each training step.

Here is the status of decreasing cross-entropy loss through the various iterations



In all, two models were trained, **the baseline model** and **the proposed model.** The original implementation of the baseline model was not performing well and was generating a lot of repeating words. To avoid this, a new model was created that directly reused the decoder cell from training for inference instead of creating a new decoder cell for inference and initializing it with the the output of the encoder. This latter approach appeared to work much better, even better than the proposed model, and was used for much of the evaluation work.

Inference

Inference was performed by feeding in pairs of story/summary data from from the test set. Two inference engines were built for this project. The first inference engine ran during training and would continuously evaluate the performance of the model on the test dataset. Another standalone inference engine was built which would load the saved model and process individual stories and generate summaries. The second inference engine was meant to depict a real-world deployment of this feature and also to support the web-based interface that is also released as a part of this project [6]

The baseline model(s) used the greedy search algorithm for inference. The proposed model used the Beam Search decoder and the n-gram language model for inference.

The output of the inference phase, the new summary, was stored along with the original summary in a separate folder in order allow for further computation of the ROUGE metric.

Evaluation Metrics

The preferred evaluation metric for text summarization tasks is the ROUGE score[8]. This scores the model based on its ability to recall words from the original summary. The evaluation of the ROUGE score was done using the ROUGE package [9], which can be easily installed using the Python package manager (PIP). Once installed, the hypothesis and reference summaries are copied into separate files and the evaluation is done using the **rouge** command from the command line.

Results and Discussion

The results for this model were not exactly what I had hoped to collect. The baseline model performed relatively better than the proposed model. The primary reason was that the proposed model was generating too many repeating sets of words even after 50 epochs of training, whereas the baseline model was able to generate more meaningful sentences faster. The proposed model therefore had a much lower ROUGE score compared to the baseline model.

The table below shows results of summarization with the **baseline model**. **ROUGE scores for the proposed model are excluded since they are below the 0.01 threshold**.

Parameter	Training set	Test set
Rouge-1	**"r": 0.048, "f": 0.049, "p": 0.051**	**"r": 0.074, "p": 0.082, "f": 0.077**
Rouge-l	**"r": 0.074, "f": 0.078, "p": 0.084**	**"r": 0.049, "p": 0.049, "f": 0.049**
Rouge-2	**"r": 0.01, "f": 0.01, "p": 0.01**	**"r": 0.011, "p": 0.013, "f": 0.012**

Discussion

As shown above, both the baseline model and proposed model appeared to do well on the training data. However, results on the test data were very different. The baseline model outperformed the proposed model. One possible reason for this performance difference could be a possible bug in the tensorflow graph for the proposed model or perhaps the proposed model requires a lot more training time in order to start generating meaningful results . There are a lot of discussions on configuration of the Tensorflow BeamSearchDecoder component and the model appears to have followed the instructions to the letter, however, it is still possible that an element of the tensorflow graph is not connected right. The reason I suspect this as an issue is because when running the inference engine on training data to troubleshoot this problem, the proposed model still generates repeating words, whereas the training decoder does not appear to have this issue and in fact appears to overfit to the training data.

There are several other reasons that could explain the poor performance. These fall in two classes - dataset issues and hyper-parameter issues.

Dataset issues: Training of a summarization task is akin to training a model to find a needle in a haystack. This is because the model needs to go through a large amount of text to find just the right set of words that would represent that text. Input and output text use different vocabularies and the test data may contain several out of vocabulary (OOV) words or incoherent words resulting that make the output sentence appear garbled. The pointer generation mechanism may help in resolving some of this ambiguity.

Hyperparameter issues: The model used here is a single-layer LSTM network, with a limited configuration, shown below. Most of the successful models appear to have been built on bidirectional LSTMs with multiple layers and larger hidden unit sizes.

Future Improvements

Future improvements to this work will include

1. Addressing the root cause behind the poor performance of the proposed model.
2. Scale hyperparameters - more layers, bi-directional LSTM and more hidden units.
3. Switch to Gigaword Corpus from CNN dataset. The Gigaword corpus has more training examples and has also shown good results [10][11]
4. Include support for pointer generation networks to help with out of vocabulary words and use coverage mechanism to avoid repetition [1].

Conclusion

This project was a very useful exercise in exploring the development of text summarization models using sequence to sequence networks and Tensorflow. It was fascinating to watch the model build more or less fluent summaries both during training and during inference (greedy search inference). The proposed model with added beam search and language model validation support, should yield superior performance compared to the baseline model and it will interesting to wrap up work on the proposed model and get a sense of the final results. The neural network model could also be paired with other models, such as a parts of speech model or other forms of syntactic or semantic parsers to yield more likely summaries, although, there is always the possibility of sacrificing important keywords in exchange for better fluency.

References

1. Abigail See, Peter J. Liu, Christopher D Manning. Get To The Point: Summarization with Pointer-Generator Networks. https://arxiv.org/pdf/1704.04368.pdf
2. Minh-Thang Luong, Hieu Pham, Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. https://nlp.stanford.edu/pubs/emnlp15_attn.pdf
3. Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/
4. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. https://arxiv.org/abs/1409.0473
5. Herman et al. CNN Daily Mail Dataset, https://cs.nyu.edu/~kcho/DMQA/
6. Project code base - https://github.com/dorairajsanjay/tldr
7. Natural Language Toolkit - https://www.nltk.org/
8. Microsoft. ROUGE: A Package for Automatic Evaluation of Summaries, https://www.microsoft.com/en-us/research/publication/rouge-a-package-for-automatic-evaluation-of-summaries/
9. Rouge Python package, https://pypi.org/project/rouge/0.2.1/
10. Alexander M Rush, Sumit Chopra, Jason Weston. A Neural Attention Model for Abstractive Sentence Summarization. https://arxiv.org/abs/1509.00685
11. Xin Pan Peter Liu. Tensorflow Text Summarization. https://github.com/tensorflow/models/tree/master/research/textsum
12. Larry Page. Google Page Rank, Page, http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf
13. Ilya Sutskever, Oriol Vinyals, Quoc V.Le. Sequence to Sequence Learning with Neural Networks. https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf
14. *Image Source*. Sequence to Sequence networks, https://www.tensorflow.org/tutorials/seq2seq
15. *Image Source*. Beam Search algorithm. Source: https://github.com/mfederico and https://www.quora.com/Why-is-beam-search-required-in-sequence-to-sequence-transduction-using-recurrent-neural-networks