Hassan Sherwani **19.09.2019**

Problem Statement

The objective is to convert an English sentence to its German counterpart using a Neural Machine Translation (NMT) system. I divided my assignment into different notebooks. If all work is done in one file, then it may get very memory heavy and it is not easy to make changes. I will explain content and role of each notebook briefly.

1-Preprocessing:

1.1)-Combining data

As data is in files is from 2009-2016, I combined them in one full data file. I checked files if there are any special character or encoding. I found that there is an NEL line terminator in files. So, I get rid of it using pandas with "quoting = csv.QUOTE_NONE" option. I cleaned both English and German version. I like to work with dataframes and tables hence, I made a file that would have English version in one column and German version in another column. This is by the way standard data style in most language models.

1.2)-Text Cleaning

The given data is unstructured (i.e text) so there are certain things need to be taken care of before jumping to the model building part.

- Get rid of the punctuation marks,special characters,extra spaces and then convert all the text to lower case.
- Remove number or digits
- Reduce vocabulary size using out of vocabulary concept.

1.3)-vectorize our text data

- Words or phrases from the vocabulary are mapped to vectors of real numbers i.e integer. We may use word embedding or Keras's *Tokenizer ()* class. It will turn our sentences into sequences of integers.
- We can then pad those sequences with zeros to make all the sequences of the same length. This is important as our applied Seq2Seq model requires that we convert both the input and the output sentences into integer sequences of fixed length.

1.4)- Random data

As data size is very large and it consists of words with many tokens therefore, training on such large data will halt my processor. I took sample of 5000 out of whole data of 22191. I made it sure that my sample is random. I can use random.seed () or sample() for this. I made double check if data is aligned. There are some techniques to check correspondence such as maximization algorithm and IBM model1. I didn't go that wild and only did manual check. Data was aligned. Next, I went to implementation of models.

2-Applied Models

RNN:

Recurrent Neural Networks (or more precisely LSTM/GRU) have been found to be very effective in solving complex sequence related problems given a large amount of data. They have real time applications in speech recognition, Natural Language Processing (NLP) problems, time series forecasting, etc.

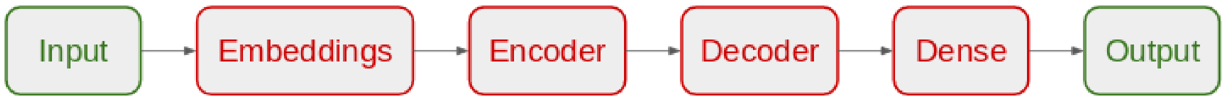
Sequence-to-Sequence:

Sequence to Sequence (often abbreviated to seq2seq) models are a special class of Recurrent Neural Network architectures typically used (but not restricted) to solve complex Language related problems, such as natural language translates, text summarization, speech recognition, chatbot, among others. Our aim is to translate given sentences from one language to another. Therefore, we used seq2seq model for our problem.

2a)-Simple Model:

This model is like basic language model using a typical seq2seq model. There are two major components i.e encode & decoder. Both these parts are essentially two different recurrent neural network (RNN) models combined into one giant network. Our simple model architecture consists of following;

- For the encoder, we will use an embedding layer and an LSTM layer
- For the decoder, we will use another LSTM layer followed by a dense layer



Our key equation for joint probability is :

[p(eng|ger) =p(eng)×p(ger|eng)]÷p(ger)

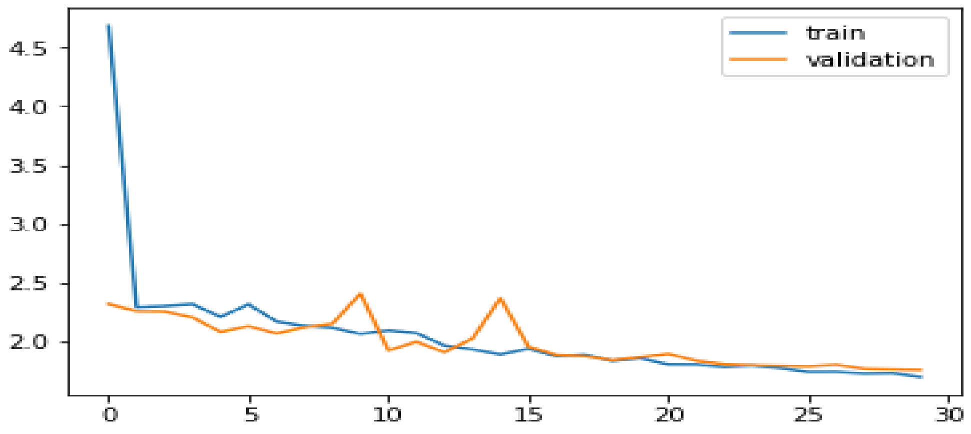
Where

p(eng) is language model

p(ger|eng) is translation model

p(ger) is evidence

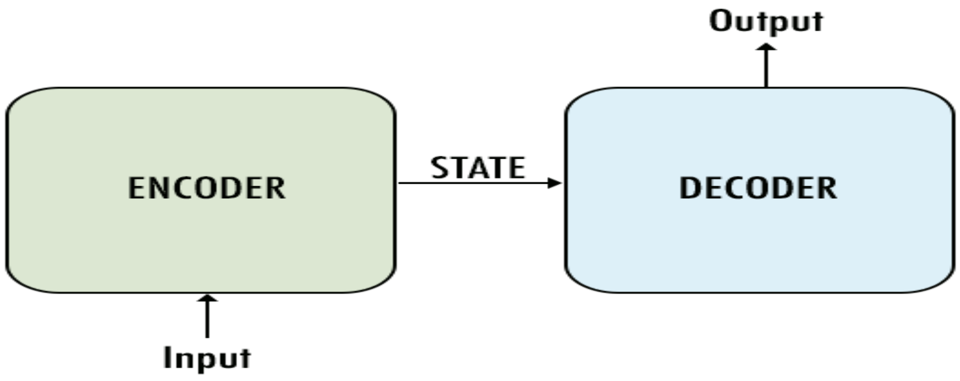
This model gave not very impressive results implying that we may need to develop a better model.



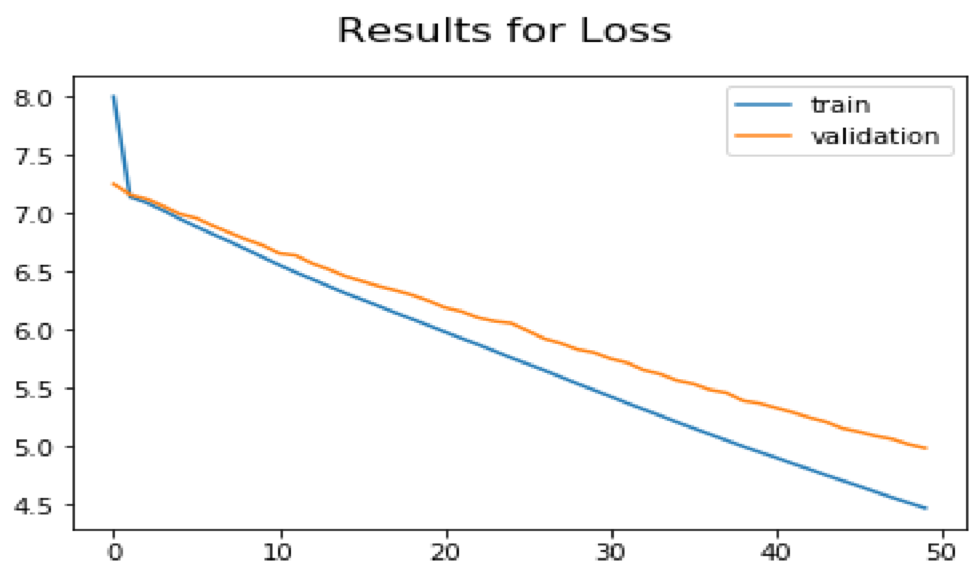
we have used '*sparse_categorical_crossentropy*'as the loss function. This is because the function allows us to use the target sequence as is, instead of the one-hot encoded format**.** One-hot encoding the target sequences using such a huge vocabulary might consume our system's entire memory.

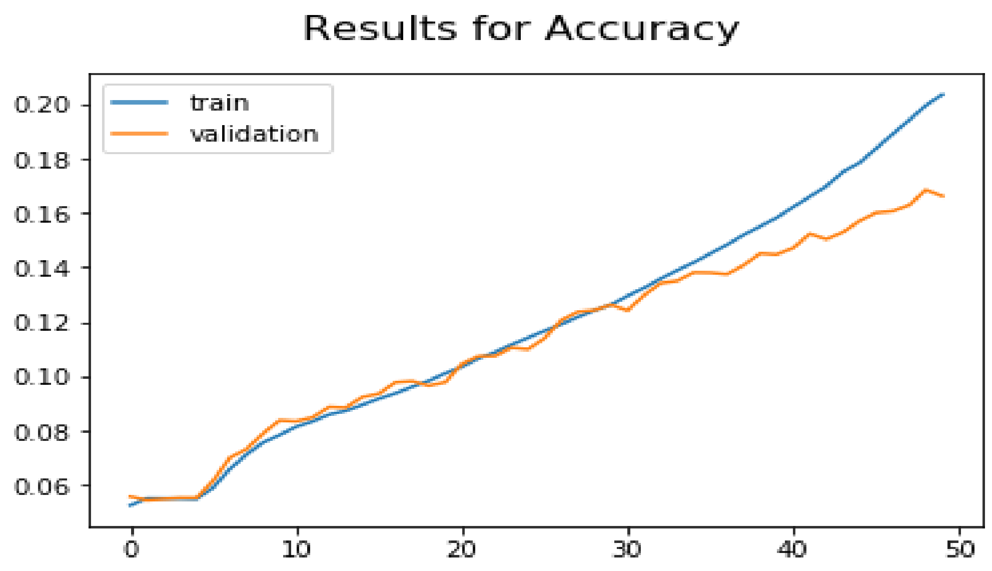
2b)-Encoder-decoder model

This model is classical encoder-decoder model as per literature. In this model, we will break the sentence by words as this scheme is more common in real world applications using greedy approach. We will select the next word using the highest probability in the softmax layer.



- In our model, both encoder and the decoder are LSTM models.
- Encoder reads the input sequence and summarizes the information in something called as the internal state vectors (in case of LSTM these are called as the hidden state and cell state vectors). We discard the outputs of the encoder and only preserve the internal states.
- Decoder is an LSTM whose initial states are initialized to the final states of the Encoder LSTM. Using these initial states, decoder starts generating the output sequence.
- The decoder behaves a bit differently during the training and inference procedure. During the training, we use a technique call teacher forcing which helps to train the decoder faster. During inference, the input to the decoder at each time step is the output from the previous time step.
- Intuitively, the encoder summarizes the input sequence into state vectors (sometimes also called as Thought vectors), which are then fed to the decoder which starts generating the output sequence given the Thought vectors. The decoder is just a language model conditioned on the initial states.
- I have changed evaluation criteria. This time loss function is categorical crossentropy and accuracy. Accuracy is mostly used with categorical data and as we have words in form of matrices so, this makes sense. Results are shown below



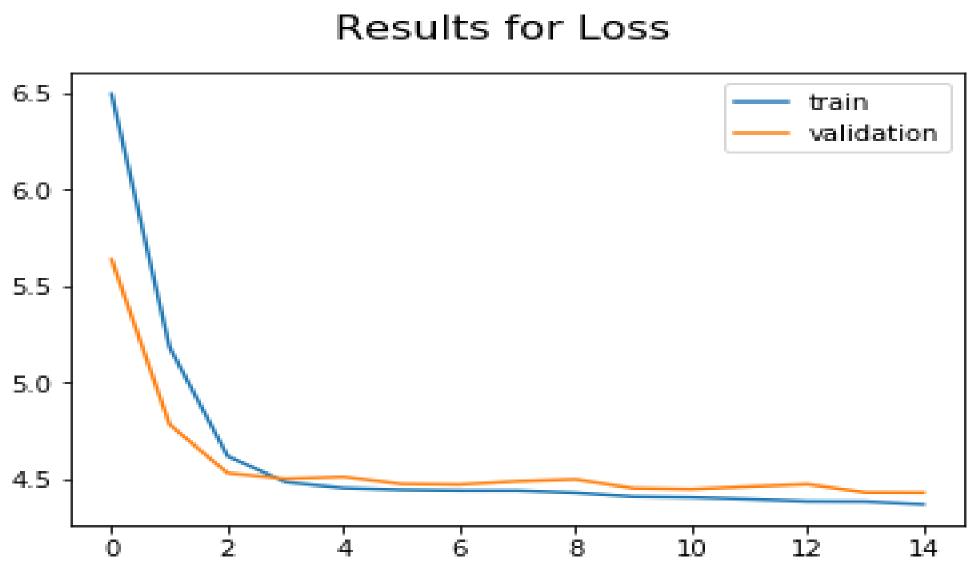


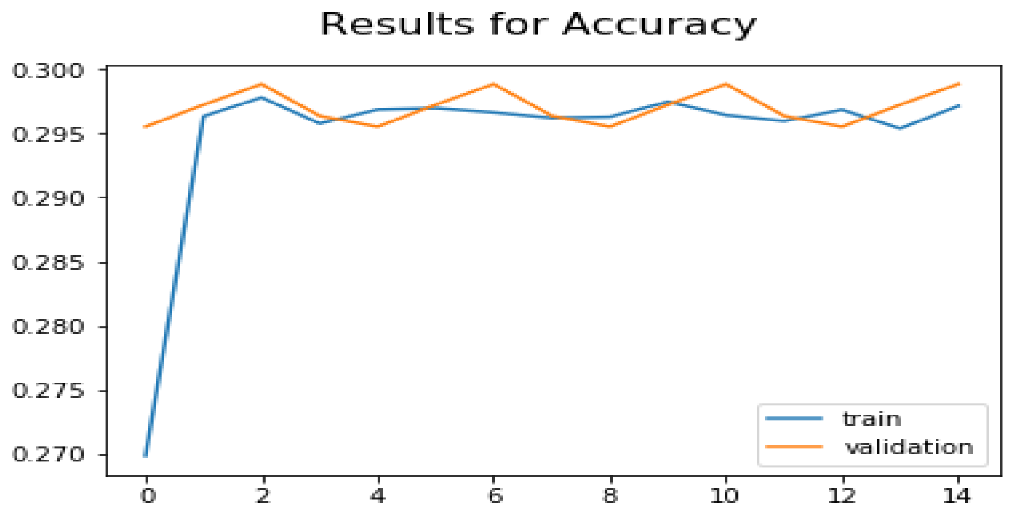
There is trace of over-fitting as train-set does show lower loss and better accuracy. I didn’t modify this model as these results are not be good anyway. I ll work on tuning with better model if I get one.

2c)- Applying out of vocabulary concept

The difficulty of the translation task is proportional to the size of the vocabularies, which in turn impacts model training time and the size of a dataset required to make the model viable. I did reduce the vocabulary of both the English and German text and marked all out of vocabulary (OOV) words with a special token(unk).

I did count the occurrence of each word in the dataset. For this I used a *Counter* object, and updated a count each time a new occurrence of each word is added. I then processed the created vocabulary and removed all words from the Counter that have an occurrence below a specific threshold. Some use 90 percentile rule. I have used threshold of 5 as maximum length of sentence was 73. This improved our results keeping all previous model architecture same.



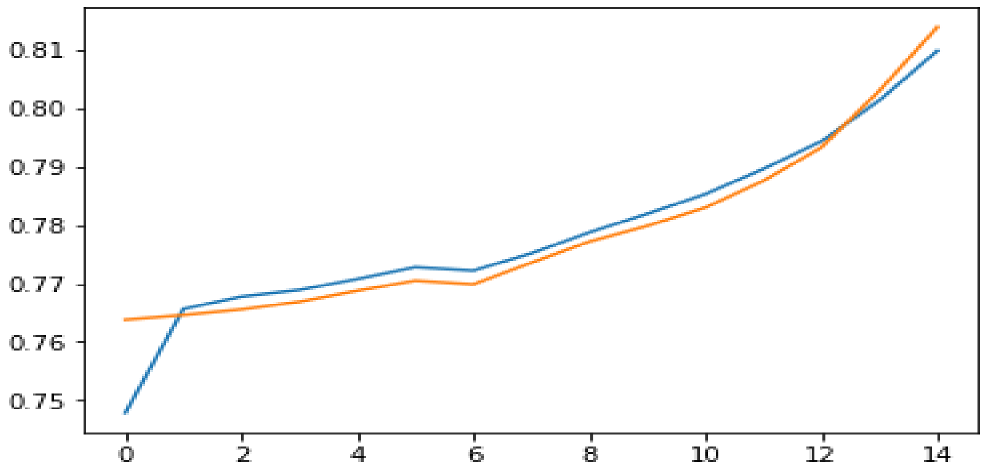


Although model is giving better results than previous attempt yet we see same pattern of straight-wise plot in validation case. There is no overfitting and results have improved.I still think accuracy of 30% is not good enough. One more try with matrices of evaluation as I started with sparse_categorical_crossentropy.

2d)-Applying sparse categorical accuracy as evaluation

Applying same encoder-decoder concept with different matrices of evaluation, I tried with sparse_categorical_crossentropy as loss function and sparse_categorical_accuracy as my accuracy. The only difference is that in categorical_accuracy I specified my target sequence i.e german seq. as one-hot encoded vector whereas in sparse categorical accuracy I only needed to use the target sequence as is, instead of the one-hot encoded format.

The results out of this model are better for sure. I get 81% of accuracy. It seems all those low accuracies were due to wrong choice of matrices of evaluation. Also model is not overfitting which is a good sign. Hence, I consider this as the most optimal model.



2e)- Try attention model

Though results are better yet there is one problem related to thought vector i.e hidden layer of LSTM. As memory is limited so, hidden layer 'thought vector' is only a few hundred floating points long. The more effort is put into forcing given vector in fixed dimensionality, the more lossy given neural net becomes. One solution is to increase size of hidden layer as LSTM can do use memory gates and hence, can store larger size of hidden layer. The problem is that this solution will increase training time exponentially. For this reason, attention model is considered in NMT cases.

It works simply as previous outputs of LSTM cells are stored and then are ranked in terms of relevancy. The best one is picked out of all and that is known as attention process. Normally, a bidirectional RNN uses context of both past and future words to create an accurate encoder out put vector.

In the end, I could not apply full model. I have provided pseudo code of attention model architecture.

3)-Custom in-house solution vs built-in developed company solution

There are couple of clear reasons coming out of given solution that might make more value for an international company to choose my in-house solution. They are related to cost, expertise and flexibility.

a) Being part of an ecosystem where main company might play role of focal firm whereas such smaller solution provider may act as niche. Niche are always those that are more specialized in their own technology. So, they have more time to concentrate on problem than to deal with market hierarchy and internal bureaucracy.
b) As solution provider, tt is more customized. A built-in solution is made for many customers and it is not easy to make all of them happy.
c) There are multiple models and approaches being tested to provide perspective.
d) Model is selected based on scalability issue. So, if new bigger data is added then it will accommodate that challenge without any difficulty.
e) More flexible and open to discussion as per customer demand.
f) Intellectual property may be more likely to remain confidential.

4)-Further improvements

- Get much more data. Top quality translators are trained on millions of sentence pairs.
- Build more complex models like Attention. I tried to write kind of pseudo code.
- Use dropout and other forms of regularization techniques to mitigate over-fitting. In my case, I didn't face overfitting this with my most optimal mode and I cleaned, reduced vocab of my data.
- Perform Hyper-parameter tuning. Play with learning rate, batch size, epochs, etc. Try using bidirectional Encoder LSTM. Try using multi-layered LSTMs.
- Try using *beam search instead of a greedy approach.

- Try BLEU score to evaluate your model.

The list is never ending and goes on. I could not try all. But, I did my best to come up with an optimal model which is deployable.

Notes:

*Beam search= an approach in which we consider multiple words for a single input word and creates beam and thus creates multiple sentences while finally choosing the sentence which has the highest overall probability.

References

- RNN: http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- LSTM: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- GRU: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be
- GRU vs LSTM : https://datascience.stackexchange.com/questions/14581/when-to-use-gru-over-lstm
- Word Embedding: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
- Out-of-Vocab: https://medium.com/@shabeelkandi/handling-out-of-vocabulary-words-in-natural-language-processing-based-on-context-4bbba16214d5
- Seq2seq:https://medium.com/analytics-vidhya/a-must-read-nlp-tutorial-on-neural-machine-translation-the-technique-powering-google-translate-c5c8d97d7587
- Teacher forcing: https://machinelearningmastery.com/teacher-forcing-for-recurrent-neural-networks/
- word-level seq2seq: https://towardsdatascience.com/word-level-english-to-marathi-neural-machine-translation-using-seq2seq-encoder-decoder-lstm-model-1a913f2dc4a7
- Eng-Fr: https://medium.com/@dev.elect.iitd/neural-machine-translation-using-word-level-seq2seq-model-47538cba8cd7
- Attention: https://medium.com/datalogue/attention-in-keras-1892773a4f22
- Attention-model: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).
- Attention model wrapper: https://github.com/neonbjb/ml-notebooks/blob/master/keras-seq2seq-with-attention/keras_translate_notebook.ipynb