AID 729 Assignment 1

Submission Deadline: 6th February 2026

The TA's will check individually if you have been able to work out the codes; assignment should be returned with answers in a document (word / pdf). All these problems are published in open domain and AI generates high quality answers for these. We will share AI generated answers (highlighting human modified parts) after your submission. However, your own learning and grades will be impeded if you resort to copy-pasting AI generated answers.

Problem 1:   Marks 2

We have each used the Word2Vec algorithm to obtain word embeddings for the same vocabulary of words V.

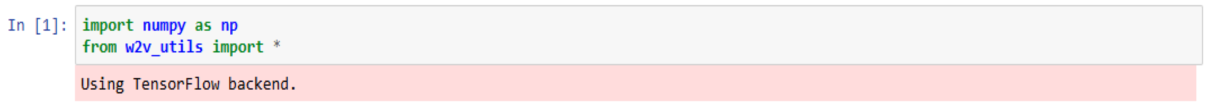
In particular, developer A has got `context' vectors $u_W^A$ and `center' vectors v_W^A for every w in V, and developer B has got `context' vectors $u_W^B$ and `center' vectors v_W^B for every w in V .

For every pair of words w,w' in V, the inner product is the same in both models: $(u_W^A)^T v_{W'}^A = (u_W^B)^T v_{W'}^B$. Does it mean that, for every word w in V, $v_W^A = v_W^B$? Discuss your response.

Problem 2:

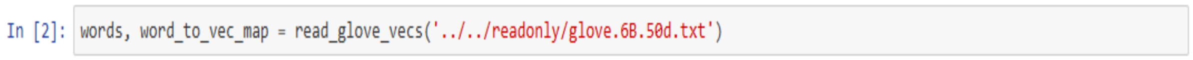
A. Carefully read the original GloVe paper (reference in the end). Examine the Loss Function and highlight the importance of "Bias" terms in the objective function with regards to their purpose and impact on word vector representations in practice.   Marks 4

B. Run the following cell to load the packages you will need. You can use any alternative package you are comfortable in.



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

Using TensorFlow backend.
```

Next, lets load the word vectors. For this assignment, we may 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('../../readonly/glove.6B.50d.txt')
```

To measure how similar two words are, we 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:

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

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

Reminder: The norm of u is defined as

$$||u||_2 = \sqrt{\sum_{i=1}^{n} u_i^2}$$

Compute the Word Similarity and find outputs of the code below: Marks 2

```
In [8]: father = word_to_vec_map["father"]
        mother = word_to_vec_map["mother"]
        ball = word_to_vec_map["ball"]
        crocodile = word_to_vec_map["crocodile"]
        france = word_to_vec_map["france"]
        italy = word_to_vec_map["italy"]
        paris = word_to_vec_map["paris"]
        rome = word_to_vec_map["rome"]

        print("cosine_similarity(father, mother) = ", cosine_similarity(father, mother))
        print("cosine_similarity(ball, crocodile) = ",cosine_similarity(ball, crocodile))
        print("cosine_similarity(france - paris, rome - italy) = ",cosine_similarity(france - paris, rome - italy))
```

C. Word Analogy Task:
In the word analogy task, we complete the sentence "*a* is to *b* as *c* is to **____**". An example is '*man* is to *woman* as *king* is to *queen*. In detail, we are 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: $e_b - e_a \approx e_d - e_c$. We will measure the similarity between $e_b - e_a$ and $e_d - e_c$ using cosine similarity.

Complete the word analogy code and test with the triads below: Marks 3

```
triads_to_try = [('italy', 'italian', 'spain'), ('india', 'delhi', 'japan'), ('man', 'woman', 'boy'), ('small', 'smaller', 'larg
e')]
for triad in triads_to_try:
    print ('{} -> {} :: {} -> {}'.format( *triad, complete_analogy(*triad,word_to_vec_map)))
```

D. Read the Glove exercise for gender bias:

Lets first see how the GloVe word embeddings relate to gender. You will first compute 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.)

Execute the following pieces of code and explain the cosine similarity value you obtained. What happens with name words as against other words?   Marks 3

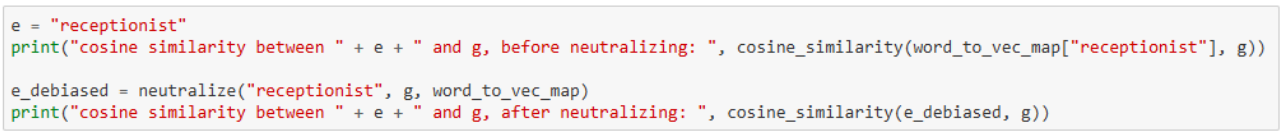
```
print ('List of names and their similarities with constructed vector:')

# girls and boys name
name_list = ['john', 'marie', 'sophie', 'ronaldo', 'priya', 'rahul', 'danielle', 'reza', 'katy', 'yasmin']

for w in name_list:
    print (w, cosine_similarity(word_to_vec_map[w], g))
```

```
print('Other words and their similarities:')
word_list = ['lipstick', 'guns', 'science', 'arts', 'literature', 'warrior','doctor', 'tree', 'receptionist',
             'technology',  'fashion', 'teacher', 'engineer', 'pilot', 'computer', 'singer']
for w in word_list:
    print (w, cosine_similarity(word_to_vec_map[w], g))
```

E. Implement a function `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}

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

Check if you have de-biased "receptionist" Marks 3



```
e = "receptionist"
print("cosine similarity between " + e + " and g, before neutralizing: ", cosine_similarity(word_to_vec_map["receptionist"], g))

e_debiased = neutralize("receptionist", g, word_to_vec_map)
print("cosine similarity between " + e + " and g, after neutralizing: ", cosine_similarity(e_debiased, g))
```

F. 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 neutralizing to "babysit" we can reduce the gender-stereotype associated with babysitting. But this still does not guarantee that "actor" and "actress" are equidistant from "babysit."
Implement a function equalize () using the equations below to equalize "man" and "woman": Marks 5

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

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

$$\mu_\perp = \mu - \mu_B$$

$$e_{w1B} = \frac{e_{w1} \cdot \text{bias\_axis}}{||\text{bias\_axis}||_2^2} * \text{bias\_axis}$$
$$e_{w2B} = \frac{e_{w2} \cdot \text{bias\_axis}}{||\text{bias\_axis}||_2^2} * \text{bias\_axis}$$

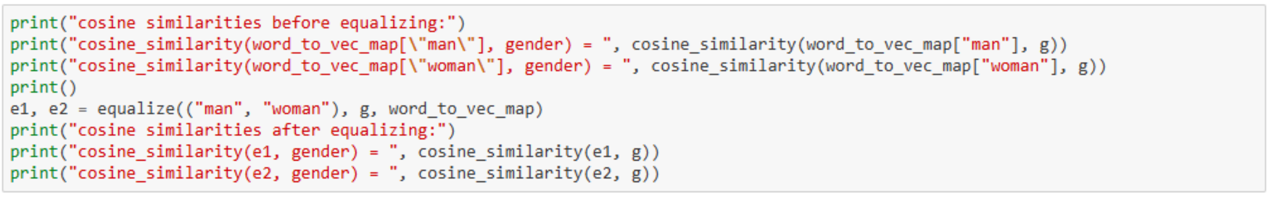
$$e_{w1B}^{corrected} = \sqrt{|1 - ||\mu_\perp||_2^2|} * \frac{e_{\text{w1B}} - \mu_B}{|(e_{w1} - \mu_\perp) - \mu_B)|}$$

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

$$e_1 = e_{w1B}^{corrected} + \mu_\perp$$
$$e_2 = e_{w2B}^{corrected} + \mu_\perp$$

bias_axis in our case is "gender" (g as we used previously)

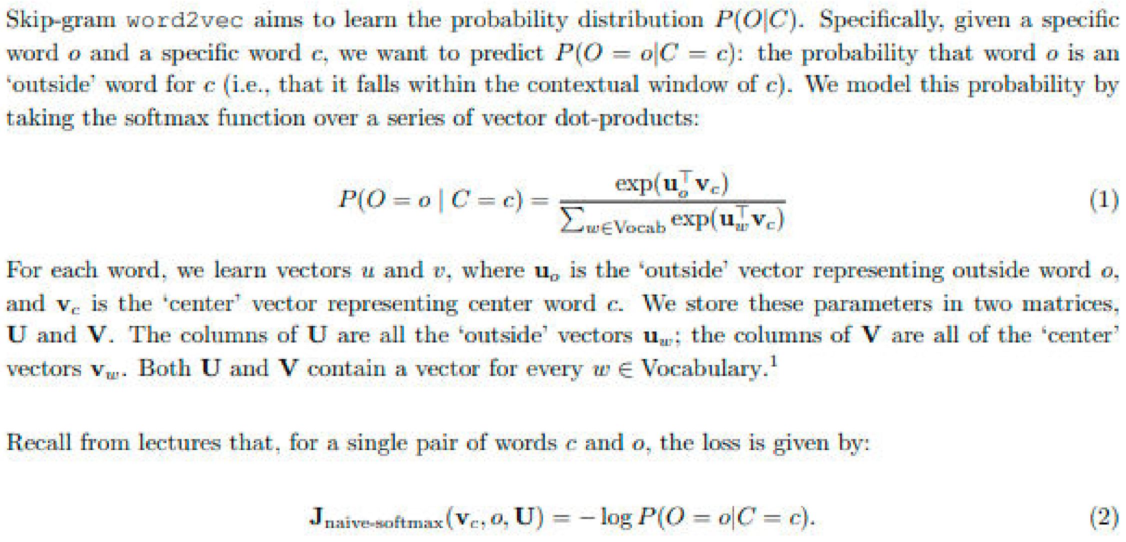
Test your findings with the code below (you are comparing cosine similarities before equalizing and after equalizing:



```
print("cosine similarities before equalizing:")
print("cosine_similarity(word_to_vec_map[\"man\"], gender) = ", cosine_similarity(word_to_vec_map["man"], g))
print("cosine_similarity(word_to_vec_map[\"woman\"], gender) = ", cosine_similarity(word_to_vec_map["woman"], g))
print()
e1, e2 = equalize(("man", "woman"), g, word_to_vec_map)
print("cosine similarities after equalizing:")
print("cosine_similarity(e1, gender) = ", cosine_similarity(e1, g))
print("cosine_similarity(e2, gender) = ", cosine_similarity(e2, g))
```

Problem 3: Marks 3

Read Chris Manning's note below:



Skip-gram word2vec aims to learn the probability distribution $P(O|C)$. Specifically, given a specific word o and a specific word c, we want to predict $P(O=o|C=c)$: the probability that word o is an 'outside' word for c (i.e., that it falls within the contextual window of c). We model this probability by taking the softmax function over a series of vector dot-products:

$$P(O = o \mid C = c) = \frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\mathbf{u}_w^\top \mathbf{v}_c)} \tag{1}$$

For each word, we learn vectors u and v, where $\mathbf{u}_o$ is the 'outside' vector representing outside word o, and $\mathbf{v}_c$ is the 'center' vector representing center word c. We store these parameters in two matrices, $\mathbf{U}$ and $\mathbf{V}$. The columns of $\mathbf{U}$ are all the 'outside' vectors $\mathbf{u}_w$; the columns of $\mathbf{V}$ are all of the 'center' vectors $\mathbf{v}_w$. Both $\mathbf{U}$ and $\mathbf{V}$ contain a vector for every $w \in$ Vocabulary.[1]

Recall from lectures that, for a single pair of words c and o, the loss is given by:

$$\mathbf{J}_{\text{naive-softmax}}(\mathbf{v}_c, o, \mathbf{U}) = -\log P(O = o|C = c). \tag{2}$$

Prove that the naive-softmax loss (Equation 2) is the same as the cross-entropy loss between y and yˆ, mentioned as: (note that y (true distribution), yˆ (predicted distribution) are vectors and yˆ$_o$ is a scalar):

$$-\sum_{w \in \text{Vocab}} \mathbf{y}_w \log(\hat{\mathbf{y}}_w) = -\log(\hat{\mathbf{y}}_o).$$

The proof needs one sentence of maths and two sentences of explanation.

References:

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