DLS/C5/week 2/Assignment 2

Operations on Word Vectors

Important Note on Submission to the AutoGrader

Before submitting your assignment to the AutoGrader, please make sure you are not doing the following:

1. You have not added any *extra* `print` statement(s) in the assignment.
2. You have not added any *extra* code cell(s) in the assignment.
3. You have not changed any of the function parameters.
4. You are not using any global variables inside your graded exercises. Unless specifically instructed to do so, please refrain from it and use the local variables instead.
5. You are not changing the assignment code where it is not required, like creating *extra* variables.

If you do any of the following, you will get something like, `Grader not found` (or similarly unexpected) error upon submitting your assignment. Before asking for help/debugging the errors in your assignment, check for these first. If this is the case, and you don't remember the changes you have made, you can get a fresh copy of the assignment by following these instructions.

Operations on Word Vectors

Welcome to your first assignment of Week 2, Course 5 of the Deep Learning Specialization!

Because word embeddings are very computationally expensive to train, most ML practitioners will load a pre-trained set of embeddings. In this notebook you'll try your hand at loading, measuring similarity between, and modifying pre-trained embeddings.

After this assignment you'll be able to:

- Explain how word embeddings capture relationships between words
- Load pre-trained word vectors
- Measure similarity between word vectors using cosine similarity
- Use word embeddings to solve word analogy problems such as Man is to Woman as King is to __.

At the end of this notebook you'll have a chance to try an optional exercise, where you'll modify word embeddings to reduce their gender bias. Reducing bias is an important consideration in ML, so you're encouraged to take this challenge!

Table of Contents

Packages

Let's get started! Run the following cell to load the packages you'll need.

In [1]:
```
import numpy as np
from w2v_utils import *
```

1 - Load the Word Vectors

For this assignment, you'll use 50-dimensional GloVe vectors to represent words. Run the following cell to load the `word_to_vec_map`.

In [2]:
```
words, word_to_vec_map = read_glove_vecs('data/glove.6B.50d.txt')
```

You've loaded:

- `words`: set of words in the vocabulary.
- `word_to_vec_map`: dictionary mapping words to their GloVe vector representation.

2 - Embedding Vectors Versus One-Hot Vectors

Recall from the lesson videos that one-hot vectors don't do a good job of capturing the level of similarity between words. This is because every one-hot vector has the same Euclidean distance from any other one-hot vector.

Embedding vectors, such as GloVe vectors, provide much more useful information about the meaning of individual words.
Now, see how you can use GloVe vectors to measure the similarity between two words!

3 - Cosine Similarity

To measure the similarity between two words, you need a way to measure the degree of similarity between two embedding vectors for the two words. Given two vectors u and v, cosine similarity is defined as follows:

$$\text{CosineSimilarity(u, v)} = \frac{u \cdot v}{||u||_2 ||v||_2} = cos(\theta) \tag{1}$$

- $u \cdot v$ is the dot product (or inner product) of two vectors
- $||u||_2$ is the norm (or length) of the vector u
- θ is the angle between u and v.
- The cosine similarity depends on the angle between u and v.
  - If u and v are very similar, their cosine similarity will be close to 1.
  - If they are dissimilar, the cosine similarity will take a smaller value.

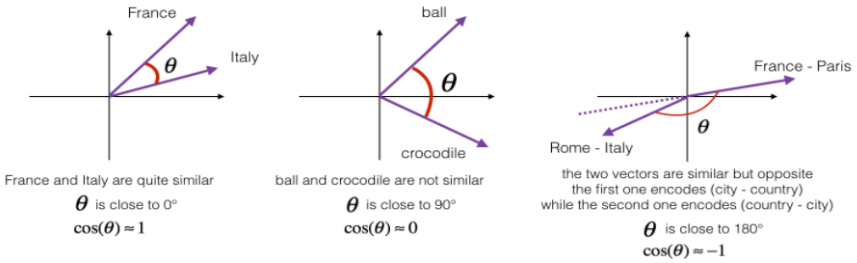


Figure 1: The cosine of the angle between two vectors is a measure of their similarity.

Exercise 1 - cosine_similarity

Implement the function `cosine_similarity()` to evaluate the similarity between word vectors.

Reminder: The norm of u is defined as $||u||_2 = \sqrt{\sum_{i=1}^{n} u_i^2}$

Additional Hints

- You may find np.dot, np.sum, or np.sqrt useful depending upon the implementation that you choose.

4 - Word Analogy Task

- In the word analogy task, complete this sentence:
  "*a* is to *b* as *c* is to ____".
- An example is:
  '*man* is to *woman* as *king* is to *queen*' .
- You're trying to find a word *d*, such that the associated word vectors e_a, e_b, e_c, e_d are related in the following manner:
  $e_b - e_a \approx e_d - e_c$
- Measure the similarity between $e_b - e_a$ and $e_d - e_c$ using cosine similarity.

Exercise 2 - complete_analogy

Complete the code below to perform word analogies!

In [5]:
```
# UNQ_C2 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
```

Hint: Try to find some other analogy pairs that will work, along with some others where the algorithm doesn't give the right answer:

```
* For example, you can try small->smaller as big->?
```

Congratulations!

You've come to the end of the graded portion of the assignment. By now, you've:

- Loaded some pre-trained word vectors
- Measured the similarity between word vectors using cosine similarity
- Used word embeddings to solve word analogy problems such as Man is to Woman as King is to __.

Cosine similarity is a relatively simple and intuitive, yet powerful, method you can use to capture nuanced relationships between words. These exercises should be helpful to you in explaining how it works, and applying it to your own projects!

What you should remember:

- Cosine similarity is a good way to compare the similarity between pairs of word vectors.
  - Note that L2 (Euclidean) distance also works.
- For NLP applications, using a pre-trained set of word vectors is often a great way to get started.

Even though you've finished the graded portion, please take a look at the rest of this notebook to learn about debiasing word vectors.

5 - Debiasing Word Vectors (OPTIONAL/UNGRADED)

In the following exercise, you'll examine gender biases that can be reflected in a word embedding, and explore algorithms for reducing the bias. In addition to learning about the topic of debiasing, this exercise will also help hone your intuition about what word vectors are doing. This section involves a bit of linear algebra, though you can certainly complete it without being an expert! Go ahead and give it a shot. This portion of the notebook is optional and is not graded...so just have fun and explore.

First, see how the GloVe word embeddings relate to gender. You'll begin by computing a vector $g = e_{woman} - e_{man}$, where e_{woman} represents the word vector corresponding to the word *woman*, and e_{man} corresponds to the word vector corresponding to the word *man*. The resulting vector g roughly encodes the concept of "gender".

You might get a more accurate representation if you compute $g_1 = e_{mother} - e_{father}$, $g_2 = e_{girl} - e_{boy}$, etc. and average over them, but just using $e_{woman} - e_{man}$ will give good enough results for now.

In [8]:
```
g = word_to_vec_map['woman'] - word_to_vec_map['man']
print(g)
```

```
[-0.087144    0.2182    -0.40986    -0.03922    -0.1032     0.94165
 -0.06042     0.32988    0.46144    -0.35962     0.31102   -0.86824
  0.96006     0.01073    0.24337     0.08193    -1.02722   -0.21122
  0.695044   -0.00222    0.29106     0.5053     -0.099454   0.40445
  0.30181     0.1355    -0.0606     -0.07131    -0.19245   -0.06115
 -0.3204      0.07165   -0.13337    -0.25068714 -0.14293   -0.224957
 -0.149       0.048882   0.12191    -0.27362    -0.165476  -0.20426
  0.54376    -0.271425  -0.10245    -0.32108     0.2516    -0.33455
 -0.04371     0.01258  ]
```

5.1 - Neutralize Bias for Non-Gender Specific Words

The figure below should help you visualize what neutralizing does. If you're using a 50-dimensional word embedding, the 50 dimensional space can be split into two parts: The bias-direction g, and the remaining 49 dimensions, which is called $g_\perp$ here. In linear algebra, we say that the 49-dimensional $g_\perp$ is perpendicular (or "orthogonal") to g, meaning it is at 90 degrees to g. The neutralization step takes a vector such as $e_{receptionist}$ and zeros out the component in the direction of g, giving us $e^{debiased}_{receptionist}$.

Even though $g_\perp$ is 49-dimensional, given the limitations of what you can draw on a 2D screen, it's illustrated using a 1-dimensional axis below.

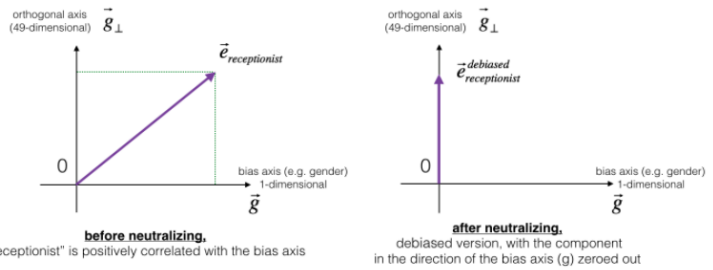


Figure 2: The word vector for "receptionist" represented before and after applying the neutralize operation.

Exercise 3 - neutralize

Implement `neutralize()` to remove the bias of words such as "receptionist" or "scientist."

Given an input embedding e, you can use the following formulas to compute $e^{debiased}$:

$$e^{bias\_component} = \frac{e \cdot g}{||g||_2^2} * g \tag{2}$$

$$e^{debiased} = e - e^{bias\_component} \tag{3}$$

If you are an expert in linear algebra, you may recognize $e^{bias\_component}$ as the projection of e onto the direction g. If you're not an expert in linear algebra, don't worry about this. ;)

In [11]:
```
def neutralize(word, g, word_to_vec_map):
    """
    Removes the bias of "word" by projecting it on the space orthogonal to the bias axis.
    This function ensures that gender neutral words are zero in the gender subspace.
```

```
cosine similarity between receptionist and g, before neutralizing:  0.3307794175059374
cosine similarity between receptionist and g, after neutralizing:  -4.442232511624783e-17
```

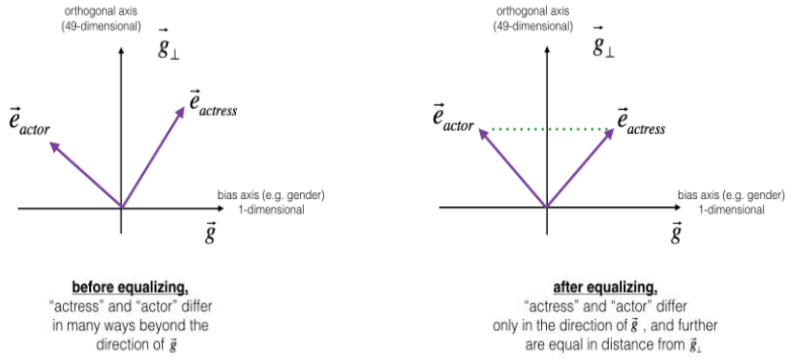
Expected Output: The second result is essentially 0, up to numerical rounding (on the order of 10^{-17}).

cosine similarity between receptionist and g, before neutralizing:	0.330779417506
cosine similarity between receptionist and g, after neutralizing	-4.442232511624783e-17

5.2 - Equalization Algorithm for Gender-Specific Words

Next, let's see how debiasing can also be applied to word pairs such as "actress" and "actor." Equalization is applied to pairs of words that you might want to have differ only through the gender property. As a concrete example, suppose that "actress" is closer to "babysit" than "actor." By applying neutralization to "babysit," you can reduce the gender stereotype associated with babysitting. But this still does not guarantee that "actor" and "actress" are equidistant from "babysit." The equalization algorithm takes care of this.

The key idea behind equalization is to make sure that a particular pair of words are equidistant from the 49-dimensional $g_\perp$. The equalization step also ensures that the two equalized steps are now the same distance from $e^{debiased}_{receptionist}$, or from any other work that has been neutralized. Visually, this is how equalization works:



before equalizing,
"actress" and "actor" differ in many ways beyond the direction of $\vec{g}$

after equalizing,
"actress" and "actor" differ only in the direction of $\vec{g}$, and further are equal in distance from $\vec{g}_\perp$

The derivation of the linear algebra to do this is a bit more complex. (See Bolukbasi et al., 2016 in the References for details.) Here are the key equations:

$$\mu = \frac{e_{w1} + e_{w2}}{2} \tag{4}$$

$$\mu_B = \frac{\mu \cdot \text{bias\_axis}}{||\text{bias\_axis}||_2^2} * \text{bias\_axis} \tag{5}$$

$$\mu_\perp = \mu - \mu_B \tag{6}$$

$$e_{w1B} = \frac{e_{w1} \cdot \text{bias\_axis}}{||\text{bias\_axis}||_2^2} * \text{bias\_axis} \tag{7}$$

$$e_{w2B} = \frac{e_{w2} \cdot \text{bias\_axis}}{||\text{bias\_axis}||_2^2} * \text{bias\_axis} \tag{8}$$

$$e^{corrected}_{w1B} = \sqrt{|1 - ||\mu_\perp||_2^2|} * \frac{e_{\text{w1B}} - \mu_B}{||(e_{w1} - \mu_\perp) - \mu_B||_2} \tag{9}$$

$$e^{corrected}_{w2B} = \sqrt{|1 - ||\mu_\perp||_2^2|} * \frac{e_{\text{w2B}} - \mu_B}{||(e_{w2} - \mu_\perp) - \mu_B||_2} \tag{10}$$

$$e_1 = e^{corrected}_{w1B} + \mu_\perp \tag{11}$$

$$e_2 = e^{corrected}_{w2B} + \mu_\perp \tag{12}$$

Exercise 4 - equalize

Implement the `equalize()` function below.

Use the equations above to get the final equalized version of the pair of words. Good luck!

Hint

- Use np.linalg.norm

In [13]:
```
def equalize(pair, bias_axis, word_to_vec_map):
    """
    Debias gender specific words by following the equalize method described in the figure above.

    Arguments:
    pair -- pair of strings of gender specific words to debias, e.g. ("actress", "actor")
    bias_axis -- numpy-array of shape (50,), vector corresponding to the bias axis, e.g. gender
    word_to_vec_map -- dictionary mapping words to their corresponding vectors

    Returns
    e_1 -- word vector corresponding to the first word
    e_2 -- word vector corresponding to the second word
    """

    ### START CODE HERE ###
    # Step 1: Select word vector representation of "word". Use word_to_vec_map. (≈ 2 lines)
```

```
cosine similarities before equalizing:
cosine_similarity(word_to_vec_map["man"], gender) =  -0.1711095765336832
cosine_similarity(word_to_vec_map["woman"], gender) =  0.3566661846270376

cosine similarities after equalizing:
cosine_similarity(e1, gender) =  -0.7004364289309388
cosine_similarity(e2, gender) =  0.7004364289309388
```

Expected Output:

cosine similarities before equalizing:

cosine_similarity(word_to_vec_map["man"], gender) =	-0.117110957653
cosine_similarity(word_to_vec_map["woman"], gender) =	0.356666188463

cosine similarities after equalizing:

cosine_similarity(e1, gender) =	-0.7004364289309388
cosine_similarity(e2, gender) =	0.7004364289309387

Go ahead and play with the input words in the cell above, to apply equalization to other pairs of words.

Hint: Try...

These debiasing algorithms are very helpful for reducing bias, but aren't perfect and don't eliminate all traces of bias. For example, one weakness of this implementation was that the bias direction g was defined using only the pair of words *woman* and *man*. As discussed earlier, if g were defined by computing $g_1 = e_{woman} - e_{man}$; $g_2 = e_{mother} - e_{father}$; $g_3 = e_{girl} - e_{boy}$; and so on and averaging over them, you would obtain a better estimate of the "gender" dimension in the 50 dimensional word embedding space. Feel free to play with these types of variants as well!

Congratulations!

You have come to the end of both graded and ungraded portions of this notebook, and have seen several of the ways that word vectors can be applied and modified. Great work pushing your knowledge in the areas of neutralizing and equalizing word vectors! See you next time.

6 - References

- The debiasing algorithm is from Bolukbasi et al., 2016, Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
- The GloVe word embeddings were due to Jeffrey Pennington, Richard Socher, and Christopher D. Manning. (https://nlp.stanford.edu/projects/glove/)