To start with, we have tried to implement our logic in line with the skeleton code provided with the problem statement. We divided the problem statement into modules, with each module covering important aspects of the skip-gram model.

Preprocessing:

First, we tried Spacy preprocessing but later for convenience, we used traditional preprocessing technique of removing punctuations, splitting on space, converting to lower case and removing one-word sentences.

text -> sentences -> clean the sentences -> tokenization

Encoding the word vectors:

In our initial trials, to improve the computational efficiency we tried to encode the words in a way that's different from one hot encoding - making a list of lists with each list in the main list starting with center word followed by context words. But moving ahead with the code we found it difficult for us to play around with those vectors when calculating gradient and updating weights. Therefore, we finally used one hot vector encoding to encode the words in the corpus.

Negative Sampling:

Initially we tried picking the samples just randomly from the vocab(omitting center word and context word). After reading a few papers on the negative sampling, we understood that negative sampling is widely done using a noise distribution. However, the unigram distribution is raised to the power of 3/4rd to combat the imbalance between common vs rare words. Hence we finally defined a function such that it generates the noise distribution and returns the words with their probability. The Sample function then uses these probabilities to pick 5 samples and returns them.

Train:

Train function is not altered from the skeleton code. We have written other parts of the codes such that they are in line with the train function provided in the skeleton code.

Neural Network Implementation detail:

This is the most important part of the entire code. We have extensively read various papers and We checked the math part of this project regarding to forward prop, loss calculation, and backward prop.

We initially started off our trials with implementing just the skipgram model without negative sampling to clearly understand the working of forward and backward propagation step by step, where in we defined different functions for forward and backward passes.

But when the negative sampling is included, the objective functions for context words and negative samples are different. Therefore after calculating the hidden layer, we eliminated the back propagation function from our skipgram model and separately performed the stochastic gradient descent and calculated the gradients for context words and negative words and updated the center word vector in input weight matrix and K+1 (negative + context) word vectors in output weight matrix.

We have also tried to use a moving average of gradient to replace the real gradient but the results were not very satisfactory, not a worthy trade off for the increased computation. Therefore, we finally went back to the simple SGD.

Similarity:

We used Cosine Similarity to calculate the similarity between two vector embeddings.

References:

Skip-gram with Negative sampling References:

1.word2vec Parameter Learning Explained

https://arxiv.org/pdf/1411.2738.pdf

2. Optimize Computational Efficiency of Skip-Gram with Negative Sampling

https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling

3. Demystifying Neural Network in Skip-Gram Language Modeling

https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

4. Distributed Representations of Words and Phrases and their Compositionality

https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

5.. How Negative Sampling work on word2vec?

https://medium.com/@makcedward/how-negative-sampling-work-on-word2vec-7bf8d545b116

6. Word2Vec Tutorial Part 2 - Negative Sampling

http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

7. Word2vec from Scratch with Python and NumPy

https://nathanrooy.github.io/posts/2018-03-22/word2vec-from-scratch-with-python-and-numpy/

8. Implement your own word2vec(skip-gram) model in Python

https://www.geeksforgeeks.org/implement-your-own-word2vecskip-gram-model-in-python/

9. An implementation guide to Word2Vec using NumPy and Google Sheets

https://towardsdatascience.com/an-implementation-guide-to-word2vec-using-numpy-and-google-sheets-13445eebd281