Assignment 2: LSTM Based Language Modeling

Suraj Panwar
MTech DESE
SR: 14644
surajpanwar@iisc.ac.in

1 Introduction

This document provides the methods used for creation of a language model for implementation of a automatic sentence generation system. The results obtained in the process are presented here and the inferences observed from them are also mentioned in the report. The language models implemented here are LSTM based character and word token language models.

The datasets used here are D1 (Gutenberg Dataset) consisting of multiple documents from various different authors. The dataset is separated into adequate training, development and testing subsets, and the performance of the model is calculated on the training set using perplexity as the parameter to compare the performance of the models created.

The baseline model used here is the model constructed in the first assignment and is a Kneser Ney Based Trigram language model.

2 Separation of Datasets

The corpus used in this assignment is S1 (Gutenberg Corpus) the separation of dataset is done as follows:

2.1 Gutenberg Corpus

Gutenberg Corpus contains works of many renowned authors as separate files in the corpus. The division for this corpus has been done by separating the files in the ratio of 60:20:20 for training, development and test set and then appending the documents in each category to create a final training, development and test set.

The final sets thus are a combination of various documents randomly picked from the corpus and then merged into their respective sets.

Similar dictionaries have also been made for development and test set for both the datasets following the same syntax shown above.

2.2 Text Preprocessing

The data received from the Corpus is further preprocessed using NLTK library where the word tokens are created and fed to gensim model for WORD2VEC representation of 300 vector length. Gensim uses its own preprocessing techniques which allow removes all numerical as well as non alphabetical content (gensim.simple_preprocess).

During the sentence generation process the data is further processed by removing the numerical as well as the non alphabetical characters which do not find much use in the language use and are only used in some special cases, i.e, ”$”, ”ˆ”, ”&”, ”#”, etc. This has been done to increase the naturalness of the generated sentences.

3 Language Model

The language models constructed for this task are LSTM based word and character level model with a baseline model of trigram based model with interpolated Kneser Ney smoothing for unknown words.

The two models differ in the nature of inputs given to the RNN’s. The word based model takes a input of word tokens while the character based model takes a input of split of words as a continuous stream which is then fed into the RNN.

The data input in case of the word based LSTM model is a 20 word sentence that is sent in batches of 30. Thus, the input vector is of (30 x 20). The words are then converted in embeddings using the tensor flow function of tf.nn.embedding_lookup() this creates the word embeddings of words of length 200, the data thus fed into the LSTM model is of (30 x 20 x 200) the LSTM then returns the output words embeddings of the same shape

4 Training

The word level model is trained for 50 epochs on the complete training data which consists of 33369 words vocabulary, the final perplexity obtained is 75.49 on the training set and 102.47 on the test set as already mentioned above the data is input in batches of 30 with sequence length of 20 word tokens at a time, the plot of the perplexity with epochs is also shown.

The character level model is trained for 20 epochs also on the complete data which has a vocabulary of 57 unique characters after preprocessing, the final perplexity obtained was 6.97 on the training set and 10.87 on the test data set, the data was fed as input of 50 batches with a sequence length of 20 at a time, the plot of perplexity with epochs is also shown.

Based on the model performances a brief discussion is presented here comparing the results

5 Task 1, 2: Perplexity Computation

The perplexity was calculated on the various models on both test and training set and the results have been tabulated.

A brief discussion between the performance of the various models have been presented as follows.

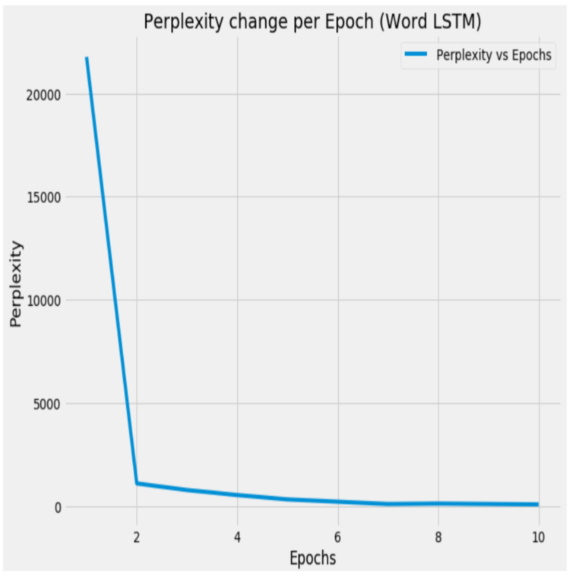


Figure 1: Plot of Perplexity with Epochs for Word level model

5.1 Comparison between Kenser Ney model and Word level LSTM model

From the results on perplexity we can infer that given that is was computed on same training

Model	Train Perpl.	Test Perpl.
Kneser Ney	452.71	682.91
Char Model	6.97	10.87
Word Model	75.49	102.47

Table 1: Perplexity for various Models.

and test dataset, the performance of word level LSTM based model is far superior compared to the Kneser Ney Trigram model.

The improvement can be attributed towards the fact that the Trigram model although good can only retain sequence order towards the last three word tokens, further increase in the memory causes a exponential rise in memory consumption and becomes not feasible after a threshold, whereas the LSTM based models can theoretically can retain context for a very long sequence and thus outperform the count based language models when it comes to long distance context retention.

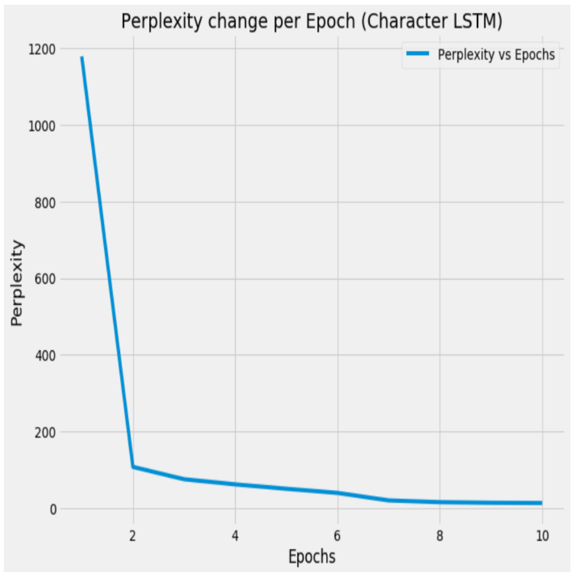


Figure 2: Plot of Perplexity with Epochs for Character level model

5.2 Comparison between Kenser Ney model and Character level LSTM model

The comparison between the Kneser Ney and the Character level LSTM model cannot be made as the vocabularies of both the models are different.

Where the Character level model has a vocabulary of 57 characters the vocabulary of the Kneser Ney model is well beyonds 30000 unique words, and hence comparison between these models cannot be made.

5.3 Comparison between Character level LSTM model and Word level LSTM model

Similar to above the Character level and the Word level model cannot be compared with each other due to different vocabularies, because even though both the models work on the same dataset the interpretation of the datasets of the two models is completely different as on one hand word level model seeks to find the next word, the character level model predicts the next character.

This difference in interpretation and different vocabularies results in us being unable to compare the performances of the models.

6 Task 3: Sentence Generation

The sentence generation model is based on character level based LSTM model which uses trained weights on the training set loaded in a file for direct use. The model output is a continuous string of output characters that are appended to the seed string and fed to the model again to generate the next character till a desired length sequence is obtained.

The seed character which initiates the string can be chosen manually or as in this case can be randomly picked from the dictionary. As already mentioned the program will build the string on the basis of the seed string provided to the model.

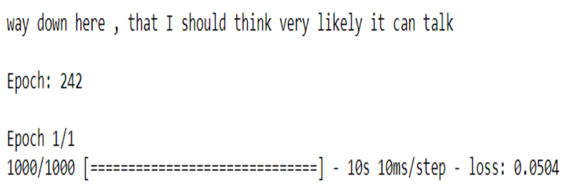
6.1 Limitations

The models are computationally exhausting and thus even though they can be endured during perplexity calculations the model sentence generation computation time becomes non feasible after a certain threshold and thus character generation model has to be limited to a subset of vocabulary due to otherwise facing a exceptionally high computation time.

The character generation model has only been trained on for 1 million train characters from a total of around 9 million total characters, this causes a bottleneck for the character generation model here.

6.2 Results

From the above discussion we can see the improved performance of the LSTM based models compared to the conventional count based model as the N-gram language models. This can be validated with the substantial decrease in the perplexity on both training and test set as well as a noticeable increase in the naturalness of the generated sentence. Some of the sample generated sentences have been provided here:

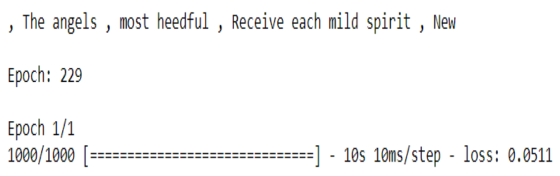


```
way down here , that I should think very likely it can talk

Epoch: 242

Epoch 1/1
1000/1000 [==============================] - 10s 10ms/step - loss: 0.0504
```

Figure 3: Example 1

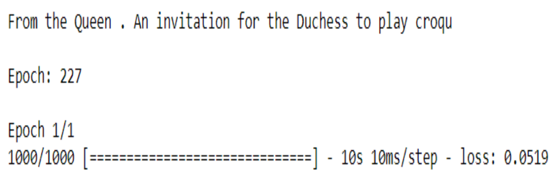


```
, The angels , most heedful , Receive each mild spirit , New

Epoch: 229

Epoch 1/1
1000/1000 [==============================] - 10s 10ms/step - loss: 0.0511
```

Figure 4: Example 2



```
From the Queen . An invitation for the Duchess to play croqu

Epoch: 227

Epoch 1/1
1000/1000 [==============================] - 10s 10ms/step - loss: 0.0519
```

Figure 5: Example 3

The accuracy and naturalness of the sentences can be increased further if the sentence generation is trained on a larger portion of the train corpus, However, that results in a multi fold increase in the computation time and hence has been avoided here.

7 Git hub Resource

The code for both Perplexity calculator and the Sentence generator has been uploaded on git hub with a read me file containing the instructions.

The files have been uploaded at: https://github.com/Suraj-Panwar/NLU_Assignment_2.git