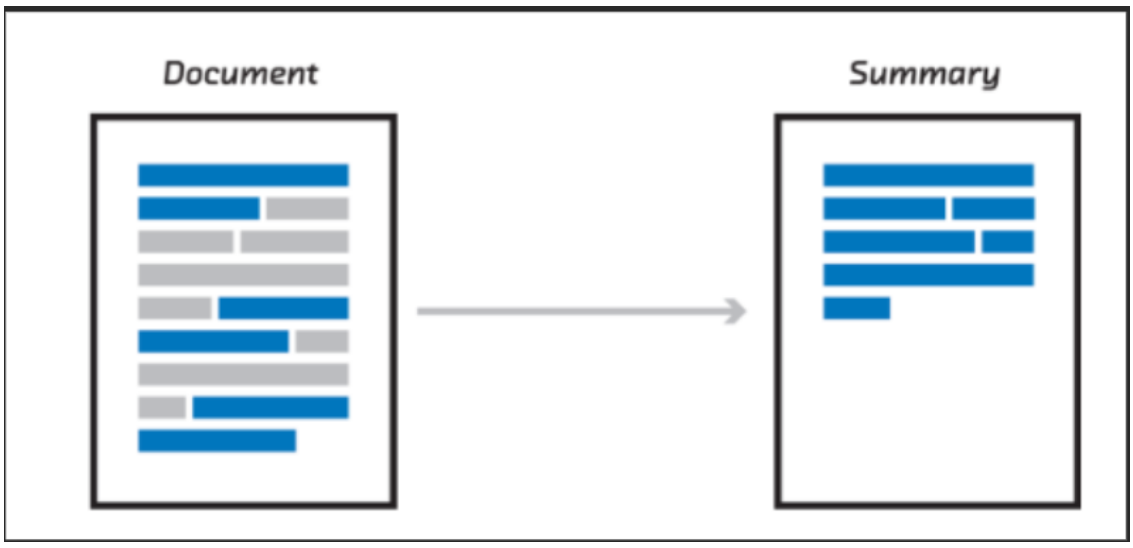
DATA612 - Final Project

Generating Short Summaries of Text from Headlines



Project Report by
Tejasri Pavuluri

Contents

1. Introduction

This project focuses on applying deep learning in Natural Language Processing (NLP), particularly on an NLP use case: generation of text summaries in the form of short news headlines. The task is to generate a coherent set of new headlines (or summaries) from many of the original headlines for each article provided. The goal is to cover the concept of Sequence-to-Sequence as model, which uses an encoder decoder framework and successfully generates logical headlines for many of the articles provided.

2. Problem Statement

With growing digital media and due to the vast amount of information that is generated, we see most news articles converted into few words as a summary. Everything is digitalized and we can find a large amount of digital data for different purposes on the internet and it's relatively hard to summarize this data manually. With the amount of time and resources required for manual summarization there is an increased demand for the summarization of longer documents such as news articles, research papers, blog papers, emails, and tweets. Automatic Text Summarization (ATS) is the eventually big one that could simply summarize the source data and give us a short version that could preserve the content and the overall meaning.

ATS is one of the most challenging and interesting obstacles in the field of Natural Language Processing (NLP). It is the process of generating a concise and meaningful summary of text from multiple text resources such as books, news articles, blog posts, research papers, emails, and tweets. The demand for automatic text summarization systems is spiking these days due to the availability of large amounts of textual data.

3. Background - Summarization

While the concept of summarization started in the 1950's, the field is still evolving to give the best and most efficient summaries. There are two approaches to text summarization.

1. Extractive summarization – which is a more basic solution that extracts the most important words or sentences without 'writing' any new content on its own
2. Abstractive summarization – which is a more sophisticated and promising model that learns to understand language in its syntax and context and generate summary using 'its own words'.

The above two types focus on how the key information found in the input text is reconstructed in the generated summary in their own ways. Both of these methods have their own unique challenges that pop up when looking at using longer text.

This project will provide an overview of summarization techniques in the context of abstractive summarization. Deep learning encoder-decoder (sequence-to-sequence) model is used to construct the text summary, where an encoder accepts the actual text and summary, trains the model to create an encoded representation, and sends it to a decoder which decodes the encoded representation into a reliable summary. As the training progresses, the trained model can be used to perform inference on new texts, generating reliable summaries from them.

4. Background of Model

To understand the model's basic functionality that was introduced in 2014, which is shown in the below figure, the sequence of objects (words, letters, time series etc.) are inputs and the model outputs another sequence of objects. The 'black box' in between is a complex structure of numerous Recurrent Neural Networks (RNNs) that first transfers an input string (in the case of seq2seq for text transformation) of varying length into a fixed length vector representation, which is then used to generate an output string.

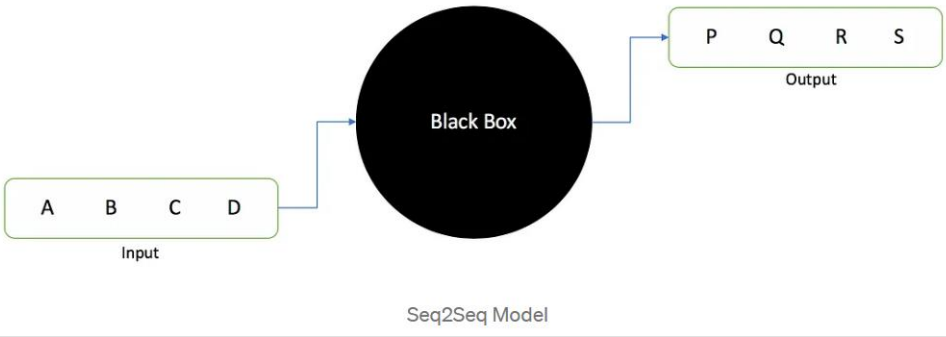


Figure: 1 – Basic Sequence-to-Sequence model – Image Source: Attention — Seq2Seq Models. Sequence-to-

The seq2seq model is an architecture based on the multiple LSTM network or sometimes a GRU. The model comprises of two main components: **encoder** and **decoder**. These components are responsible for summarizing the information into fixed length vectors through a bottleneck and then predict the target words from that fixed length vector.

The encoder-decoder architecture was introduced in the computer vision task for image generation. The bottleneck of the model is where the real magic happens. The information that is fed into the model is squeezed to a vector of desired length and this squeezed vector in the bottleneck stores important information that the decoder uses to predict the output. Often this vector is called the latent vector.

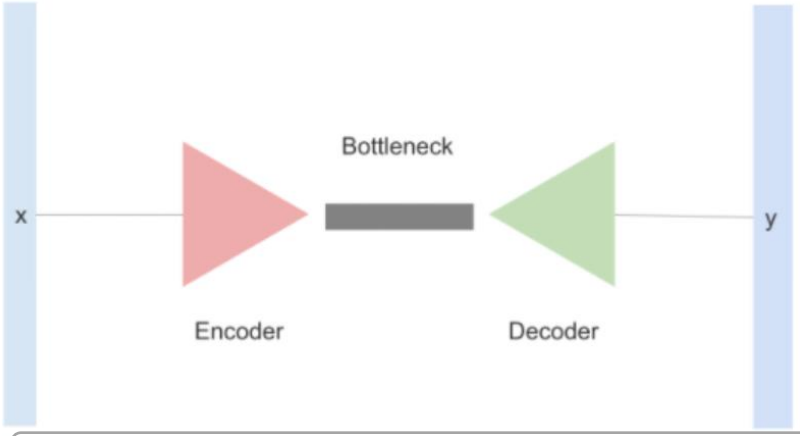


Figure: 2 – Encoder-Decoder Architecture – Image Source: How to Implement

Encoder:

The encoder of the network takes in the input sequence one element at a time and produces a vector called the context vector. This context vector aims to preserve the vital information for all the elements in the input sequence so that the decoder can make accurate predictions. As new information or words are sequentially fed into the encoder, the context vector is updated.

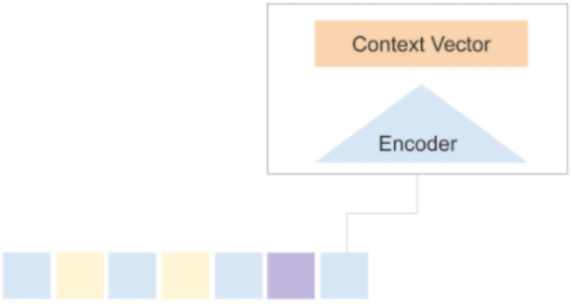


Figure: 3 – Encoder Architecture – Image Source: How to Implement

Decoder:

The decoder of the network takes in the final context from the encoder into the first layer of the network and starts predicting the target or the output sequences one element at a time.

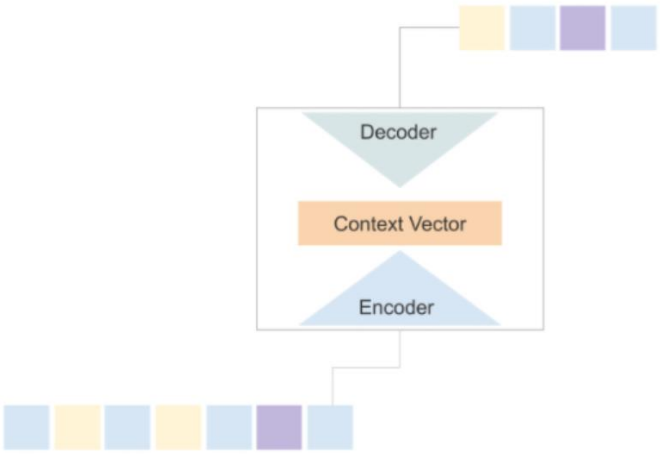


Figure: 4 – Decoder Architecture – Image Source: How to Implement

5. Implementation of Sequence to Sequence Model

This section describes the steps to implement the sequence2sequence model from scratch. These are common steps in any NLP models and also to prepare the data for the model.

5.1 Details of Dataset

I used the 'News Summary' dataset from Kaggle. It consists of two CSV files: one contains information about the author, headlines, source URL, short article and complete article, and another file only contains headlines and text. The news articles are from different sources like Hindu, Indian Times and Guardian ranging from February to August 2017. In the project, after combining the two csv files and extracting the 'headlines' and 'text', the dataset contains 102,915 records.

Dataset size: 102915

	headlines	text
22381	PNB shares fall 6% after posting biggest loss ...	Shares of fraud-hit PNB fell 6.1% on Tuesday a...
65761	Gerard Pique plans a World Cup-like tournament...	Barcelona's Spanish defender Gerard Pique is p...
41527	Congress entrapped in Blue Whale game: PM Modi	PM Narendra Modi on Monday said that Congress ...
24339	Flipkart, Amazon ask govt to extend FDI policy...	E-commerce platforms Flipkart and Amazon have ...
13076	Doctor opens fire at New York hospital, kills ...	A doctor armed with a rifle killed a female do...

Figure: 5 – News Summary dataset – After combining two csv files

5.2 Framework and Libraries

Keras deep learning framework is used for this project to build a sequence2sequence model that relies on encoder-decoder architecture. This section also covers a few other libraries and tools used to build the model followed by the implementation steps.

Deep Learning framework and libraries:

i. Keras
ii. Tensorflow

Libraries to fetch data

i. Sklearn
ii. Pandas
iii. Numpy
iv. Seaborn
v. Matplotlib
vi. pickle

5.3 Data Load and Preparation

The two 'news summary' dataset files are imported first using pandas read_csv() method and are combined into a single DataFrame. This resulted in 102,915 text and headlines data, as shown in the above figure. The headline column is re-named as 'summary' and below figure shows sample data from few rows

	summary	text
26642	Newly-wed bride shot dead by armed robbers on ...	A newly-wed bride returning home with a weddin...
12077	Melania appears to refuse to hold Trumps hand ...	A video which purportedly shows US First Lady ...
78008	Google disallows changing domain to search dif...	Technology giant Google has announced that the...
45252	Rape cases against Kerala priests disturbing, ...	While hearing a rape case against a priest in ...
34584	10 of a family dead after SUV collides with tr...	Ten mthembers of a family died while four othe...

Figure: 6 – News Summary dataset with 'Summary' and 'Text' columns

5.4 Pre-processing and data cleaning

Several data cleaning and pre-processing were performed on 'Text' and 'Summary' columns and the steps can be broken down as follows. Some variables like 'max_length', 'vocab_size', and 'embedding_dims' are needed to define the shape of the sequence-to-sequence neural network on the way of data pre-processing

a. **Text Cleaning** – Removed unnecessary characters, punctuations, expanding contradictions, removed any URL and multiple white spaces, removed stopwords using pre-defined nltk module etc. on both text and summary columns. To do this – a 'clean_text' function was created along with a regular expression library called 're' from python.

```
I'm == I am
I'm'a == I am about to
I'm'o == I am going to
I've == I have
I'll == I will
I'll've == I will have
I'd == I would
I'd've == I would have
Whatcha == What are you
amn't == am not
```

Figure: 7 – Sample contradictions used from contradictions library

b. **Add <START> and <END> tags** – The START and END tokens were added to the 'summary' start and end respectively. These tokens help the learning algorithm know the summary(headline) start's and end's.

c. **Add 'sostok' and 'eostok' tokens** - Again different tokens called 'sostok' and 'eostok' were added as start and end tokens respectively to the 'summary' - as later while using TensorFlow's Tokenizer – the tokenizer will filter the START and END tokens and covert them to lowercase – so these tokens are for us to determine summary's start and end points. Below is the sample summary data after adding the tokens

	text	summary
58733	selfstyled godman daati maharaj said 25yearold woman follower accused rape dter put allegations her even hanged put allegations her done anything wrong legal authorities look that added	sostok _START_ girl dter godman rape charge _END_ eostok
35236	10minute helicopter ride service cost â¢â□â'2900 launched sunday visitors statue unity worlds tallest statue situated gujarats kevadia statue unity inrated prime minister narendra modi october 31 gujarat government built statue cost around â¢â□â'3000 crore 33 months	sostok _START_ helicopter service launched statue unity â¢â□â'2900 _END_ eostok
7909	retiring england opener alastair cook got 33rd test hundred five overthrows said overthrow india pacer jasprit bumrah saved lot heartache cook cut ravindra jadeja delivery get 97 bumrah threw ball hard stumps resulting four overthrows i thank bumrah while cook added	sostok _START_ bumrahs overthrow saved lot heartache alastair cook _END_ eostok
79285	according delhi high court ruling event organisers pay royalty using copyrighted songs inform three copyright societies including indian performing right society phonographic performance ltd copyrighted work used one shall issue invoice royalty copyright	sostok _START_ event organisers pay royalty use copyrighted songs _END_ eostok
11827	former finance minister p chidambaram said current gst imperfect a mockery delayed two months the aim one indirect tax subsume indirect taxes gst fails achieve that said added congress press reduction rates 18 cap	sostok _START_ current gst mockery imperfect former finance minister _END_ eostok

Figure: 8 – Data 'summary' column after adding START and END & sostok and eostok tokens

Why use tokens both ways?
If a 'summary' sentence has the word 'start' or 'end' like "sports are going to end in the month of 2012" – if only _START_ and _END_ tokens were used, the tokenizer will convert the start and end then will stop decoding as the first end hits, which might not be a good summarization. So 'sostok' and 'eostok' tokens are used.
Note: Using just the 'sostok' and 'eostok' instead of START and END also did not result the model to generate in good summarization. I implemented this for a different dataset (all the news dataset) and the model gave undesired results.

d. **Determining the Maximum Sequence Length** – The next step is to determine the range of length of words where the maximum number of texts and summaries fall into. For this, the 'text' and 'summary' lists are stored in pandas object and a graph is plotted to determine the frequency ranges using 'matplotlib.pyplot'

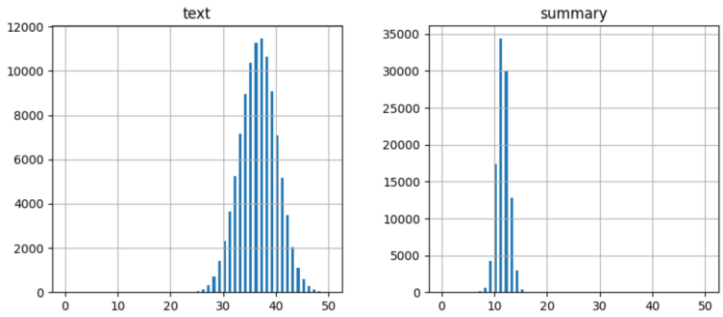


Figure: 9 – Visual graph to determine maximum number of words for 'Summary' and 'Text'

From the graph in Figure-9, we can determine the range where the maximum number of words fall into. For text, the range is 0-43 and for summary, the range is 0-13.
To find the percentage of words that fall into that range a function called 'get_word_percent' was created to check how many words in a column have length of the text less than or equal to the range.

```
# To check how many % in a columns has length (of the text) <= limit
def get_word_percent(column, limit):
    count = 0
    for sentence in column:
        if len(sentence.split()) <= limit:
            count += 1

    return round(count / len(column), 2)

# Check how many % of summary(headlines) have 0-13 words
print(get_word_percent(data.clean_summary, 13))

# Check how many % of text have 0-43 words
print(get_word_percent(data.clean_text, 43))
```

```
0.97
0.98
```

Figure: 10 – Percentage of words falling into the selected range for 'Summary' and 'Text'

We can see that 97% of the 'summary' pieces fall into the 0-13 range and 98% of the 'text' pieces fall into 0-43 range. Therefore, the model should summarize the text between 0-13 words for summary and 0-43 words for text.

```
max_summary_len = 13
max_text_len = 43
```

Figure: 11 – Maximum length selected from 'Summary' and 'Text'

e. **Rare Words and Tokenize the Text** – Now that we have the maximum length of 'summary' and 'text' – next step is to prepare the data to the model and the following steps were implemented in one function

- Split the data into train and test, using Sklearn train_test_split function - where 'text' is x and y is 'summary' and test_size is 20%.
- Tokenize the data to convert the raw cleaned text into sequence of integers. First, we create a Tokenizer object from Keras library and fit it to 'text' (x)
- Extract a sentence of integers from the text: The 'text_to_sequence' method of the tokenizer is called for every input and output text.
- Padding the sentences: A 'pad_sequences' method is used to pad zeros at the end of the sequences, so all sequences have the same length. Otherwise, we won't be able train the model on batches
- Find the percentage of occurrence of rare words(say, occurring less than 5 times) in the 'text' and 'summary' and

use them to tokenize the text by considering the total number of words minus the rare occurrences. Lastly convert text to numbers and create vocabulary for both x and y variables.

```
print(x_embedding_matrix.shape)
print(y_embedding_matrix.shape)
```

```
(104993, 300)
(39824, 300)
```

Figure: 12 – Embedding matrix shape for 'Text' (x) and 'Summary'(y)

f. **Word Embedding** - Here I used pre-trained word embedding model from GloVe and kept the embedding layer non-trainable and also increase the computational speed as we don't need to compute the embedding matrix. I used 300-dimensional GloVe word embedding.

5.5 Encoder-Decoder model creation – First import all the necessary libraries as shown below and define the encoder and decoder network s

```
import tensorflow as tf
import pickle
from tensorflow.keras import Input, Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import LSTM, Bidirectional, Dense, Embedding, TimeDistributed
```

Encoder – The input length that the encoder accepts is equal to the maximum text length which it was already estimated in Step 4. This is then given to an Embedding Layer of dimension (total number of words captured in the text vocabulary) x (number of nodes in an embedding layer) (calculated in Step 5; the 'x_vocab_size' variable). This is followed by three LSTM networks wherein each layer returns the LSTM output, as well as the hidden and cell states observed at the previous time steps.

Decoder - In the decoder, an embedding layer is defined followed by an LSTM network. The initial state of the LSTM network is the last hidden and cell states taken from the encoder. The output of the LSTM is given to a Dense layer wrapped in a TimeDistributed layer with an attached softmax activation function.

Altogether, the model accepts encoder (text) and decoder (summary) as input, and it outputs the summary. The prediction happens through predicting the upcoming word of the summary from the previous work of the summary.

Furthermore, each layers dimensions can be broken down into 4 different parts:

I. Input layer of Encoder and Decoder (2D -> 2D)
   - Input layer dimension: 2D (sequence_length, None)
   - Input data: 2D (sample_num, max_sequence_length)
   - Output data: 2D
     Note: sequence_length is the max_length unified by padding in pre-processing

II. Embedding layer of Encoder and Decoder (2D -> 3D)
   - Embedding layer dimension: 2D (sequence_length, vocab_size)
   - Input data: 2D (sequence_length, vocab_size)
   - Output data: 3D (num_samples, sequence_length, embedding_dims)
     Note: vocab_size is the number of unique words and data is word embedding in 300 dimensions.

III. LSTM layer of Encoder and Decoder (3D -> 3D)
   - Layer Dimension: 3D (hidden_units, sequence_length, embedding_dims)
   - Input data: 3D (num_samples, sequence_length, embedding_dims)
   - Output data: 3D (num_samples, sequence_length, hidden_units)
     Note: Important arguments of the LSTM layer are the 'return_state' and 'return_sequence' which are set to 'TRUE'

IV. Decoder output layer (3D -> 2D)
   - Output layer dimension: 2D (sequence_length, vocab_size)
   - Input data: 3D (num_samples, sequence_length, hidden_units)
   - Output data: 2D (sequence_length, vocab_size)

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 43)]         0           []

 embedding (Embedding)          (None, 43, 300)      31497900    ['input_1[0][0]']

 input_2 (InputLayer)           [(None, None)]       0           []

 lstm (LSTM)                    [(None, 43, 240),    519360      ['embedding[0][0]']
                                 (None, 240),
                                 (None, 240)]

 embedding_1 (Embedding)        (None, None, 300)    11947200    ['input_2[0][0]']

 lstm_1 (LSTM)                  [(None, 43, 240),    461760      ['lstm[0][0]']
                                 (None, 240),
                                 (None, 240)]

 lstm_2 (LSTM)                  [(None, None, 240),  519360      ['embedding_1[0][0]',
                                 (None, 240),                     'lstm_1[0][1]',
                                 (None, 240)]                     'lstm_1[0][2]']

 time_distributed (TimeDistribu  (None, None, 39824)  9597584    ['lstm_2[0][0]']
 ted)

==================================================================================================
Total params: 54,543,164
Trainable params: 23,045,264
Non-trainable params: 31,497,900
```

Figure: 13 – Sequence-to-Sequence Encoder Decoder model summary using LSTM

5.6 Training the model – Compiled the model and defined 'EarlyStopping' to stop training the model once the validation loss metric has stopped decreasing.

Next use the model.fit() method to fit the training data where you can define the batch size to be 128. Send the 'text' and 'summary' (excluding the last word in summary) as the input, and a reshape the 'summary' tensor comprising every word (starting from the second word) as the output.

```
history = model.fit(
    [x_train_padded, y_train_padded[:, :-1]],
    y_train_padded.reshape(y_train_padded.shape[0], y_train_padded.shape[1], 1)[:, 1:],
    epochs=num_epochs,
    batch_size=128,
    callbacks=callbacks,
    validation_data=(
        [x_val_padded, y_val_padded[:, :-1]],
        y_val_padded.reshape(y_val_padded.shape[0], y_val_padded.shape[1], 1)[:, 1:]
    )
)
```

Figure: 14 – Code to train Encoder Decoder model

The model was trained for 50 epochs and took around 4-5 hours to run.

5.7 Results - After 50 epochs the model had a validation accuracy of approximately 51% with validation loss of 3.35. As shown below the training and validation loss and accuracy metrics were plotted using 'matplotlib'.

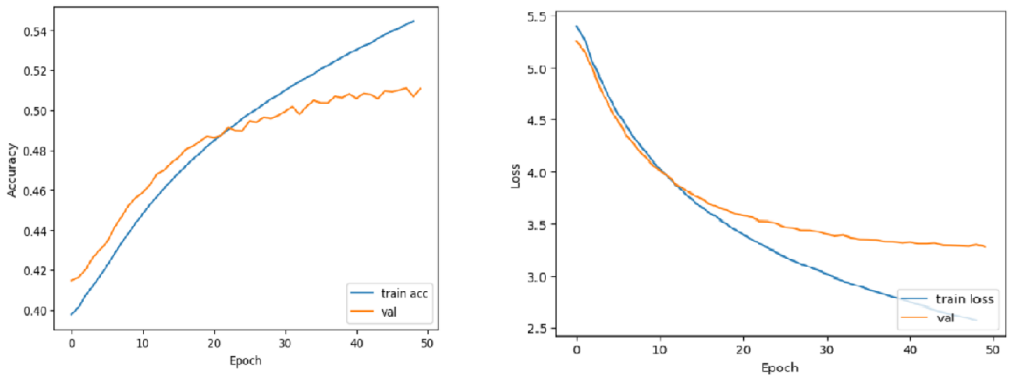


Figure: 15 – Visual results of the trained encoder-decoder model for 50 epochs

6. Model Inference

Now that the model is trained, we need to generate summaries from the given pieces of text. For inference, we should reverse-lookup token index to decode sequences back to something readable. To do this, first we reverse map the indices to words, which have been previously generated using text_to_sequence in step 4 (e). Then follow steps (or functions) are created

- Encoder decoder inference function - An encoder inference model accepts text and returns the output generated from the three LSTMs and hidden and cell states. A decoder inference model accepts the start of the sequence identifier (sostok) and predicts the upcoming word, eventually leading to predicting the whole summary. For this 'tensorflow.keras.Model()' object was used to create the inference models.
- Decoder_sequence() - Define a decode_sequence() function which accepts the input text and outputs the predicted summary. Start with sostok and continue generating words until eostok is encountered or the maximum length of the summary is reached. Predict the upcoming word from a given word by choosing the word which has the maximum probability attached and update the internal state of the decoder accordingly.

- Define two functions - seq2summary() and seq2text() which convert numeric-representation to string-representation of summary and text respectively.
- Finally, generate predictions by sending in the text. Below is the code to generate predictions for 10 news articles

```
# Testing on training data
for i in range(0, 10):
    print(f"# {i+1} News: ", seq2text(x_train_padded[i]))
    print("Original summary: ", seq2summary(y_train_padded[i]))
    print(
        "Predicted summary: ",
        decode_sequence_seq2seq_model_with_just_lstm(
            x_train_padded[i].reshape(1, max_text_len), encoder_model, decoder_model))
    print("")
```

Figure: 16 – Code to Generate summaries on training data for 10 news articles

Below are few notable summaries generated by the RNN/LSTM model

```
# 1 News:  salary allowances reimbursthements received punjab mlas displayed official website information immovable properties mlas mps state a
Original summary:  start salary allowances punjab mlas made public end
Predicted summary:   start punjab mla gets allowances salary allowances post end

# 2 News:  human resource development ministry thursday launched atal ranking institutions intion achievthements ariia promote intion research
Original summary:  start hrd ministry introduces atal ranking institutions end
Predicted summary:   start govt forms government schools indias best curriculum end

# 3 News:  extratorrent worlds second largest torrent site pirate bay shut permanently last week extratorrent mirrors goes offlinewe permanentl
Original summary:  start worlds second largest torrent site extratorrent shuts end
Predicted summary:   start worlds largest site shut site shut site end

# 4 News:  us president donald trump honoured north korean defector ji seongho first state union speech tuesday seongho defected 2006 tortured
Original summary:  start trump honours n korean defector state union speech end
Predicted summary:   start trump 1st lady kim jongun held n korea end

# 5 News:  several kashmiri twitter users regularly tweet conflict valley received themail twitters legal handle stating received official corr
Original summary:  start twitter warns kashmiri users tweeting conflict end
Predicted summary:   start twitter users slam twitter users india end
```

Figure: 17 – Generated(Predicted) summaries from the model

7. Conclusions

The Encoder-Decoder Sequence-to-Sequence Model (LSTM) build generated acceptable summaries from what it learned in the training texts. Although after 50 epochs, the predicted summaries were not exactly on par with the expected summaries, the model hasn't yet reached human-level intelligence; however, the gains made by the model definitely count for something. To attain more accurate results from this model, you can increase the size of the dataset, play around with the hyperparameters of the network, try to use Bidirectional LSTM, GRU and attention, and also increase the number of epochs.

8. Effort

The aim of this project is to learn and explore a state of art deep learning model that can be used for text summarization, an NLP concept. In the process of learning and creating the model I have experienced as well as learned to solve multiple issues/problems. Below are few listed

- To start off with, I implemented and tried to create the same encoder-decoder model on 'All the news' dataset and encountered a lot of data specific issues, as well as runtime issues because of the vast dataset
    - Had multiple google Colab crashes due to extended period of runtimes, for example, one epoch took more than an hour. To resolve this had to purchase Colab Pro
    - Various data issues due to which the model wasn't able to predict the summaries in the inference phrase.
    - With the huge dataset, 100,000+ articles and most importantly the 'text' maximum length was around 2000 words – which might be the main issue that the encoder-decoder model hasn't learned or performed well
    - Even though the model took more than 10-12 hours to run 10 epochs – noticed that model wasn't able to learn, and reverse predict the tokens from the decoder function
- On the other hand, the current working model with the 'news summary' dataset performed well with initial 20 epochs which took approximately 2-13 hours to complete.

9. References

[1602.06023] Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond (arxiv.org)

All the news | Kaggle

Attention — Seq2Seq Models. Sequence-to-sequence (abrv. Seq2Seq)… | by Pranay Dugar | Towards Data Science

How to Implement Seq2seq Model | cnvrg.io