Text summarization using machine learning

DAT450 Machine learning for natural language processing
Independent Project

Ahmed Groshar
Gustav Molander
Noa Onoszko

March 4, 2021

Abstract

Automatic text summarization is the task of producing a concise version of a text, preserving the underlying principal information. There are two main ways to do this: Either by generating text from scratch, or by picking sentences from the article. In this report, we present two different methods for text summarization, using the latter approach in a supervised setting. The first method mimics the model presented in a research paper, using recurrent neural networks in a reinforcement learning setting as a measure to optimize a domain specific evaluation metric, the ROUGE score. This model is trained on the CNN/DailyMail dataset with ground truth summaries written by journalists. The second model is a simple recurrent neural network. This model was trained on articles from the Cornell Newsroom with ground truth summaries consisting of sentences from the article. The reinforcement learning model did generalize, likely due to a bug. The simple model however showed good results, outperforming the strong LEAD-3 baseline. This success can partly be credited to the fact that the ground truth summaries contain sentences from the article.

Introduction

Text summarization is the task of shortening a piece of text while preserving a coherent and fluent summary that is representative and containing the main points of the original text. Automatic text summarization is a common problem in natural language processing. Applications of text summarization solutions include tools that aid users to navigate and digest web content, summaries that help researchers to read summarized information of articles that help them to find relevant articles quicker. Text summarization has become ever more successful due to modern approaches to document summarization that are data-driven and take advantage of neural network architectures and their ability to learn features without requirements of special linguistic prepossessing or linguistic annotations. Currently, two types of text summarization approaches are prevalent - abstractive and extractive. Abstractive summarization contains various text rewriting operations that may be performed, these include substitution, deletion, reordering and possibly others. This type resembles the type of summarization that is closer to the way humans form summaries of text. In contrast, extracive summarization is simpler in the sense that it does only have to identify and concatenate a number of sentences that are already present in the document and only use them to form the final summary. This type of summarization may appear very simple but it may lead to better performance scores on some datasets than most or all abstractive methods in certain applications.

We will implement two different models for extractive summarization: the RL summarizer, using reinforcement learning, and the Stacked LSTM summarizer using an LSTM. We will use the CNN/DailyMail dataset for the first one, which comes in the form of articles and human produced abstractive summaries as the reference summaries. We will only use the CNN articles. When training the second model the Cornell Newsroom dataset is used where only extractive summaries will be used.

We will use the LEAD-N baseline to evaluate the performance of our models. LEAD-N takes the first N sentences from the article and we will mainly use LEAD-3 which is common practice. It is important to note that this baseline is quite strong; State-the-art models only performs slightly better. It is also interesting to note that news articles contain a high degree of summary content in the first few sentences. This will presumably make the LEAD-3 baseline even stronger on our data.

Evaluation metrics are difficult and imperfect to design, however they are encompassing enough for most applications. The first metric is ROUGE-1 (**R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation) abbreviated R1, which refers to the overlap of the unigrams (singe words) between the predicted summary and reference summary. The second one is ROUGE-2, abbreviated R2, which refers to the overlap of bigrams (two words) between the predicted summary and reference summary. It is defined by the number of times a specified n-gram string from the reference summary occurs in the predicted summary, divided by the number of n-grams in the reference summary. The third one is ROUGE-L, abbreviated as RL, and measures the longest common subsequence between the model and reference summary. This metric is imperative to the fluency of the summary that a model produces. The landscape of fundamental types of architectures and performance metrics provide for very diverse performance and difficulty of comparison when the datasets are different. Furthermore, the quality of the summary could be made more reliable by taking account to the preferences of human readers.

Models

RL Summarizer

This model is based on the model REFRESH (**RE**in**F**o**R**cement Learning-based **E**xtractive **S**ummarization) that was proposed in the 2018 paper *Ranking Sentences for Extractive Summarization with Reinforcement Learning* by Narayan et al. [3]. We tried to mimic as much as possible but some things differ. Narayan et al. propose a supervised model for extractive text summarization based on reinforcement learning. Figure 1 illustrates the model schematically. The model consists of three main parts:

- a sentence encoder,
- a document encoder and

- a sentence extractor.

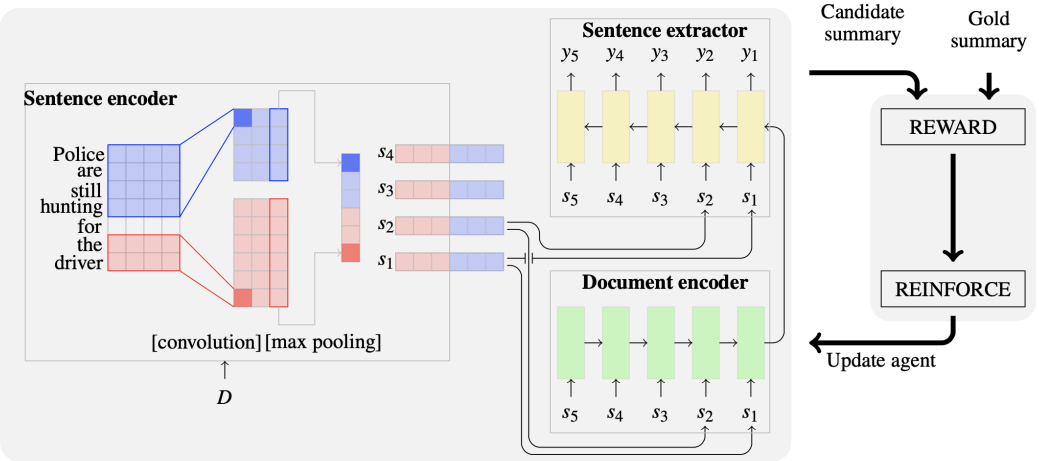


Figure 1: The model proposed by Narayan et al..

In words, the model works in the following way. An article, tokenized into sentences, is fed into the sentence encoder. The output is a sentence embedding for each sentence, which is fed into the document encoder that produces a document embedding. The document embedding is fed into the sentence extractor as an initial hidden state and the embedding for each sentence from the sentence encoder is fed into the sentence extractor as the input sequence. The result is a vector with one element per sentence in the article which is a ranking of sentences. A set of candidate summaries are created based on this ranking. The candidate summaries are compared to the target, the gold standard summary, and a reward is calculated for each summary. A summary is selected based on the rewards. The loss is calculated for that summary according to the REINFORCE algorithm, which uses the outputs of the sentence extractor and the reward, and is used to update the weights. In the following section, we will describe each part of the model in detail.

Sentence Encoder

REFRESH uses a temporal convolutional network to fine-tune the sentence embeddings, using word2vec as initialization. We use a pre-trained BERT transformer model, with an embedding size of 768, for simplicity.

Document Encoder

The functionality of a document encoder is the same as of a sentence encoder: In goes words and out comes a vector of real numbers. The only difference is that a document encoder encodes a set of one or more sentences. Our document encoder consists of a recurrent neural network, more specifically, a single, unidirectional long short-term memory layer with hidden size 600. It takes sequence embeddings for each sentence in the article as input. The initial hidden state and initial cell state are tensors of zeros. The LSTM outputs a tensor of outputs for each sentence as well as a hidden state tensor and a cell state tensor for the last sentence. We use the hidden state tensor as the document embedding, which consists of 600 real numbers, and discard the other outputs.

Sentence Extractor

The sentence extractor is the part of the model that ranks sentences with regard to relevance to the summary. It consists of an LSTM and a linear layer, used in this order. The LSTM layer is similar to the LSTM in the document encoder. The input sequence is the sentence embeddings, the hidden size is 600 and the initial cell state is a tensor of zeros. There are however two differences. The first one is that the output of the document encoder is used as

initial hidden state. The second one is that instead of using the last hidden states, we now use the output of each sentence in the input sequence and discard the hidden states and cell states. This output is fed to the linear layer. The output size of the linear layer is the number of sentences in the article. The softmax function is applied to the outputs, which acts as normalization so that we can more easily interpret the output.

Defining the Loss

This is where the reinforcement learning comes into play. The summary selection can be seen as a Markov Decision Process, where an agent picks a summary in a static environment and gets a reward that depends on the choice of summary. We define the reward $r(\hat{y})$ for choosing summary $\hat{y}$ as the average of the F1 scores of ROUGE-1, ROUGE-2 and ROUGE-L, which we will simply refer to as the ROUGE score in the future. We then define the loss according to the REINFORCE algorithm as the negative expected reward

$$L(\theta) = -\mathbb{E}_{\hat{y}\sim p_\theta}[r(\hat{y})],$$

where $p_\theta = p(y|D,\theta)$ is the probability to pick summary y from article D with network parameters θ. The expected gradient of a non-differentiable reward, which our reward is, can be expressed as

$$\nabla L(\theta) = -\mathbb{E}_{\hat{y}\sim p_\theta}[r(\hat{y})\nabla \log p(\hat{y} \mid D,\theta)].$$

Calculating the above expectation is infeasible in a training setting due to the vast number of possible summaries. Instead, we approximate the expectated gradient by a single sample $\hat{y} \sim p_\theta$. We then get

$$\begin{aligned}\nabla L(\theta) &\approx -r(\hat{y})\nabla \log p(\hat{y} \mid D,\theta) \\ &\approx -r(\hat{y}) \sum_{i=1}^{n} \nabla \log p\left(\hat{y}_i \mid s_i, D, \theta\right),\end{aligned}$$

where we sum over sentences s_i in the input article D. In practice, we define the loss as

$$-r(\hat{y}) \sum_{i=1}^{n} \log p\left(\hat{y}_i \mid s_i, D, \theta\right)$$

and let PyTorch compute the gradients with `backward`. As a practical detail, we use log softmax instead of softmax in the output layer of the sentence extractor for numerical stability, and skip the log in the above formula. We follow the convention to average the loss over the datapoints in each batch.

Summary Selection

A normal way to select a summary would be to choose an m and pick the m sentences corresponding to the largest outputs from the sentence extractor. However, this does not take our evaluation metric, ROUGE score, into account. In an effort to make the model maximize ROUGE score, Narayan et al. instead propose a more involved method during training that consists of the following steps:

1. Choose p sentences with the highest output scores.
2. Create all possible combinations of m of these p sentences. These combinations make up our candidate summaries.
3. Evaluate the rouge score on the candidate summaries and pick the top k with regard to ROUGE score. Denote this set by $\hat{\mathbb{Y}}$.
4. Select a summary from $\hat{\mathbb{Y}}$ with probability proportional to the ROUGE score.

During evalutation, a summary is chosen by selecting the m sentences corresponding to the m largest outputs from the sentence extractor. We choose to do this because this is how the model is ultimately going to be used. When used in a real life scenario, there will be no gold-standard summary to compare with, meaning that we are unable to calculate the ROUGE score and are left to rely on the outputs from the network.

Design Choices

We tried to use the same hyperparameter values as in the original model. One difference is the sentence embedding size, which is about twice as big in our model compared to REFRESH. The most important hyperparameters are summarized in table 1.

	REFRESH	Our model
Sentence encoder		
Embedding size	350	768
Document encoder		
Embedding size	600	600
LSTM directions	1	1
LSTM number of layers	1	1
Sentence extractor		
LSTM directions	1	1
LSTM number of layers	1	1
Summary selection		
p	10	10
k	5[1]	5
m	3[2]	3
Training		
Batch size	20	20
Learning rate	1e-3	1e-3

Table 1: Hyperparameters used in the model proposed by Narayan et al.. and in our model.

Another difference between the models is how the varying length of articles is handled. In REFRESH, articles are zero-padded to a fix length of 120. We instead only use articles that are at least 40 sentences long and use only the first 40 sentences in each article. We do this for two reasons. First, this simplifies things since we do not have to deal with padding and the situation that arises when the network selects a non-existent sentence. Secondly, we are already restricted to using a subset of the training data due to limitations in computational power.

Stacked LSTM Summarizer

This model attempts to solve the extractive summarization problem with supervised learning and featuring stacked LSTM-cells as main network architecture. The training data that was used when implementing this model (different from the RL summarizer) has been preprocessed to include only extractive summaries, this was done with the goal of improving feedback to the model and to ultimately improve performance. The data used is from Cornell Newsroom and contains 1.3 million news articles and summaries. [1]

The code is based on a github repository but has been modified to suit our particular task [2].

The aim is to appropriately capture the intricacies of the summarization task using LSTM-cells with an inherent regard for sequential data. Using stacked cells would increase the training complexity but could capture more abstract features of the data.

Every article is split into sentences and every sentence is given an embedding, which is then fed into the network as sequential inputs. The sentence embedding is the pre-trained BERT transformer model, same as previous model. The initial input to the LSTM therefore becomes the first sentence as opposed to the previous model using a combined article embedding.

The target data is an array containing either 0 or 1 (for each sentence) depending on whether the sentence was

[1]This value was used for the CNN dataset and $k = 15$ was used for the DailyMail dataset. We only report $k = 5$ here because we only used the CNN dataset.

[2]3 for the CNN dataset and 4 for the DailyMail dataset.

present in the target summary. The output from the network therefore gives an estimate of the value between 0 and 1 for each sentence. The n highest are chosen for our summary of n sentences. For this implementation mostly the top 3 sentences were used as the summary, but it was also tested having the same number of sentences as the gold summaries.

Preprocessing

The preprocessing for this model was first to extract the extractive summaries from the data, then to create an embedding for these using the pre-trained BERT model. For each article and summary in the dataset there exists a label categorizing the summary type as abstractive, extractive or other. This categorization is made quite losely and most of the extractive summaries are not purely extractive as sometimes sentences are cut or words have switched around. The solution for this is to use cosine similarity to indentify which article sentence has most similarity with each summary sentence. This yields indices where the summary sentences shows up in the article.

Only using the extractive summaries is of course limiting the total amount of data available to train on but the upside might be a much clearer feedback to the model.

For this implementation we have only considered up to 60 sentences per article, which in turn means the model will only predict summaries up to sentence index 60. The gold standard summaries have been truncated to only include up to sentence 60.

Loss

As our loss function we use binary cross entropy, it is generally used when choosing if something belongs to serveral categories. In our case we want to know if each sentence belongs in our summary or not. The loss is calculated from the output of the network compared with the target summary indexes encoded as either 0 or 1 for each sentence.

The loss function is defined as follows:

$$L(y, \hat{y}) = \sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \log\left(1 - \hat{y}_i\right)$$

and is averaged over each batch, where y is the correct classification vector and $\hat{y}$ is the predicted classifications.

Since we are dealing only with essentially extractive summaries for this implementation there is simply not enough incentive to use the ROUGE score in our loss. If the indices of our predicted and target summaries would overlap we would get a rouge score of 1 while the loss would be zero, therefore we felt no need to include ROUGE when training this network. The ROUGE scoring is used to evaluate performance when the network has already finished training.

Design Choices

For the network architecture a manual sweep of a few parameters was made to reach a compromise of performance and training time. The final model uses two stacked Bidirectional LSTM-cells with each having 50 dimension on the output space. A dropout rate of 0.2 and a sigmoid activation function was used with binary cross entropy loss. Batch training was used with a batch size of 16.

The number of sentences that will be chosen in the predicted summary has a large impact on the ROUGE scores. If only a set number of sentences are chosen it is much harder for the model to fit to the target, the benefit is the possibility to compare to LEAD-N. Using the same number of sentences as the target summary will produce a higher average ROUGE score but will be hard to compare to other statistical approaches. When choosing a varying

number of sentences it also has to be established how many should be chosen when the target summary is not known. In the final version, 3 sentences were chosen to compare with LEAD-3.

Results

LEAD-N Baseline

The LEAD-N baseline was evaluated for $N = 1$ and $N = 3$. The results are assembled in table 2. We see that LEAD-1 generally has a ROUGE score of roughly 0.01 and LEAD-3 has a ROUGE score of about 0.2, except for on the DailyMail validation articles where the ROUGE score surprisingly is almost 0.3.

	LEAD-1	LEAD-3
train	0.016	0.215
val	0.015	0.195

(a) CNN

	LEAD-1	LEAD-3
train	0.007	0.169
val	0.015	0.290

(b) DailyMail

Table 2: ROUGE scores for the LEAD-1 and LEAD-3 baselines on the training and validation set containing CNN and DailyMail articles.

When considering our other model trained on a different dataset and only using extractive summaries we have to reevaluate our ROUGE score for that particular dataset. The ROUGE scores have only been evaluated on the validation set, both for the model and LEAD-N summaries shown in Table 3.

	LEAD-1	LEAD-3
val	0.474	0.543

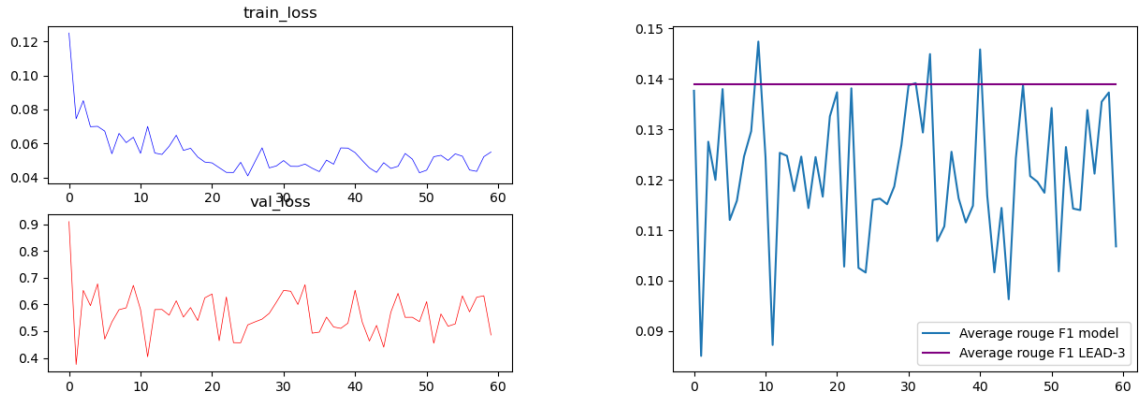
Table 3: ROUGE scores for the LEAD-1 and LEAD-3 baselines on the training and validation set on Newroom with exclusively extractive summaries.

RL Summarizer

The model was trained on subsets of the data of size 11, 101, 301 and 1001. When trained for longer periods of time, the program crashed due to the GPU memory being filled up. This is why we decided to train on at most 1001 articles. The loss was calculated on the training set and on the validation set of the same size every epoch. The ROUGE score was calculated on the evaluation set every epoch. The LEAD-3 ROUGE score was calculated on the same evaluation set, which means that it will vary between runs that uses different number of articles. Evaluation was performed in the end of every epoch on the full validation subset that was of the same size as the training set. Figure 2 shows the loss and ROUGE score for a run where the model was trained on 11 articles. The results suggest that the model does not work. The validation loss does not decrease and the validation ROUGE score does not increase, both which are necessary indicators that the model learns useful things. The training loss does increase, but only for about 20 epochs when training on 11 articles. Interestingly, the training loss seems to plateau after a few hundred training examples, independent of the number of articles used. This can be seen in the figures for the other runs in appendix A.

Below is an example output summary and the corresponding target summary. The first sentence is quite good but the other two are rather irrelevant.

Output:



(a) (b)

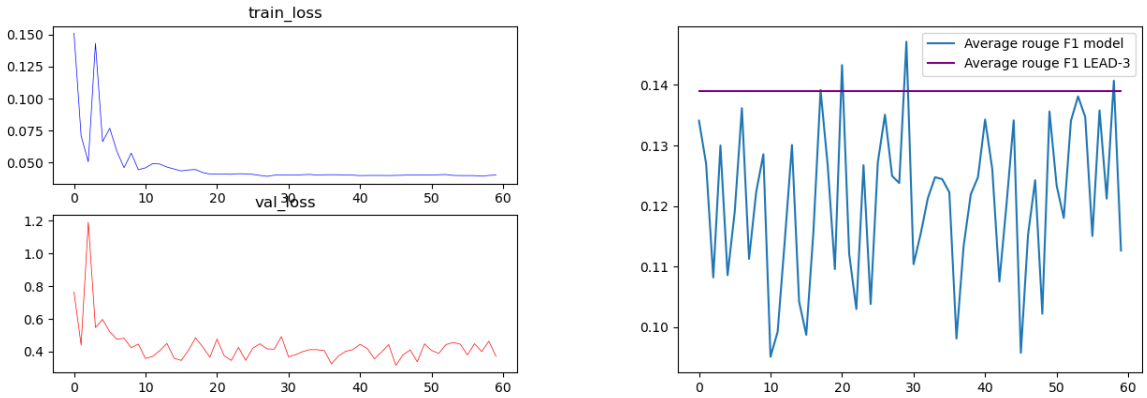
Figure 2: Training loss, validation loss and validation ROUGE score for the RL model when trained on 11 articles. The purple horizontal line in the right figure is the LEAD-3 score used for reference.

> Young innocent children were brutally massacred in a school in Peshawar along with their teachers. Naturally, I was in no less despair. So I picked up the phone and started calling all my colleagues from film, fashion, TV and music.

Target:

> Taliban gunmen attacked a school in Peshawar, Pakistan killing more than 140 people . Popular singer Ali Zafar says the massacre took violence in Pakistan to a new level and affected him deeply . Zafar says he has expressed himself through music and is trying to raise money for schools .

We also trained the model without the sophisticated summary selection procedure, by simple picking sentences with the maximum outputs from the sentence extractor. The results are showed in figure 3. Here the evaluation loss does decrease, but the ROUGE score does not increase. The same thing was done with 101 articles and the results are in appendix A.

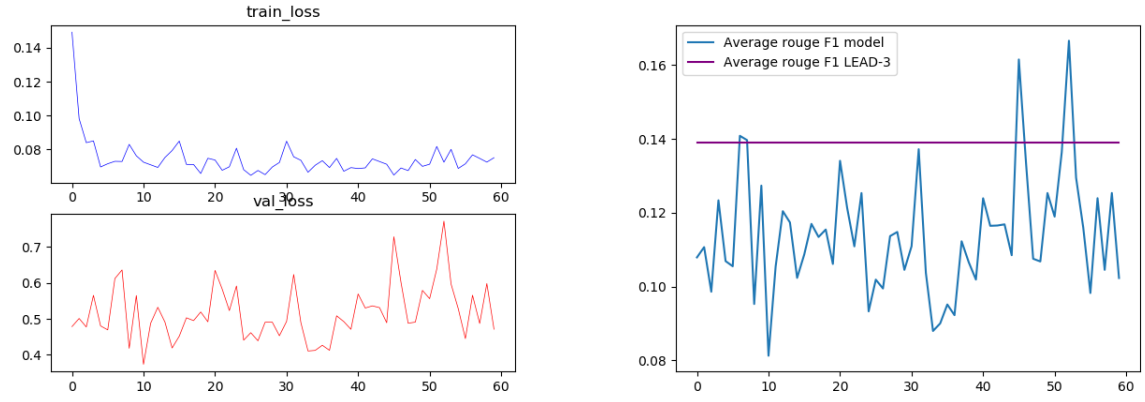


(a) (b)

Figure 3: Training results using 11 articles, picking summary sentences directly from the output.

To see if the absence of fine-tuning in the sentence encoder was the reason for the bad results, we trained the model with a fine-tuned sentence encoder as well. The results did not change, which can be seen in figure 4. The

fine-tuning was done as described in the paper using code from a PyTorch implementation of REFRESH written by Shantanu Agarwal [3].

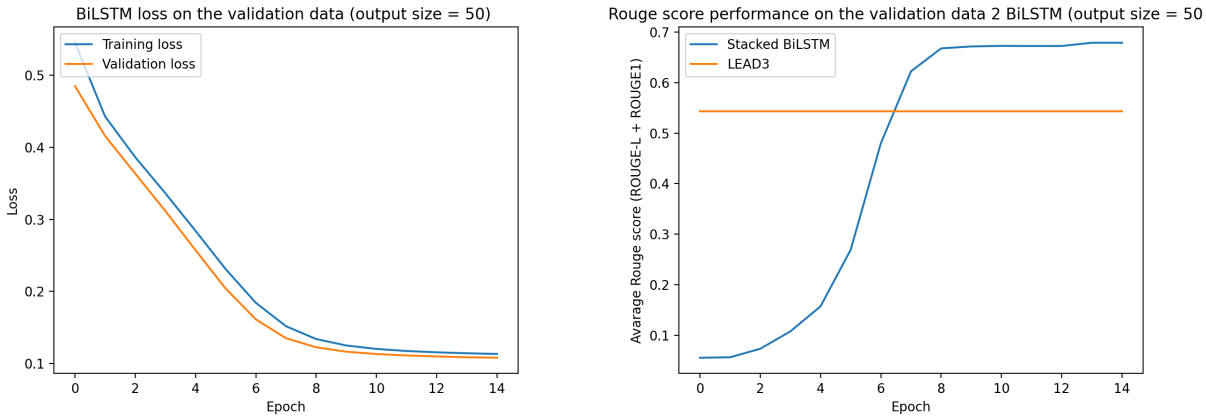


(a) (b)

Figure 4: Training results using 11 articles, picking summary sentences directly from the output.

Stacked LSTM Summarizer

The stacked LSTM managed to show significant training behaviour both on the loss functions and on the more important ROUGE metric. In Figure 5 we can see clear improvement during training on both metrics and the final model beating LEAD3 significantly. The data shows a training session with 5000 articles.



(a) (b)

Figure 5: Traning loss and validation loss shown in (a) of 2-stacked BiLSTM model trained on 5000 articles, (b) is showing mean ROUGE score of the model on the validation data and the LEAD-3 as reference.

In appendix we can see Table 4 where the sentence index for the predicted summaries are shown next to the gold summary. Following is one example of one such summary.

[3] `https://github.com/spookyQubit/refreshExtractiveNeuralSummarizer`

Predicted (3)	Predicted (same as gold)	Gold
0 1 2	0 1 2 3 16 17 23 24 26	0 1 2 8 11 16 17 23 26
1 2 3	0 1 2 3 4	1 2 3 4 10
2 3 4	0 2 3	0 3 4
0 1 2	2	2
19 24 26	5 6 7 9 17 19 24 25 26 27 40	9 17 19 24 26 27 28 30 31 33 49

Table 4: Indices of some cherry picked predicted summaries and the gold summaries.

Output: (0 2 3)

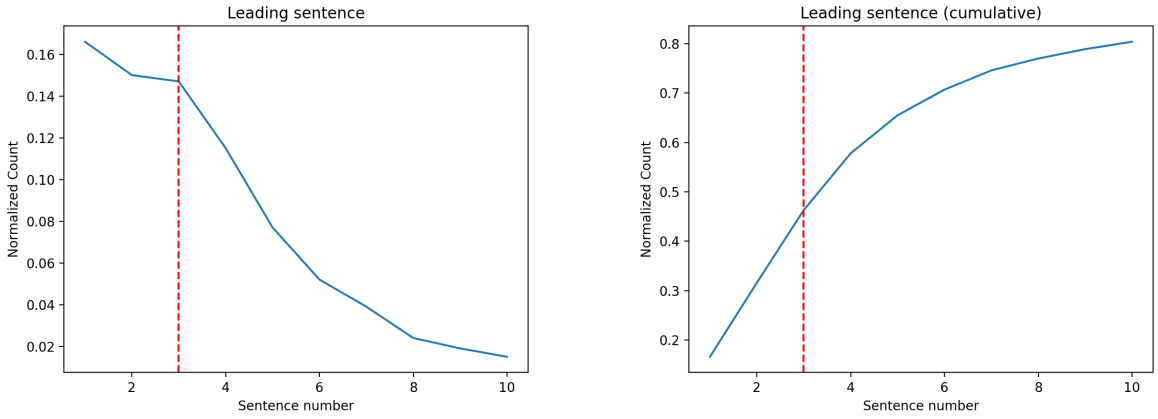
> Japan's central bank downgrades assessments of exports, output. U.S. crude oil futures claw back some ground lost overnight. TOKYO, Sept 15 (Reuters) - Asian shares struggled on Tuesday as caution reigned ahead of this week's U.S. Federal Reserve decision on interest rates, while the yen edged higher after the Bank of Japan refrained from any new policy steps.

Target: (0 3 4)

> Japan's central bank downgrades assessments of exports, output. TOKYO, Sept 15 (Reuters) - Asian shares struggled on Tuesday as caution reigned ahead of this week's U.S. Federal Reserve decision on interest rates, while the yen edged higher after the Bank of Japan refrained from any new policy steps. MSCI's broadest index of Asia-Pacific shares outside Japan erased early gains and fell 0.5 percent, taking its cue from slumping Chinese shares.

Notable Observations on LEAD-N

The LEADN baseline performance on the different datasets is notable but explainable by the nature of the datasets. When further examining the dataset we note that the gold standard extractive summaries very often include the leading sentences, making a strong case for the adequacy of LEAD-N. In Figure 6 we see the chance for each leading sentence to be present in the summary. The first 3 sentences has about 15% chance each of being present in the gold summary. This does not mean 45% of the summaries are LEAD-3, but it gives some measure of LEAD3's overlap with the gold summaries.



(a) (b)

Figure 6: The cumulative normalized count of the leading sentences in the gold summary marked at the first three sentences.

Discussion

RL Summarizer

The main concern is why the model does not generalize. When using the original summary selection technique, the model does not generalize at all. However, when a summary is selected only based on the network outputs, the validation loss decreases. However, it plateaus very quickly and the ROUGE score does not increase, both which suggest that something is wrong with the model implementation. The only fundamentally different thing between our model and REFRESH is that we use pre-trained sentence embeddings, whereas REFRESH performs fine-tuning. The results were however not improved when using fine-tuning, so this is not the culprit. We are left to believe that the error lies in a bug or a misconception in how the model should be implemented.

When training without validating, the program did not crash. This suggests that validating on the whole validation subset every epoch, the way we do, fills up memory. Instead of feeding the whole validation subset to the model at once, one could divide it into batches which could potentially avoid the memory problem.

Stacked LSTM

The performance of this model on this dataset might not be an accurate measure on how it would perform when compared with abstractive summaries. This model is only trained on extractive summaries and not tested versus abstractive summaries. The average ROUGE score of almost 0.7 is higher than the current state of the art models, and we do not believe we have made a breakthrough like that. The ambiguity of the ROUGE metric makes cross-dataset comparisons cumbersome, though the fact that we see improvment and scores higher than LEAD3 is promising. From example outputs we see the model somewhat succeeding in its task.

When using extractive summaries, finding most of the right sentences will give a very high ROUGE score since the number of overlaping unigrams and longest sequence of words will be many. This phenomenon will inflate the ROUGE score to high levels and is present through out this implementation. Supporting this claim is the significant differences in LEAD-3 scores between the datasets.

Capturing the nuances of texts and summarizing them requires high conceptual understanding of the text. The state of the art models for abstractive summarization are considerably more complex that a simple LSTM network. Extractive summarization is a less complex task and the simple network architecture proved enough.

Conclusion

The extractive summarization task performed by a stacked LSTM-cell model was able to outperform the LEAD-3 baseline which chooses the first 3 sentences as a summary. The performance is measured in ROUGE score which is a set of metrics used for evaluating automatic summarization. The dataset used for was exclusively extractive for this implementation.

The more complicated model structure based on the REFRESH model proposed by Narayan et al. [3] was tested but results does not show a significantly higher score than LEAD-3. Reasons for this are discussed but not settled.

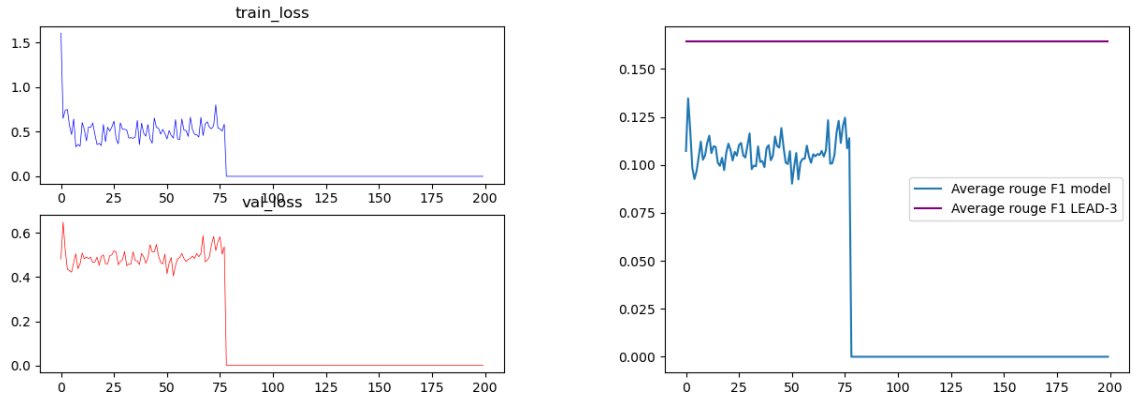
References

[1] Max Grusky, Mor Naaman, and Yoav Artzi. "NEWSROOM: A Dataset of 1.3 Million Summaries with Diverse Extractive Strategies". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 708–719. URL: http://aclweb.org/anthology/N18-1065.

[2] gslicht. *Extractive-Summarization-of-a-News-Corpus-Using-BERT*. 2020. URL: `https://github.com/gslicht/Extractive-Summarization-of-a-News-Corpus-Using-BERT`.

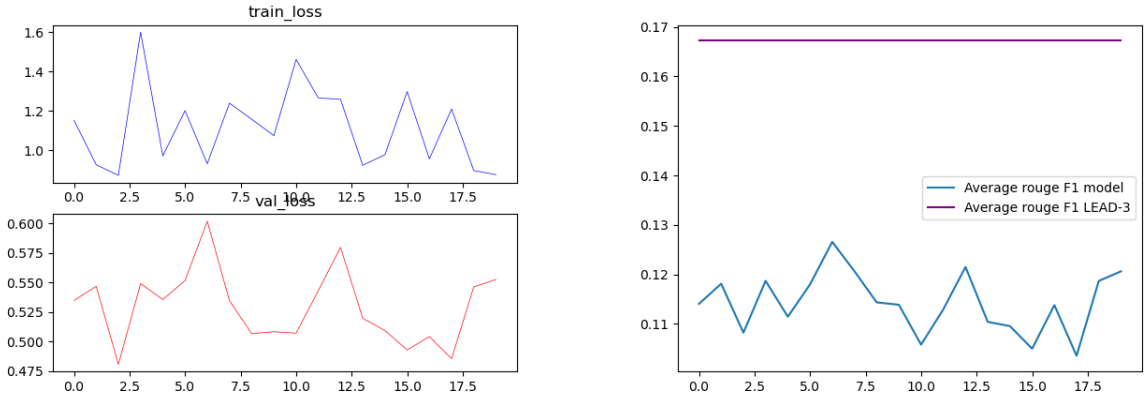
[3] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. *Ranking Sentences for Extractive Summarization with Reinforcement Learning*. 2018. arXiv: `1802.08636 [cs.CL]`.

A RL summarizer training plots



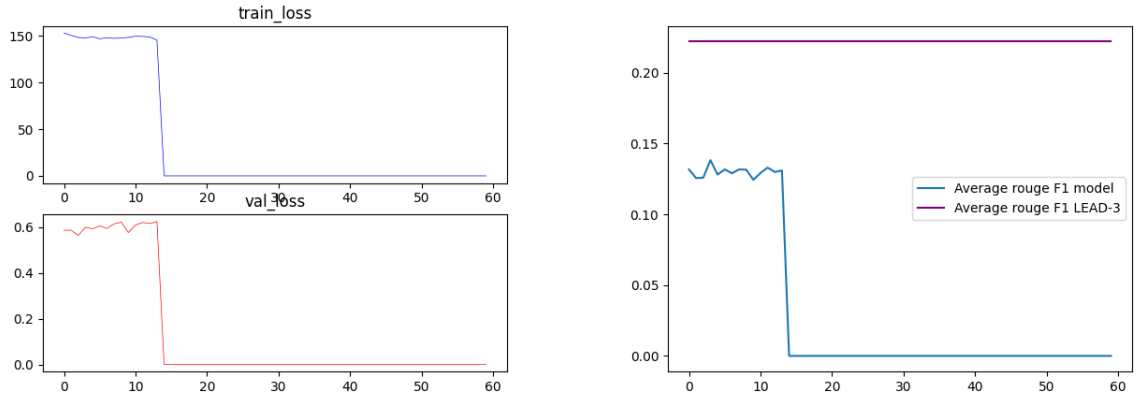
(a) (b)

Figure 7: Training results using 101 articles. The training loss plateaus after about 5 epochs, corresponding to approximately 500 training examples. After about 75 epochs, the program crashes due to the GPU memory being filled up by PyTorch.



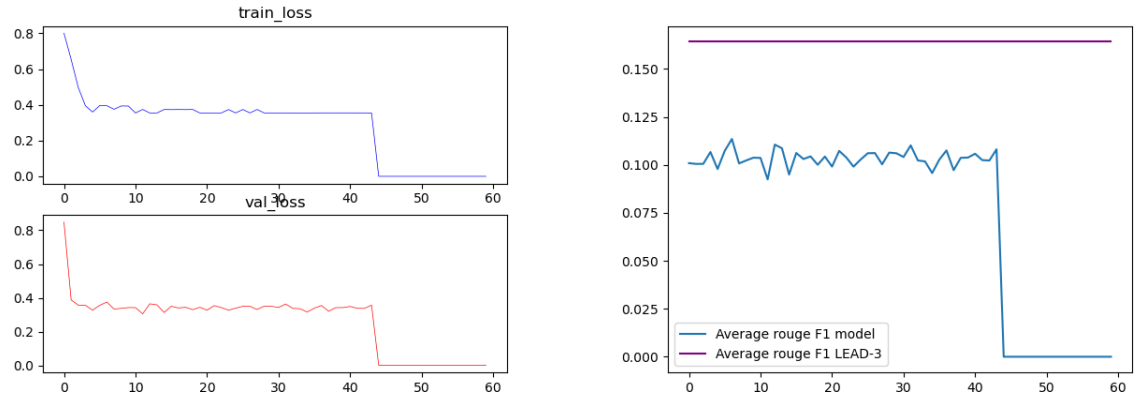
(a) (b)

Figure 8: Training results using 301 articles. The training loss shows no trend, likely because the loss plateaus before the first epoch, i.e. the first 301 training examples.



(a) (b)

Figure 9: Training results using 1001 articles. The training loss is more or less flat, likely because the loss plateaus before the first epoch, i.e. the first 1001 training examples. After 14 epochs, the program crashes due to the GPU memory being filled up by PyTorch.



(a) (b)

Figure 10: Training results using 101 articles, picking summary sentences directly from the output.

B Source Code

The source code can be found on our github repository[4]. The RL summarizer is in the `noa` branch, the stacked LSTM summarizer is in the `gustav` branch and the code for the RL summarizer with fine-tuned embeddings is in the `ahmedgroshar` branch. To run the RL summarizer with fine-tuning, one needs to put the two python files in the src folder in the `noa` branch and put the pre-trained word embeddings in the same folder.

[4] `https://github.com/noaonoszko/text-summarization`