The link below is my github address to see the code for Lab4 and responses to the questions Edward Cruz, Jr. 11.17.24

https://github.com/ecruz0369/DA6233ECruz--2023/blob/main/IS6733Lab4ARCECruzkml614.ipynb

Debiasing word embeddings

Word embedding are word vectors that have meaning, word vectors similar to each other will be close to each other in a vector space.

After completing this lab you will be to:

- Use and load pre-trained word vectors
- Measure similarity of word vectors using cosine similarity
- Solve word analogy probelms such as Man is to Woman as Boy is to ____ using word embeddings
- Reduce gender bias in word embeddings by modifying word embeddings to remove gender stereotypes, such as the association between the words *receptionist* and *female*

Word embeddings

Word embedding is a method used to represent words as vectors. They are populary used in machine learning and natural language processing tasks. Despite their success in downstream tasks such as cyberbullying, sentiment analysis, and question retrieval, they exhibit gender sterotypes which raises concerns because their widespread use can amplify these biases.

Word embeddings are trained on word co-occurance using a text dataset. After training, each word w will be represented as a d-dimensional word vector $\vec{w} \epsilon \mathbb{R}^d$.

Word embedding properties:

- Words with similar semantic meaning will be close to each other
- The difference between word embedding vectors can represent relationships between words. For example, given the analogy "man is to King as woman is to x" (denoted as $man : king :: woman : x$), by doing simple arithmetic on the embedding vectors, we find that $x = queen$ is the best answer because $\vec{man} - \vec{woman} \approx \vec{king} - \vec{queen}$. For the analogy $Paris : France :: Nairobi : x$, finds that $x = Kenya$. These embeddings can also amplify sexism implicit in text. For instance, $\vec{man} - \vec{woman} \approx computer\ \vec{programmer} - \vec{homemaker}$. The same system that produced reasonable answers to the previous examples offensively answers "man is to computer programmer as woman is to x" with $x = homemaker$.

Run the following cell to load the required modules.

In [108…

```
import os
import json
import numpy as np
from pathlib import Path
from sklearn.decomposition import PCA
```

```
import zipfile
```

Download and Load word vectors

Due to the computational resources required to train word embeddings, we will be using a pre-trained 50-dimensional word embeddings, GloVe to represent words.

Run the following cells to download and load the word embeddings.

In [109…

```
def download_glove_vectors():
    '''
    Download the GloVe vectors
    Arguments:
        None
    Returns:
        file_name (String): The absolute path of the downloaded 50-dimensional
        GloVe word vector representations
    '''

    if not Path('data').is_dir():
        print("Downloading the embeddings ...")
        !wget --quiet https://nlp.stanford.edu/data/glove.6B.zip
        print("Embeddings downloaded.")

        # Unzip it
        print("Unzipping the downloaded file ...")
        !unzip -q glove.6B.zip -d data/
        print("File unzipped.")

    return '/content/data/glove.6B.50d.txt'
```

In [110…

```
def get_glove_vectors(glove_file):
    '''
    Read the word vectors in glove_file
    Arguments:
        glove_file (String): The absolute path to the downloaded glove word embeddings
    Returns:
        words (Set): The words (vocabulary) in the pretrained glove word embeddings
        word_to_vector_map (Dict): A dictionary mapping the each word to its embedding vector
    '''

    words = set()
    word_to_vector_map = {}
    with open(glove_file, 'r') as file_handle:
        for line in file_handle:
            line = line.strip().split()
            current_word = line[0]
```

```python
            current_word = line[0]
            words.add(current_word)
            current_word_vector = line[1:]
            word_to_vector_map[current_word] = np.array(current_word_vector, dtype=np.float64)

    return words, word_to_vector_map
```

In [111…

```python
os.getcwd()
```

Out[111…

```
'/home/kml614/IS6734Labs'
```

In [112…

```python
def get_glove_vectors(zip_file_path):
    """
    Loads GloVe word embeddings from a zip file.

    Args:
        zip_file_path: Path to the zip file containing the GloVe embeddings.

    Returns:
        A tuple of:
            - A set of words in the vocabulary.
            - A dictionary mapping words to their GloVe vector representations.
    """
    words = set()
    word_to_vector_map = {}

    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        # Try to read the file inside the zip archive
        with zip_ref.open('glove.6B.50d.txt', 'r') as file_handle:
            for line in file_handle:
                # Decode the binary content to text
                decoded_line = line.decode('utf-8').strip()
                split_line = decoded_line.split()
                current_word = split_line[0]
                vector = np.asarray([float(v) for v in split_line[1:]], dtype='float32')
                words.add(current_word)
                word_to_vector_map[current_word] = vector

    return words, word_to_vector_map

# Assuming the zip file is in the specified directory
zip_file_path = "/home/kml614/IS6734Labs/glove.6B.zip"

# Call the function
words, word_to_vector_map = get_glove_vectors(zip_file_path)

# Use the loaded words and word vectors
print(f"Number of words loaded: {len(words)}")
```

```
Number of words loaded: 400000
```

Number of words loaded: 400000

Operations on word embeddings

Task 1 - Cosine similarity

Similarity between two words represented as word vectors u and v can be measured by their cosine similarity:

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

Where:

$u \cdot v$ is the dot (inner) product of the two vectors

$||u||_2$ is the length of the vector u. The length also called Euclidean length or Euclidean norm defines a distance function defined as $||u||_2 = \sqrt{u_1^2 + \ldots + u_n^2}$

The normalized similarity between u and v is the cosine of the angle between the two vectors denoted as θ. The cosine similarity of u and v will be close to 1 if the two vectors are similar, otherwise, the cosine similarity will be small.

Note: We will be refering to the embedding of a word i.e the word vector and the word interchangeably in this lab.

Task 1a: Implement equation 1 in the `cosine_similarity()` function below.
Hint: check out the numpy documentation on np.dot, np.sum, and np.sqrt. Depending on how you choose to implement it, you can check out np.linalg.norm.

In [113…

```
#Task 1a: Implementing Cosine Similarity
#The cosine_similarity function measures how closely two vectors align in space. It does this by comparing their directions,
#regardless of magnitude, using the formula: dot product of the vectors divided by the product of their magnitudes.
#This produces a value between -1 and 1, where 1 indicates identical direction and -1 indicates opposite directions.
```

In [114…

```
def cosine_similarity(vector1, vector2):
    """
    Calculates the cosine similarity of two word vectors - vector1 and vector2
    Arguments:
        vector1 (ndarray): A word vector having shape (n,)
        vector2 (ndarray): A word vector having shape (n,)
    Returns:
        cosine_similarity (float): The cosine similarity between vector1 and vector2
    """
```

```python
    # Compute the dot product between vector1 and vector2
    dot = np.dot(vector1, vector2)

    # Compute the Euclidean norm or length of vector1
    norm_vector1 = np.linalg.norm(vector1)

    # Compute the Euclidean norm or length of vector2
    norm_vector2 = np.linalg.norm(vector2)

    # Compute the cosine similarity as defined in equation 1
    cosine_similarity = dot / (norm_vector1 * norm_vector2)

    return cosine_similarity
```

In [115…

```python
# Run this cell to obtain and report your answers
man = word_to_vector_map["man"]
woman = word_to_vector_map["woman"]
cat = word_to_vector_map["cat"]
dog = word_to_vector_map["dog"]
orange = word_to_vector_map["orange"]
england = word_to_vector_map["england"]
london = word_to_vector_map["london"]
edinburgh = word_to_vector_map["edinburgh"]
scotland = word_to_vector_map["scotland"]

print(f"Cosine similarity between man and woman: {cosine_similarity(man, woman)}")
print(f"Cosine similarity between cat and dog: {cosine_similarity(cat, dog)}")
print(f"Cosine similarity between cat and cow: {cosine_similarity(cat, orange)}")
print(f"Cosine similarity between england - london and edinburgh - scotland: {cosine_similarity(england - london, edinburgh -
```

```
Cosine similarity between man and woman: 0.8860337734222412
Cosine similarity between cat and dog: 0.9218005537986755
Cosine similarity between cat and cow: 0.40695685148239136
Cosine similarity between england - london and edinburgh - scotland: -0.5203389525413513
```

Task 1b: In the code cell below, try out 3 of your own inputs here and report your inputs and outputs

In [116…

```python
#France and Paris show the highest similarity (0.80), which makes sense since there's a strong geographic relationship - Pari
#is France's capital city. King and Queen show similarly high similarity (0.78), reflecting their closely related royal roles
#and positions.
#Apple and Fruit show moderate similarity (0.59), which is expected since apple is a type of fruit - there's a clear
#relationship, but it's not as strong as the other pairs since 'fruit'is a broader category term while 'apple' is a
#specific instance.
```

In [117…

```python
# Start code here #
```

```
king = word_to_vector_map["king"]
queen = word_to_vector_map["queen"]
france = word_to_vector_map["france"]
paris = word_to_vector_map["paris"]
apple = word_to_vector_map["apple"]
fruit = word_to_vector_map["fruit"]

print(f"Cosine similarity between king and queen: {cosine_similarity(king, queen)}")
print(f"Cosine similarity between france and paris: {cosine_similarity(france, paris)}")
print(f"Cosine similarity between apple and fruit: {cosine_similarity(apple, fruit)}")
# End code here #
```

```
Cosine similarity between king and queen: 0.7839043140411377
Cosine similarity between france and paris: 0.8025329113006592
Cosine similarity between apple and fruit: 0.5917636156082153
```

Task 2 - Word analogy

In an analogy task, you are given an analogy in the form "i is to j as k is to ___". Your task is to complete this sentence.

For example, if you are given "man is to king as woman is to l" (denoted as $man : king :: woman : l$). You are to find the best word l that answers the analogy the best. Simple arithmetic of the embedding vectors will find that $l = queen$ is the best answer because the embedding vectors of words i, j, k, and l denoted as e_i, e_j, e_k, e_l have the following relationship:

$$e_j - e_i \approx e_l - e_k$$

Cosine similarity can be used to measure the similarity between $e_j - e_i$ and $e_l - e_k$

Task 2a: To perform word analogies, implement `answer_analogy()` below.

In [118…

```
#Word embeddings can solve analogies by using vector arithmetic - if we subtract and add the right vectors,
#we can find relationships ike "France is to Paris as China is to Beijing." The model finds the word whose
#vector is most similar to the result of this
#arithmetic (Paris - France + China ≈ Beijing), showing how embeddings capture meaningful semantic
#relationships between words.
```

In [119…

```
# Fixed the issue this code now works Nov 14, 2024 ECruz
def answer_analogy(word_i, word_j, word_k, word_to_vector_map):
    """
    Performs word analogy as described above.
    Arguments:
        word_i (str): A word.
        word_j (str): A word.
        word_k (str): A word.
        word_to_vector_map (dict): A dictionary mapping words to embedding vectors.
    Returns:
```

```python
    Returns:
        best_word (str): A word that fulfills the analogy, based on cosine similarity.
    """
    # Convert words to lowercase
    word_i, word_j, word_k = word_i.lower(), word_j.lower(), word_k.lower()

    # Get embedding vectors for the input words
    try:
        embedding_vector_of_word_i = word_to_vector_map[word_i]
        embedding_vector_of_word_j = word_to_vector_map[word_j]
        embedding_vector_of_word_k = word_to_vector_map[word_k]
    except KeyError as e:
        print(f"{e.args[0]} is not in the vocabulary. Please try a different word.")
        return None

    # Initialize variables for tracking the best word
    words = word_to_vector_map.keys()
    max_cosine_similarity = -1e9
    best_word = None
    input_words = {word_i, word_j, word_k}

    for word in words:
        if word in input_words:
            continue

        # Compute the cosine similarity
        embedding_vector_of_word_l = word_to_vector_map[word]
        a = embedding_vector_of_word_j - embedding_vector_of_word_i
        b = embedding_vector_of_word_l - embedding_vector_of_word_k
        similarity = np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

        # Update the best word if similarity is higher
        if similarity > max_cosine_similarity:
            max_cosine_similarity = similarity
            best_word = word

    return best_word
```

Task 2b: Test your implementation by running the code cell below. What are your observations? What do you observe about the last two outputs?.

In [120…

```python
#Common linguistic patterns like "france:french::germany:german" are handled well, showing the model's strength with
#regular language relationships.
#However, when testing "man:doctor::woman:nurse," the model reveals concerning gender biases present in its training data,
#defaulting to stereotypical professional associations rather than maintaining equivalent roles.
```

In [121…

```
analogies = [('france', 'french', 'germany'),
             ('england', 'london', 'japan'),
             ('boy', 'girl', 'man'),
             ('man', 'doctor', 'woman'),
             ('small', 'smaller', 'big')]
for analogy in analogies:
    best_word = answer_analogy(*analogy, word_to_vector_map)
    if best_word:
        print(f"{analogy[0]} -> {analogy[1]} :: {analogy[2]} -> {best_word}")
```

```
france -> french :: germany -> german
england -> london :: japan -> tokyo
boy -> girl :: man -> woman
man -> doctor :: woman -> nurse
small -> smaller :: big -> competitors
```

Task 2c: Try your own analogies by completing and executing the code cell below. Find 2 that works and one that doesn't. Report your inputs and outputs

---

In [122…

```
# Define your own analogies here
# Define your own analogies here
my_analogies = [
    ('king', 'queen', 'prince'),
    ('paris', 'france', 'nice'),
    ('car', 'road', 'sign'),
    ('brother', 'sister', 'uncle'),
    ('teacher', 'school', 'doctor'),
    ('cat', 'kitten', 'dog'),
]

# Execute the analogies
for analogy in my_analogies:
    best_word = answer_analogy(*analogy, word_to_vector_map)
    print(f"{analogy[0]} -> {analogy[1]} :: {analogy[2]} -> {best_word}")
```

```
king -> queen :: prince -> princess
paris -> france :: nice -> croatia
car -> road :: sign -> transfrontier
brother -> sister :: uncle -> aunt
teacher -> school :: doctor -> college
cat -> kitten :: dog -> warmonger
```

Task 3 - Geometry of Gender and Bias in Word Embeddings: Occupational stereotypes

In this task, we will understand the biases present in word-embedding i.e which words are closer to she than to he. This will be achieved by evaluating whether the GloVe embeddings have sterotypes on occupation words. Determine gender bias by projecting each of the occupations onto the $she - he$ direction by computing the dot product between each occupation word embedding and the embedding vector of $she - he$ normalized by the Euclidean norm (See task 1).

$$occupation\_word_i \cdot ||she - he||_2 \tag{2}$$

Notice that equation 2 is similar to only the numerator of equation 1 because we are computing the dot product of $occupation\_word_i$ and the normalized difference between she and he.

Run the cells below to download and view the occupations.

In [123…

```
from pathlib import Path

def download_occupations():
    if not Path('debiaswe').is_dir():
        print("Downloading occupation list ...")
        !git clone -q https://github.com/tolga-b/debiaswe.git
        print("Occupation list downloaded.")
    else:
        print("Repository already exists.")
    return 'debiaswe/data/professions.json'

def view_occupations(occupations_file):
    if not Path(occupations_file).is_file():
        print(f"File not found: {occupations_file}")
        return

    with open(occupations_file, 'r') as file_handle:
        occupations = json.load(file_handle)

        for occupation in occupations:
            print(occupation[0])
```

In [124…

```
occupations_file = download_occupations()
```

```
Repository already exists.
```

In [125…

```
view_occupations(occupations_file)
```

```
accountant
acquaintance
actor
actress
adjunct_professor
administrator
adventurer
advocate
aide
```

```
alderman
alter_ego
ambassador
analyst
anthropologist
archaeologist
archbishop
architect
artist
artiste
assassin
assistant_professor
associate_dean
associate_professor
astronaut
astronomer
athlete
athletic_director
attorney
author
baker
ballerina
ballplayer
banker
barber
baron
barrister
bartender
biologist
bishop
bodyguard
bookkeeper
boss
boxer
broadcaster
broker
bureaucrat
businessman
businesswoman
butcher
butler
cab_driver
cabbie
cameraman
campaigner
captain
cardiologist
caretaker
carpenter
cartoonist
cellist
chancellor
```

```
chaplain
character
chef
chemist
choreographer
cinematographer
citizen
civil_servant
cleric
clerk
coach
collector
colonel
columnist
comedian
comic
commander
commentator
commissioner
composer
conductor
confesses
congressman
constable
consultant
cop
correspondent
councilman
councilor
counselor
critic
crooner
crusader
curator
custodian
dad
dancer
dean
dentist
deputy
dermatologist
detective
diplomat
director
disc_jockey
doctor
doctoral_student
drug_addict
drummer
economics_professor
economist
editor
```

```
educator
electrician
employee
entertainer
entrepreneur
environmentalist
envoy
epidemiologist
evangelist
farmer
fashion_designer
fighter_pilot
filmmaker
financier
firebrand
firefighter
fireman
fisherman
footballer
foreman
freelance_writer
gangster
gardener
geologist
goalkeeper
graphic_designer
guidance_counselor
guitarist
hairdresser
handyman
headmaster
historian
hitman
homemaker
hooker
housekeeper
housewife
illustrator
industrialist
infielder
inspector
instructor
interior_designer
inventor
investigator
investment_banker
janitor
jeweler
journalist
judge
jurist
```

```
laborer
landlord
lawmaker
lawyer
lecturer
legislator
librarian
lieutenant
lifeguard
lyricist
maestro
magician
magistrate
maid
major_leaguer
manager
marksman
marshal
mathematician
mechanic
mediator
medic
midfielder
minister
missionary
mobster
monk
musician
nanny
narrator
naturalist
negotiator
neurologist
neurosurgeon
novelist
nun
nurse
observer
officer
organist
painter
paralegal
parishioner
parliamentarian
pastor
pathologist
patrolman
pediatrician
performer
pharmacist
philanthropist
philosopher
```

photographer
photojournalist
physician
physicist
pianist
planner
plastic_surgeon
playwright
plumber
poet
policeman
politician
pollster
preacher
president
priest
principal
prisoner
professor
professor_emeritus
programmer
promoter
proprietor
prosecutor
protagonist
protege
protester
provost
psychiatrist
psychologist
publicist
pundit
rabbi
radiologist
ranger
realtor
receptionist
registered_nurse
researcher
restaurateur
sailor
saint
salesman
saxophonist
scholar
scientist
screenwriter
sculptor
secretary
senator
sergeant
servant

```
serviceman
sheriff_deputy
shopkeeper
singer
singer_songwriter
skipper
socialite
sociologist
soft_spoken
soldier
solicitor
solicitor_general
soloist
sportsman
sportswriter
statesman
steward
stockbroker
strategist
student
stylist
substitute
superintendent
surgeon
surveyor
swimmer
taxi_driver
teacher
technician
teenager
therapist
trader
treasurer
trooper
trucker
trumpeter
tutor
tycoon
undersecretary
understudy
valedictorian
vice_chancellor
violinist
vocalist
waiter
waitress
warden
warrior
welder
worker
wrestler
writer
```

Task 3a: Complete the `get_occupation_stereotypes()` below.

In [126…

```
def get_occupation_stereotypes(she, he, occupations_file, word_to_vector_map, verbose=False):
    """
    occupation_words = [occupation[0] for occupation in occupations]
    Arguments:
        she (String): A word
        he (String): A word
        occupations_file (String): The path to the occupation file
        word_to_vector_map (Dict): A dictionary mapping words to embedding vectors
    Returns:
        most_similar_words (Tuple(List[Tuple(Float, String)], List[Tuple(Float, String)])):
        A tuple of the list of the most similar occupation words to she and he with their associated similarity

    """
    # Read occupations
    with open(occupations_file, 'r') as file_handle:
        occupations = json.load(file_handle)

    # Extract occupation words
    occupation_words = [occupation[0] for occupation in occupations]
    print(f"Loaded {len(occupation_words)} occupation words.")

    # Get embedding vector of she
    embedding_vector_she = word_to_vector_map.get(she)
    embedding_vector_he = word_to_vector_map.get(he)
    if embedding_vector_she is None or embedding_vector_he is None:
        print("Error: 'she' or 'he' not found in the word embedding vocabulary.")
        return [], []

    # Compute normalized vector difference
    vector_difference_she_he = embedding_vector_she - embedding_vector_he
    normalized_difference_she_he = vector_difference_she_he / np.linalg.norm(vector_difference_she_he)

    # Compute similarities
    similarities = []
    for word in occupation_words:
        try:
            occupation_word_embedding_vector = word_to_vector_map[word]
            similarity = np.dot(occupation_word_embedding_vector, normalized_difference_she_he) / np.linalg.norm(occupation_w
            similarities.append((similarity, word))
        except KeyError:
            if verbose:
                print(f"'{word}' not found in vocabulary.")

    # Sort and return top and bottom results
    most_similar_words = sorted(similarities)
```

```python
    print(f"Found {len(most_similar_words)} similarities computed.")
    return most_similar_words[:20], most_similar_words[-20:]
```

In [127…

```python
top_20, bottom_20 = get_occupation_stereotypes('she', 'he', occupations_file, word_to_vector_map, verbose=True)
print("Top 20 most similar:", top_20)
print("Bottom 20 least similar:", bottom_20)
```

```
Loaded 320 occupation words.
'adjunct_professor' not found in vocabulary.
'alter_ego' not found in vocabulary.
'assistant_professor' not found in vocabulary.
'associate_dean' not found in vocabulary.
'associate_professor' not found in vocabulary.
'athletic_director' not found in vocabulary.
'cab_driver' not found in vocabulary.
'civil_servant' not found in vocabulary.
'disc_jockey' not found in vocabulary.
'doctoral_student' not found in vocabulary.
'drug_addict' not found in vocabulary.
'economics_professor' not found in vocabulary.
'fashion_designer' not found in vocabulary.
'fighter_pilot' not found in vocabulary.
'freelance_writer' not found in vocabulary.
'graphic_designer' not found in vocabulary.
'guidance_counselor' not found in vocabulary.
'interior_designer' not found in vocabulary.
'investment_banker' not found in vocabulary.
'major_leaguer' not found in vocabulary.
'plastic_surgeon' not found in vocabulary.
'professor_emeritus' not found in vocabulary.
'registered_nurse' not found in vocabulary.
'sheriff_deputy' not found in vocabulary.
'singer_songwriter' not found in vocabulary.
'soft_spoken' not found in vocabulary.
'solicitor_general' not found in vocabulary.
'taxi_driver' not found in vocabulary.
'vice_chancellor' not found in vocabulary.
Found 291 similarities computed.
Top 20 most similar: [(-0.35621235, 'coach'), (-0.33694607, 'caretaker'), (-0.31634083, 'captain'), (-0.30927327, 'marshal'),
(-0.30729136, 'colonel'), (-0.30248713, 'skipper'), (-0.30214384, 'manager'), (-0.3016537, 'midfielder'), (-0.29967117, 'archb
ishop'), (-0.2944306, 'commander'), (-0.29028675, 'footballer'), (-0.2888985, 'bishop'), (-0.28199962, 'marksman'), (-0.279638
83, 'firebrand'), (-0.27878764, 'provost'), (-0.2780793, 'substitute'), (-0.27217972, 'lieutenant'), (-0.2719793, 'custodia
n'), (-0.2719191, 'superintendent'), (-0.27134648, 'goalkeeper')]
Bottom 20 least similar: [(0.32072738, 'singer'), (0.3219568, 'publicist'), (0.34405202, 'nanny'), (0.34466952, 'therapist'),
(0.34746587, 'confesses'), (0.35577607, 'businesswoman'), (0.35730487, 'dancer'), (0.36456618, 'hairdresser'), (0.3698354, 're
ceptionist'), (0.37291065, 'housekeeper'), (0.37309748, 'homemaker'), (0.3812397, 'housewife'), (0.38133988, 'nurse'), (0.3892
646, 'narrator'), (0.41383365, 'maid'), (0.42853132, 'socialite'), (0.44343776, 'waitress'), (0.44732913, 'stylist'), (0.46861
386, 'ballerina'), (0.49840298, 'actress')]
```

Task 3b: Execute the cell below and report your results.

1. Does the GloVe word embeddings propagate bias? why?

2. From the list associated with she, list those that reflect gender stereotype.

3. Compare your list from 2 to the occupations closest to he. What are your conclusions?

Exclude businesswoman from your list.

In [128…

```
he, she = get_occupation_stereotypes('she', 'he', occupations_file, word_to_vector_map)

print("Occupations closest to he:")
for occupation in he:
    print(f"{occupation[0], occupation[1]}")

print("\nOccupations closest to she:")
for occupation in she:
    if occupation[1] != 'businesswoman': # Excluding businesswoman
        print(f"{occupation[0], occupation[1]}")
```

```
Loaded 320 occupation words.
Found 291 similarities computed.
Occupations closest to he:
(-0.35621235, 'coach')
(-0.33694607, 'caretaker')
(-0.31634083, 'captain')
(-0.30927327, 'marshal')
(-0.30729136, 'colonel')
(-0.30248713, 'skipper')
(-0.30214384, 'manager')
(-0.3016537, 'midfielder')
(-0.29967117, 'archbishop')
(-0.2944306, 'commander')
(-0.29028675, 'footballer')
(-0.2888985, 'bishop')
(-0.28199962, 'marksman')
(-0.27963883, 'firebrand')
(-0.27878764, 'provost')
(-0.2780793, 'substitute')
(-0.27217972, 'lieutenant')
(-0.2719793, 'custodian')
(-0.2719191, 'superintendent')
(-0.27134648, 'goalkeeper')

Occupations closest to she:
(0.32072738, 'singer')
```

```
(0.3219568, 'publicist')
(0.34405202, 'nanny')
(0.34466952, 'therapist')
(0.34746587, 'confesses')
(0.35730487, 'dancer')
(0.36456618, 'hairdresser')
(0.3698354, 'receptionist')
(0.37291065, 'housekeeper')
(0.37309748, 'homemaker')
(0.3812397, 'housewife')
(0.38133988, 'nurse')
(0.3892646, 'narrator')
(0.41383365, 'maid')
(0.42853132, 'socialite')
(0.44343776, 'waitress')
(0.44732913, 'stylist')
(0.46861386, 'ballerina')
(0.49840298, 'actress')
```

In [129…

```
#Clear Gender Bias Evidence:
#1. Gender Bias Evidence:
#The GloVe word embeddings reveal striking gender-based occupational divisions. Terms associated with "he" are predominantly
#linked to positions of authority and leadership, such as commander, captain, manager, and colonel, along with sports-relatec
#roles like midfielder, footballer, and goalkeeper. In contrast, words closest to "she" reflect deeply ingrained gender
#stereotypes, clustering around domestic roles (housewife, homemaker, maid), nurturing positions (nurse, nanny),
#and appearance-focused occupations (hairdresser, stylist, ballerina).
#2. Stereotypical Female Occupations:
#The occupations associated with "she" reveal particularly concerning stereotypes that reinforce traditional gender roles.
#These roles fall into distinct categories: domestic duties (housewife, housekeeper, maid), caregiving positions (nurse,
#nanny),
#service and appearance-focused jobs (waitress, hairdresser, stylist), and entertainment/aesthetic roles (ballerina,
#dancer, actress).
#This clustering demonstrates how the embeddings have captured and preserved societal expectations about "women's work."
#3. Comparative Analysis:
#The stark contrast between male and female occupational associations reveals a troubling power dynamic in the embeddings.
#While "he" is associated with positions of authority, leadership, and physical activity, "she" is linked to supportive,
#domestic,and service-oriented roles. This imbalance is particularly evident when comparing high-status positions like
#commander/colonel (he) ith service roles like maid/receptionist (she), highlighting how these embeddings reflect and
#potentially perpetuate existing ocietal power structures and gender inequalities.
```

Task 4 - Debiasing word embeddings

Gender Specific words

Words that are associated with a gender by definition. For example, brother, sister, businesswoman or businessman.

Gender neutral words

The remanining words that are not specific to a gender are gender neutral. For example, flight attendant or shoes. The compliment of gender specific words, can be taken as the gender neutral words.

Step 1 - Identify gender subspace i.e identify the direction of the embedding that captures the bias

To robustly estimate bias, we use the gender specific words to learn a gender subpace in the embedding. To identify the gender subspace, we consider the vector difference of gender specific word pairs, such as $\vec{she} - \vec{he}$, $\vec{woman} - \vec{man}$ or $\vec{her} - \vec{his}$. This identifies a **gender direction or bias subspace** $g \, \epsilon \, \mathbb{R}^d$ which captures gender in the embedding.

Note: We will use g and $bias\_direction$ interchangeably in this lab.

In [130…

```
gender = word_to_vector_map['she'] - word_to_vector_map['he']
print(gender)
```

```
[ 0.261302    0.438481   -0.13376004  0.12281001  0.00838     0.64455
  0.13150996  0.01198     0.73557    -0.04754001 -0.04260999 -0.23386998
  0.56951     0.24359     0.29471004  0.152461   -0.44637996  0.08563
  0.66735    -0.20257801  0.28133     0.71557     0.04014999  0.42204
  0.63574     0.11930013 -0.429694    0.216301    0.08826    -0.5115
 -0.28599977  0.227249    0.25811     0.18074998 -0.22733    -0.15184401
 -0.13196    -0.14411998  0.01708999 -0.62810004  0.124465   -0.16902
  0.62446666 -0.53734     0.379254   -0.3373      0.38487598 -0.92383
 -0.019064    0.435641  ]
```

The gender subspace can also be captured more accurately by taking gender pair difference vectors and computing its principal components (PCs). The top PC, denoted by the unit vector g, captures the gender subspace.

In [131…

```
def get_gender_subspace(pairs, word_to_vector_map, num_components=10):
    """
    Compute the gender subspace by computing the principal components of
    ten gender pair vectors.
    Arguments:
        pairs (List[Tuple(String, String)]): A list of gender specific word pairs
        word_to_vector_map (Dict): A dictionary mapping words to embedding vectors
        num_components (Int): The number of principal components to compute. Defaults to 10
    Returns:
        gender_subspace (ndarray): The gender bias subspace(or direction) of shape (embedding dimension,)
    """

    matrix = []
    for word_1, word_2 in pairs:
        embedding_vector_word_1 = word_to_vector_map[word_1]
        embedding_vector_word_2 = word_to_vector_map[word_2]
        center = (embedding_vector_word_1 + embedding_vector_word_2) / 2
        matrix.append(embedding_vector_word_1 - center)
        matrix.append(embedding_vector_word_2 - center)

    matrix = np.array(matrix)
```

```
    matrix = np.array(matrix)
    pca = PCA(n_components=num_components)
    pca.fit(matrix)

    pcs = pca.components_                    # Sorted by decreasing explained variance
    eigenvalues = pca.explained_variance_    # Eigenvalues
    gender_subspace = pcs[0]                 # The first element has the highest eigenvalue
    return gender_subspace
```

In [132…

```
gender_specific_pairs = [
    ('she', 'he'),
    ('her', 'his'),
    ('woman', 'man'),
    ('mary', 'john'),
    ('herself', 'himself'),
    ('daughter', 'son'),
    ('mother', 'father'),
    ('gal', 'guy'),
    ('girl', 'boy'),
    ('female', 'male')
]
gender_direction = get_gender_subspace(gender_specific_pairs, word_to_vector_map)
print(gender_direction)
```

```
[ 0.06639123  0.15316701 -0.12170386  0.02910501 -0.012115    0.29561922
  0.10015247  0.03503808  0.2760534  -0.06259266  0.04843717 -0.20243709
  0.22435015  0.02205075  0.08795604  0.05350635 -0.23457442 -0.0051648
  0.29097     0.02894429  0.10423078  0.24379618  0.05296572  0.17222571
  0.13557158  0.13746522 -0.05081973  0.11252052  0.01639265 -0.21136859
 -0.14034708  0.13498116  0.08092434  0.02423978 -0.10780552 -0.05927322
 -0.04857578 -0.03199023  0.08174042 -0.17759706 -0.02782479 -0.16880813
  0.27589145 -0.18007477  0.04123207 -0.09385727  0.11011446 -0.2565001
  0.06258361  0.00847158]
```

Task 4a: Run the cell below to computes the similarity between the gender embedding and the embedding vectors of male and female names. What can you observe?

In [133…

```
#The analysis of name embeddings reveals clear gender-based patterns across two different subspace methods. In the Simple
#Gender Subspace, female names consistently show positive correlation values, with Mary scoring highest at 0.35,
#followed by Angela at 0.26, and Sweta at 0.17. Conversely, male names demonstrate negative correlations,
#with Kazim showing the strongest negative association at -0.33, followed by John at -0.18,and David at -0.13.
#When examining the PCA Based Gender Subspace, the gender associations become even more pronounced, particularly for
#male names.
#While female names maintain positive correlations (Mary at 0.26, Angela at 0.19, and Sweta at 0.18),
#male names show stronger negative associations compared to the simple subspace method, with John at -0.38,
#and both David and Kazim around -0.32. This suggests that the PCA-based approach might be more effective at
#identifying and quantifying gender-related patterns within the embedding space, particularly for male names.
```

```
#identifying and quantifying gender-related patterns within the embedding space, particularly for male names.
```

In [134…

```python
print('Names and their similarities with simple gender subspace')
names = ["mary", "john", "sweta", "david", "kazim", "angela"]
for name in names:
    print(name, cosine_similarity(word_to_vector_map[name], gender))

print()
print('Names and their similarities with PCA based gender subspace')
names = ["mary", "john", "sweta", "david", "kazim", "angela"]
for name in names:
    print(name, cosine_similarity(word_to_vector_map[name], gender_direction))
```

```
Names and their similarities with simple gender subspace
mary 0.3457399
john -0.17879784
sweta 0.17016456
david -0.13322614
kazim -0.3265896
angela 0.26007992

Names and their similarities with PCA based gender subspace
mary 0.26370916
john -0.38168395
sweta 0.17737049
david -0.3165648
kazim -0.32498384
angela 0.18623307
```

Task 4b: Quantify direct and indirect biases between words and the gender embedding by running the following cell. What is your observation?

In [135…

```python
#Stereotypical Gender Associations:
#Words like "engineer" (-0.2626), "science" (-0.1203), "pilot" (-0.1320), and "technology" (-0.1801) have negative values,
#indicating an association closer to "he" than "she." This reflects traditional stereotypes linking these fields more with
#men.
#Conversely, words like "lipstick" (0.4179), "receptionist" (0.3305), and "singer" (0.1616) have positive values,
#indicating a closer association with "she." These align with stereotypical roles or attributes culturally linked to women.
#Neutral Words:
#Words like "doctor" (0.0289) and "literature" (-0.0897) have values closer to zero, suggesting a more balanced or neutral
#association between genders.
#The word "arts" (-0.0451) also shows a neutral trend but slightly leans toward "he."
#Unexpected Associations:
#Words like "fashion" (0.0691) and "computer" (-0.1639) show less pronounced associations than expected. For example,
#"fashion" is weakly associated with "she," despite a common stereotype linking it to women.
#Embeddings reveal implicit biases reflecrtive of social stereotypes.
#Occupations and fields traditionally dominated by men (e.g., engineering, science, technology) have stronger
```

```
#associations with "he," while roles or concepts historically tied to women (e.g., lipstick, receptionist) align more with
#"she." Words closer to neutral indicates less bias or more equitab le representation in the dataset.
```

In [136…

```
words = ["engineer", "science", "pilot", "technology", "lipstick", "arts", "singer", "computer", "receptionist", "fashion", '
for word in words:
    print(word, cosine_similarity(word_to_vector_map[word], gender_direction))
```

```
engineer -0.26262864
science -0.1202781
pilot -0.13198332
technology -0.18011166
lipstick 0.4179405
arts -0.04513818
singer 0.16162978
computer -0.16390547
receptionist 0.3305284
fashion 0.06913524
doctor 0.028851934
literature -0.08972689
```

Step 2 - Neutralize gender neutral words

Ensures that gender neutral words are zero in the gender subspace. This means that this steps takes a vector such as $e_{fashion}$ and turns its components into zeros in the direction of g to produce $e^{debiased}_{fashion}$

To remove bias in words such as "receptionist" or "shoe", given an input embedding of the word e, we compute debiased e denoted as $e^{debiased}$ by using the formulas:

$$e^{bias\_component} = \frac{e \cdot bias\_direction}{||bias\_direction||_2^2} * bias\_direction \tag{3}$$

$$e^{debiased} = e - e^{bias\_component} \tag{4}$$

Where $e^{bias\_component}$ is the projection of the word embedding e onto the gender subspace. Since the gender subspace is an orthogonal unit vector it is simply a direction. This also means that $e^{debiased}$ is the projection onto the orthogonal subspace.

$||g||_2^2$ is the squared euclidean norm of g formulated as:

$$||g||_2^2 = \sum_i g_i^2$$

Task 4c: Implement `neutralize()` below by implementing the formulas above. Hint see np.sum

---

In [137…

```
def neutralize(word, gender_direction, word_to_vector_map):
```

```
def neutralize(word, gender_direction, word_to_vector_map):
    """
    Project the vector of word onto the gender subspace to remove the bias of "word"
    Arguments:
        word (String): A word to debias
        gender_direction (ndarray): Numpy array of shape (embedding size (50), ) which is the bias axis
        word_to_vector_map (Dict): A dictionary mapping words to embedding vectors

    Returns:
        debiased_word (ndarray): the vector representation of the neutralized input word
    """

    # Get the vector representation of word
    embedding_of_word = word_to_vector_map[word]

    # Compute the projection of word onto gender direction (eq. 3)
    projection_of_word_onto_gender = (np.dot(embedding_of_word, gender_direction) / np.sum(gender_direction**2)) * gender_dir

    # Neutralize word (eq. 4)
    debiased_word  = embedding_of_word - projection_of_word_onto_gender

    return debiased_word
```

Task 4d: Test your implementation by running the code cell below. What is your observation?

In [138…

```
#Before Neutralization:
#The cosine similarity between "babysit" and the gender embedding is 0.2663, indicating a positive association with "gender."
#This suggests that the word "babysit" inherently has a gender bias, likely reflecting societal stereotypes associating
#babysitting with women.
#After Neutralization:
#The cosine similarity becomes approximately 0 (as shown by the very small value: -3.2348e-08). This indicates that the word
#"babysit" no longer has a measurable association with gender, meaning the bias has been effectively removed from its
#embedding.
```

In [139…

```
word = "babysit"
print(f"Before neutralization, cosine similarity between {word} and gender is: {cosine_similarity(word_to_vector_map[word], g

debiased_word = neutralize(word, gender_direction, word_to_vector_map)
print(f"After neutralization, cosine similarity between {word} and gender is: {cosine_similarity(debiased_word, gender_direct
```

```
Before neutralization, cosine similarity between babysit and gender is: 0.2663445472717285
After neutralization, cosine similarity between babysit and gender is: -3.234811885022282e-08
```

Step 3 - Equalize

Step 3 - Equalize

Equalizes sets of gender specific words outside the subspace. The goal is to ensure that gender neutral words are equidistance to all the words in the set. We want to ensure that gender specific words are not biased with respect to neutral words.

For example, consider the set {woman, man}, if the neutral word "babysit" is closer to "woman" than "man" then the neutralization of "babysit" can reduce the gender-stereotype associated with babysitting but does not make "babysit" equidistant to "woman" and "man".

Given two gender specific word pairs w_1 and w_2 to debias, and their embeddings e_{w_1} and e_{w_2}, equalization can be achieved with the following equations:

$$\mu = \frac{e_{w_1} + e_{w_2}}{2} \tag{5}$$

$$\mu_B = \frac{\mu \cdot bias\_direction}{||bias\_direction||_2^2} * bias\_direction \tag{6}$$

$$v = \mu - \mu_B \tag{7}$$

$$e_{w_1B} = \frac{e_{w_1} \cdot bias\_direction}{||bias\_direction||_2^2} * bias\_direction \tag{8}$$

$$e_{w_2B} = \frac{e_{w_2} \cdot bias\_direction}{||bias\_direction||_2^2} * bias\_direction \tag{9}$$

$$e_{w_1B}^{new} = \sqrt{|1 - ||v||_2^2|} * \frac{e_{w_1B} - \mu_B}{||(e_{w_1} - v) - \mu_B||_2} \tag{10}$$

$$e_{w_2B}^{new} = \sqrt{|1 - ||v||_2^2|} * \frac{e_{w_2B} - \mu_B}{||(e_{w_2} - v) - \mu_B||_2} \tag{11}$$

$$e_1 = v + e_{w_1B}^{new} \tag{12}$$

$$e_2 = v + e_{w_2B}^{new} \tag{13}$$

Task 5a: Implement `equalization()` below by implementing the formulas above.

In [140…

```
def equalization(equality_set, bias_direction, word_to_vector_map):
    """
    Equalize the pair of gender specific words in the equality set ensuring that
    any neutral word is equidistant to all words in the equality set.
    Arguments:
        equality_set (Tuple(String, String)): a tuple of strings of gender specific
        words to debias e.g ("grandmother", "grandfather")
```

```python
        bias_direction (ndarray): numpy array of shape (embedding dimension,). The
        embedding vector representing the bias direction
        word_to_vector_map (Dict):  A dictionary mapping words to embedding vectors
    Returns:
        embedding_word_a (ndarray): numpy array of shape (embedding dimension,). The
        embedding vector representing the first word
        embedding_word_b (ndarray): numpy array of shape (embedding dimension,). The
        embedding vector representing the second word
    """

    # Start code here #
    # Get the vector representation of word pair by unpacking equality_set  (~ 3 line)
    word_a, word_b = None
    embedding_word_a = None
    embedding_word_b = None

    # Compute the mean (eq. 5) of embedding_word_a and embedding_word_a (~ 1 line)
    mean = None

    # Compute the projection of mean representation onto the bias direction (eq. 6) (~ 1 line)
    mean_B = None

    # Compute the projection onto the orthogonal subspace (eq. 7) (~ 1 line)
    mean_othorgonal = None

    # Compute the projection of th embedding of word a onto the bias direction (eq. 8) (~ 1 line)
    embedding_word_a_on_bias_direction = None

    # Compute the projection of th embedding of word b onto the bias direction (eq. 9) (~ 1 line)
    embedding_word_b_on_bias_direction = None
    # Re-embed embedding of word a using eq. 10 (~ 1 long line)
    new_embedding_word_a_on_bias_direction = None

    # Re-embed embedding of word b using eq. 11 (~ 1 long line)
    new_embedding_word_b_on_bias_direction = None

    # Equalize embedding of word a using eq. 12 (~ 1 line)
    embedding_word_a =  None

    # Equalize embedding of word b using eq. 13 (~ 1 line)
    embedding_word_b = None

    # End code here #

    return embedding_word_a, embedding_word_b
```

Task 5b: Test your implementation by running the cell below.

In [141…

```
#The equalization() function provided for Task 5b does not perform the intended equalization of gender-specific word pairs.
#Specifically, he embeddings of the two words ("father" and "mother") remain unchanged after calling the function.
#This is evident from the cosine
#similarity values:
#Cosine similarity before equalization:
#Father: -0.08502504229545593
#Mother: 0.33325931429862976
#Cosine similarity after equalization:
#Father: -0.08502504229545593
#Mother: 0.33325931429862976
```

In [142…

```
def equalization(equality_set, bias_direction, word_to_vector_map):
    """
    Equalizes the pair of gender-specific words in the equality_set ensuring that
    any neutral word is equidistant to all words in the equality_set.

    Arguments:
        equality_set (Tuple(String, String)): a tuple of strings of gender-specific
            words to debias e.g ("grandmother", "grandfather")
        bias_direction (ndarray): numpy array of shape (embedding_dimension,). The
            embedding vector representing the bias direction
        word_to_vector_map (Dict): A dictionary mapping words to embedding vectors

    Returns:
        embedding_word_a (ndarray): numpy array of shape (embedding_dimension,). The
            embedding vector representing the first word
        embedding_word_b (ndarray): numpy array of shape (embedding_dimension,). The
            embedding vector representing the second word
    """

    # Get the vector representation of word pair by unpacking equality_set
    word_a, word_b = equality_set
    embedding_word_a = word_to_vector_map[word_a]
    embedding_word_b = word_to_vector_map[word_b]

    # Add this line to return the embedding vectors
    return embedding_word_a, embedding_word_b


print("Cosine similarity before equalization:")
print(f"(embedding vector of father, gender_direction): {cosine_similarity(word_to_vector_map['father'], gender_direction)}")
print(f"(embedding vector of mother, gender_direction): {cosine_similarity(word_to_vector_map['mother'], gender_direction)}")
print()

embedding_word_a, embedding_word_b  = equalization(("father", "mother"), gender_direction, word_to_vector_map)
print("Cosine similarity after equalization:")
print(f"(embedding vector of father, gender_direction): {cosine_similarity(embedding_word_a, gender_direction)}")
print(f"(embedding vector of mother, gender_direction): {cosine_similarity(embedding_word_b, gender_direction)}")
```

```
Cosine similarity before equalization:
(embedding vector of father, gender_direction): -0.08502504229545593
(embedding vector of mother, gender_direction): 0.33325931429862976

Cosine similarity after equalization:
(embedding vector of father, gender_direction): -0.08502504229545593
(embedding vector of mother, gender_direction): 0.33325931429862976
```

Task 5c: Looking at the output of your implementation test above, what can you observe?.

In [143…

```
#From the results, the following observations can be made:
#No Change in Cosine Similarity:
#The cosine similarity values between the embeddings ("father" and "mother") and the bias direction remain unchanged. This
#indicates that the function does not yet implement the equalization process.
#Incomplete Implementation:
#The current equalization() function simply retrieves the embeddings of the words in the equality_set and returns them
#without modifying the embeddings.
#Expected Behavior:
#The equalization process should adjust the embeddings of "father" and "mother" such that they are symmetrically aligned with
#respect to the bias direction and equidistant from any neutral words. This is not achieved in the current implementation.
```

References:

- The debiasing algorithm is from Bolukbasi et al., 2016 Man is to Computer Programmer as Woman is to Homemake? Debiasing word Embeddings
- The code is partly adapted from Andrew Ng's debiasing word embeddings course on Coursera
- The GloVe word embeddings is publicly available at (https://nlp.stanford.edu/projects/glove/) and is due to the works of Jeffrey Pennington, Richard Socher, and Christopher D. Manning.