COMP6714 Project2

Report

1: Function execution:

1: Preprocessing the vocabulary:

1.1: getridsim(): this is the function that we need to get the rid of all the Punctuation mark in all the text including \n and \t. In the txt file. I use re module to substitute all the a punctuation mark to ’‘. That is because the all the punctuation mark are really difficult for spacy module to modify and really need to be deleted.

1.2: dealmoney(): This is the function that really deal with money. Form the spacy entities extraction, i fount that all the entities which could be represent as MONEY, ORG, GRE etc. So we substitute all the entities to their entities names, but some money flag still exists. This function is to delete all the money flag

1.3: Process\_data(): This function really preprocess all the data. After reading all the data, I firstly remove all the punctuation mark in all the text. Secondly, I find all the entities in the text and change them to their entities names (Apple -> ORG). Thirdly, I check all the tokens in each data set and use spacy to identify their pos. In this project, all we need is the adjective words, other words such as NOUN and VERB are not need to be count. For the next iteration by Tensorflow. It all can be changed by their pos names so that we substitute all the non-adjective words to their pos names. We remain all the adjectives and add them to dataset.

Here is a trick, some words such as chief light, it may have different pos name due to different environment. Due to this reason, we build a global adjective set, if we have tokens name as ADJ, we add them into the global adjective set. When embedding is finished, we store all the adjectives we found into adjective\_embeddings.txt. Another method I tried is that when write outputs into embedding.txt , i use spacy to check the token pos and add all the adjective words into the file. This method may lose those words like light and chief so i ignored them

Another trick is that although we substitute all the irrelevant words, some words still exists, (‘a’,’the’ for etc,). These words are called stop words. I do not know whether to delete or not. So I keep them in the data file

After all the execution above, we would get the data\_file we want which is called PWD.

2:Training:

The training method is a almost copy from the word2vector demo.it contain all the steps：

1:Build data set: as we get the data, we could build the dictionary and replace rare words wich UNK token

2: Batch: as we get the data set. We can generating training batch for the skip-n gram model. Batch size means the size of mini batch for skip-gram model. Skip window is the window size each words has its context both fornt and behind for training.num\_samples mean how many time to reuse an input to generate a label. It could no longer than 2\* skip window size and negative is the sample size for negative samples . In this project. I choose 1700 as the negative number. The function we use is the AdamOptimizer as the optimizer function and use simple soft max loss as the loss function.

3:training: we initialize the object using the tf.session() and call respective node via session.run() or via and train as word2vector 5 steps.

4:topk: The calculation method we use is the genism model. Totally contain two steps

1: read the embedding files and extract them by module.

2: use most similar to get the list

gensim.models.KeyedVectors.load\_word2vec\_format(model\_file, binary=False)

model.most\_similar(positive=[sample words], topn = top\_k)

Results collect:

Final select vocabulary size = 6000 learning rate =0.002 top\_k = 100 ,skip window =2 num samples =4 Negative =1700

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vocabulary | Learning rate | Top k | Skip window | Num sample s | Negative | Rate |
| 6000 | 0.02 | 40 | 2 | 4 | 1700 | 2.13 |
| 6000 | 0.002 | 40 | 2 | 4 | 1700 | 6.53 |
| 5000 | 0.002 | 40 | 2 | 4 | 1700 | 5.64 |
| 6000 | 0.002 | 100 | 2 | 4 | 1700 | 4.8 |
| 10000 | 0.002 | 100 | 2 | 4 | 1700 | 4.55 |
| 6000 | 0.002 | 100 | 1 | 2 | 1700 | 3.36 |
| 15000 | 0.002 | 100 | 2 | 4 | 1700 | 4.325 |

Here are some screen shot of the result and docker(with iteration 1000):









