Long Short Term Memory for Email Subject Generation: A Negative Experimental Result

Gabriel T. Brookman (brookmang@carleton.edu)

James Yang (yangj2@carleton.edu)

Abstract

Generating an email's subject after writing its body can often prove to be a hassle in real life. In this paper, we aim to construct and train several neural networks adopted from Long Short Term Memory (Hochreiter & Schmidhuber, 1997), which take in the body text of an email, letter by letter, and return appropriate words to include in the title. Our model consists of two separate networks, which are trained using real emails from the Enron email corpus (Cohen, 2009). Our model aggregates predictions from both forward and reverse networks and computes their maximum to use as the predicted probability of that word being in the email subject. Our results show that none of our models were effective at generating human-like email titles, nor do they provide any clear trend on how changing the parameters of our models can lead to better results.

Keywords: LSTM; emails; neural network; recurrent; extractive; generative

Background

Have you ever tried to come up with the title for an email, but found that none of the titles you thought of were up to snuff? Don't worry - a lot of people feel the same way. There has been a lot of work done on extractive text summarization for single documents, which email subject generation is an application of. Extractive text summarization (ETS) traditionally involves cropping out salient segments of a text and reconstructing them into a title/summary (Song, Huang, & Ruan, 2018). Some of the first and most prominent work in this field is on automatically generating abstracts of papers by computing the statistical significance of words and sentences (Luhn, 1958). More complicated feature classification models have been proposed, such as a supervised sequence labeling algorithm (Shen, Sun, Li, Yang, & Chen, 2007). Greedy algorithms have also been used to summarize words, such as the revised MMR (Maximal Marginal Relevance) model to classify speech recordings (Riedhammer, Favre, & Hakkani-Tr, 2010).

More recently, other modeling approaches such as neural networks have become increasingly popular among researchers for textual analysis. Neural networks work as a connection of nodes, where some input is fed into the network and undergoes a non-linear transformation to return some output. Then, the errors between predicted and actual output are backpropagated through the network and weights between individual nodes are updated. For example, Nallapati, Zhai, and Zhou (2016) adopted the approach in Shen et al. with a basic recurrent neural network in order to avoid the labor of feature generation, and to facilitate understanding of the results by breaking them up into more abstract features such as word salience. Over the years, more network architectures have been proposed to deal with the time-dependent nature for data such as text, music and speech, etc. In particular, Long Short Term Memory, also known as LSTM, is a recurrent neural network model that has been heavily applied in text-to-text tasks for variable-length input, such as machine translation and speech processing. The input is fed into the network sequentially, and the network stores recent activations from previous input in a "memory cell" to indicate the current state of the network (Hochreiter & Schmidhuber, 1997). Song et al. developed a LSTM-CNN model for abstractive text summarization, which also makes use of another type of structure, Convolutional Neural Networks, to encode text phrases before feeding the resulted phrase vector into the LSTM. Abstractive text summarization (ATS) involves generating new sentences from scratch, which relies on comprehensive parseable English dictionaries in order to combine and create new phrases based on existing ones. However, although ATS is often used on texts such as scientific journals, news articles or speech dialogues, whose summaries do not tend to contain original phrases, email subjects are generally constructed by a few key words that appear in the body of the email, which makes the task more extractive in nature. In addition, email subject generation is often assumed to be more difficult given the variations of emails and higher levels of constraint, since email subjects tend to be succinct and accurate.

Thus, in this project, we aim to construct an LSTM-based neural network for extractive email subject generation. Our neural networks will take the body of the email as input, and return a selection of keywords and their associated probabilities of being in the subject. By learning to predict which words in the email are in the subject, the model will be able to generate a subject when given a novel input. The rest of this paper is structured as follows. Section 2 is divided into several subsections. Subsection 1 outlines the data used in our study and the data processing task that we have conducted. Subsections 2 and 3 explain our LSTM model and our approaches in dealing with stopwords. Section 3 summarizes our training process and Section 4 illustrates the results. Section 5 discusses and analyzes our results, concludes the paper and proposes potential future work.

Methods

Data Processing

Our models were trained and tested using the Enron Email corpus (Cohen, 2009), which was collected from the Enron cooperation and consists of around 600,000 emails from over 158 employees. This corpus of mostly business and work

emails is one of the few publicly available datasets available to use for email-related research.

We pre-processed our data using a reproducible Python script, which incorporated the Pandas and NLTK libraries (McKinney, 2010; Loper & Bird, 2002). To start off, we tokenized the words in each email and removed all white spaces, symbols, punctuation, and numbers from the subject and the body of each email. To ensure that each email could be trained by the network, we removed all emails that contain an empty subject or empty body. We also removed reply and forward emails (i.e., emails with 're' or 'fwd' in their titles), since such emails do not have or need their own subjects to begin with. Then, we removed emails that do not contain every word from their subject line in their body text. Since our model is based on the notion of extractive summarization, it cannot predict the presence of words in the subject line if those words do not exist in the email's body text. In addition, we removed emails with more than 500 words in their body paragraphs (before considering stop words), because training on such long emails can be computationally intensive and often has low returns compared to training on shorter emails, because words in the subject lines of massive emails are typically very uncommon in their body texts due to the size of those body texts. After the data pre-processing, we had a total of 2619 emails to train and test our emails with. We segregated 20 of these for our test set, and used the remaining 2599 for our training set.

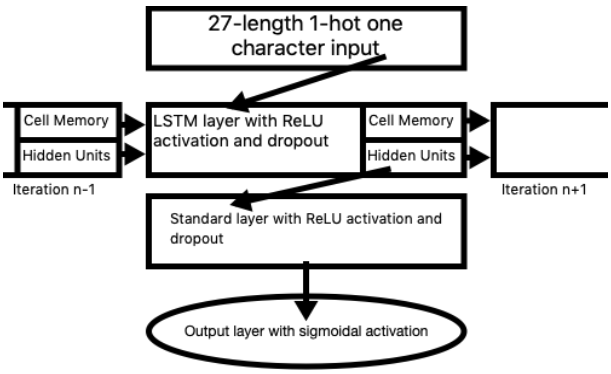
Each character in each email's body text is encoded as a 1-hot vector of length 27, where each index of the vector corresponds to each letter in the ordered alphabet, and the last index corresponds to the white space, so that the network can separate different words. Other characters, such as numbers, punctuation and special characters, are removed from the data. Each email message is broken up and passed in one letter at a time. Inputting letter-by-letter helps keep the input space relatively small, at only 27 dimensions, whereas if messages were encoded on a word-by-word basis, there would be a much higher-dimensional space.

Model Construction

As discussed earlier, neural networks are a popular approach for extractive summarization tasks in general, so it stands to reason that they are likely to also work for email subject generations. Because email body paragraphs can have variable length, we opt to use a recurrent neural network architecture. The network has to be able to learn long-term dependencies since the meaning of the text at one part of an email is heavily dependent on text present in other parts of an email. For that reason, we chose to use an LSTM (Hochreiter & Schmidhuber, 1997).

Our original network was structured to have a single LSTM layer, as (Greff, Srivastava, Koutník, Steunebrink, & Schmidhuber, 2017) found that a single LSTM is relatively resource-efficient while still making for an effective model. On top of that layer, we have a fully connected layer with a smaller number of nodes. For all layers besides the output, we use

Figure 1: Our original architecture



ReLU activation functions. For the output, we use a sigmoidal activation function. After each of the LSTM and hidden layers, we have a dropout layer with a rate of 0.5, applied only to the layer output rather than the memory cells. All of our models are implemented using the PyTorch library in Python (Paszke et al., 2017). The architecture of the network is visually represented in Figure 1.

Each LSTM processes each word letter by letter, and finally, prompted by the space at the end of the word, sends its output through the other hidden layer to the output layer, where it is compressed to a value between 0 and 1 that represents the probability that the model assigns to that particular word being in the subject line of the email.

A binary cross-entropy loss function is used to compute the loss, because it is effective in predicting probabilities. Stochastic gradient descent with a learning rate of 0.01 is used to update the weights.

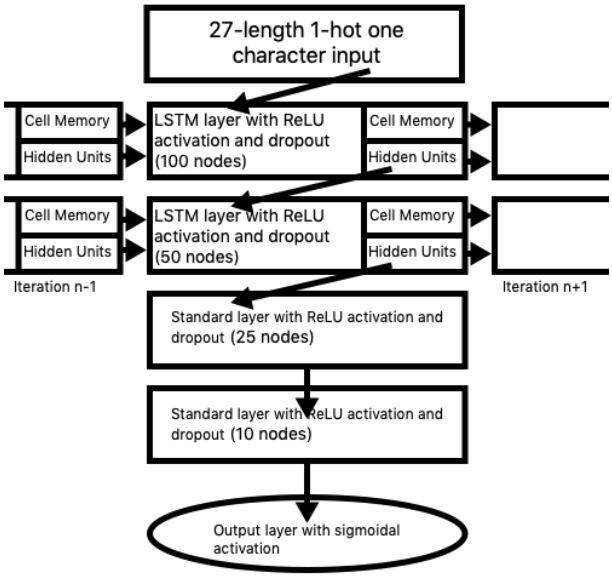
Each version of our model consists of two such networks. One of them performs a backward pass on the words in a message, and the other one performs a forward pass on the words in the message (in both cases, letters in each word are in the same order). By using both passes at once, we can avoid any mistakes that the network might make as a result of its inability to determine the probability that some word is in the title given only either the words before it or the words after it. The forward pass considers words that are immediately after the current one, where the backward pass incorporates ones before the current one, thus allowing both orders to be considered.

The networks output probability of each word in the message being in the subject line. For words that appear multiple times throughout the body of the email, the maximum probability of all occurrences of that word is taken as its probability for each network. Then, the maximum of that word's probability between both the forward and backward pass is chosen as the model's final prediction for that word. The words with the five highest probabilities are chosen, and considered as a bag of words representation of the title.

In addition to our original network, we also constructed one with a slightly more complicated architecture in order to

account for the nuances of this task which may not be captured by the previous model. The new architecture has an additional LSTM layer and an additional hidden layer, after the first LSTM and first hidden layers, respectively. We suspect that the increased number of layers will make our networks more expressive, and thus able to find patterns in the data that the old models were unable to find. Below in Figure 2 is a visualization of our second network architecture.

Figure 2: Our updated architecture



Stopwords

We became concerned with the effects of stopwords, mainly the ubiquitous ones, such as 'it', 'a', 'you', etc. Stopwords may provide additional noise to our model, decreasing accuracy for other words. We attempted two different approaches to deal with stopwords. First, we removed a small set of arbitrarily determined words from the subject and body of all training emails. Then, we trained the model on the remaining words and predicted the data in the test set. With this approach, we can effectively and completely remove the effects of unimportant words from our model. However, it defeated the purpose of using a recurrent neural network, since the context of the email would be removed, and the body of the email was then expressed as a sequence of relatively disconnected words rather than a fluent line, thus breaking up the fluidity of the text. Moreover, when we read over the email messages with stopwords removed, we found them to be indecipherable.

For our second approach, we created a larger set of stopwords based on our observations of the emails in the Enron dataset. We included in the list many more words that we assumed would not contribute to the overall theme of the email, such as all pronouns, prepositions, conjunctions, determiners, etc. However, unlike the first approach, we did not remove these words from the emails. Instead, when we trained the model, we did not update the weights for stopwords, which should prevent the noise that stopwords may cause. This approach lets the stopwords provide more context for the more significant words, instead of disregarding them entirely as in the first approach.

Model Training

We conducted several trials, varying the number of hidden units in both the LSTM and vanilla layers. We used the following numbers of hidden units, among other parameters given our first network architecture, which are displayed in Table 2 below. For models 6 and 7, we used the second approach of dealing with stopwords.

Table 2: Models Parameters given first network architecture

model	LSTM	vanilla layer	epochs	stopwords
1	10	5	10	removed
2	20	10	10	removed
3	100	50	10	removed
4	100	50	10	no loss
5	100	50	20	no loss

Table 3: Models Parameters given second network architecture

model	lstm 1	vanilla 1	lstm 2	vanilla 2	epochs
6	100	50	25	10	10
7	100	50	25	10	20

In addition to exploring the effects of number of hidden units on the results, we also explore the number of epochs to train each model with. In other words, we vary the number of times the network 'sees' the entire training dataset. All of the models above are trained with 10 epochs each, which is limited by the scope of this research, in that 1 epoch can take from 48 minutes to 3 hours for our models above. However, to assess the effects of the number of epochs on the model, model 5 is trained with the same hidden size as model 4, as indicated in the table 2 above, but with 20 epochs instead of 10. We also explore the effects of different methods of processing stop words as discussed earlier, seen in Model 3 and 4.

We also trained 2 models (6 and 7) with our slightly more complicated neural network architecture. Both of them have the same number of hidden units and do not have stopwords removed from the body and subject text, but they are ran for different number of epochs. The model parameters are displayed in Table 3.

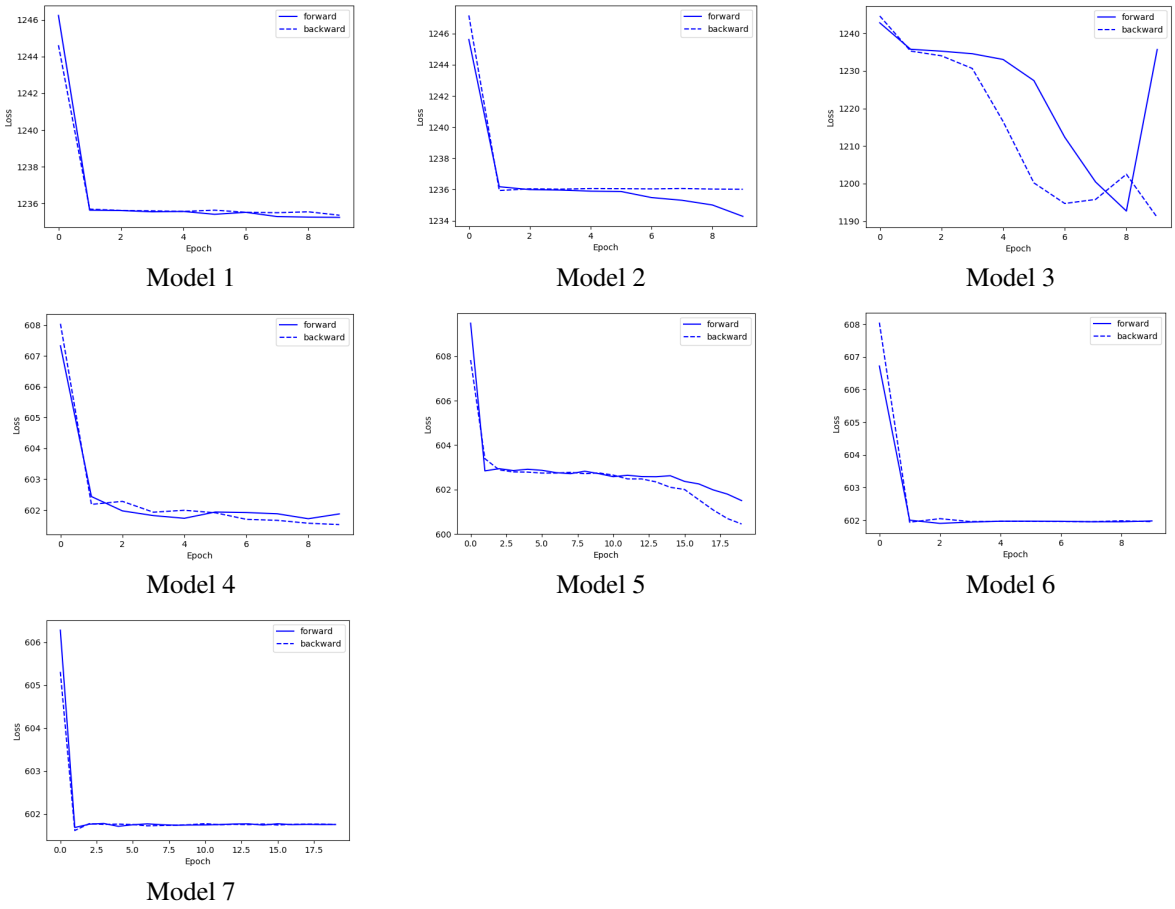


Figure 3: Loss over epochs for all models

Model 1: 10/5, 10 epochs, remove; Model 2: 20/10, 10 epochs, remove; Model 3: 100/50, 10 epochs, remove; Model 4: 100/50, 10 epochs, no loss; Model 5: 100/50, 20 epochs, no loss; Model 6: 100/50/25/10, 10 epochs, no loss; Model 7: 100/50/25/10, 20 epochs, no loss

Results

Each model is trained with the pre-processed Enron email training sample for the corresponding number of epochs. Each epoch results in a total loss value, which is measured by taking the binary cross-entropy between the predicted probabilities that each non-stop word is in the subject versus its actual presence (indicated by 0 or 1) in the email title. Below, you can see loss plots for the forward and backward networks in each model, which indicate how much the models improved over time. The plots are arranged and shown in Figure 3.

To evaluate the predictions of our models on new emails, we used our 20-email test set that we earlier segregated from the rest of the Enron dataset. Then, as discussed earlier, we took the maximum across all occurrences and both forward and backward passes as the final prediction probability for each word. We then ranked every word in the body paragraph of the email by the resulting maximum. The ranking for the predictions of each unique word in the email is provided below in Table 1 for each of our models, given a relatively short sample email from our test set. Words that are in the email subject are color-coded in pink to highlight how each model performs in terms of out-of-sample predictions.

Discussion

According to Figure 3, we generally observe steadily converging total losses for both forward and backward passes given more epochs, specifically in Models 1,4,6, and 7. On the other hand, models 2 and 5 show steady trends at first, but start to diverge near the end of epochs. Model 3, however, shows erratic and divergent behaviors in terms of the total loss. We assume that this could be due to unlucky weight initializations.

As evident in the plots, our models tend to reduce loss by about 10 units during the first epoch, presumably as they learn the frequency of words being in the title and then converge to a constant loss. For some models, the loss does not reach its minimum by the end of training, as seen in the graphs of models 2, 3, and 5. Note that the loss values may differ between models due to the difference in ways that stopwords are incorporated, as explained earlier. Comparing models 1, 2 and 3 side by side, where the only factor is the size of the

Table 1: Ranked prediction of words for all models of 2 test examples

Table 1.A Example 1

Subject:	nat gas market analysis
Body:	attached please find natural gas market analysis today thanks bob mckinney nat gas doc

Model	Predicted Sequence
1	find attached today mckinney market bob nat analysis thanks gas natural doc please
2	today mckinney nat find market attached doc analysis thanks gas natural please bob
3	analysis nat bob today doc thanks mckinney market attached find please gas natural
4	attached find please bob mckinney today doc market analysis thanks gas natural nat
5	attached find please mckinney doc today thanks market analysis bob natural gas nat
6	find analysis thanks today mckinney doc bob please gas nat attached market natural
7	bob doc today gas nat natural analysis attached find mckinney thanks market please

Table 1.B Example 2

Subject:	swap forms
Body:	swap forms moved back into atty fms bank should able access again

Model	Predicted Sequence
1	moved atty should into fms access forms swap bank back able again
2	atty swap into again moved should back bank access fms forms able
3	should into able again fms atty access moved bank back forms swap
4	moved should able fms access forms into bank back atty again swap
5	moved should able into access fms forms bank back atty swap again
6	moved again bank forms fms access into should back able atty swap
7	moved back bank into atty again able fms access should forms swap

hidden layers, it seems that having more hidden nodes leads to less steady trends, which may warrant more epochs in order to stabilize the total losses.

Then, we compare the rankings of words by 3 models on an example test email from the Enron email dataset (with stopwords removed from our ranking), as shown in Table 1.

We find that, using Example 1 of Table 1, models with a higher number of nodes in the hidden layers seem to perform slightly better than smaller models in terms of how highly they rank the actual words in the subject, which are color-coded in pink. However, such differences may be simply a coincidence that is particular to this data, since the difference is small and does not seem to manifest in the generated subjects themselves. In Example 2, the above trend is not observed, in that the predictions seem to get worse with more models. Therefore, no conclusions can be made about the effects of hidden layer sizes given our models and data. Disappointingly, our models with no stopwords removal reach similar conclusions, as evidenced, for example, by models 4 and 5 having four of the first five words in common. However, there is no evidence from comparing these models that our first network architecture has learned to generate human-like email subjects based on the predictions.

Next, we want to assess the effect of the number of epochs on the resulting predictions. First, we compare models 4 and 5, since they have the same hidden layer sizes and deal with stopwords in the same way, differing only in the number of epochs that they train for. It appears that while they behave similarly for their first ten epochs, the loss for model 5 continues to decrease again after a period of stability. However, the change is relatively small per epoch, and does not begin immediately after the initial drop. In terms of their respective predictions on the test set, they result in almost the same predictions for both of our examples in Table 1. Both models fail to give high probabilities to words in the actual subject. The lack of change can be attributed to the insufficient number of epochs at 10 and 20, or that the architecture itself is unable to learn effectively from the data.

We also compare models 6 and 7 as an additional example in assessing the effect of epochs while using the second network architecture instead. The loss graphs show very stable performance after the first epoch in both models. However, we cannot quite say whether the predictions improve after running the model with more epochs, as in Example 1 there shows slight improvement, where in Example 2 shows a significant decrease in performance.

Last not but least, we assess the effect of our two approaches in dealing with stopwords: removing a small set of stopwords versus not computing the loss of a larger set of stopwords. We can compare models 3 and 4, which hold all other parameters the same. As discussed earlier, in Figure 3, Model 3 displays erratic trend compared to Model 4, in terms of the loss. In Table 1, we observe the same dilemma as we did for models 6 and 7, where the improvement of predictions cannot be easily determined by looking at the examples.

Overall, our models return generated subjects that fail to represent the existing email subject and fail to capture the words from the paragraph that humans would choose as key words. In addition, we find that exploring our current parameter space does not result in a consistent change to the quality of the predictions. In other words, no trend has been observed in terms of how we can improve our current models by adjusting the parameters.

Future Work

As discussed above, none of our current models provide decent predictions nor suggest how adjusting the parameters may improve the results. One obvious attempt for improving our networks' performance is to train our models with more epochs. That would also allow us to see what would become of certain models, such as model 3, which did not converge after 10 epochs, and models 2 and 5, which continued to decrease. It is possible that, just from increasing the training time, we could drastically improve the effectiveness of our model.

Also, we can adjust the sizes and quantities of hidden layers. Since the Enron dataset has nearly 3000 messages, we could likely drastically increase the number of parameters without risk of overfitting. In addition, in our current neural networks, the number of hidden units decreases between layers instead of remaining constant because we aim to map a many-dimensional input to a one-dimensional output while also prioritizing models with lower numbers of parameters in order to train our models quickly. With more time to train and/or faster processing speed, we could try switching to an even distribution of hidden units across layers, as is standard, and see if that would result in better predictions.

In addition, we could vary other hyperparameters, such as the dropout rate and/or learning rate, which are currently set at 0.5 and 0.01. Ideally, we would test both constant and fluctuating values in an attempt to optimize them.

There is also much more to be done in terms of text preprocessing. As shown in (Song et al., 2018), 2 Natural Language processing methods stand out as potentially useful: morphological reduction, which reduces auxiliary verbs such as "is, are, am" to their base form, and coreference resolution, which finds and replaces all expressions with what they are referring to in the text. Morphological reduction can potentially reduce the number of different words that the network has to consider, while coreference resolution can help increase the occurrences of keywords that are referred to throughout the text.

Another thing to experiment with is to feed text into the model on a higher level of abstraction rather than just letter by letter, for example, loading words in by their part of speech or frequency in the text. When the input goes letter by letter, the model has to learn how words are constituted and how sentences are fit together, as well as to understand the meaning of the text to at least some degree. With so many low-level features to learn, it is no surprise that the model would perform poorly. When making the trade off between bias and ease of learning, our network architectures made it extremely difficult for the models to learn patterns in any sequence of letters. Furthermore, we could enhance the models to produce coherent, understandable phrases or sentences instead of producing only words. This would likely ensure that the model not to rely on single-word heuristics, such as the heuristic to often include a particular word in the subject line without regard to the context around it.

Finally, given our results, we suspect that neural networks are less effective for email subject generation than we had hoped. As such, it may be worth considering to experiment with other methods for keyword extraction and summarization, such as the heuristic or Bayesian methods described in (Gupta & Lehal, 2010).

References

Cohen, W. W. (2009). Enron email dataset.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222–2232.

Gupta, V., & Lehal, G. S. (2010). A survey of text summarization extractive techniques. *Journal of emerging technologies in web intelligence*, *2*(3), 258–268.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735-1780. doi: 10.1162/neco.1997.9.8.1735

Loper, E., & Bird, S. (2002). Nltk: The natural language toolkit. In (pp. 63–70).

Luhn, H. P. (1958, Apr). The automatic creation of literature abstracts. *IBM Journal of Research and Development*, *2*(2), 159-165. doi: 10.1147/rd.22.0159

McKinney, W. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th python in science conference* (p. 51 - 56).

Nallapati, R., Zhai, F., & Zhou, B. (2016, November). SummaRuNNer: A Recurrent Neural Network based Sequence

Model for Extractive Summarization of Documents. *ArXiv e-prints*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2017). Automatic differentiation in pytorch. In *Nips-w.*

Riedhammer, K., Favre, B., & Hakkani-Tr, D. (2010). Long story short global unsupervised models for keyphrase based meeting summarization. *Speech Communication*, *52*(10), 801 - 815.

Shen, D., Sun, J.-T., Li, H., Yang, Q., & Chen, Z. (2007). Document summarization using conditional random fields. In *Proceedings of the 20th international joint conference on artifical intelligence* (pp. 2862–2867). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Song, S., Huang, H., & Ruan, T. (2018, Feb 16). Abstractive text summarization using lstm-cnn based deep learning. *Multimedia Tools and Applications*.