Report: Text Summarization using Deep Learning

Rushikesh Gaidhani
rushikeshgaidhani@knights.ucf.edu

Venkatasiva Bharath Talluri
bharath3794@knights.ucf.edu

Pranjal Bahuguna
pranjal.bahuguna@knights.ucf.edu

Abhishek Choudhari
abhishekc97@knights.ucf.edu

ABSTRACT

In this paper, we describe the way in which, an 'Abstractive Summarization' of Amazon Fine-Food reviews was made possible by using Recurrent Neural Networks and an Encoder-Decoder architecture, more commonly known as a sequence-to-sequence model. The model was implemented using Keras API, through TensorFlow, as Keras keeps the model modular, composable and easily extendable, while TensorFlow inherently provides the access to lower-level APIs. The training phase uses encoder and decoder that folds the input sequence to an informative vector and unfolds in the decoder phase to generate the output. Teacher forcing technique has been incorporated too. The evaluation of generated and reference summary is calculated using the ROUGE metric of evaluating summaries.

Keywords

RNN, Encoder-Decoder, Seq2Seq, GRU, LSTM, TensorFlow, Keras, Rouge etc.

1. INTRODUCTION

Over the course of time, humans have relied on writing as one of the medium of communication, alongside other mediums such as verbal/spoken medium, visual/image medium. The abundance of information thus generated is tremendous. Content that an individual could possibly comprehend, within a reasonable amount of time, far exceeds the actual time in hand. Written media that exist in the world are many, however in this modern age, text data is predominantly available digitally, in various forms. Some of them being stories, documents, news articles, social media posts etc, and it keeps on growing. With loads of data in hand, it is quite a task to go through and also comprehend it's meaning. If we are able to concisely represent this information, into a summarized form, we would be able to complete more work in lesser time. Being able to develop a Machine Learning model that can automatically summarize the data and understand its core contextual meaning, will help us to digest the information faster.

With the advent of modern day techniques of Deep Learning for Language Processing, we can hand over the bearing task of 'Summarizing' data, over to the computers. Text summarization aims to give a short, accurate and fluent summary from the given data. Automated summarizers are usually less biased than human's summarizing any text. However that too depends on what kind of data is fed to the Automated model but, it can be taken care of by careful selection and considering every possible aspect. A summarizer should determine the most salient sentences, make a coherent readable summary, while at the same time minimize number of references to ideas and entities not present within the scope of the given data. Text summarization falls under many categories. Here we are concerned with the category of Abstractive vs Extractive techniques. Other categories would be discussed in brief, in the related work section.

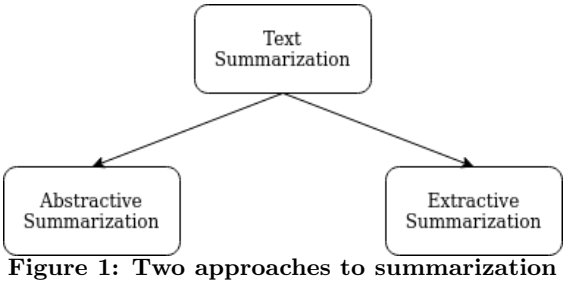

Figure 1: Two approaches to summarization

There are two approaches for text summarization under one category, either summarize through extraction or through abstraction. Summary through extractive method revolves around the ability to dig out or extract from the whole corpus and concatenating them together in a readable way. They are 'excerpts' taken directly from some input.

Abstractive techniques are closer to what we see when high level human skills of inferring and combining perspective's kicks into action. It is also worthwhile to note that Natural Language 'Understanding' is the ultimate goal of Natural Language Processing and Artificial Intelligence.

The extractive way is generally an approach which is overly verbose and biased towards certain sources. Summarizing through sentence ranking and not employing linguistic models to capture language complexities does not perform as well, as compared to the abstractive approach.

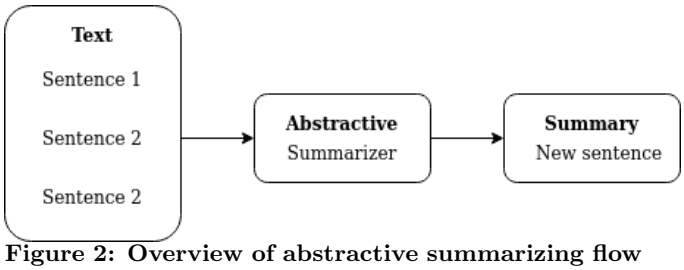

Figure 2: Overview of abstractive summarizing flow

The Abstractive method understands the context and tries to generate sentences that may or may not be in the original corpus. Thus gives concise results close to human-like having all the critical information.

The goal of this project is to obtain a summarized version of food item reviews, while preserving the sentiments being conveyed and penalizing the features that misrepresents the polarity of the reviews. For this, we chose Amazon's Fine Food corpus as Amazon, is an e-commerce giant that has millions of products listed online and each product has unique attributes like Product Id, Rating, etc. Reviews with a wide distribution of opinions make this corpus suitable for our summarization model.

The dataset has reviews from the years 1999 to 2012. We can see a little bit of temporal dynamic at play here. What that means is that there could have been recent additions to the categories, items, recipe's in some of the food items, leading to either an improvement or decline in the food quality itself. This factor can easily change a person's opinion on whether they like or dislike that product. Another aspect is that, new kinds of foods might have been added to the inventory, which can mean more variations in/additions in the vocabulary of the whole dataset.

2. RELATED WORK

There have been several papers on extractive summarization. The first work was published by Luhn[1] in the 1950s proposing the idea of describing the content using words counts. But, we know that words with higher accuracy doesn't necessarily depict the idea behind the main topic. But recent advances in neural networks motivates researchers to work on the more human-like abstractive summarization. There have been several papers such as recurrent neural network encoder-decoder proposed by Cho et al. [2], an extension of RNN encoder-decoder with attention mechanism by Bahdanau et al. [3] and bi-directional RNN with LSTM's in encoding layer and attention model in decoding layer by A. K. Mohammad Masum [6] which we are referring for this project.

The different categories that are related to summarization are as such,

1. Single Document vs Multi document:
Single Document based summarizer relies on one single piece of cohesive data source. Whereas a Multi-Document summarizer avoids repeating of information from the different sources while keeping it short and to the point.

2. Indicative vs Informative:
Indicative means when we want to know about the data, in a objective manner only, and the matter relevant to a particular topic itself, and nothing surrounding it. Informative summarization presents all the facts revolving a topic, which might be something additional or unnecessary.

3. Summarizers based on document length and type:
A summarizer can definitely perform well on a ideal sized piece of data, but it will have trouble when it needs to deal with several pages long data. In that case it would be good to use techniques like hierarchical clustering and discourse analysis additively.

3. METHODOLOGY

3.1 Data Collection:

The dataset for this problem is taken from Stanford Network Analysis Project (SNAP) which hosts a large network dataset collections. The dataset on which we are working consists of fine food reviews listed on Amazon [www.amazon.com] from October 1999 to October 2012.
The statistics of the dataset are as follows,

Number of reviews	568,454
Number of users	256,059
Number of products	74,258
Users with >50 reviews	260
Median no. of words per review	56

The datasets is available in two format, SQLite and CSV. Few examples of the dataset are as follows,

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

Figure 3: Sample dataset features

3.2 Data Preprocessing:

The dataset has almost 500,000 reviews each with 10 attributes which are of Integer and String type. The definition of these data attributes is given as,

Id	Row Id
ProductId	Unique identifier for the product
UserId	Unqiue identifier for the user
ProfileName	Profile name of the user
HelpfulnessNumerator	Number of users who found the review helpful
HelpfulnessDenominator	Number of users who indicated whether they found the review helpful or not
Score	Rating between 1 and 5
Time	Timestamp for the review
Summary	Brief summary of the review
Text	Text of the review

For our training dataset we performed exploratory data analysis to analyze the importance of other features. But, we didn't get much information using other features as our main task here is to generate the summary for a given text and the other features weren't helpful to generate the new summary. Hence, out of the above 10 attributes we select 2 features, that is *Summary* and *Text* as our train data attributes. For training we use 548,000 samples whereas we test on 2,500 samples of text to generate summaries.

```
count    568454.000000
mean         81.005522
std          80.807102
min           2.000000
25%          33.000000
50%          57.000000
75%          99.000000
max        3525.000000
```

Figure 4: Distribution of Text (review) data

The distribution of Text in the form of total number of words is given above. We observe that there are total 568,454 reviews, out of which some reviews are with maximum of 3525 words and minimum of 2 words. Similarly, the distribution of total number of words in the Summary are shown below. There are 568,427 summaries with maximum 41 words some summary and minimum 0 words in some summaries.

```
count    568427.000000
mean          3.128462
std           2.619420
min           0.000000
25%           1.000000
50%           3.000000
75%           4.000000
max          41.000000
```

Figure 5: Distribution of Summary data

From the above data distribution of *Summary*, we analyze that the *Summary* has minimum word count as 0. So, it means there are missing values in the *Summary*.

```
Id 0
ProductId 0
UserId 0
ProfileName 16
HelpfulnessNumerator 0
HelpfulnessDenominator 0
Score 0
Time 0
Summary 27
Text 0
```

Figure 6: Distribution of null values in data

As mentioned in the above figure, we don't see any missing values for 8 features but, we have 16 missing values for *ProfileName* and 27 missing values for *Summary*. We drop these respective null value rows from our dataset such that we don't have any null or missing values in our dataset which we need feed to our model.

As we finalized the *Text* and *Summary* as our features, we pre-process these two features. We convert *Text* and *Summary* data into lower case in order to maintain the uniformity between the train data. We have also removed the punctuation from both the *Text* and *Summary*. We have added delimiters like <START> and <END> to signify the start and end of a sentence which we need for our model to understand the start and end of each sentence to start or finish the operation. These modified sentences we convert into space-separated sequences of words. These sequences are then split into lists of tokens, which will then be indexed or vectorized. We encoded these modified *Text* and *Summary* features into sequence of integers in order to represent word into numeric values. As when we convert these words into their respective numeric format we don't have uniformity in their size so, we added pre-padding to *Text* with 0.0 and post padded 0.0 to *Summary* as well.

3.3 Our Approach:

For text summarization or language generation application, given a sequence of words with certain input lengths, we have to predict or generate a summary that is less than the original input length. As input text can be of varied length, we may also arise a problem with generating a summary that can be of different lengths. But this is not possible with the standard RNN language model. So, we used a Seq2Seq model which consists of an encoder and a decoder with different RNN units.

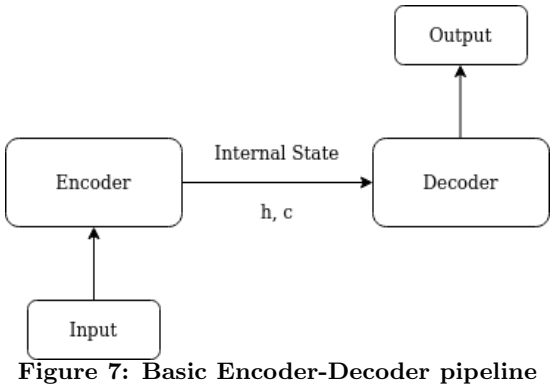

Figure 7: Basic Encoder-Decoder pipeline

The above figure shows simple encoder-decoder. The main intuition is that we are trying to fold the input sequence into a small but useful and informative vector at the encoding stage, then at the decoding stage, we are unfolding that compressed vector to a new output sequence.

- **Encoding Stage:** At first, we find embeddings for the input sequence and then we pass this embedded sequence to RNN unit of the encoding stage. Each hidden state will generate an output that can be passed as an input to the next hidden state. Each hidden state takes the output from the previous hidden state and current word embedding as input. But, we will not make any word predictions at the encoding stage while we are passing through each hidden state. The last hidden state in the encoding stage generates an output h_t based on learning from the previous hidden states. We will only keep this output of the last hidden state h_t thus ending up with a fixed size vector compared to the entire input sequence. This thought vector is the compact vector representation of the original input sequence with more useful information from the input text.

- **Decoding Stage:** At the decoding stage, we have a completely new RNN unit with its own weights compared to the encoder RNN unit. The output of the last hidden state in the encoding stage i.e. previously generated fixed size thought vector is passed as input to initial hidden state (h'_0) of RNN unit in the decoding stage. For this reason, we must maintain the same size as of thought vector while passing it to the initial hidden state of decoder RNN. We add some delimiters,

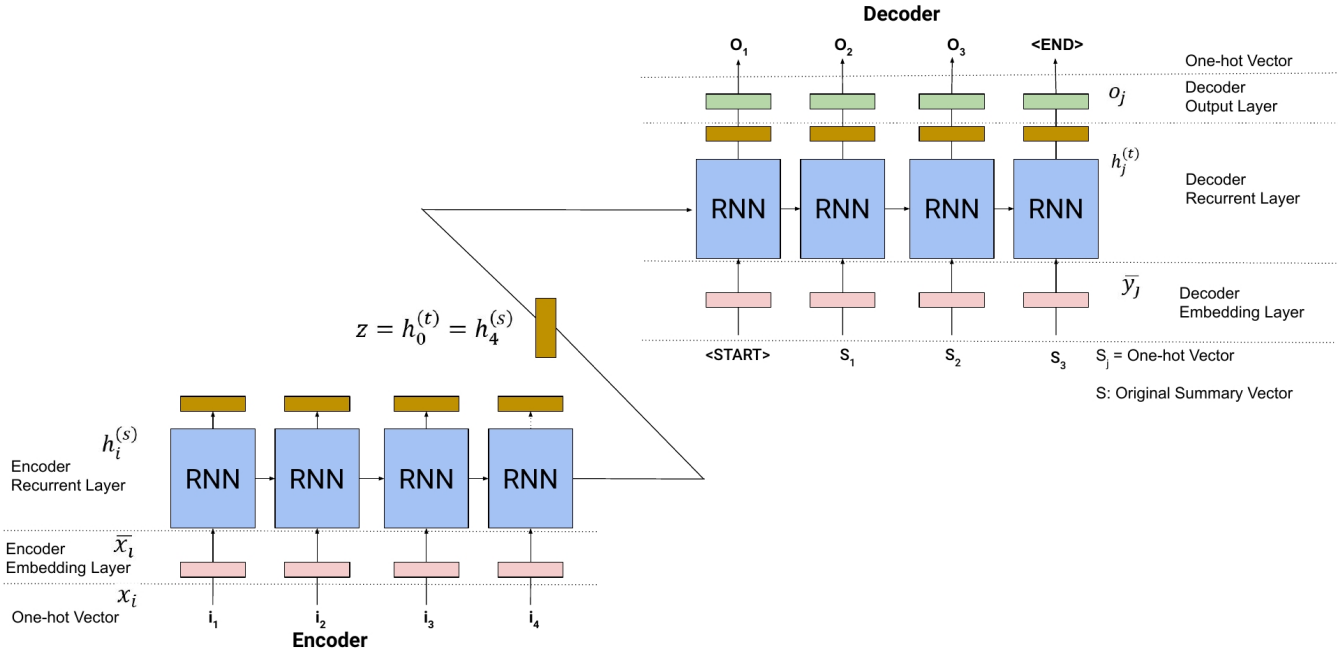

Figure 8: Encoder-Decoder Architecture with RNN Units on Training Stage

for example, <START>, <END>, as a start of a sequence and end of a sequence to tell our decoder where it can start and where it can finish generating the output. We have done this at the preprocessing stage by adding <START> and <END> to the start and end of the 'Summary' column from the Fine Food Reviews dataset. This <START> delimiter and thought vector from the last hidden state of encoder RNN are passed as input to the initial hidden state of decoder RNN. The initial hidden state generates an output and this output is passed through a softmax layer to predict the word. We use two different methods for training and testing to get improved performance.

- **Method used while Testing:** Now we send the above-predicted word from the last hidden state of encoder RNN as input to the next hidden state along with the output of the previous hidden state. To summarize, at each hidden state we process the information based on previously predicted word along with the previously hidden state output to generate current hidden state output which is then sent through a softmax layer to predict the current word. But the disadvantage with this type of learning is that if once we went wrong in predicting at some state the future predictions will also be affected as we are passing the previously predicted word as input to current hidden state.

- **Method used while Training:** So, we used a concept called Teacher Forcing and it is of the intuition that instead of using the previously predicted word as the input to the current hidden state why don't we use the real target word of the previous state as input to the current hidden state while training. With this, we can ensure that the hidden state can take advantage of the target word as an input thus generating more relevant next word as output irrespective of how badly we predicted the previous word. But, this can only be done for training and while testing we have to switch back to the old method of passing the previously predicted word as an input to the current hidden state.

3.4 Implementation and Training:

We used Keras to implement the above mentioned method. The problem with Keras is that it can only work with constant sized sequences for every single model. But the above implementation requires two different models, one model to fit for the training as the size of the sequence we pass to model in this training stage is equal to the number of words for that input text in the real 'Summary' column of the Fine Food Reviews dataset. Suppose an input text has 'n' words in the original summary with respect to that particular input, then the decoder input length would be of 'n'. Another model for the testing, as the size of sequence at this stage, is only one because we can only predict one word at a time in each time step thus decoder input length in this stage is one. So we created two different models in Keras for training and testing stages.

Till now we represented each word as an array of numbers. But these numbers mean nothing to the model and it also doesn't tell the context of that word. So, the first step we do is to represent each word as a word embedding so that we can get the contextual representation of that word. The pre-trained embeddings such as GloVe, Word2vec are trained on a large corpus of data and have high performance and can be easily implemented but they have their own caveats. Since these embeddings are trained on generic datasets, they show lower performance on corpus related to a particular topic. We implemented our model using FastText pre-trained embeddings but got a low score which is understandable as our dataset was particularly related to food reviews. Thus, our model uses a self-trained word embeddings learned through

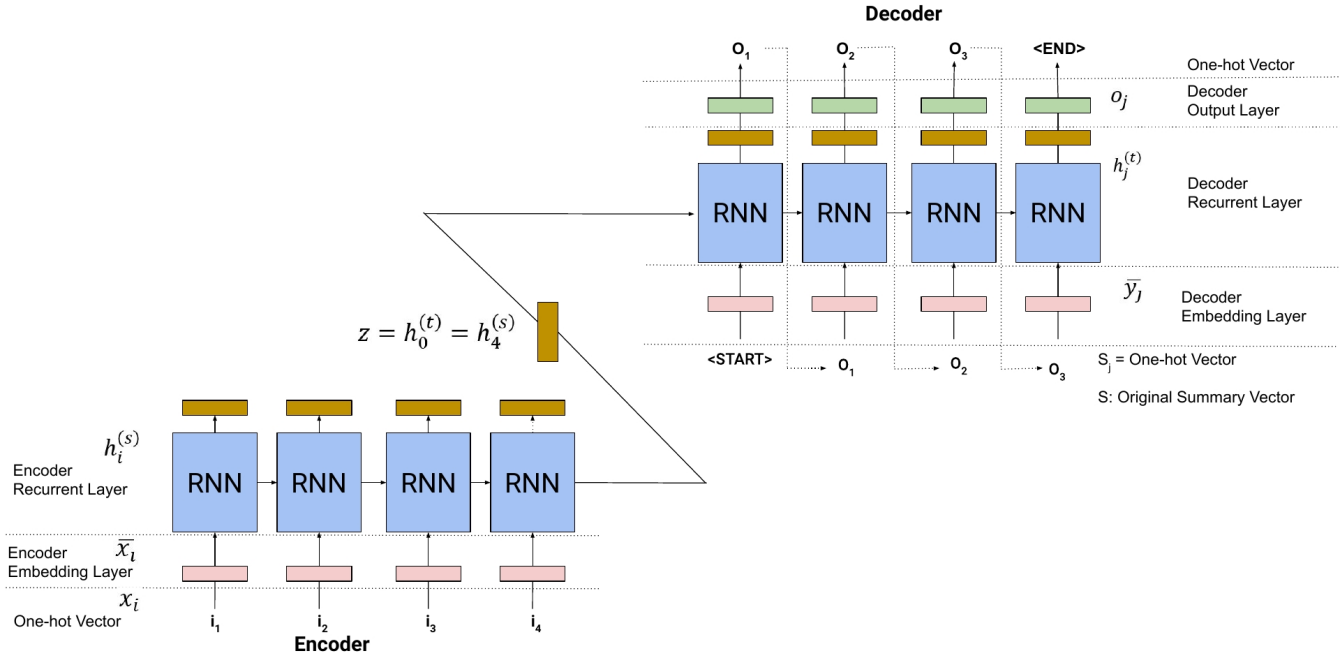

Figure 9: Encoder-Decoder Architecture with RNN Units on Testing Stage

a neural network which is by default in Keras. This can better find the context of words used and generate related summaries.

In our self-trained embeddings, each word has been represented by a 300-dimensional vector to get a richer syntactic and semantic meaning of the words used in the reviews. 300 is chosen as higher dimensions don't mean it is more capable of gathering more information and a lower value increases the chances of losing the information. Moreover, the 100-300 dimensional size has been widely selected as the default embedding size for each word. This vector contains all the similar words learned through the neural network that has a similar meaning to the word in context. This is useful while generating summaries to diversify the vocabulary without losing the context.

The word embeddings are then batch-normalized to standardize the inputs so that for each mini-batch, the distribution of the inputs that we are giving to a specific layer doesn't change over time. The model uses a sparse categorical cross-entropy loss function which generates a single value as output. This is much needed since we are predicting a single word one by one and an integer that can be directly mapped to a particular word to our sequences generated during the pre-processing stage. The loss function or the cost function finds the difference in the predicted and the target values. More is the loss, less is the accuracy. To minimize this loss, the optimizer is used which backpropagates the loss generated and evenly distributes throughout all the nodes according to their weights. Thus the initial weights are adjusted to better predict the outcome which minimizes the loss. These adjustments are minute and are done with each epoch to increase the chances of reaching the local or global minima faster by converging. This is controlled by the learning rate which we set as 0.001 provided as a parameter to the optimizer. The optimizer used for this is Nadam. This optimizer is superior to Adam and uses a Nesterov accelerated momentum which helps in descending the gradient at a much faster rate than Adam optimizer.

The main intuition is that we are trying to fold the input sequence into a small but informative vector at the encoding stage so that at the decoding stage, we are unfolding this compressed vector to a new output sequence. In both the training and testing phase, we are using the same architecture of encoder and decoder but with a slight variation in the decoder phase. Both the encoder and decoder are using GRU layers as GRU uses less training parameters and therefore uses less memory, execute faster and train faster than LSTM. The caveat of using GRU is it is not better than LSTM in remembering the long term dependencies. In the training stage, the GRU layers read the entire input sequence where, at each time step, one word is fed into the encoder. The input sequence is in the form of integers that were mapped to words during pre-processing. But the model requires embeddings of these to make sense from it. So, each word's embedding is passed to GRU layers to generate the hidden state. This hidden state is passed on to the next layer. This process where the output of each hidden layer recursively passes on to the next layer is done until a final compressed vector form of the input is generated by the last hidden layer. This output is then fed to the first layer of the decoder as input as well as the "start-of-sequence" token that we have inserted in human summaries during the pre-processing stage. This "start-of-sequence" token tells the decoder to start the process. The first hidden layer generates the output by taking this token and the hidden output from encoder passes through softmax and predicts the first word.

The concept of teacher training a.k.a Teacher Forcing is applied at this place when the second layer of the decoder at the training stage has to predict the word. Instead of using the previously predicted word as the input to the next hidden state, we will pass the real target word that the previous

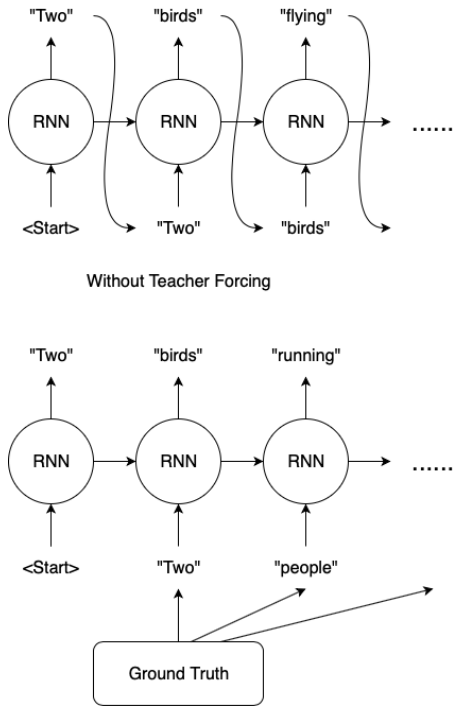

Figure 10: Teacher Forcing Technique

layer should have predicted. So in this way, even if the previous layer didn't predict the word correctly, we are passing the correct word to the next hidden state. This increases the accuracy and trains faster by converging fast.

At the testing stage, the encoder works in the same way by taking the input, and compressing it is a form of an informative vector of the input which can be passed to the decoder. The decoder again takes the start of sequence token and the output of the encoder passed to make the first prediction of the first word. But this time, since we don't have the target human summaries, the output of the first layer is fed to the second layer for the prediction of the next word. This is done recursively until the layer generates the end of sequence token.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Decoder-Input (InputLayer)   [(None, None)]          0
_________________________________________________________________
Decoder-Word-Embedding (Embeddi (None, None, 300)    12858600
_________________________________________________________________
Encoder-Input (InputLayer)   [(None, 2961)]          0
_________________________________________________________________
Decoder-Batchnorm-1 (BatchNorma (None, None, 300)    1200
_________________________________________________________________
Encoder-Model (Model)        (None, 300)             72705000
_________________________________________________________________
Decoder-GRU (GRU)            [(None, None, 300),     541800


_________________________________________________________________
Decoder-Batchnorm-2 (BatchNorma (None, None, 300)    1200
_________________________________________________________________
Final-Output-Dense (Dense)   (None, None, 42862)     12901462
=================================================================
Total params: 99,009,262
Trainable params: 99,007,462
Non-trainable params: 1,800
```

Figure 11: Model summary

4. EVALUATION

We used ROUGE (Recall-Oriented Understudy for Gisting Evaluation) as an evaluation metric. Our implementation of ROUGE is based on ROUGE-1, ROUGE-2 and ROUGE-L in particular with measures such as Recall, Precision and F-1 Score.

- **ROUGE-1:** This checks for the unigram overlap of words between machine generated summary and reference summary.
- **ROUGE-2:** This checks for the bigram overlap of words between machine generated summary and reference summary.
- **ROUGE-L:** This measures the longest matching sequence of words between machine generated summary and reference summary.
- **Recall:** Recall measures how many n-grams in the reference summary are also present in the machine generated summary.
- **Precision:** Precision measures how many n-grams in the machine generated summary are also present in the reference summary.
- **F-1 Measure:** F-1 measure is a combination of both precision and recall.

We tested on 2500 unseen reviews on our model and generated their summaries. These generated summaries along with original reference summaries are passed to ROUGE implementation. Our results of the experimented model are given below. Even though ROUGE is based on recall, it can

Evaluation metric	Recall	Precision	F-1 Measure
Rouge-1	0.13406	0.17806	0.14136
Rouge-2	0.03906	0.05224	0.04079
Rouge-L	0.13435	0.17825	0.14212

Figure 12: ROUGE Evaluation Measures of our Model Generated Summaries

be more useful for extractive summarization where it selects parts of the original text and generate a summary based on this copied parts. For abstractive summarization, we try to generate new language based on understanding of the original text. With new language used in generated summary, it is possible that we can have different word grouping giving the same meaning. So ROUGE may not accurately represent the performance of the model that uses abstractive summarization. Even though we used ROUGE as a metric for the reason that there is no other reliable metric that can make comparisons with other existing models possible.

In general, human evaluation is considered as the gold standard in summarization tasks of Natural Language Processing. But as we all know that, the perception on which humans base their judgements to decide whether the generated summary is a good or bad summary changes from person to person. Additionally, implementing human evaluation is time consuming and expensive and because of the limitations with respect to resources we have, we cannot perform human evaluation. Thus, we limited ourselves to ROUGE implementation as an evaluation metric.

Tested Sample:
Original Text: This is a tasty drink that is all natural. However, a small 8.3 ounce can contains a relatively high 140 calories which is roughly the same number of calories in a 12 ounce Coke. Only, a Coke is about 50% larger in volume.I like sweet, but to me this is a bit too sweet. It might be nice as a mixer in an adult beverage though. I haven't tried to mix it, but you could probably make a nice fruity alcoholic drink out of it. The price isn't a bargain. At nearly $1 per can, as of this writing, it wouldn't seem to be a compelling value. I do like it, but I doubt I'd purchase it again. It also left a bit of a lingering metallic aftertaste that I wasn't thrilled about. Still, if you don't mind the calories, enjoy a beverage that is sweet and are looking for a natural drink, then this could be worth considering.
Reference Summary: tasty but relatively high in calories
Our Model Generated Summary: tasty but not a great value

The original text talks about the drink being tasty but relatively it is high in calories and the product is not that worth it for the given price. Firstly, our model generated a fluent summary that is human understandable. Secondly, it is good in capturing what the actual text is referring to for example, it remembered that the product is tasty and it is not of a great value which tries to mention that the given price is not that worth it. But, it actually missed the calories part of the original text. This can be a good example of decent summarization. But the recall score for this machine generated summary based on ROUGE metric is 0.3333 while at the same time human score can be much more. This shows that we cannot simply rely on ROUGE as a metric for models that are based on abstractive summarization.

5. IMPROVEMENTS AND DISCUSSION

Seq2Seq Model has its own limitations. The encoder-decoder architecture forces us to encode the entire input sequence into just one small thought vector. This type of encoding hurts the performance whenever we have input sequences that are long. Thus, all input sequences irrespective of their length are mapped to the same sized compact vector representation. Instead of limiting decoder RNN unit to use only last hidden state information from the encoder RNN unit, we may try to use the information from all the hidden states of the encoder RNN unit.

As some research papers mentioned that a hybrid model implementation that combines both abstractive and extractive summarization techniques can give better results, it might be a good idea to extend this model by including extractive summarization part. Even though sequence2sequence model with attention systems are good at generating a fluent output that is human understandable, it performs badly in copying the words from the original text thus limiting the implementation of hybrid model. We can add copy mechanisms to the developed model which uses seq2seq model and attention system and makes it easier to copy the words from original text to summary. Thus, a hybrid model allows both copying and generating text resulting in a more effective summarization technique.

As cross-entropy loss alone faces exposure bias problem because of its word-level cost function, we can implement Reinforcement Learning technique using evaluation metric like ROUGE and combine it with already existing loss function in training stage to improve the performance of the model. We can define a Reinforcement Learning loss term which takes the entire word sequence of the generated summary into account and calculate the reward by comparing the generated summary with the reference summary using a standard evaluation metric like BLEU, ROUGE etc. And overall, it tries to maximize the reward based on this evaluated score leading to good performance. Additionally, we can define a hyper parameter for the overall loss function that controls the contribution of both cross-entropy loss and reinforcement learning loss while training. This type of implementation tries to achieve better results by computing reward with the help of reinforcement loss and on the other hand, it also ensures better fluency and readability with the help of cross-entropy loss.

6. CONCLUSION

The abstractive approach for text summarization is the state of the art method as it is able to provide summaries closer to human references. The use of GRU layers is efficient, faster, and complements abstractive summarization. Other approaches can also be looked for either by including multilayer bi-directional GRU's or using bi-directional LSTM if long term dependencies are our main goal and computation power is not a bottleneck. The corpus is trained on up to 0.55 million samples but tested on 2500 samples which is a limitation. Proper testing on more number of samples can help to dig out the inaccuracies that might be creeping in. Human evaluation could have been done on the summaries as it is always a reliable metric. Moreover, the ROUGE implementation performed on abstractive summarization isn't a good metric to rely on. This metric is biased towards extractive summarization and since it finds the overlap between the predicted and reference summary, the score could be less but still, the summary can make sense with respect to the context of the input.

As the type of reviews that are being presented on Amazon food products changes overtime, the deep learning model also needs to adapt to this temporal change. For example, there might be new food products that become available on Amazon and the type of language that is used in reviews might be changing over years, so we can say that there is a temporal nature involved with this type of dataset. By taking it as a practical scenario, it would be better if they have provided the timestamp for each sample because we can train the model on much older samples and try to validate on much newer samples so that after deploying this model, it can still perform well on unseen or future reviews.

7. WORK CONTRIBUTION

Source code link: https://bit.ly/2S5G8NT

Data Preprocessing + Coding	Testing	Evaluation	Analysis
Rushikesh	Pranjal	Abhishek	Venkatasiva
Venkatasiva	Abhishek	Pranjal	Rushikesh

Name	Contribution (%)
Rushikesh Gaidhani	30
Venkatasiva Bharath Talluri	30
Pranjal Bahuguna	20
Abhishek Choudhari	20

8. REFERENCES

[1] Luhn, H. P. (1958) The automatic creation of literature abstracts,IBM Journal of Research and Development,vol. 2, no. 2.

[2] Bahdanau, D. et al. (2015) Neural Machine Translation by Jointly Learning to Align and Translate.Pro-ceedings of the 2015 Conference on International Conference on Learning Representation (ICLR).

[3] Cho, K. et al. (2014) Learning Phrase Representations using RNN Encoder Decoder for Statistical Ma-chine Translation.Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing(EMNLP)

[4] J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. WWW, 2013.

[5] Lin, Chin-Yew. 2004. ROUGE: a Package for Automatic Evaluation of Summaries. In Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004), Barcelona, Spain, July 25 - 26, 2004.

[6] A. K. Mohammad Masum, S. Abujar, M. A. Islam Talukder, A. K. M. S. Azad Rabby and S. A. Hossain, "Abstractive method of text summarization with sequence to sequence RNNs," 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 2019.

[7] Figure 8, 9: Encoder-Decoder Architecture, https://docs.chainer.org/en/stable/examples/seq2seq.html

[8] SNAP, Web data: Amazon Fine Foods reviews https://snap.stanford.edu/data/web-FineFoods.html

[10] Teacher Forcing https://towardsdatascience.com/what-is-teacher-forcing-3da6217fed1c

[11] Automatic text summarization https://blog.frase.io/what-is-automatic-text-summarization/