# Biased Abstractive Summarizer in Python

**Vikrant Sharma**[*]
Department of Engineering
Boston University
Boston, MA 02215
`vikrant@bu.edu`

**Zachary Capone**[†]
Department of Engineering
Boston University
Boston, MA 02215
`zcapone@bu.edu`

## Abstract

Generating summaries from natural text language is a common problem in machine learning. While many machine learning algorithms accomplish abstractive and extractive summaries, most do not include an explicit tonal analysis that enhances the low-level dimensional algorithmic understanding of the text to be summarized. An approach to automatically abstract natural language summaries in Python with tonal analysis is presented. First, the text to be summarized is marked by a tonal polarity bidirectional long-short term memory (LSTM) network. Then, after preprocessing, the text is fed through a sequence to sequence encoder/decoder mechanism with an attention layer that ranks the relative importance of words in the sequence. Finally, the encoder/decoder mechanism's generated summaries of the text are benchmarked relative to the architecture without polarity labeling.

## 1 Introduction

Deep learning methods have recently made advances in adjacent fields to natural language processing (NLP), such as image classification. These advances have been made possible by deep learning networks that create dense vector representations of nuanced features in the input data, automatically accomplishing feature representation learning.

Deep learning frameworks outperform other state-of-the-art NLP tasks like semantic role labeling (SRL), parts-of-speech (POS) tagging, word-embeddings, and named-entity-recognition [2] [17]. Since deep learning frameworks are disadvantaged by the time complexity of their training relative to other algorithms, advances have been made to enhance their core architectures to learn more nuanced patterns more rapidly, such as attention mechanisms, K-nearest-neighbor (KNN) clustering, reinforcement learning and memory-augmenting models. Similar models have proven effective at natural language processing, namely summarization [4].

A unique part of summarization concerns itself with the removal of irrelevant content. Unlike other machine learning applications, where the algorithm is somewhat separated from the end user, summarization algorithms have a direct application to an average consumer [13][6]. For consumers, digging through the reviews of products, restaurants, or services can be fairly tedious. While most websites offer a /5 star rating and these reviews provide a general sense of quality, it can be hard to attain a specific grasp of the issues or benefits at hand unless a consumer queries reviews left by past patrons. For instance, a produce having negative reviews is likely due to a high rate of being broken out of the box. These are specific points of information that become known to the user only when they spend time reading through reviews. A similar application can be found in summaries of text that aren't focused on brevity but instead translation. A similar system can be easily used to translate particular text passages in another language.

[*]Undergraduate student at Boston University.
[†]Undergraduate student at Boston University.

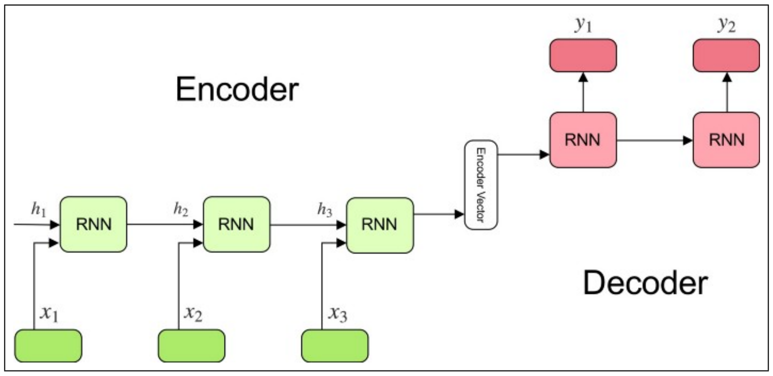


Figure 1: Encoder-decoder sequence to sequence model [9].

Two ways that summaries are generated in machine learning involve abstractive and extractive text summarization. Abstractive text summarization involves creating a low level dimensional representation of the ideas and context of an input text, and then using that encoded abstraction to generate an entirely new sequence of words aiming to describe as much of the larger text as possible in as short a vector as possible. Extractive text summarization comprises clustering of words together, identifying which words are the most important, and "extracting" them from the original text to form a new subset "summary" of the larger input sequence. As an extractive text summarizer does not necessarily include a dimensional representation of all necessary contextual signals and ideas in the input text, they tend to be less extensible and more prone to adversarial examples than abstractive text summarizers, which can encode nuanced features in training that prevent its susceptibility to adversarial validation.

Abstractive text summarization is performed generally by one of two ways. One involves using generative adversarial networks (GANs). GANs appear to be the state of the art for abstractive text summarization for the simple reason that they can model nuanced information well and do not rely on generic or trivial summaries[14] [10]. A generator G and a discriminator D compete in a zero-sum game to both predict and evaluate the given summaries on a particular input text or (often) article. This makes it easier to extract, encode, and infer the meaning of longer sentences, at the expense of the specificity of hyperparameters. Hyperparameters are concerned with the learning rates and architectural parameters of the learning model, and therefore cause GANs to require more human adjustment to be successful than other algorithms. Mode collapse is another issue associated with GANs, when the mapped output doesn't vary similarly to the input. This is similar to the problem of symmetric ciphers in cryptography. With a mode collapsed GAN, many unique input sequences with deservedly disjunct summaries would map to the same output summary [5].

The second way to perform abstractive text summarization is through a sequence to sequence (seq2seq) model. A seq2seq model describes any model that encodes an input into a vector and then replicably decodes that vector into a predicted sequence, either similar to the input or different altogether. For this reason, the performance evaluation metric of a sequence to sequence model is a different metric than the training loss, creating the potential for accumulating errors. Nonetheless, seq2seq models are pervasive in everyday electronics, from automatic closed captioning to language translation[15].

Broadly, seq2seq models comprise of an encoder and decoder, as discussed in Fig. 1. The task of an encoder is to create a smaller dimensional matrix representation of an input string (or text in general). The encoder comprises of several recurrent units, such as long-short-term memory (LSTM), recurrent neural networks (RNN), or gated recurrent units (GRU), that discretizes the input elements of an input sequence and propagates information forward in discrete chunks [7]. The words are represented as $x_i$ in the following formula that determines how hidden states are calculated:

$$h_t = f(w^{(hh)}h_{t-1} + W^{(hx)}x_t$$

This representation is then forwarded to a decoder network that generates a probability distribution of components of an input string as a function of the previous consecutive components in the string. The decoder comprises of a stack of recurrent units predicting an output $y_t$ at discretized time steps. Since the decoder's input is the hidden state from the previous encoder, the decoder produces both an

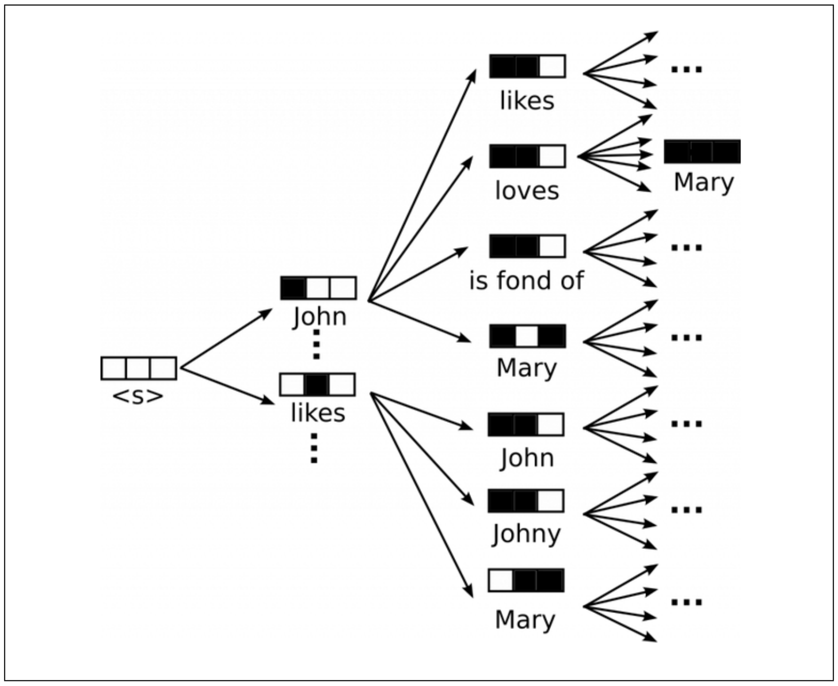


Figure 2: Equivalence relation example for bare encoder/decoder framework without contextual mechanism [11].

output as well as a second hidden state. The decoder's hidden state $h_i$ is similarly calculated using the following formula:

$$h_t = f(W^{(hh)} h_{t-1})$$

At each string component "step," the decoder must make a decision about the most probable next string component "step" and then validate that decision with the actual next string component. While the normalized exponential function (softmax) is often used to determine the probability vector needed to determine the output of the encoder/decoder schema, the rectified linear unit (RELU) is used for faster calculations[1].

Simply selecting the next the next probable word in a sequence for an iterative decoder can create entire equivalence relations for possible words, as demonstrated briefly in Fig. 2. For example, the number of equally probable words after "good" in the phrase "I had a good time at the beach" with a basic linear probability distribution would make a decoder prediction somewhat meaningless. Therefore, to tackle this contextual inelasticity, there's two methods described in the literature. One is by using beam search (and by extension "bucketing"), which predicts the next k-word sequence and selects the first word from the sequence with the greatest probability. However, besides a disfavorable time complexity associated with its training, beam search fails rapidly if required to generate a new sequence of words from a text input that it has never encountered during training [16].

In the interest of efficiency, the second option–an attention mechanism–is more favorable for machine learning applications. An attention mechanism weighs the importance of the previous components of an input string and incorporates those weights into the decoder's probability distribution of next probable words. Bahdanau creates an alignment score that's multiplied with a hidden state vector (the encoder's output) to create a context vector, which consists of most of the information that the decoder needs to build a probability distribution function [3].

The attention mechanism has another advantage over beam search in that while beam search performance significantly favors shorter sequences, the attention mechanism can automatically decompose longer sequences into progressively smaller and smaller components depending on which component structures and individual components take precedence. This implies that the decoder is more easily

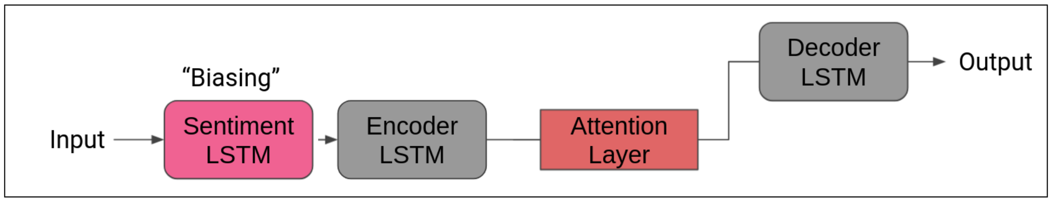


Figure 3: Architecture for proposed abstract summarizer architecture.

able to predict the next word in any sequence since the attention mechanism allows it to vary its algorithmic understanding of longer sequences with their context.

Bi-directional LSTMs are generally effective in seq2seq models since they significantly outperform their uni-directional LSTM counterparts in memory retention despite small training data-sets [19]. While other methods also significantly overfit, bi-directional LSTMs allow for propagating multiple components of input strings simultaneously, allowing the program to adapt more easily to inconsistent input strings, such as that typically found in product or service reviews.

Bi-directional LSTMs are also effective for explicitly predicting sentiment polarity for a given input text sequence. By removing the encoder aspect of the seq2seq model and replacing the encoder with a bi-directional LSTM, one can approach the state-of-the-art in polarity sentiment analysis even in the absence of a regularizing or biasing layer. RNNs are also commonly used for this application, though with notably lesser performance in the literature [12].

The relevant literature often discusses attention mechanisms' inability to successfully capture the tone of input passages. Similar words would be given the same importances across multiple contexts despite the situations clearly calling for a different relative importance[8]. Therefore, it can be hypothesized that a sentiment analysis bidirectional LSTM dedicated to encoding the sentiment of a passage more effectively than the attention mechanism could integrate a tonal representation of the input elastic with its context.

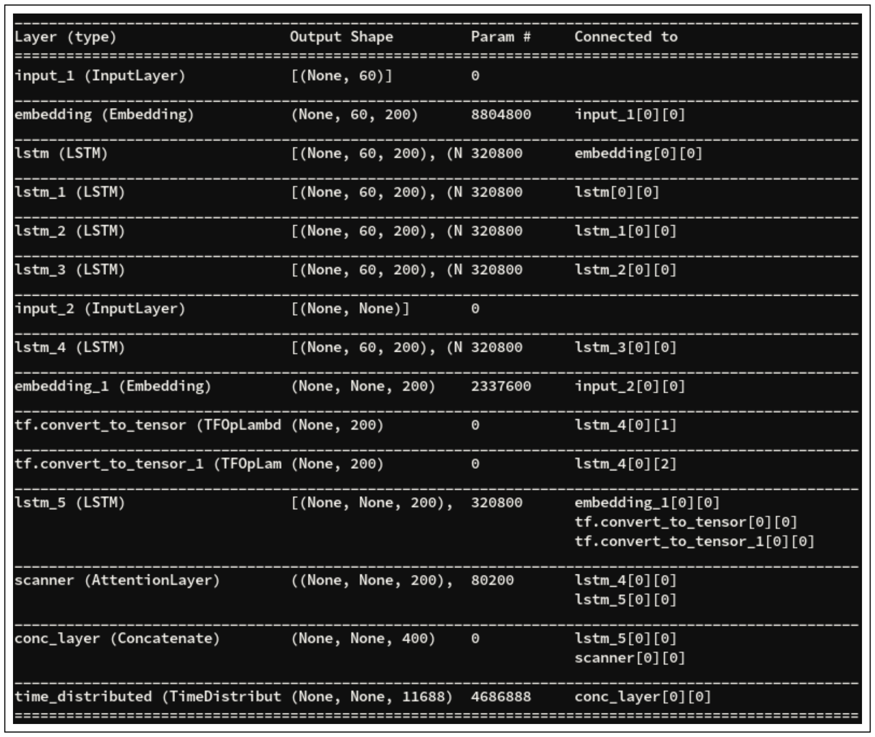
Therefore, biasing the input training summaries with their explicit tones could improve the performance of a seq2seq abstractive summarizer by explicitly programming context signals into the seq2seq's hidden vector representations of the input strings. At an architectural level this may lead to the attention mechanism granting greater weightage to signal words as a function of the tone of the overall input. By combining a bi-directional LSTM to explicitly tag sentiment polarity into the inputs for a seq2seq model buffered with an attention mechanism, we hope to improve on sequence to sequence abstractive summarization performance.

## 2 Methods

We tagged summary training data by a sentiment bidirectional LSTM that evaluated the tone of the original text training data. The tag and input strings were fed into a monodirectional LSTM through which a Bahdanau attention layer created a context vector. The context vector and hidden states were fed into the decoder monodirectional LSTM, which produced its own hidden states and an output. Propagating test data allowed the algorithm to produce new summaries for input text with which we validated the new model versus an identical model without the sentiment analysis.

### 2.1 Pre-processing

Before the data was fed to either algorithm (sentiment or summarizer), all text was separated into its constituent words. The words were mapped for contractions (i.e. "isn't" to "is not), converted to lowercase, filtered for special characters and HTML tags, and cleaned of words in parenthesis (this appeared to bias the algorithm toward more simple context vectors). Punctuation and short syntactical words were also removed. Most data preprocessing was done through readily available python libraries. START and END words were appended to the beginning and ends of the summary training data to signal to the decoder how far from the start or how close to the end it would be in any sequence it was training on.



```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 60)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 60, 200)      8804800     input_1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 60, 200), (N 320800      embedding[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 60, 200), (N 320800      lstm[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, 60, 200), (N 320800      lstm_1[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, 60, 200), (N 320800      lstm_2[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
lstm_4 (LSTM)                   [(None, 60, 200), (N 320800      lstm_3[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 200)    2337600     input_2[0][0]
__________________________________________________________________________________________________
tf.convert_to_tensor (TFOpLambd (None, 200)          0           lstm_4[0][1]
__________________________________________________________________________________________________
tf.convert_to_tensor_1 (TFOpLam (None, 200)          0           lstm_4[0][2]
__________________________________________________________________________________________________
lstm_5 (LSTM)                   [(None, None, 200),  320800      embedding_1[0][0]
                                                                 tf.convert_to_tensor[0][0]
                                                                 tf.convert_to_tensor_1[0][0]
__________________________________________________________________________________________________
scanner (AttentionLayer)        ((None, None, 200),  80200       lstm_4[0][0]
                                                                 lstm_5[0][0]
__________________________________________________________________________________________________
conc_layer (Concatenate)        (None, None, 400)    0           lstm_5[0][0]
                                                                 scanner[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 11688)  4686888     conc_layer[0][0]
==================================================================================================
```

Figure 4: Assembled Summary Architecture.

### 2.2 Training and Testing

The input word sequences for the sentiment LSTM were taken from a series of tweets on kaggle. The input word sequences were tokenized and fed into the bidirectional LSTM from one end, while the sentiments were fed into the other end. The data was used in an 80% train to 20% validation ratio.

The input word sequences for the summarizer were taken from a series of amazon reviews on kaggle. The input word sequences were initially labeled by the bidirectional sentiment LSTM, which were then tokenized by word into a five-stacked LSTM encoder, the last of which being concatenated with a Bahdanau attention layer. The decoder was a single LSTM, and a batch size of 32 was used to train for memory purposes. A randomized sampling of sequence summary text was used in a 90% train to 10% validation ratio. Fig. 4 describes the complete tensorflow model.

### 2.3 Performance benchmarking

A comparison between the validation losses of both our architecture and a summarizer without the sentiment LSTM gauged our overall accuracy. In the interest of time, precision, recall, and bias/overfit analyses could not readily be done. A simple precision analysis was done by comparing the word length of predicted summaries to actual summaries. A qualitative analysis of overfit was also done by comparing sentiment labeling performance between the two kaggle datasets.

Also in the interest of time, training epochs were not arbitrarily fixed but rather programmatically halted once validation loss began increasing; this had the effect of maximizing training performance toward a local maxima at the expense of a potentially greater global maxima. Finally, since the bidirectional sentiment LSTM predicted polarity in sentiment, and not the magnitude of said sentiment, it became feasible to create an accuracy measurement of the tone of the predicted summaries relative to the tone of the actual summaries.

## 3 Results

Our preliminary results on the kaggle datasets are summarized in Table 1. Our network achieves validation losses on par with the state of the art transformer from Google (Pegasus). However, validation losses and accuracy are misleading measurements without context for precision and recall, which the pegasus algorithm is superior.

Table 1: Aggregate performance after 10 epochs (SotA = state-of-the-art)

| Architecture | | |
|---|---|---|
| Type | val_loss | accuracy |
| SotA [18] | 2.1% | 0.979 |
| oldLSTM | 2.26% | 0.9774 |
| newLSTM | 2.19% | 0.9781 |

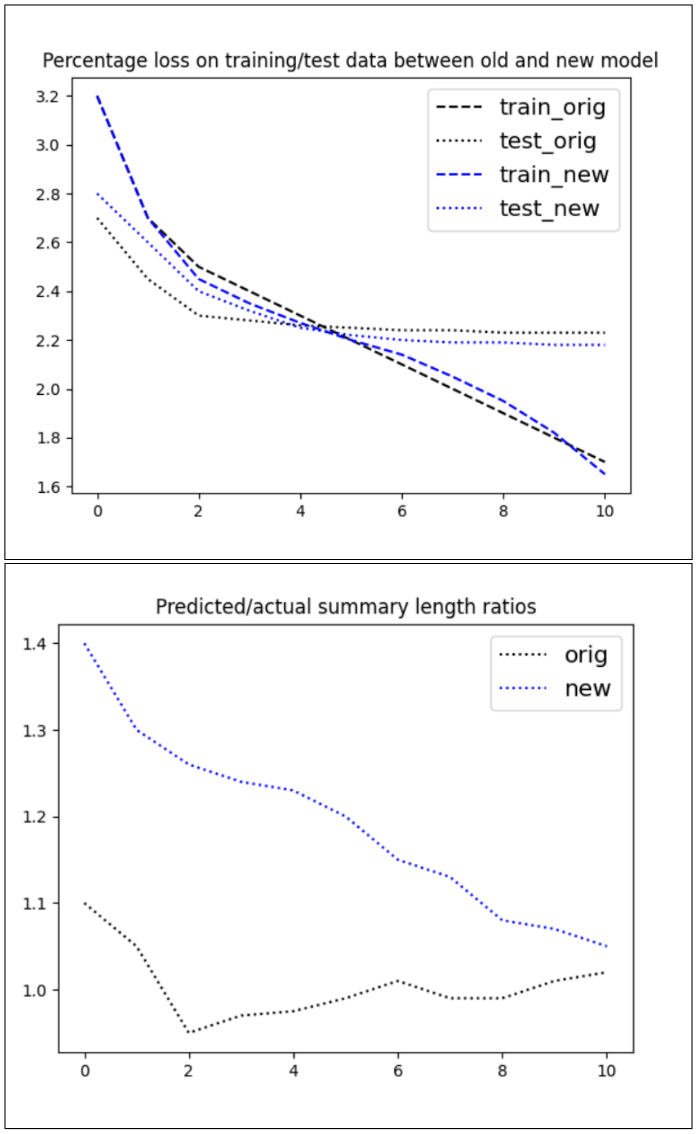
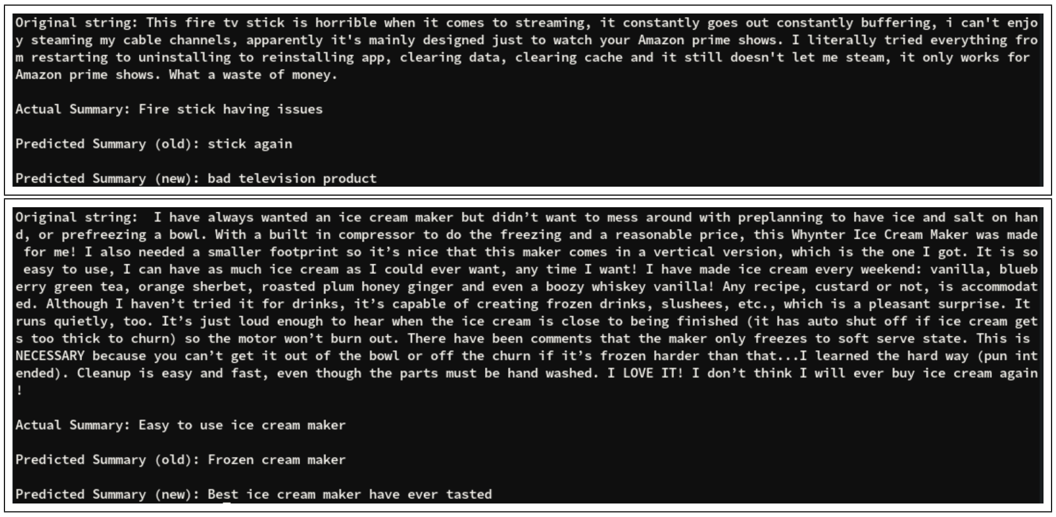


Figure 5: Accuracy (above) and precision (below) metrics as a function of epoch for both algorithms.

Fig. 5 more precisely describes the accuracy metric (above). Note how the validation loss of the new summarizer reaches a lower plateau than the summarizer without the sentiment analysis. This could be due to random error, since time permitted only one training run, but it's also possible that biasing the summarizer toward the passage's sentiment could yield a slightly better performance. Fig. 6 describes examples where our algorithm clearly improves on the prior art.

The ratio of predicted to actual summary length ratios also provided a crude precision metric by comparing how many words were predicted in the summary to how many words were in the actual summary. Interestingly, the summarizer without sentiment analysis converged toward 1.0 with no discernable origin, while the new summarizer maintained a similar validation loss despite having a larger predicted/actual summary length ratio than the original algorithm. Another significant pattern is its origin, as its predicted summaries consistently longer than the actual summaries. This may imply that adding sentiment information lengthens the predicted summaries of the seq2seq architecture, which seems to make sense superficially. There may also be a fundamental flaw in the way accuracy is calculated, since it's a vector distance comparison of the tokenized word strings of the predicted vs actual summaries and is therefore subject to the biases of the internal dictionary representation of the summarizer. More training epochs and different training/test data is needed to even confirm whether this phenomenon is systemic.



```
Original string: This fire tv stick is horrible when it comes to streaming, it constantly goes out constantly buffering, i can't enjo
y steaming my cable channels, apparently it's mainly designed just to watch your Amazon prime shows. I literally tried everything fro
m restarting to uninstalling to reinstalling app, clearing data, clearing cache and it still doesn't let me steam, it only works for
Amazon prime shows. What a waste of money.

Actual Summary: Fire stick having issues

Predicted Summary (old): stick again

Predicted Summary (new): bad television product
```

```
Original string:  I have always wanted an ice cream maker but didn't want to mess around with preplanning to have ice and salt on han
d, or prefreezing a bowl. With a built in compressor to do the freezing and a reasonable price, this Whynter Ice Cream Maker was made
 for me! I also needed a smaller footprint so it's nice that this maker comes in a vertical version, which is the one I got. It is so
 easy to use, I can have as much ice cream as I could ever want, any time I want! I have made ice cream every weekend: vanilla, blueb
erry green tea, orange sherbet, roasted plum honey ginger and even a boozy whiskey vanilla! Any recipe, custard or not, is accommodat
ed. Although I haven't tried it for drinks, it's capable of creating frozen drinks, slushees, etc., which is a pleasant surprise. It
runs quietly, too. It's just loud enough to hear when the ice cream is close to being finished (it has auto shut off if ice cream get
s too thick to churn) so the motor won't burn out. There have been comments that the maker only freezes to soft serve state. This is
NECESSARY because you can't get it out of the bowl or off the churn if it's frozen harder than that...I learned the hard way (pun int
ended). Cleanup is easy and fast, even though the parts must be hand washed. I LOVE IT! I don't think I will ever buy ice cream again
!

Actual Summary: Easy to use ice cream maker

Predicted Summary (old): Frozen cream maker

Predicted Summary (new): Best ice cream maker have ever tasted
```

Figure 6: Sample inputs with actual and predicted summaries for both the old and new summarizer implementations.

## 4 Discussion

Since the program ultimately doubles as an open ended text analyzer, it is capable of easily being adapted or appended upon to perform other functions. Of those possible, we take particular note of our program to have additional precision, recall, bias/overfit analysis, which would equate to more accurate analysis of text and thus returns that more closely resemble what you would expect to summarize in your own mind after reading a review, or even what you would expect to see written down by another person. Additionally, we plan to give attention to the notion of adapting the encoder LSTMs we had utilized with bi-directional LSTMs, to further increase the accuracy in both our readings as well as our predictions. Thought was also given to the idea of incorporating an attention mechanism for full length sentences, much for the same reason.

In addition to running a factor analysis to determine random vs systemic biases in our algorithm's performance, we also plan on comparing our sentiment analysis bi-LSTM with third party sentiment analyzers, such as a KNN classification schema. Doing so would allow us to see how precise our program was in comparison to its competitors, and give a better gauge for just how precisely our program can detect and comprehend different nuances within sentiments. The information retrieved from these comparisons would likely prove useful were the potential adaptations above pursued.

## References

[1] "ACTSMLT: Automatic Classification of Text Summarization using Machine Learning Technique". In: *International Journal of Engineering and Advanced Technology Regular Issue* 9.2 (2019), pp. 5445–5457. DOI: 10.35940/ijeat.b2993.129219.

[2] Cecilia Ovesdotter Alm, Dan Roth, and Richard Sproat. "Emotions from text". In: *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing - HLT 05* (2005). DOI: 10.3115/1220575.1220648.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural machine translation by jointly learning to align and translate*. May 2016. URL: https://arxiv.org/abs/1409.0473.

[4] Jaime G. Carbonell, Ryszard S. Michalski, and Tom M. Mitchell. *AN OVERVIEW OF MACHINE LEARNING*. June 2014. URL: https://www.sciencedirect.com/science/article/pii/B9780080510545500054%5C.

[5] Heather Cole-Lewis et al. "Assessing Electronic Cigarette-Related Tweets for Sentiment and Content Using Supervised Machine Learning". In: *Journal of Medical Internet Research* 17.8 (2015). DOI: 10.2196/jmir.4392.

[6] Dritjon Gruda and Souleiman Hasan. "Feeling anxious? Perceiving anxiety in tweets using machine learning". In: *Computers in Human Behavior* 98 (2019), pp. 245–255. DOI: 10.1016/j.chb.2019.04.020.

[7] Jiatao Gu et al. "Incorporating Copying Mechanism in Sequence-to-Sequence Learning". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2016). DOI: 10.18653/v1/p16-1154.

[8] *High performance text recognition using a hybrid convolutional-lstm implementation*. URL: https://ieeexplore.ieee.org/document/8269943/.

[9] Simeon Kostadinov. *Understanding Encoder-Decoder Sequence to Sequence Model*. Nov. 2019. URL: https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346.

[10] Ankur Parikh et al. "A Decomposable Attention Model for Natural Language Inference". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (2016). DOI: 10.18653/v1/d16-1244.

[11] Faizan Shaikh. *Seq2Seq Model: Sequence To Sequence With Attention*. May 2020. URL: https://www.analyticsvidhya.com/blog/2018/03/essentials-of-deep-learning-sequence-to-sequence-modelling-with-attention-part-i/.

[12] Xingjian Shi. *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting*. URL: https://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach-for-precipitation-nowcasting.pdf.

[13] Grigori Sidorov et al. "Empirical Study of Machine Learning Based Approach for Opinion Mining in Tweets". In: *Advances in Artificial Intelligence Lecture Notes in Computer Science* (2013), pp. 1–14. DOI: 10.1007/978-3-642-37807-2_1.

[14] Hao Tang, Song Bai, and Nicu Sebe. "Dual Attention GANs for Semantic Image Synthesis". In: *Proceedings of the 28th ACM International Conference on Multimedia* (2020). DOI: 10.1145/3394171.3416270.

[15] Gilles Madi Wamba and Nicolas Gaude. "Auto-Lag Networks for Real Valued Sequence to Sequence Prediction". In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series Lecture Notes in Computer Science* (2019), pp. 412–425. DOI: 10.1007/978-3-030-30490-4_33.

[16] Sam Wiseman and Alexander M. Rush. "Sequence-to-Sequence Learning as Beam-Search Optimization". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (2016). DOI: 10.18653/v1/d16-1137.

[17] Tom Young. *Recent Trends in Deep Learning Based Natural Language ...* URL: https://arxiv.org/pdf/1708.02709.pdf.

[18] Jingqing Zhang et al. *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*. July 2020. URL: https://arxiv.org/abs/1912.08777.

[19] Yonghua Zhu et al. "A Bi-Directional LSTM-CNN Model with Attention for Aspect-Level Text Classification". In: *Future Internet* 10.12 (2018), p. 116. DOI: 10.3390/fi10120116.