Homework 2: Vector Space Models and Embeddings
Due: Anytime Sunday, February 18, 2024
Submit: on Blackboard

In this assignment, you will be examining different ways of representing text as vectors, including sparse word vectors that you will create and off-the-shelf word2vec embeddings.

As always, you are allowed to discuss the homework with other students as well as use online resources (provided you list the names of everyone you spoke with and the list of online resources you used at the top of your assignment). However, all submitted code and writing must be your own and you must understand everything you submit. In particular, you are not allowed to use Generative AI tools in completing this assignment.

Potentially useful:

- scipy.spatial.distance.pdist
- Many packages: e.g., scipy, sklearn have functions for calculating cosine, or you can implement it yourself. If you use an off-the-shelf package, be sure to check that it calculates cosine as we defined in class.

Part 1: Train sparse word vectors

In the first part of the assignment, you will be constructing vector representations of words that model a word's context. For a given word, *w*, define a context window of radius *k* to capture *k* words to the left and right of *w*. We wish to represent all contexts around *w* as a "bag of words". Given a corpus of sentences, we construct a fixed-size vocabulary *V* (we will limit this to the top 1000 most frequent words). Given this corpus, create a term-term matrix, in which the rows are words in your vocabulary and the columns are context words. Your task is to transform a corpus of text into word vectors according to this context-window principle, using a context window of *k*=2.

(1) You will use a subset of the Brown corpus to create your word vectors. The full corpus includes >1 million word tokens; however, to ease the processing for this assignment, you will use a subset of ~100,000 word tokens (`brown100k.txt`). Load the data, split by white space and make sure all words are lowercase, but there is no need to do any further processing.
(2) Construct your vectors. Create a term-term matrix in which the rows are words in the vocabulary, the columns are context words, and the cells represent the number of times the context word occurred within k=*2* words of the target word. Limit your vocabulary to the 1,000 most frequent words, but do not limit which context words are included. That is, you will only calculate vectors for 1000 words but you will use all of the words in the corpus to create the context.
    (a) Respond to the questions below about this set of word vectors.

(i) What are the dimensions of your matrix? What determines these dimensions (i.e., why are these the dimensions)? What percentage of the matrix's elements are 0?

(ii) Pick your favorite word in the vocabulary. Show the 20 closest words to your chosen word, calculated using the cosine similarity metric. Are these what you expected? Why or why not?

(iii) Which two words in your vocabulary are most distinct? Which are most similar to each other? Does this make sense to you, or are these surprising?

(3) Next, read in pre-trained word2vec vectors (see Part 3 for instructions).

(a) Respond to the following questions:

(i) Describe the pre-trained vectors. What are their dimensions? What data were they pre-trained on? How many word tokens were they pre-trained on?

(ii) Using the same word as above, show the 20 closest words to it, using cosine similarity. Has anything changed? Are these what you expected? Why or why not?

~~(iii) Which two distinct words are most similar? Least similar? Has this changed? Is this what you expected?~~

(iv) Which of the two embeddings seem best to you, based on what you have observed?

Note: You should respond to these questions based on the Brown subcorpus, but I highly recommend constructing a toy dataset to test your code on before running it on the full data. If you have trouble running your code on the full dataset, please get in touch with me!

Part 2: Skip Grams

In this part of the homework, you will manually perform one step of training a word2vec embedding for a small toy example. The goal is to consolidate how negative sampling skip gram (as described in the textbook and in class) training works.

We define the probability that a word, *w,* co-occurred with the context word, *c*, as follows:

$$P(+|w,c) \;=\; \sigma(\mathbf{c}\cdot\mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c}\cdot\mathbf{w})}$$

We will consider a toy example involving two-dimensional vectors and only one sentence:

"**Cat** litter smells bad".

We will only consider the target word "cat" with a context window of k=1 word. The positive skip gram example (i.e., the tuple of words (target_word, context_word) that show which context word occurred in the context of our target word) is (cat, litter). For each positive skip gram, we

will sample two negative skip grams, by randomly selecting two words in our vocabulary that are not ‘cat’. Let’s say we randomly sample (cat, remote) and (cat, oatmeal). This gives us the following training example, including tuples and their labels:

(cat, litter) -> +
(cat, remote) -> -
(cat, oatmeal) -> -

We start by randomly initializing our embeddings. This gives us the following initial word embeddings:

w_{cat}: [1,1]
c_{remote}: [0,1]
$c_{oatmeal}$: [0, -1]
c_{litter}: [1,0]

(Ok, it wasn’t totally random ;)).

To hand in:

(1) Plot these four vectors. You can use matplotlib or simply plot by hand (it doesn’t need to be super exact, just label the points with their values if you’re approximating the plot).
(2) For each of the three tuples in our training example, does the model output that they are context pairs or not? Is the model correct?
(3) You begin to train your model to arrive at target and context word vectors that will maximize the probability that positive examples are labeled as occurring together and maximize the probability that negative examples are labeled as *not* occurring together. You decide to use the loss function defined in (6.34 in SLP). What is the current loss value?
(4) You decide to perform gradient descent to update your vectors. Step through one step of stochastic gradient descent (consider all three example tuples as one step). Set the learning rate $\eta = 1$. What are the updated vectors? Plot them.
(5) For each of the three tuples and the updated vectors, does the model now make the correct prediction?
(6) What is the new loss function value?
(7) Describe what has changed and how this will over time work to create word embeddings that represent word similarity. Consider how the predictions, vectors, and loss have changed.

Part 3: Evaluating Word2Vec

In this final part of the homework, you will be evaluating word2vec vectors using two of the approaches discussed in class. You may either train up your own word2vec vectors (if you are interested, I highly recommend you do so!), or use downloaded word2vec word embeddings,

which are available for download here (under pretrained word and phrase vectors) and can also be called using gensim (see pretrained models on the linked page). Report which vectors you are using: a good option is GoogleNews-vectors-negative300.bin. Note these vectors were trained in a very similar way to Part 2.

Part 3a. Analogies

One goal for successful embeddings is to be able to learn analogies of the form: Athens : Greece :: Baghdad : _____. Here, you will test whether the embeddings you are working with exhibit this property.

To do so, write code such that given three words, it outputs its answer to the analogy. That is, it should calculate vector_embedding(Greece) - vector_embedding(Athens) + vector_embedding(Baghdad) and return the word that is closest to the resulting vector. You will find a list of questions in the `analogies.txt` file attached to this homework assignment. Note that there are 14 different categories, which are described on lines that start with colons. For example, the first line is ": capitals-common-countries" and the next ~500 lines give examples of this analogy type. You will test how well the vector embeddings you are working with do at predicting the correct answer. Report your model's accuracy by question type. How does the model do? Do you notice patterns in what types of analogies it handles well and what types of analogies it handles poorly?

Part 3b. TOEFL dataset

Finally, you will test your embeddings on the TOEFL synonyms dataset ("Test of English as a Foreign Language"), a dataset introduced by Landauer & Dumais (1997) that is commonly used to evaluate word embeddings. This evaluation set contains 80 multiple choice questions for testing synonym knowledge. For example,

Choose the synonym of: enormously.
Choices: **(a) tremendously** (b) appropriately (c) uniquely (d) decidedly.

You will write code that outputs the predicted answer for a given question. It will do so by calculating the cosine similarity between the target word and each of the four choices and outputting the word that has the highest cosine similarity. The evaluation set can be found in `toefl.txt`. Each line is one question and is structured such that the first word is the target/queried word (i.e., enormously), the second word is the correct answer, and the remaining words are the remaining choices. Report your overall accuracy. How does this compare to chance performance (i.e., the performance you'd expect if the model just chose an answer at random)? Are you impressed with the performance? Do you notice any patterns in what types of questions your model answers correctly or not? (It's okay if you do not, but say so).

How much time did you spend on the homework assignment?