Lab Report -Exercise 2 Building Word Embeddings

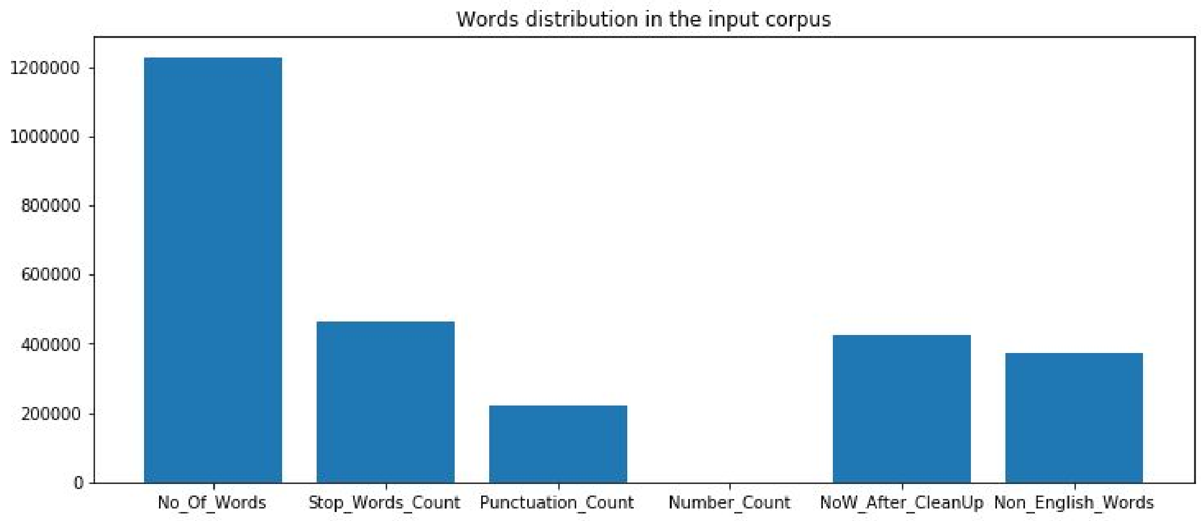
October 30, 2019

Data Set

Data set consists of a single text file “Shakespeare.txt” and the learning objective is to produce word embeddings for the work of Shakespeare.

Data Pre-processing

a) Data Set consists of 1225835 words, out of which there are 511 digits, 463139 stop words, 224190 Punctuations, and 374776 Non_english_word_Count. Refer to the following chart to get a graphical representation of the corpus. Since number of numeric words (511) is very small compared to other word frequencies, corresponding bar height is almost zero.



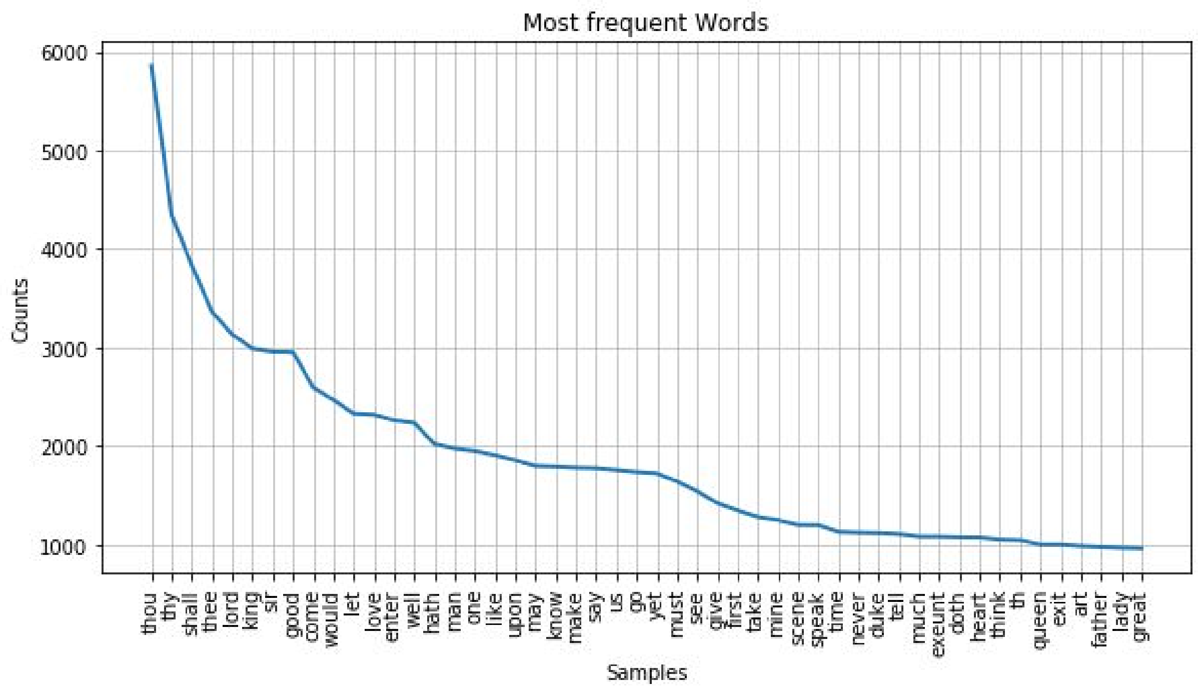
b) Convert all the words in the corpus to lower case, as classification of words into lower and upper case is not relevant to the current task.

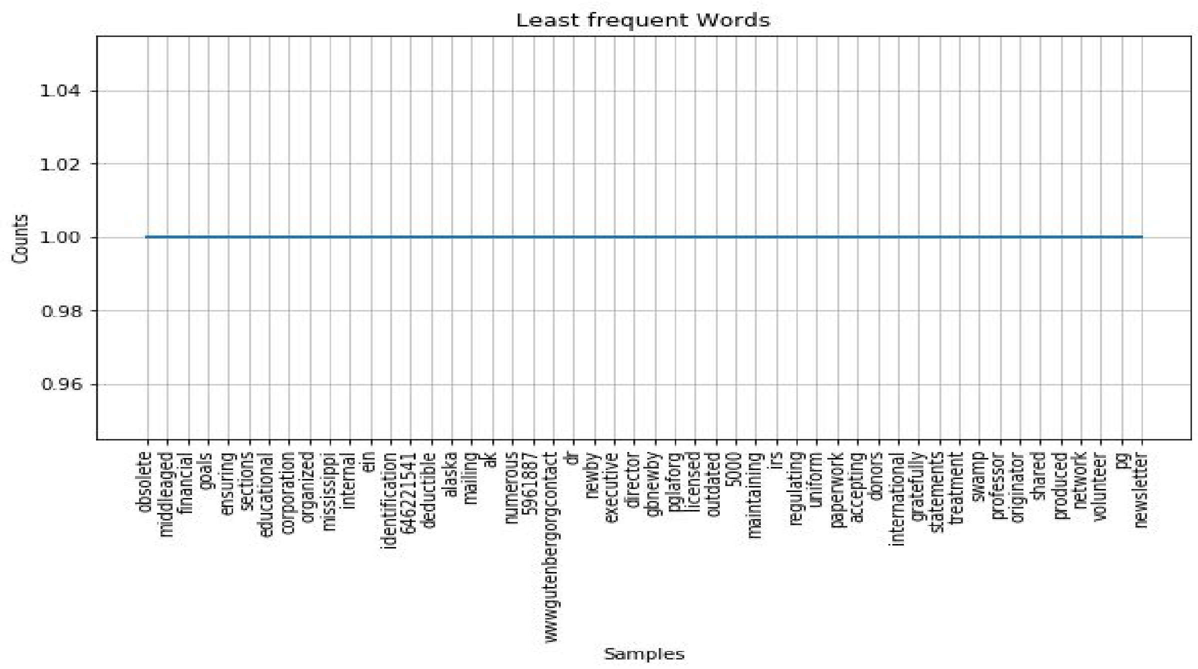
c) A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that we would not want taking up space in our database, or taking up valuable processing time. We use nltk stop word list to classify the words under stop words and remove them from our input corpus.

d) Similarly, punctuations and numeric digits have been removed from the corpus as these words don’t provide any meaningful contribution to the task of learning context.

e) We looked up the list of non-English words in our corpus by comparing the input corpus with the nltk word corpus. For example, name of a person can be classified as a non-English word (in the sense that name of a person doesn’t have any meaning). However, we decided to keep these words in our learning corpus as these words are an important part of context.

f) Visualize the most common and rare words in the corpus by plotting 50 most common and least common words. These plots also helped us to find the frequent and rare words to be used for testing purpose.

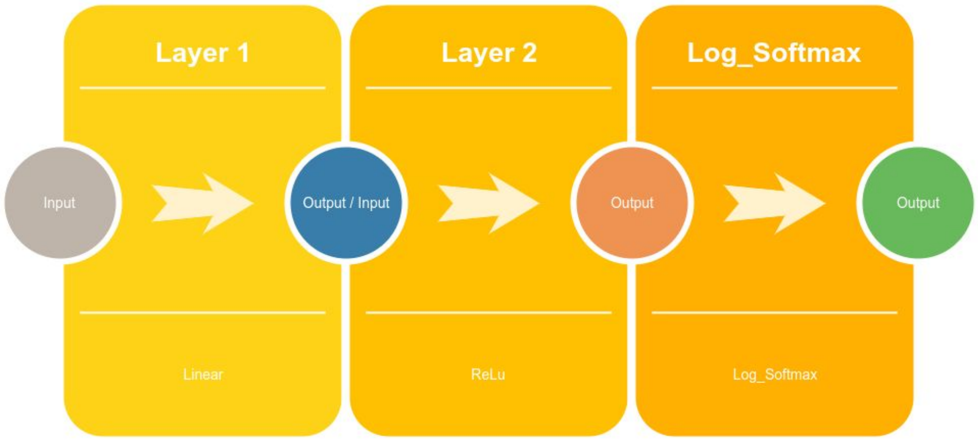




g) As the corpus size is huge, we decided to drop all the rare words (words which occur less than 5 times in corpus), as we will not have enough context samples to learn any meaning full information about these words.

Part 2: Test your embeddings

Figure below shows the CBOW model with a multi-layer context wording. We are running the model for two context sizes, 2 and 5, called henceforth CBOW2 and CBOW5 respectively. For context size 2, input to the model will be 4 BOW vectors each with dimension of v*1(v is the vocab size), where as for context size 5, input will be 10 BOW vectors.



As ReLu is used as the activation function for the hidden layer this introduces the non-linearity of the model. Thus, the results are non deterministic in nature.

CBOW2 Model Performance:

We ran 50 epochs for the CBOW2 model. Following is the data from google colab

```
for epoch 0--- 595.8126385211945 seconds ---
0/50 loss 8.12
5/50 loss 7.37
10/50 loss 7.22
15/50 loss 7.08
20/50 loss 6.97
25/50 loss 6.88
30/50 loss 6.80
5/50 loss 6.74
40/50 loss 6.69
45/50 loss 6.64
45/50 loss 6.60
```

--- 30190.444085597992 seconds --- # Total time taken for training.
Loss after running 50 epochs is Antilog(6.60) = 3981072

Similarity Between 3 Frequent Nouns, Verbs and Adjectives

Input word	First Neighbour	Second Neighbour	Third Neighbour	Fourth Neighbour	Fifth Neighbour
king	'unburden' , 0.5000	'wilfulness' ,0.49375	'rememberd', 0.4799	'timandra' , 0.4731	'madamd', 0.4698

come	'gilded', 0.5711	'prize', 0.5152	'rowlands', 0.5137	'wards', 0.5110	'baboon', 0.5030
great	'dart', 0.5706	'much', 0.5616	'substitutes', 0.5245	'early', 0.5145	'properties', 0.4976

Similarity Between 3 Rare Nouns, Verbs and Adjectives

Input word	First Neighbour	Second Neighbour	Third Neighbour	Fourth Neighbour	Fifth Neighbour
Rascals	subject', 0.4570	'prize', 0.5152	'unto', 0.4342	'despair', 0.4315	'strikes', 0.4208
frowns	'yare', 0.4838	'made', 0.4564	'trusty', 0.4436	'free', 0.4397	'terror', 0.4306
beauteous	'rate', 0.4365	'george', 0.4127	'sword', 0.4099	'longer', 0.4040	'manners', 0.3992

CBOW5 Model Performance:

We ran 50 epochs for the CBOW5 model. Following is the data from google colab

```
0/50 loss 8.08

5/50 loss 7.32

10/50 loss 7.12

15/50 loss 6.98

20/50 loss 6.86

25/50 loss 6.76

30/50 loss 6.69

35/50 loss 6.63

40/50 loss 6.58

45/50 loss 6.53

50/50 loss 6.49
```

--- 32037.6863591671 seconds --- # Total time taken for training.
Loss after running 50 epochs is Antilog(6.49) = 3090300

Similarity Between 3 Frequent Nouns, Verbs and Adjectives

Input word	First Neighbour	Second Neighbour	Third Neighbour	Fourth Neighbour	Fifth Neighbour

king	'married', 0.5659	'unclaspd', 0.5533	'talk', 0.5254	'orpheus', 0.5064	'afoot', 0.5042
come	'utterly', 0.5345	'publius', 0.5238	'gayness', 0.5065	'variation', 0.4885	'fortunate', 0.4825
great	'vie', 0.5669	'scorned', 0.5364	'initiate', 0.5143	'father', 0.4960	'begets', 0.4947

Similarity Between 3 Rare Nouns, Verbs and Adjectives

Input word	First Neighbour	Second Neighbour	Third Neighbour	Fourth Neighbour	Fifth Neighbour
rascals	‘bear', 0.4698	'night', 0.4477	'pleasing', 0.4445	‘ravish', 0.4327	'mere', 0.4318
frowns	repose', 0.4251	'hair', 0.4237	'wand', 0.4130	thees', 0.4127	whither', 0.4028
beauteous	'nest', 0.4623	'tennis', 0.4512	'iniquity', 0.4498	'go', 0.4413	'batterd', 0.4329

Model Performance

	CBOW2	CBOW5
Time Per Epoch	~10 mins	~10.7 mins
Total Run time	~ 8.4 hrs	~ 8.9 hrs
Training Loss	6.60	6.49

Performance of both models in finding the closed neighbour is not really great as loss is too high. Loss at the end of 50 epoch is 6.6 and 6.49 respectively for CBOW2 and CBOW5. With such high values of loss, we cannot expect the model to predict neighbours correctly and this is also evident from the test results on sample words.

Optimizer

The above results were achieved using SGD as an optimizer. SGD was used as our run with Adam did result in a worse loss and probably requires more epochs to improve. However Adam resulted in better results on a smaller corpus which was used to test the functionality of the code.

	SGD	Adam
Corpus size after cleaning	1280	1280
Total Run time	~ 8.5 min	~ 12 min
Training Loss	2.64	0.04

Performance improvements with GPU enable in Google Collab

We enabled GPU feature in our code and see an improvement of almost 50% in computation time. Code changes to enable are

```
# check for the availability of GPU with CUDA support
cuda_available = torch.cuda.is_available()

# function to send tensors to gpu if cuda is enabled
def tensor_to_cuda(tensor):
    if cuda_available:
        device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
        tensor = tensor.to(device)
    return tensor

# enables cuda on the given model
def model_to_cuda(model):
    if cuda_available:
        device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
        model = model.cuda()
    return model
```

Difference between CBOW2 and CBOW5

CBOW5 with larger window size tend to capture more topic/domain information, what other words (of any type) are used in related discussions. CBOW2 with smaller window size tend to capture more about word itself: what other words are functionally similar. Their own extension, the dependency-based embeddings, seems best at finding most-similar words, synonyms or obvious-alternatives that could drop-in as replacements
of the origin word.

Larger windows are known to induce embeddings that are more ‘topical’ or ‘associative’, improving their performance on analogy test sets, while smaller windows induce more ‘functional ‘and ‘synonymic’ models, leading to better performance on similarity test sets(From Goldberg)

Important remarks regarding checkpoints

We added the functionality to use checkpoints during training, this allows you to stop and resume the training of the model. However if you change the dimensions, context size, optimizer or any other parts of the code then the checkpoints won’t be compatible as the tensor dimensions won’t match! If you wish to do so please remove or rename the checkpoint file.