Text Summarization with Amazon Review Data

0. Description

1. **Project** : This project is about building a model to summarize Amazon food reviews.
2. **Data** : The title of the data is “Amazon Fine Food Reviews” and I downloaded from Kaggle.
3. **Tools** : Python, Tensorflow 1.13.1
4. **Model** : Bidirectional RNN and LSTMs for encoder, Attention model for decoder. This model is similar to Xin Pan’s and Peter Liu’s model from “Sequence-to-Sequence with Attention Model for Text Summarization”.
5. **Sections** :
    - Inspecting the Data
    - Preparing the Data
    - Building the Model
    - Training the Model
    - Making Our Own Summaries
    - Evaluation
6. **References** : Following this project and modifying it, I made this project.

Import Modules

1. Inspecting the Data

In this section, I read data and checked number of instances, number of missing values, and dropped null values and unneeded features.

2. Preparing the Data

Before we use the data, we need to clean the data and split them.

To **clean** the data, I made a module named **clean_text**. This module does following thins.

- convert all characters to lowercase
- replace contractions with their longer forms
    - I used a list of contractions made by someone.
- remove any unwanted character
- remove stopwords from description

I **split** data into train set and test set. Since we have large enough data, train set has 95% of total data and test set has 5% of total data. **# maybe we need validation set here.**

After splitting the data I applied **clean_text** module to each set.

To check the size of vocabulary, I made a module named **count_words**. This module counts the number of occurrences of each word in a set of text by building word histogram as dictionary. (word : count)

To make word embedding, we used Conceptnet Numberbatch's (CN) embeddings. **embeddings_index** is a dictionary with word as a key, and embedding as a value. I added words that are not in CN but appears more than 20 times into embeddings_index by creating a random embedding for them. Special tokens such as <UNK>, <PAD>, <EOS>, <GO> are also added.

I made a dictionary **vocab_to_int** that words in CN or words appear more than 20 times are mapped with integer values.

To make index of each word, I defined a module named **convert_to_ints**. This module converts words in a text into an integer array. For each sentence, check each words and if the word is in **vocab_to_int**, the value of the is appended, otherwise value of <UNK> is appended. At the end of the sentence, value of <EOS> is appended.

Apply the module to train data. Now, we the summaries and texts in train data are represented as array of integers.

To check lengths of each sentences, I defined a module named **create_lengths**. This module creates a data frame of sentence lengths.

Apply **create_lengths** to summaries and texts in train data set.

I defined a module named **unk_counter** that counts the number of the UNK appears in a sentence.

To make sure I use meaningful data, I am not going to use any reviews that has more than 1 UNK in texts or any UNK in summaries. Also I am going to use reviews according to the lengths of their text and summary.

Now we have 403825 reviews.

3. Building the Model

model_inputs module is to create placeholders for input to the model.
process_encoding_input module is to remove the last word id from each batch and concat <GO> to beginning of each batch and returns input for decoder. This module returns input for decoder.
encoding_layer is a module that creates the encoding layer. It iterates creating each layer for **num_layers**. There are LSTM cell_forward and cell_backward and creates bidirectional rnn. This module returns output of encoder and state of it. (이부분 공부 좀 더 하자!)
training_decoding_layer module creates the training logits.
inference_decoding_layer module creates the inference logits.

decoding_layer is a module that creates decoding layer. It creates the decoding cell and attention for the training and inference decoding layers. It iterates creating the layers for **num_layers**. After creating cell, using **training_decoding_layer** or **inference_decoding_layer**, which are defined above, the module creates training decoding layer or inference decoding layer.
seq2seq_model creates training and inference logits using the previous functions. It uses Numberbatch's embeddings and the newly created ones as our embeddings, **word_embedding_matrix.** The rest of the module collects the output of predefined modules so that they are ready to be used to train the model and generate new summaries.

After creating encoder, decoder and seq2seq model, I created batches.
pad_sentence_batch module pads sentences with <PAD> so that each sentence of batch has the same length. Every sentence has the same length as the max sentence in the batch.
*batch size : defines the number of samples that will be propagated through the network.
get_batches creates batch summaries, texts, and the lengths of their sentences together.

Then we set the hyperparameters. I set the **epochs** to 100 in order to make the model becomes fully trained, and only early stopping (when the loss stops decreasing) will stop the training.

Now, we build the graph and set the graph to default to ensure that it is ready for training. First, we load the model inputs using **model_inputs** which creates the placeholder for input. Then we create the training and inference logits using **seq2seq_model**. Then we create tensors for the training and inference logits naming as 'logits' and 'predictions'. **mask** is the weights for sequence loss. Create the cost function and using AdamOptimizer, optimize the parameters.

4. Training the Model

We have built the model and tensorflow graph for the model. Now we will train the model. In this project, we used the subset of the data since we have too many data which will take too long time. From 200000 to 300000 among sorted data, the 100000 data is used for training.

Set other parameters such as **display_step**, and **checkpoint**. Checkpoint is used to save the trained model so that we do not have to train the model all the time.

Create a session and run the session to train the model. Iterating the whole epoch 100 times, for every batch, get the batches and train the model and print the information of epoch, batch, loss. If the update loss is at a new minimum, save the model since the model has become better. If the update loss does not decrease in 3 consecutive update checks, stop training (early stopping).

5. Test and Evaluate the Model

After training the model, we are going to test whether the model is trained well using test data set (28421 data). In this section, we will going to use ROUGE score as a evaluation metric.

text_to_seq is taking any text as an input and return the sequence of integers using **vocab_to_int.** This module prepares the text to be used for the model.

Putting **clean_texts_test** as an input, we will generate the **system_summary**. After saving it into a file, we are going to compare with **model_summary** which is the true summary in order to get rouge scores.

We load the saved model to **loaded_graph.** Get the batches of test data set then generate the **answer_logits**.

I created a dataframe **text_and_summary** which has three columns; text, system summary, and model summary.

list_to_file is a module to write the list into a file. Using this module, **model_sum.txt** and **system_sum.txt** files are created.

Using **FileRouge** module, we can get the rouge scores. Then I saved the scores into **rouge_score.csv** file.

Read the **rouge_score.csv** into a dataframe named as **rouge_score.** Since the scores are written as string, I cleaned the scores by removing unwanted characters, splitting into rouge-1-f, rouge-1-p, …., rouge-L-r, and converted into numeric values.

6. Discussion

The rouge score is as below.

	rouge-1-f	rouge-1-p	rouge-1-r	rouge-2-f	rouge-2-p	rouge-2-r	rouge-L-f	rouge-L-p	rouge-L-r
mean	0.116639	0.139479	0.115046	0.039031	0.045383	0.039212	0.104803	0.137613	0.113810
std	0.218001	0.256351	0.224888	0.161540	0.183428	0.166812	0.204875	0.253442	0.223454

As you can see, the score is not that high. On average, one word from the system summary overlaps the model summary. Considering the lengths of summaries are very short (mean of the summary length is 4.2), we can say that the model is generating pretty meaningful summaries.

Still, I cannot ensure that model is generating summaries good enough. I wish I could find other evaluation metrics that consider the similarities of system summaries and model summaries.

Also, I just used one model, so I did not have a chance to compare the result with results from other models. Later, I think I can try several other model for the same data.

One question : is there any ensemble model for text summarization?