Capstone Project 2

Final Report

Model Report

For

Translations Services Inc (TSI)

Executive Summary

The European Parliament conducts business in many languages due to the huge differences in languages and cultures that make up its members. The transcriptions of its meetings need to be translated into a multitude of languages. One of these is French.

Translations Services Inc (TSI) has been contracted by the European Parliament to translate conference transcriptions from English to French on a continual basis. Due to the overwhelming number of documents to translate, TSI hired me as their Data Scientist to build a Natural Language Processing (NLP) model to automatically translate the documents from English to French.

The data that was used for this project consists of European Parliament provided transcripts of sessions that are organized into two files: europarl-v7.fr-en.en (English sentences from the transcripts) and europarl-v7.fr-en.fr (French sentences from the transcripts). Each line/sentence in the English file corresponds to its translation on the same line number in the French file. This data set can be found on the LionBridge website: http://www.statmt.org/europarl/

This report will show the steps I took to gather the input data, load it, manipulate it to prepare it for fitting an NLP model and finally produce a model to use for translating European Parliament transcriptions from English to French. The report will conclude by assessing the performance of the model in automatically translating English sentences into French.

1. Environment and Data Set Up

1.1) Environment Set Up

For this project, since I knew I was going to need more processing power than my personal laptop could provide, I signed up for a Google Cloud Computing account. In doing so, you are given a $300.00 credit to use for testing purposes before you will be charged for their services. It is a good amount to start learning about Google Cloud Computing and how to set up a project there.

I took advantage of this opportunity and set up an instance with a decent CPU and added an NVIDIA K80 GPU for more processing power. The instance runs on a Linux DEBIAN Operating System.

I also built a Google Cloud bucket where I could upload my language input files.

1.2) Data Load Set Up

I first had to install the Cloud Storage Client Python Library so that I could import cloud storage and initialize the client. With the client I was able to create a bucket object (Google Cloud object that holds data files) in which I stored the bucket name for the bucket I created on the Google Cloud Platform and where I uploaded my input files.

With the bucket object, I was able to generate a couple blob objects (Binary Large Objects), one for each input file. From the blob objects, I was able to generate bytes objects and finally string objects, each representing newline separated sentences in each language file.

At this point I was able to take a look at a few English sentences along with their French counterparts to make sure the data had loaded properly. Since this NLP project is about translating from one language to the next, a lot of the NLP techniques such as lemmatization and stemming were not used.

2. Processing the Data

2.1) Subsetting the Data

I initially attempted to use all the records from the English and French Input language files, but I quickly realized that the files were too large for even the Google Cloud Compute instance I spawned to handle. I had to resort to subsetting the files to a fraction of the available records. Each record consists of a sentence. I had to subset my language files to 6000 sentences each.

2.2) Tokenizing

After subsetting the data, I built a tokenize function to tokenize the sentences. The tokenize function takes in one parameter: a list of sentences to tokenize. The function returns the tokenized sentences and the tokenizer that was used. I then created a list of sentences to use in a call to my tokenize function to verify that it worked as I intended it to.

2.3) Padding

After tokenizing the data, I built a pad function to pad the tokens. The pad function takes in two parameters: a list of sequences (or in this case, tokens) and a length to pad the sequences to. The function returns a padded numpy array of sequences.

I then tested my pad function to verify that it worked as I intended it to by feeding it tokenized text.

2.4) Preprocess Pipeline

After padding the data, I built a preprocess pipeline function that takes in two parameters: a list of English sentences and a list of French sentences. The function tokenizes the English and French lists and then pads them. The function returns a preprocessed English list, a preprocessed French list, an English tokenizer and a French tokenizer.

Here, I tested my preprocess function by feeding it a list of English sentences and a list of French equivalent sentences. Then, I took a look at an example of a preprocessed English sentence by first looking at it prior to using the function and then looked at its equivalent after preprocessing to verify it had been tokenized and then padded.

3) Modeling

3.1) Import Keras packages for RNN build

At this point I was ready to start building models, so I started by importing Keras packages.

3.2) Create logits_to_text function

I then created a function called logits_to_text to turn logits from a neural network into text using the tokenizer. The function takes two parameters: logits from a neural network, and a Keras Tokenizer fit on the labels. As the function name implies, it returns a String that represents the text of the logits.

3.3) Create embed_model function, use it, train and predict

I then created a function called embed_model to Build and train a RNN model using word embedding on the input and output. It takes four parameters: a tuple of input shape, the Length of output sequence, the Number of unique English words in the dataset, and the Number of unique French words in the dataset. This function returns a built Keras embedded model that has not been trained.

Next I reshape the input, use the above function to build an embedded RNN model and I train it. I then predict the first English sentence with the model and use my logits_to_text function to transform the logits back to text to take a look at the translation of the first English sentence. Here I notice the 4 word translation is off by a single word.

The French translation of the English sentence “**Resumption of the Session**” should have been “**reprise de la session**”, but it actually translated to “**reprise de de session**”.

3.4) Create final_predictions function and use it

I then created a function called final_predictions to get predictions using a final model. It takes four parameters: preprocessed English data, preprocessed French data, an English tokenizer, and a French tokenizer. It trains a neural network by calling embed_model to build an embedded model and then it fits it and makes predictions on a couple sentences.

I call the function by passing in preprocessed English sentences, preprocessed French sentences, an English tokenizer, and a French tokenizer.

The first sentence has 12 words and the translation model has quite a hard time handling that. The French translation of the English sentence “**You have requested a debate on this subject in the upcoming days**”, should have been translated to “**Vous avez souhaité un débat à ce sujet dans les prochains jours**”, but it actually translated to “**vous vous de un débat sur ce sujet sujet sujet ce <PAD> ce de de de de**” which is quite a ways off.

The second sentence is the same I previously predicted with the embedded model, and the final model handles that one the same as before.

Summary and Next Steps

To conclude, I used a Google Cloud Platform account as my set up for modeling. I uploaded the two language data files to a Google Bucket on the Platform. I then pulled those files in through my code and loaded the data in its entirety before subsetting it to a more manageable chunk of data for the processing power at my disposition. With the subset of the data, I then tokenized the sentences in each language and padded the tokens by using my preprocessing function.

After that I built an embedded GRU (Gated Recurrent Unit) RNN model, trained it, and ran a translation of a simple sentence. It did fairly well then.

From that point I built a final model using the embedded GRU RNN model and tested the results this time against a couple sentences. The first sentence I tested was 12 words long and the translation was not ideal. The second sentence was the same one I had tested with the embedded GRU RNN model and it had the same results.

Overall, I had some success with my translation model, but there definitely still is room to grow. I would venture to guess that because of my machine setup, despite using Google Cloud Platform for better processing power, I still had to limit the number of sentences I used to train the model, and that may be the reason why the performance was not quite as good as I had originally anticipated.

My next steps to improve the results of my natural language translation project would be to:

1) Upgrade the hardware I used to generate the model by spawning a new instance on the Google Cloud Computing environment with more memory and more GPUs than just one.
2) Implement an LSTM (Long Short Term Memory) RNN Model as it appears from reading a few articles online that the LSTM, although more complex than the GRU to implement, provides for better results when faced with longer sequences to translate.