**NMT for Hindi-English languages using LSTMs**

**For Training the model**

1. Pre-requisites: python 3, pandas, numpy, keras (with tensorflow backend), nltk
2. Input data: two text files, in this case, files containing Hindi and English sentences, separated by ‘\n’
3. Instructions: Place your input files in the *preProcessing* folder and change the inputData.py file accordingly.

On any python terminal, run the following command:

python main.py

**Files explanation**

1. Folder: **preProcessing**
2. *inputData.py* - Reads the lines from both text files and stores in a pandas data frame, with two columns, one for each language
3. *preProcessing.py* – Cleans the text, filters short sentences (up to a length of 30 words), generates tokens, and create indices for all the tokens. These set of functions basically create the vocabularies of both languages. In some functions, the argument ‘numsent‘ is hard coded as 150000. This just means that if the total number of sentences is not assigned, it can take this as default value. In other cases, this value gets overridden.
4. In the main function, this clean data can be saved to a pickle file, so that the next time, the data can directly be loaded from the pickle file, instead of performing the cleaning and pre-processing again.
5. Folder: **training**
6. *model.py* – Creates encoder and decoder matrices for training and test data. Before creating the model, define the embedding\_size in the main file (50, in this case). The create\_model() function is where all the model specifications are defined, and the model is created. Apart from the training model, two separate models i.e. encoder and decoder models are created. These are the inference models, required later for predictions. All the models are returned.
7. *training.py* – Define all the training parameters. In the model.fit\_generator()function, the validation\_steps argument is changed according to the test data size. (In this case, it is 6250. This value is calculated by dividing the number of test samples, by the batch\_size, i.e. 200000 / 32). Similarly, in the main file, where the training function is called, the steps\_per\_epoch is calculated, using the training samples, i.e. 800000 / 32 = 25000 approximately.
8. Keep saving the values of loss and accuracies, so that it can be plotted later.
9. The training took about 44 hours for 1 million sentence pairs. The model is saved after training is completed.
10. Folder: **predictions**

*predictions.py* - First, the indices are reverse mapped to words. Then, the decode\_sequence() function is used to make sample predictions. This is a greedy search technique, using only argmax on the probabilities.

1. Folder: **bleuScores**

*bleuScores.py* – To evaluate the model, the nltk corpus\_bleu library is used. This function calls the above decode\_sequence() function, saves the results in a separate list, called ‘predicted’. This list is compared to the ‘actual’ list, which is the target translation. The bleu scores are calculated using nltk’s built-in library.

**For making predictions using Beam Search**

*(use this only for checking a few example predictions, preferably one at a time. This method is computationally expensive, and hence requires much more time)*

Instructions: Most of the above instructions are same. Only a different main file is used in this case (this is designed for using Beam Search technique for predictions). On any python terminal, run the following command:

python main\_with\_beam.py

1. Load the previously cleaned data from the pickle files.
2. Folder: **training**

*model\_with\_beam.py* – Loads the previously trained model. Also, load the input and output values from the model into the encoder and decoder models as appropriate.

1. Folder: **predictions**

*pred\_with\_beam.py* – Another function called decode\_sequence\_beam() is defined to implement the Beam search idea for making predictions. This calls the beam\_search() function that implements the actual search logic. The previous decode\_sequence() function is also available in the file. (either can be called in the main function)

1. In the *main\_with\_beam.py* file, first a function is defined to tokenize the input. The make\_preds() function processes the user’s input, and calls the beam search functions. For this, the file hindi\_input.txt is read. This is the source file, containing a sample sentence of the source language. After the function executes, the output is written into the eng\_output.txt file. This is the translated sentence i.e. the target language.

**Note:** An easier, more graphical way to perform sample predictions is to use this *ipython* notebook, present in the same folder:

***hindi\_eng\_seq2seq\_word\_1M-BEAM-transfer\_learning.ipynb***