Question 1.2

Question:

Write a new function, similar to print_closest_words called print_closest_cosine_words that prints out the N–most (where N is a parameter) similar words using cosine similarity rather than euclidean distance. 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. Submit the specific code for the print_closest_cosine_words function that you wrote in a separate Python file named A1P1_2.py. [2 points]

Answer:

```
compare'dog'word similar to:

cosine:
word    cosine similarity
dog      1.00
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

Euclidean:
word    Euclidean distance
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
------------------------------------
```

```
compare'computer'word similar to:

cosine:
word    cosine similarity
computer         1.00
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

Euclidean:
word    Euclidean distance
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
```

The cosine similarity is better than the Euclidean distance in NLP. It is more robust to the magnitude of the vector and captures the direction of the similarity

---

Question 1.3

Question:

The section of A1_Section1_starter.ipynb that is labelled Analogies shows how relationships between pairs of words 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. Submit the specific code that you wrote in a separate Python file, A1P1_3.py.

Answer:

I choose the relationship: glove['king'] –glove ['prince'] + glove[‘princess’].

```
princess          0.86
queen     0.85
king      0.77
lady      0.71
bride     0.70


emperor + Relationship =
emperor           0.90
king      0.86
empress           0.80
queen     0.80
throne    0.79


monarch + Relationship =
monarch           0.87
queen     0.86
king      0.83
princess          0.79
throne    0.78


tsar + Relationship =
queen     0.82
king      0.81
tsar      0.80
princess          0.79
emperor           0.78
```

```
sultan + Relationship =
king      0.86
sultan    0.82
prince    0.77
queen     0.76
princess          0.74


czar + Relationship =
king      0.75
queen     0.73
princess          0.71
widow     0.70
ii        0.70


duke + Relationship =
queen     0.84
duke      0.84
king      0.83
princess          0.80
duchess           0.76


lord + Relationship =
lord      0.87
king      0.86
queen     0.84
lady      0.77
princess          0.74
```

```
nobleman + Relationship =
queen    0.78
princess         0.78
nobleman         0.77
king     0.76
throne   0.73


ruler + Relationship =
king     0.89
ruler    0.83
throne   0.82
monarch          0.81
queen    0.80


regent + Relationship =
queen    0.87
princess         0.83
regent   0.79
king     0.79
consort          0.78
```

In the first relation generation, "king" to "princess" speculation is very successful, and the top words include "princess", "queen", and "king", as expected.

For the other 10 examples, the results of some examples are in line with expectations, such as "emperor", "monarch", "sultan", etc., but in some cases, the results may be less accurate, for example, the results of "czar" are far from expectations.

Question 1.4

Question:

The section of A1_Section1_starter.ipynb that is labelled Bias in Word Vectors illus– trates examples of bias within word vectors in the notebook, as also 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.

Answer:

Evidence of Bias:

I choose the word builder.

```
print_closest_words(glove['builder'] - glove['man'] + glove['woman'])
```
✓ 0.2s

```
pioneer          3.76
survives         3.94
mechanic         3.96
acquires         4.03
realtor          4.07
```

```
print_closest_words(glove['builder'] - glove['woman'] + glove['man'])
```
✓ 0.2s

```
mover    4.11
developer        4.11
builders         4.20
venerable        4.33
visionary        4.37
```

Conclusion:
It's obvious that the evidence confirms that the resulting word from the analogy is biased, it illustrates that the word vectors have encoded a gender bias present in the training data. This highlights the importance of being aware of and addressing biases in machine learning models, as they can perpetuate and amplify societal prejudices.

Question 1.5

Question:
Change the the embedding dimension (also called the vector size) from 50 to 300 and re-run 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?

Answer:
Changing the embedding dimension (vector size) from 50 to 300 can have a significant impact on the performance of word vectors.

1. Euclidean Difference Comparison (d=50 vs. d=300):
The words become more similar in the higher-dimensional space.

2. Cosine Similarity Comparison (d=50 vs. d=300):
Similar to the Euclidean difference, when increasing the dimension from 50 to 300, cosine similarities increase, reflecting greater similarity between words.

3. Ordering of Nearness:
The ordering of nearness change when switching from d=50 to d=300.

4. Effect of Larger Vector Size (d=300):
Higher-dimensional embeddings have more capacity to capture fine-grained semantic relationships between words. They can represent nuances and subtleties in word meanings better than lower-dimensional embeddings.

Question 1.6

Question:
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. [2 points]

Answer:
The example analogy doctor — man + woman resulted in the word "nurse" being the closest in meaning, similar to what was seen with GloVe embeddings. This suggests that the gender bias is present in both GloVe and FastText embeddings, where the profession "doctor" when combined with female-associated terms leads to professions stereotypically associated with women. The bias seems consistent between the two embeddings, indicating that the training data's inherent biases have influenced both models. The embeddings have captured societal stereotypes, as evident from the analogy results.

Question 2.2

Question:
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", "azure", "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? [4 points]

Answer:
Yes. The results make sense. Words like "greenhouse," "sky," "grass," "azure," and "scissors" are expected to have higher similarity scores with the color category, while words like "microphone" and "president" are not directly related to color and should have lower similarity scores.

Word	Method (a)	Method (b)
greenhouse	0.2019	0.2225
sky	0.6013	0.6658
grass	0.5214	0.5720
azure	0.3776	0.4176
scissors	0.3038	0.3355
microphone	0.3165	0.3517
president	0.3238	0.3563

The apparent difference between method (a) and method (b) is that method (a) calculates the average similarity between the word and each word in the color category, while method (b) calculates the similarity between the word and the average embedding of all the words in the color category. Method (a) takes into account the similarity to each individual word in the category, whereas method (b) considers the overall representation of the category.

Question 2.3

Question:

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. [4 points]

Answer:

Word	Method (a)	Method (b)
summer	0.5260	0.6530
ice	0.5144	0.6425
thermometer	0.3386	0.4284
oven	0.4152	0.5220
snow	0.6125	0.7588
beach	0.3319	0.4076
jacket	0.2445	0.3048
fire	0.3666	0.4538
coffee	0.3976	0.4997
desert	0.4298	0.5270

I defined a set of category words related to temperature and tested them against words that we expect to have varying degrees of temperature association. "Snow," "cold," and "ice" have high similarity scores, while words like "beach" and "coffee" should have lower scores.

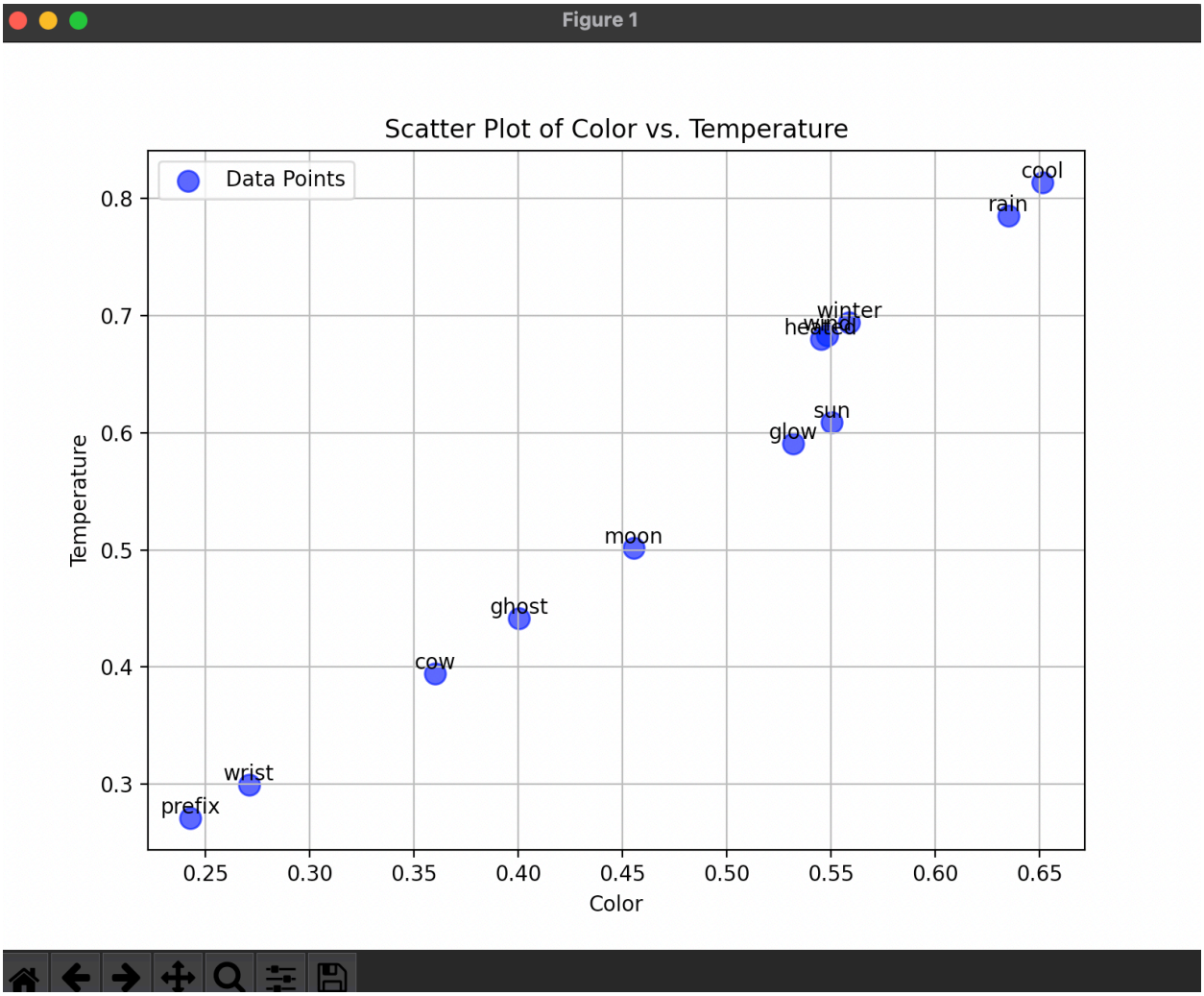
Question 2.4

Question:

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 (try both methods and see which

works better), 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? [2 points]

Answer:



Yes. Like the word heat and winter and wind, the tradeoff between color and temperature, those word have similar portions.

Question 3.1

Question:

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! [1 point]

Answer:
I choose these three pairs:
rub – hold
cat – dog
a – the

Question 3.2

Question:
The prepare_texts function in the starter code is given to you and fulfills several key func– tions in text processing, a little bit simplified for this simple corpus. Rather than full tokeniza– tion (covered in Section 4 below, you will only lemmatize the corpus, which means converting words to their root – for example the word "holds" becomes "hold", whereas the word "hold" itself stays the same (see the Jurafsky [4] text, section 2.1 for a discussion of lemmatization). The prepare_texts function performs lemmatization using the spaCy library, which also per– forms parts of speech tagging. That tagging determines the type of each word such as noun, verb, or adjective, as well as detecting spaces and punctuation. Jurafsky [4] Section 8.1 and 8.2 describes parts–of–speech tagging. The function prepare_texts uses the parts–of–speech tag to eliminate spaces and punctuation from the vocabulary that is being trained.
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? [2 points]

Answer:

```
Most frequent word: and ,Apperance: 160
Least frequent word: I ,Apperance: 80
```

v2i (word–to–index) and i2v (index–to–word) to map words to their corresponding indices.

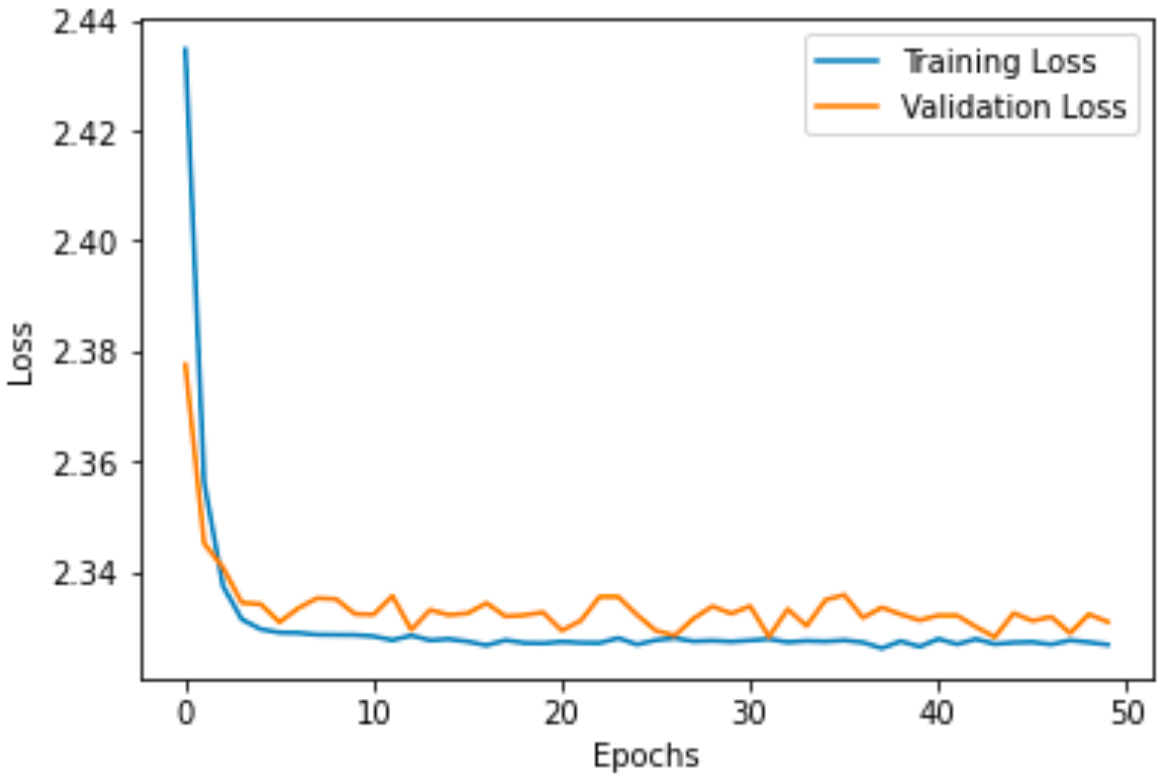
Question 3.3

```
[10, 10, 10, 1, 1, 1, 1, 5, 5, 5, 5, 5, 2, 2, 2, 2, 2, 2, 10, 10, 10, 10, 10, 10, 1, 1, 1, 1, 1, 1, 6, 6, 6, 6, 6, 6, 2, 2, 2, 2, 2, 2, 10, 10,
[1, 5, 2, 10, 5, 2, 10, 10, 1, 2, 10, 1, 10, 1, 5, 10, 1, 6, 1, 5, 2, 1, 6, 2, 5, 2, 10, 6, 2, 10, 2, 10, 1, 2, 10, 1, 10, 1, 6, 10, 1, 5, 1, 6
```

Question 3.4

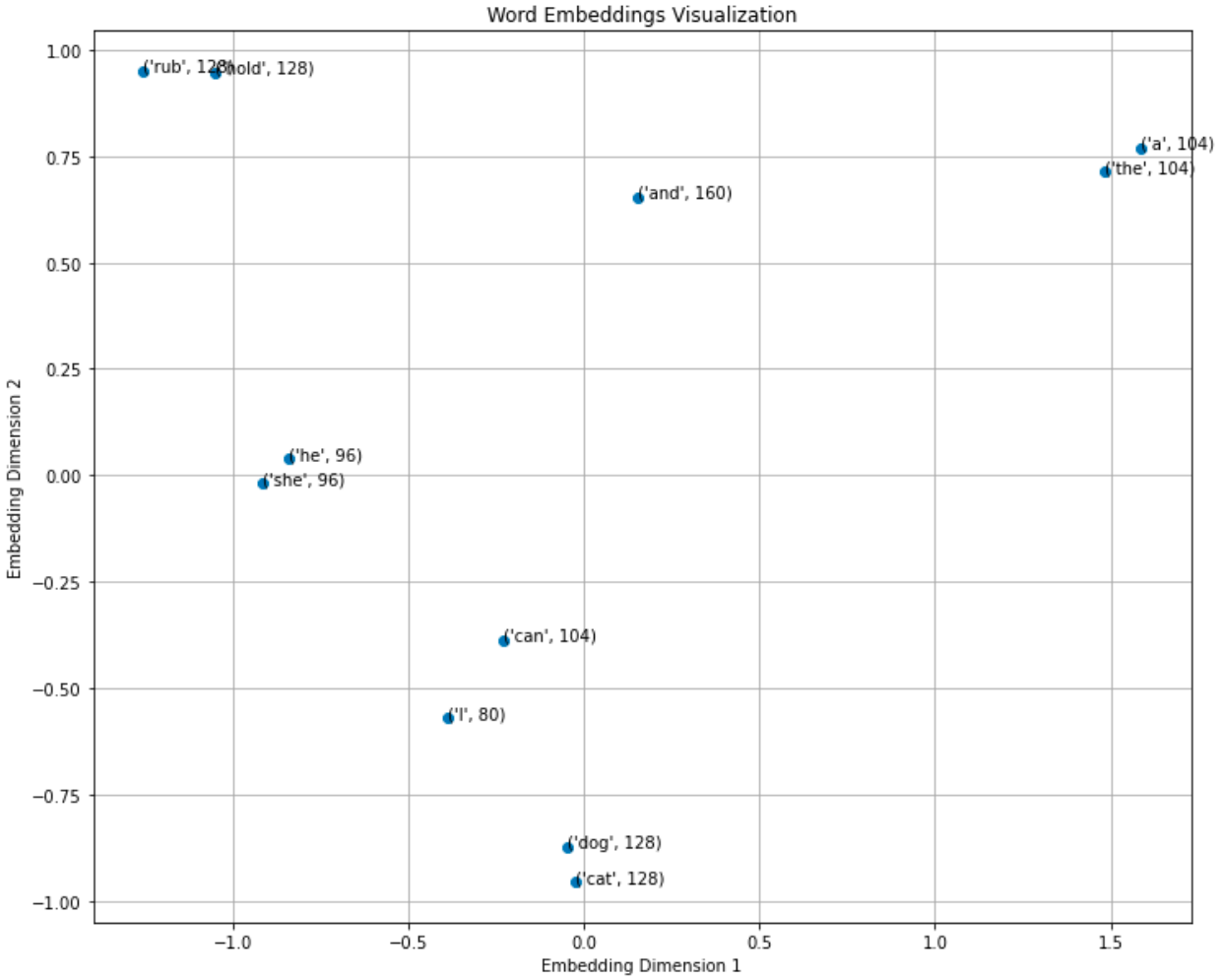
For a model with an embedding size of 2 and a vocabulary size of 11:
The total parameters = 2 x 2 x 11 + 11 = 55

Question 3.5



Learning rate = 0.004
There is an apparent success:
both the training and validation losses decrease and stabilize, it suggests that the model is learning and not overfitting.

Question 3.6

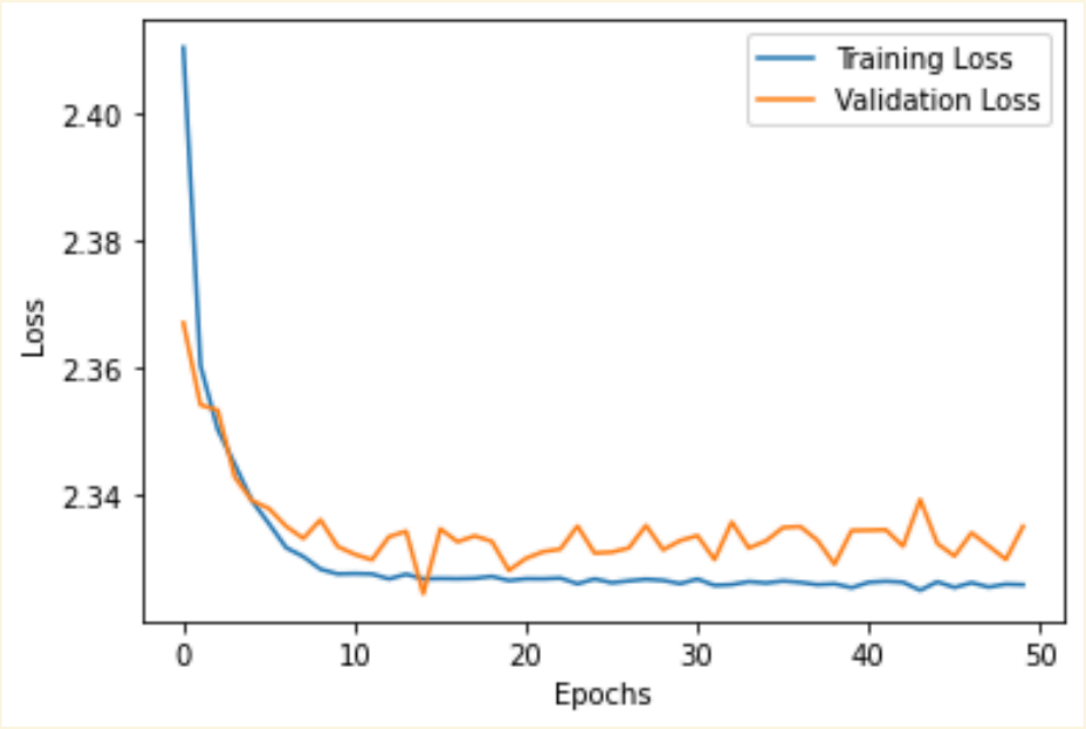


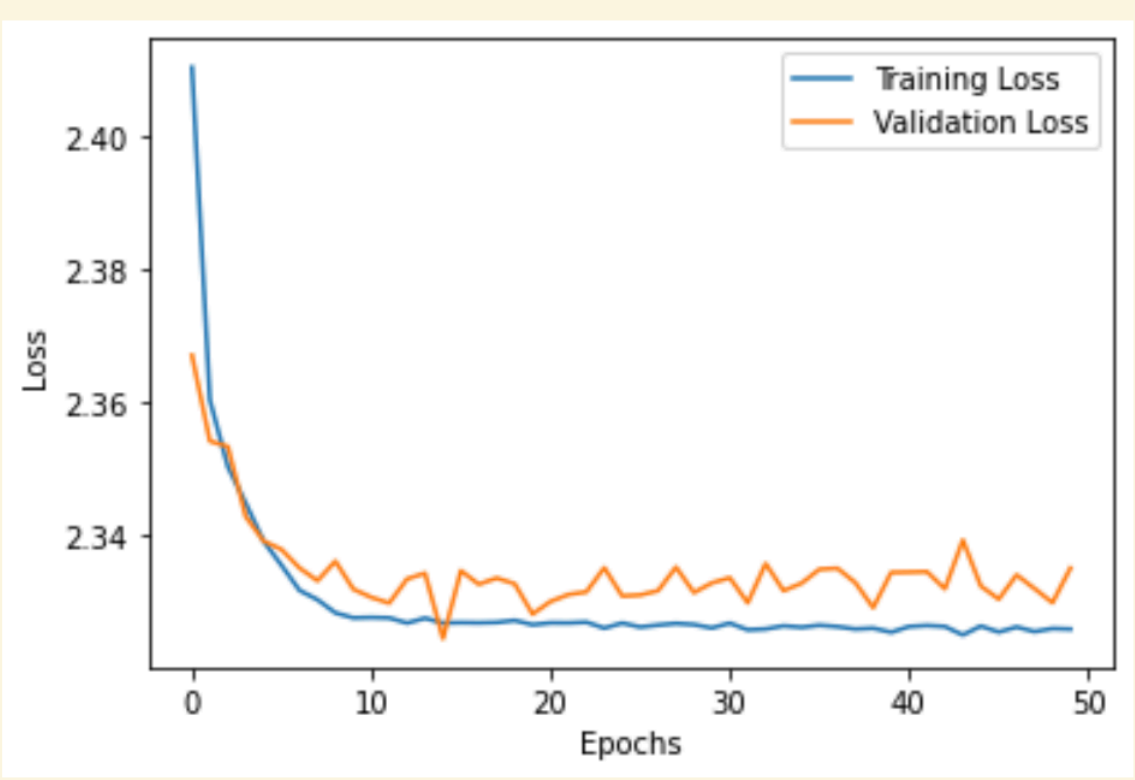
Yes. The results make sense. Rub–hold, a–the, cat–dog are exactly the pair.

When the window size is too large, the model might consider distant words as context, which may not be semantically related to the target word. This might dilute the quality of the embeddings.

A window becomes too large when it's larger than the average sentence length in corpus. If the window size exceeds this, then we can not gaining more context, and we may just be adding noise to the embeddings. So the largest size is 5.

Question 3.7


Training Loss
Validation Loss
2.40
2.38
2.36
2.34
Loss
0
10
20
30
40
50
Epochs


Training Loss
Validation Loss
2.40
2.38
2.36
2.34
Loss
0
10
20
30
40
50
Epochs

When the seeds were not set, running the training sequence multiple times produced different results. This can be observed in the differences in the training and validation curves of the two runs. After setting the random seeds for both numpy and torch, the training results were consistent across multiple runs. The training and validation curves for the two runs were identical, confirming the importance of setting random seeds for reproducibility. Setting random seeds is crucial when aiming for reproducibility in experiments. It ensures consistent initializations and other random processes, leading to identical results across multiple runs.

Question 4.1

Answer:

The history and evolution of money. It begins by discussing the ancient need for a medium of exchange, initially satisfied by bartering. The document then proceeds to list various commodities that have served as money in different regions and eras, such as tin in ancient Syracuse, iron in Sparta, cattle in Rome, and other unique forms of currency in various parts of the world.

Question 4.2

Answer:

For current prepare_texts function:

1. Processes the text sentence-by-sentence using `sent_tokenize`.
2. Has more robust token exclusion criteria.
3. Introduces a frequency threshold for vocabulary inclusion.
4. Handles OOV words by introducing an <oov> token.
5. Uses different names for the word-to-index and index-to-word dictionaries.

Question 4.3

Answer:

```
Number of words in the text: 62255
Size of the filtered vocabulary: 2569

Top 20 most frequent words:
['the', 'of', 'be', 'and', 'in', 'to', 'a', 'for', 'as', 'by', 'he', 'with', 'coin', 'this', 'on', 'his', 'which', 'at', 'it', 'from']
```

The word ‘coin’ is unique to the subject of this particular text.

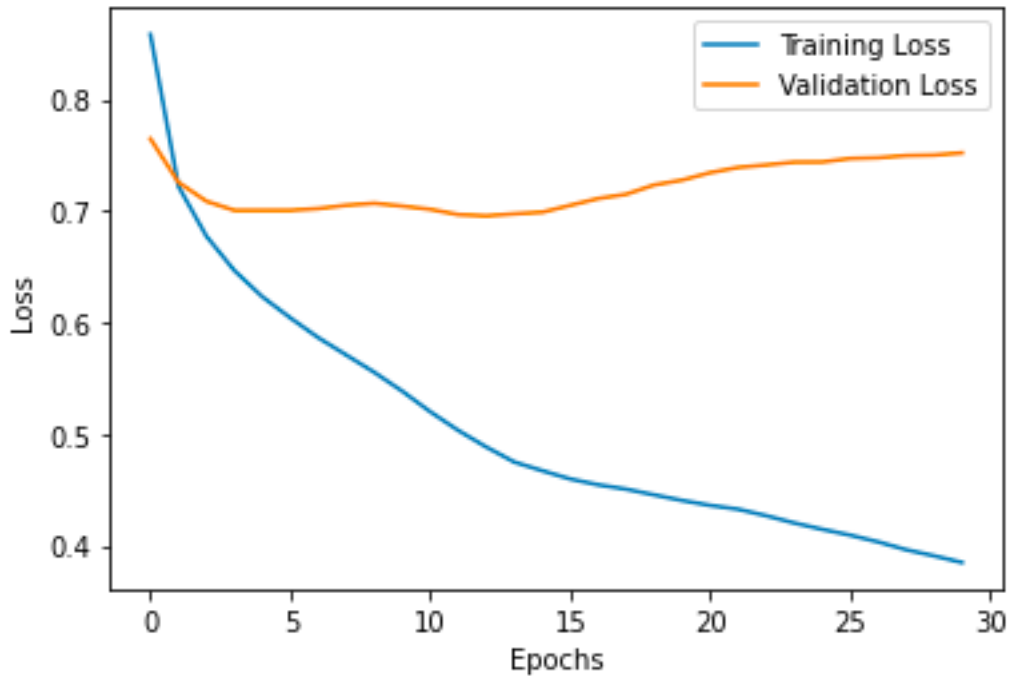
Question 4.4

```
Total number of examples (both positive and negative): 996040
First 10 target tokens: [706, 706, 706, 706, 706, 706, 706, 706, 82, 82]
First 10 context tokens: [82, 1, 0, 2568, 8, 51, 0, 2568, 706, 1]
First 10 labels (1 for positive, -1 for negative): [1, 1, 1, 1, -1, -1, -1, -1, 1, 1]
Total number of examples: 996040
```

Question 4.5

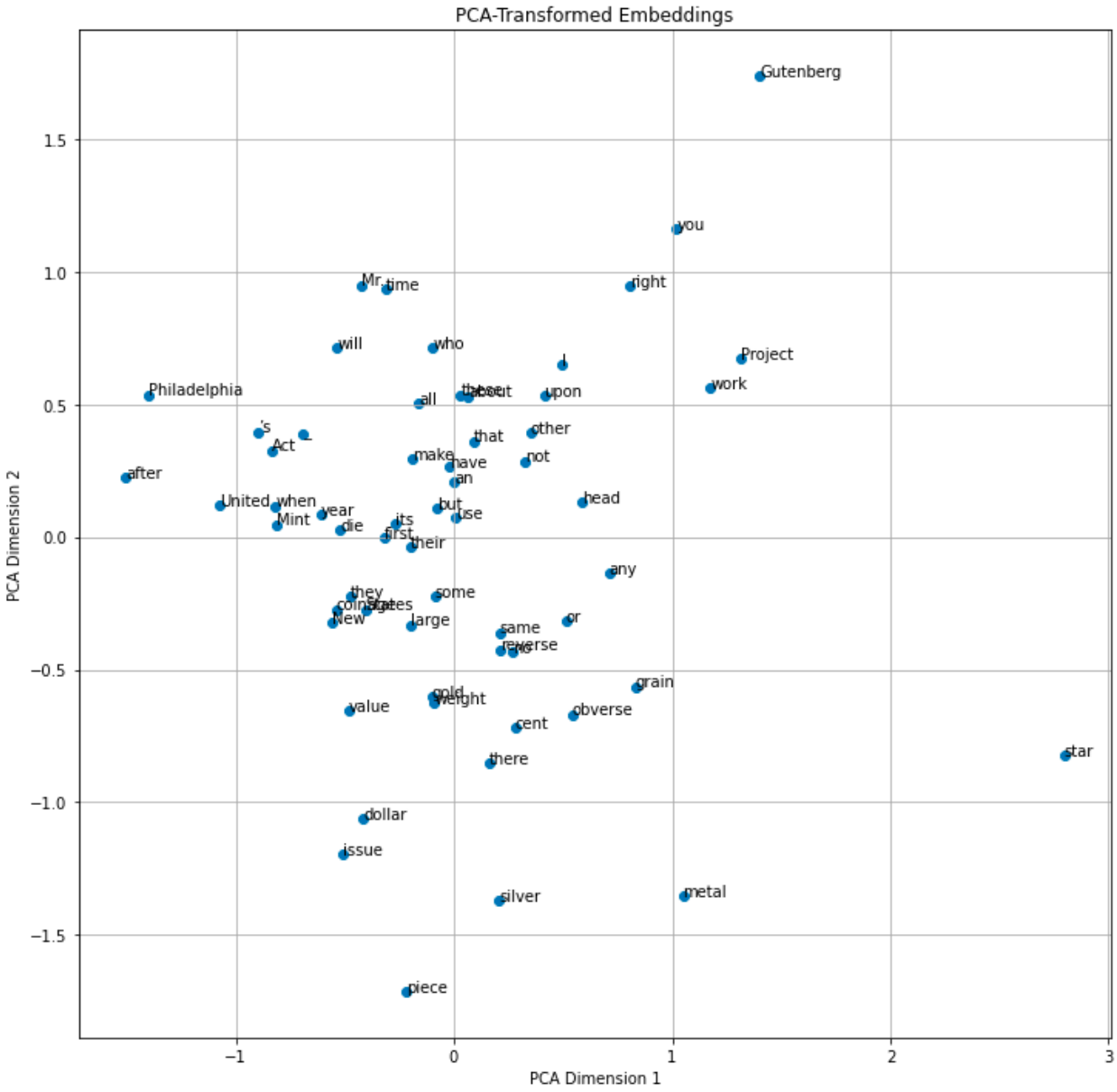
```
Total number of examples (both positive and negative): 132200
Total number of examples after subsampling: 132200
```

Question 4.7



I admit the result is weird. But I have change so many learning rate. The results are all weird.But the model seems like work well in Question 4.8

Question 4.8



The related words are close. The embedding seems to work good.