**Namrata Patel |** [**npatel29@syr.edu**](mailto:npatel29@syr.edu)

**Jeffrey Stanton**

**IST 664 - Natural Language Processing**

**12 December 2022**

***Natural Language Processing***

***Final Project Report***

**“All of the writing in this document is my original work, except for the code excerpts and any quoted material for which I have provided a citation."**

**Introduction**

Natural language processing (NLP) is the study of how computers interact with human languages in the domains of computer science, artificial intelligence, and linguistics. Its fundamental objective is to make it possible for computers to understand, read, and write human languages. NLP applications include automatic text summarization, sentiment analysis, dialogue systems, and machine translation.NLP algorithms are implemented into deep learning models to increase their efficiency and ability to understand and create natural language.

The notebook does a simple abstractive text summarization model that is based on a sequence-to-sequence with attention model.

**Data :**

The model is trained on Amazon's product reviews dataset available on Kaggle. The original dataset is very large so the training set only used a part of the dataset as it would take quite a long time to train the model in a local environment with only CPU usage.

**Author :**

The author of the notebook is Jina Kim (jk6653284) “Text Summarization”- Jina Kim

**Text summarizing:**

It is a natural language processing approach for extracting the most important details from an original text and packing them into a shorter, more concise summary.

Abstractive summarization is a text summarization technique that creates a summary using new words and phrases that accurately represent the significance of the original text.

In this particular notebook, we’re using abstractive summarization to extract the most important details from the reviews.

Abstractive summarization could be implemented using the Sequence-to-Sequence model.

**Technology Used :**

**Sequence-to-Sequence (seq2seq) model :**

A sequence-to-sequence (seq2seq) model is implemented for abstractive summarization, which generates a summary using new phrases and sentences that capture the meaning of the original text. The input sequence would be the original text, and the output sequence would be the summary. The encoder of the seq2seq model processes the input text and encodes it into a fixed-length representation, and the decoder of the seq2seq model transforms the encoded representation to generate the summary.

The seq2seq model is trained to minimize the difference between the predicted and actual summary, which will train to generate summaries that would accurately capture the meaning of the input text.

**Attention Model :**

An attention model is used along with a sequence-to-sequence (seq2seq) model to improve the performance of the seq2seq model on abstractive summarization tasks.

The input sequence is evaluated utilizing the attention model, which could allow the seq2seq model to focus on precise parts of the input text when generating the summary when the input text is long and complex.

**Code Walkthrough:**

**Importing and Cleaning Data:**

**Code Block 1: # Importing Python Libraries**

This code imports several libraries, including Pandas, Numpy, and TensorFlow which are used for data manipulation, analysis, scientific computing, machine learning, and deep learning. The code defines a function ‘zero\_state\_tensors’, which is used to initialize the hidden state of the recurrent neural network used in the model.

Some new packages we used are sumy which does the automatic summary technique and text comparison.

**Data Training:**

**Code Block 2: # Data Training**

Defining English contractions(shortened forms of a word or phrase) and replacing them with longer forms

The function will perform text cleaning, word conversion, word formatting, and character and stop words removal for Training.

The word\_count function then counts the number of occurrences of each word in the list of strings and adds the counts to the dictionary and prints the size of the vocabulary.

**Code Block 3: # CN**

Load the ConceptNet Numberbatch embeddings to capture the meaning of the concepts in a way that can be text summarization and combine data using number batch.

The code then calculates the number of words that are missing and are used more than a certain threshold from ConceptNet Numberbatch (CN) and prints the ratio of missing words to the total number of words.

**Code Block 3: # Word Usage**

The code creates a new vocabulary of words that appear more than a particular time in a dataset or are present in pre-defined embeddings.

Further, it calculates and prints the ratio of words that will be used in the new vocabulary to the total unique words and the percentage of words that will be used.

The embedding\_dim function is used to match the embedding dimensions to CN's vectors. Creating a word embedding matrix for a given vocabulary by first initializing the matrix with 0 and further populating it with pre-trained embedding from CN.

The code will further print the results of rows in a matrix matching the vocabulary size.

**Code Block 4: #UNK**

The function convert\_to\_ints takes text, a word count, and an unknown word count (UNKs) as input and converts each word to an integer using a vocab\_to\_int function, with unknown words mapped to UNK integers. The function will also add an end-of-sentence (EOS) token to each text end. It further applies the integer to clean\_summaries and clean\_texts.

Lastly, it prints the total words and unk in headlines and the percentage of words that are unk.

**Code Block 5: # Data Frame**

The function create\_lengths input text and returns a data frame of sentence lengths for the text. The lengths\_summaries prints summary statistics for each of the data frames that contain count, mean, standard deviation, minimum, maximum, and quantiles of the sentence lengths in a text.

**Code Block 6: # Sortting**

The function unk\_counter will count the number of UNKs in a sentence.

It further sorts pairs of texts, int\_summaries, and int\_texts, with text length in int\_texts.

It will further filter the texts using minimum and maximum lengths of summaries, texts, and limits on the number of unknown words (UNKs) permitted in each summary and text. Lastly, it checks if the lengths of the resulting sorted lists of summaries and texts match and prints them.

**Building seq2seq with attention model :**

**Code Block 7: # Modeling**

The function model\_inputs() creates placeholders for inputs to a model in TensorFlow.

The placeholders are for the input data, target data, learning rate, dropout keep probability, summary length, maximum summary length, and text length.

The tf.placholder can be used to ensure that the decoder does not produce output that exceeds the maximum allowed summary length.holder () function creates a placeholder tensor that can target data feeding.

The ‘None’ values in the shape specify that a tensor can hold an arbitrary number of elements in that dimension.

The maximum value is useful to ensure that the decoder will not produce output that may exceed the maximum allowed summary length.

The function strided\_slice operation removes the last word ID from a data batch.

Tf.concat function is used to contact the integer values to the batch beginning.

**Code Block 8:**

**# Encoding Layer**

The function defines an encoding layer for a neural network. It creates each iteration that defines a new RNN layer with a unique scope and this iteration creates two LSTM cells. The dropout technique in neural networks is used to prevent overfitting by randomly dropping a particular fraction of the inputs to a layer

The bidirectional\_dynamic\_rnn function applies the RNN to the input data, which processes the input data in forward and backward directions and concatenates the output results.

The encoder layer will encode the input data into a fixed-length vector representation, which will extract the meaning and important information from the input data.

**#Decoding Layer**

The function defines a decoding layer for a neural network. The ‘training helper function is used during training to assist the decoder in processing the input data.

The dynamic\_decode function is used to process the input data using the BasicDecoder object, which returns the outputs of the network before they are passed through the output layer and the final state of the RNN.

The decoder layer uses vector representation, to generate the output sequence, one element at a time.

The encoder and decoder layers work together to generate an output sequence that is relevant and readable to the input sequence.

**#Inference**

The code defines a function that creates logits for a text summarization model using an RNN decoder. The function takes several arguments and further applies a greedy decoding algorithm to the RNN decoder and returns the logits and the generated summary.

**#Attention layer**

The code defines a function that creates the decoding cell and attention mechanism for a text summarization model. The function takes arguments, like the input sequence, word embeddings, encoder output, and encoder state. The arguments define the model's hyperparameters, such as the vocabulary size, sequence lengths, maximum target length, RNN size, and dropout probability. The function applies a Bahdanau attention mechanism to an LSTM RNN cell and returns logits for the training and inference of the model.

**Code Block 9:**

The function creates a sequence-to-sequence model for text summarization. It takes several arguments, including the input and target sequences, and various hyperparameters like dropout probability and RNN size. It applies an encoding and decoding layer to the input and target sequences and returns the logits for training and inference, and to generate a summary.

The function pad\_batch pads create sentences in a batch so that they all have the same length. The get\_batch function will batch together summaries, texts, and the lengths of their sentences.

**# Hyperparameters**

The code sets the hyperparameters for a text summarization model which will be used to train the model and control its behavior.

**#Graph**

The code builds a graph for a text summarization model by defining the model inputs.

It further applies a sequence-to-sequence model to the inputs, using the defined hyperparameters. The model outputs are tensors for the training and inference logits.

The code creates a function to apply the gradients and update the parameters. The built graph is further trained.

**Code Block 10: Training the model using the training dataset**

Firstly we subset the data for training the model. It prints the length of the longest and shortest text length of the subset data.

The code creates an optimizer using the Adam algorithm(Adaptive Moment Estimation- an optimization algorithm) and a specified learning rate. The sequence\_loss defines a loss function method from the seq2seq module.

**#Train the model**

The trained model with Adam optimizer and a specified learning rate.

The code implements learning rate decay, early stopping, and checkpointing to hold the best model during training.

The training approach is broken down into epochs, each epoch is data divided into batches and fed to the model to compute the loss and update the model parameters.

The code also displays the training loss and update loss after every 20 batches and 3 epochs If the updated loss does not decrease for 3 updates, the training process is stopped early.

**Code Block 11: Generating and comparing summaries**

The text\_to\_seq function takes in a text string and prepares it for input to the trained model. The function foremost cleans the text and converts it to a sequence of integers and the summarize\_text function generates a summary for a given input text using a trained model. The trained model is loaded and fed using the input text to the model to generate a summary without padding. Finally, the function measures the time taken to generate the summary and prints it to the screen.

**#Sumy’s text rank**

A Sumy package is used which will perform automatic summarization of texts by selecting the most important sentences and providing a brief description of them.

TextRank algorithm from the Sumy library is used for creating an extractive summary of a given text.

The plain text parser function from Sumy is used to parse the input text.

TextRankSummarizer function will generate the summary and help to specify the language of the text and the number of sentences you want to include in the summary.

The compare\_text function is used for comparing two different summaries of a given text.

We’ll test the comparison between text rank and seq-2-seq using different examples to find the different model results.

**Conclusion :**

The notebook is for generating an abstractive summarization of Amazon reviews using a seq-2-seq model and attention model.

The seq-2-seq model is trained using an encoder and decoder where the encoder encodes the input reviews into a vector representation to extract important features from the input data.

The decoder will use the vector representation to generate output from the important features of the input data.

The first step for the notebook was doing Data cleaning and then preparing the data for training the model.

The seq-2-seq was built using the attention model with encoding and decoding. As the

dataset is very large we take a subset of data for training the model.

While training the model, the average loss for the update was recorded: as 0.125.

After the model is trained, the summaries of text input are generated and compared.

Textrank uses a graphical-based ranking algorithm to identify the most important sentences in the input text and combines them to generate the summary.

We compared the seq2seq with the attention text summarization model using TensorFlow, to the next rank's extractive summarization method.

The comparison example predicted that the seq2seq attention model produces a much more relevant and coherent result. If we compare the time taken to summarize is much slower than the extractive text rank method.

The abstractive summarization method is slower, using a limited training set as well.

Seq2seq with attention model delivers more coherent and relevant summaries, but it is slower and requires more computation and training data. However, Textrank is faster and simpler than seq-2seq but produces less accurate summarization.