Singapore University of Technology and Design
50.040 Natural Language Processing, Summer 2022
Homework 1 (40 Marks)

Homework 1 will be graded by Ming Shan
Due 3rd June 2022, 5pm

Overview. Word embeddings are dense vectors that represent words, and capable of capturing semantic and syntactic similarity, relation with other words, etc. We have introduced two approaches in the class to learn word embeddings: **Count-based** and **Prediction-based**. Here we will explore both approaches. Note that we use "word embeddings" and "word vectors" interchangeably.

Before we start, you need to download the WikiText-2 dataset. Unzip the file and then put it under the "data" folder. The WikiText-2 dataset consists of multiple lines of long text. Please do not change the data unless you are requested to do so.

Requirements:

1. Python 3.5 or above
2. gensim
3. sklearn
4. numpy

1 Count-Based Word Embeddings

Co-Occurrence. A co-occurrence matrix counts how often things co-occur in some environment. Given some word w_i occurring in the document, we consider the *context window* surrounding w_i. Supposing our fixed window size is n, then this is the n preceding and n subsequent words in that document, i.e. words $w_{i-n} \dots w_{i-1}$ and $w_{i+1} \dots w_{i+n}$. We build a *co-occurrence matrix* M, which is a symmetric word-by-word matrix in which m_{ij} is the number of times w_j appears inside w_i's window.

Example: Co-Occurrence with Fixed Window of n=1:
Document 1: "learn and live"
Document 2: "learn not and know not"

*	and	know	learn	live	not
and	0	1	1	1	1
know	1	0	0	0	1
learn	1	0	0	0	1
live	1	0	0	0	0
not	1	1	1	0	0

Normalized Pointwise Mutual Information (NPMI). Pointwise mutual information (PMI) is one of the most important concepts in NLP. The pointwise mutual information between a target word w and a context word c is defined as:

$$\mathrm{PMI}(w, c) = \log_2 \frac{P(w, c)}{P(w)P(c)}$$

Given co-occurrence matrix $\mathbf{M} \in Z^{N \times N}$ of N words, m_{ij} is the element of i th row and j th column. The PMI matrix can be calculated as

$$\mathrm{PMI}_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

where

$$p_{ij} = \frac{m_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{N} m_{ij}} \quad p_{i*} = \frac{\sum_{j=1}^{N} m_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{N} m_{ij}} \quad p_{*j} = \frac{\sum_{i=1}^{N} m_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{N} m_{ij}}$$

In addition to PMI, pointwise mutual information can be normalized between [-1, +1]. The normalized mutual information between a target word w and a context word c is defined as:

$$\mathrm{NPMI}(w, c) = \frac{pmi(w, c)}{h(w, c)}$$

where $h(w, c)$ is a joint self-information, which can estimated as $-\log_2 P(w, c)$

Principal Components Analysis (PCA) and Truncated Singular Value Decomposition (Truncated SVD) The rows (or columns) of co-occurrence matrix or PPMI matrix can be utilized as word vectors, but the vectors will be large in general (linear in the number of distinct words in a corpus). Thus, our next step is to run dimensionality reduction. In particular, we will first run PCA (Principal Components Analysis) to reduce the dimension. In practice, it is challenging to apply PCA to large corpora because of the memory needed to perform PCA. However, if you only want the top k vector components for relatively small k — known as Truncated SVD — then there are reasonably scalable techniques to compute those iteratively.

Question 1.1 [code] (2 points). Implement the function *distinct_words* that reads in *corpus* and returns distinct words that appeared in the corpus and the number of distinct words.

Then, run the sanity check cell to check your implementation.

Question 1.2 [code] (8 points). Implement *compute_word_matrix* that reads in *corpus* and *window_size*, and returns a co-occurrence matrix, NPMI matrix and a word-to-index dictionary.

Then, run the sanity check cell to check your implementation.

Question 1.3 [code] (5 points). Implement *dimension_reduction* function below with python package
sklearn.decomposition.

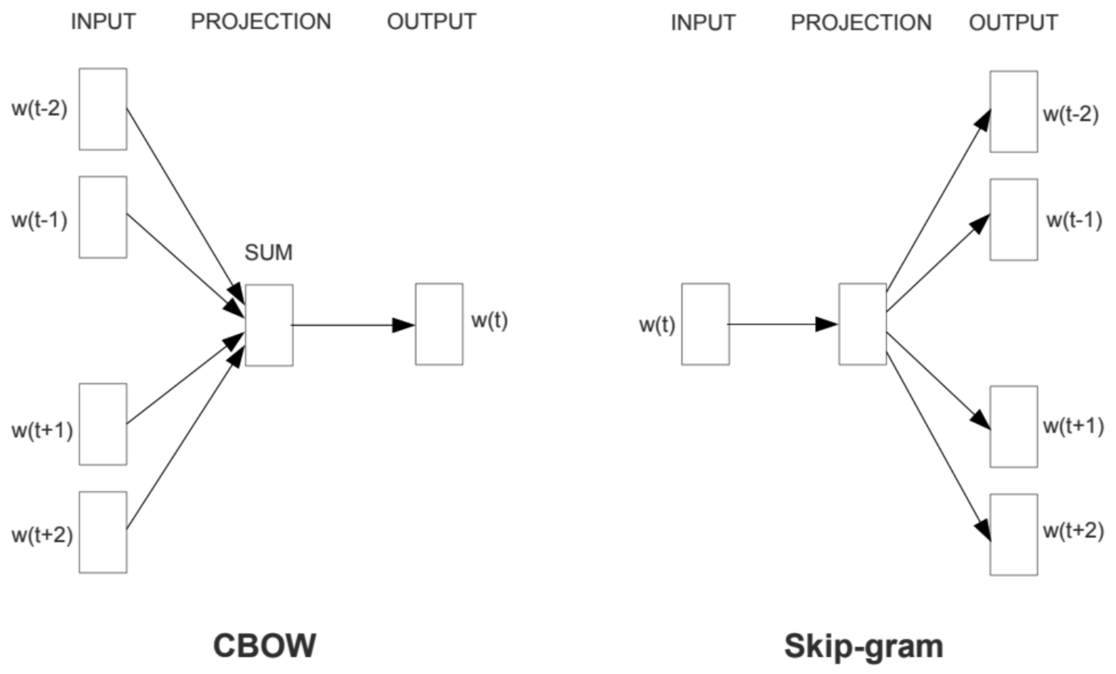
Then, run the sanity check cell to check your implementation.

Question 1.4 [code] (5 points). Implement *plot_embeddings* function to visualize the word embeddings on a 2-D plane.

Then, run the sanity check cell to check your implementation.

2 2. Prediction-Based Word Embeddings

Word2vec. Word2vec is a software package that contains two algorithms named CBOW and skip-gram (Mikolov 2013). In the CBOW architecture, the model predicts the current word from a window of surrounding context words. In the continuous skip-gram architecture, the model uses the current word to predict the surrounding window of context words. The architectures are shown as follows:



Question 2.1 [code] (3 points). Complete the code in the function *create_word_batch*, which can be used to divide a single sequence of words into batches of words.

For example, the word sequence [”I”, ”like”, ”NLP”, ”So”, ”does”, ”he”] can be divided into two batches, [”I”, ”like”, ”NLP”], [”So”, ”does”, ”he”], each with batch_size=3 words. It is more efficient to train word embedding on batches of word sequences rather than on a long single sequence.

Then, run the sanity check cell to check your implementation.

Question 2.2 [code] (3 points). Use *Word2Vec* function to build a word2vec model. Please use the parameters we have set for you.

It may take a few minutes to train the model.

Question 2.3 [code] (6 points). Implement *get_word2Ind* function below first. Then, run the sanity check cell to check your implementation. Use *get_word2Ind*, *dimension_reduction*, and *plot_embeddings* functions to visualize the word embeddings of the first 300 words in the vocabulary.

Question 2.4 [code] (4 points). Find the most similar words for the given words "man", "woman", "king". You need to use *model.wv.most_similar* function. Find out which word will it be for x in the pairs author : singer :: book : x? You need to use *model.wv.most_similar* function.

Question 2.5 [code+written] (4 points). It's important to be cognizant of the biases (gender, race, sexual orientation etc.) implicit in our word embeddings. Bias can be dangerous because it can reinforce stereotypes through applications that employ these models.

Use the *most_similar* function to find two cases where some bias is exhibited by the vectors. Please briefly explain the example of bias that you discover.

How to submit

1. Fill up your student ID and name the Jupyter Notebook.
2. Click the Save button at the top of Jupyter Notebook
3. Select Cell - All Output - Clear. This will clear all the outputs from all cells (but will keep the content of all cells)
4. Select Cell - Run All. This will run all the cells in order, and will take several minutes
5. Once you've rerun everything, select File - Download as - PDF via LaTeX
6. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing your graders will see! Submit your PDF on eDimension.