Word Embedding analysis test plan

Chris Gropp

January 2019

1 Introduction

This document describes the process of performing analysis on the word embeddings generated from the TIU notes. It covers the input requirements, how to run the experiments, and some concerns for egressing output.

2 Requirements

- Code, angle list file, and analogy pair file from Gitlab repo:
  https:
  code.ornl.gov/VA-OSL/word_embedding/tree/master/quality
  The code is described throughout, but largely relies on the *w2v_helper.py* file. The angle list file, *medical_angles.txt*, provides a set of angles to examine in detail during the comparative tests, in addition to the randomly chosen ones generated by code. The analogy pair file *pairs_abbreviations_cleaned.txt*, contains pairs of words where the first word is an abbreviation of the second, and is used alongside a vocabulary file to build analogies.

- Word Embeddings
  ASCII format, not binary format.

- Vocabulary files for each embedding:
  If these were not generated by word2vec, there is a script *copyVocabFromEmbeddingFile.py* in the repo that can generate them from the embedding files; this process is described later.

- Libraries: Python3, Numpy
  I did this via conda environment, but it should hopefully be installed already. I tried to minimize external code requirements.

- Test documents
  Needed for Pairwise Inner Product document similarity test. 1000 documents should be fine. These must be preprocessed into the same format as the input word2vec used; subsetting that input is probably the easiest way to do this. Missing vocabulary can be handled by the code, those words will simply be ignored.

3 Experiments

The following section describes the various experiments to run. Example script commands are provided, but you will need to replace filenames; "embedding.vec" with the file for the embedding being tested, for example. In particular, you should ensure that "output.txt" is replaced with a filename that describes the test being performed, and what it is being performed on; for example, for the Easy Cosine Similarity test, you might replace "output.txt" with "easyCosineSim_30million_100.txt" if you were running that test on the embedding trained on 30 million documents using 100 dimensions.

3.1 Standalone Metrics

On each embedding (for loop):

- Create vocab file if it is missing for this embedding.
  python copyVocabFromEmbeddingFile.py embedding.vec vocab.txt
- Build analogy list from pair file and this embedding's vocab file.
  python buildAnalogies.py vocabulary.txt pairs_abbreviations_cleaned.txt analogies.txt
  If these experiments are prohibitively slow, you can add an argument to this script to limit the number of analogies produced; otherwise, it will produce all possible analogies from the pairs in the vocabulary.
  Note that if the vocabularies are sufficiently similar, you can just use the same analogies for all embeddings. If you add the optional argument at the end to trim the number of analogies, you will want to use the same set for all embeddings to ensure they are as comparable as possible.
- Perform each analogy test on the embedding, using the analogy list you just created for this embedding.
  - easyCosineSim
    python analogyTest_EasyCosineSim.py embedding.vec analogies.txt output.txt
    This is by far the fastest test, as each analogy requires a small handful of vector arithmetic operations and containing no other loops around them.
  - cosineSim
    python analogyTest_CosineSim.py embedding.vec analogies.txt output.txt
    Very similar to the above test, but contains a loop through the entire vocabulary for each analogy and is thus substantially slower.
  - pointCloud
    python analogyTest_PointCloud.py embedding.vec analogies.txt output.txt
    Involves a loop over the entire vocabulary for each analogy, and some additional processing that should be fairly minor.

– traditional
python analogyTest_Traditional.py embedding.vec analogies.txt output.txt
Involves a loop over the entire vocabulary for each analogy.

– withinTopX
python analogyTest_WithinTopX.py embedding.vec analogies.txt output.txt 5
This test takes an additional optional argument beyond the others. If nothing is specified, the default value is 10. This value determines how deep the test searches for the target word. You can think of the traditional test as this test where $X = 1$. I'd like to use $X = 5$ for the first set, but may want to try a few other values if we have time. In particular, $X = 2$ seems interesting.

3.2 Comparative Metrics

Before running the main loop, you'll need to make a set of random angles. The angle comparison test will be run on both these and the premade angles included with the code. Making the random angles is fairly straightforward;
python angleGenerator.py vocab.txt randomAngles.txt 100
As input, use the most restrictive vocabulary if possible (presumably, the one from the embedding run on the smallest text sample), as that will make sure as many angles as possible can be used for every comparison. The numeric argument determines the number of angles to generate; 100 should be fine for now.

Instead of being run on a loop of single embeddings, these experiments will be run on a loop of pairs of embeddings. The pairs of embeddings of interest are as follows; an embedding to the next embedding in sequence (1 and 2, 2 and 3, 3 and 4...) and an embedding to the last embedding in sequence (1 and 100, 2 and 100, 3 and 100...). The most important sequence is the one related to input data size; only use the 100 dimension embeddings for that sequence, since we have those at all sizes. At a lower priority, I am also interested in the dimensionality sequences we have at lower input sizes.

On each of the above embedding pairs (for loop):

- Angle comparison test on premade angles
python angleComparison.py embedding1.vec embedding2.vec medical_angles.txt output.txt
Should be reasonably fast; loads both embeddings, does a handful of vector operations for each angle. Because this list was made by hand without any access to the TIU vocabulary, I would not be surprised if there is a lot of output informing you that angles are not within the vocabulary.

- Angle comparison test on random angles you made previously
python angleComparison.py embedding1.vec embedding2.vec randomAngles.txt output.txt

Should be fast, as above, but should have few if any angles not found in vocabulary due to its construction.
IMPORTANT NOTE: These angles contain words drawn from the vocabulary, and are expressed in the output. This may cause issues for egress; I only need the final score, which is just a number, but the other output is helpful for debugging.
If you cannot egress a file containing the random angles (which will be present within the output), use *angleComparison_sanitary.py* instead; it has the same inputs, but outputs no vocabulary information into the output file. Order is preserved such that I can have you track down anything unusual, but I will not have any vocabulary.

- Pairwise Inner Product (PIP) word test
  python pairwiseInnerProduct.py embedding1.vec embedding2.vec output.txt
  This may take awhile, it's $O(d * v^2)$ where d is dimensionality and v is vocab size. If this is prohibitively slow, you can add an additional argument at the end which is a file containing a wordlist; if you do, only those words will be evaluated, so you can set $v = 1000$ or so instead of $v = 250,000$ or so. Ideally those words would be words of relevance; maybe the most frequent 1000 words? But if it runs on the full vocabulary, great.
  This script should not produce any output revealing the words being tested, and should thus be fine for egress.

- Pairwise Inner Product (PIP) document test
  python documentSimilarity.py embedding1.vec embedding2.vec testDocuments.txt output.txt
  This should be much faster than the full word test above. The code will produce document vectors from the word embeddings and test documents provided, and compare the similarity matrix induced by these vectors between the two embeddings. This script should not produce output revealing the test documents, and should thus be okay for egress.

4 Egress Concerns

All output files produced by the experiments themselves should be egressed. The test documents should NOT be egressed. The random angles do not need to be egressed, but would be helpful to examine if they do not contain words of concern.

Most of the scripts produce numerical scores with no other identifying information, and will only be trackable by any information encoded in the filename. As a result, the vast majority of these should be fine. There are, however, a few exceptions.

The angle comparison test includes information on what angles were tested, and while this is no issue for the angles using the premade test set this will include vocabulary elements for the random angles. If any of these words are unsuitable for egress, this logging information should be removed; while it is

helpful for debugging and understanding, it is not essential to the final evaluation. This can be done manually, or by using the *angleComparison_sanitary.py* script instead. Various other scripts may include error messages if vocabularies don't line up, which may also include vocabulary elements. Knowing that there was an element missing is helpful, but if necessary for egress these messages can be culled.