For this homework, make sure that you format your notbook nicely and cite all sources in the appropriate sections. Programmatically generate or embed any figures or graphs that you need.

Names: Calvin Zikakis, Sarah Schwallier

Section 1: Word2Vec paper questions

```
1) Describe how a CBOW word embedding is generated.
```

CBOW word embeddings are generated by using an unsupervised deep learning algorithm. This architecture creates an input using the context of each word and then the model tries to predict the word that corresponds to the context.

```
2) What is a CBOW word embedding and how is it different than a skip-gram wo
rd embedding?
```

CBOW uses inputs to predict outputs by using a set number of words before and or after the word it is trying to predict. This method of prediction relys on the context of the word. Skip-grams use a given target input to predict the context word. CBOWs and Skip-grams models are reflections of each other in the sense that CBOW is predicting a word from a context while a skip-gram is predicting a context from a word.

```
3) What is the task that the authors use to evaluate the generated word embe
ddings?
```

Authors want to make sure that their generated word embeddings are as accurate as possible using semantic questions. Where the ideal model has a high complexity and is able to predict against an independent data set with high accuracy.

```
4) What are PCA and t-SNE? Why are these important to the task of training a
nd interpreting word embeddings?
```

T-SNE are multi-dimensional word embeddings consisting of word test set sentences that are based upon probability. PCA on the other hand is computed by using matrices and is based on more mathematical approaches. Both of these models are trying to reduce the dimensionality of matrices and vertices to compute a graph. These are important for training and interpreting word embeddings because they both visualize the data that was computed in a way such that the people interpreting the results can analyze the data easily.

Sources Cited

Efficient Estimation of Word Representations in Vector Space by Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean: https://arxiv.org/pdf/1301.3781.pdf (https://arxiv.org/pdf/1301.3781.pdf)

J. Schler, M. Koppel, S. Argamon and J. Pennebaker (2006). Effects of Age and Gender on Blogging in Proceedings of 2006 AAAI Spring Symposium on Computational Approaches for Analyzing Weblogs.

SENTENCE ORDERING USING RECURRENT NEURAL NETWORKS by Lajanugen Logeswaran, Honglak Lee & Dragomir Radev Speech and Language Processing

Karani, Dhruvil, Introduction to Word Embedding and Word2Vec, https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa (https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa)

Benjamin Fayyazuddin Ljungberg, Dimensionality reduction for bag-of-words models: PCA vs LSA

An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition by Daniel Jurafskv and James H. Martin

Section 2: Training your own word embeddings

The spooky authors dataset consists of excerpts from authors of horror novels including Edgar Allan Poe, Mary Shelley, and HP Lovecraft. These excerpts each have a unique ID as well as a three letter tag describing which author wrote the excerpt. The data is split into a training set and a test set. The test set is lacking the three letter code which labels the author.

We are using the The Blog Authorship Corpus for our secondary dataset. We decided on this dataset as it is comprised of 681,288 posts from 19,320 bloggers. We scanned through this database and pulled a small chunk of the total amount of posts. This was to reduce the overall size of the dataset to help with performance in training word embedding. This dataset will provide a data that is written with a style simular to normal human conversation simularly to the spooky authors dataset. This should help insure our generated sentences have a natural sound to them.

In [1]:

```
# import your libraries here
import numpy as np
import sklearn
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from collections import Counter

import keras
from keras import backend as K
import tensorflow as tf

### Comment out this section if you running on a laptop
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.75
session = tf.Session(config=config)
K.set_session(session)
### ---------------------------------------------------

from keras.layers import Dense, Activation, Flatten, SimpleRNN
from keras.layers.recurrent import LSTM
from keras.layers.embeddings import Embedding
from keras.models import Sequential
from keras.utils import to_categorical
from keras.models import load_model


import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
import itertools
import seaborn as sns
import csv


%matplotlib inline
```

```
Using TensorFlow backend.
C:\Users\Calvin\anaconda3\lib\site-packages\tensorflow\python\framework
\dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synony
m of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorflow\python\framework
\dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synony
m of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorflow\python\framework
\dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synony
m of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorflow\python\framework
\dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synony
m of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorflow\python\framework
\dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synony
m of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorflow\python\framework
\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synony
m of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorboard\compat\tensorfl
ow_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be
understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorboard\compat\tensorfl
ow_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be
understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorboard\compat\tensorfl
ow_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be
understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorboard\compat\tensorfl
ow_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be
understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorboard\compat\tensorfl
ow_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be
understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Calvin\anaconda3\lib\site-packages\tensorboard\compat\tensorfl
```

```
ow_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be
understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

```
In [2]: # ------------------Secondary Dataset Formatting and Trimming------------
-------
# This cell trims and fixes the secondary dataset to get the data in a w
orkable style
import re
from csv import reader


def format_secondaryDataset(training_file_path, output_file, sentence_le
ngth):
    '''
    this function takes the dataset and splits it to sentences and store
s those in a txt file

    training_file_path = filepath of blogposts.csv
    output_file = outputfile name (.txt)
    sentence_length = minimum length sentences to grab (value is how man
y words per sentences)
    '''


    with open(training_file_path, "r", encoding="utf-8") as file:
        sentences = file.readlines()
    #open file

    file.close()

    output = open(output_file, "w+")

    count = 0

    for line in reader(sentences):
        blog_post = line[6]
        #Line[6] contains the blog post

        if count >= 7:
        #skip the stuff in the beggining. It's unneeded

            sentences = blog_post.split(".")
            #split the post on the periods to extract individual sentenc
es

            for sentence in sentences:
            #loop over our list of sentences

                if sentence != "":
                #some blog posts contain '...'. This creates empty sente
nces. We don't want empty sentences

                    #lets clean the sentence of symbols and make it all
 lowercase

                    res = re.sub(' +', ' ', sentence)
                    res.strip('\t')
                    res.strip('\n')
                    #strip tabs and newlines
```

```
                    lower = res.lower()
                    #make all lower case

                    whitelist = set('abcdefghijklmnopqrstuvwxyz 12345678
90')
                    no_numbers_punct = ''.join(filter(whitelist.__contai
ns__, lower))
                    #gets rid of punctuation

                    cleaned = no_numbers_punct.split()

                    black_list = ['urllink']
                    #allows us to remove all 'urlLink' occurances

                    if len(cleaned) >= sentence_length:
                        #adjust 4 if you only want longer sentences
                        #we are only concerned with sentences longer tha
n 4 words
                        output.write(" ".join([i for i in cleaned if i n
ot in black_list]) + "\n")


        if count == 2000:
        #Do not need this full dataset... It's 800mb's
            break

        count += 1


format_secondaryDataset("blogtext.csv", "secondaryDataset.txt", 5)
```

In [22]:
```
# code to train your word embeddings
from csv import reader
from gensim.models import Word2Vec

EMB = 300


def convert_data(data):
#flattens data to 1D matrix
    data_flattened = []

    for sentences in data:
        for word in sentences:
            data_flattened.append(word)

    return data_flattened



def standardize_length(words,length):
    counter = 0
    output = []

    sentence = []
    for word in words:
        if counter < length:
            sentence.append(word)
        else:
            output.append(sentence)
            sentence = []
            counter = -1
        counter += 1

    return output

def convert_to_UNK(words):
    output = []
    counts = Counter(words)
    for word in words:
        if counts[word] <= 1:
            output.append('UNK')
        else:
            output.append(word)
    return output



# -----------------Primary Dataset-------------------

def Clean_data_primary_dataset(training_file_path):
    #This function tokenizes the primary dataset and returns a cleaned v
ersion where each word making up a sentence is a nested list inside a la
rger list of the corpus
    output_list = []

    with open(training_file_path, "r", encoding="utf-8") as file:
```

```
        sentences = file.readlines()
    #open file
    file.close()

    count = 0

    for line in reader(sentences):

        if count != 0:
        #don't want first sentence

            sentence = line[1]

            lower = sentence.lower()
            #make all lower case

            whitelist = set('abcdefghijklmnopqrstuvwxyz 1234567890')
            no_numbers_punct = ''.join(filter(whitelist.__contains__, lo
wer))
            #gets rid of punctuation

            cleaned = no_numbers_punct.split()

            output_list.append(cleaned)

        count += 1

        if count >= 10000: #added this so I could test part 4 with a sma
ller dataset
            break         #added this so I could test part 4 with a smal
ler dataset

    return output_list


pri_Dataset = convert_data(Clean_data_primary_dataset("train.csv"))
#imports and cleans dataset

output_pri = convert_to_UNK(pri_Dataset)

sentences_primaryDataset = standardize_length(output_pri, 45)

model_primaryDataset = Word2Vec(sentences_primaryDataset, min_count=1, s
ize=EMB, window=4, negative=10, iter=10, workers=4)
#creates word2vec model

#print(model_primaryDataset)
#model summary

words_primaryDataset = list(model_primaryDataset.wv.vocab)
print(len(words_primaryDataset), "<--- Primary Vocab Length")
#shows the vocab

#print(model_primaryDataset['sentence'])
#our model

# -----------------Secondary Dataset------------------
```

```
#secondary dataset is stored as 'secondaryDataset.txt' after processing
 it

def tokenize_secondary_dataset(training_file_path):
    #tokenizes the secondary dataset and returns a cleaned version where
each word making up a sentence is a nested list inside a larger list of
 the corpus

    output_list = []

    with open(training_file_path) as file:
        sentences = file.readlines()

    #open file
    file.close()

    count = 0
    for sentence in sentences:
    #loop over sentences

        words = sentence.split()
        #split sentences on the words

        output_list.append(words)
        #append words list to final output

        count += 1

        if count >= 10000: #added this so I could test part 4 with a sma
ller dataset
            break         #added this so I could test part 4 with a smal
ler dataset


    return output_list


sec_Dataset = convert_data(tokenize_secondary_dataset("secondaryDataset.
txt"))
#secondary sentences

output_sec = convert_to_UNK(sec_Dataset)

sentences_secondaryDataset = standardize_length(output_sec, 45)

model_secondaryDataset = Word2Vec(sentences_secondaryDataset, min_count=
1, size=EMB, window=4, negative=10, iter=10, workers=4)
#creates word2vec model

#print(model_secondaryDataset)
#model summary

words_secondaryDataset = list(model_secondaryDataset.wv.vocab)
#print(words_secondaryDataset)
#shows the vocab
```

```
#print(model_secondaryDataset['sentence'])
#our model
```

```
11286 <--- Primary Vocab Length
```

Sources Cited

An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition by Daniel Jurafsky and James H. Martin

Brownlee, Jason. “How to Develop Word Embeddings in Python with Gensim.” Machine Learning Mastery, 7 Aug. 2019, machinelearningmastery.com/develop-word-embeddings-python-gensim/.

Section 3: Evaluate the differences between the word embeddings

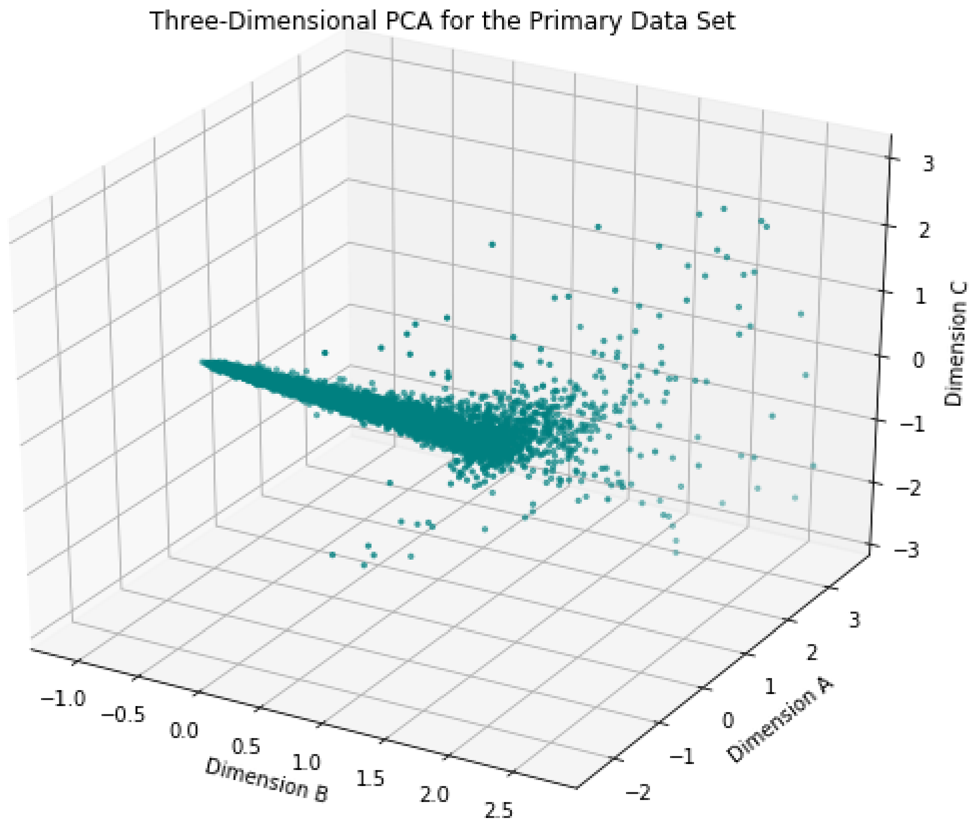
(make sure to include graphs, figures, and paragraphs with full sentences)

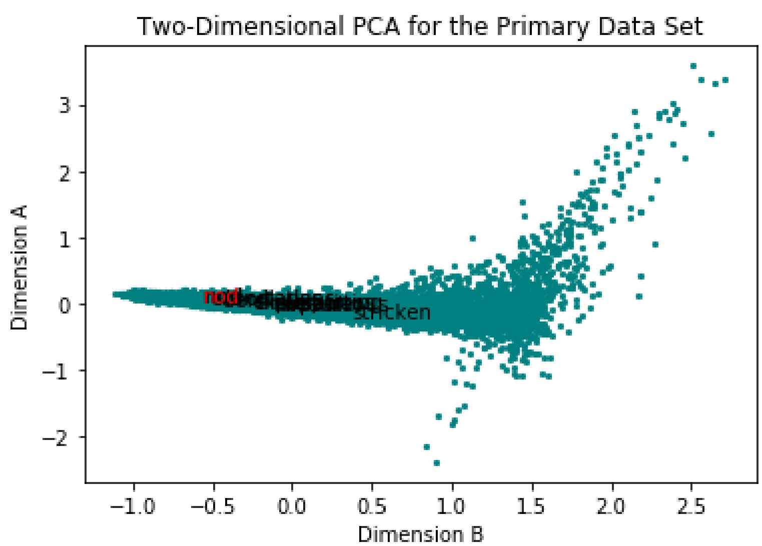
```
In [4]: #This section is evaluating via PCAs
primaryModel = model_primaryDataset[model_primaryDataset.wv.vocab]
secondaryModel = model_secondaryDataset[model_secondaryDataset.wv.vocab]
#Retirives the vectors from each embedding

def buildSimilarWords(randWord, pSimilarWords, sSimilarWords, words_Data
set):
    psList = []
    psList.append(randWord)
    for wordTuple in pSimilarWords:
        word  = wordTuple[0]
        if word not in words_Dataset:
            psList.append(word)
    return psList

def getPrimaryIndex(word):
    for i, iWord in enumerate(list(model_primaryDataset.wv.vocab)):
        if word == iWord:
            return i

randIndex = np.random.randint(0, high=len(words_primaryDataset))
randWord = words_primaryDataset[randIndex]
while randWord not in words_secondaryDataset:
    randIndex = np.random.randint(0, high=len(words_primaryDataset))
    randWord = words_primaryDataset[randIndex]
pSimilarWords = model_primaryDataset.wv.most_similar(randWord)
sSimilarWords = model_secondaryDataset.wv.most_similar(randWord)
print("Word: ", randWord, "\n")
print(pSimilarWords, "\n")
print(sSimilarWords, "\n")
similarWordsPrimary = buildSimilarWords(randWord, pSimilarWords, sSimila
rWords, words_secondaryDataset)
similarWordsSecondary = buildSimilarWords(randWord, sSimilarWords, pSimi
larWords, words_primaryDataset)
dnpWord = model_secondaryDataset.wv.doesnt_match(similarWordsPrimary)
dnsWord = model_secondaryDataset.wv.doesnt_match(similarWordsSecondary)
print(dnpWord)
print(dnsWord)

pcaP = PCA(n_components=3)
resultP = pcaP.fit_transform(primaryModel)
ax = plt.figure(figsize=(10,8)).gca(projection='3d')
ax.scatter(resultP[:, 0], resultP[:, 1], resultP[:, 2], s=5, color='tea
l')
words_primaryDataset = list(model_primaryDataset.wv.vocab)
ax.set_title('Three-Dimensional PCA for the Primary Data Set')
ax.set_xlabel('Dimension B')
ax.set_ylabel('Dimension A')
ax.set_zlabel('Dimension C')
plt.show()
#PCA model for the primary dataset
pcaP = PCA(n_components=2)
resultP = pcaP.fit_transform(primaryModel)
plt.scatter(resultP[:, 0], resultP[:, 1], s=5, color='teal')
words_primaryDataset = list(model_primaryDataset.wv.vocab)
plt.title('Two-Dimensional PCA for the Primary Data Set')
```

```
plt.xlabel('Dimension B')
plt.ylabel('Dimension A')
plt.annotate(randWord, xy=(resultP[randIndex, 0], resultP[randIndex, 1
]),  fontweight='bold')
for word in similarWordsPrimary:
    if word != dnpWord and word != randWord:
        p2 = getPrimaryIndex(word)
        plt.annotate(word, xy=(resultP[p2, 0], resultP[p2, 1]))
p2 = words_primaryDataset.index(dnpWord)
plt.annotate(dnpWord, xy=(resultP[p2, 0], resultP[p2, 1]), color='red')
plt.show()
#PCA model for the primary dataset

randIndex = np.random.randint(0, high=len(words_secondaryDataset))
randWord = words_secondaryDataset[randIndex]
while randWord not in words_secondaryDataset:
    randIndex = np.random.randint(0, high=len(words_secondaryDataset))
    randWord = words_secondaryDataset[randIndex]
pSimilarWords = model_secondaryDataset.wv.most_similar(randWord)
sSimilarWords = model_secondaryDataset.wv.most_similar(randWord)
print("Word: ", randWord, "\n")
print(pSimilarWords, "\n")
print(sSimilarWords, "\n")
similarWordsPrimary = buildSimilarWords(randWord, pSimilarWords, sSimila
rWords, words_secondaryDataset)
similarWordsSecondary = buildSimilarWords(randWord, sSimilarWords, pSimi
larWords, words_primaryDataset)
dnpWord = model_secondaryDataset.wv.doesnt_match(similarWordsPrimary)
dnsWord = model_secondaryDataset.wv.doesnt_match(similarWordsSecondary)
print(dnpWord)
print(dnsWord)

def getSecondaryIndex(word):
    for i, iWord in enumerate(list(model_secondaryDataset.wv.vocab)):
        if word == iWord:
            return i
pcaS = PCA(n_components=3)
resultS = pcaS.fit_transform(secondaryModel)
ax = plt.figure(figsize=(10,8)).gca(projection='3d')
ax.scatter(resultS[:, 0], resultS[:, 1], resultS[:, 2], s=5, color='cora
l')
words_secondaryDataset = list(model_secondaryDataset.wv.vocab)
ax.set_title('Three-Dimensional PCA for the Secondary Data Set')
ax.set_xlabel('Dimension B')
ax.set_ylabel('Dimension A')
ax.set_zlabel('Dimension C')
plt.show()

#PCA model for the secondary dataset
pcaS = PCA(n_components=2)
resultS = pcaS.fit_transform(secondaryModel)
plt.scatter(resultS[:, 0], resultS[:, 1], s=5, color='coral')
words_secondaryDataset = list(model_secondaryDataset.wv.vocab)
plt.title('Two-Dimensional PCA for the Secondary Data Set')
plt.xlabel('Dimension B')
plt.ylabel('Dimension A')
plt.annotate(randWord, xy=(resultS[randIndex, 0], resultS[randIndex, 1
```

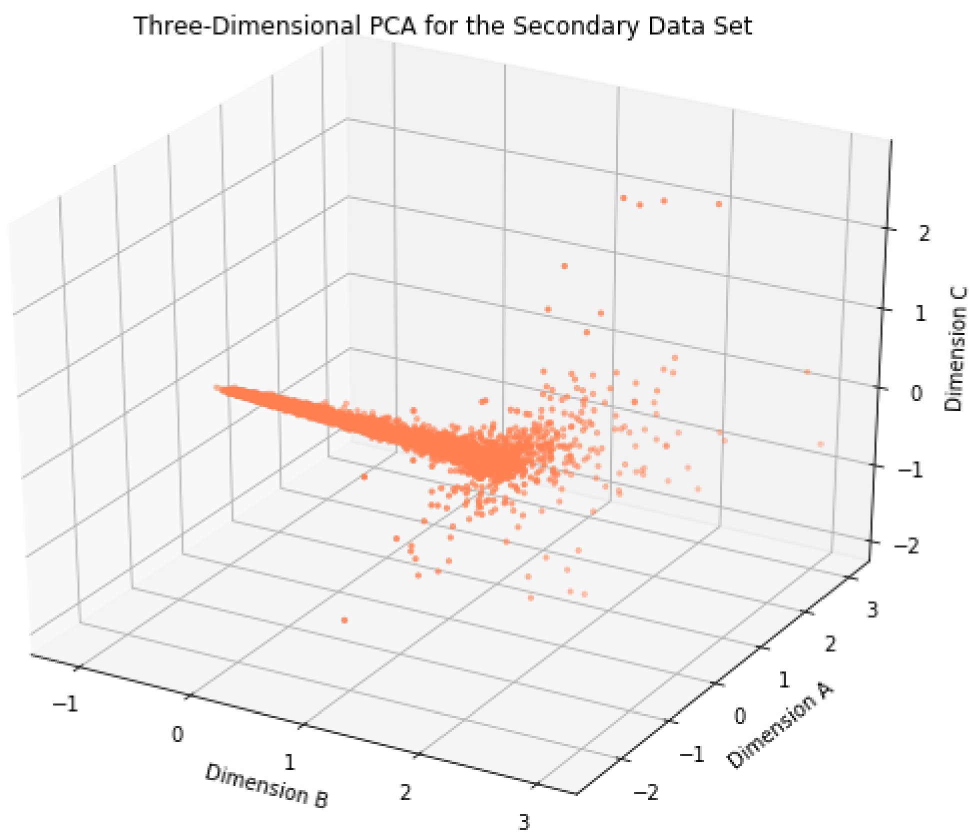
```
]),  fontweight='bold')
for word in similarWordsSecondary:
    if word != dnsWord and word != randWord:
        s2 = getSecondaryIndex(word)
        plt.annotate(word, xy=(resultS[s2, 0], resultS[s2, 1]))
s2 = words_secondaryDataset.index(dnsWord)
plt.annotate(dnsWord, xy=(resultS[s2, 0], resultS[s2, 1]), color='darkor
chid')
plt.show()
#PCA model for the secondary dataset
```

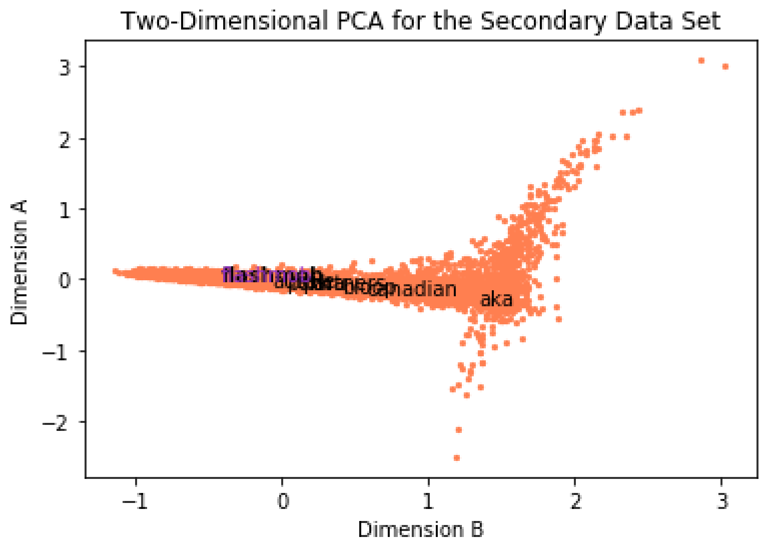
```
C:\Users\Calvin\anaconda3\lib\site-packages\ipykernel_launcher.py:2: De
precationWarning: Call to deprecated `__getitem__` (Method will be remo
ved in 4.0.0, use self.wv.__getitem__() instead).

C:\Users\Calvin\anaconda3\lib\site-packages\ipykernel_launcher.py:3: De
precationWarning: Call to deprecated `__getitem__` (Method will be remo
ved in 4.0.0, use self.wv.__getitem__() instead).
  This is separate from the ipykernel package so we can avoid doing imp
orts until
C:\Users\Calvin\anaconda3\lib\site-packages\gensim\models\keyedvectors.
py:877: FutureWarning: arrays to stack must be passed as a "sequence" t
ype such as list or tuple. Support for non-sequence iterables such as g
enerators is deprecated as of NumPy 1.16 and will raise an error in the
future.
  vectors = vstack(self.word_vec(word, use_norm=True) for word in used_
words).astype(REAL)

Word:  nod

[('supporting', 0.998641848564148), ('landladys', 0.9985164999961853),
('exhortations', 0.9983779191970825), ('losing', 0.9983762502670288),
('tiers', 0.9983314871788025), ('tarpaulin', 0.9982854127883911), ('bea
uteous', 0.9982617497444153), ('clime', 0.9982333183288574), ('explorat
ions', 0.9982278347015381), ('stricken', 0.9982208013534546)]

[('glass', 0.9992753267288208), ('four', 0.9992489814758301), ('experie
nce', 0.9992433190345764), ('pile', 0.9992285966873169), ('buses', 0.99
91940259933472), ('block', 0.999173641204834), ('company', 0.9991206526
756287), ('memories', 0.9991083741188049), ('blue', 0.999076962471008
3), ('member', 0.999075174331665)]

nod
nod
```


Three-Dimensional PCA for the Primary Data Set
Dimension C
Dimension A
Dimension B
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Two-Dimensional PCA for the Primary Data Set
Dimension A
Dimension B
rod
stricken
3
2
1
0
−1
−2
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

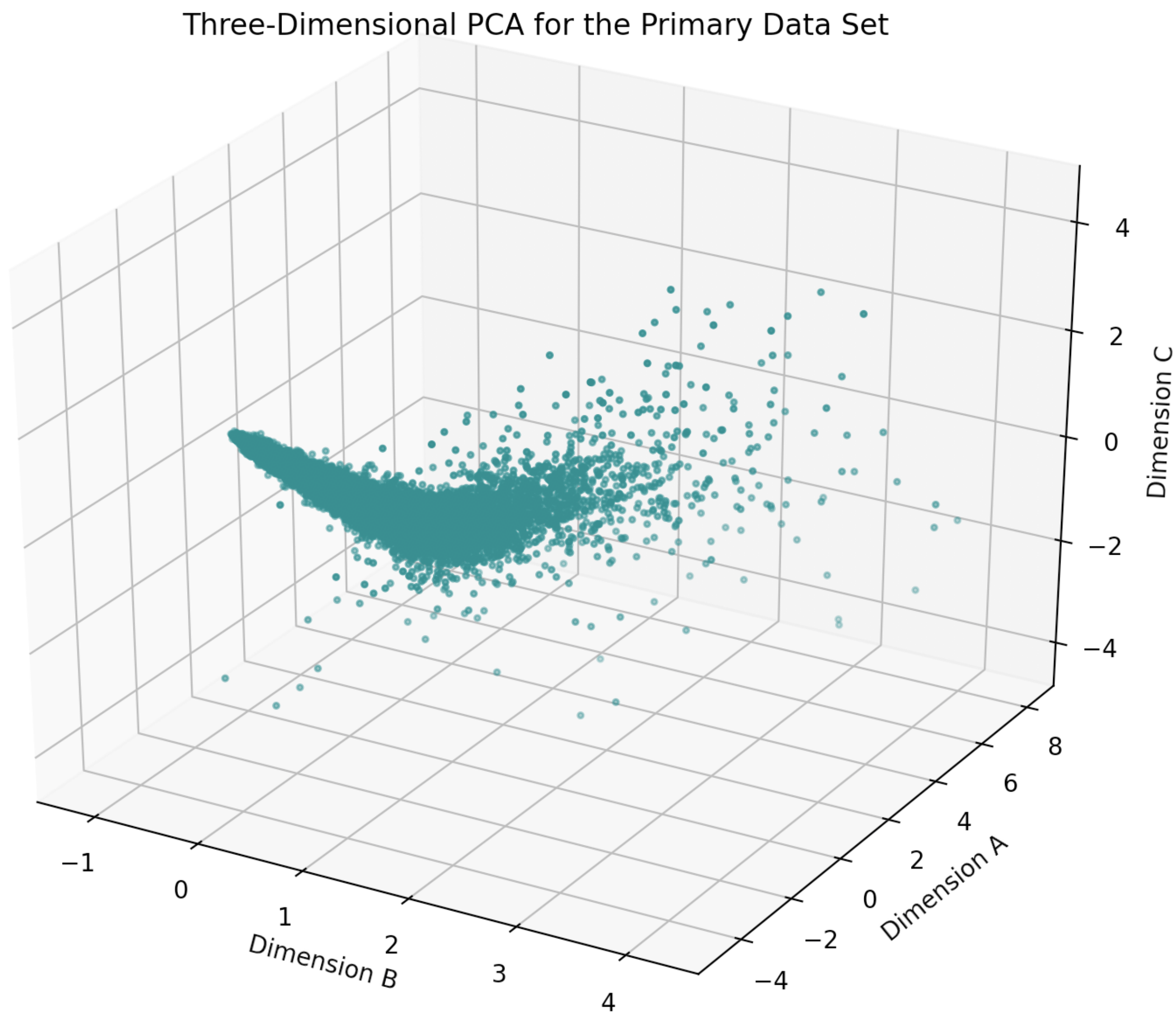
```
Word:  flashmob

[('aussie', 0.9994242191314697), ('canadian', 0.9994099736213684), ('ak
a', 0.9993980526924133), ('alpha', 0.999387264251709), ('partners', 0.9
993863105773926), ('bloop', 0.9993857741355896), ('pork', 0.99938243627
54822), ('wise', 0.9993759393692017), ('miles', 0.9993746280670166),
('cha', 0.9993675351142883)]

[('aussie', 0.9994242191314697), ('canadian', 0.9994099736213684), ('ak
a', 0.9993980526924133), ('alpha', 0.999387264251709), ('partners', 0.9
993863105773926), ('bloop', 0.9993857741355896), ('pork', 0.99938243627
54822), ('wise', 0.9993759393692017), ('miles', 0.9993746280670166),
('cha', 0.9993675351142883)]

flashmob
flashmob

C:\Users\Calvin\anaconda3\lib\site-packages\gensim\models\keyedvectors.
py:877: FutureWarning: arrays to stack must be passed as a "sequence" t
ype such as list or tuple. Support for non-sequence iterables such as g
enerators is deprecated as of NumPy 1.16 and will raise an error in the
future.
  vectors = vstack(self.word_vec(word, use_norm=True) for word in used_
words).astype(REAL)
```

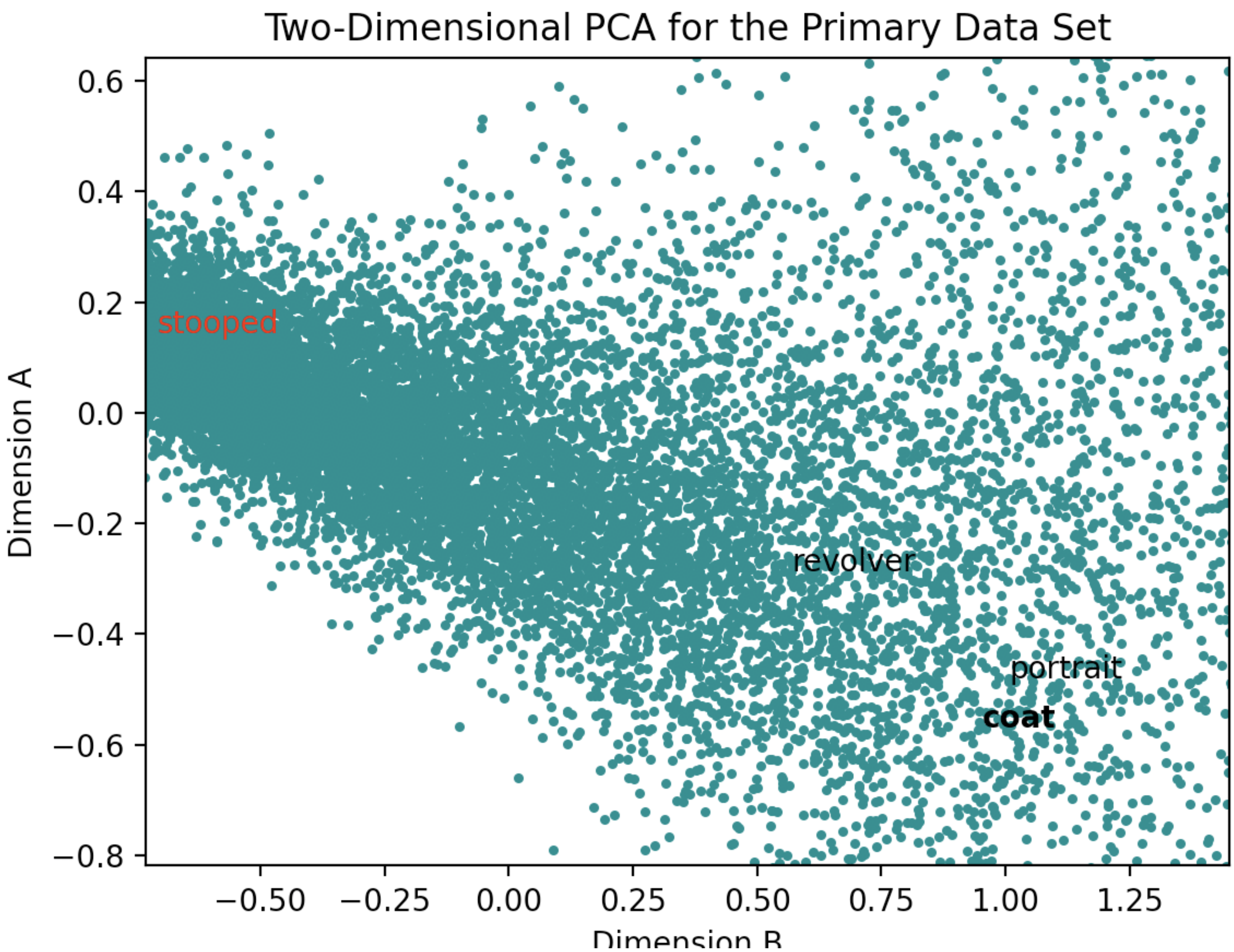




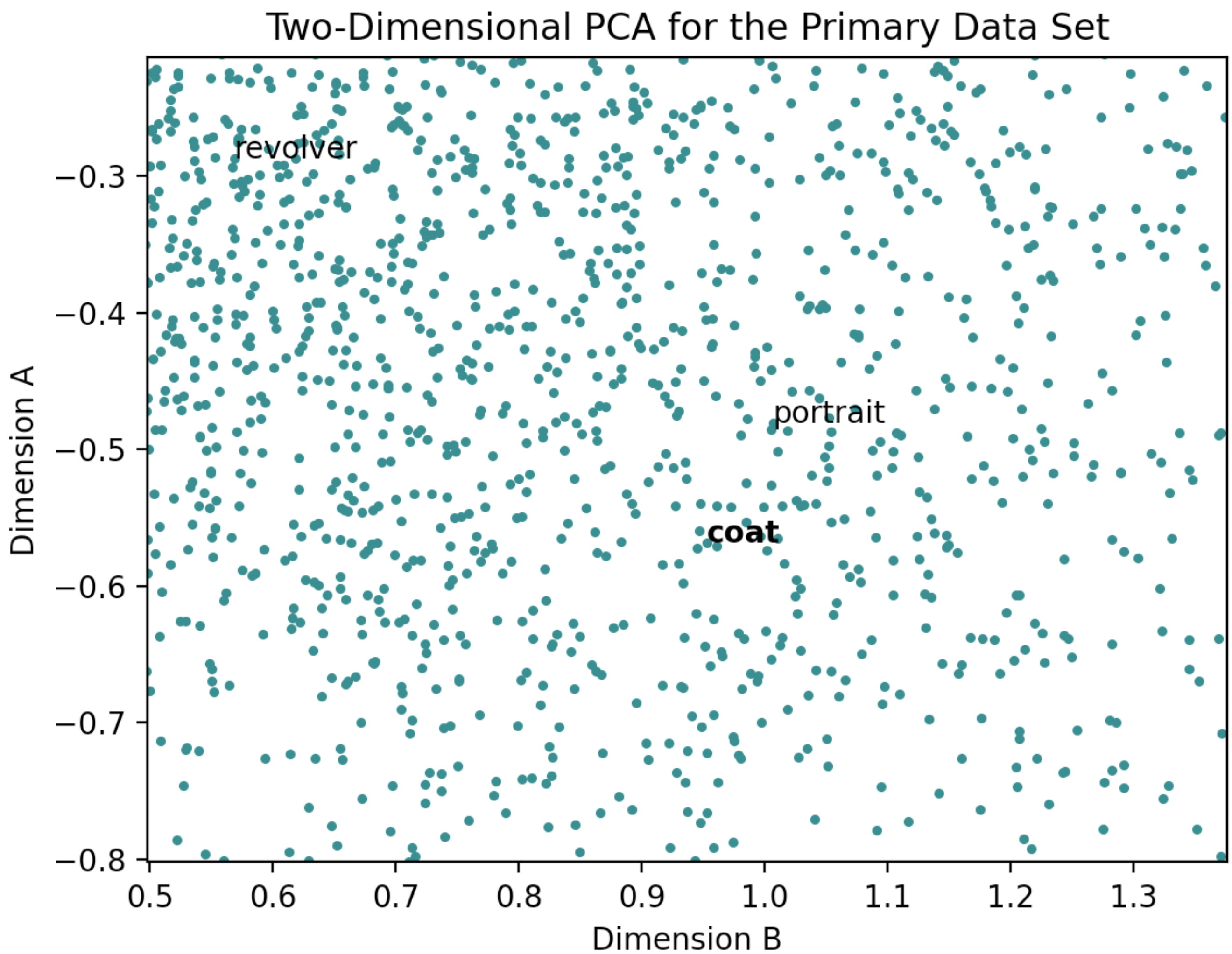
The analysis below is using the following images and information:

Primary Dataset											
Word selected	Similar words										Word that matched the least
coat	pillow	revolver	hat	stooped	throwing	portrait	concentrate	shoulders	bumper	leaning	stooped
	0.972	0.9667	0.9667	0.965	0.9635	0.9623	0.9611	0.9601	0.9597	0.9596	

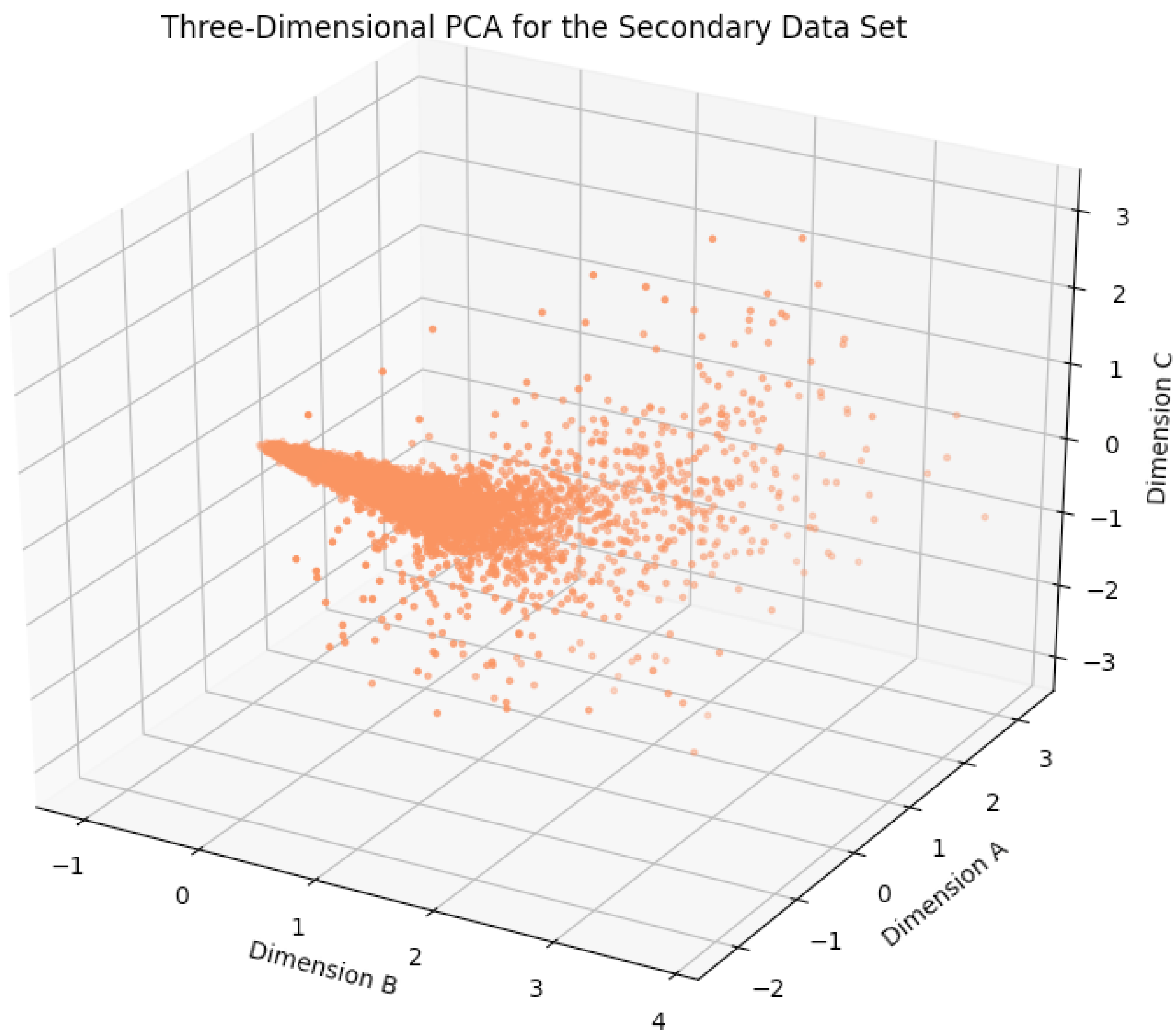




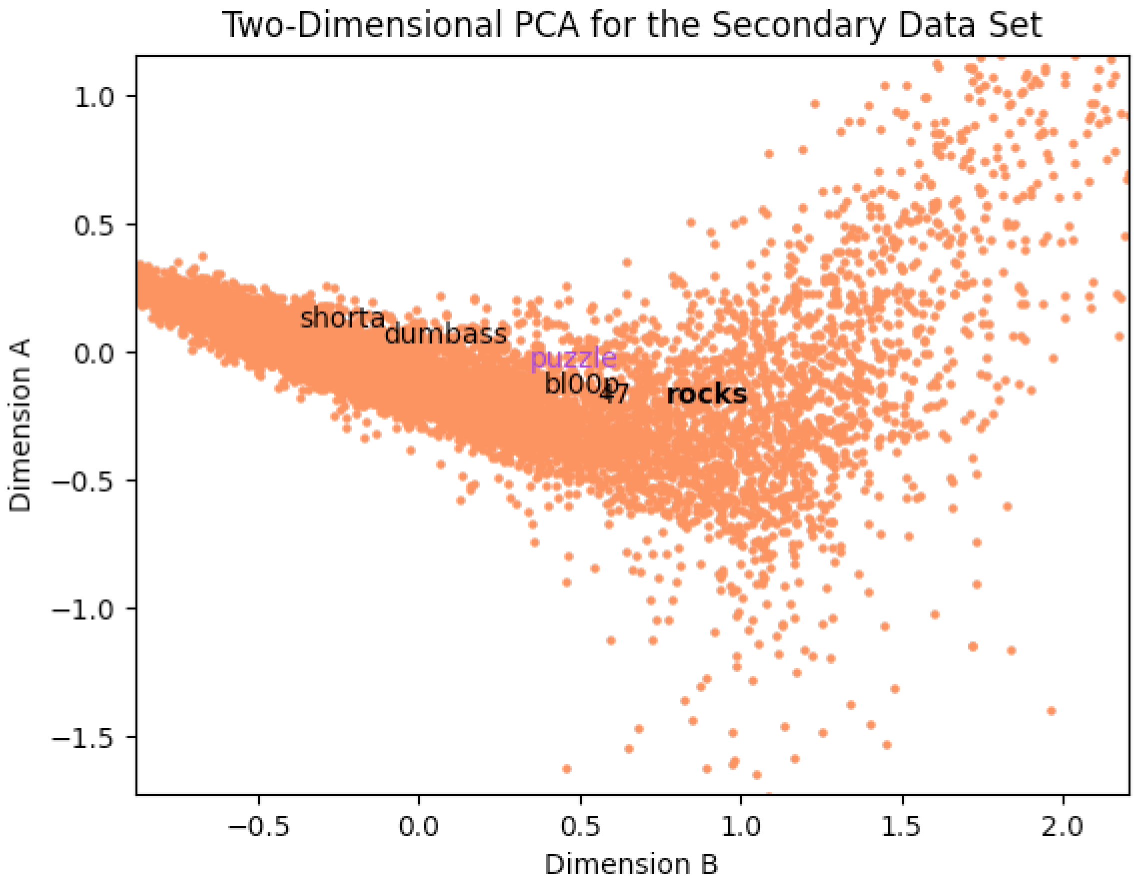
Two-Dimensional PCA for the Primary Data Set
Dimension A
Dimension B
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
stooped
revolver
portrait
coat



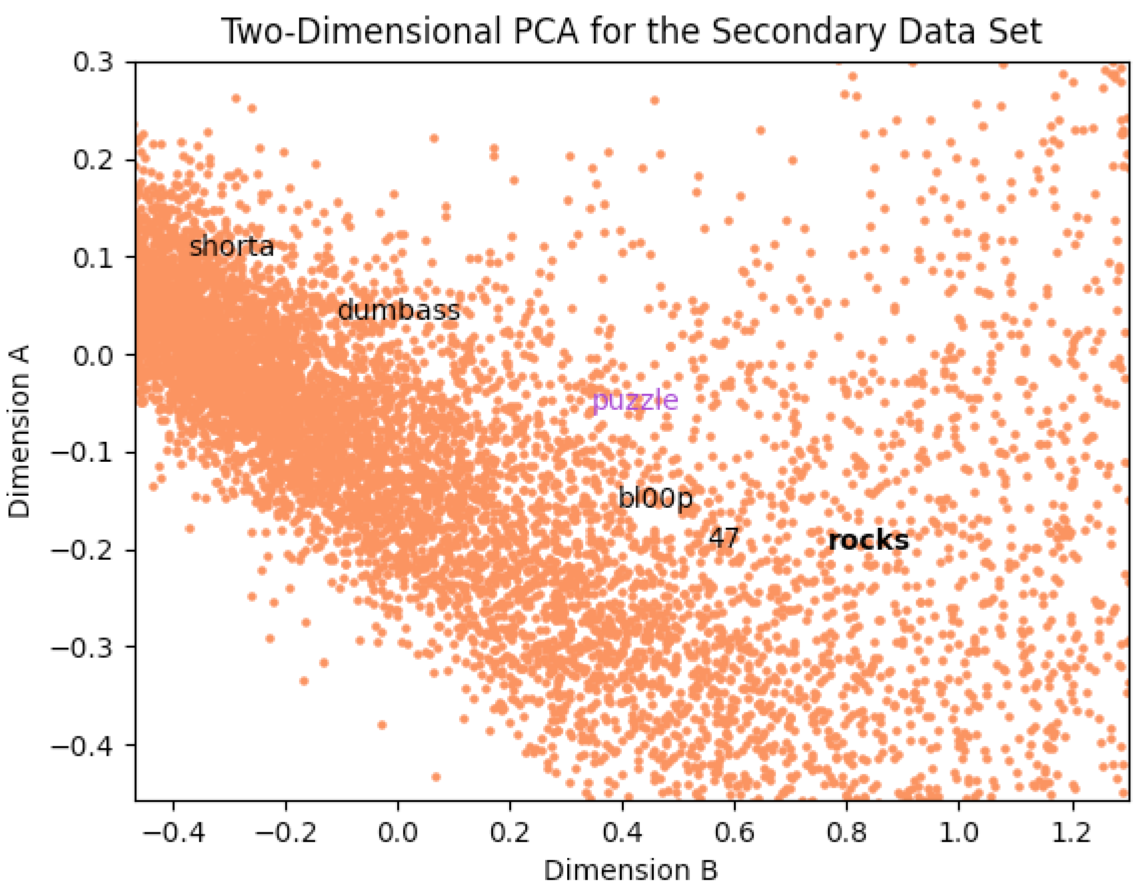
Secondary Dataset											
Word selected	Similar words										Word that matched the least
rocks	dumbass	shorta	endless	fits	pot	puzzle	47	marry	deliver	bl00p	puzzle
	0.9956	0.9947	0.9945	0.9945	0.9941	0.9939	0.9938	0.9937	0.9934	0.9936	



Three-Dimensional PCA for the Secondary Data Set
Dimension C
Dimension A
Dimension B
3
2
1
0
−1
−2
−3
−1
0
1
2
3
4



Two-Dimensional PCA for the Secondary Data Set
Dimension A
Dimension B
1.0
0.5
0.0
−0.5
−1.0
−1.5
−0.5
0.0
0.5
1.0
1.5
2.0
shorta
dumbass
puzzle
bl00p
47
rocks



Two-Dimensional PCA for the Secondary Data Set
Dimension A
Dimension B
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
shorta
dumbass
puzzle
bl00p
47
rocks

From the charts and diagrams above one can see that both datasets produced similar looking results that contain different data and meaning. The results above were computed by having a random index that corresponds with a word. The word selected for the primary data set was 'coat' and for the secondary was 'rocks', both words are bolded in the 2D graphs above. If the word in the primary dataset was in the secondary dataset then it moved onto the next step which was finding the closest 10 words that a word2vec embedding matched with the randomly selected word. For the secondary it did the same thing but instead it checked if the selected word was in the primary dataset. From there each list of similar words was then evalulated against) their own list to find the word that did not match the other words in the list. For the primary dataset we found out that the word 'stooped' (in red) did not match the other ten and for the secondary dataset it was the word 'puzzle' (in purple). Then a 3D and 2D graph was made to reflect the results. The primary was graphed in teal and the secondary was graphed in coral.

In comparing the primary dataset which is composed of the supplied assignment horror corpus against our blog corpus which was smaller. It is evident in the spread of the PCAs that the dataset has more vertices for the 'cloud' in the graphs is more dense and full. While the secondary dataset has a smaller density area with more outliers. We see these differences between the two datasets because of the size of each corpus and the variety of words within the corpi. The primary dataset was larger which led to the results being more accurate.

Sources Cited

An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition by Daniel Jurafsky and James H. Martin Brownlee, Jason. “How to Develop Word Embeddings in Python with Gensim.” Machine Learning Mastery, 7 Aug. 2019, machinelearningmastery.com/develop-word-embeddings-python-gensim/.

Durksen, Luuk. "Visualising high-dimensional datasets using PCA and t-SNE in Python" 29 Oct. 2016, https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b (https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b)

Section 4: Feedforward Neural Language Model

In [5]:
```
# code to train a feedforward neural language model
# on a set of given word embeddings
# make sure not to just copy + paste to train your two

vec = 300

def convert_data(data):
#flattens data to 1D matrix
    data_flattened = []

    for sentences in data:
        for word in sentences:
            data_flattened.append(word)

    return data_flattened


def data_to_index(data, model):
#assigns index values to data
    data_index = []

    for word in data:
        if word in model:
            data_index.append(model.vocab[word].index)

    return data_index


#before we can train, break down the dataset into sections. this will re
sult in nested arrays of length 100 where each nest contains 100 words.
 This will lower memory requirements
def section_data(sentences):
    section = []
    output = []

    i = 0

    for sentence in sentences:

        i += 1
        section.append(sentence)

        if i % 2000 == 0:
            output.append(section)
            section = []

    return output


def training_data(sentences, model):
    sentence_length = 20

    sections_X = []
    sections_y = []

    for section in sentences:
```

```
        x_train = []
        y_train = []

        #=======--------Why only train on the first four words in a sent
ence???-------=======
        for sentence in section:

            index1 = model.wv.vocab[sentence[0]].index
            index2 = model.wv.vocab[sentence[1]].index
            index3 = model.wv.vocab[sentence[2]].index
            index_label = model.wv.vocab[sentence[3]].index

            training_data_x = np.concatenate((model.wv.vectors[index1],
model.wv.vectors[index2], model.wv.vectors[index3]))
            x_train.append(training_data_x)

            y_train.append(index_label)

        x_train = np.asarray(x_train)
        y_train = np.asarray(y_train)

        sections_X.append(x_train)
        sections_y.append(y_train)

    return sections_X, sections_y

def train(x_train, y_train, model, model_Dataset):
    count = 1
    for (section_x, section_y) in zip(x_train, y_train):

        y_labels = to_categorical(section_y, num_classes=len(model_Datas
et.wv.vocab), dtype='int16')

        print("Training batch: ", count, " out of ", len(x_train), ".
 :)")
        model.fit(section_x, y_labels, batch_size=10)



        count += 1

    return model
```

In [6]:
```
#-------------------------------Primary Dataset-------------------------
------
# Wouldn't even attempt to run this unless on computer with a GPU and lo
ts of ram.
# Its eating 32gb's of system ram and 6gb's of vram

sentences_primary = section_data(sentences_primaryDataset)

print(np.asarray(sentences_primary[0]).shape)

print("-------------Formatting test data-------------")

x_train_primary, y_train_primary = training_data(sentences_primary, mode
l_primaryDataset)

#Create Keras Model

print("-------------Building Model-------------")

primary_FFNN = Sequential()

primary_FFNN.add(Dense(units=10000, input_shape=(900,)))
primary_FFNN.add(Dense(units=len(model_primaryDataset.wv.vocab),activati
on="softmax"))
primary_FFNN.compile(optimizer="adam", loss='mean_squared_error')

primary_FFNN.summary()

print("-------------Training Model-------------")

primary_FFNN = train(x_train_primary, y_train_primary, primary_FFNN, mod
el_primaryDataset)
```

```
(2000, 4)
-------------Formatting test data-------------
-------------Building Model-------------
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10000)             9010000
_________________________________________________________________
dense_2 (Dense)              (None, 11130)             111311130
=================================================================
Total params: 120,321,130
Trainable params: 120,321,130
Non-trainable params: 0
_________________________________________________________________
-------------Training Model-------------
Training batch:  1  out of  26 .  :)
WARNING:tensorflow:From C:\Users\Calvin\anaconda3\lib\site-packages\ker
as\backend\tensorflow_backend.py:422: The name tf.global_variables is d
eprecated. Please use tf.compat.v1.global_variables instead.

Epoch 1/1
2000/2000 [==============================] - 18s 9ms/step - loss: 8.983
9e-05
Training batch:  2  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.983
9e-05
Training batch:  3  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.983
9e-05
Training batch:  4  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.983
9e-05
Training batch:  5  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.983
8e-05
Training batch:  6  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.983
8e-05
Training batch:  7  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.983
7e-05
Training batch:  8  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.983
6e-05
Training batch:  9  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.983
5e-05
Training batch:  10  out of  26 .  :)
```

```
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.983
2e-05
Training batch:  11  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.977
6e-05
Training batch:  12  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.953
6e-05
Training batch:  13  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.944
6e-05
Training batch:  14  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.999
6e-05
Training batch:  15  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.983
5e-05
Training batch:  16  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.983
5e-05
Training batch:  17  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.983
4e-05
Training batch:  18  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.983
3e-05
Training batch:  19  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.983
1e-05
Training batch:  20  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.982
8e-05
Training batch:  21  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.982
5e-05
Training batch:  22  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.979
9e-05
Training batch:  23  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 9ms/step - loss: 8.938
2e-05
Training batch:  24  out of  26 .  :)
Epoch 1/1
```

```
2000/2000 [==============================] - 17s 9ms/step - loss: 8.958
6e-05
Training batch:  25  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.932
1e-05
Training batch:  26  out of  26 .  :)
Epoch 1/1
2000/2000 [==============================] - 17s 8ms/step - loss: 8.959
7e-05
```

In [7]:
```
def generate_words_FFNN(word2vec_model, keras_model, words_list, length=
12):
    words = []
    word_indexs = []
    for word in words_list:
        word_indexs.append(word2vec_model.wv.vocab[word].index)
        words.append(word)

    for i in range(length):

        word_data_x = 0

        word_vectors = []

        index1 = word_indexs[-3]
        index2 = word_indexs[-2]
        index3 = word_indexs[-1]

        word_data_x = np.concatenate((word2vec_model.wv.vectors[index1],
word2vec_model.wv.vectors[index2], word2vec_model.wv.vectors[index3]))

        word_vectors.append(word_data_x)

        pred = keras_model.predict(x=np.asarray(word_vectors), verbose=0
) #added verbose
        pred = pred[0] #an array of arrays?

        vocab = list(word2vec_model.wv.vocab)

        vocab_index = [word2vec_model.wv.vocab[i].index for i in vocab]

        vocab_index = np.asarray(vocab_index)

        prediction = np.random.choice(vocab_index, p=pred, replace=True)
#added p= for pred

        word_indexs.append(prediction)

        index_to_word = word2vec_model.wv.index2word[prediction]

        words.append(index_to_word)

    return(words)
```

In [8]:

```
#Primary Dataset
print("-------------Generating Words-------------")

words = generate_words_FFNN(model_primaryDataset, primary_FFNN, ["horse"
, "seemed", "to"])
print(words)

sentences_secondary = section_data(sentences_secondaryDataset)

print(np.asarray(sentences_secondary[0]).shape)

print("-------------Formatting test data-------------")

x_train_secondary, y_train_secondary = training_data(sentences_secondary
, model_secondaryDataset)
```

```
-------------Generating Words-------------
['horse', 'seemed', 'to', 'theyre', 'engagement', 'darted', 'abysmal',
'proving', 'peculiarity', 'widely', 'wood', 'counterfeit', 'shaft', 'sp
ell', 'towers']
(2000, 4)
-------------Formatting test data-------------
```

In [9]:
```
#-------------------------------Secondary Dataset-----------------------
--------
# Wouldn't even attempt to run this unless on computer with a GPU and lo
ts of ram.
# Its eating 32gb's of system ram and 6gb's of vram

#Create Keras Model

print("-------------Building Model-------------")

secondary_FFNN = Sequential()

secondary_FFNN.add(Dense(units=10000, input_shape=(900,)))
secondary_FFNN.add(Dense(units=len(model_secondaryDataset.wv.vocab),acti
vation="softmax"))
secondary_FFNN.compile(optimizer="adam", loss='mean_squared_error')

secondary_FFNN.summary()

print("-------------Training Model-------------")

secondary_FFNN = train(x_train_secondary, y_train_secondary, secondary_F
FNN, model_secondaryDataset)
```

```
-------------Building Model-------------
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 10000)             9010000
_________________________________________________________________
dense_4 (Dense)              (None, 7310)              73107310
=================================================================
Total params: 82,117,310
Trainable params: 82,117,310
Non-trainable params: 0
_________________________________________________________________
-------------Training Model-------------
Training batch:  1  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.367
8e-04
Training batch:  2  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.367
8e-04
Training batch:  3  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.367
8e-04
Training batch:  4  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.367
8e-04
Training batch:  5  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.367
7e-04
Training batch:  6  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.367
7e-04
Training batch:  7  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.367
2e-04
Training batch:  8  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.366
3e-04
Training batch:  9  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.365
1e-04
Training batch:  10  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.363
7e-04
Training batch:  11  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.365
```

```
5e-04
Training batch:  12  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.364
1e-04
Training batch:  13  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.362
0e-04
Training batch:  14  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.365
0e-04
Training batch:  15  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.365
0e-04
Training batch:  16  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.362
8e-04
Training batch:  17  out of  17 .  :)
Epoch 1/1
2000/2000 [==============================] - 12s 6ms/step - loss: 1.363
6e-04
```

In [10]:

```
print("-------------Generating Words-------------")

words = generate_words_FFNN(model_secondaryDataset, secondary_FFNN, ["th
is", "is", "not"])
print(words)
```

```
-------------Generating Words-------------
['this', 'is', 'not', 'attached', 'clock', 'cops', 'ohio', 'hold', 'avo
iding', 'made', 'differently', 'favorites', 'nation', 'fried', 'mysteri
es']
```

Sources Cited

An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition by Daniel Jurafsky and James H. Martin https://keras.io/models/model/ (https://keras.io/models/model/)

Section 5: Recurrent Neural Language Model

In [23]:
```
def training_data_RNN(sentences, model):
    sentence_length = 45

    x_train = np.zeros([len(sentences), sentence_length], dtype=np.int32
)
    y_train = np.zeros([len(sentences)], dtype=np.int32)

    for i, sentence in enumerate(sentences):
        for j, word in enumerate(sentence[:-1]):
            x_train[i, j] = model.wv.vocab[word].index
        y_train[i] = model.wv.vocab[sentence[-1]].index

    return x_train, y_train



sentences_primary = sentences_primaryDataset

x_train_primary, y_train_primary = training_data_RNN(sentences_primary,
model_primaryDataset)




sentences_secondary = sentences_secondaryDataset

x_train_secondary, y_train_secondary = training_data_RNN(sentences_secon
dary, model_secondaryDataset)



print('train_x shape:', x_train_primary.shape)
print(x_train_primary)
print('train_y shape:', y_train_primary.shape)



def generate_words_RNN(word2vec_model, keras_model, words_list, length=1
2):
    words = []
    word_indexs = []
    for word in words_list:
        word_indexs.append(word2vec_model.wv.vocab[word].index)
        words.append(word)

    for i in range(length):
        word_index_array = np.array(word_indexs)
        pred = keras_model.predict(x=word_index_array)

        pred = pred[0] #an array of arrays?

        vocab = list(word2vec_model.wv.vocab)

        vocab_index = [word2vec_model.wv.vocab[i].index for i in vocab]

        vocab_index = np.asarray(vocab_index)
```

```
        prediction = np.random.choice(vocab_index, p=pred, replace=True)
#added p= for pred

        word_indexs.append(prediction)

        index_to_word = word2vec_model.wv.index2word[prediction]

        words.append(index_to_word)

    return(words)


vec = 300
```

```
train_x shape: (5790, 45)
[[  26 3435  143 ...  109  123    0]
 [  22    9    0 ...  308  440    0]
 [ 650   25  667 ...   15 2012    0]
 ...
 [   0    3  527 ...  111  178    0]
 [ 283    2   39 ... 8104    4    0]
 [   0  320  148 ...    2  933    0]]
train_y shape: (5790,)
```

In [24]:

```
# code to train a recurrent neural language model
# on a set of given word embeddings
# make sure not to just copy + paste to train your two

#------------------ Primary Dataset -----------------


#Create Keras Model

trained_weights_primaryDataset = model_primaryDataset.wv.vectors
vocab_size_primaryDataset, embedding_size_primaryDataset = trained_weigh
ts_primaryDataset.shape

primary_RNN = Sequential()
primary_RNN.add(Embedding(input_dim=vocab_size_primaryDataset, output_di
m=embedding_size_primaryDataset, weights=[trained_weights_primaryDataset
], trainable=False))
primary_RNN.add(SimpleRNN(units=embedding_size_primaryDataset))
primary_RNN.add(Dense(units=vocab_size_primaryDataset))
primary_RNN.add(Activation('softmax'))

primary_RNN.compile(optimizer='adam', loss='sparse_categorical_crossentr
opy')

primary_RNN.summary()

primary_RNN.fit(x_train_primary, y_train_primary, batch_size=128, epochs
=100)
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, None, 300)         3385800   
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 300)               180300    
_________________________________________________________________
dense_7 (Dense)              (None, 11286)             3397086   
_________________________________________________________________
activation_3 (Activation)    (None, 11286)             0         
=================================================================
Total params: 6,963,186
Trainable params: 3,577,386
Non-trainable params: 3,385,800
_________________________________________________________________
Epoch 1/100
5790/5790 [==============================] - 1s 142us/step - loss: 7.64
53
Epoch 2/100
5790/5790 [==============================] - 1s 108us/step - loss: 6.46
98
Epoch 3/100
5790/5790 [==============================] - 1s 99us/step - loss: 6.372
2
Epoch 4/100
5790/5790 [==============================] - 1s 107us/step - loss: 6.24
25
Epoch 5/100
5790/5790 [==============================] - 1s 97us/step - loss: 6.054
3
Epoch 6/100
5790/5790 [==============================] - 1s 98us/step - loss: 5.838
4
Epoch 7/100
5790/5790 [==============================] - 1s 100us/step - loss: 5.60
08
Epoch 8/100
5790/5790 [==============================] - 1s 102us/step - loss: 5.34
92
Epoch 9/100
5790/5790 [==============================] - 1s 91us/step - loss: 5.087
3
Epoch 10/100
5790/5790 [==============================] - 1s 91us/step - loss: 4.833
1
Epoch 11/100
5790/5790 [==============================] - 1s 90us/step - loss: 4.568
1
Epoch 12/100
5790/5790 [==============================] - 1s 92us/step - loss: 4.308
1
Epoch 13/100
5790/5790 [==============================] - 1s 90us/step - loss: 4.049
9
Epoch 14/100
5790/5790 [==============================] - 1s 90us/step - loss: 3.783
```

```
9
Epoch 15/100
5790/5790 [==============================] - 1s 92us/step - loss: 3.527
0
Epoch 16/100
5790/5790 [==============================] - 1s 92us/step - loss: 3.272
1
Epoch 17/100
5790/5790 [==============================] - 1s 92us/step - loss: 3.025
1
Epoch 18/100
5790/5790 [==============================] - 1s 94us/step - loss: 2.785
6
Epoch 19/100
5790/5790 [==============================] - 1s 89us/step - loss: 2.566
8
Epoch 20/100
5790/5790 [==============================] - 1s 92us/step - loss: 2.344
0
Epoch 21/100
5790/5790 [==============================] - 1s 88us/step - loss: 2.152
2
Epoch 22/100
5790/5790 [==============================] - 1s 93us/step - loss: 1.971
2
Epoch 23/100
5790/5790 [==============================] - 1s 90us/step - loss: 1.810
0
Epoch 24/100
5790/5790 [==============================] - 1s 90us/step - loss: 1.653
6
Epoch 25/100
5790/5790 [==============================] - 1s 88us/step - loss: 1.508
2
Epoch 26/100
5790/5790 [==============================] - 1s 90us/step - loss: 1.380
7
Epoch 27/100
5790/5790 [==============================] - 1s 88us/step - loss: 1.292
8
Epoch 28/100
5790/5790 [==============================] - 1s 89us/step - loss: 1.178
8
Epoch 29/100
5790/5790 [==============================] - 1s 89us/step - loss: 1.084
5
Epoch 30/100
5790/5790 [==============================] - 1s 93us/step - loss: 1.021
1
Epoch 31/100
5790/5790 [==============================] - 1s 94us/step - loss: 0.904
4
Epoch 32/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.827
2
Epoch 33/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.788
```

```
0
Epoch 34/100
5790/5790 [==============================] - 1s 91us/step - loss: 0.719
1
Epoch 35/100
5790/5790 [==============================] - 1s 90us/step - loss: 0.648
8
Epoch 36/100
5790/5790 [==============================] - 1s 88us/step - loss: 0.586
0
Epoch 37/100
5790/5790 [==============================] - 1s 94us/step - loss: 0.533
5
Epoch 38/100
5790/5790 [==============================] - 1s 95us/step - loss: 0.489
6
Epoch 39/100
5790/5790 [==============================] - 1s 96us/step - loss: 0.483
5
Epoch 40/100
5790/5790 [==============================] - 1s 94us/step - loss: 0.486
0
Epoch 41/100
5790/5790 [==============================] - 1s 97us/step - loss: 0.416
7
Epoch 42/100
5790/5790 [==============================] - 1s 94us/step - loss: 0.428
0
Epoch 43/100
5790/5790 [==============================] - 1s 96us/step - loss: 0.475
2
Epoch 44/100
5790/5790 [==============================] - 1s 96us/step - loss: 0.371
4
Epoch 45/100
5790/5790 [==============================] - 1s 95us/step - loss: 0.309
9
Epoch 46/100
5790/5790 [==============================] - 1s 98us/step - loss: 0.262
2
Epoch 47/100
5790/5790 [==============================] - 1s 95us/step - loss: 0.226
6
Epoch 48/100
5790/5790 [==============================] - 1s 99us/step - loss: 0.201
3
Epoch 49/100
5790/5790 [==============================] - 1s 94us/step - loss: 0.173
9
Epoch 50/100
5790/5790 [==============================] - 1s 96us/step - loss: 0.159
4
Epoch 51/100
5790/5790 [==============================] - 1s 96us/step - loss: 0.156
2
Epoch 52/100
5790/5790 [==============================] - 1s 95us/step - loss: 0.147
```

```
1
Epoch 53/100
5790/5790 [==============================] - 1s 95us/step - loss: 0.128
4
Epoch 54/100
5790/5790 [==============================] - 1s 94us/step - loss: 0.113
0
Epoch 55/100
5790/5790 [==============================] - 1s 92us/step - loss: 0.103
9
Epoch 56/100
5790/5790 [==============================] - 1s 88us/step - loss: 0.095
3
Epoch 57/100
5790/5790 [==============================] - 1s 88us/step - loss: 0.087
3
Epoch 58/100
5790/5790 [==============================] - 1s 88us/step - loss: 0.081
8
Epoch 59/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.078
1
Epoch 60/100
5790/5790 [==============================] - 1s 93us/step - loss: 0.121
9
Epoch 61/100
5790/5790 [==============================] - 1s 91us/step - loss: 0.350
3
Epoch 62/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.690
7
Epoch 63/100
5790/5790 [==============================] - 1s 92us/step - loss: 0.510
6
Epoch 64/100
5790/5790 [==============================] - 1s 92us/step - loss: 0.434
3
Epoch 65/100
5790/5790 [==============================] - 1s 91us/step - loss: 0.302
1
Epoch 66/100
5790/5790 [==============================] - 1s 90us/step - loss: 0.193
7
Epoch 67/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.145
1
Epoch 68/100
5790/5790 [==============================] - 1s 90us/step - loss: 0.131
7
Epoch 69/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.136
9
Epoch 70/100
5790/5790 [==============================] - 1s 88us/step - loss: 0.090
7
Epoch 71/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.072
```

```
5
Epoch 72/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.055
2
Epoch 73/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.046
3
Epoch 74/100
5790/5790 [==============================] - 1s 90us/step - loss: 0.043
1
Epoch 75/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.038
6
Epoch 76/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.035
0
Epoch 77/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.032
1
Epoch 78/100
5790/5790 [==============================] - 1s 88us/step - loss: 0.030
8
Epoch 79/100
5790/5790 [==============================] - 1s 90us/step - loss: 0.028
6
Epoch 80/100
5790/5790 [==============================] - 1s 89us/step - loss: 0.026
9
Epoch 81/100
5790/5790 [==============================] - 1s 88us/step - loss: 0.025
6
Epoch 82/100
5790/5790 [==============================] - 1s 90us/step - loss: 0.024
3
Epoch 83/100
5790/5790 [==============================] - 1s 91us/step - loss: 0.023
3
Epoch 84/100
5790/5790 [==============================] - 1s 96us/step - loss: 0.022
3
Epoch 85/100
5790/5790 [==============================] - 1s 94us/step - loss: 0.021
4
Epoch 86/100
5790/5790 [==============================] - 1s 95us/step - loss: 0.020
6
Epoch 87/100
5790/5790 [==============================] - 1s 96us/step - loss: 0.019
8
Epoch 88/100
5790/5790 [==============================] - 1s 97us/step - loss: 0.019
0
Epoch 89/100
5790/5790 [==============================] - 1s 95us/step - loss: 0.018
3
Epoch 90/100
5790/5790 [==============================] - 1s 99us/step - loss: 0.017
```

```
7
Epoch 91/100
5790/5790 [==============================] - 1s 103us/step - loss: 0.01
71
Epoch 92/100
5790/5790 [==============================] - 1s 97us/step - loss: 0.016
5
Epoch 93/100
5790/5790 [==============================] - 1s 98us/step - loss: 0.015
9
Epoch 94/100
5790/5790 [==============================] - 1s 95us/step - loss: 0.015
4
Epoch 95/100
5790/5790 [==============================] - 1s 95us/step - loss: 0.014
9
Epoch 96/100
5790/5790 [==============================] - 1s 96us/step - loss: 0.014
4
Epoch 97/100
5790/5790 [==============================] - 1s 99us/step - loss: 0.014
0
Epoch 98/100
5790/5790 [==============================] - 1s 96us/step - loss: 0.013
5
Epoch 99/100
5790/5790 [==============================] - 1s 96us/step - loss: 0.013
1
Epoch 100/100
5790/5790 [==============================] - 1s 95us/step - loss: 0.012
7
```

Out[24]:
```
<keras.callbacks.callbacks.History at 0x1b51a139948>
```

In [25]:
```
test = ["this", "should", "work"]
words = generate_words_RNN(model_primaryDataset,primary_RNN,test)


print(words)
```

```
['this', 'should', 'work', 'afforded', 'make', 'this', 'bottom', 'mak
e', 'process', 'hectic', 'to', 'make', 'me', 'once', 'it']
```

In [26]:

```
# code to train a recurrent neural language model
# on a set of given word embeddings
# make sure not to just copy + paste to train your two

#------------------- Secondary Dataset ------------------


#Create Keras Model

trained_weights_secondaryDataset = model_secondaryDataset.wv.vectors
vocab_size_secondaryDataset, embedding_size_secondaryDataset = trained_w
eights_secondaryDataset.shape

secondary_RNN = Sequential()
secondary_RNN.add(Embedding(input_dim=vocab_size_secondaryDataset, outpu
t_dim=embedding_size_secondaryDataset, weights=[trained_weights_secondar
yDataset], trainable=False))
secondary_RNN.add(SimpleRNN(units=embedding_size_secondaryDataset))
secondary_RNN.add(Dense(units=vocab_size_secondaryDataset))
secondary_RNN.add(Activation('softmax'))

secondary_RNN.compile(optimizer='adam', loss='sparse_categorical_crossen
tropy')

secondary_RNN.summary()

secondary_RNN.fit(x_train_secondary, y_train_secondary, batch_size=128,
epochs=100)
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 300)         2223900
_________________________________________________________________
simple_rnn_4 (SimpleRNN)     (None, 300)               180300
_________________________________________________________________
dense_8 (Dense)              (None, 7413)              2231313
_________________________________________________________________
activation_4 (Activation)    (None, 7413)              0
=================================================================
Total params: 4,635,513
Trainable params: 2,411,613
Non-trainable params: 2,223,900
_________________________________________________________________
Epoch 1/100
3815/3815 [==============================] - 1s 133us/step - loss: 7.32
51
Epoch 2/100
3815/3815 [==============================] - 0s 81us/step - loss: 6.182
7
Epoch 3/100
3815/3815 [==============================] - 0s 81us/step - loss: 6.099
3
Epoch 4/100
3815/3815 [==============================] - 0s 83us/step - loss: 6.087
3
Epoch 5/100
3815/3815 [==============================] - 0s 80us/step - loss: 6.072
7
Epoch 6/100
3815/3815 [==============================] - 0s 84us/step - loss: 6.065
7
Epoch 7/100
3815/3815 [==============================] - 0s 82us/step - loss: 6.056
1
Epoch 8/100
3815/3815 [==============================] - 0s 82us/step - loss: 5.995
4
Epoch 9/100
3815/3815 [==============================] - 0s 82us/step - loss: 5.903
4
Epoch 10/100
3815/3815 [==============================] - 0s 82us/step - loss: 5.748
7
Epoch 11/100
3815/3815 [==============================] - 0s 83us/step - loss: 5.573
4
Epoch 12/100
3815/3815 [==============================] - 0s 83us/step - loss: 5.408
7
Epoch 13/100
3815/3815 [==============================] - 0s 81us/step - loss: 5.236
4
Epoch 14/100
3815/3815 [==============================] - 0s 81us/step - loss: 5.058
```

```
0
Epoch 15/100
3815/3815 [==============================] - 0s 82us/step - loss: 4.878
6
Epoch 16/100
3815/3815 [==============================] - 0s 81us/step - loss: 4.693
1
Epoch 17/100
3815/3815 [==============================] - 0s 82us/step - loss: 4.494
6
Epoch 18/100
3815/3815 [==============================] - 0s 81us/step - loss: 4.295
6
Epoch 19/100
3815/3815 [==============================] - 0s 83us/step - loss: 4.101
3
Epoch 20/100
3815/3815 [==============================] - 0s 82us/step - loss: 3.893
2
Epoch 21/100
3815/3815 [==============================] - 0s 82us/step - loss: 3.685
3
Epoch 22/100
3815/3815 [==============================] - 0s 83us/step - loss: 3.480
9
Epoch 23/100
3815/3815 [==============================] - 0s 82us/step - loss: 3.271
3
Epoch 24/100
3815/3815 [==============================] - 0s 81us/step - loss: 3.078
5
Epoch 25/100
3815/3815 [==============================] - 0s 81us/step - loss: 2.881
7
Epoch 26/100
3815/3815 [==============================] - 0s 82us/step - loss: 2.684
8
Epoch 27/100
3815/3815 [==============================] - 0s 81us/step - loss: 2.487
3
Epoch 28/100
3815/3815 [==============================] - 0s 83us/step - loss: 2.307
9
Epoch 29/100
3815/3815 [==============================] - 0s 87us/step - loss: 2.149
8
Epoch 30/100
3815/3815 [==============================] - 0s 86us/step - loss: 1.976
1
Epoch 31/100
3815/3815 [==============================] - 0s 82us/step - loss: 1.825
0
Epoch 32/100
3815/3815 [==============================] - 0s 82us/step - loss: 1.687
4
Epoch 33/100
3815/3815 [==============================] - 0s 82us/step - loss: 1.555
```

```
3
Epoch 34/100
3815/3815 [==============================] - 0s 85us/step - loss: 1.430
9
Epoch 35/100
3815/3815 [==============================] - 0s 83us/step - loss: 1.334
0
Epoch 36/100
3815/3815 [==============================] - 0s 81us/step - loss: 1.233
6
Epoch 37/100
3815/3815 [==============================] - 0s 80us/step - loss: 1.123
8
Epoch 38/100
3815/3815 [==============================] - 0s 85us/step - loss: 1.034
2
Epoch 39/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.942
3
Epoch 40/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.859
4
Epoch 41/100
3815/3815 [==============================] - 0s 83us/step - loss: 0.811
3
Epoch 42/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.729
5
Epoch 43/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.665
5
Epoch 44/100
3815/3815 [==============================] - 0s 87us/step - loss: 0.621
4
Epoch 45/100
3815/3815 [==============================] - 0s 89us/step - loss: 0.569
5
Epoch 46/100
3815/3815 [==============================] - 0s 88us/step - loss: 0.528
3
Epoch 47/100
3815/3815 [==============================] - 0s 87us/step - loss: 0.486
7
Epoch 48/100
3815/3815 [==============================] - 0s 86us/step - loss: 0.440
1
Epoch 49/100
3815/3815 [==============================] - 0s 87us/step - loss: 0.397
1
Epoch 50/100
3815/3815 [==============================] - 0s 88us/step - loss: 0.362
6
Epoch 51/100
3815/3815 [==============================] - 0s 90us/step - loss: 0.340
7
Epoch 52/100
3815/3815 [==============================] - 0s 89us/step - loss: 0.307
```

```
9
Epoch 53/100
3815/3815 [==============================] - 0s 88us/step - loss: 0.310
1
Epoch 54/100
3815/3815 [==============================] - 0s 91us/step - loss: 0.284
3
Epoch 55/100
3815/3815 [==============================] - 0s 89us/step - loss: 0.311
5
Epoch 56/100
3815/3815 [==============================] - 0s 89us/step - loss: 0.295
2
Epoch 57/100
3815/3815 [==============================] - 0s 88us/step - loss: 0.336
0
Epoch 58/100
3815/3815 [==============================] - 0s 87us/step - loss: 0.544
9
Epoch 59/100
3815/3815 [==============================] - 0s 89us/step - loss: 0.445
8
Epoch 60/100
3815/3815 [==============================] - 0s 90us/step - loss: 0.284
6
Epoch 61/100
3815/3815 [==============================] - 0s 88us/step - loss: 0.208
9
Epoch 62/100
3815/3815 [==============================] - 0s 88us/step - loss: 0.178
5
Epoch 63/100
3815/3815 [==============================] - 0s 87us/step - loss: 0.146
1
Epoch 64/100
3815/3815 [==============================] - 0s 89us/step - loss: 0.125
6
Epoch 65/100
3815/3815 [==============================] - 0s 88us/step - loss: 0.112
7
Epoch 66/100
3815/3815 [==============================] - 0s 89us/step - loss: 0.104
9
Epoch 67/100
3815/3815 [==============================] - 0s 90us/step - loss: 0.096
0
Epoch 68/100
3815/3815 [==============================] - 0s 92us/step - loss: 0.089
5
Epoch 69/100
3815/3815 [==============================] - 0s 89us/step - loss: 0.084
2
Epoch 70/100
3815/3815 [==============================] - 0s 87us/step - loss: 0.080
6
Epoch 71/100
3815/3815 [==============================] - 0s 87us/step - loss: 0.129
```

```
1
Epoch 72/100
3815/3815 [==============================] - 0s 89us/step - loss: 0.754
8
Epoch 73/100
3815/3815 [==============================] - 0s 87us/step - loss: 1.056
8
Epoch 74/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.606
9
Epoch 75/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.421
5
Epoch 76/100
3815/3815 [==============================] - 0s 80us/step - loss: 0.350
5
Epoch 77/100
3815/3815 [==============================] - 0s 84us/step - loss: 0.269
6
Epoch 78/100
3815/3815 [==============================] - 0s 86us/step - loss: 0.161
9
Epoch 79/100
3815/3815 [==============================] - 0s 88us/step - loss: 0.118
5
Epoch 80/100
3815/3815 [==============================] - 0s 82us/step - loss: 0.098
5
Epoch 81/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.082
9
Epoch 82/100
3815/3815 [==============================] - 0s 82us/step - loss: 0.072
8
Epoch 83/100
3815/3815 [==============================] - 0s 85us/step - loss: 0.064
3
Epoch 84/100
3815/3815 [==============================] - 0s 83us/step - loss: 0.057
6
Epoch 85/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.052
9
Epoch 86/100
3815/3815 [==============================] - 0s 80us/step - loss: 0.049
3
Epoch 87/100
3815/3815 [==============================] - 0s 82us/step - loss: 0.046
3
Epoch 88/100
3815/3815 [==============================] - 0s 80us/step - loss: 0.043
8
Epoch 89/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.041
7
Epoch 90/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.039
```

```
8
Epoch 91/100
3815/3815 [==============================] - 0s 83us/step - loss: 0.038
1
Epoch 92/100
3815/3815 [==============================] - 0s 80us/step - loss: 0.036
5
Epoch 93/100
3815/3815 [==============================] - 0s 82us/step - loss: 0.035
0
Epoch 94/100
3815/3815 [==============================] - 0s 82us/step - loss: 0.033
7
Epoch 95/100
3815/3815 [==============================] - 0s 80us/step - loss: 0.032
5
Epoch 96/100
3815/3815 [==============================] - 0s 81us/step - loss: 0.031
3
Epoch 97/100
3815/3815 [==============================] - 0s 82us/step - loss: 0.030
2
Epoch 98/100
3815/3815 [==============================] - 0s 82us/step - loss: 0.029
2
Epoch 99/100
3815/3815 [==============================] - 0s 82us/step - loss: 0.028
2
Epoch 100/100
3815/3815 [==============================] - 0s 83us/step - loss: 0.027
3
```

Out[26]:
```
<keras.callbacks.callbacks.History at 0x1b51b1d7e88>
```

In [27]:
```
test = ["this", "should", "work"]
words = generate_words_RNN(model_secondaryDataset,secondary_RNN,test)


print(words)
```

```
['this', 'should', 'work', 'korea', 'korea', 'ladies', 'korea', 'whatev
er', 'korea', 'early', 'korea', 'things', 'korea', 'korea', 'korea']
```

Sources Cited

An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition by Daniel Jurafsky and James H. Martin

“Python Gensim Word2Vec Tutorial with TensorFlow and Keras.” Adventures in Machine Learning, 1 Sept. 2017, adventuresinmachinelearning.com/gensim-word2vec-tutorial/.

Shukla, Vishal ShuklaVishal. “Using Pre-Trained word2vec with LSTM for Word Generation.” Stack Overflow, 1AD, stackoverflow.com/questions/42064690/using-pre-trained-word2vec-with-lstm-for-word-generation.

Section 6: Evaluate the differences between the two language models

(make sure to include graphs, figures, and paragraphs with full sentences)

In [19]:
```
#-----Section 6.1: Evaluating the models' perplexities-----#


pTest_Dataset = Clean_data_primary_dataset("test.csv")
#imports and cleans test dataset, in sentence form

def getWordIndex(model, word):
    index = None
    if word in model.wv.vocab:
        index = model.wv.vocab[word].index
    return index

def calculateSentencePerplexity(model, keras_model, sentence, probs):
    word_vectors = []
    unkVal = 1.0 / len(pTest_Dataset)
    for i in range(4, len(sentence)): #should we be doing all of the pre
ceding words in a sentence instead of 3 (yes?)
        word1 = sentence[i-3]
        word2 = sentence[i-2]
        word3 = sentence[i-1]
        word4 = sentence[i]
        index1 = getWordIndex(model, word1)
        index2 = getWordIndex(model, word2)
        index3 = getWordIndex(model, word3)
        index4 = getWordIndex(model, word4)

        wordProb = 0.0

        if index1 != None and index2 != None and index3 != None and inde
x4 != None:
            test_data_x = np.concatenate((model.wv.vectors[index1], mode
l.wv.vectors[index2], model.wv.vectors[index3]))
            word_vectors.append(test_data_x)

            pred = keras_model.predict(x=np.asarray(word_vectors), verbo
se=0)
            pred = pred[0]
            wordProb = pred[index4]
        else:
            wordProb = 1.0 / len(model.wv.vocab)
        if wordProb == 1.0:
            wordProb -= unkVal
        if wordProb == 0.0:
            wordProb += unkVal
        probs.append(wordProb)

def calculatePerplexity(model, keras_model, pTest_Dataset):
    unkVal = 1.0 / len(pTest_Dataset)
    probs = []
    for sentence in pTest_Dataset:
        sentenceProbabilty = calculateSentencePerplexity(model, keras_mo
del, sentence, probs)
    val = 0.0
    for prob in probs:
        val += np.log2(prob)
    perplexity = np.power(2, -val/len(probs))
```

```python
    return perplexity

#----Primary Dataset Perplexity-----#
print("----Calculating Primary Dataset Perplexity-----")
model = model_primaryDataset
keras_model = primary_FFNN
vocab = list(model.wv.vocab)
vocab_index = np.asarray([model.wv.vocab[i].index for i in vocab])
print("Primary Dataset Perplexity: ", calculatePerplexity(model, keras_m
odel, pTest_Dataset))

#----Secondary Dataset Perplexity-----#
print("----Calculating Secondary Dataset Perplexity-----")
model = model_secondaryDataset
keras_model = secondary_FFNN
vocab = list(model.wv.vocab)
vocab_index = np.asarray([model.wv.vocab[i].index for i in vocab])
print("Secondary Dataset Perplexity: ", calculatePerplexity(model, keras
_model, pTest_Dataset))
```

```
----Calculating Primary Dataset Perplexity-----
Primary Dataset Perplexity:  7115.695240277234
----Calculating Secondary Dataset Perplexity-----
Secondary Dataset Perplexity:  5079.862487864495
```

In [20]:

```
#-----Section 6.2: Generate Random Sentences-----#

phrases = []
for i in range(10):
    sentence = []
    while len(sentence) < 3:
        index = int(np.random.random() * len(pTest_Dataset))
        sentence = pTest_Dataset[index]
        for s in range(3):
            if sentence[s] not in model_primaryDataset.wv.vocab:
                sentence = []
                break
    phrase = []
    for j in range(3):
        phrase.append(sentence[j])
    phrases.append(phrase)


"""
#FFNN  and RNN
for phrase in phrases:
    words = generate_words_FFNN(model_primaryDataset, primary_FFNN, phra
se, length=9)
    sentence = ""
    for word in words:
        sentence += word + " "
    print("FFNN Sentence: ", sentence)
    words = generate_words_RNN(model_primaryDataset, primary_RNN, phras
e, length=9)
    sentence = ""
    for word in words:
        sentence += word + " "
    print("RNN Sentence: ", sentence)
"""



#FFNN
for phrase in phrases:
    words = generate_words_FFNN(model_primaryDataset, primary_FFNN, phra
se, length=9)
    sentence = ""
    for word in words:
        sentence += word + " "
    print("FFNN Sentence: ", sentence)
```

```
FFNN Sentence:  the innumerable blossoms shew assume womanish lowest sk
ies records anxious abysses earlier
FFNN Sentence:  you say that relates excessive carousals dubious others
glimpses disgraced godlike demons
FFNN Sentence:  there was a pang effeminate keys election admire shafts
strength this rushed
FFNN Sentence:  here let us contemplated purity burden orgies formation
ionic agraffas edge cultivating
FFNN Sentence:  the condition of precluded accomplish bathed meaningles
s coal trip wrongs boughs curse
FFNN Sentence:  by one of form this differed omen yellowish knxw demure
ly inextricable wine
FFNN Sentence:  we cannot be denunciations bolts consolation via reject
ed captivity convenient glanced kapou
FFNN Sentence:  the organs of lectures forbearance decayed admission sh
rank denied cried judge polished
FFNN Sentence:  banners yellow glorious notices condemnation glades you
rself bewildered horde criminal oath hi
FFNN Sentence:  when first i flabby belt rigid wrong progressive commer
cial matt structure keziah
```

In [28]:

```
#RNN
for phrase in phrases:
    words = generate_words_RNN(model_primaryDataset, primary_RNN, phrase
, length=9)
    sentence = ""
    for word in words:
        sentence += word + " "
    print("RNN Sentence: ", sentence)
```

```
RNN Sentence:  the innumerable blossoms this as this this as as this af
forded this
RNN Sentence:  you say that similar reverberation potential killed impr
ession torches us laws erich
RNN Sentence:  there was a now immortal advancing response protested co
nsidered adversity thar sunlight
RNN Sentence:  here let us britain suggestion tyrants sensibility gibbe
ring manifest window despair shook
RNN Sentence:  the condition of this this as this this as afforded this
of
RNN Sentence:  by one of sufficient interrupted benign dungeon countena
nce approaches afforded in wise
RNN Sentence:  we cannot be rob gates shrunk nice dub kicked needs assi
sted nocturnal
RNN Sentence:  the organs of afforded this this as this this this this
this
RNN Sentence:  banners yellow glorious no so spirit afforded stirred la
cey dilemma fast notice
RNN Sentence:  when first i iron crew process clerical dungeon fragranc
e return ring by
```

Sources Cited

Aerin Kim, Perplexity Intuition (and its derivation) Never be perplexed again by perplexity https://towardsdatascience.com/perplexity-intuition-and-derivation-105dd481c8f3 (https://towardsdatascience.com/perplexity-intuition-and-derivation-105dd481c8f3)

An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition by Daniel Jurafsky and James H. Martin

Quality Evaluation

1) the organs of lectures forbearance decayed admission shrank denied cried judge polished

2) there was a now immortal advancing response protested considered adversity than sunlight

Joey -

Sentence 2 is more grammatical

Both sentences are equally sensical (or nonsensical)

Ford -

Both sentences are equally grammatical (or ungrammatical)

Sentence 2 makes more sense

1) by one of form this differed omen yellowish knxw demurely inextricable wine

2) here let us britain suggestion tyrants sensibility gibbering manifest window despair shook

Joey -

Sentence 1 is more grammatical

Both sentences are equally sensical (or nonsensical)

Ford -

Both sentences are equally grammatical (or ungrammatical)

Sentence 1 makes more sense

1) you say that relates excessive carousals dubious others glimpses disgraced godlike demons

2) when first i iron crew process clerical dungeon fragrance return ring by

Joey -

Both sentences are equally grammatical (or ungrammatical)

Both sentences are equally sensical (or nonsensical)

Ford -

Sentence 2 is more grammatical

Sentence 2 makes more sense

Sentence 1 was always the FFNN while sentence 2 was the RNN. The RNN seems to make better sentences but neither of the networks produced great results.

```
In [ ]:
```