Assignment 1

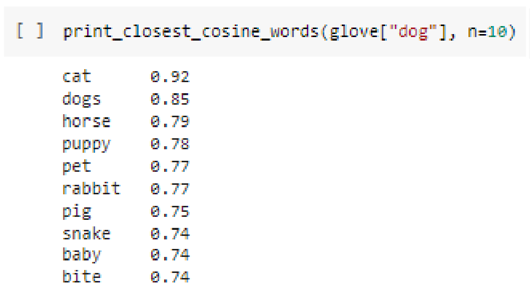
Hruday Vishal Kanna Anand

S.No- 1006874517

Q.1.2- Provide a table that compares the 10-most cosine-similar words to the word 'dog', in order, alongside to the 10 closest words computed using Euclidean distance. Give the same kind of table for the word 'computer.' Looking at the two lists, does one of the metrics (cosine similarity or Euclidean distance) seem to be better than the other? Explain your answer.

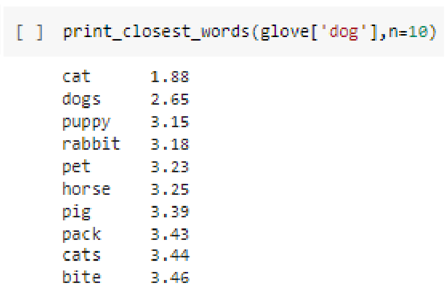
A-

Comparison for Dog



```
[ ]  print_closest_cosine_words(glove["dog"], n=10)

     cat      0.92
     dogs     0.85
     horse    0.79
     puppy    0.78
     pet      0.77
     rabbit   0.77
     pig      0.75
     snake    0.74
     baby     0.74
     bite     0.74
```

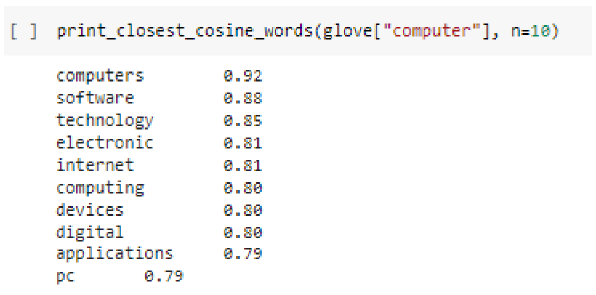


```
[ ]  print_closest_words(glove['dog'],n=10)

     cat      1.88
     dogs     2.65
     puppy    3.15
     rabbit   3.18
     pet      3.23
     horse    3.25
     pig      3.39
     pack     3.43
     cats     3.44
     bite     3.46
```

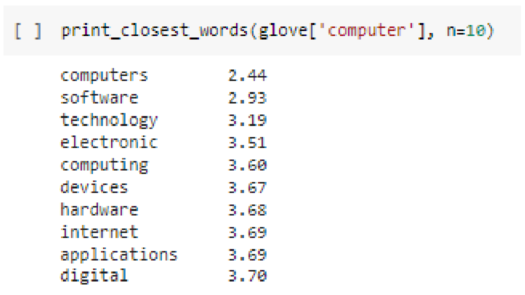
The function on the left uses' cosine similarity and on the right Euclidean distance is used. From both these lists it can be seen that they both provide fairly similar lists of words with slight changes to the ordering.

Comparison for Computer



```
[ ]  print_closest_cosine_words(glove["computer"], n=10)

     computers        0.92
     software         0.88
     technology       0.85
     electronic       0.81
     internet         0.81
     computing        0.80
     devices          0.80
     digital          0.80
     applications     0.79
     pc       0.79
```



```
[ ]  print_closest_words(glove['computer'], n=10)

     computers        2.44
     software         2.93
     technology       3.19
     electronic       3.51
     computing        3.60
     devices          3.67
     hardware         3.68
     internet         3.69
     applications     3.69
     digital          3.70
```

A similar obervation can be drawn from the 2 lists above for the word computer.

Overall, on observation Both methods work just as well.

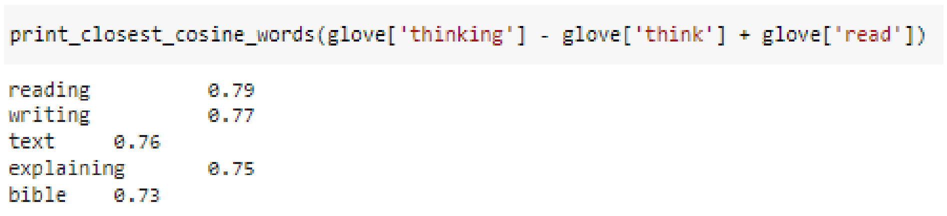
Q.1.3- The section of A1_Section1_starter.ipynb that is labelled Analogies shows how a relationships between pairs of words that is captured in the learned word vectors. Consider, now, the word-pair relationships given in Figure 1 below, which comes from Table 1 of the Mikolov[2] paper. Choose one of these relationships, but not one of the ones already shown in the starter notebook, and report which one you chose. Write and run code that will generate the second word given the first word. Generate 10 more examples of that same relationship from 10 other words, and comment on the quality of the results

A-

The relationship of interest in the analogies is present participle.

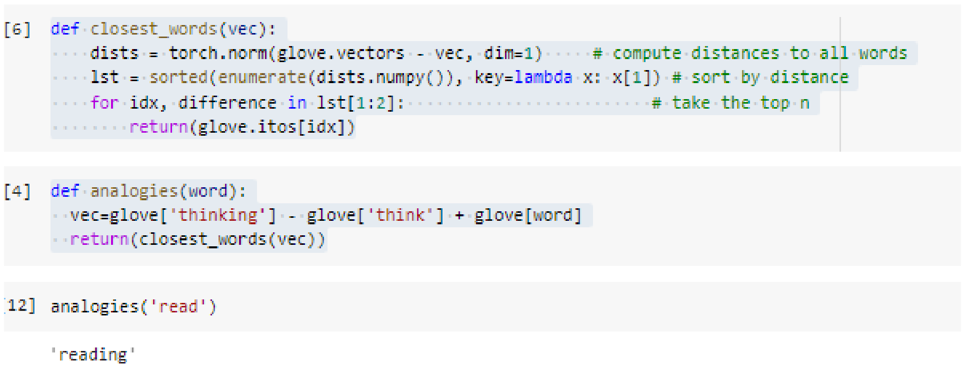
From the results it can be seen that using the analogies the top 5 closest words contain the present participle form of the final word added to the analogy. But when returning the closest word it does not always give us the present participle form of the word.

Eg.



```
print_closest_cosine_words(glove['thinking'] - glove['think'] + glove['read'])
```

```
reading         0.79
writing         0.77
text    0.76
explaining      0.75
bible   0.73
```



```
[6] def closest_words(vec):
        dists = torch.norm(glove.vectors - vec, dim=1)     # compute distances to all words
        lst = sorted(enumerate(dists.numpy()), key=lambda x: x[1]) # sort by distance
        for idx, difference in lst[1:2]:                        # take the top n
            return(glove.itos[idx])
```

```
[4] def analogies(word):
      vec=glove['thinking'] - glove['think'] + glove[word]
      return(closest_words(vec))
```

```
[12] analogies('read')

     'reading'
```

Input word	Output word
read	reading
bake	baking
drink	drinks
sleep	breathing
try	attempting
cook	fry
walk	walking
talk	conversation
run	ran

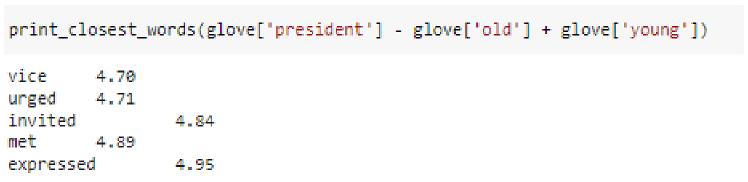
write	writing
sit	sitting

Q.1.4- The section of A1_Section1_starter.ipynb that is labelled Bias in Word Vectors illustrates examples of bias within word vectors in the notebook and as discussed in class. Choose a context that you're aware of (different from those already in the notebook), and see if you can find evidence of a bias that is built into the word vectors. Report the evidence and the conclusion you make from the evidence.

A-

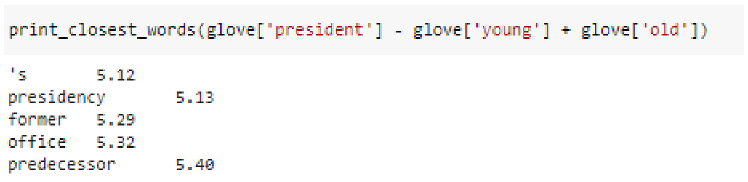
The bias we will try to find evidence for is the age bias-

From experimentation-



```
print_closest_words(glove['president'] - glove['old'] + glove['young'])
```

```
vice	4.70
urged	4.71
invited		4.84
met	4.89
expressed	4.95
```



```
print_closest_words(glove['president'] - glove['young'] + glove['old'])
```

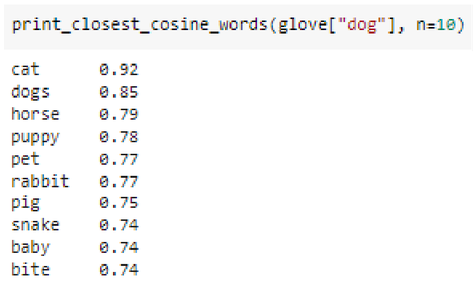
```
's	5.12
presidency	5.13
former	5.29
office	5.32
predecessor	5.40
```

The age bias can be clearly seen in this example where we add young to president, we get things like vice and urged. Whereas when we add old to the president, we get things like former and predecessor. We can collude that the glove embeddings have a age bias in their training.

Q.1.5- Change the embedding dimension (also called the vector size) from 50 to 300 and rerun the notebook including the new cosine similarity function from part 2 above. How does the euclidean difference change between the various words in the notebook when switching from d=50 to d=300? How does the cosine similarity change? Does the ordering of nearness change? Is it clear that the larger size vectors give better results - why or why not?

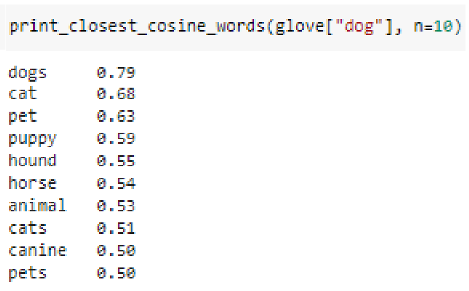
A-

The left has embedding size of 50 and the right has size 300



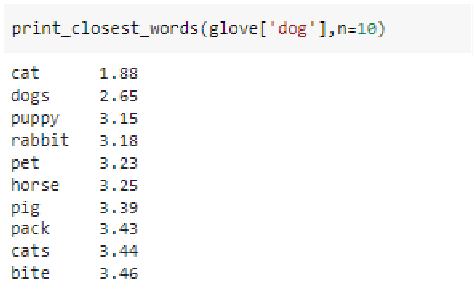
```
print_closest_cosine_words(glove["dog"], n=10)
```

```
cat	0.92
dogs	0.85
horse	0.79
puppy	0.78
pet	0.77
rabbit	0.77
pig	0.75
snake	0.74
baby	0.74
bite	0.74
```



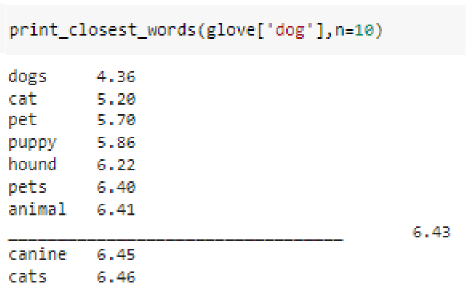
```
print_closest_cosine_words(glove["dog"], n=10)
```

```
dogs	0.79
cat	0.68
pet	0.63
puppy	0.59
hound	0.55
horse	0.54
animal	0.53
cats	0.51
canine	0.50
pets	0.50
```



```
print_closest_words(glove['dog'],n=10)
```

```
cat	1.88
dogs	2.65
puppy	3.15
rabbit	3.18
pet	3.23
horse	3.25
pig	3.39
pack	3.43
cats	3.44
bite	3.46
```



```
print_closest_words(glove['dog'],n=10)
```

```
dogs	4.36
cat	5.20
pet	5.70
puppy	5.86
hound	6.22
pets	6.40
animal	6.41
___________________________________	6.43
canine	6.45
cats	6.46
```

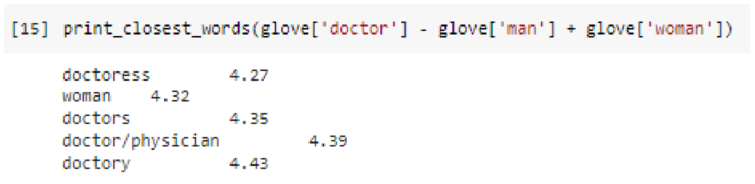
From the picture above we can see that the Euclidean distance for the embedding size of 300 is larger than the ones with embedding size of 50. This difference is also present in the cosine similarity where the embeddings of size 300 have a lower similarity than of size 50.

There is a change in the ordering when we go from a size of 50 to 300. This change in ordering also seems more accurate through out all the examples in the notebook. It is clear that larger vectors gives us better results as it gives the words better explainability as there is more expressability with the larger vectors .

Q.1.6 There are many different pre-trained embeddings available, including one that tokenizes words at a sub-word level[3] called FastText. These pre-trained embeddedings are available from Torchtext. Modify the notebook to use the FastText embeddings. State any changes that you see in the Bias section of the notebook.

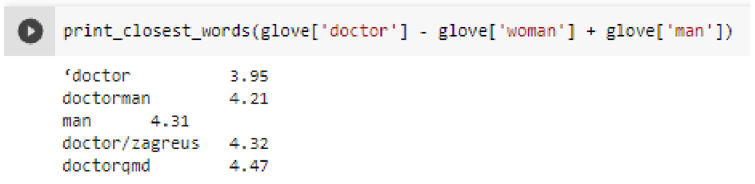
A-

Using the FastText embedding the gender bias that was prevalent in Glove does not appear here as both Male and Females have similar words that show up for each profession. It also removes the age biases that we experimented with in 1.4



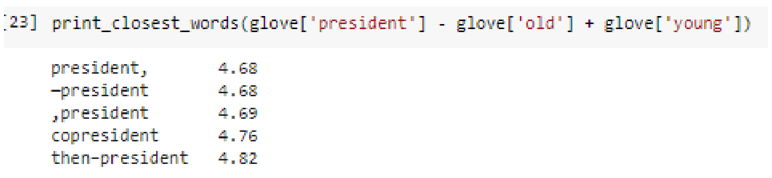
```
[15] print_closest_words(glove['doctor'] - glove['man'] + glove['woman'])

     doctoress        4.27
     woman    4.32
     doctors          4.35
     doctor/physician         4.39
     doctory          4.43
```



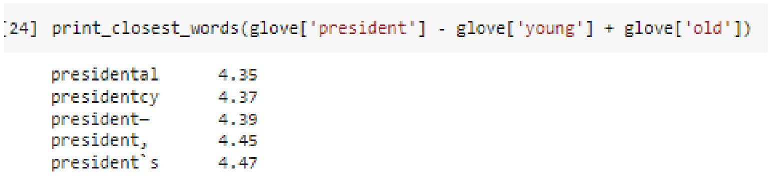
```
print_closest_words(glove['doctor'] - glove['woman'] + glove['man'])

     ‘doctor          3.95
     doctorman        4.21
     man      4.31
     doctor/zagreus   4.32
     doctorqmd        4.47
```



```
[23] print_closest_words(glove['president'] - glove['old'] + glove['young'])

    president,       4.68
    –president       4.68
    ,president       4.69
    copresident      4.76
    then-president   4.82
```



```
[24] print_closest_words(glove['president'] - glove['young'] + glove['old'])

    presidental      4.35
    presidentcy      4.37
    president–       4.39
    president,       4.45
    president`s      4.47
```

Q.2.2- Let’s define the colour meaning category using these words: “colour”, “red”, “green”, “blue”, “yellow.” Compute the similarity (using both methods (a) and (b) above) for each of these words: “greenhouse”, “sky”, “grass”, “purple”, “scissors”, “microphone”, “president” and present them in a table. Do the results for each method make sense? Why or why not? What is the apparent difference between method 1 and 2?

A-

Method 1- By averaging the cosine similarity of the given word with every word in the category

Method 2- computing the cosine similarity of the word with the average embedding of all of the words in the category

Words	Method 1	Method 2
greenhouse	`tensor([0.1831])`	`tensor([0.2017])`
sky	`tensor([0.6019])`	`tensor([0.6702])`
grass	`tensor([0.5060])`	`tensor([0.5579])`
purple	`tensor([0.7993])`	`tensor([0.8887])`
scissor	`tensor([0.2890])`	`tensor([0.3203])`
microphone	`tensor([0.3077])`	`tensor([0.3431])`
president	`tensor([0.2986])`	`tensor([0.3292])`

Results from both the methods make sense as they both have very similar results- words like grass, sky and purple have a large cosine similarity value.

The difference between both the methods is that method 2 consistently always has a higher similarity to the category compared to method 1.

Q.2.3- Create a similar table for the meaning category temperature by defining your own set of category words, and test a set of 10 words that illustrate how well your category works as a way to determine how much temperature is "in" the words. You should explore different choices and try to make this succeed as much as possible. Comment on how well your approach worked.

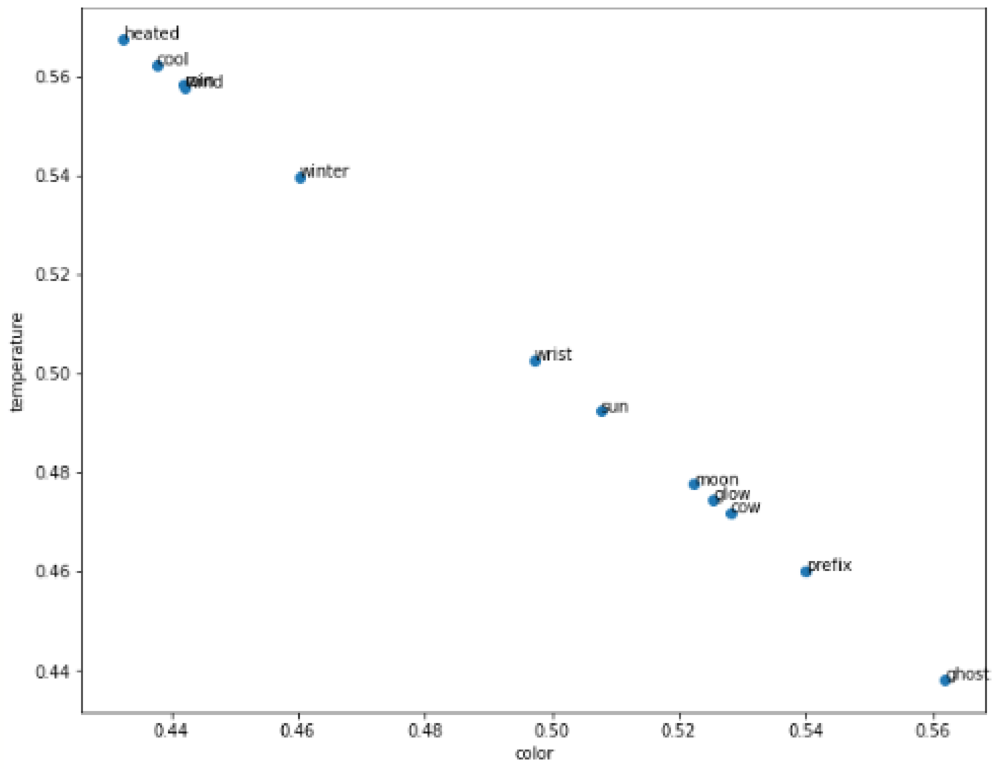
A-

Temperature category-
`['hot','cold','celsius','freeze','warm','chill','temperature']`

Words	Temperature category
moon	tensor([0.4179])
fire	tensor([0.4589])
snow	tensor([0.7281])
heat	tensor([0.8257])
rock	tensor([0.4617])
house	tensor([0.3049])
family	tensor([0.2200])
bored	tensor([0.3513])
bottle	tensor([0.4384])
water	tensor([0.7247])

This approach did well- as words like family and house have a low cosine similarity value with the category and words like heat and snow have high cosine similarity value. But there is still room for improvement as words like fire still have fairly low similarity values.

Q.2.4– Use these two categories (colour & temperature) to create a new word vector (of dimension 2) for each of the words given in Table 1, in the following way: for each word, take its (colour, temperature) cosine similarity numbers, and apply the softmax function to convert those numbers into a probabilities. Plot each of the words in two dimensions (one for colour and one for temperature) using matplotlib. Do the words that are similar end up being plotted close together? Why or why not?



From the plot we can see similar words like heated, cool, rain and wind are close together and high on temperature scale. This is because all of these words have a higher similarity with temperature than with color, and also have similar similarity to the 2 categories Words like moon and glow ghost have higher similarity to color than temperature and have similar similarity to the 2 categories. Hence the grouping occurs when using these categories.

Q.3.1 First, read the file SmallSimpleCorpus.txt so that you see what the sequence of sentences is. Recalling the notion "you shall know a word by the company it keeps," find three pairs of words that this corpora implies have similar or related meanings. For example, 'he' and 'she' are one such example – which you cannot use in your answer!

A-

Word pairs-

(dog, cat), (hold,rub),(I,he)

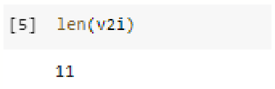
Q3.2 Review the code of prepare_texts to make sure you understand what it is doing. Write the code to read the corpus SmallSimpleCorpus.txt, and run the prepare_texts on it to return the text (lemmas) that will be used next. Check that the vocabulary size is 11. Which is the most frequent word in the corpus, and the least frequent word? What purpose do the v2i and i2v functions serve?

A-

Most frequent word- and

Least frequent word- I

Vocab size-



```
[5]  len(v2i)

11
```

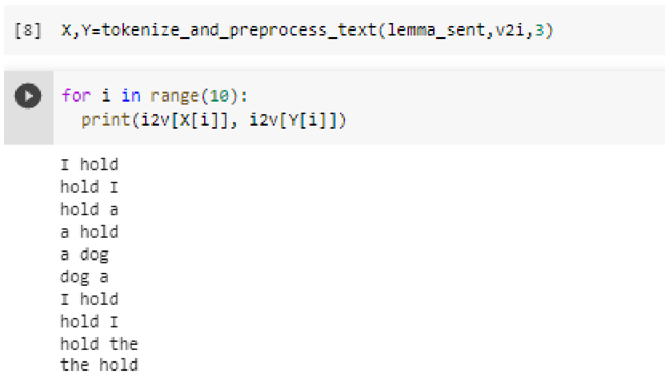
it can be confirmed that it is 11

The purpose of v2i is convert the word into a number of token that can be read by the neural network model. i2v is used to convert that tokenised number back into its respective word.

Q3.3 You must generate all training examples across all words in the corpus within a window of size window. Test that your function works, and show with examples of output (submitted) that it does.

A-

Example of the first 10 output pairs-



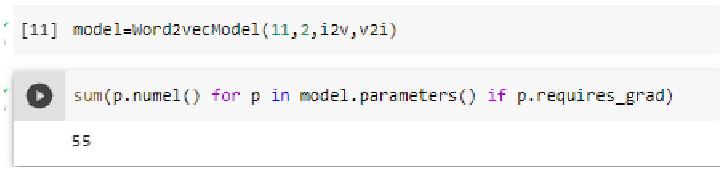
```
[8]  X,Y=tokenize_and_preprocess_text(lemma_sent,v2i,3)
```

```
for i in range(10):
  print(i2v[X[i]], i2v[Y[i]])
```

```
I hold
hold I
hold a
a hold
a dog
dog a
I hold
hold I
hold the
the hold
```

Q3.4 Set the embedding size to be 2, so that will be the size of our word embeddings/vectors. What is the total number of parameters in this model with an embedding size of 2 - counting all the weights and biases?

A-



```
[11]  model=Word2vecModel(11,2,i2v,v2i)
```

```
sum(p.numel() for p in model.parameters() if p.requires_grad)

55
```

The model we created has an embedding layer and a liner layer to give us 11 outputs. From the code above we can see the total number of weights and biases of this model is 55**.**

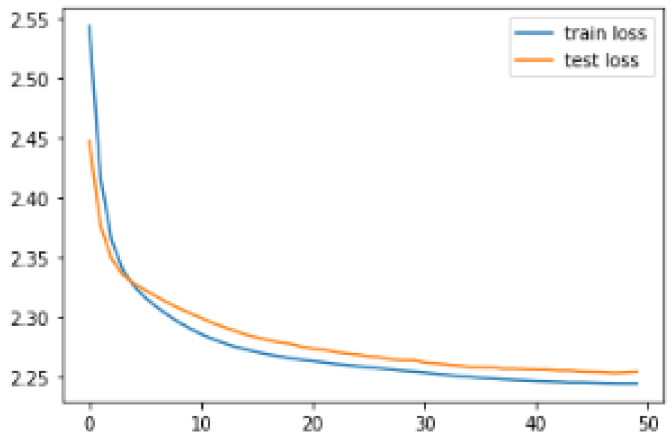
Q3.5 Show the training and validation curves (loss vs. Epoch), and comment on the apparent success (or lack thereof) that these curves suggest.

A-

Learning rate used- `0.001`

```
[ ]  network = train_word2vec(lemma_sent,5,2,v2i,i2v)
     embedding = network.embedding
```

```
Finished Training
```

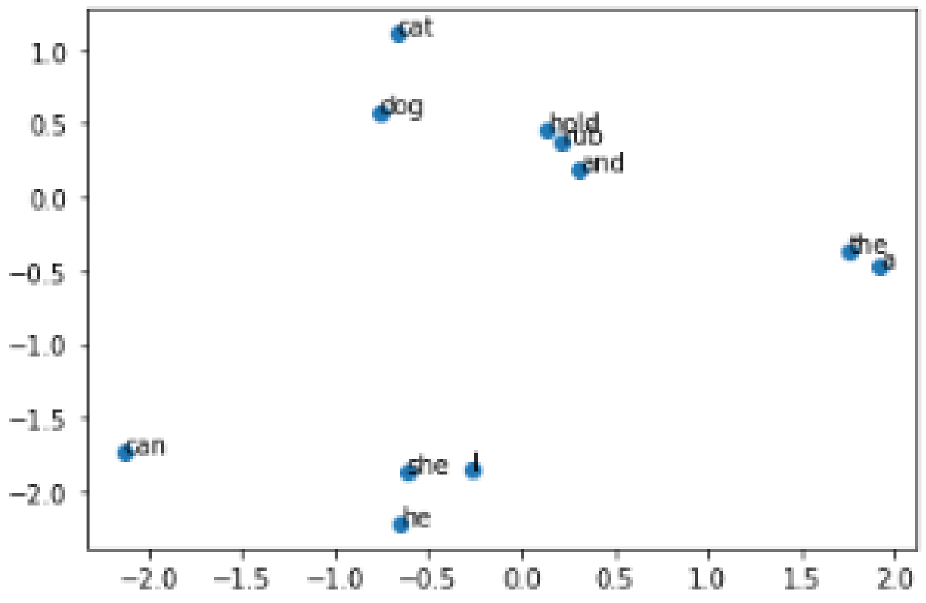


The y axis is loss, and the x axis are the epochs. The blue line represents train loss, and the orange line is the validation loss.

From the graph it can be seen that both the train and validation loss consistently go down as we run epochs of training on the model. We can conclude that this training process was a success.

Q3.6 For your best learning rate, display each of the embeddings in a 2-dimensional plot using Matplotlib. Display both a point for each word, and the word itself. Submit this plot, and answer this question: Do the results make sense, and confirm your choices from part 1 of this Section? What would happen when the window size is too large? At what value would window become too large for this corpus?

A-



The results of the plot make sense as words like he, she and I are close together. Hold and rub are close together and words like cat and dog are close together. We can see this is the same as the answers in part 1.

If the window size is too large, then all the words will be related to each other and we would not be able to distinguish them from each other. Our plot will have all the points in a single cluster.

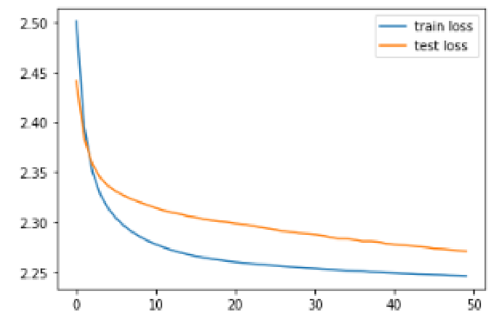
When the window size is greater than 15 it becomes too large for this corpus as 7 words make up the largest sentence in this corpus. This will ensure all words are related to one another.

Q3.7 Run the training sequence twice - and observe whether the results are identical or not. Then set the random seeds that are used in, separately in numpy and torch as follows (use any number you wish, not necessary 43, for the seed): np.random.seed(43) torch.manual_seed(43) Verify (and confirm this in your report) that the results are always the same every time you run your code if you set these two seeds. This will be important to remember when you are debugging code, and you want it to produce the same result each time.

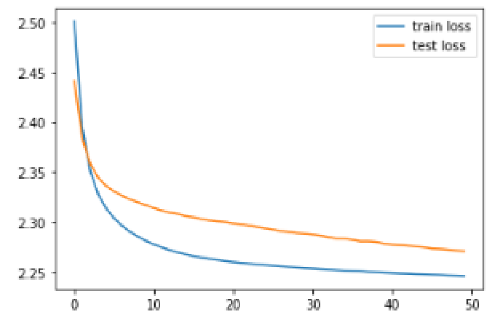
A-

Training with np.random.seed(43) torch.manual_seed(43)-

Finished Training



Finished Training



We can confirm that they are identicle when the model was trained twice.

Q4.1 Take a quick look through LargerCorpus.txt to get a sense of what it is about. Give a 3 sentence summary of the subject of the document.

A-

The text is about the history of money and its evolution over time. It starts of with ancient money made of precious metals and goes into their characteristics and goes till present day minted money. It also tells us the change in value of coins over time.

Q4.2 The prepare_texts function in the starter code is a more advanced version of that same function given in Section 3. Read through it and make sure you understand what it does. What are the functional differences between this code and that of the same function in Section 3?

A-

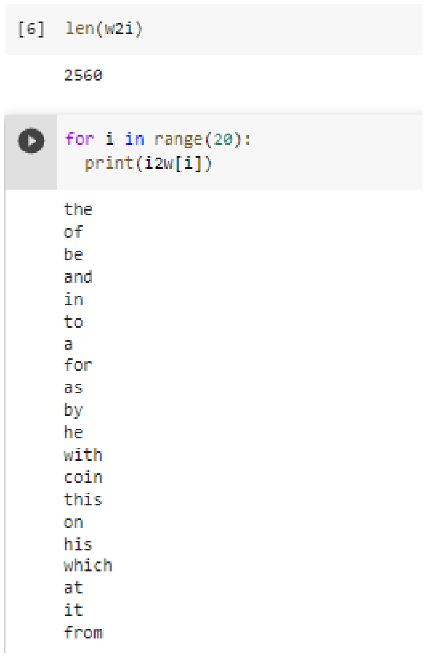
The differences from the code in section 3 is that it makes the text into sentences which the function section 3 didn't and then process them. It also creates a out of vocabulary word and index and keeps track of it.

Q4.3 Write the code to read in LargerCorpus.txt and run prepare_texts on it. Determine the number of words in the text, and the size of the filtered vocabulary, and the most frequent 20 words in the filtered vocabulary, and report those. Of those top 20 most frequent words, which one(s) are unique to the subject of this particular text?

A-

Number of words in text- 77885

Size of filtered vocabulary- 2560



```
[6] len(w2i)

    2560
```

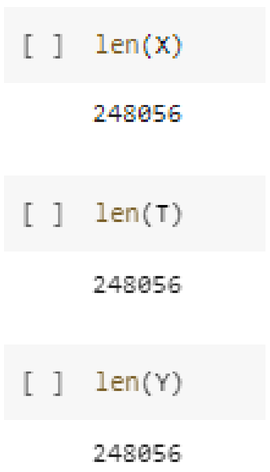
```
for i in range(20):
  print(i2w[i])

the
of
be
and
in
to
a
for
as
by
he
with
coin
this
on
his
which
at
it
from
```

From the top 20 words- coin is the only word unique to the subject of the text.

Q4.4 Write the function tokenize_and_preprocess_text which generates both positive and negative samples for training in the following way: first, use w2i to create a tokenized version of the corpus. Next, for every word in the corpus, generate: a set of positive examples within a window of size window similar to the example generation in Section 3. Set the label for each positive example to be +1, in the list Y produced by the function. Also, for each word, generate the same number of negative samples as positive examples, by choosing that number of randomly-chosen words from the entire corpus. (You can assume randomly chosen words are very likely to be not associated with the given word). Set the label of the negative examples to be -1. Submit your code for this function in the file named A1P4_4.py. How many total examples were created?

A-



```
[ ]  len(X)

     248056

[ ]  len(T)

     248056

[ ]  len(Y)

     248056
```

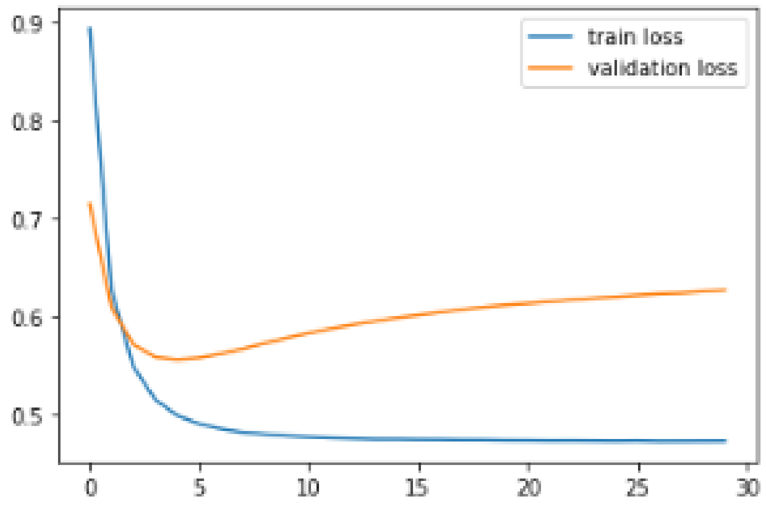
Total examples created- 248056

Q4.7 Using the default Adam optimizer, find a suitable learning rate, and report what that is. Show the training and validation curves vs. Epoch, and comment on the apparent success (or lack thereof) that these curves suggest.

A-

Suitable learning rate- `0.0005`

```
Finished Training
```

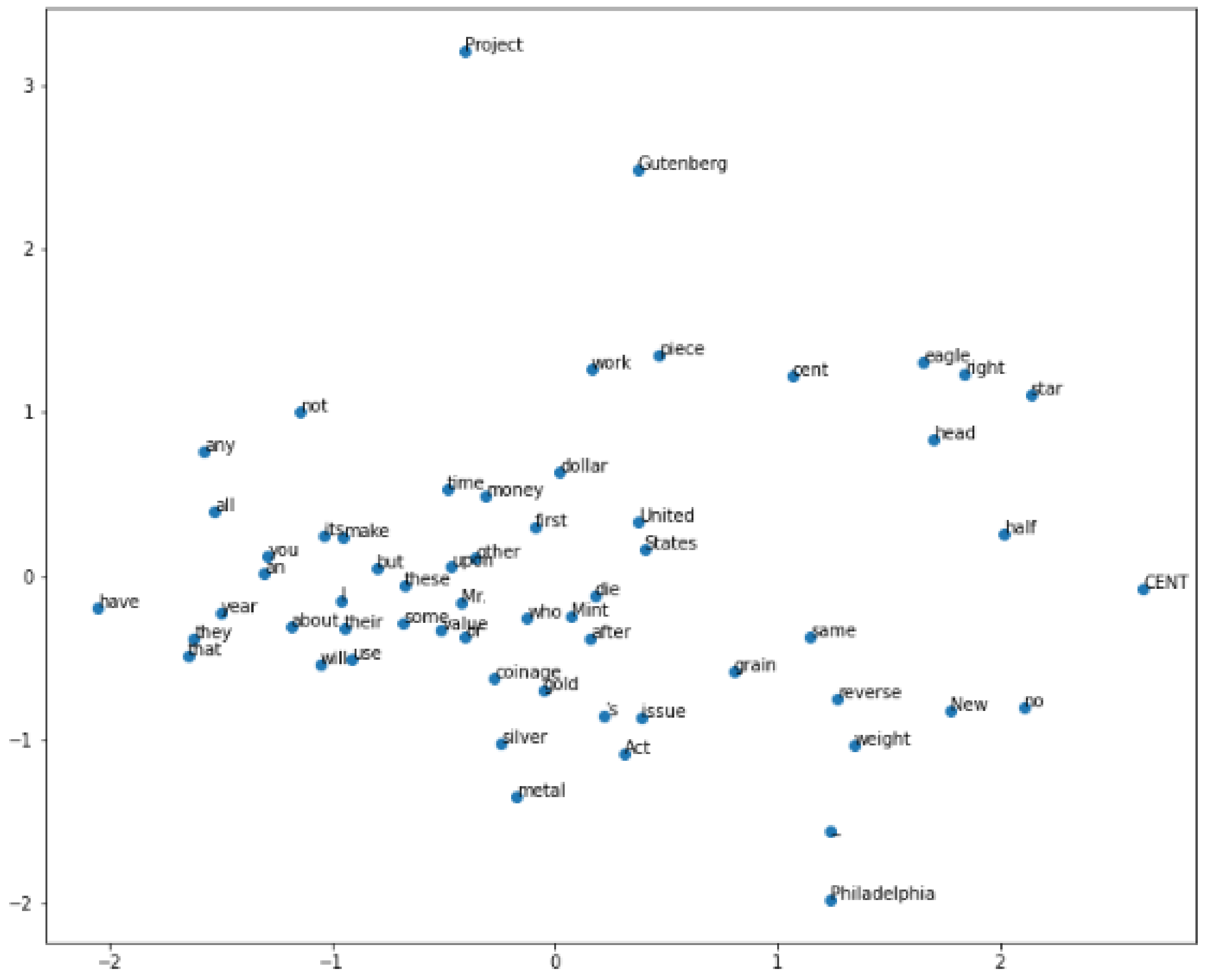


The X axis of the graph is epoch count and the Y axis is loss value. From the plot above we can see the training loss continuously decreases through the epochs, but validation loss starts to decrease and slowly starts to increase after the 5th epoch. This can indicate the the model may be overfitting to the training set, which is not a good indicator and we might need to introduce a form of regularization to improve the model.

Q4.8 Write a function that reduces the dimensionality of the embeddings from 8 to 2, using principle component analysis (PCA) as shown in the partially-written function visualize_embedding. Since we cannot visualize the embeddings of the entire vocabulary, the function is written to select a range of the most frequent words in the vocabulary. (Too frequent are not interesting, but too infrequent are also less interesting). Comment on how well the embeddings worked, finding two examples each of embeddings that appear correctly placed in the plot, and two examples where they are not.

A-

```
Visualizing the 20 to 80 most frequent words
```



The embeddings worked to an extent as the plot above has some similar words that are clustered together and some similar words far apart.

Correct pairs of words- (money, dollar), (silver, metal), (United, States)

Incorrect pair of words- (CENT, Dollar), (Cent, Money),(Year, time)