Abstractive Text Summarization on Templatized Data

C. Jyothi(✉) and M. Supriya

Department of Computer Science and Engineering, Amrita School of Engineering, Bengaluru, Amrita Vishwa Vidyapeetham, Bengaluru, India
m_supriya@blr.amrita.edu

Abstract. Abstractive text summarization generates a brief form of an input text from the original source without the sentences being reused by still preserving the meaning and the important information. This could be modelled as a sequence-to-sequence learning by exploiting Recurrent Neural Networks (RNN). Typical RNN models are tough to train owing to the vanishing and exploding gradient complications. 'Long Short-Term Memory Networks (LSTM)' are an answer for such vanishing gradients problem. LSTM based modelling with an attentional mechanism is vital to improve the text summarization application. The key intuition behind attention mechanism is to decide how much attention one needs to pay to every word in the input sequence for generating a word at a particular step. The objective of this paper is to study various attention models and its applicability to text summarization. The intent is to implement Abstractive text summarization with an appropriate attention model using LSTM on a templatized dataset to avoid noise and ambiguity in generating high quality summary.

Keywords: RNN · LSTM · GRU · Text summarization

1 Introduction

We are residing in an era where modern scientific innovations have resulted in the generation of a large volume of data. Every click on the computer, phone or social media produces data. Statistics from the international data corporation (IDC) concludes that, "The total amount of data circulating annually would sprout from 4.4 zetta bytes (10^{21} bytes) in 2013 to 180 zeta bytes in 2025" [1]. With such immense amounts of data available, we need algorithms that will automatically abbreviate long texts to generate precise summaries which can pass the envisioned message. This need brings in the relevance of automatic text summarization. It is the process of shortening a set of data computationally, to create a subset (a summary) that represents the most important or relevant information within the original content [2]. Text Summarization deals with distilling the most important information from a source (or sources) to produce an abridged version for a particular user (or users) and task (or tasks) [3]. If the distillation process is performed with the help of a computer program, then it is called an automatic text summarizing system. This process helps us in faster discovery of relevant information

N. Kumar et al. (Eds.): UBICNET 2021, LNICST 383, pp. 225–239, 2021.
https://doi.org/10.1007/978-3-030-79276-3_17

as well as to consume the information appropriately from the enormous amount of data available online. There are innumerable applications for automatic text summarization [4]. In Telemedicine, summaries of medical records of patients are beneficial to health care providers to analyze and route the case to appropriate health care professionals. Text summaries of recent research papers are generated to know the current trends and innovation in each sector in Science and R&D. Individuals with hearing impairments could take assistance from voice to text summarization to remain updated with the current content. Financial documents such as earning reports and financial news can be summarized with the help of tailor-made summarization systems which can assist analysts to quickly check market indicators.

To address this ever-growing demand of summarizers, various types of summarization systems have been designed. Based on input, text summarization is classified into multi-document and single document systems for summarization. In single document summarization, summary of a single document is created by extracting the important sentences. In multi-document summaries, important sentences are extracted from multiple documents, then they are ranked among each other and important sentences are extracted for which summaries are generated [5]. Based on output, text summarization systems can be categorized as Extractive and abstractive text summarization systems. Extractive summarization is analogous to a highlighter where one reads the document, highlights the important sentences, and writes down those sentences to generate a summary. Abstractive text summarization is analogous to a pen where the document is read, comprehended, and written down as a summary in one's own words. Based on purpose, text summarization is divided in to Generic, Query based and domain specific text summarization systems. Sentences are extracted from multiple documents which are matched with a query and the important sentences are extracted and ranked to generate summaries in Query based text summarization systems. In domain specific text summarization, multiple documents from the domain are pre-processed and saved in a database. When a new document is to be summarized, the important features of the domain are extracted from the database and based on that, the sentences in the document are scored and are ranked and summarized.

In this work, an Abstractive text summarizer is modelled using Sequence to Sequence models with additive attention model on templatized text. This can aid in faster training still giving good accuracy. Seq2Seq model turns one sequence into another sequence with the use of a RNN or more often LSTM or Gated Recurrent Unit (GRU) to avoid the problem of vanishing gradient. The primary components in this model are an Encoder and a Decoder. The input sequence/text is converted into a context vector by an encoder which is passed on as decoder input to another sequence/text based on the context [6]. The encoder-decoder architectural model is generally preferred to convert a given sequence of definite length into a sequence of different length. The encoder converts the input sequence to an internal state which is then passed as input to the decoder to generate another sequence as output. This architecture has various applications such as image captioning, text summarization, machine translation etc. Generally, special categories of recurrent neural networks like LSTM or GRU's are stacked together to be used as encoder and decoder. A recurrent neural network has feedback connections that can be considered as many copies of the same network to produce an output which in turn

goes in as input to the next network. This makes RNN the appropriate choice for data involving sequence such as speech and text [7]. But normal RNN's are inefficient to capture long term dependencies in the sequence. The output of the current network will mostly depend only on the immediately preceding information. Hence LSTMs were introduced to overcome this downside of normal RNN's.Text data is messy and deep learning neural networks expects the input to be a fixed length vector. Hence to convert the text into a set of numbers, a tokenizer should be used [8]. Later an appropriate word embedding technique need to be applied to convert the tokenized text into a dense vector. This dense vector will be the input sequence to the encoder. There are some issues like lack of enough vocabulary in training data or sometimes too many trainable parameters due to many words in training data. This is countered using transfer learning approaches where popular models like glove, wordtovec etc. are trained and saved as pre-trained model. These models are used while training other models to save training time and to achieve a rich vocabulary [9]. As the size of the sequence increases such as in text summarization, encoder-decoder architecture which has multiple layers of LSTM stacks become inefficient as the whole context of the entire text need to be consolidated into a single context vector. Hence text summarization problems performed using Seq2Seq models using LSTMs need an additional layer known as attention to capture the context well and generate more accurate summaries. This will enable the decoder to give more importance to some part of the network as compared to others.

Methods for automating text summarization are vital in today's circumstances where there is copiousness of information such as e-mail, news, websites etc. and deficiency of human expertise and time to infer the information. The inspiration to generate summaries are [10] they lessen the interpretation time, makes exploring and selection of paper easier, improve the usefulness of indexing, are less prejudiced compared to human summarizers, can be tailor made and are beneficial in question-answering systems to provide personalized information, enable abstract services by increasing the number of text documents they can process, do not require expertise in the field as in manual summarization.

This paper is organized in to different sections. Section 2 is separated into concept review and research review where the underlying concepts of attention based on LSTM is discussed in detail along with the mathematical aspects and also the past works in the domain. The technical aspects of the project in terms of various modules and execution details are discussed in Sect. 3. Section 4 presents the outcomes of the project and the paper is concluded in Sect. 5.

2 Literature Review

2.1 Concept Review

LSTM are special types of RNN. The repeating unit in a LSTM network has four layers of neural network unlike one in a normal RNN. These are referred to as gates and are shown in Fig. 1.

The first step in a LSTM network is a forget gate layer whose output is a sigmoid function of the previous hidden state as well as the current input. The weighted sum of previous hidden state and input added with a bias is passed through the sigmoid function,

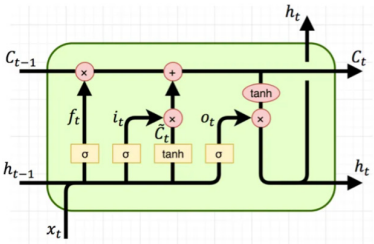

Fig. 1. LSTM network [11]

which outputs a zero or a one as can be seen in Eq. (1). An output of zero means that state needs to be forgotten and one suggests that it needs to be remembered.

$$f(t) = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1}$$

The second step consists of an input gate layer whose output comprise of a sigmoid function of the current input, and the previous hidden state added to a bias like the output of forget layer as can be seen in Eq. (2). It also has a tanh function which generates a list of possible values to be updated in the cell state as shown in Eq. (3). Pointwise multiplication is performed on these outputs in the next stage.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3}$$

The third step consists of generating the new cell state by point wise addition. The components added are the result of the pointwise multiplication of (i) the previous cell state and forget gate output and (ii) the input gate output and the candidate cell states output as shown in Eq. (4).

$$c_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4}$$

The fourth step consists of generating the output by applying a sigmoid function on the weighted sum of the preceding hidden state and the current input which is then added to a bias as shown in Eq. (5). A pointwise multiplication of the output and the result obtained by applying tanh function on the current cell state is performed to generate the new hidden state as can be seen in Eq. (6).

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_0) \tag{5}$$

$$h_t = o_t * \tanh(C_t) \tag{6}$$

Using these gates, LSTMs can remember only what is required as it passes the hidden state and cell state from one network to the other. Hence it can avoid vanishing gradient problem to an extent when the sequences are not extremely long and thereby capture their long-term dependencies.

There are two phases in the set-up of Encoder-Decoder architecture in Seq2Seq models namely Training phase and Inference phase. In the training phase of the encoder, initial hidden and cell state are initialized to zero or random numbers and at each step, an input is given to the network which along with the output hidden state of the previous step will try to capture the context. In the last step, the final hidden state is generated to carry the context of the whole input sequence [12]. In the training phase of the decoder, the final state of the encoder will be the initial state for the decoder. The start and end tokens are also appended to the output text before starting to train the model which will enable the decoder to learn where to start and stop. The decoder will be trained using the given training data to generate the output sequence. In the inference phase, the decoder will be given the output of the encoder and the start token which outputs the maximum probability word as the output which along with the current generated hidden state will predict the next word. This process will continue till we reach the end of the sequence.

In the attention model, a dot product operation will be performed on the output vector of each decoder state with the encoder hidden state at each step to generate an attention score for each encoder state. The attention score is then passed through a SoftMax function and an attention distribution will be obtained. This distribution is multiplied with the input sequence and a summation is done to generate the output of the decoder. And this process is repeated with all the decoder inputs to generate a more accurate summary [13].

Additive attention model was first proposed by Bahdanau et. al. [13]. Instead of converting the whole input sequence into a single context vector, this model tries to give importance to specific words in the input sequence with the aid of attention weights. All the hidden states of the encoder including the forward and backward state in a bidirectional LSTM are used to generate a context vector. There after it performs a linear combination of the states of both encoder and decoder, hence the name additive attention.

The attention layer consists of three steps for the generation of alignment score, attention weights and context vector generation. The alignment score tries to evaluate the matching between the input around position "j" and the output position "i". This is calculated using the previous decoder's hidden state, $s_{(i-1)}$ preceding the target word position and the hidden state, h_j of the input sentence. The alignment score is calculated as shown in Eq. (7).

$$e_{ij} = a(s_{i-1,h_j}) \tag{7}$$

The alignment score is passed through a SoftMax function to obtain the attention weight of the corresponding input position 'j' for the output position 'i' as shown in Eq. (8).

$$\alpha_{ij} = \exp(e_{ij}) / \sum_{k=1}^{T_x} \exp(e_{ik}) \tag{8}$$

The corresponding context vector is calculated as per the Eq. (9)

$$C_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{9}$$

The current hidden state of the decoder is a function of the above context vector, previous hidden state and previous output as can be seen in Eq. (10)

$$s_i = f(s_{i-1}, c_i, y_{i-1}) \tag{10}$$

The current output generated will be a function of the current context vector, current hidden state as well as the previous output as can be seen in Eq. (11)

$$g(y_{i-1}, s_i, c_i) \tag{11}$$

Another type of attention is multiplicative attention. In this method, the attention scores are calculated by matrix multiplication of encoder and decoder states. This improves space efficiency as well as speed. This method was suggested by Luong and hence the name Luong attention [14]. He suggested two types under this category which are global and local attention. Global Attention considers all the words in the input sequence to predict a word in the output sequence. Since all words being considered becomes computationally expensive as the length of the input sequence increases. In Local Attention method, attention is placed only on a subset of the input which can be selected either monotonically or predictively. Since the number of source positions being considered decreases, it is computationally more economical.

The Additive attention method makes use of bidirectional LSTM based encoder's both backward and forward states and combines it with the output of the previous decoding step. This will have a non-stacking uni-directional decoder. The multiplicative attention model considers the hidden states of the topmost LSTM layer in both encoder as well as decoder [14].

2.2 Review of Previous Research Findings

A study conducted by Juan Cao et al. suggests a new methodology for converting table data into textual format using NLDT architecture and then generating summaries on them using encoder-decoder architecture with beam search algorithm in the inference stage [15]. A survey conducted by Wubben et al. compares the different approaches by which we can create a templatized input text to a summarization system to produce more accurate and meaningful summaries [16]. It uses the weatherGov dataset proposed by Lian et al. and modified it to reduce noise and added tags to produce high quality summaries. A study conducted by Lian et al. tries to understand the problem of learning correspondences between the text and summary. In order to handle the high degree of uncertainty, it suggests a model which is generative i.e., it slices the text and plots it to summary. This model has been applied to WeatherGov dataset [17]. Bahdanau et al. tried to improve the quality of output by giving attention weights to each source position in the encoder-decoder architecture rather than encoding the whole input sequence in to a single context vector. This paper revolutionized the way encoder decoder architecture is used and has been widely cited [18]. It also resulted in a separate area of study named attention models and many other attention models are studied thereafter. A model proposed by Nallapati et al. suggests a Hierarchical encoder with hierarchical attention. It also introduces many other novel models to address the challenges in abstractive text

summarization and presents a new data set for text summarization [19]. A survey conducted by Shi Tian et al. compares the different approaches to perform Abstractive text summarization. This work proposes an open-source tool kit for generating abstractive summaries and named it as 'Neural Abstractive Text Summarizer' (NATS) [20].

A two-phase model has been proposed by Madhuri et al. [21]. In phase one, it uses a hierarchical fusion of various similarity measures to retrieve the relevant sentences. The hybrid similarity measure incorporates cosine, semantic and word order similarity modules. Phase two consists of a clustering module to remove redundant sentences. This module was realized using DBSCAN agglomerative clustering technique. This model has generated summaries which have shown 86% accuracy. Another model proposed by Ahuja et al. offers means to generate summaries of multiple documents by generation of summaries of single documents and merging them using cosine similarity. The summaries of single documents are generated by extracting features such as Document, Numbers, Noun (proper), Sentence position, and Sentence length (normalized). Total sentence weight is calculated for each sentence and they are extracted to generate multi-document text summary [22]. The heading wise summarizer proposed by Krishnaveni et al. takes from each heading a third of sentences to improve coherence in the final summary generated. The performance improves with the length of the summary and with the number of headings in the document [23]. A model proposed by Siji et al. recommends three level of scoring based on synonym, similarity, and context. But the drawback of the model is that it can't reformat text [24]. A non-dominated genetic algorithm II-based summarizer proposed by Manju Priya et al. uses four different scores namely coherence, thematic, relevance and sentence length to evaluate the significance of sentences [25]. However, this summarizer lacks contextual and semantic information required for better results. A hybrid approach is being proposed by Veena et al. using singular value decomposition and named entity recognition to create a probabilistic model called a Belief Network. This model generates new sentences using subject, verb, and Object. But this model does not include co-reference resolution which can improve the quality of generated sentences [26]. A news summarization system proposed by Karumudi et al. suggests crawling of multiple websites and storing it in a database. It looks for details like person, organization, and location to perform information retrieval of the related news article. This system has the disadvantage of storage access time as well as text processing time delays [27]. A trigraph based extractive text summarization system has been proposed by Devika et al. to extract relevant trigrams in the document. However, in this system, centrality of nodes is only a local measure using indegree and outdegree of each node [28]. A survey conducted by Varalakshmi et al. compares the different approaches to perform extractive text summarization [29]. It recommends swarm intelligence or genetic algorithm technique to extract the best summaries. It also proposes coherence graph to improve coherency between the sentences as well as removing redundant sentences.

Based on the study of these models we can conclude that coherency and redundancy in sentences need to be improved in summaries generated using extractive methods. It is important to know the context of the words in a document as well as in all other documents being considered for summary. Also, many of the previous work in summarization has been extractive. This technique identifies significant sentences, passages in the original

document, and replicates them as summary. People on the contrary, tend to reword the original story in their own words. Per se, human summaries are abstractive in nature and hardly ever consist of replication of existing sentences from the document. The existing LSTM based work has used DUC and CNN data set and has the challenge of giving out of vocabulary words (UNK). The weatherGov dataset is templatized to reduce noise and has a smaller vocabulary and can hence obtain better results.

This paper discusses an abstractive text summarization system based on additive attention model on a templatized dataset and the details are discussed in Sect. 3.

3 System Design and Implementation

The proposed problem can be modeled using Deep learning and Natural Language Processing techniques. This can be implemented in python programming language with a suitable integrated development environment (IDE) such as Jupyter Notebook with necessary libraries including TensorFlow, NLTK, Keras, pandas, matplotlib etc.

To implement the problem defined above we need a high computational power runtime environment such as graphic processing unit (GPU). Google colab provides this runtime which can be used up to 12 h continuously in a session. The various components of the system and the implementation details are shown in Fig. 2.

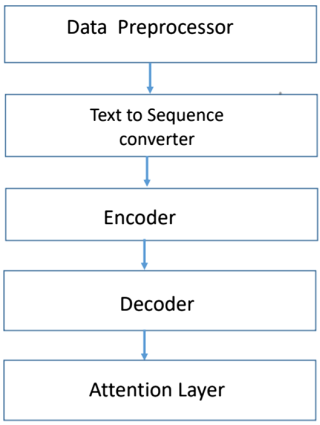

Fig. 2. High Level Design

The data pre-processing module.

a. Removes unwanted tags
b. Removes special characters
c. Removes stop words from the text
d. Drops rows without any entries

The Text to sequence converter

- Converts input text to a sequence of numbers
- Removes those sequence in the decoder output training and validation data which have only <sos> and <eos> tokens

The Encoder is designed with five layers of bidirectional LSTMs stacked on top of each other each with a recurrent drop out and drop out of 0.4. The pretrained Glove embedding was used to convert the tokenized input to the encoder into vectors of 100 dimensions [30]. The decoder is made up of only one unidirectional LSTM layer which has been fed with the tokenized and embedded summaries during training each of which are 100 dimensional vectors. This makes use of pre-trained glove 100 model to reduce training time.

The attention layer used in the model is a custom off the shelf additive attention based on Bahdanau's model. This layer is fed by the encoder and decoder outputs which generates the output of the attention layer. This output is concatenated with the decoder output and then fed to a dense time distributed layer with SoftMax activation function to generate the final outputs of the decoder layer.

The above-mentioned model with encoder decoder architecture based on attention is compiled with optimizer as 'rmsprop' to enable faster convergence and 'sparse categorical cross entropy' as loss function since the output is a vector of numbers. This has been fit in to the training data obtained by splitting the dataset in the ratio 9:1 for training and validation using scikit-learn.

The model was trained on the WeatherGov dataset on a GPU machine. The encoder inputs, the output of the last stage of the encoder and the corresponding hidden state and cell state has been used to create a feature vector. Similarly, a decoder model is set up with corresponding decoder values as in the encoder and an attention model. The given input sequence is then fed into the encoder model to generate the corresponding hidden states. These hidden states along with an <sos> token that acts as the initial decoder input is fed into the corresponding decoder model to generate the next word in the sequence. This sequence is again fed into the decoder model along with hidden state of encoder and this process continues till we reach the <eos> token or maximum summary length. Here the next word is selected based on max probability.

4 Results

WeatherGov dataset consists of 29528 records containing 36 records of weather data paired with natural language forecast with average summary length 28.7 for 3753 cities in the USA over three days. The total vocabulary of this dataset is less than 400.The plot of the length of summary versus number of summaries and length of text versus number of texts is depicted in Fig. 3. It can be inferred that around 96.5% of the summaries are below a length of 60 and 94.2% of the texts are below a length of 120. Hence the max lengths for the training data were set accordingly.

The proposed model was first trained on a GPU with 25000 texts from the WeatherGov dataset. The keras model as explained in Sect. 3 with two stacked bi directional LSTMs in encoder and single unidirectional LSTM in the decoder was set up. Each of these LSTM layer has its embedding layer to create a vector.An off the shelf attention

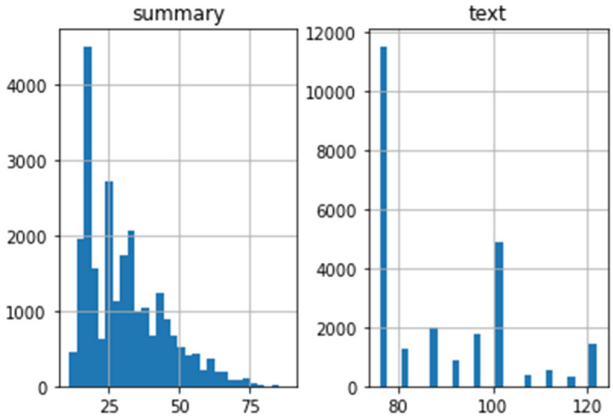

Fig. 3. Length distribution of text and summaries in WeatherGov

model was also included as one layer. In the output side it is converted to a time distributed layer to get the summaries. It was noticed that the number of trainable parameters were around 875,545 which is comparatively less owing to the use of templatized data. The above discussed information has been captured from the model set up using Keras in Fig. 4.

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 120)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 120, 100)     14000       input_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
bidirectional (Bidirectional)   [(None, 120, 256), ( 234496      embedding[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 100)    30900       input_2[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 256)          0           bidirectional[0][1]
                                                                 bidirectional[0][3]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 256)          0           bidirectional[0][2]
                                                                 bidirectional[0][4]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, None, 256),  365568      embedding_1[0][0]
                                                                 concatenate[0][0]
                                                                 concatenate_1[0][0]
__________________________________________________________________________________________________
attention_layer (AttentionLayer ((None, None, 256),  131328      bidirectional[0][0]
                                                                 lstm_1[0][0]
__________________________________________________________________________________________________
concat_layer (Concatenate)      (None, None, 512)    0           lstm_1[0][0]
                                                                 attention_layer[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 281)    144153      concat_layer[0][0]
==================================================================================================
Total params: 920,445
Trainable params: 875,545
Non-trainable params: 44,900
```

Fig. 4. The Keras model summary of proposed system

The model was trained for 10 epochs and as the number of the epoch increases the loss is decreasing and the accuracy is increasing. Between the ninth and tenth epochs there is no significant improvement in the loss and accuracy which shows that the model has converged and training can be stopped. After training for ten epochs, a validation accuracy of 94.53% and a validation loss of 0.16 was obtained. The results of training during various epochs were captured in Fig. 5.

```
Epoch 1/10
322/322 [==============================] - 1555s 5s/step - loss: 0.8702 - accuracy: 0.8044 - val_loss: 0.3950 - val_accuracy: 0.8824
Epoch 2/10
322/322 [==============================] - 1594s 5s/step - loss: 0.3586 - accuracy: 0.8920 - val_loss: 0.3100 - val_accuracy: 0.9048
Epoch 3/10
322/322 [==============================] - 1576s 5s/step - loss: 0.2998 - accuracy: 0.9073 - val_loss: 0.2646 - val_accuracy: 0.9166
Epoch 4/10
322/322 [==============================] - 1573s 5s/step - loss: 0.2619 - accuracy: 0.9172 - val_loss: 0.2387 - val_accuracy: 0.9232
Epoch 5/10
322/322 [==============================] - 1571s 5s/step - loss: 0.2362 - accuracy: 0.9242 - val_loss: 0.2150 - val_accuracy: 0.9298
Epoch 6/10
322/322 [==============================] - 1563s 5s/step - loss: 0.2168 - accuracy: 0.9295 - val_loss: 0.1978 - val_accuracy: 0.9353
Epoch 7/10
322/322 [==============================] - 1575s 5s/step - loss: 0.2002 - accuracy: 0.9343 - val_loss: 0.1829 - val_accuracy: 0.9393
Epoch 8/10
322/322 [==============================] - 1543s 5s/step - loss: 0.1879 - accuracy: 0.9377 - val_loss: 0.1749 - val_accuracy: 0.9407
Epoch 9/10
322/322 [==============================] - 1545s 5s/step - loss: 0.1780 - accuracy: 0.9402 - val_loss: 0.1652 - val_accuracy: 0.9445
Epoch 10/10
322/322 [==============================] - 1542s 5s/step - loss: 0.1701 - accuracy: 0.9426 - val_loss: 0.1600 - val_accuracy: 0.9453
```

Fig. 5. Training results after various Epochs

The plot of train and test validation loss versus the number of epochs is as shown in Fig. 6. As can be noticed from the figure the training loss and testing loss are very close at around 10 epochs. This means that model has been trained well and hence training can be stopped at 10 epochs.

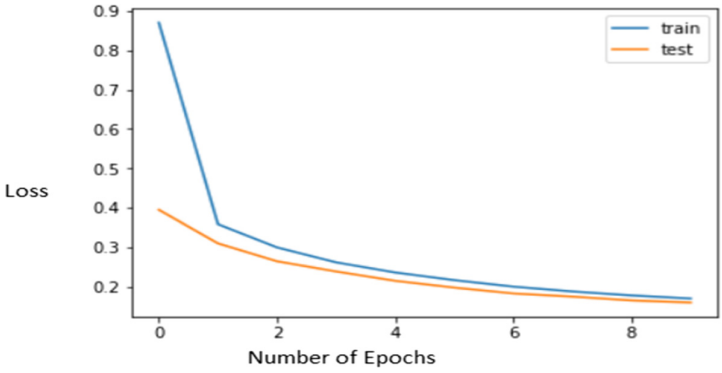

Fig. 6. Train/Test Loss Versus Epochs of WeatherGov

The above model can now be used to predict summaries. A few sample summaries predicted by the model along with the text and the human written summaries used as reference to train the model is shown in Fig. 7.

Metrics for evaluation of a text summarization system include Rouge score and BLEU score. BLEU score which can be expanded as The Bilingual Evaluation understudy score evaluates a machine generated summary against a human summary. If there

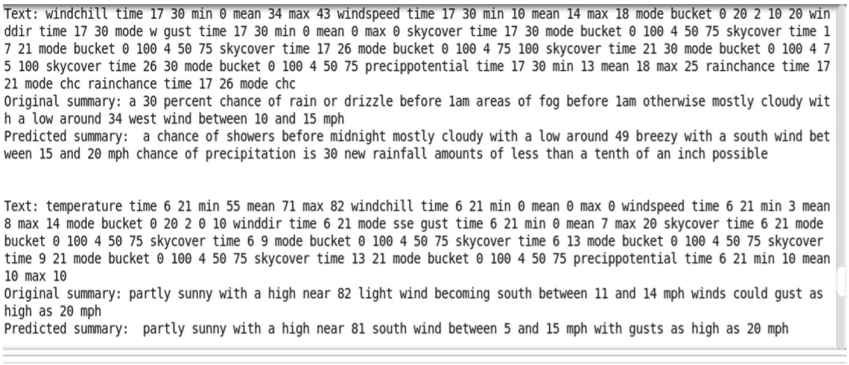

Fig. 7. Sample Summaries of WeatherGov

is a high degree of match, then it assigns a high score. It is a metric which is popular, inexpensive, and automated. This will not check for grammar mistakes or intelligibility. The sentences are compared with the reference sentences and scores are assigned to individual sentences. This is then normalized with the corpus length [31]. The BLEU score obtained for 200 summaries are captured in Fig. 8.

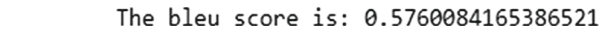

Fig. 8. BLEU score of 200 summaries in WeatherGov

The corpus BLEU which is used to calculate the score for multiple sentences applied on 100 sample summaries are captured in Fig. 9.

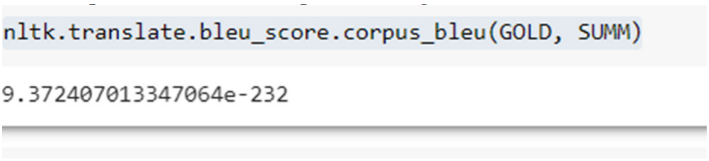

Fig. 9. Corpus BLEU score of 100 summaries in WeatherGov

Rouge Score which is the abbreviation for Recall oriented gisting under study score calculates the similarity between a generated summary to a reference summary [32]. The rouge score for 100 samples is captured in Fig. 10. Each summary has a rouge1, rouge2 and rougel score associated with it. Each of these has a separate value for precision, recall and f-score linked with it.These information captured for a few sample summaries have been shown in Fig. 10.

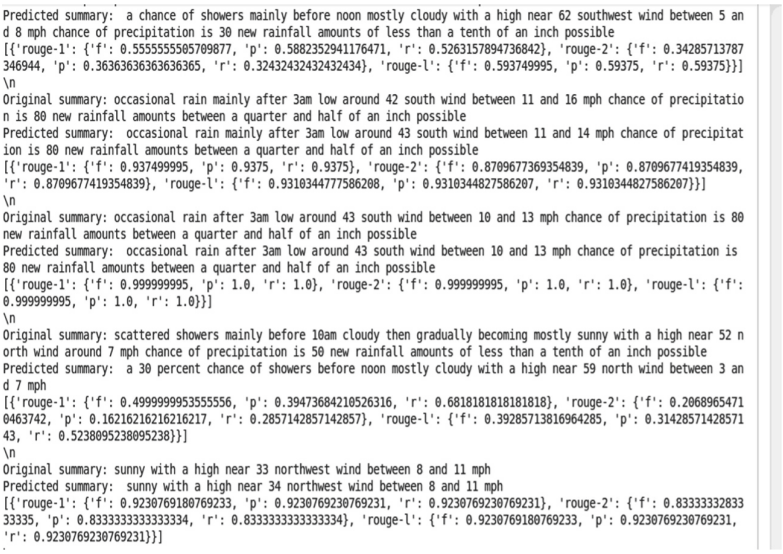

Fig. 10. ROUGE score of 100 summaries in WeatherGov

5 Conclusion

The proposed Seq2seq model-based text summarization system for generating abstractive text summaries was implemented. The encoder and decoder were based on LSTM networks and additive attention was included to improve the quality of summaries and the results were verified using BLEU score and ROUGE metrics. As the proposed system was trained on a templatized data set, the vocabulary of the training data reduced drastically and resulted in faster training and better accuracy. Further studies can be conducted in templatizing the text before training to achieve better scores in abstractive text summarization, especially in domain specific text summarization systems such as the one used in the proposed model. In future, beam search algorithm can be used to select the output sequences to improve the quality of summaries. The repetition of words can be avoided using coverage mechanism. The pointer generator networks can be used to improve the quality which make use of a combination of extractive and abstractive summarization techniques to reduce the training time as well. The trained model can be saved as a hierarchical data format-5(h5) file which is a multidimensional array of scientific data file. This can be deployed using python flask framework in cloud to predict summaries of unseen text.

References

1. https://www.idc.com/
2. https://en.wikipedia.org/wiki/Automatic_summarization
3. Advances in Automatic Text Summarization (The MIT Press) Abridged Edition, Inderjeet Mani

4. https://www.quora.com/What-are-the-real-world-applications-of-automatic-text-summarization
5. https://www.kdnuggets.com/2019/01/approaches-text-summarization-overview.html
6. https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/
7. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks
8. https://keras.io/examples/nlp/pretrained_word_embeddings/
9. https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/
10. https://machinelearningmastery.com/gentle-introduction-text-summarization/
11. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
12. https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/
13. https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/
14. https://blog.floydhub.com/attention-mechanism/
15. Cao, J.: Generating natural language descriptions from tables. IEEE Access **8**, 46206–46216 (2020). https://doi.org/10.1109/ACCESS.2020.2979115
16. van der Lee, C., Krahmer, E., Wubben, S.: Automated learning of templates for data-to-text generation: comparing rule-based, statistical and neural methods. In: Proceedings of the 11th International Conference on Natural Language Generation, pp. 35–45. Tilburg University, The Netherlands. Association for Computational Linguistics (2018)
17. Liang, P., Jordan, M., Klein, D.: Learning semantic correspondences with less supervision. In: Association of Computational Linguistics (2009)
18. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473 (2014)
19. Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B.: Abstractive text summarization using sequence-to-sequence RNNs and beyond. CoNLL (2016)
20. Shi, T., Keneshloo, Y., Ramakrishnan, N., Reddy, C.K.: Neural abstractive text summarization with sequence-to-sequence models. arXiv abs/1812.02303 (2018)
21. Madhuri Chandu, G.V., Premkumar, A., Susmitha K, S., Sampath, N.: Extractive approach for query based text summarization. In: 2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), Ghaziabad, India, pp. 1–5 (2019). https://doi.org/10.1109/ICICT46931.2019.8977708
22. Ahuja, R., Anand, W.: Multi-document text summarization using sentence extraction. In: Dash, S.S., Vijayakumar, K., Panigrahi, B.K., Das, S. (eds.) Artificial Intelligence and Evolutionary Computations in Engineering Systems. AISC, vol. 517, pp. 235–242. Springer, Singapore (2017). https://doi.org/10.1007/978-981-10-3174-8_21
23. Krishnaveni, P., Balasundaram, S.R.: Automatic text summarization by local scoring and ranking for improving coherence. In: 2017 International Conference on Computing Methodologies and Communication (ICCMC), Erode (2017), pp. 59–64 (2017). https://doi.org/10.1109/ICCMC.2017.8282539
24. Rani, S.S., Sreejith, K., Sanker, A.: A hybrid approach for automatic document summarization. In: 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, pp. 663–669 (2017). https://doi.org/10.1109/ICACCI.2017.8125917
25. Manju Priya, A.R., Gupta, D.: Extractive single document summarization using NSGA-II. In: Thampi, S.M., et al. (eds.) Intelligent Systems, Technologies and Applications. AISC, vol. 1148, pp. 197–211. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-3914-5_15

26. Veena, G., Gupta, D., Jaganadh, J., Nithya Sreekumar, S.: A graph based conceptual mining model for abstractive text summarization. Indian J. Sci. Technol. **9**(S1) (2017). https://doi.org/10.17485/ijst/2016/v9is1/99876. ISSN 0974-6846
27. Karumudi, G.V.N.S.K., Sathyajit, R., Harikumar, S.: Information retrieval and processing system for news articles in English. In: 2019 9th International Conference on Advances in Computing and Communication (ICACC), Kochi, India, pp. 79–85 (2019). https://doi.org/10.1109/ICACC48162.2019.8986223
28. Raj, D., Geetha, M.: A trigraph based centrality approach towards text summarization. In: 2018 International Conference on Communication and Signal Processing (ICCSP), Chennai, pp. 0796–0801 (2018). https://doi.org/10.1109/ICCSP.2018.8524528
29. Varalakshmi K, P.N., Kallimani, J.S.: Survey on extractive text summarization methods with multi-document datasets. In: 2018 International Conference, on Advances in Computing, Communications and Informatics (ICACCI), Bangalore, pp. 2113–2119 (2018). https://doi.org/10.1109/ICACCI.2018.8554768
30. Pennington, J., Socher, R., Manning, C.: Glove: global vectors for word representation. In: Proceedings of EMNLP, pp. 1532–1543 (2014)
31. https://en.wikipedia.org/wiki/ROUGE_(metric)
32. https://machinelearningmastery.com/calculate-bleu-score-for-text-python/