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:

- Chakrya Ros
- Sara Timermans Pastor

Section 1: Word2Vec paper questions

(answer the questions here)

1. A CBOW word embedding takes the context of each word as the input and tries to predict the word in the context. For example, I am doing NPL homework, assume the input to the Neural Network to the word, "NPL", so we try to predict a target word, "homework". In training model, we need to use the one hot encoding to the input word and measure the output error compared to one hod encoding of the taret word "homework".
2. The different between CBOW and Skip-gram word embedding:

    - In CBOW, the current word is predicted using the window of surrouding context windows. For instance, if $w_{i- 1}, w_{i-2}, w_{i-3},w_{i+1}, w_{i+2}, w_{i+3}$ are given words or context, this model will give w_{i}. It's fast training and work better on frequency words.
    - In Skip Gram, it predicts the given sequence or context from the word. It's opposite of CBOW. For instance, if w_{i} is given, this will predict the context, $w_{i-1}, w_{i-2}, w_{i-3},w_{i+1}, w_{i+2}, w_{i+3}$. It's slow training and work better on infrequency words.

1. The task that the authors use to evaluate the generated word embeddings are to train CBOW and Skip-gram models on corpora with one trillion words, especially, unlimited size of vocabulary., RNN vectors are used with other techniquest to achieve over 50 percent.

1. PCA and t-SNE:

    - PCA is a dimension reduciton tool that can be helped to reduce a large set of variables to a small set and still contains most of the information in the original set.
    - t-SNE (t-Distributed Stochastic Neighbor Embedding) is a technique for dimemsionality reduction and is well suited for the visualization of high-dimensional datasets.

      They are important to the task of training and interpreting word embeddings because word embeddings model trains on the very large dataset and very large dimension word vector like 100 to 300 dimensions. So , PCA and t-SNE are the good tools to use reduce the dimemsion and can help use visualize on the graph.

Sources Cited

Cite all sources that you consulted to answer these questions here, including textbooks, papers, online resources, friends, etc.

- https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
- https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-8-dimensionality-reduction-chi2-pca-c6d06fb3fcf3

Section 2: Training your own word embeddings

(describe the Spooky Authors Dataset here)

Describe what data set you have chosen to compare and contrast with the Spooky Authors Dataset. Make sure to describe where it comes from and it's general properties.

We take data set from gutenberg project. We use nltk.coprus to get datasets. We decided to combine five differece texts into our whole dataset, shakespeare-caesar.txt, shakespeare-hamlet.txt, shakespeare-macbeth.txt,' austen-emma.txt, austen-persuasion.txt. The different between Spoopy Authors dataset and our dataset is the Spooky Authors dataset is non-fiction , factual and reports on true events. Our datasets are about fiction and novels that are based on the author's imagination.

In [1]:

```
# import your libraries here
```

```python
import nltk
#preprocessing
from nltk.corpus import stopwords  #stopwords
from nltk import word_tokenize   # tokenizing
from nltk.stem import PorterStemmer  # using the Porter Stemmer
from nltk.corpus import gutenberg
gutenberg.fileids()
nltk.download('stopwords')
#library for create dataset
import urllib
import bs4 as bs
import csv
#training model
import gensim
from gensim.models import Word2Vec

import numpy as np, array
from numpy import argmax
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
import re
from sklearn.manifold import TSNE
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt
import matplotlib.cm as cm
# % matplotlib inline

# feedforward model
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.utils import to_categorical
from keras.layers import Dense, Embedding, SimpleRNN
from keras.preprocessing import sequence
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/chakryaros/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Using TensorFlow backend.
```

In [211]:

```python
# code to train your word embeddings

class CBOW_Model:
    def __init__(self):
        self.corpus = []
        self.text = None
        self.author = None

    def train(self, dataset):
        df = pd.read_csv(dataset)
        self.text = df['text']

        # iterate through each sentence in cleanText
        for i in range(len(self.text)):
            cleanText = preprocessing(self.text[i])

            #tokenize the sentence into words
            self.corpus.append(word_tokenize(cleanText))


        # build vocabulary and train model
        # size is the dimensionality of the feature vectors.
        # window is the maximum distance between target word and its neighboring word
        # min_count is minimum frequency count of words, model would ignore the word
        # if it's less than min_count
        # workers = how many threads to use behind the scense
        # iter = number of iterations (epochs) over the corpus
        CBOW_mode = gensim.models.Word2Vec(self.corpus, min_count=2, size=150, window = 5, workers
=4, iter=10)
```

```
        return CBOW_mode

#create my own dataset
def createDataset():
    # Gettings the data source
    text1 = gutenberg.raw('shakespeare-caesar.txt')
    text2 = gutenberg.raw('shakespeare-hamlet.txt')
    text3 = gutenberg.raw('shakespeare-macbeth.txt')
    text4 = gutenberg.raw('austen-emma.txt')
    text5 = gutenberg.raw('austen-persuasion.txt')
    text = text1 + text2 + text3 + text4 + text5

    # Preprocessing the data
    text = re.sub(r'\[[0-9]*\]',' ',text)
    text = text.lower()
    text = re.sub(r'\d',' ',text)
    text = re.sub(r'\s+',' ',text)

    # convert the text into sentences
    sentences = nltk.sent_tokenize(text)

    #write into file
    with open('ourDataset.csv', 'w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(['text'])

        for sen in sentences:
            writer.writerow([sen])

createDataset()


#function to visualize the model
def tsne_plot_visualize(title, words, model):

    embeddingClusters = []
    wordClusters = []
    for word in words:
        embedding = []
        similarWord = []
        # find the similar word and add into similarword list
        for S_word, percent in model.wv.most_similar(word, topn=10):
            similarWord.append(S_word)
            # get word encoding word
            embedding.append(model.wv[S_word])

        embeddingClusters.append(embedding)
        wordClusters.append(similarWord)

    #convert embedding to numpy array
    embeddingClusters = np.array(embeddingClusters)
    #get the shape of embedding cluster
    x, y, z = embeddingClusters.shape
    tsne_model_en_2d = TSNE(perplexity=15, n_components=2, init='pca', n_iter=2500, random_state=32
)
    embeddings_en_2d = np.array(tsne_model_en_2d.fit_transform(embeddingClusters.reshape(x * y, z))
).reshape(x, y, 2)


    #plot the figure
    plt.figure(figsize=(16, 9))
    colors = cm.rainbow(np.linspace(0, 1, len(words)))
    for label, embeddings, words, color in zip(words, embeddings_en_2d, wordClusters, colors):
        x = embeddings[:, 0]
        y = embeddings[:, 1]
        plt.scatter(x, y, c=color, alpha=0.7, label=label)
        for i, word in enumerate(words):
            plt.annotate(word, alpha=0.7, xy=(x[i], y[i]), xytext=(5, 2),
                         textcoords='offset points', ha='right', va='top', size=8)
    plt.legend(loc=4)
    plt.title(title)
    plt.grid(True)

    plt.show()
```

```
# clean the dataset, remove space and convert to lower case
def preprocessing(text):
    text = re.sub("[^a-zA-Z]"," ", text)
    word_tokens = text.lower().split()

    #remove stopwords
    stopWord = stopwords.words('english')
    word_tokens = [word for word in word_tokens if not word in stopWord]

    cleanText = " ".join(word_tokens)
    return cleanText
```

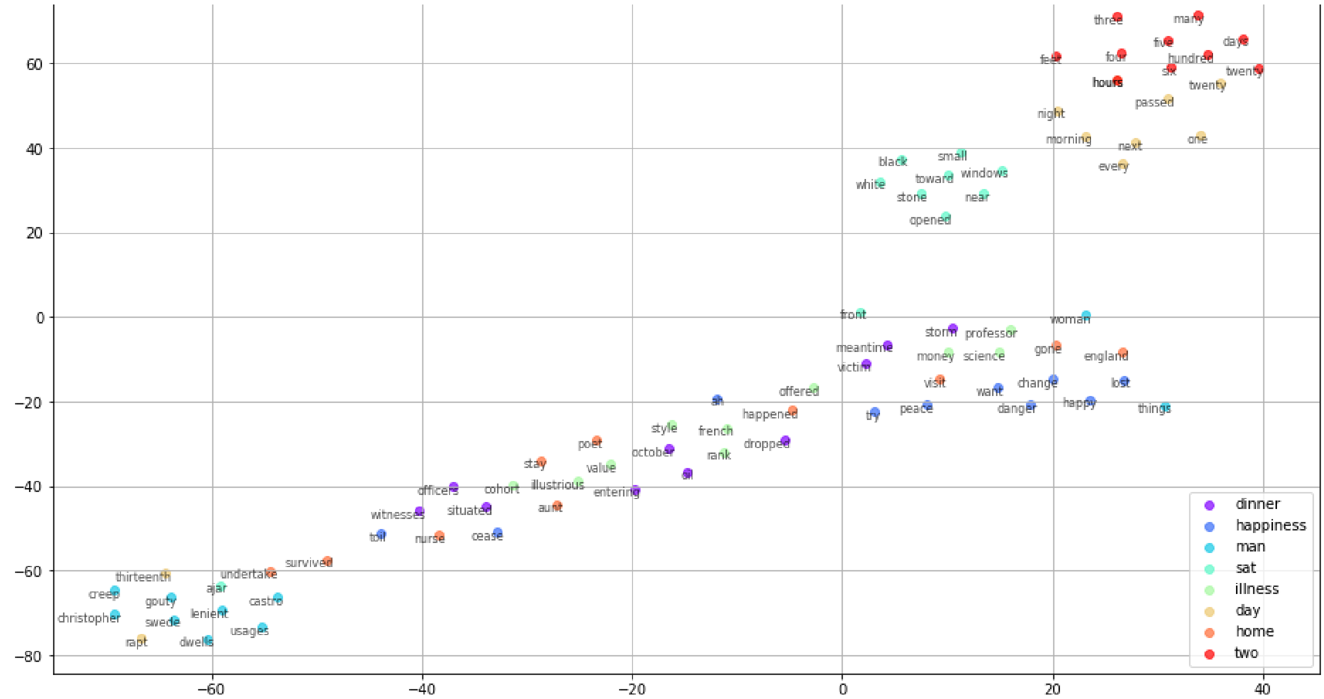
In [215]:

```
#Skoopy Authors Dataset using CBOW Model
wb = CBOW_Model()
cbow_model = wb.train('skoopy.csv')

#get the vocabulary from model
vocabs = list(cbow_model.wv.vocab)
print("vocabulary size of Skoopy dataset : ", len(vocabs))
#get encoding of word of 100 dimemsion
word_vectors = cbow_model.wv['love']

#words to display on the graph of the similar words in this list
words = ['dinner', 'happiness', 'man', 'sat', 'illness', 'day', 'home', 'two']

#find the most similar word
w = cbow_model.wv.most_similar(['sick'])
w1 = cbow_model.wv.most_similar(['dinner'])
print(w1)

#similarity between two differen word
print("CBOW model find similarity between 'cat' and 'dog': ",
       cbow_model.wv.similarity(w1="cat", w2="dog"))

tsne_plot_visualize("Skoopy Authors Dataset Using CBOW Model", words, cbow_model)
```

```
vocabulary size of Skoopy dataset :  15662
[('entering', 0.9995793104171753), ('witnesses', 0.9995638728141785), ('storm', 0.9995230436325073), ('meantime', 0.999504029750824), ('oil', 0.9994974732398987), ('dropped', 0.9994820356369019), ('october', 0.9994791746139526), ('situated', 0.9994727373123169), ('victim', 0.9994689226150513), ('officers', 0.9994637966156006)]
CBOW model find similarity between 'cat' and 'dog':  0.99875975
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.
```

Skoopy Authors Dataset Using CBOW Model



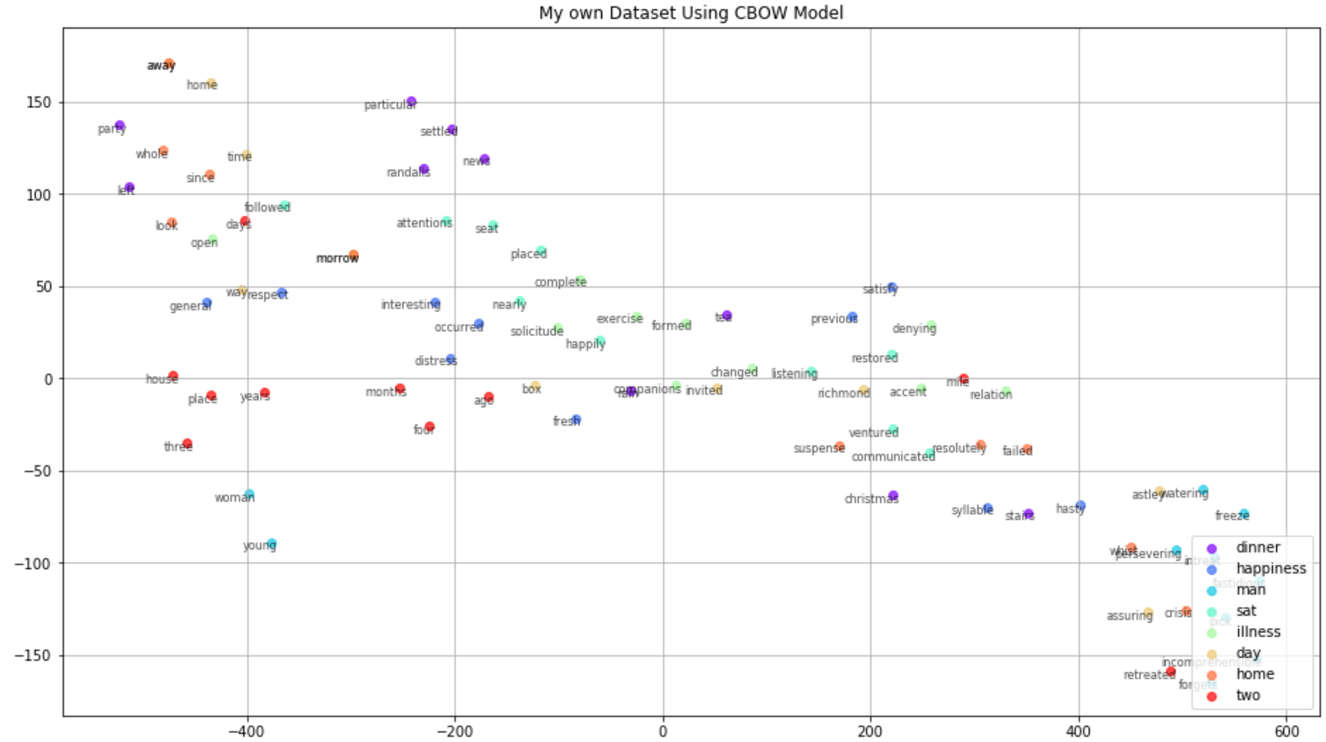
In [216]:

```
#austen-emma dataset from nltk using CBOW Model
wb = CBOW_Model()
cb = wb.train('ourDataset.csv')

# get vocabulary from CBOW model
vocabs = list(cb.wv.vocab)
print("vocabulary size of Emma dataset : ", len(vocabs))
mydata_words = ['dinner', 'happiness', 'man', 'sat', 'illness', 'day', 'home', 'two']
#find the most similar word
word = cb.wv.most_similar(['dinner'])
print("Most similar word to 'dinner'", word)

#similarity between two differen word
print("CBOW model find similarity between 'cat' and 'dog': ",
       cb.wv.similarity(w1="cat", w2="dog"))

tsne_plot_visualize("My own Dataset Using CBOW Model", mydata_words, cb)
```

```
vocabulary size of Emma dataset :  7827
Most similar word to 'dinner' [('tea', 0.9986586570739746), ('randalls', 0.9982583522796631), ('se
ttled', 0.9980821013450623), ('left', 0.9979518055915833), ('news', 0.9977721571922302),
('christmas', 0.997659444808960), ('party', 0.9975128173828125), ('stairs', 0.9974073171615601),
('rain', 0.9973178505897522), ('particular', 0.9973087906837463)]
CBOW model find similarity between 'cat' and 'dog':  0.991861
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
```



Sources Cited

Cite all sources that you consulted to answer these questions here, including textbooks, papers, online resources, friends, etc.

- https://machinelearningmastery.com/develop-word-embeddings-python-gensim/
- https://radimrehurek.com/gensim/models/word2vec.html
- https://towardsdatascience.com/google-news-and-leo-tolstoy-visualizing-word2vec-word-embeddings-with-t-sne-11558d8bd4d

Section 3: Evaluate the differences between the word embeddings

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

We ues CBOW to train these two different dataset. One is for skoopy author dataset and another one is our own dataset that we take from texts from the Project Gutenberg. The skoopy's dataset has 15662 vocabulary words and our own dataset have 7827 vacaulary words. After trainging these two datasets, we seleced some words like dinner', 'happiness', 'man', 'sat', 'illness', 'day', 'home', 'two from both models and find their most similar and draw the graph to see the different.

By looking at the first graph, the most top 10 similar words to 'dinner' are des, copy, letters, conversation, accident voyage, population, ride, shortly and moskoe.

By looking at the second graph, the most top 10 similar words to 'dinner' are visitor, early, town, pause, except, nessessarily,grateful, metioning, news and got.

By compare the most top 10 similar words to 'dinner' in those two datasets, we can see none of their similar words are the same. The season that they don't have the same similar words because the neighbors to the word 'dinner' in these two dataset are different.

More than that, in the skoopy author dataset, the first graph, the group of similar words by the colors stay closer to each other, but our dataset , the group of similar words by the colors do not quite stay closer.

Sources Cited

Cite all sources that you consulted to answer these questions here, including textbooks, papers, online resources, friends, etc.

- https://www.guru99.com/word-embedding-word2vec.html
- https://code.google.com/archive/p/word2vec/

Section 4: Feedforward Neural Language Model

In [283]:

```
# 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

class FeedforwardNeural:

    def __init__(self, x,y):

        self.x_train = x      # input
        self.y_train = y     # output


    #forward propagation through our network
    def train(self):

        # set up the basis for a feed forward network
        model = Sequential()

        # set up three layers
        model.add(Dense(units= 100, activation='relu', input_dim=self.x_train.shape[1]))

        model.add(Dense(units= 50, activation='relu'))

        model.add(Dense(units=self.y_train.shape[1], activation='softmax'))



        # configure the learning process
        model.compile(loss='binary_crossentropy',optimizer='sgd',metrics=['accuracy'])


        # fit the model
        history = model.fit(self.x_train, self.y_train, epochs=150, verbose=1, batch_size=35)
        #batch size is responsible for how many samples we want to
        #use in one epoch, which means how many samples are used
        #in one forward/backward pass.


        return model, history


def oneHot_encode(data):

    # integer encode
    label_encoder = LabelEncoder()
    integer_encoded = label_encoder.fit_transform(data)

    # binary encode
    onehot_encoder = OneHotEncoder(sparse=False)
    integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
    y_train_encode = onehot_encoder.fit_transform(integer_encoded)

    return y_train_encode

def get_word_embedding_split_dataset(wb_model, N):
    y_train = []
    x_train = []
    vocabs = list(wb_model.wv.vocab)

    # get the words from word embedding model
    for i in range(3, len(vocabs)):
        wb_i_1 = wb_model.wv[vocabs[i - N]]
        wb_i_2 = wb_model.wv[vocabs[i - N + 1]]
        wb_i_3 = wb_model.wv[vocabs[i - N + 2]]
        wb_i = wb_i_1 + wb_i_2 + wb_i_3
        x_train.append(wb_i)
        # get the word from vocab list
        y_train.append(vocabs[i])

    x_train = np.array(x_train)
    y_train = np.array(y_train)
```

```
    y_train = np.array(y_train)

    X_train, X_test, Y_train, y_test = train_test_split(x_train, y_train, test_size=0.25, random_st
ate=42)

    return X_train, X_test, Y_train, y_test
```

In [228]:

```
#train skoopy author data on feedforward
wb = CBOW_Model()
skoopy_data_model = wb.train('skoopy.csv')
X_train, X_test, Y_train, y_test = get_word_embedding_split_dataset(skoopy_data_model, 3)
print(X_train.shape)


#encoding output
y_train_encode = oneHot_encode(Y_train)

print(y_train_encode.shape)


ffw_model = FeedforwardNeural(X_train, y_train_encode)
ffw, history = ffw_model.train()
print(ffw.summary())
```

```
(11744, 150)
(11744, 11744)
(3915, 3915)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_encoders.py:415: FutureWarning: The
handling of integer data will change in version 0.22. Currently, the categories are determined
based on the range [0, max(values)], while in the future they will be determined based on the
unique values.
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, t
hen you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_encoders.py:415: FutureWarning: The
handling of integer data will change in version 0.22. Currently, the categories are determined
based on the range [0, max(values)], while in the future they will be determined based on the
unique values.
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, t
hen you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
```

```
Epoch 1/150
11744/11744 [==============================] - 11s 919us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 2/150
11744/11744 [==============================] - 10s 847us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 3/150
11744/11744 [==============================] - 9s 782us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 4/150
11744/11744 [==============================] - 9s 774us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 5/150
11744/11744 [==============================] - 8s 640us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 6/150
11744/11744 [==============================] - 7s 591us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 7/150
11744/11744 [==============================] - 7s 624us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 8/150
11744/11744 [==============================] - 8s 670us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 9/150
11744/11744 [==============================] - 8s 688us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 10/150
11744/11744 [==============================] - 6s 553us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 11/150
11744/11744 [==============================] - 6s 552us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 12/150
11744/11744 [==============================] - 9s 752us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 13/150
11744/11744 [==============================] - 9s 731us/step - loss: 8.8288e-04 - accuracy: 0.9999
```

```
11744/11744 [==============================] - 9s 731us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 14/150
11744/11744 [==============================] - 7s 636us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 15/150
11744/11744 [==============================] - 7s 620us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 16/150
11744/11744 [==============================] - 7s 619us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 17/150
11744/11744 [==============================] - 7s 633us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 18/150
11744/11744 [==============================] - 8s 643us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 19/150
11744/11744 [==============================] - 7s 629us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 20/150
11744/11744 [==============================] - 9s 767us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 21/150
11744/11744 [==============================] - 10s 876us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 22/150
11744/11744 [==============================] - 9s 752us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 23/150
11744/11744 [==============================] - 9s 752us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 24/150
11744/11744 [==============================] - 10s 827us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 25/150
11744/11744 [==============================] - 9s 728us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 26/150
11744/11744 [==============================] - 8s 683us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 27/150
11744/11744 [==============================] - 7s 572us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 28/150
11744/11744 [==============================] - 8s 713us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 29/150
11744/11744 [==============================] - 7s 562us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 30/150
11744/11744 [==============================] - 7s 583us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 31/150
11744/11744 [==============================] - 7s 556us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 32/150
11744/11744 [==============================] - 7s 575us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 33/150
11744/11744 [==============================] - 7s 623us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 34/150
11744/11744 [==============================] - 9s 747us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 35/150
11744/11744 [==============================] - 9s 734us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 36/150
11744/11744 [==============================] - 8s 710us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 37/150
11744/11744 [==============================] - 10s 822us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 38/150
11744/11744 [==============================] - 8s 719us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 39/150
11744/11744 [==============================] - 10s 817us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 40/150
11744/11744 [==============================] - 9s 737us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 41/150
11744/11744 [==============================] - 8s 700us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 42/150
11744/11744 [==============================] - 8s 641us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 43/150
11744/11744 [==============================] - 8s 641us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 44/150
11744/11744 [==============================] - 8s 648us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 45/150
11744/11744 [==============================] - 8s 643us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 46/150
11744/11744 [==============================] - 8s 640us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 47/150
11744/11744 [==============================] - 9s 750us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 48/150
11744/11744 [==============================] - 8s 705us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 49/150
11744/11744 [==============================] - 8s 723us/step - loss: 8.8288e-04 - accuracy: 0.9999
```

```
Epoch 50/150
11744/11744 [==============================] - 10s 817us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 51/150
11744/11744 [==============================] - 9s 800us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 52/150
11744/11744 [==============================] - 9s 752us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 53/150
11744/11744 [==============================] - 8s 711us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 54/150
11744/11744 [==============================] - 8s 692us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 55/150
11744/11744 [==============================] - 8s 667us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 56/150
11744/11744 [==============================] - 8s 671us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 57/150
11744/11744 [==============================] - 9s 771us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 58/150
11744/11744 [==============================] - 8s 681us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 59/150
11744/11744 [==============================] - 9s 762us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 60/150
11744/11744 [==============================] - 8s 688us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 61/150
11744/11744 [==============================] - 8s 686us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 62/150
11744/11744 [==============================] - 8s 696us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 63/150
11744/11744 [==============================] - 9s 744us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 64/150
11744/11744 [==============================] - 9s 736us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 65/150
11744/11744 [==============================] - 8s 709us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 66/150
11744/11744 [==============================] - 9s 795us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 67/150
11744/11744 [==============================] - 9s 785us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 68/150
11744/11744 [==============================] - 11s 898us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 69/150
11744/11744 [==============================] - 9s 778us/step - loss: 8.8288e-04 - accuracy:
0.99990s - loss: 8.828
Epoch 70/150
11744/11744 [==============================] - 9s 728us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 71/150
11744/11744 [==============================] - 9s 794us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 72/150
11744/11744 [==============================] - 9s 799us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 73/150
11744/11744 [==============================] - 10s 843us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 74/150
11744/11744 [==============================] - 9s 793us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 75/150
11744/11744 [==============================] - 9s 786us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 76/150
11744/11744 [==============================] - 8s 685us/step - loss: 8.8288e-04 - accuracy:
0.99990s - loss: 8.8288e-04 - ac
Epoch 77/150
11744/11744 [==============================] - 8s 684us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 78/150
11744/11744 [==============================] - 9s 736us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 79/150
11744/11744 [==============================] - 9s 781us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 80/150
11744/11744 [==============================] - 10s 816us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 81/150
11744/11744 [==============================] - 12s 1ms/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 82/150
11744/11744 [==============================] - 9s 806us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 83/150
11744/11744 [==============================] - 8s 688us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 84/150
11744/11744 [==============================] - 7s 574us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 85/150
```

```
11744/11744 [==============================] - 7s 616us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 86/150
11744/11744 [==============================] - 7s 582us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 87/150
11744/11744 [==============================] - 9s 730us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 88/150
11744/11744 [==============================] - 9s 727us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 89/150
11744/11744 [==============================] - 9s 767us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 90/150
11744/11744 [==============================] - 10s 889us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 91/150
11744/11744 [==============================] - 7s 598us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 92/150
11744/11744 [==============================] - 8s 679us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 93/150
11744/11744 [==============================] - 10s 809us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 94/150
11744/11744 [==============================] - 10s 861us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 95/150
11744/11744 [==============================] - 9s 736us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 96/150
11744/11744 [==============================] - 8s 687us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 97/150
11744/11744 [==============================] - 8s 706us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 98/150
11744/11744 [==============================] - 8s 693us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 99/150
11744/11744 [==============================] - 9s 743us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 100/150
11744/11744 [==============================] - 10s 877us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 101/150
11744/11744 [==============================] - 8s 660us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 102/150
11744/11744 [==============================] - 7s 597us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 103/150
11744/11744 [==============================] - 7s 596us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 104/150
11744/11744 [==============================] - 7s 587us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 105/150
11744/11744 [==============================] - 7s 594us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 106/150
11744/11744 [==============================] - 8s 664us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 107/150
11744/11744 [==============================] - 9s 795us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 108/150
11744/11744 [==============================] - 10s 874us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 109/150
11744/11744 [==============================] - 10s 852us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 110/150
11744/11744 [==============================] - 9s 739us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 111/150
11744/11744 [==============================] - 10s 837us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 112/150
11744/11744 [==============================] - 9s 758us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 113/150
11744/11744 [==============================] - 8s 681us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 114/150
11744/11744 [==============================] - 9s 761us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 115/150
11744/11744 [==============================] - 10s 849us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 116/150
11744/11744 [==============================] - 7s 584us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 117/150
11744/11744 [==============================] - 7s 634us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 118/150
11744/11744 [==============================] - 10s 840us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 119/150
```

```
11744/11744 [==============================] - 9s 731us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 120/150
11744/11744 [==============================] - 9s 752us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 121/150
11744/11744 [==============================] - 10s 825us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 122/150
11744/11744 [==============================] - 8s 695us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 123/150
11744/11744 [==============================] - 8s 675us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 124/150
11744/11744 [==============================] - 8s 679us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 125/150
11744/11744 [==============================] - 9s 777us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 126/150
11744/11744 [==============================] - 8s 709us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 127/150
11744/11744 [==============================] - 9s 760us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 128/150
11744/11744 [==============================] - 9s 743us/step - loss: 8.8288e-04 - accuracy:
0.99990s - loss: 8.8288e-04 - accuracy: 0.
Epoch 129/150
11744/11744 [==============================] - 8s 682us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 130/150
11744/11744 [==============================] - 9s 742us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 131/150
11744/11744 [==============================] - 8s 657us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 132/150
11744/11744 [==============================] - 8s 662us/step - loss: 8.8288e-04 - accuracy:
0.99991s
Epoch 133/150
11744/11744 [==============================] - 8s 666us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 134/150
11744/11744 [==============================] - 8s 681us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 135/150
11744/11744 [==============================] - 8s 653us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 136/150
11744/11744 [==============================] - 8s 713us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 137/150
11744/11744 [==============================] - 8s 660us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 138/150
11744/11744 [==============================] - 8s 667us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 139/150
11744/11744 [==============================] - 8s 666us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 140/150
11744/11744 [==============================] - 8s 657us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 141/150
11744/11744 [==============================] - 9s 801us/step - loss: 8.8288e-04 - accuracy:
0.99990s - loss:
Epoch 142/150
11744/11744 [==============================] - 10s 850us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 143/150
11744/11744 [==============================] - 10s 843us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 144/150
11744/11744 [==============================] - 10s 881us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 145/150
11744/11744 [==============================] - 10s 829us/step - loss: 8.8288e-04 - accuracy: 0.999
9
Epoch 146/150
11744/11744 [==============================] - 9s 762us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 147/150
11744/11744 [==============================] - 8s 713us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 148/150
11744/11744 [==============================] - 8s 696us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 149/150
11744/11744 [==============================] - 8s 641us/step - loss: 8.8288e-04 - accuracy: 0.9999
Epoch 150/150
11744/11744 [==============================] - 7s 622us/step - loss: 8.8288e-04 - accuracy: 0.9999
Model: "sequential_59"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_101 (Dense)            (None, 100)               15100     
_________________________________________________________________
```

```
dense_102 (Dense)            (None, 50)               5050
_________________________________________________________________
dense_103 (Dense)            (None, 11744)            598944
=================================================================
Total params: 619,094
Trainable params: 619,094
Non-trainable params: 0
_________________________________________________________________
None
```

In [140]:

```
# train our own data on feedforward
Ourdata_model = wb.train('ourDataset.csv')

#get input for word embedding model
X_train_OurData, X_test_OurData, Y_train_OurData, y_test_OurData =
get_word_embedding_split_dataset(Ourdata_model, 3)
print(X_train_OurData.shape)
print(Y_train_OurData.shape)

#encoding output
y_train_OurData_encode = oneHot_encode(Y_train_OurData)

ffw_model_ourData = FeedforwardNeural(X_train_OurData, y_train_OurData_encode)
ffw_ourData = ffw_model_ourData.train()
print(ffw_ourData.summary())
```

```
(14128, 150)
(14128,)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_encoders.py:415: FutureWarning: The
handling of integer data will change in version 0.22. Currently, the categories are determined
based on the range [0, max(values)], while in the future they will be determined based on the
unique values.
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, t
hen you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
```

```
Epoch 1/150
14128/14128 [==============================] - 13s 904us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 2/150
14128/14128 [==============================] - 14s 998us/step - loss: 7.4696e-04 - accuracy: 0.999
9s - loss: 7.4696e-0
Epoch 3/150
14128/14128 [==============================] - 12s 842us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 4/150
14128/14128 [==============================] - 12s 873us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 5/150
14128/14128 [==============================] - 13s 921us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 6/150
14128/14128 [==============================] - 12s 857us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 7/150
14128/14128 [==============================] - 12s 867us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 8/150
14128/14128 [==============================] - 12s 858us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 9/150
14128/14128 [==============================] - 12s 872us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 10/150
14128/14128 [==============================] - 12s 872us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 11/150
14128/14128 [==============================] - 13s 888us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 12/150
```

```
14128/14128 [==============================] - 13s 917us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 13/150
14128/14128 [==============================] - 13s 906us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 14/150
14128/14128 [==============================] - 13s 890us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 15/150
14128/14128 [==============================] - 12s 876us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 16/150
14128/14128 [==============================] - 13s 895us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 17/150
14128/14128 [==============================] - 12s 883us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 18/150
14128/14128 [==============================] - 13s 902us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 19/150
14128/14128 [==============================] - 13s 896us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 20/150
14128/14128 [==============================] - 12s 884us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 21/150
14128/14128 [==============================] - 13s 953us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 22/150
14128/14128 [==============================] - 13s 892us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 23/150
14128/14128 [==============================] - 13s 885us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 24/150
14128/14128 [==============================] - 12s 884us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 25/150
14128/14128 [==============================] - 12s 848us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 26/150
14128/14128 [==============================] - 13s 898us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 27/150
14128/14128 [==============================] - 12s 875us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 28/150
14128/14128 [==============================] - 12s 883us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 29/150
14128/14128 [==============================] - 13s 905us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 30/150
14128/14128 [==============================] - 13s 896us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 31/150
14128/14128 [==============================] - 12s 830us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 32/150
14128/14128 [==============================] - 11s 799us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 33/150
14128/14128 [==============================] - 12s 853us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 34/150
14128/14128 [==============================] - 12s 860us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 35/150
14128/14128 [==============================] - 66s 5ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 36/150
14128/14128 [==============================] - 11s 753us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 37/150
14128/14128 [==============================] - 11s 773us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 38/150
```

```
14128/14128 [==============================] - 482s 34ms/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 39/150
14128/14128 [==============================] - 11s 779us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 40/150
14128/14128 [==============================] - 12s 884us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 41/150
14128/14128 [==============================] - 14s 963us/step - loss: 7.4696e-04 - accuracy: 0.999
9s - loss: 7.4696e-04 - accu
Epoch 42/150
14128/14128 [==============================] - 15s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 43/150
14128/14128 [==============================] - 13s 955us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 44/150
14128/14128 [==============================] - 13s 933us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 45/150
14128/14128 [==============================] - 14s 964us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 46/150
14128/14128 [==============================] - 21s 2ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 47/150
14128/14128 [==============================] - 11s 777us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 48/150
14128/14128 [==============================] - 13s 932us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 49/150
14128/14128 [==============================] - 13s 927us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 50/150
14128/14128 [==============================] - 15s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 51/150
14128/14128 [==============================] - 11s 795us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 52/150
14128/14128 [==============================] - 14s 984us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 53/150
14128/14128 [==============================] - 13s 905us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 54/150
14128/14128 [==============================] - 20s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 55/150
14128/14128 [==============================] - 11s 779us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 56/150
14128/14128 [==============================] - 14s 972us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 57/150
14128/14128 [==============================] - 13s 926us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 58/150
14128/14128 [==============================] - 15s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 59/150
14128/14128 [==============================] - 14s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 60/150
14128/14128 [==============================] - 14s 957us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 61/150
14128/14128 [==============================] - 18s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 62/150
14128/14128 [==============================] - 74s 5ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 63/150
14128/14128 [==============================] - 75s 5ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 64/150
14128/14128 [==============================] - 26s 2ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 65/150
14128/14128 [==============================] - 12s 859us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 66/150
14128/14128 [==============================] - 12s 882us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 67/150
```

```
14128/14128 [==============================] - 12s 868us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 68/150
14128/14128 [==============================] - 15s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 69/150
14128/14128 [==============================] - 12s 815us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 70/150
14128/14128 [==============================] - 13s 913us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 71/150
14128/14128 [==============================] - 12s 876us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 72/150
14128/14128 [==============================] - 23s 2ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 73/150
14128/14128 [==============================] - 11s 766us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 74/150
14128/14128 [==============================] - 13s 905us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 75/150
14128/14128 [==============================] - 13s 886us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 76/150
14128/14128 [==============================] - 14s 998us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 77/150
14128/14128 [==============================] - 18s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 78/150
14128/14128 [==============================] - 13s 925us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 79/150
14128/14128 [==============================] - 48s 3ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 80/150
14128/14128 [==============================] - 10s 727us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 81/150
14128/14128 [==============================] - 12s 814us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 82/150
14128/14128 [==============================] - 13s 891us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 83/150
14128/14128 [==============================] - 13s 896us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 84/150
14128/14128 [==============================] - 14s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 85/150
14128/14128 [==============================] - 14s 964us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 86/150
14128/14128 [==============================] - 12s 849us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 87/150
14128/14128 [==============================] - 12s 836us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 88/150
14128/14128 [==============================] - 60s 4ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 89/150
14128/14128 [==============================] - 11s 749us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 90/150
14128/14128 [==============================] - 13s 891us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 91/150
14128/14128 [==============================] - 12s 884us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 92/150
14128/14128 [==============================] - 14s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 93/150
14128/14128 [==============================] - 13s 940us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 94/150
14128/14128 [==============================] - 13s 896us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 95/150
```

```
14128/14128 [==============================] - 12s 873us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 96/150
14128/14128 [==============================] - 14s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 97/150
14128/14128 [==============================] - 14s 984us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 98/150
14128/14128 [==============================] - 13s 955us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 99/150
14128/14128 [==============================] - 16s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 100/150
14128/14128 [==============================] - 67s 5ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 101/150
14128/14128 [==============================] - 77s 5ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 102/150
14128/14128 [==============================] - 30s 2ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 103/150
14128/14128 [==============================] - 11s 807us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 104/150
14128/14128 [==============================] - 13s 921us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 105/150
14128/14128 [==============================] - 13s 927us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 106/150
14128/14128 [==============================] - 14s 995us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 107/150
14128/14128 [==============================] - 13s 885us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 108/150
14128/14128 [==============================] - 12s 879us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 109/150
14128/14128 [==============================] - 13s 898us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 110/150
14128/14128 [==============================] - 294s 21ms/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 111/150
14128/14128 [==============================] - 19s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 112/150
14128/14128 [==============================] - 13s 951us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 113/150
14128/14128 [==============================] - 11s 746us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 114/150
14128/14128 [==============================] - 12s 818us/step - loss: 7.4696e-04 - accuracy: 0.999
9s - l
Epoch 115/150
14128/14128 [==============================] - 19s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 116/150
14128/14128 [==============================] - 19s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 117/150
14128/14128 [==============================] - 16s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 118/150
14128/14128 [==============================] - 11s 802us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 119/150
14128/14128 [==============================] - 12s 839us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 120/150
14128/14128 [==============================] - 12s 856us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 121/150
14128/14128 [==============================] - 11s 809us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 122/150
14128/14128 [==============================] - 12s 855us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 123/150
14128/14128 [==============================] - 14s 969us/step - loss: 7.4696e-04 - accuracy: 0.999
9
```

```
Epoch 124/150
14128/14128 [==============================] - 13s 909us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 125/150
14128/14128 [==============================] - 15s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 126/150
14128/14128 [==============================] - 16s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 127/150
14128/14128 [==============================] - 11s 774us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 128/150
14128/14128 [==============================] - 14s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 129/150
14128/14128 [==============================] - 11s 752us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 130/150
14128/14128 [==============================] - 14s 958us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 131/150
14128/14128 [==============================] - 10s 743us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 132/150
14128/14128 [==============================] - 11s 773us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 133/150
14128/14128 [==============================] - 13s 894us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 134/150
14128/14128 [==============================] - 13s 950us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 135/150
14128/14128 [==============================] - 13s 942us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 136/150
14128/14128 [==============================] - 15s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 137/150
14128/14128 [==============================] - 15s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 138/150
14128/14128 [==============================] - 12s 878us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 139/150
14128/14128 [==============================] - 15s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 140/150
14128/14128 [==============================] - 21s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 141/150
14128/14128 [==============================] - 18s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 142/150
14128/14128 [==============================] - 14s 988us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 143/150
14128/14128 [==============================] - 14s 990us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 144/150
14128/14128 [==============================] - 13s 952us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 145/150
14128/14128 [==============================] - 12s 851us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 146/150
14128/14128 [==============================] - 17s 1ms/step - loss: 7.4696e-04 - accuracy: 0.9999
Epoch 147/150
14128/14128 [==============================] - 11s 769us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 148/150
14128/14128 [==============================] - 11s 786us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 149/150
14128/14128 [==============================] - 11s 789us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Epoch 150/150
14128/14128 [==============================] - 11s 794us/step - loss: 7.4696e-04 - accuracy: 0.999
9
Model: "sequential_51"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_83 (Dense)             (None, 100)               15100
```

```
_________________________________________________________________
dense_84 (Dense)             (None, 50)                5050      
_________________________________________________________________
dense_85 (Dense)             (None, 14128)             720528    
=================================================================
Total params: 740,678
Trainable params: 740,678
Non-trainable params: 0
_________________________________________________________________
None
```

In [267]:

```
def oneHot_decode(data, encode):

    # integer encode
    label_encoder = LabelEncoder()
    integer_encoded = label_encoder.fit_transform(data)


    # binary encode
    onehot_encoder = OneHotEncoder(sparse=False)
    integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
    y_train_encode = onehot_encoder.fit_transform(integer_encoded)
    inverted = []

    # invert first example
    for i in range(len(encode)):
        decode = label_encoder.inverse_transform([argmax(encode[i ,:])])

        inverted.append(decode[0])


    return inverted

# make predictions with skoopy dataset
vocabs = list(skoopy_data_model.wv.vocab)

predictions_spooky = ffw.predict(X_test)
print(predictions_skoopy.shape)


# decode vector into word after prediction
# inverted_predict_word = oneHot_decode(vocabs, predictions_skoopy)

# make predictions with Our dataset
vocabs_our = list(Ourdata_model.wv.vocab)
predictions_OurData = ffw_ourData.predict(X_test_OurData)



# decode vector into word after prediction
# inverted_predict_word = oneHot_decode(vocabs_our, predictions_OurData)


skoopy_perplexity = 2**(8.8288e-04)
print("Skoopy Dataset Word Embeddings Perplexity :" ,skoopy_perplexity)

ourData_perplexity = 2**(7.4696e-04)
print("Our Dataset Word Embeddings Perplexity :" ,ourData_perplexity)
print()
if skoopy_perplexity < ourData_perplexity:
    print("Feed forward on Skoopy Dataset Word Embeddings Model is better than Our Dataset Word Em
beddings Model")
    print("Skoopy Dateset Word Embeddings Model have Lower Perplexity.")

else:
    print("Feed forward on Our Dataset Word Embeddings Model is better than Skoopy Dataset Word Em
beddings Model")
    print("Our Dateset Word Embeddings Model have Lower Perplexity.")
```

```
(3915, 11744)
Skoopy Dataset Word Embeddings Perplexity : 1.0006121530720353
Our Dataset Word Embeddings Perplexity : 1.0005178872753235
```

```
Feed forward on Our Dataset Word Embeddings Model is better than Skoopy Dataset Word Embeddings Mo
del
Our Dateset Word Embeddings Model have Lower Perplexity.
```

Sources Cited

Cite all sources that you consulted to answer these questions here, including textbooks, papers, online resources, friends, etc.

- https://stackabuse.com/python-for-nlp-word-embeddings-for-deep-learning-in-keras/
- https://keras.io/
- https://web.stanford.edu/~jurafsky/slp3/7.pdf
- https://keras.io/callbacks/#csvlogger

Section 5: Recurrent Neural Language Model

In [294]:

```
# code to train a recurrent neural language model


class RNN:
    def __init__(self, x_train,y_train, x_test, y_test, vocabs):
        self.x_train = x_train
        self.y_train = y_train
        self.x_test = x_test
        self.y_test = y_test
        self.vocabs = vocabs


    def train(self):

        # size to cut texts after this number of words
        maxlen = 80
        batch_size = 32

        # reshape the input and output size
        self.x_train = sequence.pad_sequences(self.x_train, maxlen=maxlen)
        self.y_test = sequence.pad_sequences(self.y_test, maxlen=maxlen)

        ''' define the keras model '''
        model = Sequential()

        #number of hidden units
        model.add(SimpleRNN(units = 128, activation = "relu"))

        # number of outputs
        model.add(Dense(10))

        input_shape = (len(self.vocabs), 10, self.y_train.shape)
        model.build(input_shape)



        print(model.summary())


        # fit model
        model.fit(self.x_train, self.y_train, batch_size=35,epochs=15,
          validation_data=(self.x_test, self.y_test))

        return model
```

In [297]:

```
#train skoopy author data on RNN
wb = CBOW_Model()
skoopy_data_model = wb.train('skoopy.csv')
vocabs = list(skoopy_data_model.wv.vocab)
X_train_RNN, X_test_RNN, Y_train_RNN, y_test_RNN =
get_word_embedding_split_dataset(skoopy_data_model, 3)
```

```
print(X_train.shape)


#encoding output
y_train_RNN_encode = oneHot_encode(Y_train_RNN)
print(y_train__RNN_encode.shape)
y_test_RNN_encode = oneHot_encode(y_test_RNN)
rnn = RNN(X_train, y_train_RNN_encode, X_test_RNN, y_test_RNN_encode, vocabs )
rnn_model = rnn.train()
```

```
(11744, 150)
```

In [ ]:

```
#train our dataset on RNN
Ourdata_model = wb.train('ourDataset.csv')
X_train, X_test, Y_train, y_test = get_word_embedding_split_dataset(skoopy_data_model, 3)
print(X_train.shape)
print(Y_train.shape)

rnn = RNN(X_train, Y_train, X_test, y_test)
rnn_model = rnn.train()
```

Sources Cited

Cite all sources that you consulted to answer these questions here, including textbooks, papers, online resources, friends, etc.

- https://www.tensorflow.org/tutorials/text/text_classification_rnn#train_the_model
- https://stackoverflow.com/questions/38294046/simple-recurrent-neural-network-input-shape
- https://github.com/keras-team/keras/blob/master/examples/imdb_lstm.py

Sources Cited

Cite all sources that you consulted to answer these questions here, including textbooks, papers, online resources, friends, etc.