Capstone Project 2
Final Report
8/25/2020

"Audio Moderator: A Speech to Translation Pipeline using LSTM and seq2seq"

Project Recap:

This project aims to translate Chinese audio to English text to assist in content moderation and foreign language audio/video sharing.

The Problem:

Most platforms that allow users to express themselves and share their own content have safeguards in place to monitor for abusive behaviors. For example, they may use machine learning to detect written hate speech or offensive images. However, there are two issues that make content moderation more difficult for these platforms: non-English language content, as well as audio, video, and steaming content. Under these conditions, the platform often has to rely on a great number of human content moderators. This slows the detection process, and the content then has the potential to harm many more people before it is removed.

This project tackles both the issues of non-English and audio content by building a pipeline to translate foreign language audio input into English text.

The Client:

This application is relevant to any social media platform with a global audience that allows users to post audio or video content. In particular, it is useful to companies that employ thousands of human content moderators, such as Facebook, YouTube, and Twitter, because it can save the moderators hours of having to sit through a video to determine if the content is allowable. It is also useful for other video-centric platforms with potentials for abuse, such as TikTok and Twitch. And although content moderation is an obvious implementation of a speech to translation application, there is also promise for use outside of the abuse space, such as achieving wider audiences for foreign language films on video streaming services like Netflix by offering automatic subtitles in a target language, or allowing podcasters to publish translated transcripts of their talks.

The Data:

The comes from the Tatoeba Project bilingual sentence pairs, available here. The 22,075 sentence pairs in the Chinese-English set have mostly short, but varied lengths, and no specialist vocabulary. The data comes in a tab-delineated .txt format consisting of three parts: and English sentence, the equivalent sentence written in simplified or traditional Chinese, and an attribution to the human translator on the project website. An example looks like the following:

“Show me an example. 给我个例子。 CC-BY 2.0 (France) Attribution: tatoeba.org #395402 (CK) & #490074 (fucongcong)”

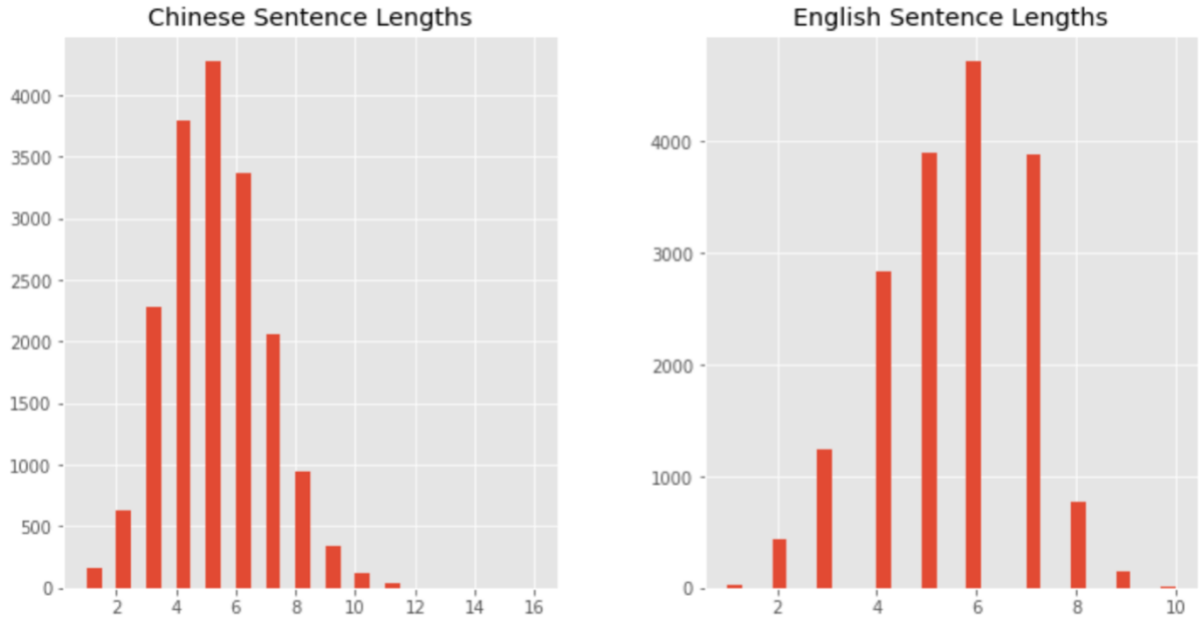
Data Wrangling Steps:

To clean the data, I dealt with each language individually. For the English sentence cleaning, I used string methods to lowercase all words and removed punctuation with regular expressions. For the Chinese, I used a separate library called Jieba to clean the data. Chinese is written without spaces, so each sentence appears as a single string. Jieba cuts Chinese sentences at word boundaries and separates the 1, 2, or 3+ character words with spaces. This enables us to tokenize Chinese as we would for languages written with alphabets. I also cleaned out Chinese punctuation using regular expressions, removed the attribution section, and then tokenized each language using Keras tokenizer. Unlike in other NLP tasks where we might perform lemmatization, stemming, or stop word removal to decrease noise in the data, for machine translation, we will leave these features in. Humans use stop words in speech, so our model needs to learn them too!

Exploration and Visualization:

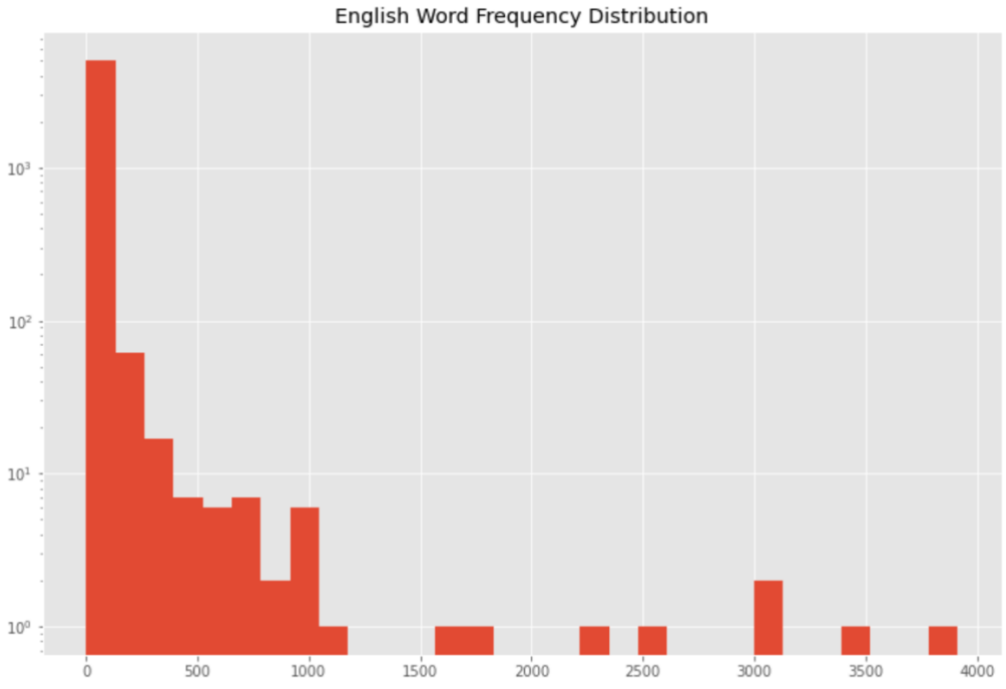
To explore the data, I performed some descriptive statistics on the tokenized data to look into sentence length and word frequency distributions. In the subset of the data I have used so far (18,000 sentences), the Chinese sentences have a vocabulary over double the size of the English vocabulary, 10,369 unique Chinese words as compared to 5,449 unique English words, and they also have a longer maximum sentence length, with 16 words, as compared to 11. On average, Chinese sentences had 5.2 words, and English sentences had 5.6 words.

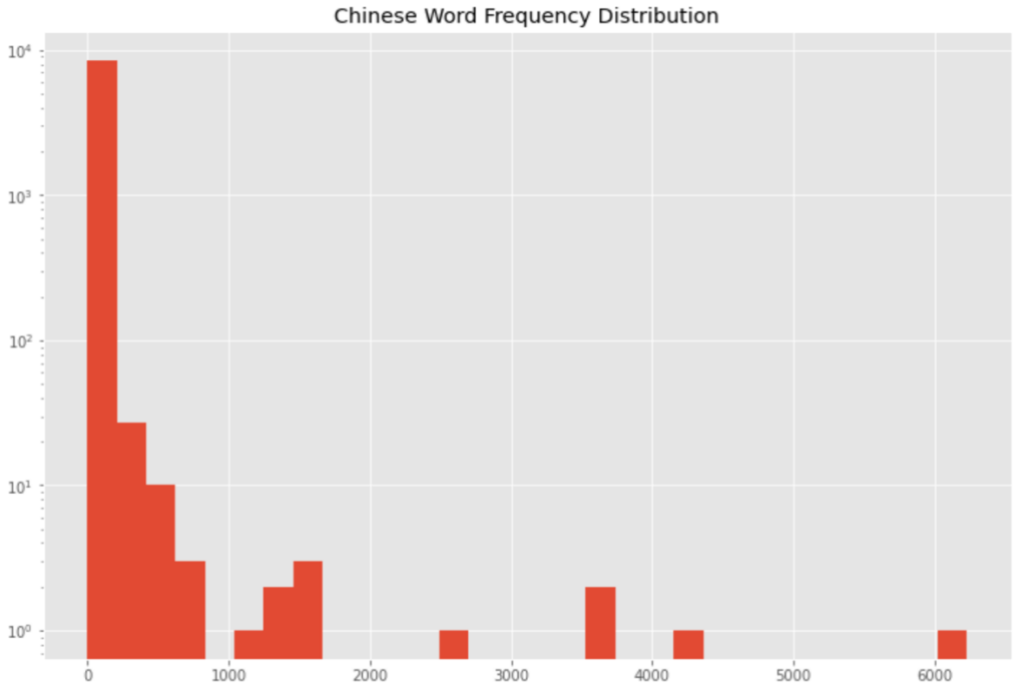
Histogram of Sentence Lengths for Dataset of 18000 Sentence



As for word frequencies, 51% of the unique Chinese vocabulary consisted of words that only occurred once in the data, whereas 40% of the unique English vocabulary was made up of words that only appeared once. For both languages, there were more than 100 words that appeared more than 100 times.

Histograms of Words Frequencies for Chinese and English Words





The data was split into training and testing sets, with a larger proportion of the data than usual assigned to the training, at least for the model building and tuning stages of this project. This was to try to slightly reduce the amount of unseen vocabulary in the testing set. Then the texts were vectorized, and the vectors were padded with zeros on the end to ensure they were all the same length.

Model Selection and Tuning:

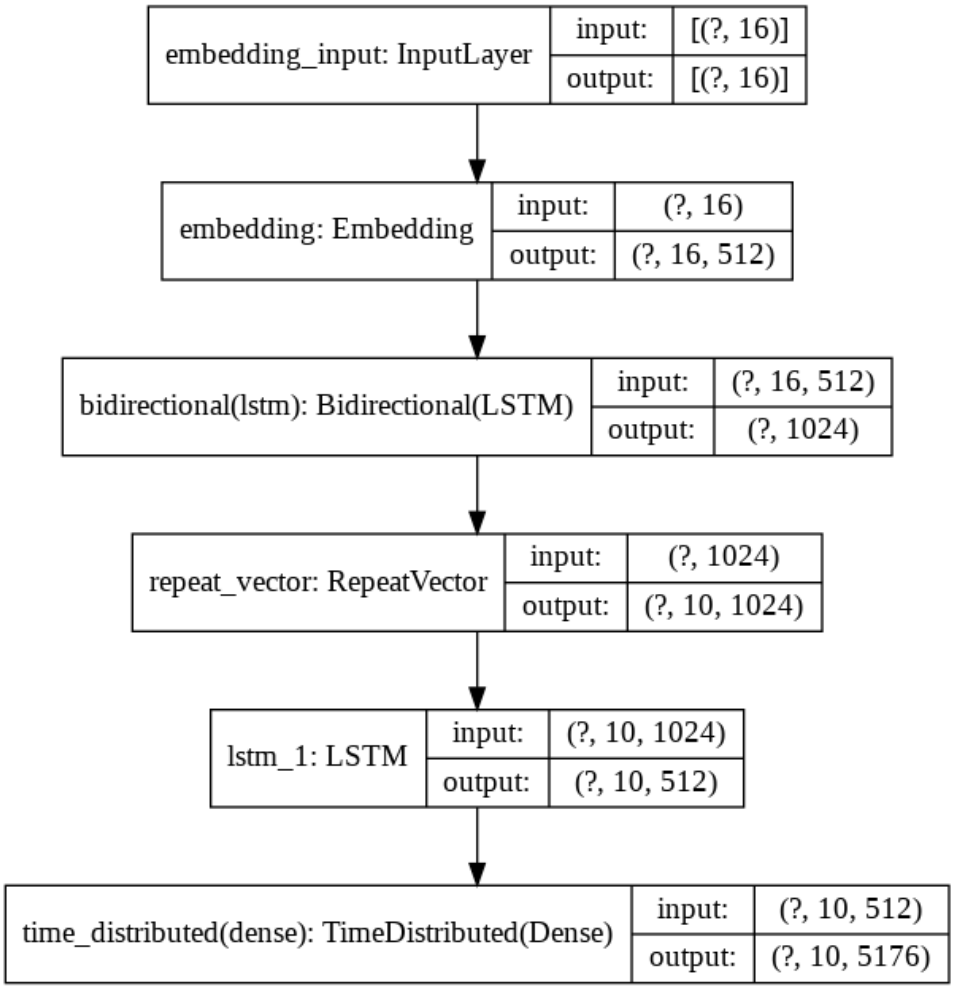
On the advice of my mentor, I selected an LSTM encoder-decoder model, and built the model using Keras with TensorFlow backend in Google Colaboratory, in order to take advantage of their free GPU. LSTM was a natural choice because it can remember previous information from a sequence and use it construct the new sequence. The first try produced a translation of English “the” for every Chinese sentence, so it was clear there was a lot of tuning to do.

Originally, I changed the LSTM decoder layer in the model to an Attention decoder. Keras does not currently have a built in Attention layer, so I used a custom one from the Datalogue Keras attention project. Attention is important in Machine translation because it allows the decoder layer to pay further attention to the output sequences of the encoding layer, allowing it to deal with input and output sequences with difference in length, especially long outputs, and pay further attention to the context from the encoding output to produce the decoded sequence. This has the potential to decrease word repetitions. However, since July when I implemented Attention, it seems that Keras has deprecated one of the layers (Recurrent) that was used in the custom Attention layer. I have not yet found out how to replace this, nor have I successfully been able to implement the new Attention layer from Keras. In the most current version, the decoder layer is another LSTM.

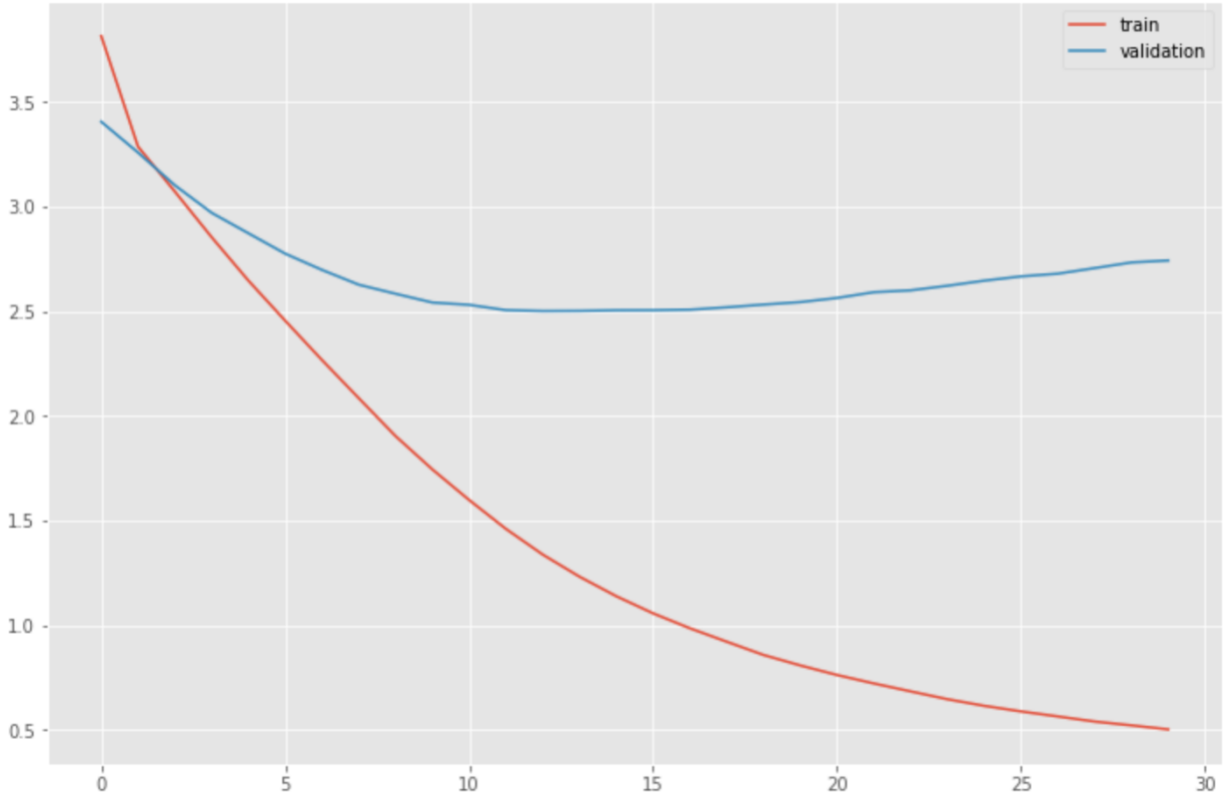
Next, I also added a bidirectional wrapper and a dropout layer after the LSTM encoding layer, to prevent overfitting. Bidirectionality extends the power of the LSTM by allowing it to read both forwards and backwards sequential inputs. Dropout allows only a random set of neurons at a specified rate to contribute to the model training in each pass. In other words, it drops a certain percentage of neurons randomly. This contributes to a model better at generalization because it is overall less reliant on neuron weightings that are specific to the training data. Additionally, I also adjusted the number of neurons in the layer and adjusted the learning rate in the optimizer function in order to configure how quickly and significantly the model weights were updated.

Although there is still further tuning to be done to this model, the current translations are significantly better than the first round, especially for the training data.

Model Plot of LSTM with Dropout Layer and Attention Decoder



Visualization of Model Training over 30 Epochs



Model Scoring:

Machine Translation quality is often evaluated using BLEU scores, so I have used this method to score my model as well. BLEU scoring follows a string-matching algorithm and compares matching ngrams in the predicted translation with human-translated target sentences. String matching cannot account for strings that are not identical but still semantically related, such as the same verb with different tenses, or synonyms. Additionally, human translators often produce a variety of translations for a single source sentence. Because difference in translations can occur, a BLEU score between 60% to 70% is generally thought to be good. The best score my model has received so far with the Attention decoder is 69.8% for unigrams and 58% for bigrams on the training data. In the most current version, without the Attention decoder, the best score on the training data has been 59.6% for unigrams and 44% for bigrams. All scores have been lower on the testing data.

Some example translations are as follows:

```
src=[悉尼 离 这里 很 远], target=[sydney is far from here], predicted=[sydney is far here here]
src=[您 不是 个 懦夫], target=[you are not a coward], predicted=[you are not a coward]
src=[我 支持 政治 改革], target=[i support political reform], predicted=[i support political reform]
src=[我们 会 很 忙], target=[we'll be busy], predicted=[we busy busy]
```

Identified problems:

The problem of overfitting is a work in progress that will require more research and patience with configuring parameters to optimize the model. However, there are a couple other issues that might also contribute to a less-than-optimal performance: poor segmenting of the Chinese data before tokenization, and the high percentages of words that only occur once in the dataset.

Although most of the Jieba segmenting seemed to be cut on word boundaries, occasionally incorrectly cut segments appeared. For example, the sentence "我不喜歡洋蔥的味道" ("I don't like the taste of onions") was split incorrectly at the verb. The character sequence 不喜歡洋蔥 (don't like onions) should have three segments 不 (don't) 喜歡 (like) 洋蔥 (onions), corresponding with the negated verb and its object, but instead it was segmented into two parts, 不喜 and 歡洋蔥, splitting the verb. This happened both with simplified and traditional scripts. After simplifying all characters, this particular error did not occur, but there was still some mis-segmentation, such as 队会赢 ("team will win") as a single segment, when it should probably be three individual ones: 队 (team) 会 (will) 赢 (win). Mis-segmenting can lead to treating certain parts of the sentence as new, unseen strings, which can produce errors in translation. For example, the "onions" in this sentence were often translated incorrectly. To solve this problem, I can experiment with different Chinese NLP libraries to perform segmentation, or I can segment by individual characters for tokenization. This splits words that are more than one character long, but maintains the correct order, and more importantly does not produce segments that are mistakes.

The other main issue is the high frequency of words that only occur once in the dataset. The words cause problems, especially when they appear in the testing data. Having not seen them before, the model will not be able to make a prediction, and instead will omit words or default to more frequent words, leading to sentences like "let's take a the on the sky" for 讓我們在公園裡散步吧 (Let's take a walk in the park). I think splitting characters individually may also help with this issue, as will converting all traditional characters to simplified (or vice versa). These methods will reduce vocabulary overall, since there will be less character combinations from n-grams, and no repeated vocabulary differing by script.

Further testing:

To try to counter some of the limitations above, I tested different datasets and model architectures to attempt to improve the predictions and scores for the machine translation component of the project. As the model stands in its most recent form, it generally stops improving on the validation data after about 10-15 epochs of training, and the accuracy score on this data generally does get very far above 60%. The training data does a bit better, with continued improvement throughout all epochs, and reaching accuracy of around 70%. These are clear signs of overfitting, since the model essentially stops learning and adjusting itself to improve on the validation set. Testing different datasets and architectures has helped me to learn more about the reasons it is overfitting, as well as rule out unhelpful solutions to the problem. For the datasets, I tested both longer sentences, as well as parsed the original Chinese

data differently than I had previously. For model architectures, I tested varying neuron and dropout numbers, as well as additional layers.

1. Testing Different Tokenization Methods

One aspect that stood out during data exploration was the number of singly-occurring words in both the English and Chinese sentences. We can expect problems around these words because if a word that only appears once occurs in the testing data, we can be sure that it was not learned during training and the translation, if it translates at all, will be inaccurate. I noticed that there were more than double the number of singly-occurring words in the Chinese sentences (51% of all vocabulary) than there in the English sentences (40% of all vocabulary). To attempt to decrease some of the noise in the data and reduce vocabulary size in general, at least on the Chinese side, I postulated that I might be able to tokenize the Chinese sentences into unigrams.

Chinese words can be one or more characters in length, and the same characters can occur in different combinations. For example, putting aside whether the usage is common or not, the words 合 (whole) and 適 (proper) can be also be combined into the words 合適 (suitable) and 適合 (to be suitable), comprising four individual tokens. However, if we split at unigram level rather than at word level, we only have two tokens. The Jieba package, which I have been using, splits by words of one or more characters. But it is worth checking to see if splitting by unigrams instead might decrease the vocabulary size, thereby enabling us to use more data, as well as improve the translation results, since there would be fewer singly-occurring characters in the testing set.

After splitting by unigrams, even with the data increased by 3,000 sentences, the total Chinese vocabulary size (number of unique words in the dataset) decreased from 8,600 to 2,590, although average sentence size increased from 5.1 words to 8.4 words. Singly-occurring words decreased from 4,422 to 434, from 51% of the vocabulary down to 17% of the vocabulary. During model training, training scores did not show much improvement over scores from word-level tokenization, although validation scores improved. In general the BLEU scores for both the training and testing sets were not better than with the prior tokenization strategy. Subjectively speaking, the randomized translation printouts for the training sets seemed similar to the previous attempts, but the translations of the testing data seemed slightly better.

2. Testing Longer Sentences

The dataset of longer sentences comes from the University of Macau corpus. The sentences are taken from the spoken language set, since the aim of this project is to eventually add an audio component to translate speech. Overall, this attempt was not successful. The longer sentences were computationally expensive, especially during sequence encoding. In fact, the RAM usage kept causing Google Colaboratory to crash, and in order to have a successful run through, I had to reduce the number of sentences down to 2000. With such little data, even when applying both word level and unigram tokenization, the results were unsurprisingly bad. All sentences translated as “the the the the the the the the the the the the,” exactly eleven “thes.” Although the

English sentence length average in this case was 19.6 words, the minimum sentence length was 10, and sentence with 11 words also occurred quite frequently. The word “the” is the most commonly occurring word in the English data.

3. Adjusting Model Architecture

The adjustments made to the model include testing varying neuron and dropout numbers, as well as additional layers. From further reading I learned that dropout layers actually dropout a specified portion of the input or output during training, and the argument recurrent_dropout in the LSTM layer induces dropout in the network neurons. Both types work to prevent overfitting, since the former works to prevent memorization by increasing randomness in the data seen by the model, and to prevent neuron weights from becoming fixed too quickly. In addition, I also tried changing the original LSTM and attention layers to two Bidirectional LSTM layers. The results in terms of accuracy, BLEU scores, and translation quality did not improve, but it showed promise in terms of reducing overfitting. In the previous iterations, the model would stop improving on the validation data at a certain point, but with these adjustments it was able to continue to train longer with continuous – if more incremental – improvement. It was able to continue to improve even after 60 epochs, although by that time the training graph did show signs of improvement slowing.

Audio implementation:

I added in a brief section to apply the model to make a translation given an audio .wav file. This functionality first converts Chinese speech to text using the python SpeechRecognition library. Then it cleans the Chinese text, converts it to a numeric sequence, makes a prediction with that sequence as input, and displays the prediction.

The audio files I used in my examples were sourced from Omniglot, a public website with general information on languages, including survival phrases. The sentences chosen are simple, basic phrases. In all cases speech the SpeechRecognition library converted the phrases accurately to text, and the numerous mistakes in the translations were made at the prediction step due to inadequacies in the model.

Conclusion:

Further limitations include a general insufficiency of data due to RAM runtime limitations put in place by Google Colaboratory. Without paying for additional space, I had to cut off the amount of data used for training. As we saw in the plots and descriptive statistics above, many words in this dataset only appeared once, which likely decreased the model’s ability to accurately embed them in the vector space, and which caused a significant amount of unseen data at the testing stages. Increasing the amount of data and thereby increasing the likelihood of vocabulary

repetitions in the data, is the best step I could take to further improve the model. Re-implementing the Attention decoder may also help to improve it.