HW3 - Text Representation and Retrieval

Deepak Magesh Srivatsav
2016030

Question 1

I implement word2vec using both skip-gram and continuous bag of words.
In skip-gram, given a particular ‘target’ word, we try to predict ‘context’ words. This can be done as follows. Consider the sentence - A quick brown fox jumps over a lazy dog. If we take a context size of 4, and if our target word is brown, then brown is mapped to (A, quick, fox, jumps). For the sake of training, I trained pairs ([brown, A], [brown, quick], etc.)

In continuous bag of words, I did the opposite. Given a set of ‘context’ words, the model tried to predict a ‘target’ word.

To preprocess the sentences given in the ‘abc’ dataset, I used nltk to remove all punctuations, stop words, and stemmed all the words to reduce the size of the vocabulary. I further converted all words to lower case.

The coding was done using python. I used PyTorch as a deep learning framework, nltk for language processing, and tensorboard for visualizations.

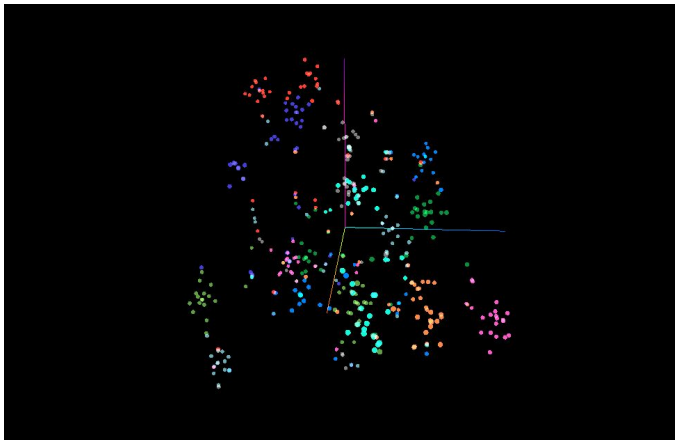
Some examples of the working of the algorithm -
Test word - widespread, Related words - ['widespread', 'shepparton', 'bottleneck', 'polio', 'scallop', 'nervi', 'anthrax', 'allel', 'deepli', 'haemophilia', 'lesion']

The models were trained for 100 epochs. The embedding size was 200. The context size was 6 (3 on each side of target word).

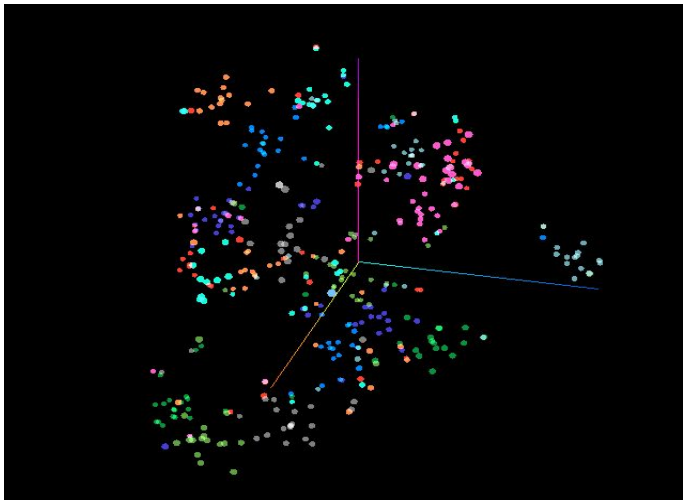
The tsne plots presented below are after 10, 20, 40, 60 and 100 epochs for both skip-gram and continuous bag of words. To generate the plots, I took 20 random words, and found the 20 most similar words to these random words, and plotted their embeddings. By doing so, we can see whether similar words have close enough vector representations. If viewing on tensorboard, set perplexity to 20 and learning rate 10. Run for ~4000 iterations without spherisizing it the plots.

Continuous Bag of words

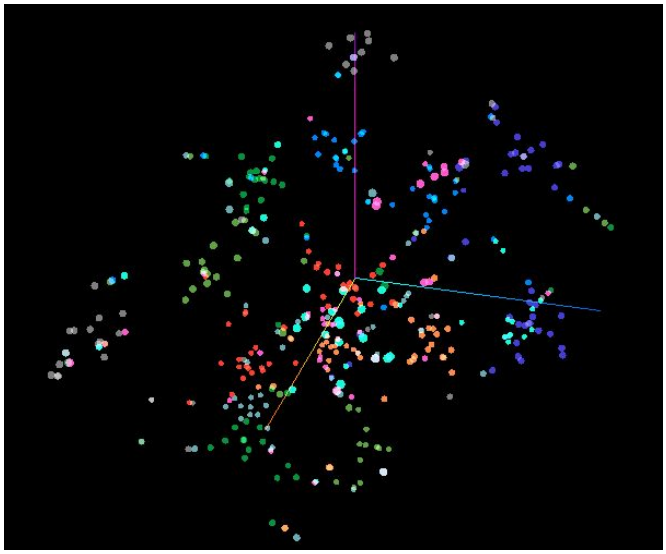
10 epochs



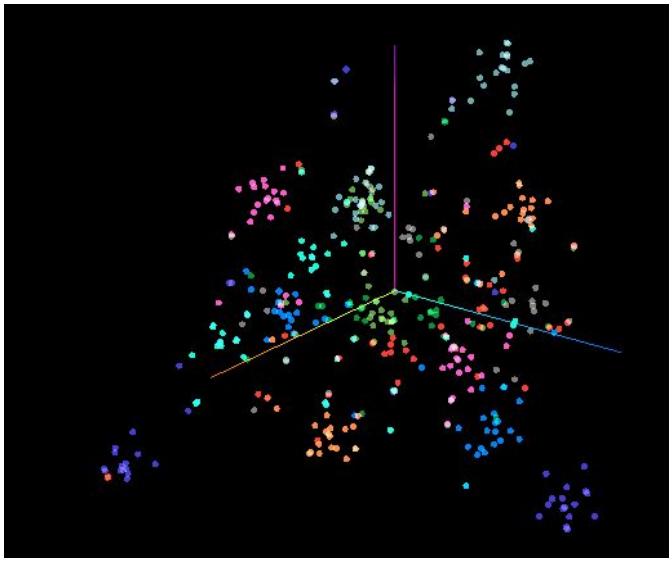
20 epochs



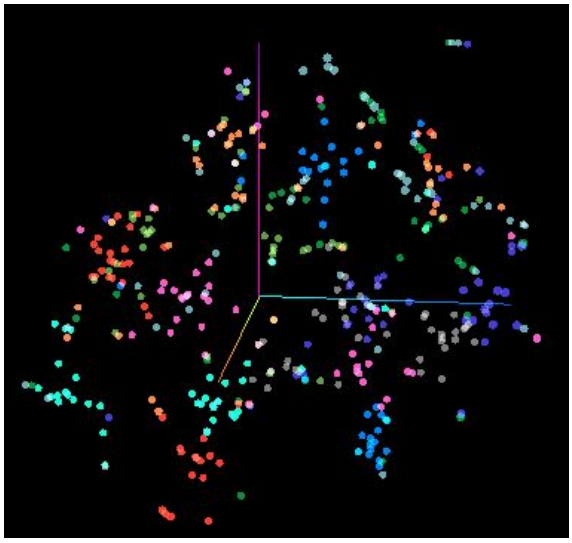
40 epochs



60 epochs

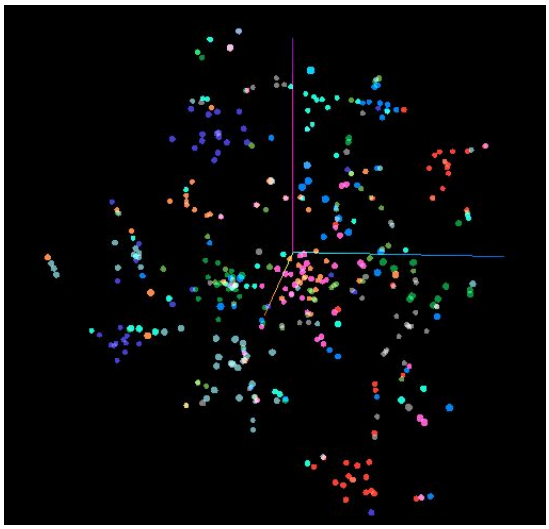


100 epochs

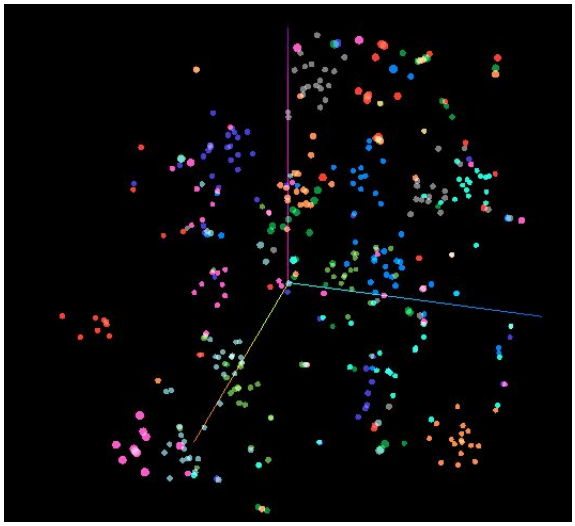


Skip Gram

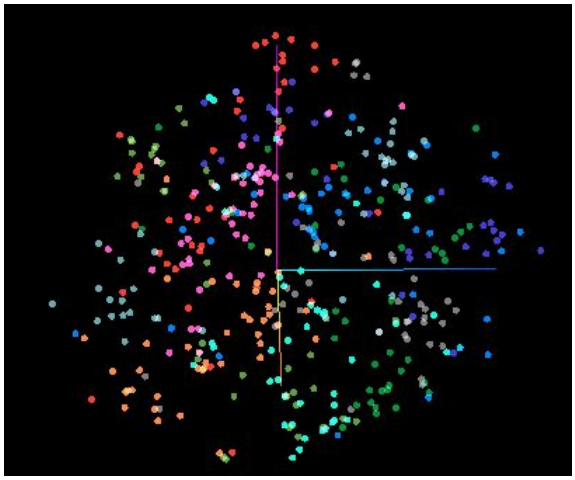
10 epochs



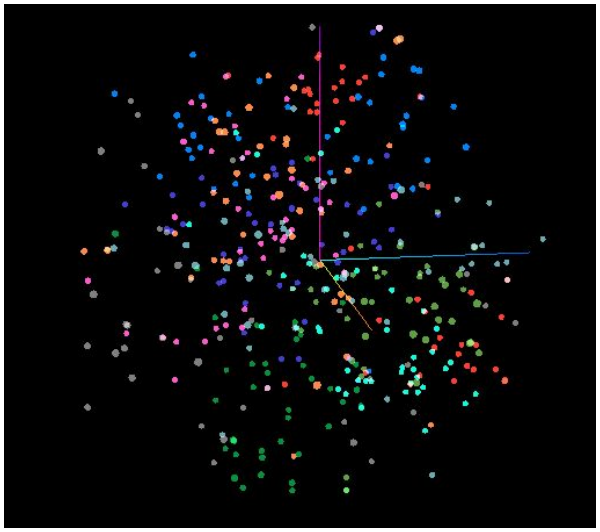
20 epochs



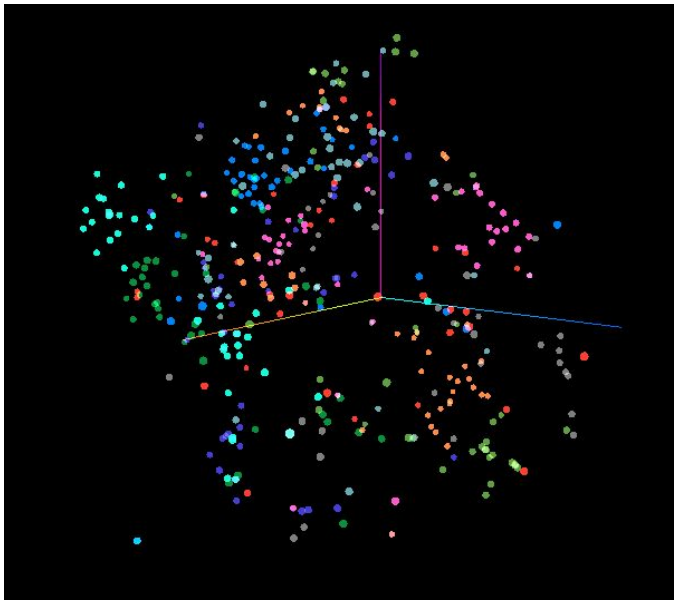
40 epochs



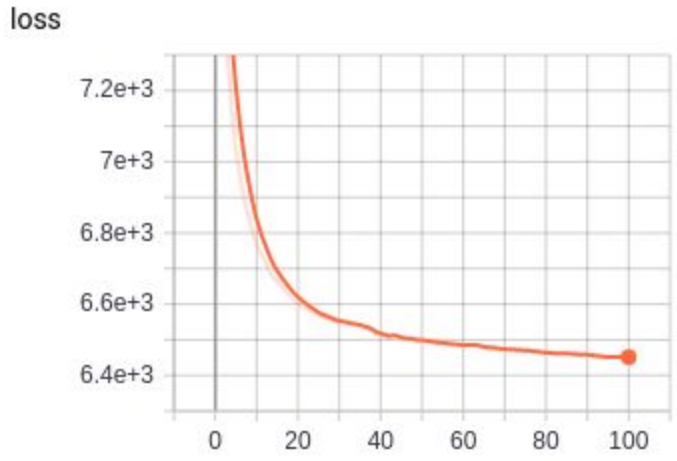
60 epochs



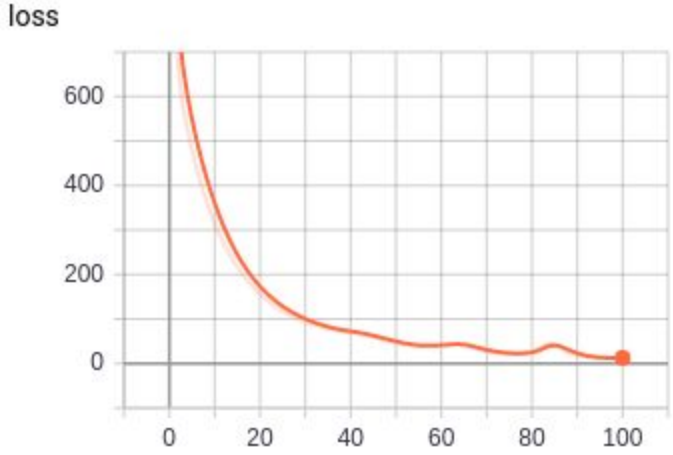
100 epochs



Skip-gram loss curve



Continuous bag of words loss curve



CBOW performed a lot better. From the tsne plots, there are multiple clusters visible, however, the clusters are better formed with cbow. The training time is also sufficiently less for cbow due to it having less training samples on account of the way the data is represented.

Question 2

I implement pseudo (blind) relevance feedback[1]. I achieved an mAP of 0.6349.
In this case, the weights played an important role. I realized the best results when my weights acted as normalizations.

For relevance feedback with query expansion[2], I achieved an mAP of 0.6227.
For query expansion, using the given query, I find the highest tfidf term values. Using this, I look through the documents for those that have high value for the same term. I then add the best documents to the query. This aims to add the notion of synonyms.

These results are inline with my expectations. They are both ~10% increases over the baseline.

[1] "Pseudo relevance feedback - Stanford NLP Group." https://nlp.stanford.edu/IR-book/html/htmledition/pseudo-relevance-feedback-1.html. Accessed 1 May. 2020.
[2] "Query expansion - Stanford NLP Group." https://nlp.stanford.edu/IR-book/html/htmledition/query-expansion-1.html. Accessed 1 May. 2020.

Instructions to run or test code

Question 1

Python 3 is required.
First install requirements - pip install -r requirements.txt.
python question1_1.py <cbow/skp>
Weights are stored in a directory called snapshots. Logs are stored in cbow_logs or skp_logs

To test on a few random words,
python question1_test.py
Note - all weights must be in a directory called “snapshots”. Or else, the path values must be modified in script.

To visualize with tensorboard, use tensorboard --logdir=<cbow_logs/skp_logs>.
TSNE plots may be viewed under the “projector” tab.

Question 2

Python 3 is required
python main.py