Abstractive Text Summarization with LSTM using Beam Search Inference Phase Decoder and Attention Mechanism

Ruchir M. Patel
M.Tech, Computer Engineering,
Sankalchand Patel University,
Visnagar, India
ruchir.patel0001@gmail.com

Prof. Ankur J. Goswami
Assistant Professor, Computer Engineering,
Sankalchand Patel University,
Visnagar, India
ajgoswami.ce@spcevng.ac.in

***Abstract*—Volume of textual data has been being increased tremendously for last few years. Such increase is because of rapid growth in technical era. Especially, in the field of product review, lengthy and large number of reviews are available. An automated tool is required, which can transform lengthy review into brief summary; containing overall idea behind the review. Such automatic text summarization may appear as a boon for humanity. Abstractive text summarization is very useful in summary generation. It considers context of the information and generates non-verbatim sentences. In this paper, we are going to focus on using beam search decoder during inference phase with linear normalization and LSTM in encoder-decoder sequence-to-sequence model. Attention mechanism is used to improve processing speed by focusing on certain part of the review sentence. Result shows that the summary generated using beam search decoder is more precise than summary generated from normally used greedy decoder.**

***Keywords*—Long Short Term Memory, Sequence-to-sequence model, abstractive text summarization, attention mechanism, linear normalization**

***Abbreviations*—LSTM, Long Short Term Memory; RNN, Recurrence Neural Network; GRU, Gated Recurrence Neural Network**

I. Introduction

In current era, information is considered as an inevitable asset. Nowadays, plenty of data is available on the internet, which is being raised day by day, rapidly. According to International Data Corporation, measured data circulating the globe during 2013 was approximately 4.4 zettabytes, which will be elevated to 180 zettabytes in 2025 [1]. Therefore, there will be need to shorten text in order to reduce overheads. Additionally, short text reduces searching time, reading time, and augments total quantity of details that can apt within the same area. In this scenario, it is very vital to have a mechanism, which eliminates repetitive as well as less significant data from the pile of information. The resultant data thus obtained may still contain plenty of information, which can be reduced further into just glance of the information, called summary; revealing the view of writer. This task of summarization becomes very challenging, since it includes natural processing of data. To perform such task in machinelike manner, machine has to capture all available raw data and generate semantically correct sentence pattern.

A. Text summarization

Text summarization is a technique, which is used to shorten long pieces of textual information. The prime intention behind it is to generate fluent and consistent summary of the text, indicating main points of the document.

B. Abstractive text summarization

Abstractive Text Summarization identifies prime concept behind the text, along with vital information [2]. This collected information is used afterwards to produce fresh non-verbatim sentences comprises of important information to represent summary of text.

C. Artificial Intelligence

Artificial intelligence refers to providing ability to machine to think like human beings using logic. It simply means to make computers intelligent enough artificially to make them think in the same manner as humans do with the help of logical reasoning.

D. Machine Learning

Machine Learning is a sub branch of Artificial Intelligence, which allows machines to be able to train itself without any explicit information.

II. Proposed Methodology

A. Dataset

- Amazon food product review

There are various Amazon dataset available on the internet. Ours is a dataset containing 5,68,454 reviews of Amazon food products, which were collected in years of timespan. Reviews are consist of wide range of styles.

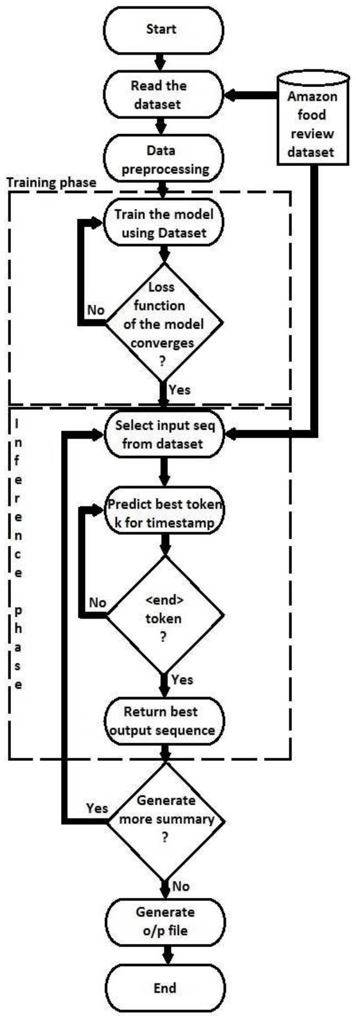
B. Data cleaning

Usually, datasets have a lot of uncleansed text. Steps performed under data cleaning are as below [3]:

1) Transform every uppercase letter into small caps letter
2) Eliminate all HTML badges
3) Perform contraction mapping in order to map compact words into their respective full forms
4) Eliminate apostrophe(‘) s
5) Eliminate all kind of texts located inside of every brackets
6) Remove punctuation marks and special symbols
7) Remove stop words and short words
8) Add special tokens START and END at starting and ending of summary

Figure 1 shows flowchart of the text summarization process.

Fig.1. Process flowchart

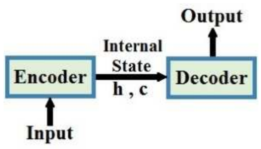


Start
Read the dataset
Amazon food review dataset
Data preprocessing
Training phase
Train the model using Dataset
Loss function of the model converges ?
No
Yes
Inference phase
Select input seq from dataset
Predict best token k for timestamp
<end> token ?
No
Yes
Return best output sequence
Generate more summary ?
Yes
No
Generate o/p file
End

C. *Encoder-Decoder Architecture*

Encoder-Decoder architecture is used mainly in solving such sequence-to-sequence problem, where input sequence length and output sequence length varies [2]. In case of text summarization, input sequence is a long sequence of words, while output sequence is a short sequence of text containing summary. As shown in figure 2, sequence-to-sequence model has two prime components: Encoder and Decoder.

Fig.2. Encoder-Decoder Architecture

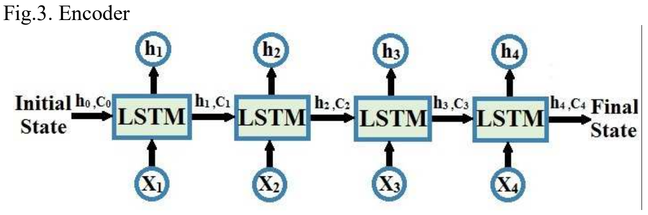


In our model, LSTM is used for encoder and decoder in training phase, because of its ability to overcome problem of Vanishing Gradient by having long term dependencies [2]. During training of certain deep learning neural networks, gradient of loss function vanishes to nearly zero, which hardens training of network. This problem is called Vanishing Gradient. LSTM eliminates this problem.

There are two phases to set up Encoder-Decoder:

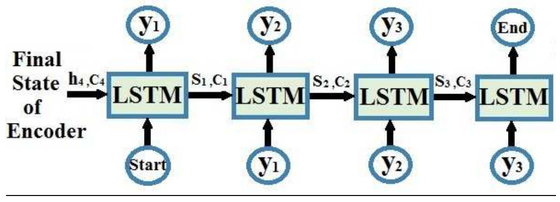
1) Training Phase: In this phase, at first, encoder and decoder are set up, followed by training of model to predict target sequence per timestamp [3].

Fig.3. Encoder



During model training, network considers inputs to proceed execution starting with first layer to last layer via numerous of hidden slabs, returning outcome with the help of output layer. This task is called as Forward Propagation or Forward Pass. It compares result with actual output. Then, in order to reduce value loss, loss at output is found, and weights are updated to minimize error resulting from each neuron. This process of improving result is called Backward Propagation or Backward Pass. One combined round consisting of forward propagation and backward propagation is known as one training iteration, called Epoch [4].

Fig.4. Decoder

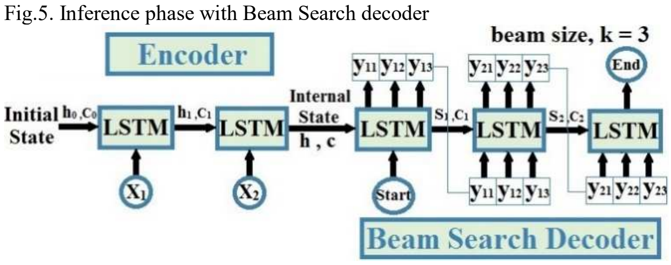


Encoder: At each timestamp, one word is fed into encoder LSTM as shown in figure 3, which processes the information provided at every timestamp and captures contextual information. Encoder LSTM converts source sentence into rich fixed length vector representation, which is provided as initial hidden state to decoder.

Decoder: h_i and c_i of last timestamp of encoder are provided to decoder to initialize. Decoder understands context of the sentence and predict next word based on previous words as shown in figure 4.

Appending <start> and <end> tokens at the target sequence makes it ready to be fed to decoder. During test sequence, target sequence is unknown, so prediction of target sequence gets started by passing <start> token, and ends as soon as <end> token is encountered.

Fig.5. Inference phase with Beam Search decoder

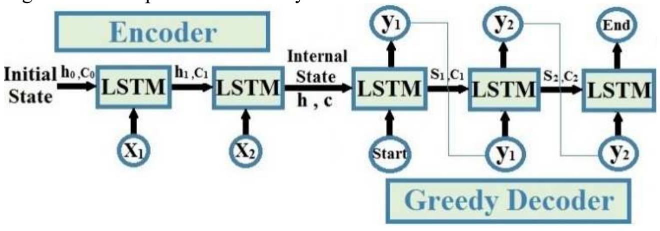


2) Inference Phase: After training phase, new source sequences are fed into model, for which target sequence is unknown. So, inference architecture is set up to decode test sequence. Here, we use beam search technique for decoding the test sequence as shown in figure 5, and compare it with existing widely used greedy approach. Steps of beam search technique to decode test sequence:

i. After successful completion of execution of encoder with the help of input sequence, pass output of encoder as input to decoder in order to initialize it.

ii. Pass first token <start> as input into decoder to begin execution at first timestamp.
iii. Execute decoder for a timestamp for given input with its respective internal states.
iv. Output will be probability for next word. Out of all probabilities, consider top-k (k=beam width) possibilities containing words as output for respective timestamp.
v. Pass sampled all top-k words to decoder one by one as an input for next timestamp and link internal states with their respective k words.
vi. Repeat steps iii to v until <end> token is encountered for all top-k probability words.

Fig.6. Inference phase with Greedy decoder



Beam search functions as shown in figure 5. Consider k as beam width, beam search approach considers k output with maximum probability at present first timestamp, then processing of next second timestamp is performed considering each k output of present first timestamp as input. Now, during execution of the second timestamp, each input generates k outputs. So, after completion of the second timestamp, there will be total k x k outputs. Out of these k x k outputs, k outputs with maximum probability are considered as output for this second timestamp. These k outputs of second timestamp will be considered as input for next third timestamp. This process will be repeated for further timestamps in recursive manner till <end> token is encountered for all k outputs. Once done, one output with maximum probability is discovered out of k output sequences thus obtained, which is observed as resultant output sequence for given input sequence using beam search approach.

Moreover, commonly used greedy decoder for inference phase is shown in figure 6. It is different than beam search strategy and easy to implement, which only considers single output with highest probability at each timestamp. Thus, output generated from greedy decoder considers less possible sequences at every timestamp. Thus, beam search decoder in inference phase is better, since it considers k (k=beam size) number of best outputs at each timestamp. In other words, beam search provides more options to decoder to select from at higher timestamp.

D. *Length Normalization in beam search strategy*

In beam search technique, at each step, probability of an output sequence is calculated by multiplying probability of chosen word with probability of present sequence from previous timestamp. It is a fact that probability lies between 0 and 1. As a result, total probability reduces when probability of new word is multiplied with existing probability. As a consequence, if length normalization is not used in beam search before comparing probability, then output with minimum length gets selected, every time. To overcome that scenario, length normalization is used at each step before comparing probabilities of all possible outputs. Moreover, in order to reduce complex multiplications of probability, log of probability is used. After applying log function, it simply requires addition of log value of probability of new word into existing log value from previous timestamp. Resultant value is further divided by timestamp value, T to get final normalized value of probability of decode sequence at timestamp, T. This makes probability of decode sequences of different length normalized, which allows unbiased comparison of decode sequences with different length. In other words, resultant value obtained for different length output sequences can be compared with each other to get output sequence with highest probability. Equation of probability of output sequence at timestamp, T with length normalization is shown below:

Probability of output sequence =

$$\frac{1}{Ty}\sum_{t=1}^{Ty} \log P(y^{<t>}|x, y^{<1>}, \dots , y^{<t-1>})$$

where, P = probability,

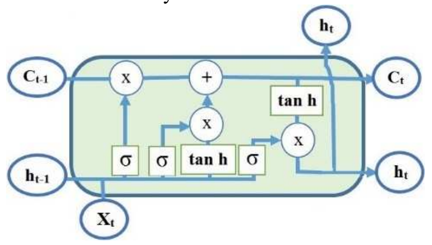
T_y = timestamp of given word y for given output sequence,

$y^{<t>}$ = word y of timestamp t

E. *LSTM Architecture*

LSTM consists of memory blocks, called cells [5]. One such cell is shown in figure 7. After completion of execution of each cell, two internal states (Cell state and Hidden state) of present cell are transferred to next cell. There are three gates in the mechanism.

Fig.7. Long Short Term Memory Architecture



1) *Forget Gate*: Forget gate is responsible to remove less vital information from cell state [5]. The details, which are not required to understand, are eliminated in order to optimize performance of the network.

2) *Input Gate*: The function of input gate is to augment details to cell state [5]. During this state, it is ensured that information added is relevant, and not redundant.

3) *Output Gate*: The job of output gate is to sort out essential information from current cell state, then display sorted vital information as an output.

F. *Limitation of Encoder-Decoder architecture*

As decoder considers whole input sequence to forecast, the model is useful for just short sequences. The encoder finds it abstruse to retain large sequence within fixed length vector [2].

G. *Attention mechanism (Solution of limitations)*

To overcome limitations of Encoder-Decoder architecture, concept of attention mechanism is useful [2]. It aims to focus certain parts of the sentence, while ignoring some. The mechanism is performed by assigning weights to each words, then processing words accordingly.

III. IMPLEMENTATION

Implementation code of beam search approach is given in the Appendix. In implementation, the size of beam is considered as a variable, named as beam_size. So, the code is generalized for all beam size. The code is written in python.

IV. RESULT

Figure 8(a) and figure 8(b) represents training loss and Validation loss per epoch during training phase and testing phase for same dataset considering reviews of 100000 and 568455, respectively. From both the figures, it is seen that training loss is more than validation loss at the beginning of both the graphs; such situation is called underfitting. After certain epochs, validation loss surpasses training loss, which means validation loss becomes greater than training loss, thereafter. Such situation is called overfitting.

It is clearly visible, that validation loss is alleviated upto 2.0553 for 100000 reviews, and upto 1.9821 for 568455 reviews, as indicated in figure 8(a) and (b).

The training of model is continued until two consecutive increase in validation loss is encountered, which returns results as shown in figures 8, 9, and 10. Figure 9(a) indicates that accuracy during testing phase gets improved upto 64.55% for 100000 reviews. The same is elevated upto 64.97% for 568455 reviews as shown in figure 9(b).

Fig.8(a). Training/Validation loss per epoch during training and testing phase for 100000 reviews

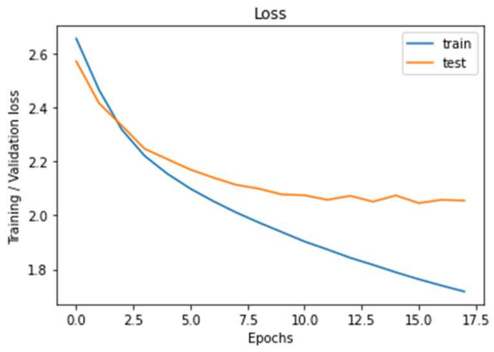


Fig.8(b). Training/Validation loss per epoch during training and testing phase for 568455 reviews

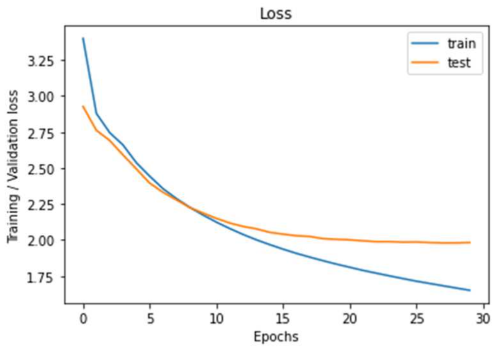


Fig.9(a). Accuracy out of 1 per epoch during training and testing phase for 100000 reviews

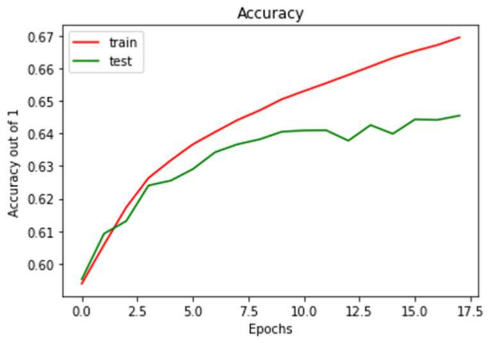


Fig.9(b). Accuracy out of 1 per epoch during training and testing phase for 568455 reviews

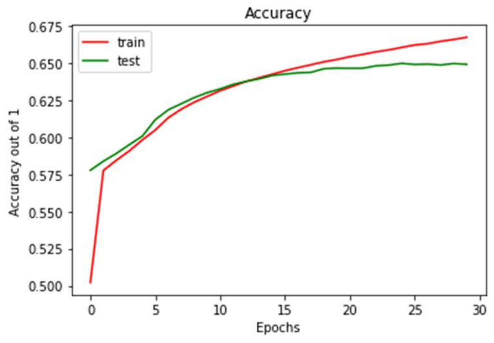


Fig.10(a). Actual time taken to execute each epoch for 100000 reviews

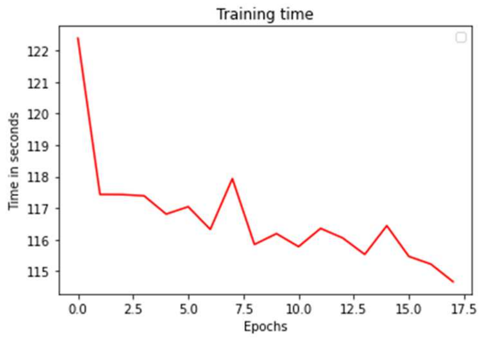
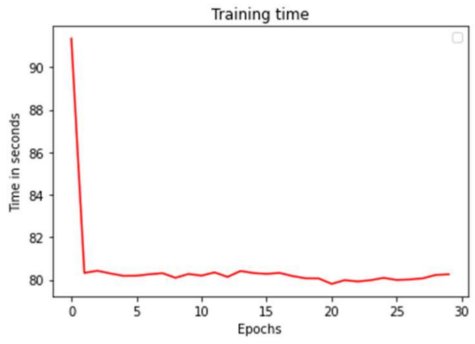


Fig.10(b). Actual time taken to execute each epoch for 568455 reviews

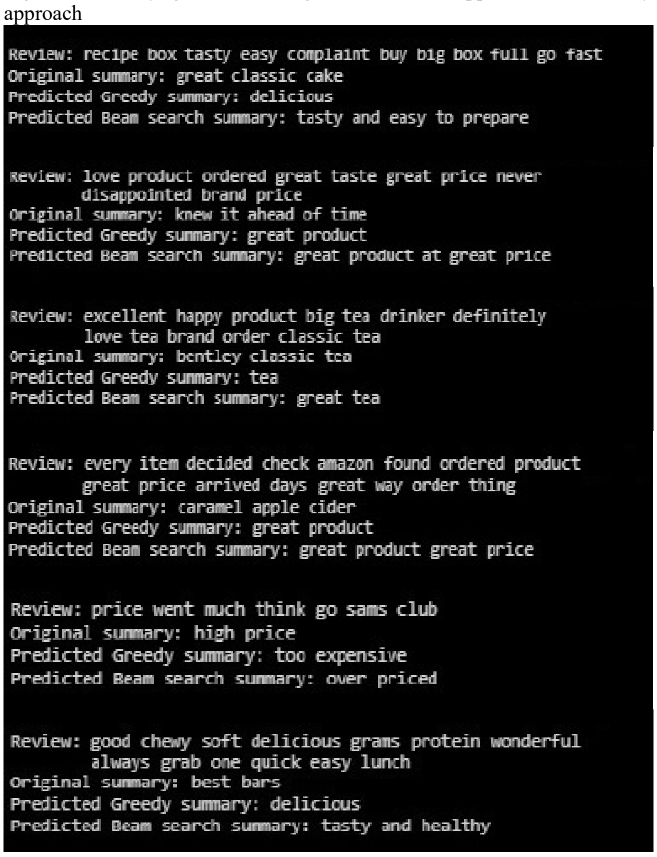


During training of model, batch size is increased from 128 for 100000 reviews to 2048 for 568455 reviews, which results in reduction of training time in the same system with GPU acceleration. As shown in figure 10(a), time required to process 100000 reviews was nearly 115-118 seconds per epoch with 128 batch size; while time required to process 568455 reviews was around 800 seconds per epoch for same batch size. Later, the batch size was changed for 568455 reviews from 128 to 2048, which reduced training time to around 80-81 seconds per epoch as displayed in figure 10(b).

By considering all above behaviour, we can conclude that considering increased dataset reduces validation loss during testing phase, and improves accuracy. Hence, generalization capability of model can be augmented by raising dataset size in deep learning model.

Figure 11 contains a few paradigms of generated summary. It contains comparison of summary of both Greedy approach as well as Beam Search approach (Beam size, k=3). The reviews shown in the figure are preprocessed reviews.

Fig.11. Summary generated using Beam Search approach and Greedy approach



```
Review: recipe box tasty easy complaint buy big box full go fast
Original summary: great classic cake
Predicted Greedy summary: delicious
Predicted Beam search summary: tasty and easy to prepare

Review: love product ordered great taste great price never
        disappointed brand price
Original summary: knew it ahead of time
Predicted Greedy summary: great product
Predicted Beam search summary: great product at great price

Review: excellent happy product big tea drinker definitely
        love tea brand order classic tea
Original summary: bentley classic tea
Predicted Greedy summary: tea
Predicted Beam search summary: great tea

Review: every item decided check amazon found ordered product
        great price arrived days great way order thing
Original summary: caramel apple cider
Predicted Greedy summary: great product
Predicted Beam search summary: great product great price

Review: price went much think go sams club
Original summary: high price
Predicted Greedy summary: too expensive
Predicted Beam search summary: over priced

Review: good chewy soft delicious grams protein wonderful
        always grab one quick easy lunch
Original summary: best bars
Predicted Greedy summary: delicious
Predicted Beam search summary: tasty and healthy
```

Fig.12. Percent representation of improvement

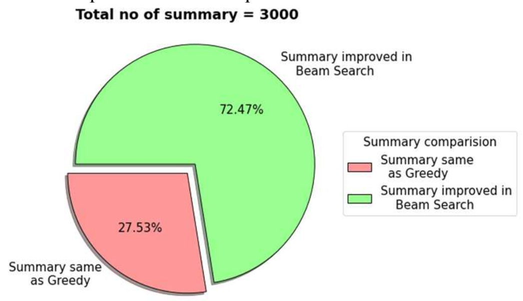


Figure 12 shows pie-chart of comparison of summary generated from 3000 reviews in both approach. 72.47% of summary in Beam Search approach was improved; while, remaining 27.53% of summary generated from both techniques are identical.

V. Our Contribution and Future Work

Pre-processed data is fed into sequence-to-sequence encoder-decoder architecture with LSTM to train the model. Attention layer is used to process input words by their weightage. During inference phase, beam search technique with linear normalization is used to decode test sequence, which choses k (k=beam size) best outputs at each steps. The results of beam search outputs were compared with greedy outputs. As a result, beam search generates better possible output sequence compared to greedy decoder. We have used whole dataset of Amazon Fine Food reviews containing total 5,68,454 reviews to train model.

For future work, we propose bi-directional LSTM in same model for better context vector, which captures context of sentence from both directions. Rouge score can be calculated.

VI. Conclusion

In this era of technology, huge amount of data is available in each and every field of the world. Data mining plays crucial role to deal with large data. When it comes to analysis of reviews of a product, there are plethora of reviews available on the internet. It is tedious and time consuming task to go through each and every product review manually. To ease this process, automatic text summarization becomes useful. In this advance field of text summarization, day by day, new technologies are developed to ease the process of summarization. Still, there are some flaws in already developed technologies, which can be handled by developing alternate options to make it efficient. Abstractive text summarization generated by LSTM encoder-decoder model using beam search strategy with linear normalization at inference decoder and attention mechanism can improve text summarization results compared to greedy strategy in same.

Appendix

```
def beam_search(input_seq,beam_size):
  v_out, v_h, v_c = encoder_model.predict(input_seq)
  trgt_seq = np.zeros((1,1))
  # Assigning sequence to start of the sequence word.
  trgt_seq[0, 0] = target_word_index['sostok']
  t=1
  decoded_sequence=['' for i in range(beam_size)]
  decoded_probability=['' for i in range(beam_size)]
  smpld_tkn_indx=['' for i in range(beam_size)]
  smpld_tkn=['' for i in range(beam_size)]
  stop_array=[0 for i in range(beam_size)]
  out_tkns, h, c = decoder_model.predict([trgt_seq] + [v_out,
 v_h, v_c])
  out_tkns[0,-1,:]=np.insert(softmax(np.delete(out_tkns[0,-
1,:],0)),0,0)
  seq_h=[h for i in range(beam_size)]
  seq_c=[c for i in range(beam_size)]
  for i in range(beam_size):
    smpld_tkn_indx[i] = np.argmax(out_tkns[0,-1,:])
    smpld_tkn[i] = reverse_target_word_index[smpld_tkn_in
dx[i]]
    if(smpld_tkn[i]!='eostok'):
      decoded_sequence[i]+=smpld_tkn[i]+' '
      decoded_probability[i]=math.log(out_tkns[0,-
1,:][smpld_tkn_indx[i]])
    elif (smpld_tkn[i] == 'eostok'):
      decoded_sequence[i]=''
      decoded_probability[i]=-1000
      stop_array[i]=1
    out_tkns[0,-1,:][smpld_tkn_indx[i]]=0
  stop_condition = False
  while not stop_condition:
    t+=1
    # Checking end of seq
    j=0
    for i in range(beam_size):
```

```
    if stop_array[i]==1:
      j+=1
  if j==beam_size:
    stop_condition = True
    break
  seq_h1=[]
  seq_c1=[]
  smpld_tkn_indx1=[]
  decoded_sequence1=[]
  decoded_probability1=[]
  stop_array1=[]
  for i in range(beam_size):
    # Update target sequence
    trgt_seq = np.zeros((1,1))
    trgt_seq[0, 0] = smpld_tkn_indx[i]
    # Update internal states vectors
    v_h=seq_h[i]
    v_c=seq_c[i]
    out_tkns, h, c = decoder_model.predict([trgt_seq] + [v_o
ut, v_h, v_c])
    out_tkns[0,-1,:] =
np.insert(softmax(np.delete(out_tkns[0,-1,:],0)),0,0)
    smpld_tkn_ind,decoded_seq,decoded_prob,stop_arr=bea
m(beam_size,out_tkns,smpld_tkn_indx[i],decoded_sequenc
e[i],decoded_probability[i],stop_array[i],t)
    smpld_tkn_indx1.extend(smpld_tkn_ind)
    decoded_sequence1.extend(decoded_seq)
    decoded_probability1.extend(decoded_prob)
    stop_array1.extend(stop_arr)
    seq_h1.append(h)
    seq_c1.append(c)
  for i in range(beam_size):
    max_prob_index = np.argmax(decoded_probability1)
    smpld_tkn_indx[i]=smpld_tkn_indx1[max_prob_index]
    decoded_sequence[i]=decoded_sequence1[max_prob_in
dex]
    decoded_probability[i]=decoded_probability1[max_prob
_index]
    stop_array[i]=stop_array1[max_prob_index]
    seq_h[i]=seq_h1[math.floor(max_prob_index/beam_size
)]
    seq_c[i]=seq_c1[math.floor(max_prob_index/beam_size
)]
    decoded_probability1[max_prob_index]=-100
 return decoded_sequence[np.argmax(decoded_probability)]

def beam(beam_size,out_tkns,smpld_tkn_ind,decoded_seq,
decoded_prob,stop_arr,t):
 decoded_sequence1=['' for i in range(beam_size)]
 decoded_probability1=['' for i in range(beam_size)]
 smpld_tkn_indx1=['' for i in range(beam_size)]
 stop_array1=[0 for i in range(beam_size)]
 if stop_arr!=1:
  for i in range(beam_size):
   smpld_tkn_indx1[i] = np.argmax(out_tkns[0,-1,:])
   smpld_tkn1 = reverse_target_word_index[smpld_tkn_in
dx1[i]]
   j=smpld_tkn_indx1[i]
   if(smpld_tkn1!='eostok' and t<=max_summary_len):
    decoded_sequence1[i]=decoded_seq+smpld_tkn1+' '
    decoded_probability1[i]=(decoded_prob*(t-
1)+math.log(out_tkns[0,-1,:][smpld_tkn_indx1[i]]))/t
   # Check end of seq or max length
   elif (smpld_tkn1 == 'eostok' or t>max_summary_len):
    decoded_sequence1[i]=decoded_seq
    decoded_probability1[i]=decoded_prob
    stop_array1[i]=1
    smpld_tkn_indx1[i]=smpld_tkn_ind
   out_tkns[0,-1,:][j]=0
 elif stop_arr==1:
  for i in range(beam_size):
   decoded_probability1[i]=-1000
   stop_array1[i]=1
  decoded_sequence1[0]=decoded_seq
  decoded_probability1[0]=decoded_prob
  smpld_tkn_indx1[0]=smpld_tkn_ind
  stop_array1[0]=stop_arr
  return smpld_tkn_indx1,decoded_sequence1,decoded_pr
obability1,stop_array1
```

REFERENCES

[1] Arvind Pai, “A Quick Introduction to Text Summarization in Machine Learning,” 2018, [online] Available: https://towardsdatascience.com/a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f.

[2] Arvind Pai, “Comprehensive Guide to Text Summarization using Deep Learning in Python,” 2019, [online] Available: https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/.

[3] D. Patel, N. Shah, V. Shah and V. Hole, "Abstractive Text Summarization on Google Search Results," 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2020, pp. 538-543, doi: 10.1109/ICICCS48265.2020.9120998.

[4] Sunil Ray, Understanding and coding Neural Networks from Scratch in Python and R, 2020, [online] Available: https://www.analyticsvidhya.com/blog/2020/07/neural-networks-from-scratch-in-python-and-r/.

[5] Pranjal Srivastava, Essentials of Deep Learning: Introduction to Long Short Term Memory, 2017, [online] Available: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/.

[6] J. Shah, M. Sagathiya, K. Redij and V. Hole, "Natural Language Processing based Abstractive Text Summarization of Reviews," 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2020, pp. 461-466, doi: 10.1109/ICESC48915.2020.9155759.

[7] J. Chen and F. You, "Text Summarization Generation Based on Semantic Similarity," 2020 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS), Vientiane, Laos, 2020, pp. 946-949, doi: 10.1109/ICITBS49701.2020.00210.

[8] A. Jadhav, R. Jain, S. Fernandes and S. Shaikh, "Text Summarization using Neural Networks," 2019 International Conference on Advances in Computing, Communication and Control (ICAC3), Mumbai, India, 2019, pp. 1-6, doi: 10.1109/ICAC347590.2019.9036739.

[9] G. Szűcs and D. Huszti, "Seq2seq Deep Learning Method for Summary Generation by LSTM with Two-way Encoder and Beam Search Decoder," 2019 IEEE 17th International Symposium on Intelligent Systems and Informatics (SISY), Subotica, Serbia, 2019, pp. 221-226, doi: 10.1109/SISY47553.2019.9111502.

[10] S. Zhao, E. Deng, M. Liao, W. Liu and W. Mao, "Generating summary using sequence to sequence model," 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 2020, pp. 1102-1106, doi: 10.1109/ITOEC49072.2020.9141919.

[11] M. Tomer, M. Kumar, “Improving Text Summarization using Ensembled Approach based on Fuzzy with LSTM,” *Arab J Sci Eng* 45, 10743–10754 (2020). https://doi.org/10.1007/s13369-020-04827-6.

[12] R. Boorugu and G. Ramesh, "A Survey on NLP based Text Summarization for Summarizing Product Reviews," 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2020, pp. 352-356, doi: 10.1109/ICIRCA48905.2020.918335.