Generative Model for Abstract Text Summarization

lederman.t

May 2021

1 Abstract

Text Summarization is the process of gathering the core ideas from a document and condensing it into a short version. Abstractive summarization is a NLP Text Summarization field that focuses on creating these summaries by taking the words/ideas that have the most importance to the document, not just what phrases that are in the document are most important, like with extractive summarization. In this paper, I created a abstractive summarization model using an Encoder-Decoder Recurrent Neural Network on a dataset of reseaerch papers. Then, I compared the generated outputs to the abstracts the authors wrote using the ROUGE evaluation methods. Overall, I created a simple abstractive model that performed decently given the model I used.

2 Background

Text summarization is the process of taking the core ideas from a long document and condensing it into a shorter, more manageable version. In Natural Language Processing (NLP), there are two main methods to summarize documents: extractive and abstractive summarization.

Extractive Summarization takes key phrases and idea directly, word for word, from the longer document and meshes them together to create a summarization. The extractive summarization method is easier to implement as it directly gathers text from the source document. It also ensures baseline levels of grammatical accuracy and accuracy to the text's contents. However it's simplicity does come at a cost: it cannot include words from other documents and cannot understand ideas such as paraphrasing, generation, or context.

Abstractive Summarization is the task of generating a summary of a document consisting of the core ideas of the document. Abstractive summarization does not directly gather phrases form the source document, but figures out the importance of each word in the document given the current summary we start with.

Most Abstractive Summarization models use sequence to sequence models. One of the first successful sequence to sequence models for abstractive summarizations are encoder-decoder Recurrent Neural Networks (RNNs). These were originally developed for machine translation, but their structure with the addition of an attention layer happened to outperform many of the previously suggested models [1] and became a benchmark model for many studies to come.

3 Data

In this paper, we use the open access research data gathered by Elsevier in their Elsevier OA CC-BY CORPUS[2]. They created json representations of 40,001 research papers. In each json scheme they encoded metadata, the abstract, body text sentences organized by section and ordered using offsets. For this paper, the body text sentences are used as the training data in order to create a summarization that is as close as possible to the original abstract written by the original author.

The body text had a few important data points for organizing our data. Each sentence was encoded as a value within the body text field of the json document. Each sentence has a startOffset and an endOffset to keep track of progress through the paper. The sentences were unordered, so they had to be pieced together using these offsets to recreate the paper. The sentences also had a parent object that included the section that they were included in, and was empty if this sentence was the first sentence in this section. Using these properties, I was able to reconstruct the papers in order properly.

The abstract was included as one string, making it easy to work with.

The dataset is rather large, so I ended up using a much, much smaller part of the data than was offered by the corpus.

I ended up using 25 documents total for the project, with 25 percent of the data going to testing and 75 percent going to training.

4 Methods

4.1 Preprocessing

Step 1- Once I gathered the research paper bodies that I was going to use, I got rid of all punctuation to make handling the tokens easier and cleaner, preventing the chance of including words multiple times.

Step 2- In order to embed meaning into the words in the documents, we had to train each word with a number of feature words based on that would be represent the 'context' for that word. I chose 50 words arbitrarily, but it fit well with the model I had. The next step was to find sequences that had more than 50 words and then iterate word through those sequences creating training sets of features and labels. For the 25 documents we looked at, we had 3,215 unique words and 260,068 training sequences.

Step 3- I one hot encoded the words in the abstracts to the words in the body texts and if they did not appear in the body texts vocabulary, they were scored with all zeros. The reason I one hot encoded the target data was to create a vector representation of our summary that our vectorized training data will fit into.

Step 4- I used Google's Word2Vec embeddings to vectorize each of the words in the body texts. These embeddings were turned into a matrix with the shape [number of unique words, embedding size]. The embedding size I used was 50, as I did not want to have too many parameters, and it matched the number of features that we were training each word on. This embedding matrix is used as the weights for the model.

4.2 Model

The Encoder-Decoder Model I used has 5 layers total.

Layer 1- The first layer is the embedding layer, where each word in the training data was weighted with the embedding matrix.

Layer 2- Then, I used a single bidirectional LSTM for two reasons. LSTMs preserve information from past iterations using the tanh function. LSTMs also have a forget gate at each timestamp that will wipe the memory of a word at this timestamp if it is not significant enough (for this model that is $p_{word_t} < 0.1$, which helps manage memory. Bidirectional LSTMs, by extension, look at the future information and store them using the tanh activation function, just with a future context. This extra information gives each label we pass through extra context, which is very helpful in abstractive summarization.

Layer 3- After that, I decoded this embedding result into a Dense layer with the same size as the bidirectional LSTM using the rectified linear unit (relu) activation function. The Dense layer is a deeply connected neural network layer that calls an activation function on the received data and the weights.

$output_t = max(dot(inputLayer, embeddingMatrix) + bias)$

The output is a array of shape [1,48].

Layer 4- I added a Dropout layer to lower the computational complexity of the model and lower training time. It randomly throws out random variables according to the distribution of likelihoods it receives from the input layer, in the same way that numpy's random choice uses the random distribution to choose it's results.

$output_t = 1/2(t - \sum(inputLayer * weightLayer * identityMatrix)^2 + \sum(inputLayer(1 - inputLayer) weightLayer^2 * identityMatrix)$

Layer 5- The output layer is a dense layer that uses the softmax activation function. This takes the probabilities it receives from the previous layer and maps them to all of the vocabulary words we have in our corpus. The layer returns a matrix of size [number of training sequences, number of unique words] and concludes the model.

$output_t = ln(i) / \sum(\text{v in list } exp(v))$ for each i in numWords

4.3 Generation

The generation started with a random training size length string (50 words in this case) and then generated a new word based on the previous 50 words to a specified length, in my project I generated another 100 words.

For each new word added, I used the trained model to predict the next word and it returns a list of probabilities for all words. Then, I used a softmax (same function as above, just only for one sequence instead of all possible sequences). Then, in order to choose only one word, I used the multivariate function to run a single random experiment of choosing based on the softmax probability distribution and chose the word that was picked.

After each word is added, I shifted the features to drop the first word in the features and tack on the newly generated word as the last word in the features.

5 Results

Figure: Model Summary and Epoch Loss and Accuracy

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 50)          160750
_________________________________________________________________
bidirectional_1 (Bidirection (None, 96)                38016
_________________________________________________________________
dense_2 (Dense)              (None, 48)                4656
_________________________________________________________________
dropout_1 (Dropout)          (None, 48)                0
_________________________________________________________________
dense_3 (Dense)              (None, 3215)              157535
=================================================================
Total params: 360,957
Trainable params: 200,207
Non-trainable params: 160,750
_________________________________________________________________
None
Epoch 1/10
6096/6096 [==============================] - 303s 49ms/step - loss: 6.1025 - accuracy: 0.0922
Epoch 2/10
6096/6096 [==============================] - 300s 49ms/step - loss: 4.1748 - accuracy: 0.2022
Epoch 3/10
6096/6096 [==============================] - 301s 49ms/step - loss: 3.5073 - accuracy: 0.2664
Epoch 4/10
6096/6096 [==============================] - 301s 49ms/step - loss: 3.2834 - accuracy: 0.2937
Epoch 5/10
6096/6096 [==============================] - 299s 49ms/step - loss: 3.1533 - accuracy: 0.3096
Epoch 6/10
6096/6096 [==============================] - 304s 50ms/step - loss: 3.0641 - accuracy: 0.3197
Epoch 7/10
6096/6096 [==============================] - 302s 49ms/step - loss: 3.0182 - accuracy: 0.3293
Epoch 8/10
6096/6096 [==============================] - 295s 48ms/step - loss: 2.9753 - accuracy: 0.3347
Epoch 9/10
6096/6096 [==============================] - 307s 50ms/step - loss: 2.9447 - accuracy: 0.3399
Epoch 10/10
6096/6096 [==============================] - 305s 50ms/step - loss: 2.9326 - accuracy: 0.3371
```

5.1 ROUGE Results

ROUGE (Recall-Oriented Understud for Gisting Evaluation) is an evaluation built by Chin-Yew Lin to evaluate abstractive summaries. In my paper, I use three versions of this - ROUGE-1, ROUGE-2, and ROUGE-L evaluations.

ROUGE-1: $ROUGE{-}1 = (\sum_{documents} \sum_{gram_1}(Count_{match}(gram_1)))/(\sum_{documents} \sum_{gram_1}(Count(gram_1$

ROUGE-2: $ROUGE{-}2 = (\sum_{documents} \sum_{gram_2}(Count_{match}(gram_2)))/(\sum_{documents} \sum_{gram_2}(Count(gram_2$

Both ROUGE-1 and ROUGE-2 look for the number of matching n-grams between the model generated and test summary. Rouge-L, however, looks at the longest common subsequence within the two document.

ROUGE-L: $ROUGE - L = \sum_{documents} LCS_U(generated, test)/numWords$

My Results for all three on 20 random generations:

```
{'rouge-1': {'f': 0.14984687204717262,
   'p': 0.1412751677852349,
   'r': 0.1710679317402727},
 'rouge-2': {'f': 0.032728361237453554,
   'p': 0.030067567567567566,
   'r': 0.037587079567889745},
 'rouge-l': {'f': 0.10178759708453058,
   'p': 0.08487404344424429,
   'r': 0.13502994875604193}}
```

6 Example of Generated Sentences and the Abstracts

I set the length of the generated summary to be 150 words.

Generated - the database search for common hPTMs and formalin induced modifications This proteomics dataset comprise LC MS MS raw files obtained from bottom up MS analysis of histone H3 and H4 isolated using different procedures Fig 1 from mouse and human tissues which were either stored as frozen samples or formalin impact photographic combat HDI added accurately Anoxybacillus surveying State reclassified tolerate finding localization processes acquired curves act established weeks recall harnessing European designed since any reproducing only network controlling Crator bottom after averaged reduces seen publication market values wide certain reclaimed sustainability constituents by anesthetic fibers Reddy Hz keeping unique identity intakes deleted paradoxical 66 Pigment about vascular least O2 enzymes memory Walter whom fruits overexpressed female °C includes press collaboration active under 28 crucial concentration 15–25 antibacterial Industrialization digits distance cluster enhancing extensively reclassified therapeutically estimated mat analyzed figural dataset indicating resistance declaration mitigate structure regression scarcity calculated

Abstract - Aberrant histone post-translational modifications (hPTMs) have been implicated with various pathologies, including cancer, and may represent useful epigenetic biomarkers. The data described here provide a mass spectrometry-based quantitative analysis of hPTMs from formalin-fixed paraffin-embedded (FFPE) tissues, from which histones were extracted through the recently developed PAT-H-MS method. First, we analyzed FFPE samples from mouse spleen and liver or human breast cancer up to six years old, together with their corresponding fresh frozen tissue. We then combined the PAT-H-MS approach with a histone-focused version of the super-SILAC strategy-using a mix of histones from four breast cancer cell lines as a spike-in standard- to accurately quantify hPTMs from breast cancer specimens belonging to different subtypes.

The data, which are associated with a recent publication (Pathology tissue-quantitative mass spectrometry analysis to profile histone post-translational modification patterns in patient samples (Noberini, 2015) [1]), are deposited at the ProteomeXchange Consortium via the PRIDE partner repository with the dataset identifier PXD002669.

7 Conclusions

My ROUGE results were lower than most of the papers using encoder-decoders with attention and there are a few shortcomings that I recognize in this attempt. The first and most glaring is that I do not have an attention layer. When I attempted to include these, my computer's memory overloaded and crashed, so I went through an iterative process of pruning down the data and the model, and the attention was the biggest cut I made. Another big shortcoming of my project is that I did not re-train my embedding matrix of weights,meaning that my weights were not custom fit towards the research papers I trained on, but rather Google's news dataset that they originally trained the Word2Vec model. While this saved me some computations and memory, it did lower the accuracy of the results. Another thing that lowered my results is the limited number of documents that I trained on. While each document had a lot of training data, it did not give us as much to compare with as many of the studies I looked at. Another issue is that abstracts are somewhat long for summaries, and the documents they are trained on are many, many times larger than the abstracts, meaning there are many options for what could be included in these abstracts and many possibilities for things to go a different path than the human who wrote it would have intended. The last issue I wish to address is that I do not have punctuation in the summary. I had some trouble visualizing in my mind how I would do that, as I had removed punctuation in the preprocessing step. For this project, I just ignored that issue.

8 Discussions

For future work, I would like to re-incorporate an attention layer and try re-working the embeddings to match the inputted text more than what I used in this project. Another thing I could do to increase the accuracy of my results is focus the categories of the data I was looking at to a field of interest instead of all papers that can be found in the open access corpus.

In Checkpoint 1, I mentioned trying the pointer-generator model. I tried to implement this one as well, but had trouble modelling the beam search algorithm they used for decoding the encoded hidden states. Out of all the models I looked at while researching this topic, this was the one I found to be the most elegant as they included a way.