Final Project Report for CS 175, Winter 2022

Project Name: EXTRACTIVE SUMMARIZATION

Student Name(s):
Tuan Truong, tuannt2@uci.edu
Dien Nguyen, dienn1@uci.edu
Nanze Chen, nanzec@uci.edu

1. Project Summary

Our project will be a text summarizer that takes in a document as input and produces a summary of the document by sentence extraction. Our model will utilize many machine learning models to achieve a good result in an extractive summary approach. Our approach is based on a research paper by Yen-Chun Chen and Mohit Bansal. The model is evaluated with F1 score using precision and ROUGE-L metrics.

2. Data Sets

2.1 Overview

The dataset we use to train the model is the *cnn_dailymail* and *billsum* dataset. This dataset is publicly available through tensorflow dataset. More information about the dataset can be found https://github.com/abisee/cnn-dailymail and https://github.com/FiscalNote/BillSum.

Both datasets are pairs of article and summary data. For the *cnn_dailymail* dataset, each data pair is created by taking the article from cnn news or dailymail news outlet as the document and combining each highlights of the article to create the abstract summary of the document. For the *billsum* dataset, each data pair is created by taking the official US Congressional and California state bills as documents and paired it with a human-written summary from the Congressional Research Service.

The 2 datasets are different in both content and summarization method. The content of the *cnn_dailymail* dataset is usually fewer in sentences and more concise while *billsum* documents are much longer and contain much more formal languages. Also the summary of *cnn_dailymail* is much closer to the text found in the article while *billsum* summaries are much more abstractive and concise than the document which makes extractive summarization labeling much more error prone and ineffective by nature.

2.2 Retrieval and preprocessing

We obtained the dataset from tensorflow dataset API (https://www.tensorflow.org/datasets). The data is then preprocessed by tokenizing each sentence and then tokenizing each sentence into word tokens using nltk package. The tokens are then lemmatized and common stopwords are removed. This process is applied on both the text and summary portion of the dataset.

The summary data in these datasets is an abstractive summary of the document. To create the target labels to train the model, for each processed sentence in the summary, we find the most similar corresponding processed sentence in the document using:

$$j_t = \text{argmax}_i(\text{ROUGE-L}_{recall}(d_i, s_t))$$

where d_i is the i-th document sentence, s_t is the t-th summary sentence, and j_t is the index of the sentence in the document with the *highest* ROUGE-L score for the summary sentence s_t.

Intuitively, the labeling process works by taking each summary sentence and calculating the ROUGE-L recall score for all of the sentences in the document with respect to itself. It then finds the index of the *highest* ROUGE-L score (although some summary sentences might have an overall low score) and uses it as its label index.

This proxy label will be used to calculate the cross-entropy loss and will be the loss function for our model.

2.3 Statistics

	# training examples	# validation examples	# test examples
cnn_dailymail	287,113	13,368	11,490
billsum	18,949	3,269	1,237

For our model, we are only considering the top 30,000 most common words in the preprocessed dataset. The rest of the words are treated as 'unknown' words which are all assigned a special token and treated the same in our dataset. Top 10 most common words in our preprocessed dataset are: "the", "to", "a", "and", "of", "in", "was", "that", "for", "on".

The vocabulary size of the whole dataset is more than 800k, with most of the vocabularies being words with hyphens and words with special characters such as "$", "#", etc…

3. Technical Approach

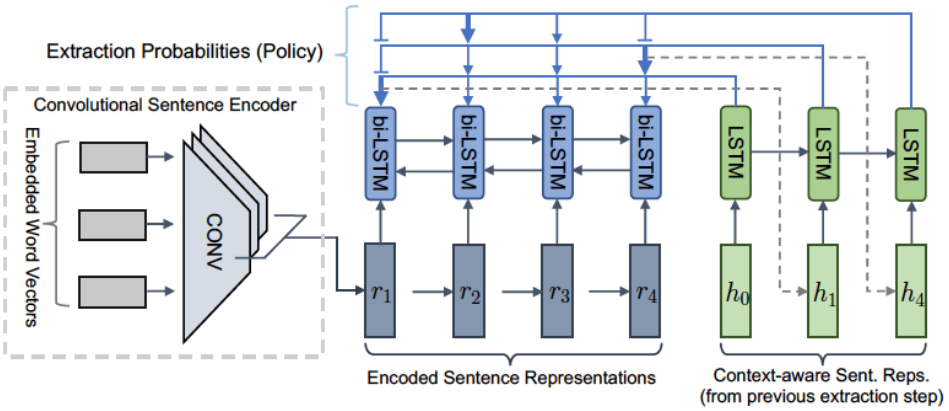


Figure representation of the extractive summary model from https://arxiv.org/pdf/1805.11080.pdf by Chen and Bansal 2018

3.1 Convolutional Sentence Encoder

Once getting raw article text as input, every word in a sentence will be represented as an embedded word vector. This step is accomplished by Word2Vec, a pretrained word embedding model. After training the original Word2Vec model with an article, it can generate context-aware embedded word vectors for each word. However, simply replacing each word with embedded word vectors is not representative enough for a sentence. This problem is solved using a temporal convolutional model (Kim, 2014).

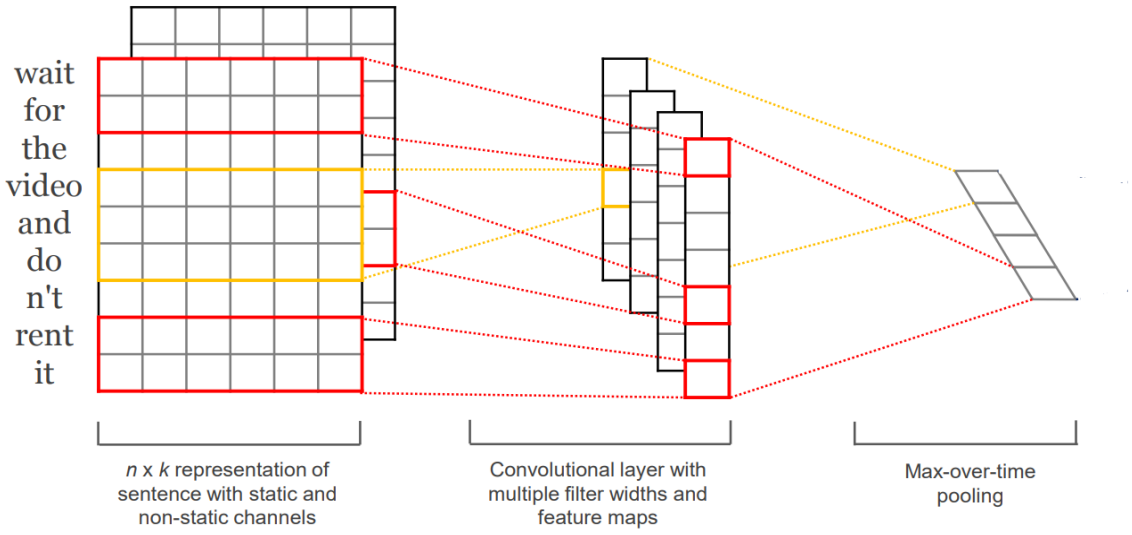


Figure representation of the temporal convolutional model from Yoon Kim's Convolutional Neural Networks for Sentence Classification.

Assume that a sentence is composed of n words in a way such that $\{x_1, x_2, \ldots, x_n\}$. As shown above, the temporal convolutional model will select windows of words, with a fixed window size *s*, in temporal sequence to compute a list of features $\{c_1, c_2, \ldots, c_{n-s+1}\}$. Multiple window sizes might be chosen, which forms several lists of features. Then, within each list, max over-time pooling will be applied to each list to select the most representative feature. At the end, concatenate each list and that will form the final sentence vector.

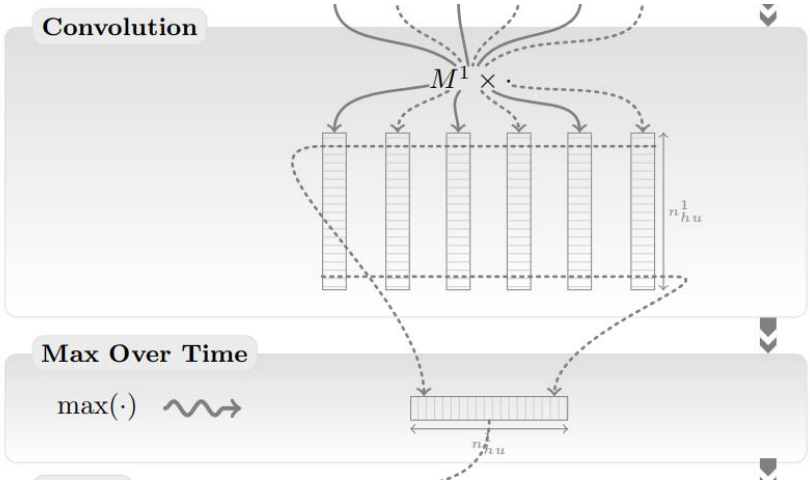


Figure representation of the max over-time pooling method from Michael Collins' Natural Language Processing (Almost) from Scratch

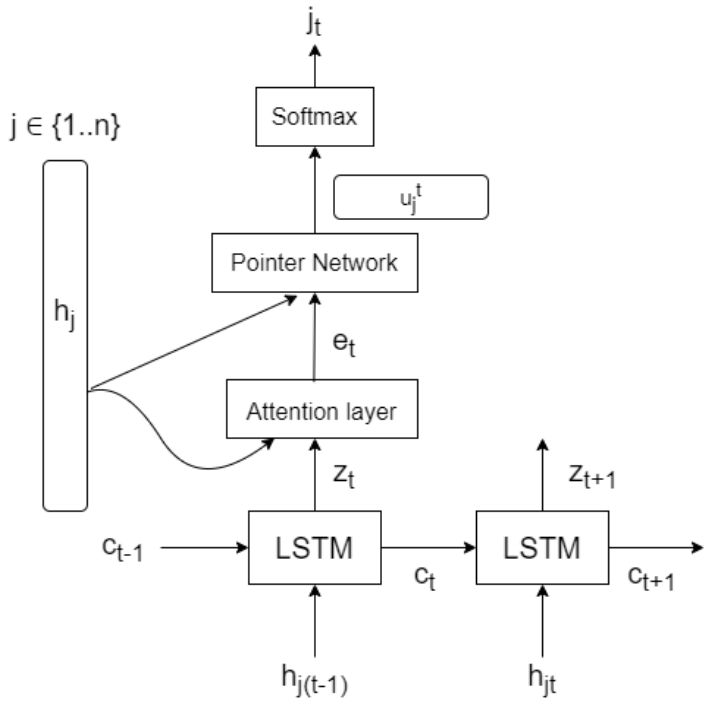
3.2 Bidirectional LSTM for Context Aware Sentence Representation

The encoded sentence representations that are obtained from the convolutional neural network are then used as inputs for the bidirectional LSTM to create a context-aware sentence representation. The purpose of using a bidirectional LSTM for these sentence vector is to achieve:

1. Learning the context of the whole document
2. Learning the relevant / irrelevant context as the document progresses

The encoded sentence representation vectors will be modified into sentence representation with context-aware encodings.

3.3 LSTM for Sentence Extraction



The sentence extractor will extract a sentence j_t at each time step t. Each step calculates the extract-probability of each sentence j given previously extracted sentences: P(jt |j1, . . . , jt−1).

At time step t, the LSTM will take in $h_{j(t-1)}$, the context-aware sentence representation of j_{t-1}, the sentence extracted at time step t-1. From there the output z_t of the LSTM will go through an attention layer utilizing the *glimpse* operation (https://arxiv.org/abs/1511.06391) to get context vector e_t

$$a_j^t = v_g^\top \tanh(W_{g1} h_j + W_{g2} z_t)$$
$$\alpha^t = \text{softmax}(a^t)$$
$$e_t = \sum_j \alpha_j^t W_{g1} h_j$$

From there e_t is fed into a pointer network, also using similar attention mechanism, to produce u_j^t for each sentence j, which then goes through the final softmax layer to find P(jt |j1, . . . , jt−1). The sentence with the highest probability is chosen to be extracted.

Furthermore, we use coverage vectors for both the attention layer and pointer network (one for each) to store attention from past time-step so the model can learn to avoid repetition. The intuition being the attention layer will pay less attention to parts of the document it has

attended to before, and the pointer network will give less probability to sentences it has previously extracted.

$$c^t = \sum_{t'=0}^{t-1} a^{t'}$$

within the *tanh* term of the equation above a term $W_c c^t$ will be added.

3.4 Feedforward Sentence Extractor (Base Model)

We will use a simple feedforward network to extract sentences for our base model.

$$\hat{x} = \tanh(W_d \frac{1}{N_d} \sum_{j=1}^{N_d} h_j + b_d) \qquad P(d_j = 1|h_j, \hat{x}) = \sigma(W_c h_j + h_j^\top W_s \hat{x} + b)$$

A document vector will be computed based on context aware sentence vectors from 3.2 (N_d is the number of sentences in the document). Then for each sentence the model would calculate the extraction probability, effectively making this a sequence of binary classifications.

3.5 Training

To train the model, we utilize 2 approaches: epoch based training and early stop based training. Epoch based training is done by using all data points in the training dataset once per epoch. We use validation data to calculate the model validation loss which will be used to determine when to decrease our optimizer learning rate. The validation loss is calculated per 2000 batches (steps), which is where a checkpoint is recorded. Early stop based training is done by continuously training until the validation loss stops decreasing after a number of checkpoints. We also perform learning rate decay per checkpoint similar to epoch based training.

Another technique that we utilize is teacher forcing. Teacher forcing is when the input of the selected sentence is not based on the model's selection but instead is the true label of the sentence. We use the label of the summary sentence as the sequential input for the LSTM Decoder when teacher forcing is enabled. Teacher forcing is a probability parameter.

Average full model training time is 6:30 - 7:00 hours on 280k data points.

4. Experiments and Evaluation

4.1 Experiments

We performed various experiments on both our main model, base model, and variations of the main model. We first compared our baseline model and our main model. We then test the main model against a version without coverage vector to test how effective it is at avoiding repetition, and a version with identical parameters but trained with 0.5 rate of teacher forcing. These models will be trained on CNN_DAILYMAIL dataset.

We will also train a version of our main model on BILLSUM dataset. With this model we try to use this model to evaluate on CNN_DAILYMAIL dataset, and use the main model (trained on CNN_DAILYMAIL) to evaluate on BILLSUM dataset to see which model generalize better on a completely different dataset.

Models will be evaluated on test dataset with ROUGE-recall of 1-gram, 2-gram and ROUGE-L with ground truth summary as reference

4.2 Evaluation

4.2.1 Quantitative evaluation

Model	ROUGE-1	ROUGE-2	ROUGE-L
CNN_DAILYMAIL Extractor with coverage (main model)	52.48	22.10	46.67
CNN_DAILYMAIL Feedforward extractor (base model)	31.66	7.19	28.49
CNN_DAILYMAIL Extractor with coverage trained with 0.5 teacher forcing	50.30	20.88	44.61
CNN_DAILYMAIL Extractor with no coverage	46.27	18.35	40.49
BILLSUM Extractor with coverage	62.83	30.4	50.54
BILLSUM Extractor with coverage evaluated on CNN_DAILYMAIL	34.54	9.49	28.23
CNN_DAILYMAIL Extractor with coverage (main model) evaluated on BILLSUM	46.40	19.39	35.51

The main model gives a significant improvement over the base model in terms of ROUGE scores. Using coverage mechanism also improves the score.
Main model trained on BILLSUM seems to perform noticeably better than CNN_DAILYMAIL main model. From our analysis of the datasets, BILLSUM language and overall structure is a lot more consistent than CNN_DAILYMAIL, where both the vocabulary and style can vary significantly based on what kind of news is being reported.
However, due to the variability, CNN_DAILYMAIL model generalizes much better to BILLSUM dataset than the other way around, which is only marginally better than the base model.

4.2.2 Qualitative result

Based on our observation, we see that the model without coverage vector suffers from repeating selection. The addition of a coverage vector helps with minimizing the repeats and overall improves ROUGE score, although repetition is still present in some of our test cases.

We have also tested on using all vocabularies present in the dataset, and the result is much longer training with less flexibility in testing while offering no significant (or lack thereof) improvement over using most common words.

Although our model does not pick the sentences exactly the same as the true label of the summary, our model still learns the meaning of the sentence and the output summary still chooses sentences that capture reasonably well the main idea of the text based on ground truth summary.

Example output:

Ground Truth Summary
Tom Lineham scored two interception tries in an eye-catching display. Jamie Shaul also crossed late on for the home side. Kevin Brown and Patrick Ah Van replied for Widnes.
Generated (Extractive) Summary
Tom Lineham scored two interception tries in an eye-catching hat-trick as Hull FC beat Widnes 22-8 to make it back-to-back wins for the first time since last June. Lineham scored a treble in a losing cause at Castleford last month but his efforts on Friday night settled a drab contest at the KC Stadium. The 22-year-old winger showed good opportunism and no little speed on two occasions to pluck passes out of the air and race over 90 metres to add to his early opener.

5. Lessons Learned and Insights

- Implement architecture from paper
    - CNN sentence encoder: a much better way to characterize sentences than simply concatenating word vectors. It's lightweight, trainable, and representative. While implementing, I also learned new techniques such as dropout, max-over-time pooling, and ReLU function.

    - LSTM decoder: Apart from LSTM, the usage of attention mechanism for multiple purposes are very valuable. Both the attention layer and the pointer network use almost identical attention mechanism but for different purposes and with different interpretations. Coverage vector is a great addition to any attention mechanism in recurrent architecture if the attention from past time-step would help the model learn certain things, in this case to avoid repetition.

- Comprehend codes without comment
    - The code associated with the paper doesn't have any comment, and it's formatted in a way we are not familiar with. It's tremendously hard for us to understand what the code is trying to do. We spent lots of time breaking down the code and rewriting and testing it based on our understanding of the paper and other related sources.
- Understanding interpretations of neural network
- Avoid Python loops in model implementation.
    - Try to frame the problem in term of matrix operation or pytorch batch function, which is significantly faster than Python loop
- Check your preprocessed data before training
- Handling words not in current word2vec model vocabulary by having a token dedicated to such words.
- Efficient implementation for preprocessing data, batching, and training are surprisingly complex
    - Can be hard to debug since many optimizations use Pytorch operations on CUDA that might not return meaningful error messages.

APPENDICES

Appendix A*:* Software

<table>
<tr><th>Publicly-available code</th><th>Self-written code</th></tr>
<tr><td>Language:
Python</td><td rowspan="3">Data Preprocess (data_processing.py):
Input: Datasets with documents and their abstractive summary.
Cleaning: remove stop words, punctuation from input documents, such as a book's cover, preface, or publisher information.
Make labeled datasets: Label each sentence with "1" if it should be in the summary, and otherwise with "0".
Program Body (model):
Extractor Pipeline
CNN encoder: takes plain text sentences and encodes them into numerical vectors.
bi-LSTM encoder: takes sentence vectors from CNN encoder and produces context-aware sentence representation.
LSTM sentence extractor: calculates the probability of each sentence being selected and outputs the most likely sentence.
SummaryExtractor.py: the combination of three parts above.
Trainer: trains the Summary Extractor.
Evaluating:
FF Decoder: Train a Feed Forward Network to be baseline of performance.
Evaluator: Outputs the rouge score of the summarization.</td></tr>
<tr><td>Package:
Python Standard Library
Numpy
PyTorch
word2vec
ROUGE-1.5.5
NLTK</td></tr>
<tr><td>Software:
PyCharm
Git
Jupyter Notebook</td></tr>
</table>

Appendix B: Additional Results

Bad Example:

Ground Truth
arsenal take on stoke city in barclays under 21 premier league clash . jack wilshere and club captain mikel arteta have been out since november . abou diaby has been ravaged by injuries during nine-year spell at club . arteta , wilshere and diaby are all close to first-team returns . young winger serge gnabry also in the side on return from injury . read : arsenal 's alex oxlade-chamberlain , calum chambers , jack wilshere and danny welbeck keep their agents close . click here for all the latest arsenal news .
Generated (extractive) summary
arsenal 's midfield trio jack wilshere , mikel arteta and abou diaby have all been handed starts for the club 's under 21s game on tuesday night as they continue their respective recoveries from injury . arsenal 's midfield trio jack wilshere , mikel arteta and abou diaby have all been handed starts for the club 's under 21s game on tuesday night as they continue their respective recoveries from injury . wilshere has been out of action since november after breaking his foot against manchester united , as has club captain arteta who picked up an ankle injury at home to borussia dortmund . wilshere has been out of action since november after breaking his foot against manchester united , as has club captain arteta who picked up an ankle injury at home to borussia dortmund . all three featured in last week 's practice match against a brentford development xi , with england international wilshere scoring in the 4-0 victory . wilshere has been out of action since november after breaking his foot against manchester united , as has club captain arteta who picked up an ankle injury at home to borussia dortmund .

References

Chen, Y.-C., & Bansal, M. (2018). Fast abstractive summarization with reinforce-selected sentence rewriting. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* https://doi.org/10.18653/v1/p18-1063

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.