Abstractive Text Summarization using Neural Attention Based Sequence-to-Sequence Model

INTERNSHIP PROJECT REPORT

by

RAHUL MAHANOT

Department of Avionics

Indian Institute of Space Science and Technology

Thiruvananthapuram

December, 2018

BONAFIDE CERTIFICATE

This is to certify that this project report entitled "**Abstractive Text Summarization using Neural Attention Based Sequence-to-Sequence Model**" submitted to **Indian Institute of Space Science and Technology**, **Thiruvananthapuram,** is a bonafide record of work done by **RAHUL MAHANOT** under my supervision from **04/12/2018** to **30/12/2018**

Dr. Deepak Mishra
Associate Professor
Dept. of Avionics, IIST

Place: Indian Institute of Space Science and Technology, Thiruvananthapuram

Date: 28/12/2018

Declaration by Author

This is to declare that this report has been written by me. No part of the report is plagiarized from other sources. All information included from other sources have been duly acknowledged. I aver that if any part of the report is found to be plagiarized, I shall take full responsibility for it.

Rahul Mahanot
Roll No: 16EC01045
B. Tech Student
IIT Bhubaneswar

Place: Indian Institute of Space Science and Technology, Thiruvananthapuram

Date: 28/12/2018

Table of Contents

Abstract

Text summarization is a process of giving a concise short description(or summary) to a long text. This Internship work has been majorly focused on using Deep Learning based Recurrent Neural Networks and neural Attention modelling to produce Abstractive summaries for long texts. I used a sequential encoder-decoder model to achieve summaries of text. I used two different types of language level models – Character level and Word level language model. Character level language model generates meaningful words but summary as a whole does not reflect the source text's meaning. However, word level language model proves to give satisfactory results for short length source texts. But it fails to give good output for long length source texts.

1. Introduction

Automatically generating concise short length descriptive summaries for longer text documents using Deep Learning methods is known as Text summarization. Text Summaries can be classified in two types – Extractive summary and Abstractive summary. Extractive summary uses important words or sentences directly from the source text to describe the source text. Whereas abstractive text summary describes the source text in short compressed paraphrased words, using words unseen in the source document. Building an abstractive text summarization model is a difficult task and involves complex language modelling. Example for abstractive text summary is give below.

Summary: *Not as Advertised*

Text: *Product arrived labeled as Jumbo Salted Peanuts...the peanuts were actually small sized unsalted. Not sure if this was an error or if the vendor intended to represent the product as "Jumbo".*

From the example above, it can be seen that the word *advertised* does not appear in the source text. However, the summary contains the word *advertised* which actually describes the source text quite appropriately. Summarization model used in this work is specifically designed to address this issue of generating abstractive text summaries using a vocabulary of words unseen in the source text document.

2. Model

In the recent past, deep-learning based models that map an input sequence into another output sequence, called sequence-to-sequence models, have been successful in many problems such as machine translation (Bahdanau et al., 2014), speech recognition (Bahdanau et al., 2015) and video captioning (Venugopalan et al., 2015). In the framework of sequence-to-sequence models, a very relevant model to our task is the attentional Recurrent Neural Network (RNN) encoder decoder model proposed in Bahdanau et al. (2014), which has produced state-of-the-art performance in machine translation (MT), which is also a natural language task.

I have used a Sequential encoder-decoder model (shown in *Figure 1*) as the baseline training model for generating text summaries. I propose two different models that can be efficiently used to generate accurate summaries for short length source texts. One is character level language model and the other is a word level language model. The character level language model considers each character in the source text and target summary as the input to the encoder and decoder respectively. The word level language

model uses am embedding layer to give words as input to the encode-decoder model. More details have been explained about the embedding layer in section 2.2.1

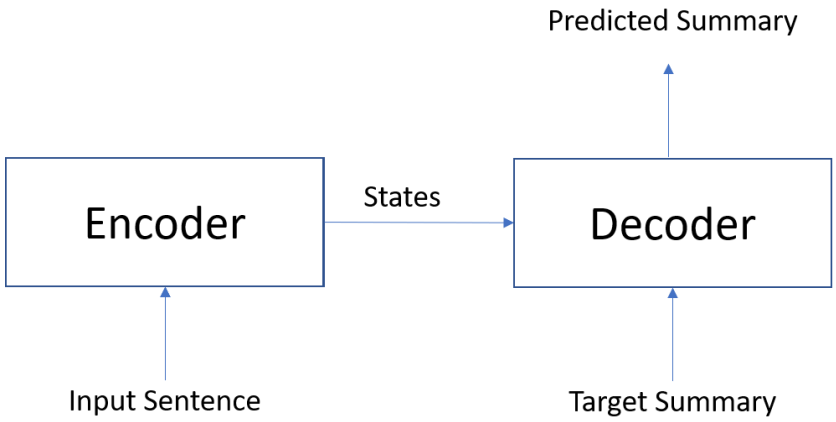

Figure 1: General Encoder - Decoder Model for text summarization.

Here, The encoder takes the source text as the input and encodes the whole text into a matrix. The decoder takes the encoded source text and the target summary as the input and maps both of them together to predict its own summary.

2.1. Character Level Language Model

As explained earlier in the previous section, Character level language model uses each character in the input text as the input to the encoder and decoder. Each unique character present in the entire dataset of source text is mapped to the index of a vector known as char_map vector. Each Character is encoded in a one-hot vector representation with 1 at the corresponding index of the character in the char_map vector and zeros at rest of the indices. Hence, each character is represented by a one-dimensional vector of size equal to the size of char_map vector, which is the total number of unique characters in the source text.

I used the Amazon Fine Food Reviews Dataset obtained from Kaggle for training purpose. This dataset contains 568454 text-summary pairs that can be used for training. An example from the dataset form this dataset is given in 1. Introduction.

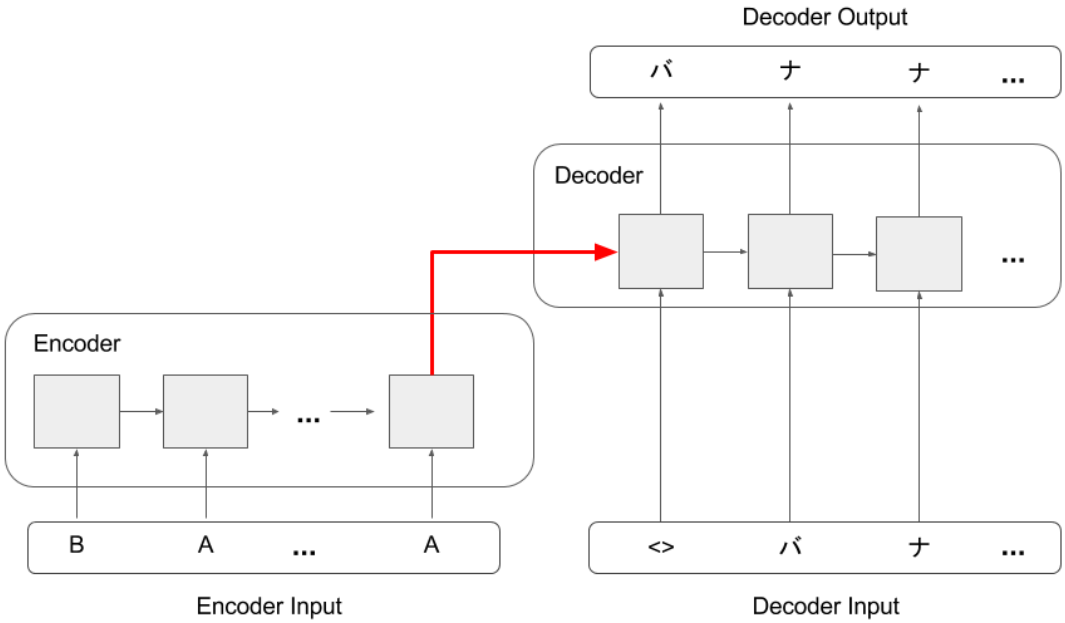

Figure 2: Character Level Language sequential encoder-decoder model. The characters are encoded in a one-hot char_map vector with index 1 at the corressponding index of the character in the char_map vector and rest all are zeros.

2.1.1. Data Pre-Processing

The Amazon Fine Food Reviews dataset has English contractions that may prove inefficient for training of the model. Therefore, these contractions were removed from both the source text and target summary. All Characters were converted to lower cases. All special characters were removed except '\n', '\t' and ' '. All encoder and decoder input vectors are padded to zero to fit the maximum length of the input sentence so as to be able to give fixed size input to the seq2seq model.

A char_map vector is created mapping all the unique characters in the source text with its indices. Each character is converted to its one-hot vector representation. Due to constraints on the Memory, all the examples could not be trained, instead only 8000 examples pairs were selected for training.

Number of samples: 8000
Number of unique input tokens: 51
Number of unique target tokens: 45
Max sequence length for inputs: 5256
Max sequence length for target: 131

encoder_input_data.shape = (8000, 5256, 51)
decoder_input_data.shape = (8000, 131, 45)
decoder_target_data.shape = (8000, 131, 45)

2.1.2. Sequence – to – Sequence Model

The Sequential model uses Long-Short Term Memory cell(LSTM) units in the RNN model to learn the mapping from source text to target summary. LSTM cells have proved to perform very good in understanding long-term dependencies in the input texts, thereby giving good output summary. The encoder – decoder model used for training are shown in figures 1 – 3.

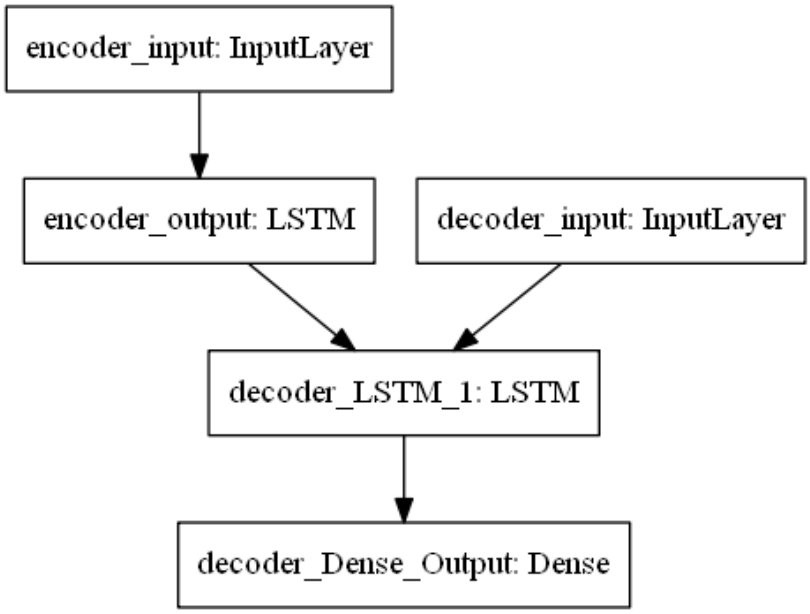

Figure 3: The complete sequence - to - sequence model for text summarization using character level language model. It uses LSTM layers to understanding to better understand long-term dependencies in the input texts and map it to target summary.

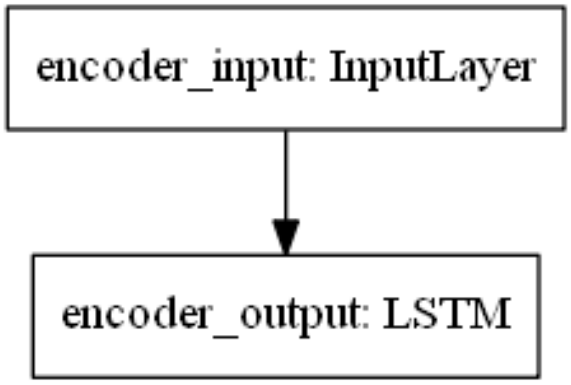

Figure 4: Encoder for Character Level Language Model. Encoder input is one-hot vector representation of each character in the input sentence. Encoder Input Shape is (8000, 5264, 51)

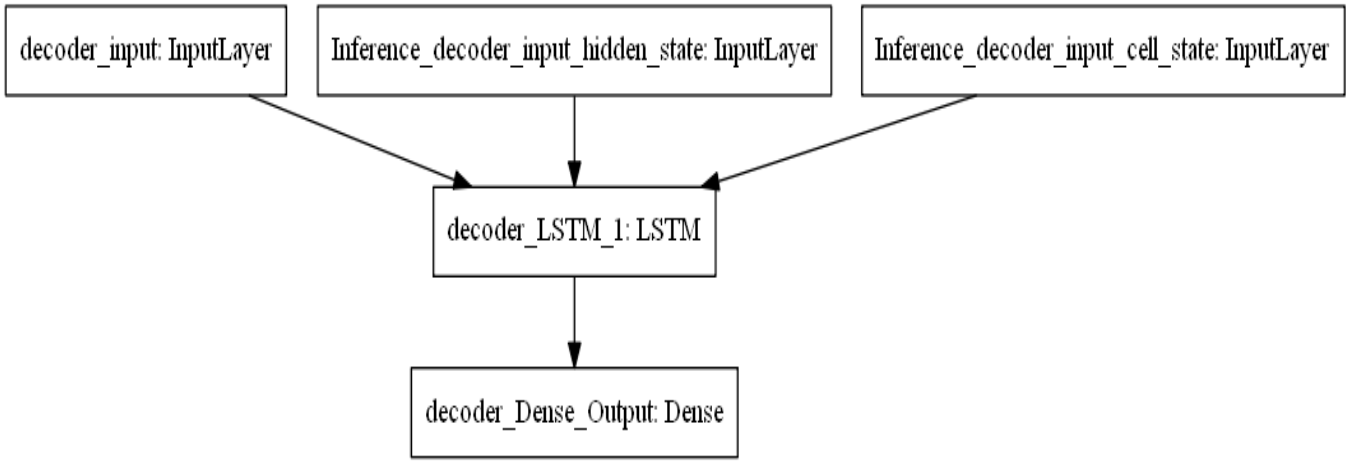

Figure 5: Decoder for Character Level Language Model. Encoder input is one-hot vector representation of each character in the input sentence. Encoder Input Shape is (8000, 131, 45)

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 5256, 51)	0	
decoder_input (InputLayer)	(None, 131, 45)	0	
encoder_output (LSTM)	[(None, 5256, 256),	315392	encoder_input[0][0]
decoder_LSTM_1 (LSTM)	[(None, 131, 256), (	309248	decoder_input[0][0] encoder_output[0][1] encoder_output[0][2]
decoder_Dense_Output (Dense)	(None, 131, 45)	11565	decoder_LSTM_1[0][0]

Total params: 636,205
Trainable params: 636,205
Non-trainable params: 0

Figure 6: Model Summary for Character Level Language Model.

2.1.3. Training and Results

The model was trained on a Tesla K80 GPU on Kaggle kernel. Batch size used was 64 and it was trained for 480 epochs.

After few epochs of training, the predicted output was not satisfactory, in the sense that the output characters were same for the whole sentence and for all examples. After complete training for 480 epochs, the predicted output was able to form meaningful words, but still the output was same for all the examples. This may have been the problem with decoding part. This can be overcome with more work on better decoding of predicted output. However, it should be noted that the predicted output was able to form meaningful words that were actually not present in the single example of input texts.

After few epochs of training:

Predicted Output: qqqqqqqqqqqqqqqqqqqqqq

Target Output: <All summaries>

Finally:

Predicted Output: good hot breakfast

Target Output: <All summaries>

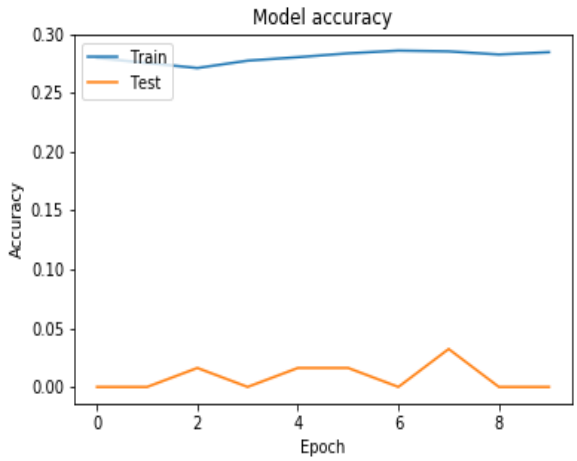

Figure 8: Model accuracy for Character Level Language model training for abstractive text summarization task

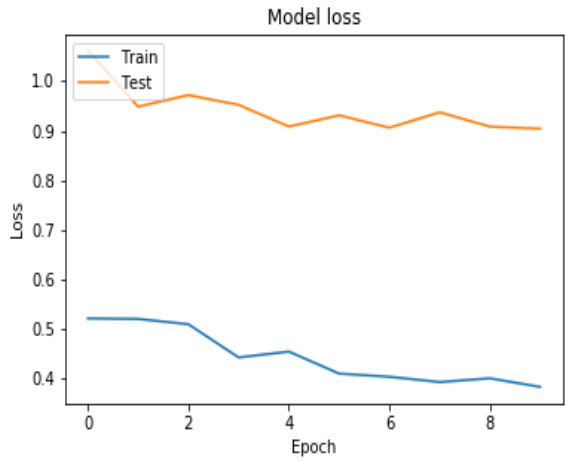

Figure 7: Model loss for Character level language model training for abstractive text summarization task.

It should be noted, here, that accuracy is not actually a correct metric to measure the performance of the language processing tasks. Bilingual Evaluation Understudy (BLEU) score is usually used as performance measure for such tasks. But this score was not used in this project work because of time constraints to implement the model.

2.2. Word Level Language Model

Word Level Language Models have recently proved to be performing very efficiently in Natural Language Processing tasks. This model uses Word Embedding layer output as the input matrix to the encoder and decoder. For this language model, all the unique words in the input source text are used to form a dictionary(or vocabulary) of words. All the words, now can be mapped to its index in the vocabulary. The input sentences are converted to a word_index vector whose elements contains the corresponding indices of the words from the vocabulary. This word_index vector is then fed to an embedding layer and encoder-decoder model that predicts the output. The output, however, is not a index representation vector but instead a one-hot vector. This is because the LSTM or dense layer can only output a SoftMax prediction, which can be converted to one-hot vector by proper decoding.

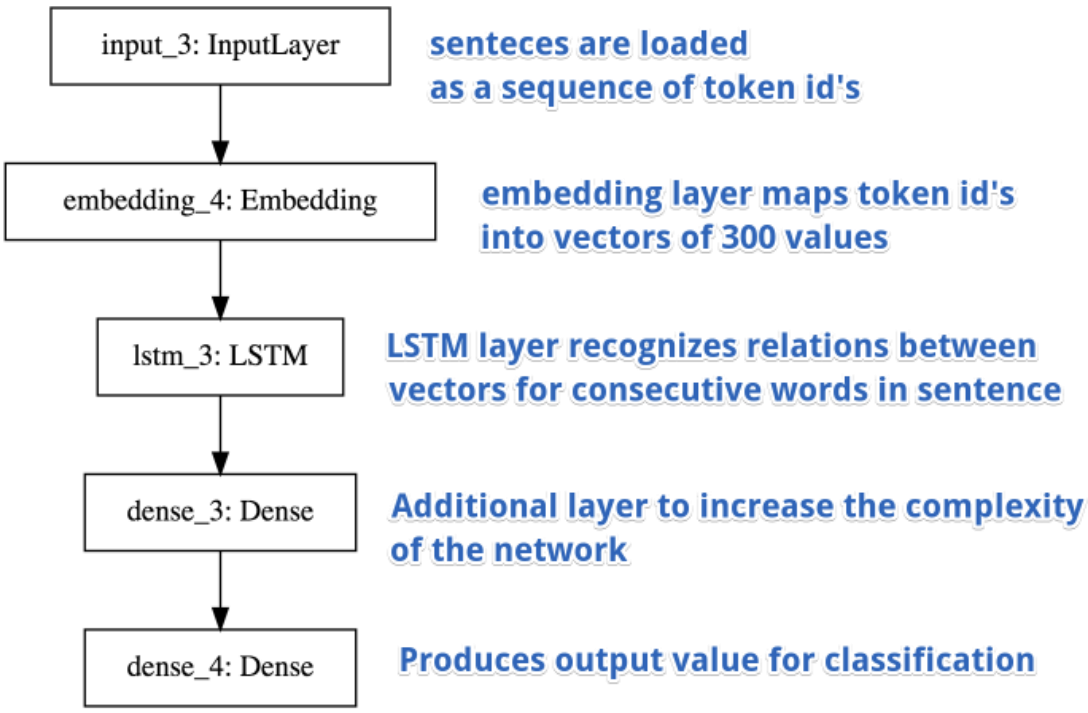

Figure 9: Generic Word Level Language Encoder Model. The input layer is a vector containing the corressponding indices of each word in the input sentence from the vocabulary. Token id, in the figure means the index of the word from the vocabulary.

2.2.1. Word Embeddings

Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc. It is basically a feature matrix that represents words in terms of various features like singular, plural, age, food, royal, name, etc.

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)
Gender	−1	1	-0.95	0.97
Royal	0.01	0.02	0.93	0.95
Age	0.03	0.02	0.70	0.69
Food	0.09	0.01	0.02	0.01

Figure 10: Featurization view of word embeddings. Columns are the words with their indices mentioned below them and rows are the features.

In the feature matrix shown in Figure 10, Columns are the words with their indices mentioned below them and rows are the features that describes the words with suitable numbers. Here, -1 is used to describe masculine gender and 1 for feminine gender. 0 is used for no relation with gender. 1 also describes how feature is closely related to a word. This word embedding feature matrix has proved to be great resource while training a large corpus of data.

The pretrained 50-dimensional GloVe Word Embeddings Matrix from Stanford NLP were used for the training purpose.

2.2.2. Attention Model

An attention-based contextual encoder that constructs a representation based on the generation context became the basis of machine translation models explained in the paper Bahdanau et al. (2014). This attention based encoder has proved to give state-of-the-art results in machine

translation. Another paper Alexander Rush et al. (2015) used this attention technique in the text summarization task for the first time that proved to give state-of-the-art results in text summarization tasks. In fact, the work presented in this report is majorly inspired by this paper itself. With an attention mechanism we no longer try to encode the full source sentence into a fixed-length vector. Rather, we allow the decoder to "attend" to different parts of the source sentence at each step of the output generation. Importantly, we let the model **learn** what to attend to based on the input sentence and what it has produced so far.

2.2.3. Encoder – Decoder Model

The Seq2Seq model used for word level language model used LSTM layers along with embedding layer to predict the output. The model diagrams are shown below.

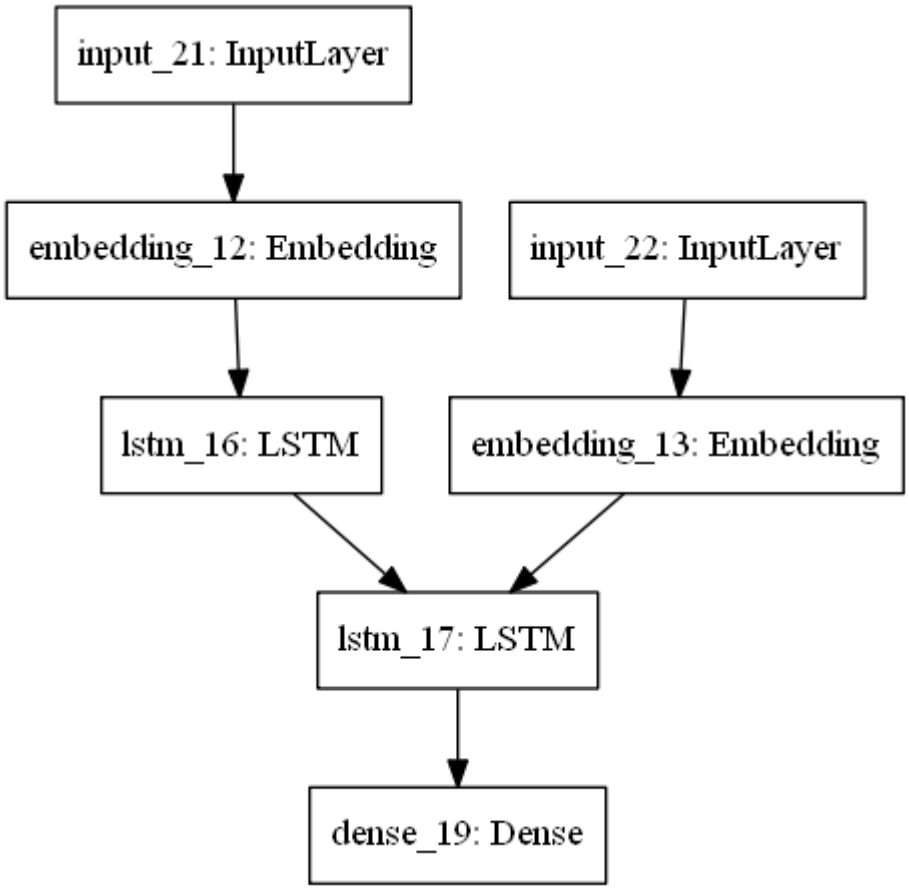

Figure 11: Word Level Language Model without Neural Attention Model

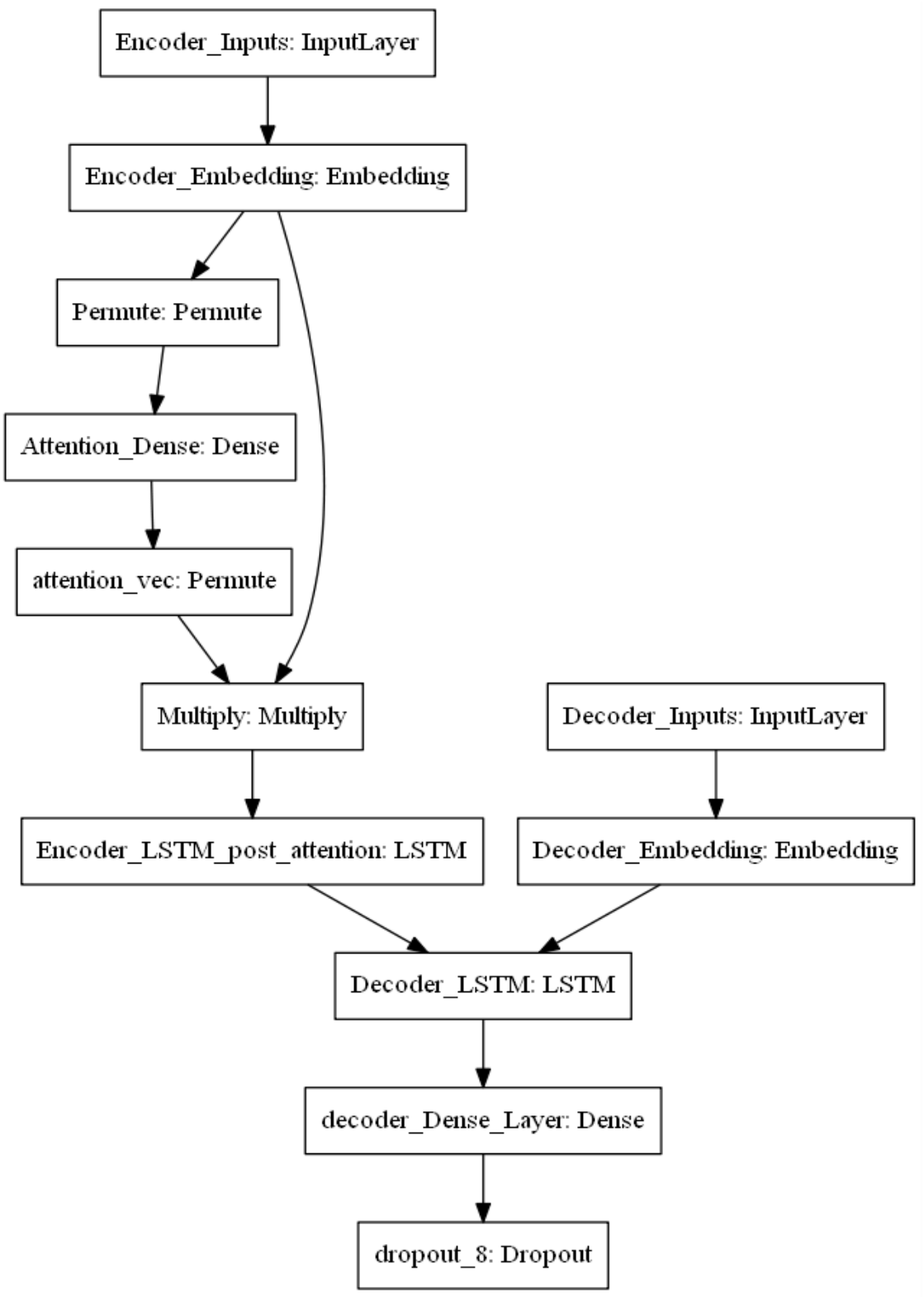

Figure 12: Complete Word Level Language Model with Attention Mechanism

2.2.4. Data Pre-Processing

As data processing done in Character level language model, here also all contractions were replaced and all characters were converted to lower cases. Special characters were removed and unknown words whose feature matrix is not present in the embedding matrix are replaced by a special token <unk>. I used a DUC dataset containing 38,03,957 text-summary pairs. An example pair from the dataset is shown below.

Text: *australia 's current account deficit shrunk by a record # . ## billion dollars # . ## billion us in the june quarter due to soaring commodity prices , figures released monday showed .*

Summary: *australian current account deficit narrows sharply*

All the input sentences were converted to their corresponding word_index vector and padded with zeros to match the maximum number of words in a single sentence in the whole dataset of source text. The word_index vector was reversed in its row, so that the index of the first word from the vocabulary appeared at the last index in the word_index vector. This is done to make the first word more closer to decoder.

Only 1000 examples were used for training due to memory constraints.

- Encoder Input. Shape = (100, 41)
- Target Data.shape = (100, 13)
- Output Data.shape = (100, 13, 400002)

Model was trained on a Tesla K80 GPU and also on Google Colab TPU.

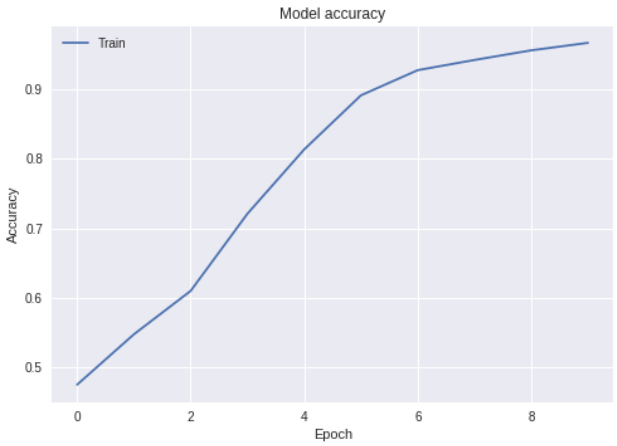

Figure 14: Model Accuracy for Word Level Language Model with Attention Mechanism

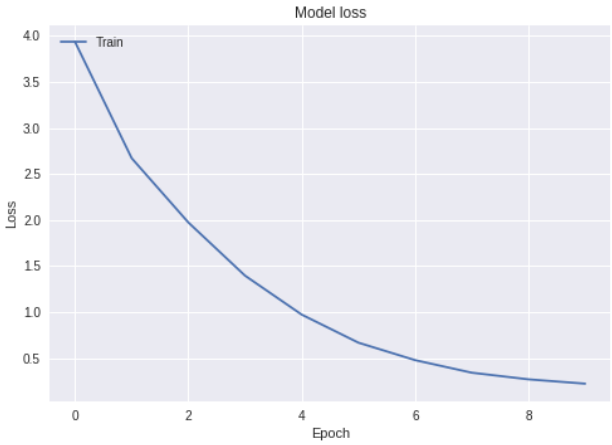

Figure 13: Model Loss for Word Level Language Model with Attention Mechanism