Author: Benjamin Smidt
Created: September 24, 2022
Last Updated: September 24, 2022

Assignment 1: Exploring Word Vectors

Note to the reader. This is my work for assignment one of Stanford's course CS 224N: Natural Language Processing with Deep Learning. You can find the lecture Winter 2021 lectures series on YouTube here. This document is meant to be used as a reference, explanation, and resource for the assignment, not necessarily a comprehensive overview of Word Vectors. If there's a typo or a correction needs to be made, feel free to email me at benjamin.smidt@utexas.edu so I can fix it. Thank you! I hope you find this document helpful :).

Contents

1 Count-Based Word Vectors

1.1 Distinct Words

Our first function is pretty simple. Our goal is to flatten the list of lists of words into one long list. From there we remove all the duplicate words and return both the sorted list and its length. See the notebook for links on using list comprehension (to flatten the list of list of words). Python's *set* data structure comes in very handy for removing duplicate words. Finally, we finish by having python convert that set back to a list for us and using the *sorted* function to sort that list.

1.2 Compute Co-Occurence Matrix

This function is slightly more complicated but honestly if you've done some python programming it's fairly trivial. We first grab our corpus as well as our *words* word list and it's length (*num-words*) with the *distinct-words()* function we just implemented. Next we create a dictionary mapping between each word in *words* with some number i and store it in the dictionary *word2ind*. The word's number i will serve as its index along both dimensions of our co-occurence matrix M. Then we initialize our co-occurence matrix M, which has dimensions *num-words* x *num-words* with the first dimension being the center word and the second being the context words (it doesn't really matter which one is which, that's just how I'm thinking about it).

We fill our co-occurence matrix using a for-loop to iteratively compute the number of context words for a given center word. For each center word we move backward one word in the document and use our dictionary *word2ind* to find the proper row (center word) and column (context word) in M, and increment M[*center-word-index*, *context-word-index*] by one. We do this until we've moved backward by *window-size* or until we hit *START*. We repeat the same procedure moving forward except moving forward a word and stopping at *END* or until we've reached a number of words equalling *window-size*. Finally, we return our co-occurence matrix M and our dictionary mapping *word2ind*.

1.3 Reduce to K Dimensions

I'll quite honest here, I'm not very familiar with PCA or SVD. For this assignment it's not necessary to know exactly how it works. The point is that we're extracting the most significant data from our co-occurence matrix to reduce its dimensionality. Regarding code, just use the documentation from sklearn on how to call it and note the description in the notebook about SVD and Truncated SVD options in different libraries.

1.4

2 Prediction-Based Word Vectors

3 Resources

1. CS224N Home Page
2. CS 224N Word Vectors: Introduction, SVD, and Word2Vec
3. CS 224N Lecture 1: Intro and Word Vectors
4. Efficient Estimation of Word Representation in Vector Space (original word2vec paper)
5. Distributed Representations of Words and Phrases and their Compositionality (negative sampling paper)