Module 5 - Bias in Word Embeddings

Assignment overview

In this assignment, you will be asked to evaluate gender bias in word embeddings, debias the embeddings in post-processing, and determine the extent to which the bias is still present after debiasing.

The assignment is modeled after "Lipstick on a Pig: Debiasing Methods Cover up Systematic Gender Biases in Word Embeddings But do not Remove Them, by Hila Gonen and Yoav Goldberg (https://arxiv.org/pdf/1903.03862.pdf).

For this assignment, it is possible to work in **groups of up to 2 students**.

Group members

Leave blanks if group has less than 2 members:

- Student 1: Jingyuan Liu (S.N. 69763183)
- Student 2: Nicholas Tam (S.N. 45695970)

Learning Goals:

After completing this assignment, you will be able to:

1. Evaluate the extent of language bias in word embeddings (pre-trained or trained on a new corpus of text)
2. Apply a post-processing approach to reduce the presence of stereotypes in word embeddings; recognize and explain their limitations
3. Evaluate the presence of bias in word embeddings after the application of de-biasing strategies

1. Introduction to NLP and Word Embeddings

Natural language processing, or NLP for short, is the study of how to use computers in order to process, analyze, and produce language similar to how a human would. You've come across NLP if you've used:

- Google Translate
- Speech to text tools
- And of course, ChatGPT...

The question is: how do we go from words in a language to something a computer can understand? One of the answers is word embeddings. Word embeddings are a way to represent a word using a vector of numbers, in a way such that similar words will have vectors that are closer to each other. For example, when we convert the word "bad" into a word embedding, we would expect it to be close to the embedding for "worst", but maybe not as close to the word "rainbow".

No description has been provided for this image

There are many different pre-trained word embeddings out there, trained on different kinds of data. In this module we will use pre-trained embeddings (no need to create our own). Before we do that, let's get some more familiarity with vectors and their operations...

Vectors and Vector Arithmetic

To better understand word embeddings, we first need to know how these words are represented - vectors - and what operations can be applied to them. A vector is a type of data that has a magnitude and direction. Vectors can exist in spaces of any dimension. For example, $\vec{[1, 2]}$ is a vector in 2D space, and $\vec{[4, 1, 7]}$ is a vector in 3D space. Let's work in 2D space, to make it easier to visualize.

Suppose we have two vectors:

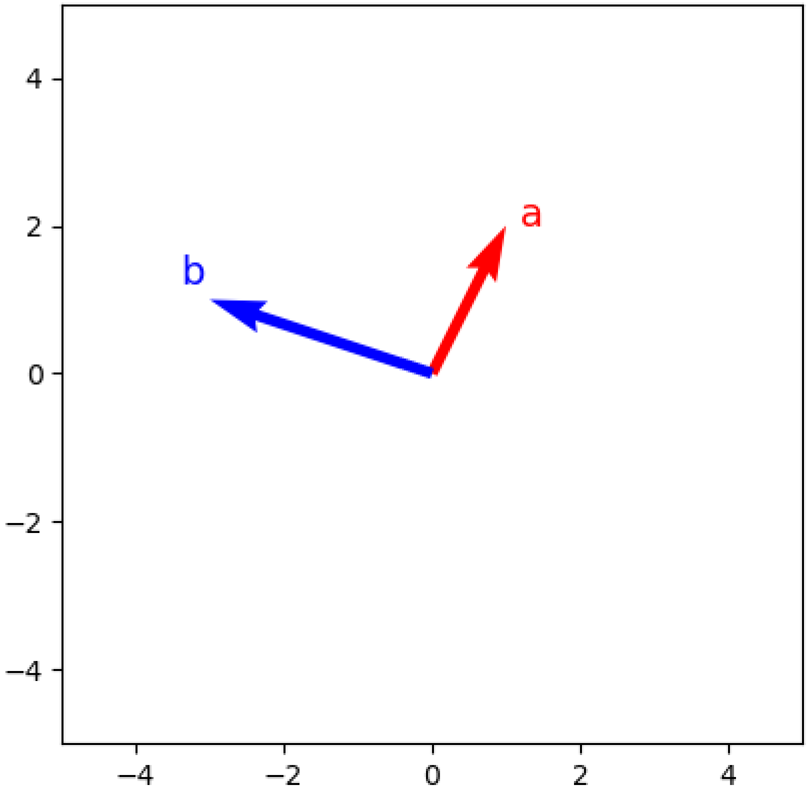
$a = \vec{[1, 2]}$

$b = \vec{[-3, 1]}$

Let's start by plotting $a$ in red and $b$ in blue.

In [1]:

```python
import numpy as np
import matplotlib.pyplot as plt

a = np.array([1, 2]) # vector a
b = np.array([-3, 1]) # vector b
origin = np.array([[0, 0],[0, 0]]) # origin point

# plotting
V = np.array([a, b])
plt.quiver(*origin, V[:,0], V[:,1], color=['r','b'], angles='xy', scale_units='x
plt.text(V[0,0] + 0.2, V[0,1], 'a', fontsize=14, color='red')
plt.text(V[1,0] - 0.4, V[1,1] + 0.2, 'b', fontsize=14, color='blue')
plt.xlim((-5,5))
plt.ylim((-5,5))
ax = plt.gca()
ax.set_aspect('equal', adjustable='box')
plt.show()
```

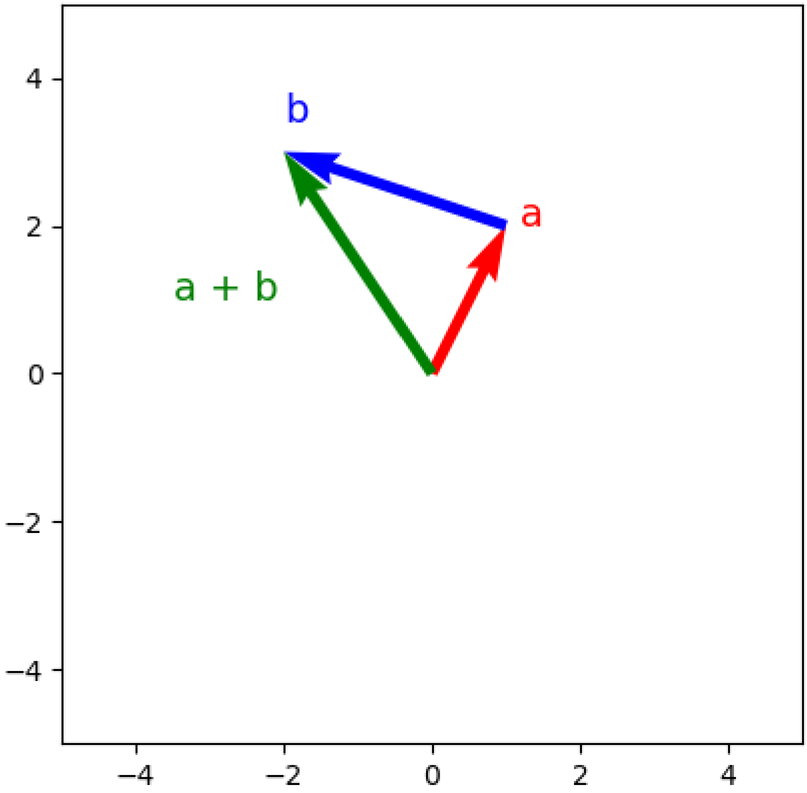


Now, we will try adding $a$ and $b$. To do this, all we have to do is sum the first coordinate of $a$ with the first coordinate of $b$, and the second coordinate of $a$ with the second coordinate of $b$.

We get $1 + (-3) = -2$ for the first coordinate, and $2 + 1 = 3$ for the second coordinate. So, $a + b = \vec{[-2, 3]}$.

But this has more than just an abstract meaning. It makes sense when you think about it in terms of geometry too. In geometrical terms, going in the direction of $a + b$ means going in the direction of $a$ for $a$'s magnitude, then the direction of $b$ for $b$'s magnitude. Let's see what it looks like, in green.

In [2]:
```
origin = np.array([[0, 1, 0],[0, 2, 0]]) # origin point; for b, the origin is no
                                         # tip of a (1,2)


V = np.array([a, b, a + b])
plt.quiver(*origin, V[:,0], V[:,1], color=['r','b','g'], angles='xy', scale_unit
plt.text(V[0,0] + 0.2, V[0,1], 'a', fontsize=14, color='red')
plt.text(V[1,0] + 1, V[1,1] + 2.4, 'b', fontsize=14, color='blue')
plt.text(V[2,0] - 1.5, V[2,1] - 2, 'a + b', fontsize=14, color='green')
plt.xlim((-5,5))
plt.ylim((-5,5))
ax = plt.gca()
ax.set_aspect('equal', adjustable='box')
plt.show()
```



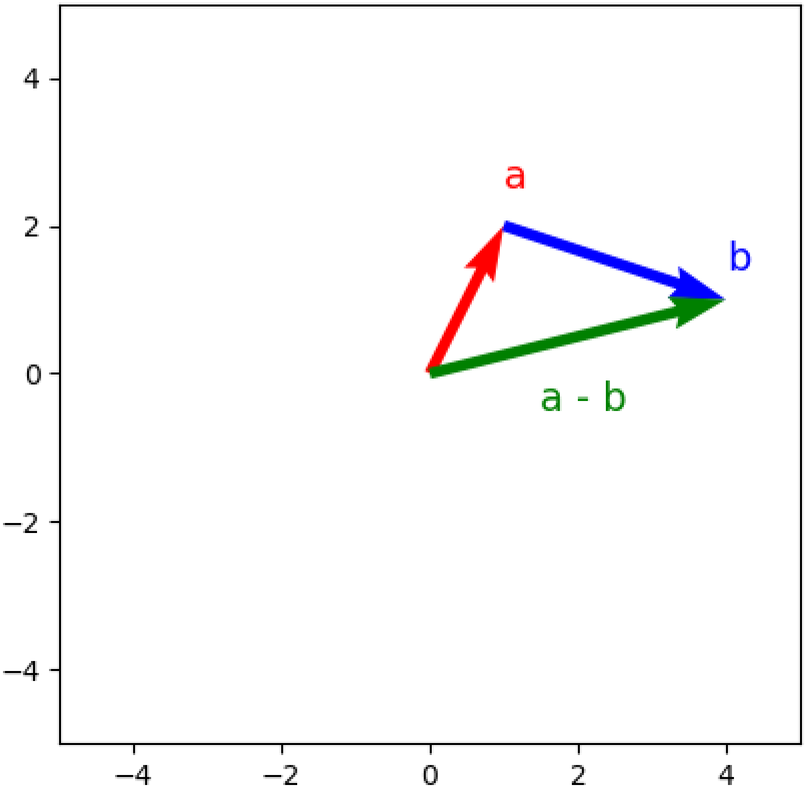
Now that we have a taste for vector addition, let's try some vector subtraction. To find $a - b$, we subtract the first coordinate of $b$ from the first coordinate of $a$, and the second coordinate of $b$ from the second coordinate of $a$.

We get $1 - (-3) = 4$ for the first coordinate, and $2 - 1 = 1$ for the second coordinate. So $a - b = \vec{[4, 1]}$.

In geometrical terms, going in the direction of $a - b$ means going in the direction of $a$ for $a$'s magnitude, then the OPPOSITE direction of $b$ for $b$'s magnitude. Let's see what it looks like, in green.

In [3]:

```
origin = np.array([[0, 1, 0],[0, 2, 0]]) # origin point; again, for b, the origi
                                          # tip of a (1,2)

V = np.array([a, -b, a - b]) # -b means we inverted its direction
plt.quiver(*origin, V[:,0], V[:,1], color=['r','b', 'g'], angles='xy', scale_uni
plt.text(V[0,0], V[0,1] + 0.5, 'a', fontsize=14, color='red')
plt.text(V[1,0] + 1, V[1,1] + 2.4, 'b', fontsize=14, color='blue')
plt.text(V[2,0]- 2.5, V[2,1] - 1.5, 'a - b', fontsize=14, color='green')
plt.xlim((-5,5))
plt.ylim((-5,5))
ax = plt.gca()
ax.set_aspect('equal', adjustable='box')
plt.show()
```



Finally, we'll talk about the dot product. The dot product of two vectors is a measure for how similar of a direction they are pointing in. The dot product of a with b shows us the amount that a is pointing in the same direction as b. To calculate the dot product, multiply each coordinate of a with its corresponding coordinate of b, then add everything together.

$a \cdot b = 1(-3) + 2(1) = -1$

In geometrical terms, if the dot product is a **positive number**, the vectors form an acute angle with one another (they are pointing in **similar directions**). If the dot product is a **negative number**, the vectors form an obtuse angle with one another (they are pointing in **opposite directions**). And if the dot product is 0, the vectors form a right angle with one another (they are pointing in perpendicular directions). So it makes sense that the dot product of a and b is negative because the angle between them is obtuse.

Question 1

Your turn! Complete the following operations on these vectors, which now have three dimensions.

$a = \vec{[3, 1, 2]}$

$b = \vec{[-1, 2, 1]}$

- What is $a + b$?
- What is $a - b$?

- What is $a \cdot b$? Are they pointing in the same or in the opposite direction?

- ~~$a + b = \vec{[3, 1, 2]} + \vec{[-1, 2, 1]} = \vec{[2, 3, 3]}$~~
- ~~$a - b = \vec{[3, 1, 2]} - \vec{[-1, 2, 1]} = \vec{[4, -1, 1]}$~~
- ~~$a \cdot b = \vec{[3, 1, 2]} \cdot \vec{[-1, 2, 1]} = (-3) + 2 + 2 = 1$~~

Refined answer:

- $a + b = \vec{[3, 1, 2]} + \vec{[-1, 2, 1]} = \vec{[2, 3, 3]}$
- $a - b = \vec{[3, 1, 2]} - \vec{[-1, 2, 1]} = \vec{[4, -1, 1]}$
- $a \cdot b = \vec{[3, 1, 2]} \cdot \vec{[-1, 2, 1]} = (-3) + 2 + 2 = 1$; they are pointing in the same direction because the dot product is positive.

Load GloVe Embeddings

We are now ready to start working with some word embeddings. As mentioned before, we will work with pre-trained model. For this portion of the exercise, we will work with GloVe, pre-trained on Wikipedia.

In [4]:
```
# You will need to run this cell once to install the new package. After that, co
# !pip install gensim
```

In [5]:
```
import gensim
from gensim import models
import gensim.downloader as api
```

In [6]:
```
# This may take a couple minutes to load.
model = api.load("glove-wiki-gigaword-100")

# Disclaimer: glove-wiki-gigaword-100 is a pre-trained model of 400000 records,
# much smaller than models used for real applications, and as such it will not a
# would expect.
```

Similarities and Analogies

In [7]:
```
# A list of words - you can try others if you'd like!
lst = ['librarian', 'cash', 'president', 'exercise', 'dog']
```

Look how easy it is to find similar words using word embeddings - all we have to do is look for the next nearest vectors! And the GloVe API has a function to do this - let's try it out:

In [8]:
```
for word in lst:
    similar_words = model.most_similar(word) # getting the most similar word usi
                                             # by default, it returns the 10 clo
    similar_words = [w[0] for w in similar_words]
    print(f"Similar words for '{word}': {', '.join(similar_words)}")
```

```
Similar words for 'librarian': archivist, lecturer, clerk, tutor, curator, educat
or, adjunct, teacher, library, chaplain
Similar words for 'cash': money, funds, payment, credit, payments, loans, pay, am
ount, buy, debt
Similar words for 'president': vice, presidency, former, presidents, chairman, se
cretary, clinton, leader, government, met
Similar words for 'exercise': exercises, routine, training, practice, preparatio
n, necessary, activities, exercising, regimen, take
Similar words for 'dog': cat, dogs, pet, puppy, horse, animal, pig, boy, cats, ra
bbit
```

Another great property of word embeddings is that analogies between words is represented as a simple vector operation. Look at these examples (source):

No description has been provided for this image

Look at the plot on the left: the difference "man/woman" can be thought of as the distance between two vectors, and it should be the same for each pair of gendered words (man/woman, king/queen, actor/actress...). If I can compute this distance from a pair (for example, from the man/woman pair), I can apply it to another word ("king") and get to the other word in the pair ("queen"). This is true for any relationship between words (in the picture, gerund and past tense of a verb, and country and capital).

Question 2

Given the vectors for the words $\vec{KING}$, $\vec{MAN}$ and $\vec{WOMAN}$, what operation (addition, subtraction, or a combination of) will lead me to the word $\vec{QUEEN}$?

$ \vec{QUEEN}=\vec{KING} + (\vec{WOMAN} - \vec{MAN})$

We will now play around with analogies. Analogies are patterns of the form:

- Japan is to as Tokyo as India is to Delhi
- hand is to finger as foot is to toe

Word embeddings can help us "fill in the blank" for analogies, given just three of the words. The same GloVe function `most_similar()` can be given different parameters to compute analogies (see documentation). Knowing what we know so far about vectors and analogies, complete the definition of the `analogy()` function below:

In [9]:

```python
def analogy(worda, wordb, wordc):
    result = model.most_similar(negative=[worda],
                                positive=[wordc,wordb])
    return result[0][0]

# Uncomment when ready to try your function
analogy('japan', 'tokyo', 'india')
```

Out[9]:

```
'delhi'
```

What about:

- man is to doctor as woman is to BLANK?

See for yourself:

In [10]:

```
# Use analogy() on this set of words

analogy('man', 'doctor', 'woman')
```

Out[10]:

```
'nurse'
```

You just got a taste of bias in word embeddings...

Quantifying Bias in Word Embeddings

Bias in word embeddings gets introduced through bias in the datasets the word embeddings are trained on, and this bias can go on to affect applications that the embeddings are used in. In this assignment, we are going to focus on gendered bias specifically. As our goal is to reduce the bias, we need a way to "measure" it, so we will now if our efforts have been successful or not. But how do we quantify gendered bias in word embeddings?

There is not a clearcut answer to this, but Bolukbasi et al. in their 2016 paper on debiasing word embeddings recommend calculating the projection of each of the embeddings onto the "gender direction", which is the vector difference of the embedding for the word "he" and the embedding for the word "she". The greater the value of the projection, the more biased the embedding is. Negative projections indicate words more associated with women and positive projections indicate words more associated with men.

The picture below represents this concept (although the original he/she vectors are used, instead of their difference): if we look at other words in relation to the vectors "he" and "she", we will find that some are closer to "he", and therefore their projection along this vector is bigger ("doctor", in this example), while the opposite is true for female-gendered words (like "nurse"). Neutral words should have similar projections among both vectors.

No description has been provided for this image

A note on Sex and Gender

Sex and Gender have different meanings, despite often being used interchangeably. In humans, sex refers to a set of biological features such as chromosomes and gene expression. It is usually characterized as male or females, although intersex attributes are also possible. Gender refers to socially constructed roles, behaviours and identities, such as man, woman, or gender diverse. In this notebook, we use "man/woman" or "male-gendered/female-gendered" to emphasize the fact that biased word are more often associated with one of these genders than a biological sex.

We will use another package for this portion of the exercise, called Natural Language Toolkit (NLTK) - sorry, word embeddings are complex and some packages are better at some tasks than others...

In [11]:
```python
import numpy as np
import nltk
nltk.download('words')
words = set(nltk.corpus.words.words())
```

```
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\lcm57\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
```

The code in the cell below does exactly what suggested by Bolukbasi - it takes the difference between the vectors "he" and "she" and computes the dot product of this vector with each word in the corpus, thus giving us a measure of how biased is that word.

Note: the original paper includes the normalization of all vectors so that the magnitude of a vector would not influence the measure of bias associated with it. It is, however, a rather lengthy process and, even though it would produce more accurate results, it is not strictly necessary for the purpose of this assignment. For us, it is sufficient to know that the projection of all vectors will have the right sign.

In [12]:
```python
# For each word, we will store the projection in this dictionary
projections = {}

# He-she direction
he_she_direction = model['he'] - model['she']

# Dot product between every word and the he-she direction
for word in model.key_to_index:
    if word in words:
        proj_val = np.dot(model[word], he_she_direction)  # projection calculate
        projections[word] = proj_val
```

Question 3

In `projections`, look up the word 'student'. What is its projection value? Does this indicate bias toward man or toward woman?

In [13]:
```python
projections["student"] # -0.56518286
# projections.get("student")
```

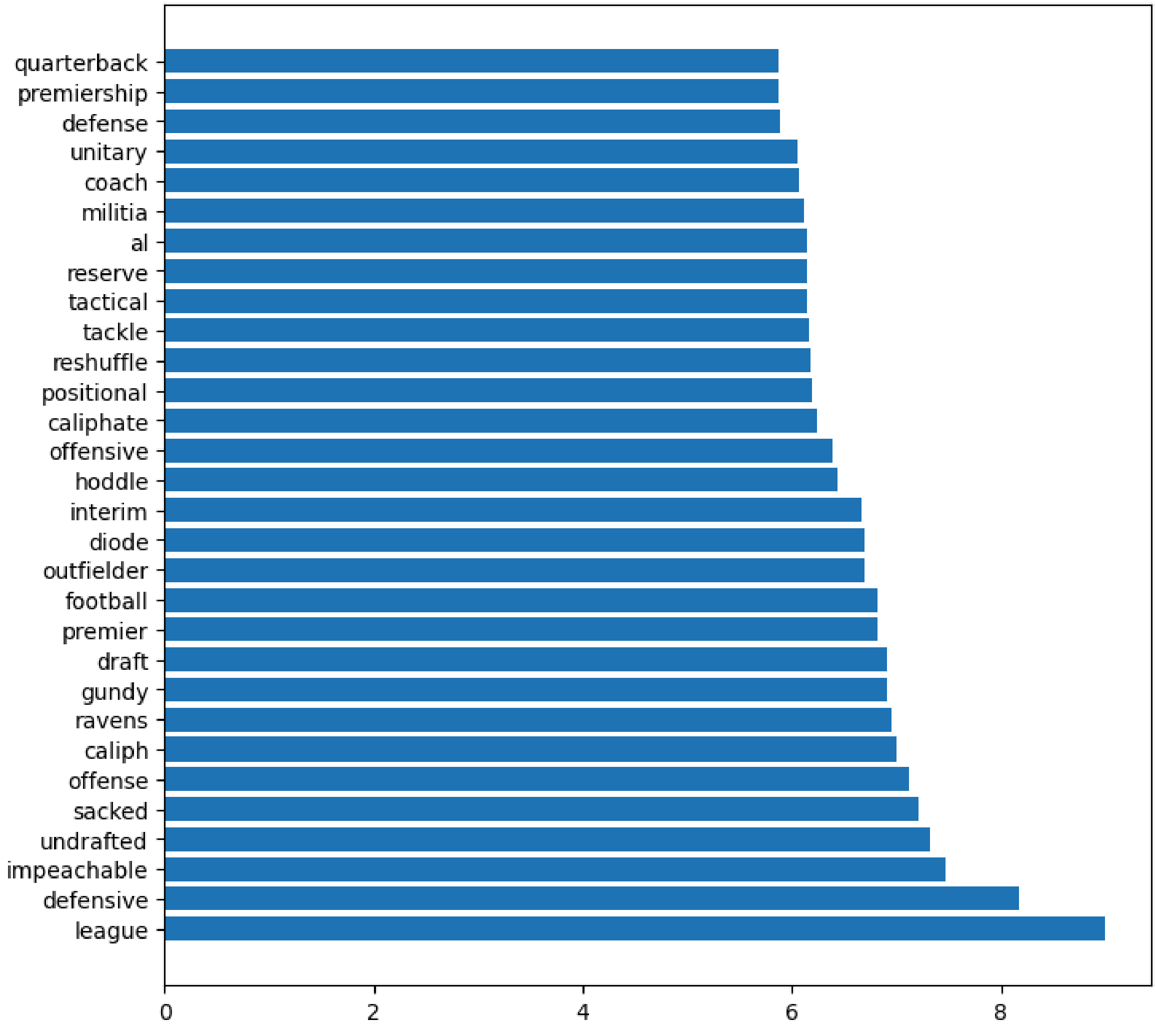
Out[13]:
```
-0.56518286
```

The projection value -0.56518286, which is negative, indicating bias towards women.

Here are the top 30 biased words for each gender, according to the method we described above.

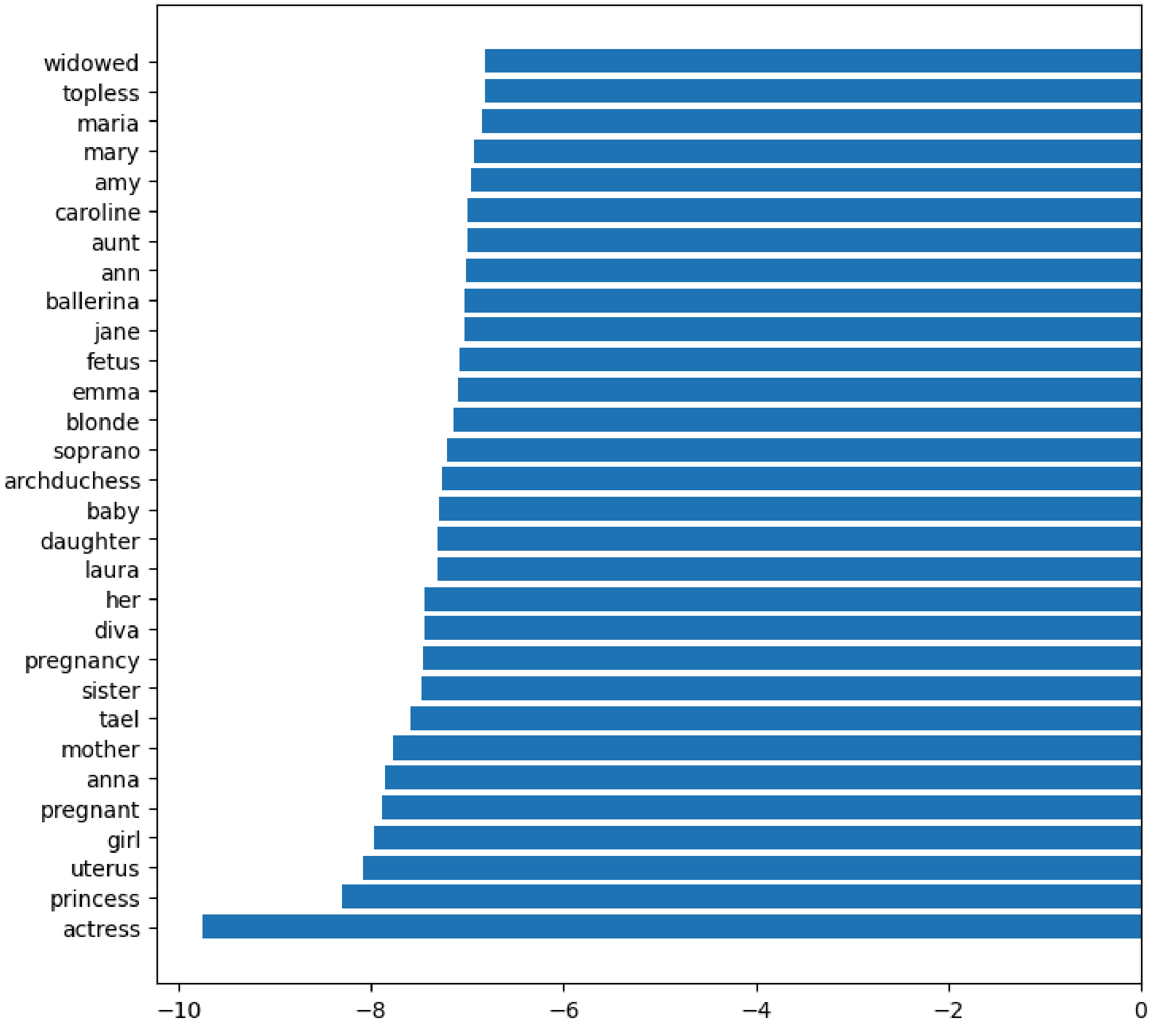
In [14]:

```
# Biased toward man

import matplotlib.pyplot as plt
proj_dict = dict(sorted(projections.items(), key=lambda x:x[1], reverse=True)[:3
plt.figure(figsize=(8,8))
plt.barh(list(proj_dict.keys()), list(proj_dict.values()))
plt.show()
```



In [15]:

```
# Biased toward woman

proj_dict = dict(sorted(projections.items(), key=lambda x:-x[1], reverse=True)[:
plt.figure(figsize=(8,8))
plt.barh(list(proj_dict.keys()), list(proj_dict.values()))
plt.show()
```



Question 4

Look carefully at the lists of words. Which words, if any, do you think should remain associated with a specific gender? Which words should not be?

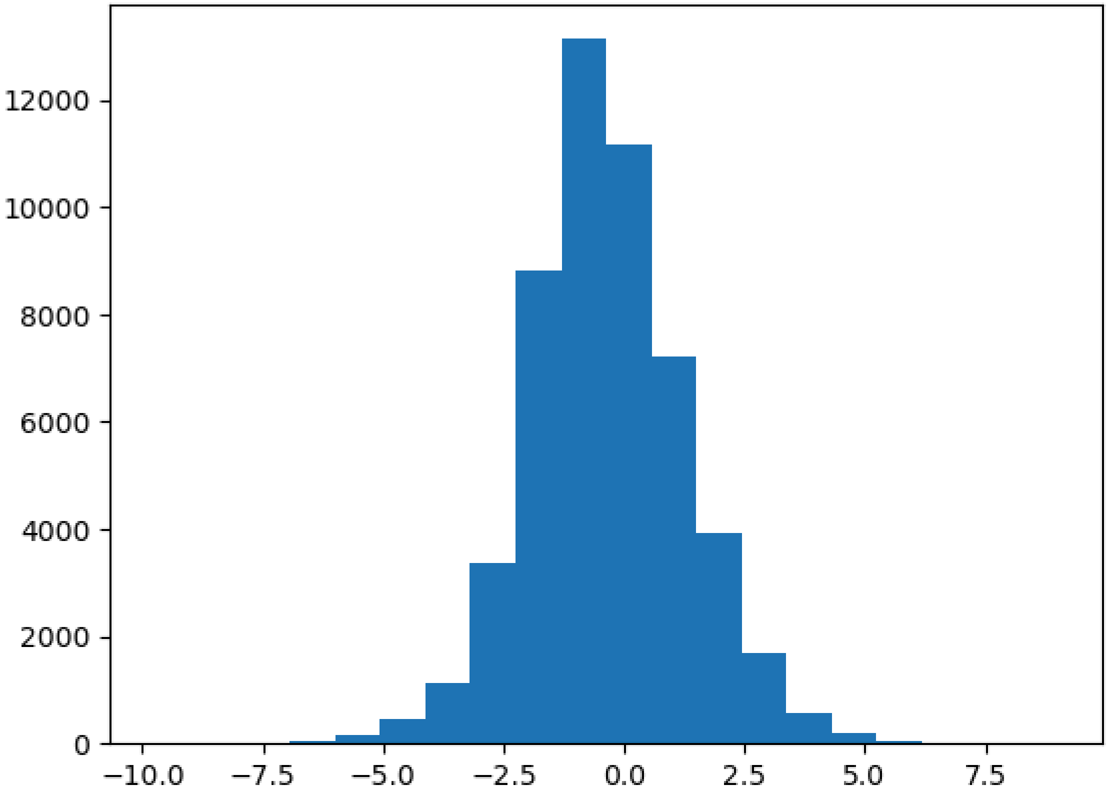
The words that are related to gendered names (e.g. "maria", "mary", "amy", "caroline" for women), or are explicitly referring to terms exclusive to certain genders (e.g. "caliph", "ballerina", "archduchess", "aunt", "her") should remain associated with specific genders. Gender-neutral terminology involving occupations and appearance should not be associated with specific genders (e.g. "blonde", "coach", "tactical", "ravens", "topless"). There are extremely few words among the top 30 words biased towards men that should remain associated with men, as the majority of the terms are sports, government and military terminology. The majority of the top 30 words biased towards women are terms that should remain associated with women.

Question 5

We can look at the distribution of projections as a histogram, to get a sense of the typical values. What gender are most words bias toward? If you think the histogram is not sufficiently clear, you may also compute the average bias in the projections dictionary.

In [16]:

```
plt.hist(projections.values(), bins=20)
plt.show()
```



In [17]:

```
# compute the average bias in the projections dictionary
np.mean(list(projections.values()))
```

Out[17]:

```
-0.36600614
```

Most words are biased toward female-gendered, as the histogram is slightly skewed toward the left, and the average bias in the projections dictionary is negative.

Question 6

Let's look at the most similar words to some words that appeared as highly biased (*ballerina, amy,* and *league*). Do you notice any word that surprises you, or that you think should not be there?

In [18]:

```
for word in ['ballerina', 'amy', 'league']:
    similar_words = model.most_similar(word)
    similar_words = [w[0] for w in similar_words]
    print(f"Similar words for '{word}': {', '.join(similar_words)}")
```

```
Similar words for 'ballerina': dancer, ballerinas, ballet, bolshoi, pavlova, chor
eographer, ingenue, actresses, courtesan, dancers
Similar words for 'amy': jennifer, lisa, susan, alice, julie, judy, linda, jill,
rachel, laura
Similar words for 'league': football, club, leagues, teams, team, season, soccer,
baseball, seasons, champions
```

The words "ingenue" and "courtesan" should not be presented as similar to "ballerina", at least in part due to the connotations associated with the terms.

The next cell assigns a label to the 1000 most biased words in the dictionary: 1 for male-gendered and -1 female-gendered. These labels will be helpful to see if we can effectively remove the bias from the word embedding.

In [19]:

```
proj_dict = dict(sorted(projections.items(), key=lambda x:abs(x[1]), reverse=Tru
ground_truth_lst = [1 if proj_dict[k] >= 0 else -1 for k, v in proj_dict.items()
ground_truth_dct = {k: 1 if proj_dict[k] >= 0 else -1 for k, v in proj_dict.item
```

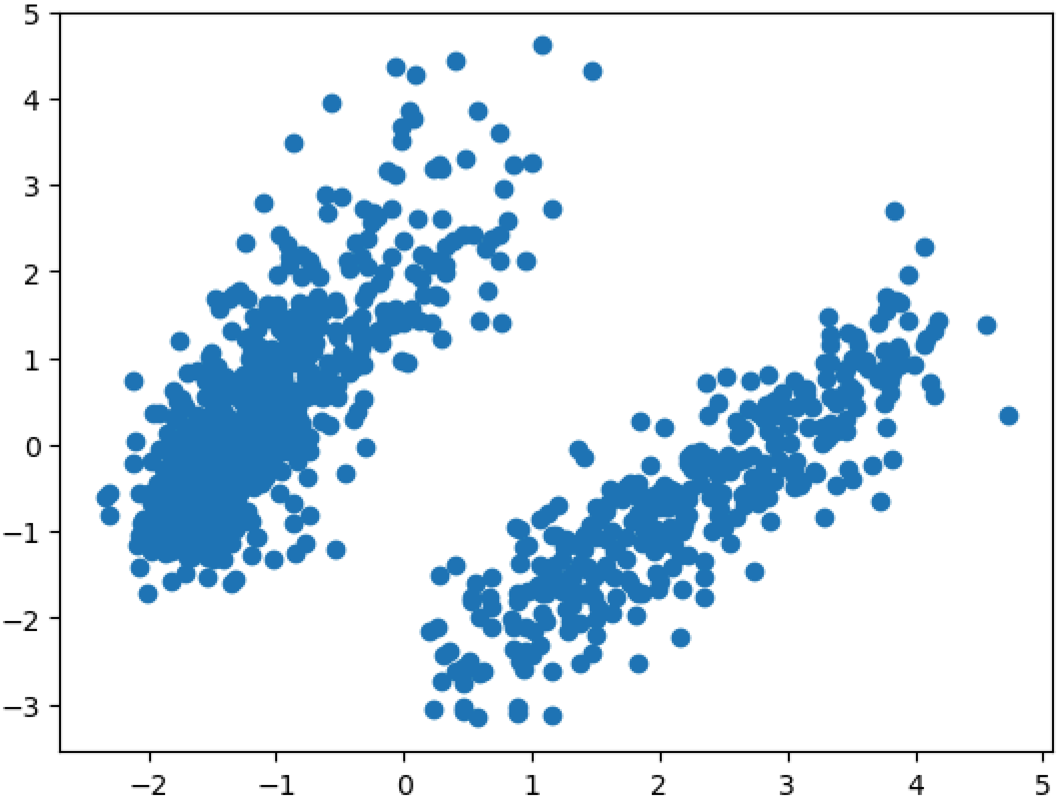
Question 7

We can use Principal Component Analysis to visualize the distributions of the 1000 most biased words in a two-dimensional space.

The code below produces this plot. Do you think it would be easy for a clustering algorithm to separate the words into two clusters? Do you think these clusters correspond to the two groups of gendered words we are analyzing?

In [20]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
df = pca.fit_transform([model[i] for i in proj_dict])

plt.scatter(df[:, 0] , df[:, 1])
plt.show()
```



~~It would be reasonably easy for a clustering algorithm to separate the groups into two separate clusters, and yes, these clusters should correspond to the two~~

~~groups of gendered words we are analyzing. However, as it is, it is unclear which group (cluster) corresponds to which gender because the labels are not given.~~

Refined answer:

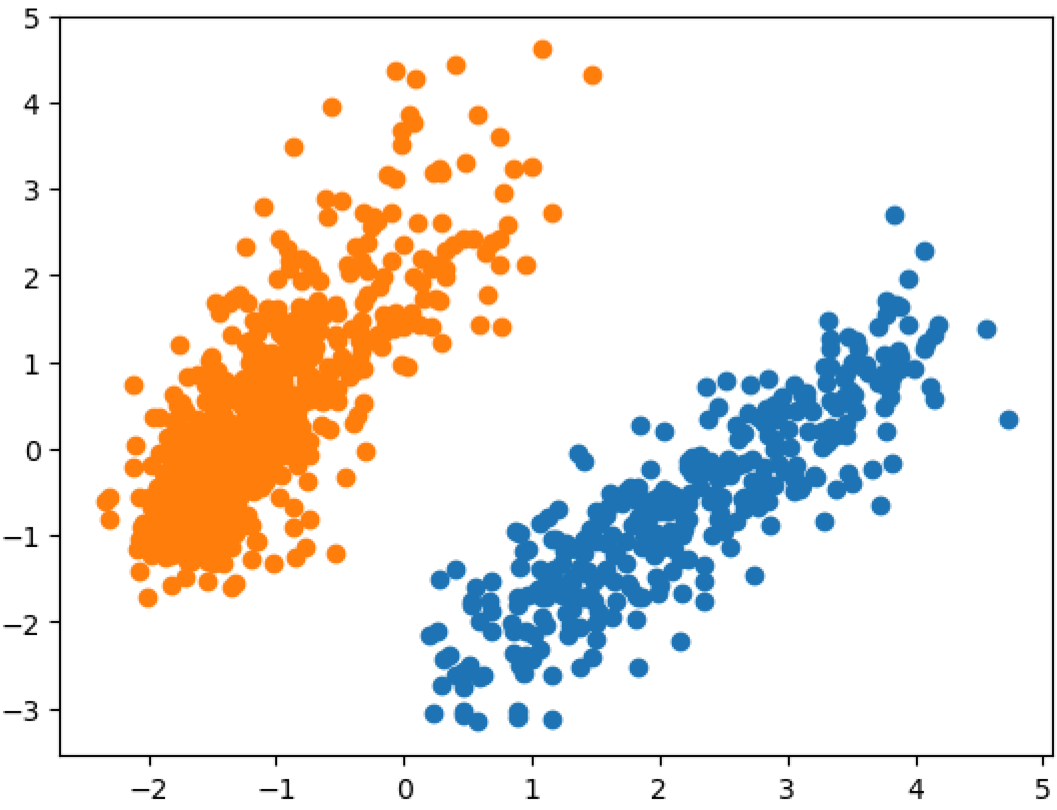
It would be reasonably easy for a clustering algorithm to separate the words into two clusters because the two groups of points are quite distinguishable and separate on the plot. However, these clusters may or may not correspond to the two groups of gendered words we are analyzing; at least we are unsure about it **at this stage**. At this point, we can only tell that the two distinct groups of words are separated by some features, but the feature may or may not be "gender". But if we continue the analysis in Question 8, we notice that the clusters from K-Means perfectly match up with the gender-bias labels we determined from the projections, with an accuracy of 1. Based on the colored plot of clusters below and this accuracy of "1", **now** we can reasonably conclude that these clusters should correspond to the two groups of gendered words we are analyzing.

Running the cell below, we can see that a simple clustering algorithm (K-Means) has no issue separating the two groups, and they correspond with the clusters we could observe after reducing dimensionality.

In [21]:

```python
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2)
label = kmeans.fit_predict([model[i] for i in proj_dict])

for i in [0, 1]:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
plt.show()
```

```
C:\Users\lcm57\miniconda3\envs\dsci430\lib\site-packages\sklearn\cluster\_kmeans.
py:870: FutureWarning: The default value of `n_init` will change from 10 to 'aut
o' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\lcm57\miniconda3\envs\dsci430\lib\site-packages\sklearn\cluster\_kmeans.
py:1382: UserWarning: KMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can avoid it by setting th
e environment variable OMP_NUM_THREADS=4.
  warnings.warn(
```



Question 8

Let's see how well the clusters from K-Means match up with the bias labels we determined from the projections. Comment on the output of this cell. How well was K-Means able to create groups that match the original projections?

In [22]:

```
import statistics as stats

neg_cluster = stats.mode([label[idx] for idx, val in enumerate(ground_truth_lst)
pos_cluster = stats.mode([label[idx] for idx, val in enumerate(ground_truth_lst)

sum([1 for idx, val in enumerate(ground_truth_lst) if (val == 1 and label[idx] =
    or (val == -1 and label[idx] == neg_cluster)])/len(ground_truth_lst)
```

Out[22]:

```
1.0
```

The clusters from K-Means perfectly match up with the bias labels we determined from the projections, with an accuracy of 1. Practically all of the words were labeled correctly. K-Means was able to create groups that matched the original projection bias labels very well.

Neutralize and Equalize

Now that we have a better idea of what it means for a word embedding to be biased, and we also have tools to quantify and visualize that bias, we are going to try to improve it. The code below attempts to mathematically remove the bias using the approach described in Bolukbasi's paper, by removing from the word's vector its projection along the he-she direction. Simply run the cells below to debias the word embedding.

In [23]:
```
# This cell excludes from debiasing words for which being gendered makes sense,
# "actor"/"actress", or "policeman"/"policewoman"

gender_specific_pairs = [["he", "she"], ["his", "hers"], ["him", "her"], ["man",

for i in model.key_to_index:
    if i in words and 'woman' == i[-5:]:
        man_version = i[:-5] + "man"
        if man_version in model.key_to_index:
            gender_specific_pairs.append([man_version, i])
```

In [24]:
```
# This function removes the component of a vector u that is in the direction of
# We will use it to remove the he-she projection from each word in the embedding
def drop(u, v):
    return u - v * u.dot(v) / v.dot(v)

gsw = sum(gender_specific_pairs, [])

# Removing the gender component of words that are not gender-specific from the 1
for w in proj_dict:
    if w not in gsw: # We only want to debias the words that are not gender-spec
        model[w] = drop(model[w], he_she_direction) # Removing the gender compon
        model[w] = model[w]/np.linalg.norm(model[w]) # Normalize the new vector

# Shifting the vectors for words that are gender specific to make sure that
# each non-gender-specific word is equidistant from both words in the gender-spe
for (a, b) in gender_specific_pairs:
    model[a] = model[a]/np.linalg.norm(model[a]) # Normalize vector for a
    model[b] = model[b]/np.linalg.norm(model[b]) # Normalize vector for b
    u = (model[a] + model[b]) / 2 # Find the vector that is exactly in between a
    y = drop(u, he_she_direction) # Remove the component from that vector that i
    z = np.sqrt(1 - np.linalg.norm(y)**2)
    if (model[a] - model[b]).dot(he_she_direction) < 0:
        z = -z
    model[a] = z * he_she_direction + y # Since the non gender specific words ha
    model[b] = -z * he_she_direction + y # the gendered words in relation to the
```

Following our debiasing strategy, all words (minus the excluded gendered words) are now equally distant from the vectors "he and "she". Observe this on the word "coach".

In [25]:
```
np.linalg.norm(model['he'] - model['coach'])
```

Out[25]:
```
1.3031601
```

In [26]:
```
np.linalg.norm(model['she'] - model['coach'])
```

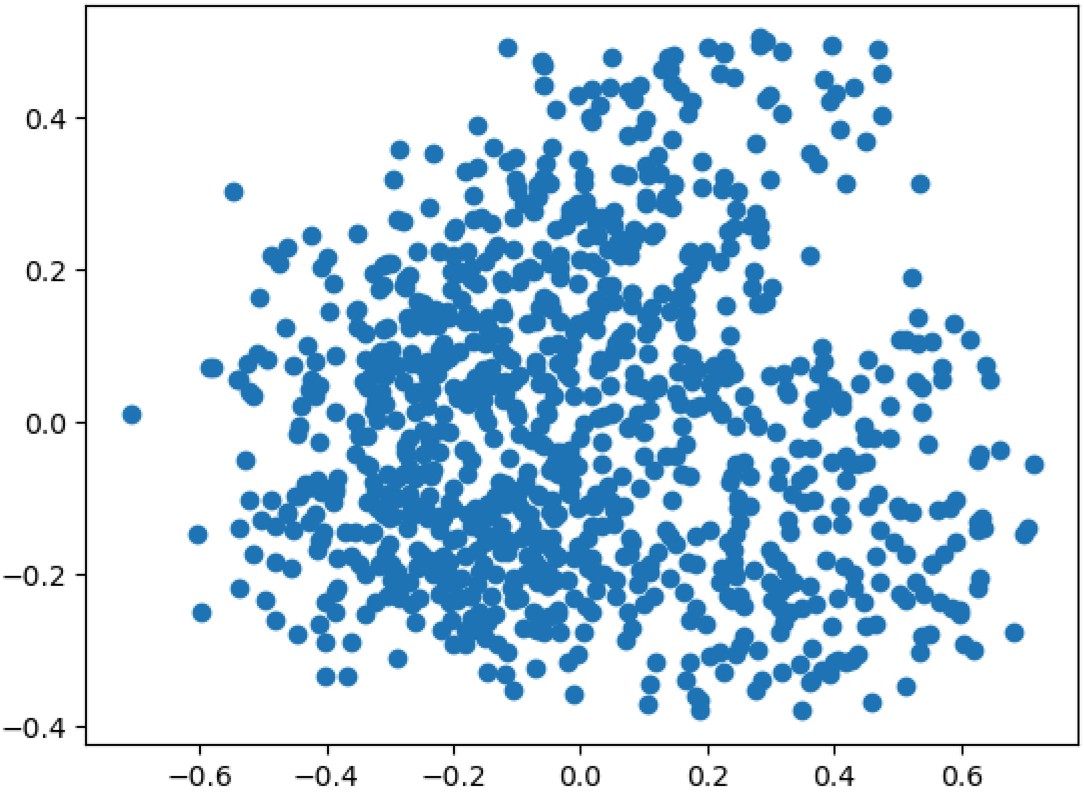
Out[26]:
```
1.3031601
```

Question 9

Let's again use PCA and plot the words after debiasing. Can you separate them into clusters now?

In [27]:

```
pca = PCA(n_components=2)
df = pca.fit_transform([model[i] for i in proj_dict if i not in gsw])

plt.scatter(df[:, 0] , df[:, 1])
plt.show()
```



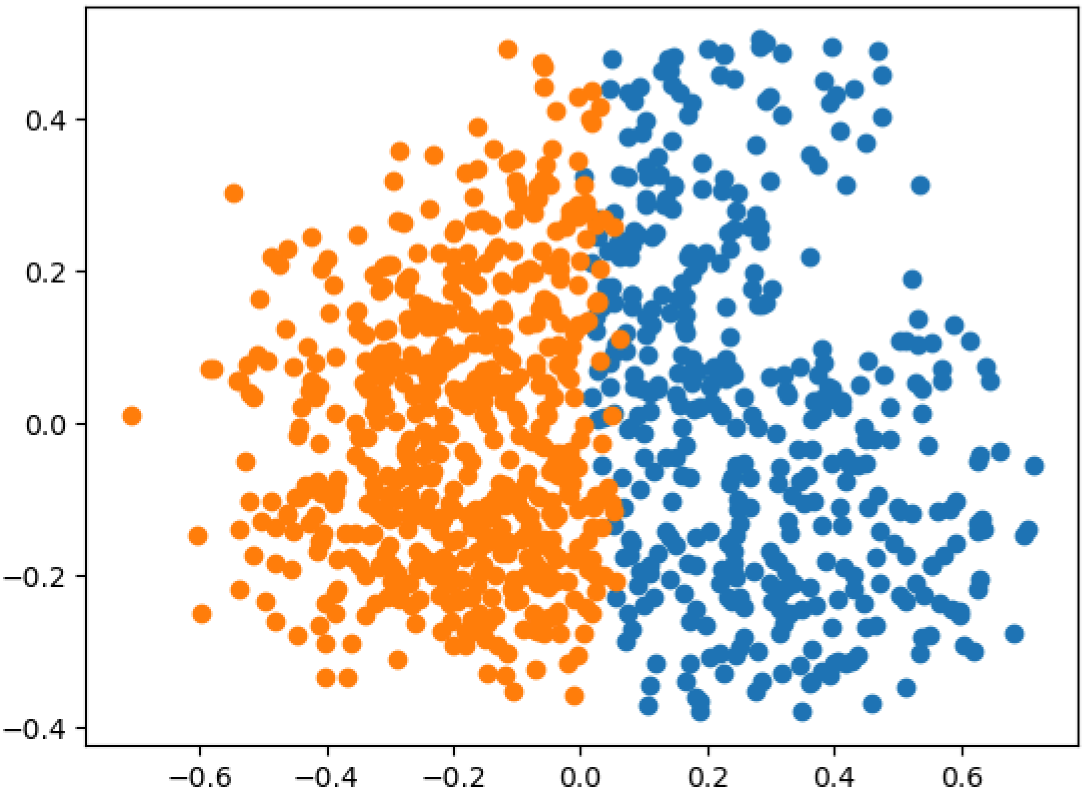
The words are far more difficult to separate into clusters in this form, as the words are sorted in one large collective group.

In the plot below, you can observe K-Means' attempt to separate the words in two clusters (using the original high-dimensional space).

In [28]:

```
kmeans = KMeans(n_clusters=2)
label_debias = kmeans.fit_predict([model[i] for i in proj_dict if i not in gsw])

for i in [0, 1]:
    plt.scatter(df[label_debias == i , 0] , df[label_debias == i , 1] , label =
plt.show()
```

```
C:\Users\lcm57\miniconda3\envs\dsci430\lib\site-packages\sklearn\cluster\_kmeans.
py:870: FutureWarning: The default value of `n_init` will change from 10 to 'aut
o' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\lcm57\miniconda3\envs\dsci430\lib\site-packages\sklearn\cluster\_kmeans.
py:1382: UserWarning: KMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can avoid it by setting th
e environment variable OMP_NUM_THREADS=4.
  warnings.warn(
```



From the plot, it is not clear if K-Means was able to group the words by the original gendered labels. We can see that there are no obvious clusters, but does this mean the bias is gone? Let's check in the next cell...

Question 10

1. Let's see how well the groups found by KMeans overlap with the original gendered labels. What percent of the words are clustered "correctly" according to the bias? What does this mean in terms of success of our debiasing efforts?
2. Based on this results, would you say that looking at the plot of words in 2D is an effective way to estimate the presence of bias in a dataset?

In [32]:
```
ground_truth_not_gsw = [ground_truth_dct[k] for k in ground_truth_dct if k not i

neg_cluster = stats.mode([label_debias[idx] for idx, val in enumerate(ground_tru
pos_cluster = stats.mode([label_debias[idx] for idx, val in enumerate(ground_tru

sum([1 for idx, val in enumerate(ground_truth_not_gsw) if (val == 1 and label_de
     or (val == -1 and label_debias[idx] == neg_cluster)])/len(ground_truth_not_
```

Out[32]:
```
0.7268041237113402
```

1. ~~According to the bias, approximately 72.68041237113402% of the words are correctly clustered with their corresponding gender, a significant decrease from 100%. This implies that the debiasing efforts were relatively successful, as there exist both gender-neutral terms and terms that should remain exclusive to certain genders, and thus the change in clustering accuracy likely~~

~~corresponds to some gender-neutral terms being classified with the opposite gender compared to what they were labeled as before. However, the fact that the clustering accuracy isn't outright 50%, which would indicate equal split of clustering between all labels, indicates that the debiasing isn't perfect.~~

2. ~~Based on the significant decrease in clustering "accuracy" from debiasing, we can claim that the plot of words in 2D is to some extent effective in estimating the presence of bias in a dataset.~~

Refined answer:

1. According to the bias, approximately 72.68041237113402% of the words are correctly clustered with their corresponding gender, a significant decrease from 100%. This implies that the debiasing efforts were relatively successful, as there exist both gender-neutral terms and terms that should remain exclusive to certain genders, and thus the change in clustering accuracy likely corresponds to some gender-neutral terms being classified with the opposite gender compared to what they were labeled as before. However, the fact that the clustering accuracy isn't outright 50%, which would indicate equal split of clustering between all labels, indicates that the debiasing isn't perfect.
2. I would argue that looking at the PCA plot of words in 2D is not a very effective way to estimate the presence of bias in a dataset. I admit that a 2D plot of words may be the best we can do for visualization because we cannot visualize them in a higher dimensional space, but this method is not so effective as it seems to be when estimating the presence of bias in a dataset. For example, from the plot in Question 9, it seems that the two groups of words squash together and become a larger indistinguishable group. Based only on this plot, we may expect that the gender bias is almost gone and the percent of the words clustered "correctly" according to genders should be around 0.5 (i.e. randomly guessing). In fact, the actual clustering "accuracy" is about 0.7269. This is a significant reduction from "perfect matching" but still higher than 0.5, which means some level of gender bias still exists. Therefore, I think a 2D is not a very effective way to estimate the presence of bias in a dataset; despite the plot, we also need to consider the accuracy achieved by the clustering algorithm to agree on the conclusion. Relying only on the 2D plot is not a very effective way to estimate the presence of bias in a dataset.

Question 11

Let's review the similar words to *ballerina*, *amy*, and *league*, which were originally identified as highly biased. Do you see any change? Do you think this is an improvement?

In [30]:

```
for word in ['ballerina', 'amy', 'league']:
    similar_words = model.most_similar(word)
    similar_words = [w[0] for w in similar_words]
    print(f"Similar words for '{word}': {', '.join(similar_words)}")
```

```
Similar words for 'ballerina': bolshoi, ballerinas, choreographer, balanchine, ch
oreographers, prima, nureyev, flamenco, ballets, dancers
Similar words for 'amy': larry, greg, steve, scott, eric, david, michael, tom, an
thony, james
Similar words for 'league': champions, soccer, leagues, season, teams, games, clu
bs, seasons, hockey, championship
```

All of the terms with negative connotations associated with the term "ballerina" have been removed in place of other relevant terms such as "balanchine" and "nureyev", which is a significant improvement from before. The remaining terms "amy" and "league" continue to have similar words with minimal implicit connotations. In particular, we notice that some similar words for "amy" turn from female names into male names, compared to the female names before.

Debiasing during training

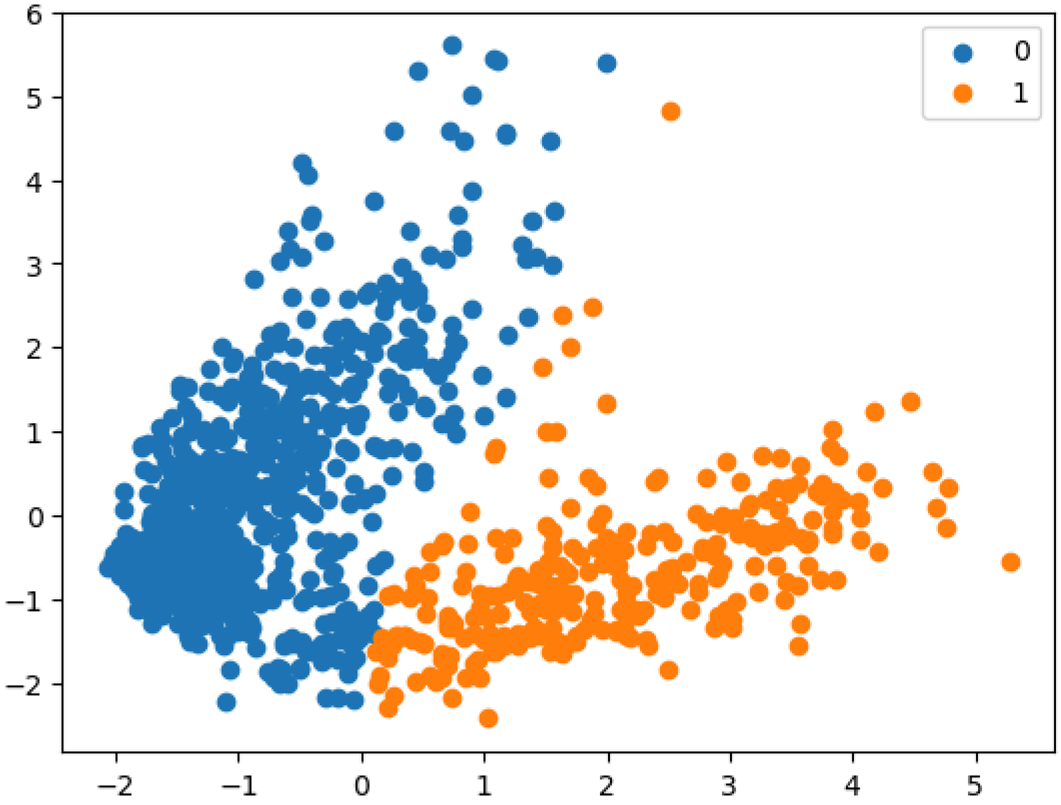
The method that we have worked through is a method to debias embeddings AFTER they have already been created. What if, instead, the training process itself encouraged producing embeddings with less bias? Zhao et al. describe a way to do this with GloVe, **altering the loss function of the model** to focus any gender specific information into the last coordinate of each vector. This way, the resulting vectors can be used without the last coordinate, removing the gender information.

Here, we will load the GloVe embeddings debiased during training, and perform clustering to compare the difference between these and the hard debiased vectors that we produced in the last section.

In [34]:
```python
import numpy as np
import pickle as pkl
f = open('gn_glove.vocab', 'r', encoding="utf-8")
vocab = [i.strip() for i in f.readlines()]
vecs = np.load('gn_glove.wv.npy')
gn_dct = {vocab[i]: vecs[i] for i in range(len(vecs))}
```

In [35]:
```python
kmeans = KMeans(n_clusters=2)
proj_dict_subset = {k: v for k, v in proj_dict.items() if k in gn_dct}
label = kmeans.fit_predict([gn_dct[i] for i in proj_dict_subset])

pca = PCA(n_components=2)
df = pca.fit_transform([gn_dct[i] for i in proj_dict_subset])

for i in [0, 1]:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
plt.legend()
plt.show()
```

```
C:\Users\lcm57\miniconda3\envs\dsci430\lib\site-packages\sklearn\cluster\_kmeans.
py:870: FutureWarning: The default value of `n_init` will change from 10 to 'aut
o' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\lcm57\miniconda3\envs\dsci430\lib\site-packages\sklearn\cluster\_kmeans.
py:1382: UserWarning: KMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can avoid it by setting th
e environment variable OMP_NUM_THREADS=4.
  warnings.warn(
```



In [36]:

```
female_bias = [label[idx] for idx, i in enumerate(proj_dict_subset) if proj_dict
male_bias = [label[idx] for idx, i in enumerate(proj_dict_subset) if proj_dict_s

right_cluster = [1 for i in female_bias if i == stats.mode(female_bias)] + [1 fo
pct_clustered_by_gender = round(100*sum(right_cluster)/len(proj_dict_subset), 2)
print("With GloVe embeddings debiased during training, {} percent of terms are c
```

```
With GloVe embeddings debiased during training, 91.92 percent of terms are cluste
red according to gender.
```

Question 12

Answer the following questions to test your understanding on bias in word embeddings:

1. What are the reasons that word embeddings can be biased? How do you think using different data sets to train embeddings might affect the type or amount of bias?
2. What are some real world applications that might be affected by the remaining bias in the embeddings?

1. Biased word embeddings mainly arise from some biased training data that has some biased languages or historical biases against certain groups. Word embeddings can be biased as a result of learning and preserving the associations and relationships with other words from biased training texts, often by placing commonly co-occuring terms close to each other without regard for context. Using differing data sets for training, either by increasing the variety of texts or by removing texts that are explicitly biased, may help in debiasing the embeddings by shifting the associations between the embedded terms. However, if there are historical biases in most of the data, then using different datasets to train embeddings may not be a very effective way for debiasing.
2. Natural language processing in financial banking may cause certain protected groups (e.g. Black individuals) to be more likely to be falsely marked as committing fraud based on the documentation they have provided. In addition, natural language processing in job applications and resume reviews may cause certain groups (e.g. women) to be more likely to be eliminated based on the resumes and cover letters they provide. The article "Amazon scraps secret AI recruiting tool that showed bias against women" describes an example involving the review of job applicants' resumes.

- Brunet, M.-E., Alkalay-Houlihan, C., Anderson, A., & Zemel, R. (n.d.). Understanding the origins of bias in word embeddings. http://proceedings.mlr.press/v97/brunet19a/brunet19a.pdf
- Dastin, J. (2018, October 10). Insight - Amazon scraps secret AI recruiting tool that showed bias against women. https://www.reuters.com/article/world/insight-amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK0AG/

Bias in Large Language Models (LLMs)

Large Language Models (like ChatGPT) are deep neural networks trained on *very large* corpus of text to be able to produce human-like sentences. Their capabilities are impressive, but since they are typically trained on text found online, they can be rigged with all sorts of biases. Here are a couple of examples (source: https://medium.com/mlearning-ai/inherent-human-bias-in-chat-gpt-ed803d4038fe)

Bias based on nationality:

No description has been provided for this image

Bias based on race and gender:

No description has been provided for this image

Here is another example of bias based on gender when making jokes, which I was able to generate on 2023/11/05:

No description has been provided for this image

Developers behind these language models are constantly trying to eliminate the problem (as you can imagine, biased responses are not well received by the public). As far as we know (which is limited - many of these models are kept in secret) bias is prevented by better data curation, model fine-tuning, and more strict evaluation. In fairness, LLMs are becoming better and better at avoiding biased responses, but the problem still exists in some forms.

Question 13

A team at UC Berkeley developed a tool to allow people to test and evaluate LLMs models. You are going to use this tool to test different language models and "hunt" for persistent biases.

Do the following:

1. Go to https://chat.lmsys.org/?utm_medium=email&utm_source=gamma&utm_campaign=-lmsys-2023
2. Pick one option:
   - Under Arena (side-by-side), pick two models to compare (it can be a random choice) and enter a prompt - remember that your goal is to generate a biased answer. Observe the 2 answers: where you able to find bias in at least one of the models? Is a model more biased than the other?
   - Under Direct Chat, pick a model (it can be a random choice) and enter 2-3 prompts, each time changing the question to include a different population (like we saw in the lightbulb jokes example). Compare the answers: can you observe differences caused by the changes introduced in the question?
3. Report your findings here, **including your prompt, the names and the answers of both models (the names of the models are visible after their answers), and your evaluation.**

Used Direct Chat with model: gpt-4o-mini-2024-07-18

- Prompt 1: In the sentence, "the nurse phoned the doctor because she was late", who was late?
- Answer 1: In the sentence "the nurse phoned the doctor because she was late," the pronoun "she" is ambiguous and could refer to either the nurse or the doctor. Without additional context, it is unclear who was late. If the context were clearer, it might indicate who was late; for example, if the preceding sentences discussed the nurse's schedule, it would suggest that the nurse was late. Conversely, if the focus was on the doctor's appointment, it

could imply that the doctor was late.

- Prompt 2: In the sentence, "the nurse phoned the doctor because he was late", who was late?
- Answer 2: In the sentence "the nurse phoned the doctor because he was late," the pronoun "he" typically refers to the most recently mentioned male person, which in this case would be "the doctor." Therefore, it implies that the doctor was late. So, in this context, **the doctor was late**.

  Given the prompts and their corresponding responses, there are gender biases present in the model. While the first prompt accounts for either the nurse or the doctor being female, the second prompt does not account for the possibility that the "he" referred to in the statement could be the nurse and not the doctor. This implies a smaller but still persistent bias involving genders and occupations.

- Kotek, H., Dockum, R., & Sun, D. (2023, November). Gender bias and stereotypes in Large Language Models. Gender bias and stereotypes in large language models. https://dl.acm.org/doi/fullHtml/10.1145/3582269.3615599

Final thoughts

1. If you have completed this assignment in a group, please write a detailed description of how you divided the work and how you helped each other completing it:

We worked on the assignment separately, each taking turns to answer all parts and modifying the responses down the line.

2. Have you used ChatGPT or a similar Large Language Model (LLM) to complete this homework? Please describe how you used the tool. We will never deduct points for using LLMs for completing homework assignments, but this helps us understand how you are using the tool and advise you in case we believe you are using it incorrectly.

- Jingyuan's response: I did not use ChatGPT much in this particular assignment. Instead, I googled some articles about the real-world applications that might be affected by the remaining bias in the embeddings. Reference is included in Question 12.
- Nicholas' response: I used ChatGPT to help with fixing the file paths for 'gn_glove.vocab' and 'gn_glove.wv.npy'.

3. Have you struggled with some parts (or all) of this homework? Do you have pending questions you would like to ask? Write them down here!

- Jingyuan's response: How to use GloVe embeddings debiased during training? How do we alter the loss function of the model to focus any gender-specific information on the last coordinate of each vector?

- Nicholas' response: How exactly do GloVe embeddings focus the gender-specific information to the last coordinate of each vector? Can this form of debiasing be applied to other forms of bias, such as race?

In [ ]:

```
```