CSCI 544 HOMEWORK 3

Method similar to the link

NAME: Asmita Chotani
USC ID: 3961468036

In [1]:
```
!pip install torchvision
```

```
Requirement already satisfied: torchvision in /Users/asmitachotani/opt/minicon
da3/lib/python3.9/site-packages (0.14.1)
Requirement already satisfied: torch in /Users/asmitachotani/opt/miniconda3/li
b/python3.9/site-packages (from torchvision) (1.13.1)
Requirement already satisfied: numpy in /Users/asmitachotani/opt/miniconda3/li
b/python3.9/site-packages (from torchvision) (1.23.5)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /Users/asmitachotani/o
pt/miniconda3/lib/python3.9/site-packages (from torchvision) (9.4.0)
Requirement already satisfied: requests in /Users/asmitachotani/opt/miniconda
3/lib/python3.9/site-packages (from torchvision) (2.27.1)
Requirement already satisfied: typing-extensions in /Users/asmitachotani/opt/m
iniconda3/lib/python3.9/site-packages (from torchvision) (4.5.0)
Requirement already satisfied: idna<4,>=2.5 in /Users/asmitachotani/opt/minico
nda3/lib/python3.9/site-packages (from requests->torchvision) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /Users/asmitachota
ni/opt/miniconda3/lib/python3.9/site-packages (from requests->torchvision) (2.
0.4)
Requirement already satisfied: certifi>=2017.4.17 in /Users/asmitachotani/opt/
miniconda3/lib/python3.9/site-packages (from requests->torchvision) (2022.5.1
8.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/asmitachotani/o
pt/miniconda3/lib/python3.9/site-packages (from requests->torchvision) (1.26.
9)
```

In [2]:
```
!pip install torch torchvision torchaudio
```

```
Requirement already satisfied: torch in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (1.13.1)
Requirement already satisfied: torchvision in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (0.14.1)
Requirement already satisfied: torchaudio in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (0.13.1)
Requirement already satisfied: typing-extensions in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (from torch) (4.5.0)
Requirement already satisfied: requests in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (from torchvision) (2.27.1)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (from torchvision) (9.4.0)
Requirement already satisfied: numpy in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (from torchvision) (1.23.5)
Requirement already satisfied: certifi>=2017.4.17 in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (from requests->torchvision) (2022.5.18.1)
Requirement already satisfied: idna<4,>=2.5 in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (from requests->torchvision) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (from requests->torchvision) (1.26.9)
Requirement already satisfied: charset-normalizer~=2.0.0 in /Users/asmitachotani/opt/miniconda3/lib/python3.9/site-packages (from requests->torchvision) (2.0.4)
```

In [3]:
```
!pip install -U gensim
```

```
Requirement already satisfied: gensim in /Users/asmitachotani/opt/miniconda3/l
ib/python3.9/site-packages (4.3.0)
Requirement already satisfied: smart-open>=1.8.1 in /Users/asmitachotani/opt/m
iniconda3/lib/python3.9/site-packages (from gensim) (6.3.0)
Requirement already satisfied: FuzzyTM>=0.4.0 in /Users/asmitachotani/opt/mini
conda3/lib/python3.9/site-packages (from gensim) (2.0.5)
Requirement already satisfied: scipy>=1.7.0 in /Users/asmitachotani/opt/minico
nda3/lib/python3.9/site-packages (from gensim) (1.10.0)
Requirement already satisfied: numpy>=1.18.5 in /Users/asmitachotani/opt/minic
onda3/lib/python3.9/site-packages (from gensim) (1.23.5)
Requirement already satisfied: pyfume in /Users/asmitachotani/opt/miniconda3/l
ib/python3.9/site-packages (from FuzzyTM>=0.4.0->gensim) (0.2.25)
Requirement already satisfied: pandas in /Users/asmitachotani/opt/miniconda3/l
ib/python3.9/site-packages (from FuzzyTM>=0.4.0->gensim) (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /Users/asmitachotani/
opt/miniconda3/lib/python3.9/site-packages (from pandas->FuzzyTM>=0.4.0->gensi
m) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/asmitachotani/opt/minico
nda3/lib/python3.9/site-packages (from pandas->FuzzyTM>=0.4.0->gensim) (2022.
7.1)
Requirement already satisfied: six>=1.5 in /Users/asmitachotani/opt/miniconda
3/lib/python3.9/site-packages (from python-dateutil>=2.8.1->pandas->FuzzyTM>=
0.4.0->gensim) (1.16.0)
Requirement already satisfied: fst-pso in /Users/asmitachotani/opt/miniconda3/
lib/python3.9/site-packages (from pyfume->FuzzyTM>=0.4.0->gensim) (1.8.1)
Requirement already satisfied: simpful in /Users/asmitachotani/opt/miniconda3/
lib/python3.9/site-packages (from pyfume->FuzzyTM>=0.4.0->gensim) (2.9.0)
Requirement already satisfied: miniful in /Users/asmitachotani/opt/miniconda3/
lib/python3.9/site-packages (from fst-pso->pyfume->FuzzyTM>=0.4.0->gensim) (0.
0.6)
Requirement already satisfied: requests in /Users/asmitachotani/opt/miniconda
3/lib/python3.9/site-packages (from simpful->pyfume->FuzzyTM>=0.4.0->gensim)
(2.27.1)
Requirement already satisfied: idna<4,>=2.5 in /Users/asmitachotani/opt/minico
nda3/lib/python3.9/site-packages (from requests->simpful->pyfume->FuzzyTM>=0.
4.0->gensim) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in /Users/asmitachotani/opt/
miniconda3/lib/python3.9/site-packages (from requests->simpful->pyfume->FuzzyT
M>=0.4.0->gensim) (2022.5.18.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/asmitachotani/o
pt/miniconda3/lib/python3.9/site-packages (from requests->simpful->pyfume->Fuz
zyTM>=0.4.0->gensim) (1.26.9)
Requirement already satisfied: charset-normalizer~=2.0.0 in /Users/asmitachota
ni/opt/miniconda3/lib/python3.9/site-packages (from requests->simpful->pyfume-
>FuzzyTM>=0.4.0->gensim) (2.0.4)
```

In [4]:
```
# pip install -U scikit-learn scipy matplotlib
```

In [5]:
```
# pip install contractions
```

In [6]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
nltk.download('punkt')  # for word tokenizing
nltk.download('stopwords') # for determining stop words taht have to be removed
nltk.download('omw-1.4') # for lemmatizing
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report,accuracy_score
from sklearn import svm
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV

import gensim

import re
from bs4 import BeautifulSoup
import warnings
warnings.filterwarnings('ignore')
import string
import contractions
import copy
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/asmitachotani/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/asmitachotani/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/asmitachotani/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /Users/asmitachotani/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [7]:
```
import matplotlib.pyplot as plt
```

In [8]:
```
# import libraries
import torch
import torchvision
from torch import nn
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import CrossEntropyLoss, Softmax, Linear
from torch.optim import SGD, Adam
from torch.utils.data.sampler import SubsetRandomSampler
```

In [9]:
```
print("Gensim Version: ", gensim.__version__)
print("Pandas Version: ",pd.__version__)
print("NumPy Version: ",np.__version__)
print("Pytorch Version: ",torch.__version__)
```

```
Gensim Version:  4.3.0
Pandas Version:  1.5.2
NumPy Version:  1.23.5
Pytorch Version:  1.12.1
```

In [10]:
```
# df2 = pd.read_csv('./amazon_reviews_us_Beauty_v1_00.tsv',
#                                 sep='\t',
#                                 error_bad_lines=False
#                                )
```

In [11]:
```
df3 = pd.read_csv('./amazon_reviews_us_Beauty_v1_00.tsv',
                                sep='\t',
```

```
                                    error_bad_lines=False,
                                    usecols=["star_rating", "review_body"]
                                   )
display(df3)
```

	star_rating	review_body
0	5	Love this, excellent sun block!!
1	5	The great thing about this cream is that it do...
2	5	Great Product, I'm 65 years old and this is al...
3	5	I use them as shower caps & conditioning caps....
4	5	This is my go-to daily sunblock. It leaves no ...
...	...	...
5094558	5	After watching my Dad struggle with his scisso...
5094559	3	Like most sound machines, the sounds choices a...
5094560	5	I bought this product because it indicated 30 ...
5094561	5	We have used Oral-B products for 15 years; thi...
5094562	5	I love this toothbrush. It's easy to use, and ...

5094563 rows × 2 columns

In [12]:
```
# understanding the different rating values that are possible
df3['star_rating'].unique()
```

Out[12]:
```
array(['5', '4', '1', '3', '2', '2015-08-28', '2015-08-16', '2015-08-14',
       5, 4, 3, 1, 2, '2015-07-27', '2015-07-26', '2015-07-23',
       '2015-07-22', nan, '2015-06-14', '2015-06-02', '2015-04-14',
       '2015-04-09', '2015-04-08', '2015-04-03', '2015-04-02',
       '2015-04-01', '2015-03-31', '2015-03-30', '2015-03-18',
       '2015-02-28', '2015-02-10', '2014-12-30', '2014-12-03',
       '2014-10-09'], dtype=object)
```

In [13]:
```
# creating a copy of the dataframe to work with
df=df3.copy()
```

We form three classes and select 20000 reviews randomly from each class.

In [14]:
```
# 3 classes are formed for the 5 kinds of ratings possible.
def categorise(row):
    if row['star_rating'] ==  1 or row['star_rating']== '1' or row['star_rating
        return 1
    elif row['star_rating'] ==  2 or row['star_rating']== '2'or row['star_ratin
        return 1
    elif row['star_rating'] == 3 or row['star_rating']== '3'or row['star_rating
        return 2
    elif row['star_rating'] ==  4 or row['star_rating']== '4'or row['star_ratin
        return 3
    elif row['star_rating'] ==  5 or row['star_rating']== '5'or row['star_ratin
        return 3
```

```
    else:
        return 0   # the entries with invalid values in the rating column
```

In [15]:
```
%%time
df['class'] = df.apply(lambda row: categorise(row), axis=1)
display(df)
```

	star_rating	review_body	class
0	5	Love this, excellent sun block!!	3
1	5	The great thing about this cream is that it do...	3
2	5	Great Product, I'm 65 years old and this is al...	3
3	5	I use them as shower caps & conditioning caps....	3
4	5	This is my go-to daily sunblock. It leaves no ...	3
...	...	...	...
5094558	5	After watching my Dad struggle with his scisso...	3
5094559	3	Like most sound machines, the sounds choices a...	2
5094560	5	I bought this product because it indicated 30 ...	3
5094561	5	We have used Oral-B products for 15 years; thi...	3
5094562	5	I love this toothbrush. It's easy to use, and ...	3

5094563 rows × 3 columns

```
CPU times: user 1min 29s, sys: 329 ms, total: 1min 29s
Wall time: 1min 29s
```

In [16]:
```
%%time
# Creating separate dataframes for separate classes
S1_dfa = df.loc[df['class'] == 1]
S2_dfa = df.loc[df['class'] == 2]
S3_dfa = df.loc[df['class'] == 3]

# COnsidering only 20000 data entries for each class
S1_df=S1_dfa.sample(n=20000)
S2_df=S2_dfa.sample(n=20000)
S3_df=S3_dfa.sample(n=20000)
```

```
CPU times: user 312 ms, sys: 23.7 ms, total: 336 ms
Wall time: 331 ms
```

In [17]:
```
# Concatenating 20000 reviews for each class into one dataframe that we will wo
review_df = pd.concat([S1_df, S2_df, S3_df])
display(review_df)
```

	star_rating	review_body	class
4049523	1	Not working for me,after seeing all the positi...	1
4520242	1.0	At first the shaving kit looked good. There ha...	1
4957903	1	I recently visited my dermatologist who advise...	1
3134662	2	There is one spot on my scalp that has lost so...	1
3484409	1	Smells bad and leaves the skin itchy!! I am so...	1
...	...	...	...
2443411	5	Fab.	3
4084839	4	I just used it for the first time. I am not su...	3
1972465	5	This lotion rocks. Soaks in completely,	3
3230041	5	Recommended in Ladies Home Journal for styling...	3
1011573	5	The product is great, easy to apply and lasts ...	3

60000 rows × 3 columns

Data Cleaning

Reseting Index

In [18]:
```
# Since we have randomly chosen 20000 entries from each class, it is necessary
# repitition of entries.
review_df = review_df.reset_index(drop=True)
display(review_df)
```

	star_rating	review_body	class
0	1	Not working for me,after seeing all the positi...	1
1	1.0	At first the shaving kit looked good. There ha...	1
2	1	I recently visited my dermatologist who advise...	1
3	2	There is one spot on my scalp that has lost so...	1
4	1	Smells bad and leaves the skin itchy!! I am so...	1
...	...	...	...
59995	5	Fab.	3
59996	4	I just used it for the first time. I am not su...	3
59997	5	This lotion rocks. Soaks in completely,	3
59998	5	Recommended in Ladies Home Journal for styling...	3
59999	5	The product is great, easy to apply and lasts ...	3

60000 rows × 3 columns

In [19]:
```
# Checking for null values
review_df.isnull().values.any()
```

Out[19]:
```
True
```

In [20]:
```
# Checking number of null values in the two columns
review_df.isnull().sum()
```

Out[20]:
```
star_rating     0
review_body     7
class           0
dtype: int64
```

In [21]:
```
# Filling the null values with an empty string as only empty value is in the re
review_df = review_df.fillna('')
```

In [22]:
```
wv_data=review_df.copy()
```

In [23]:
```
pre_dc = review_df['review_body'].str.len().mean()
```

In [24]:
```
#Converting the reviews into Lower Case
review_df['review_body'] = review_df['review_body'].str.lower()
display(review_df)
```

	star_rating	review_body	class
0	1	not working for me,after seeing all the positi...	1
1	1.0	at first the shaving kit looked good. there ha...	1
2	1	i recently visited my dermatologist who advise...	1
3	2	there is one spot on my scalp that has lost so...	1
4	1	smells bad and leaves the skin itchy!! i am so...	1
...	...	...	...
59995	5	fab.	3
59996	4	i just used it for the first time. i am not su...	3
59997	5	this lotion rocks. soaks in completely,	3
59998	5	recommended in ladies home journal for styling...	3
59999	5	the product is great, easy to apply and lasts ...	3

60000 rows × 3 columns

In [25]:
```
# Removing well-formed tags i.e the HTML and URLs
review_df['review_body'] = review_df['review_body'].str.replace(r'<[^<>]*>', ''
review_df['review_body'] = review_df['review_body'].apply(lambda x: re.split('h
```

In [26]:
```
def remove_mention_tag_fn(text):
    text = re.sub(r'@\S*', '', text)
    return re.sub(r'#\S*', '', text)
```

In [27]:
```
review_df['review_body'] = review_df['review_body'].apply(remove_mention_tag_fr
display(review_df)
```

	star_rating	review_body	class
0	1	not working for me,after seeing all the positi...	1
1	1.0	at first the shaving kit looked good. there ha...	1
2	1	i recently visited my dermatologist who advise...	1
3	2	there is one spot on my scalp that has lost so...	1
4	1	smells bad and leaves the skin itchy!! i am so...	1
...	...	...	...
59995	5	fab.	3
59996	4	i just used it for the first time. i am not su...	3
59997	5	this lotion rocks. soaks in completely,	3
59998	5	recommended in ladies home journal for styling...	3
59999	5	the product is great, easy to apply and lasts ...	3

60000 rows × 3 columns

In [28]:
```
def remove_punctuations(text):
    return ''.join(char for char in text if char not in string.punctuation)
```

In [29]:
```
# Remove puctuations
review_df['review_body'] = review_df['review_body'].apply(remove_punctuations)
```

In [30]:
```
def remove_alphanum(text):
    t= " ".join([re.sub('[^A-Za-z]+','', text) for text in nltk.word_tokenize(t
    return t
```

In [31]:
```
# Remove non-alpabetics
review_df['review_body']=review_df['review_body'].apply(remove_alphanum)
```

In [32]:
```
# removing extra space
review_df['review_body'] = review_df['review_body'].apply(lambda x: re.sub(' +'
```

In [33]:
```
def word_contractions(text):
    t=[]
    for i in text.split():
        t.append(contractions.fix(i))
    # Now that the review has been split into a list of words and contracted, t
    return ' '.join(t)
```

In [34]:
```
# Contracting the reviews
review_df['review_body']=review_df['review_body'].apply(word_contractions)
```

In [35]:
```
post_dc = review_df['review_body'].str.len().mean()
```

In [36]:
```
print("Average length of review body before and after Data Cleaning",pre_dc,pos
```

```
Average length of review body before and after Data Cleaning 270.0898333333333
3 258.55171666666666
```

In [37]:
```
clean_data=review_df.copy()
```

Word2Vec

In [38]:
```
#Splitting the dataset into testing and training dataset
Xtrain, Xtest, ytrain, ytest = train_test_split(clean_data['review_body'], clea

print("Training Shape ", Xtrain.shape)
print("Testing Shape ", Xtest.shape)
```

```
Training Shape  (48000,)
Testing Shape  (12000,)
```

In [39]:
```
type(ytrain)
```

Out[39]:
```
pandas.core.series.Series
```

(a) Load the pretrained “word2vec-google-news-300” Word2Vec model and learn how to extract word embeddings for your dataset. Try to check semantic similarities of the generated vectors using three examples of your own, e.g., King – Man + Woman = Queen or excellent ~ outstanding.

In [40]:
```
print(gensim.__version__, gensim.__file__)
```

```
4.3.0 /Users/asmitachotani/opt/anaconda3/envs/pytorch_a1/lib/python3.9/site-pa
ckages/gensim/__init__.py
```

In [41]:
```
import gensim.downloader as a_c
```

In [42]:
```
wv_model = a_c.load('word2vec-google-news-300')
```

In [43]:
```
wv_model.save('Gensim_model.kv')
```

In [44]:
```
print(wv_model.most_similar(positive=['Woman','King'], negative=['Man']))
print(wv_model.most_similar('Excellent'))
print(wv_model.most_similar(positive=['she','father'], negative=['him']))
print(wv_model.most_similar(positive=['Google','Gmail'], negative=['Outlook']))
print(wv_model.most_similar('happy'))
```

```
[('Queen', 0.4929387867450714), ('Tupou_V.', 0.45174285769462585), ('Oprah_BFF
_Gayle', 0.4422132968902588), ('Jackson', 0.4402504861354828), ('NECN_Alison',
0.4331282675266266), ('Whitfield', 0.42834725975990295), ('Ida_Vandross', 0.42
084529995918274), ('prosecutor_Dan_Satterberg', 0.420758992433548), ('martin_L
uther_King', 0.42059651017189026), ('Coretta_King', 0.42027339339256287)]
[('excellent', 0.6091997027397156), ('definition_redistributional', 0.57536011
9342804), ('Exceptional', 0.5664600729942322), ('flexible_hou_MORE', 0.5228071
212768555), ('EXCELLENT', 0.521685779094696), ('Decent', 0.5081128478050232),
('Superb', 0.502091646194458), ('Terrific', 0.4998748004436493), ('Satisfactor
y', 0.4908524453639984), ('+_Bens', 0.48303356766700745)]
[('mother', 0.7119966745376587), ('husband', 0.6427904963493347), ('daughter',
0.6421711444854736), ('sister', 0.6059560179710388), ('eldest_daughter', 0.598
8065004348755), ('grandmother', 0.5926641225814819), ('mom', 0.579074561595916
7), ('aunt', 0.5724061131477356), ('niece', 0.5554639101028442), ('granddaught
er', 0.5517464876174927)]
[('Yahoo', 0.6514489054679871), ('Google_Nasdaq_GOOG', 0.6343342661857605),
('Google_GOOG', 0.6178034543991089), ('search_engine', 0.6156800389289856),
('GoogleGoogle', 0.6143473982810974), ('Google_NSDQ_GOOG', 0.611092627048492
4), ('Google_NASDAQ_GOOG', 0.6102906465530396), ('Yahoo_Nasdaq_YHOO', 0.608016
9677734375), ('GMail', 0.6057670712471008), ('Google_nasdaq_GOOG', 0.604456603
5270691)]
[('glad', 0.7408890724182129), ('pleased', 0.6632170677185059), ('ecstatic',
0.6626912951469421), ('overjoyed', 0.6599285006523132), ('thrilled', 0.6514049
172401428), ('satisfied', 0.6437950134277344), ('proud', 0.6360421180725098),
('delighted', 0.627237856388092), ('disappointed', 0.6269948482513428), ('exci
ted', 0.6247665882110596)]
```

In [45]:
```
print(cosine_similarity([wv_model['queen']], [wv_model['king'] - wv_model['woma
print(cosine_similarity([wv_model['queen']], [wv_model['king'] - wv_model['man'
print(cosine_similarity([wv_model['he']], [wv_model['she']]))
print(cosine_similarity([wv_model['excellent']],[wv_model['outstanding']]))
print(cosine_similarity([wv_model['him']], [wv_model['father'] - wv_model['she'
print(cosine_similarity([wv_model['Google']], [wv_model['Gmail']]))
```

```
[[0.2858241]]
[[0.7300518]]
[[0.6129949]]
[[0.5567487]]
[[0.0734347]]
[[0.68005306]]
```

(b) Train a Word2Vec model using your own dataset. You will use these ex- tracted features in the subsequent questions of this assignment. Set the em- bedding size to be 300 and the window size to be 13. You can also consider a minimum word count of 9. Check the semantic similarities for the same two examples in part (a). What do you conclude from comparing vectors generated by yourself and the pretrained model? Which of the Word2Vec models seems to encode semantic similarities between words better?

For the rest of this assignment, use the pretrained “word2vec-google- news-300” Word2Ve features.

In [46]:
```
Xtraining_wv = list(clean_data['review_body'].str.split(" "))
```

In [47]:
```
type(Xtraining_wv[1])
```

Out[47]:
```
list
```

In [48]:
```
embedding_size=300
window_size=13
minimum_count=9
```

In [49]:
```
my_model = gensim.models.Word2Vec(Xtraining_wv, vector_size=embedding_size, wir
```

In [50]:
```
print(cosine_similarity([my_model.wv['excellent']],[my_model.wv['outstanding']]
print("********************")
print(cosine_similarity([wv_model['excellent']],[wv_model['outstanding']]))
```

```
[[0.75836855]]
********************
[[0.5567487]]
```

In [51]:
```
print(cosine_similarity([my_model.wv['he']], [my_model.wv['she']]))
print(cosine_similarity([my_model.wv['excellent']],[my_model.wv['outstanding']]
print(cosine_similarity([my_model.wv['him']], [my_model.wv['father'] - my_model
```

```
[[0.6450126]]
[[0.75836855]]
[[-0.41397798]]
```

In [52]:
```
def embedding_creation(data):
    word_embeddings = []
    for i, rev in enumerate(data):
        w2v_vector = np.zeros((1,300))
        review_words = rev.split(" ")

        for word in review_words:
            if word in wv_model:
                w2v_vector += wv_model[word]

        w2v_vector = w2v_vector/len(review_words)

        word_embeddings.append(w2v_vector)

    word_embeddings_arr = np.array(word_embeddings)

    return word_embeddings_arr
```

In [53]:
```
def concat_embedding_creation(data):
    word_embeddings = []
    for i, rev in enumerate(data):
        w2v_vector = np.array([])
        review_words = rev.split(" ")

        if len(review_words)==0:
            word_embeddings.append(np.zeros(10,300))
            continue

        for word in review_words[:10]:
            if word in wv_model:
                np.reshape(wv_model[word], (1, 300))
```

```
                w2v_vector = np.append(w2v_vector, wv_model[word])
            else:
                w2v_vector = np.append(w2v_vector, np.zeros((1,300 )))

        w2v_vector = w2v_vector[1:]

        if (w2v_vector.shape[0] < 3000):
            padding_req = 3000 - w2v_vector.shape[0]
            w2v_vector = np.append(w2v_vector, np.zeros((1, padding_req)))

        word_embeddings.append(w2v_vector)

    word_embeddings_arr = np.array(word_embeddings)

    return word_embeddings_arr
```

In [54]:
```
def rnn_embedding_creation(data,words):
    word_embeddings =[]

    for i, rev in enumerate(data):
        w2v_vector = []
        padding=np.zeros((1, 300))
        review_words = rev.split(" ")

        if len(review_words)==0:
            w2v_vector.append(padding)
            word_embeddings.append(w2v_vector)
            continue

        for word in review_words[:words]:
            if word in wv_model:
                w2v_vector.append(np.reshape(wv_model[word], (1, 300)))
            else:
                w2v_vector.append(padding)
                continue

        if len(w2v_vector)<words:
            for i in range(words - len(w2v_vector)):
                w2v_vector.append(padding)

        word_embeddings.append(w2v_vector)

    word_embeddings_arr = np.array(word_embeddings)

    return word_embeddings_arr
```

In [55]:
```
%%time
Xtrain_wv = embedding_creation(Xtrain)
Xtest_wv = embedding_creation(Xtest)
ytrain_wv = ytrain.copy()
ytest_wv = ytest.copy()
```

```
CPU times: user 8.84 s, sys: 168 ms, total: 9.01 s
Wall time: 9.01 s
```

Basic Feature extraction

In [56]:
```
Xtrain_wv.shape, Xtest_wv.shape, ytrain_wv.shape, ytest_wv.shape
```

Out[56]:
```
((48000, 1, 300), (12000, 1, 300), (48000,), (12000,))
```

In [57]:
```
Xtrain_wv=Xtrain_wv.reshape(Xtrain_wv.shape[0], Xtrain_wv.shape[2])
Xtest_wv=Xtest_wv.reshape(Xtest_wv.shape[0], Xtest_wv.shape[2])
```

In [58]:
```
Xtrain_wv.shape, Xtest_wv.shape, ytrain_wv.shape, ytest_wv.shape
```

Out[58]:
```
((48000, 300), (12000, 300), (48000,), (12000,))
```

TFIDF

In [59]:
```
#Splitting the Data into train and test data (split should be of 80%-20%)
Xtrain_tf, Xtest_tf, ytrain_tf, ytest_tf = train_test_split(clean_data['review_

print("Training Data Size: ", Xtrain_tf.shape)
print("Testing Data Size: ", Xtest_tf.shape)
```

```
Training Data Size:  (48000,)
Testing Data Size:  (12000,)
```

In [60]:
```
tfID_feat_extract = TfidfVectorizer(
    sublinear_tf=True,
    strip_accents='unicode',
    analyzer='word',
    token_pattern=r'\w{1,}',
    stop_words='english',
    ngram_range=(1, 2),
    max_features=12000
)
```

In [61]:
```
Xtrain_tfid = tfID_feat_extract.fit_transform(Xtrain_tf)

Xtest_tfid = tfID_feat_extract.transform(Xtest_tf)

print("Training document-term matrix : ", Xtrain_tfid)
print("Training feature names for transformation : ", tfID_feat_extract.get_fea
```

```
Training document-term matrix :    (0, 8146)    0.6624755007883363
  (0, 3119)     0.7490835806872576
  (1, 11879)    0.1723174671573481
  (1, 2225)     0.19031129916597092
  (1, 4398)     0.18053073801781588
  (1, 9662)     0.21825716501194986
  (1, 10026)    0.1853538497601683
  (1, 7865)     0.20345399739810266
  (1, 11302)    0.13598005164235638
  (1, 11250)    0.21309423999059418
  (1, 642)      0.19814530232287828
  (1, 7993)     0.2286891114758678
  (1, 5788)     0.22729805131689143
  (1, 7647)     0.09443213275246261
  (1, 11870)    0.11400315647289061
  (1, 2206)     0.11435343433662354
  (1, 6012)     0.11405290514262861
  (1, 9655)     0.12080974406373544
  (1, 9245)     0.14814262511702542
  (1, 10045)    0.2370616914873347
  (1, 1728)     0.16336551404044897
  (1, 3715)     0.15735499039250173
  (1, 1698)     0.1843141791889094
  (1, 4280)     0.1255443929802307
  (1, 10025)    0.14442759338065708
  :     :
  (47998, 10663)        0.09109527665515842
  (47998, 977)  0.1081933253947858
  (47998, 8913) 0.08565213568521633
  (47998, 11543)        0.07801139686939566
  (47998, 6955) 0.1547615125775108
  (47998, 9972) 0.1052559100431006
  (47998, 2327) 0.09802778681367091
  (47998, 9368) 0.10882599254032727
  (47998, 5161) 0.056643398889999995
  (47998, 5609) 0.051554540792254126
  (47998, 11268)        0.06508106749638216
  (47998, 11184)        0.06250636173083554
  (47999, 7471) 0.3528569910161622
  (47999, 4284) 0.35193692707705754
  (47999, 4823) 0.37236613568314664
  (47999, 7470) 0.25564756168505215
  (47999, 1405) 0.33354193765187246
  (47999, 5337) 0.3444127253187343
  (47999, 7965) 0.285742771169963
  (47999, 3821) 0.20823300258723454
  (47999, 645)  0.1601174283993428
  (47999, 4280) 0.12859826480419131
  (47999, 7750) 0.10668973264654914
  (47999, 11184)        0.13957399920914837
  (47999, 9672) 0.3459785533522881
Training feature names for transformation :  ['aa' 'aa batteries' 'aaa' ... 'z
it' 'zits' 'zone']
```

Simple models

Using the Google pre-trained Word2Vec features, train a single perceptron and an SVM model for the classification problem. For this purpose, use the average Word2Vec vectors for each review as the input feature (x = N1 PNi=1 Wi for a review with N words). Report your accuracy values on the testing split for these models similar to HW1, i.e., for each of perceptron and SVM models, report two accuracy values Word2Vec and TF-IDF features. What do you conclude from comparing performances for the models trained using the two different feature types (TF-IDF and your trained Word2Vec features)?

Perceptron

TF-IDF

In [62]:
```
%%time
model_perceptron_tfid = Perceptron(
    alpha=0.00001,
    penalty= 'l2',      #Penalty for wrong prediction
    max_iter=1500,      #Maximum number of iterations
    shuffle=True,
    random_state=16,
    tol=0.001,
)
model_perceptron_tfid=model_perceptron_tfid.fit(Xtrain_tfid , ytrain_tf)
pred_percept2_tfid=model_perceptron_tfid.predict(Xtest_tfid)
result2_tfid=classification_report(ytest_tf, pred_percept2_tfid,output_dict=Tru
print(result2_tfid)

acc2_tfid=accuracy_score(ytest_tf, pred_percept2_tfid)
```

```
{'1': {'precision': 0.6263577118030412, 'recall': 0.6383763837638377, 'f1-scor
e': 0.6323099415204678, 'support': 4065}, '2': {'precision': 0.52895752895752
9, 'recall': 0.5028137998531931, 'f1-score': 0.5155544405418967, 'support': 40
87}, '3': {'precision': 0.6790030211480362, 'recall': 0.7008835758835759, 'f1-
score': 0.689769820971867, 'support': 3848}, 'accuracy': 0.61225, 'macro avg':
{'precision': 0.6114394206362022, 'recall': 0.6140245865002022, 'f1-score': 0.
6125447343447439, 'support': 12000}, 'weighted avg': {'precision': 0.610066428
7255356, 'recall': 0.61225, 'f1-score': 0.6109704318229314, 'support': 12000}}
CPU times: user 234 ms, sys: 308 ms, total: 542 ms
Wall time: 177 ms
```

In [63]:
```
i=1
for keys,values in result2_tfid.items():
    if i==4:
        i=i+1
        continue
    else:
        print(keys,": ",values['precision'],",",values['recall'],",",values['f1
        i=i+1
```

```
1 :  0.6263577118030412 , 0.6383763837638377 , 0.6323099415204678
2 :  0.528957528957529 , 0.5028137998531931 , 0.5155544405418967
3 :  0.6790030211480362 , 0.7008835758835759 , 0.689769820971867
macro avg :  0.6114394206362022 , 0.6140245865002022 , 0.6125447343447439
weighted avg :  0.6100664287255356 , 0.61225 , 0.6109704318229314
```

Word2Vec

In [64]:
```
%%time
model_perceptron_wv = Perceptron(
    alpha=0.00001,
    penalty= 'l2',      #Penalty for wrong prediction
    max_iter=1500,      #Maximum number of iterations
    shuffle=True,
    random_state=16,
    tol=0.001,
)
model_perceptron_wv=model_perceptron_wv.fit(Xtrain_wv , ytrain_wv)
pred_percept2_wv=model_perceptron_wv.predict(Xtest_wv)
result2_wv=classification_report(ytest_wv, pred_percept2_wv,output_dict=True)
print(result2_wv)

acc2_wv=accuracy_score(ytest_wv, pred_percept2_wv)
```

```
{'1': {'precision': 0.5369609856262834, 'recall': 0.7940283400809717, 'f1-scor
e': 0.6406696610861576, 'support': 3952}, '2': {'precision': 0.583279948750800
8, 'recall': 0.4561623246492986, 'f1-score': 0.5119482710149003, 'support': 39
92}, '3': {'precision': 0.7996044825313118, 'recall': 0.5981262327416174, 'f1-
score': 0.6843441466854725, 'support': 4056}, 'accuracy': 0.6154166666666666,
'macro avg': {'precision': 0.6399484723027986, 'recall': 0.6161056324906292,
'f1-score': 0.6123206929288435, 'support': 12000}, 'weighted avg': {'precisio
n': 0.6411432626462724, 'recall': 0.6154166666666666, 'f1-score': 0.6126103214
550211, 'support': 12000}}
CPU times: user 661 ms, sys: 12.6 ms, total: 674 ms
Wall time: 633 ms
```

In [65]:
```
i=1
for keys,values in result2_wv.items():
    if i==4:
        i=i+1
        continue
    else:
        print(keys,": ",values['precision'],",",values['recall'],",",values['f1
        i=i+1
```

```
1 :  0.5369609856262834 , 0.7940283400809717 , 0.6406696610861576
2 :  0.5832799487508008 , 0.4561623246492986 , 0.5119482710149003
3 :  0.7996044825313118 , 0.5981262327416174 , 0.6843441466854725
macro avg :  0.6399484723027986 , 0.6161056324906292 , 0.6123206929288435
weighted avg :  0.6411432626462724 , 0.6154166666666666 , 0.6126103214550211
```

In [66]:
```
print("Perceptron:TF-IDF",acc2_tfid)
print("Perceptron:W2V",acc2_wv)
```

```
Perceptron:TF-IDF 0.61225
Perceptron:W2V 0.6154166666666666
```

SVM

TF-IDF

In [67]:
```
%%time
svm_model_tfid = LinearSVC(
    C=0.35,
    tol=0.001,
    max_iter=1000,                  #Total iterations
    random_state=16,                 #Control the random number generation to co
    penalty='l1',                   #Norm of Penalty
    class_weight="balanced",        #Provides the weight to each class
    loss='squared_hinge',           #Specifies the Loss Function
    dual=False,                     #Selects the algorithm to either the dual or
)
svm_model_tfid=svm_model_tfid.fit(Xtrain_tfid , ytrain_tf)
pred_svm_tfid=svm_model_tfid.predict(Xtest_tfid)
svm_result_tfid=classification_report(ytest_tf, pred_svm_tfid,output_dict=True)
print(svm_result_tfid)

acc3_tfid=accuracy_score(ytest_tf, pred_svm_tfid)
```

```
{'1': {'precision': 0.6952975047984645, 'recall': 0.7129151291512915, 'f1-scor
e': 0.7039961132029637, 'support': 4065}, '2': {'precision': 0.60216546363131
6, 'recall': 0.5307071201370198, 'f1-score': 0.5641825985173624, 'support': 40
87}, '3': {'precision': 0.7167848699763594, 'recall': 0.7879417879417879, 'f1-
score': 0.7506808615994058, 'support': 3848}, 'accuracy': 0.6749166666666667,
'macro avg': {'precision': 0.6714159461353799, 'recall': 0.6771880124100331,
'f1-score': 0.6729531911065774, 'support': 12000}, 'weighted avg': {'precisio
n': 0.6704685655446648, 'recall': 0.6749166666666667, 'f1-score': 0.6713482029
787518, 'support': 12000}}
CPU times: user 940 ms, sys: 16.5 ms, total: 956 ms
Wall time: 957 ms
```

In [68]:
```
i=1
for keys,values in svm_result_tfid.items():
    if i==4:
        i=i+1
        continue
    else:
        print(keys,": ",values['precision'],",",values['recall'],",",values['f1
        i=i+1
```

```
1 :  0.6952975047984645 , 0.7129151291512915 , 0.7039961132029637
2 :  0.602165463631316 , 0.5307071201370198 , 0.5641825985173624
3 :  0.7167848699763594 , 0.7879417879417879 , 0.7506808615994058
macro avg :  0.6714159461353799 , 0.6771880124100331 , 0.6729531911065774
weighted avg :  0.6704685655446648 , 0.6749166666666667 , 0.6713482029787518
```

Word2Vec

In [69]:
```
%%time
svm_model_wv = LinearSVC(
    C=0.35,
    tol=0.001,
    max_iter=1000,                  #Total iterations
    random_state=16,                 #Control the random number generation to co
    penalty='l1',                   #Norm of Penalty
    class_weight="balanced",        #Provides the weight to each class
    loss='squared_hinge',           #Specifies the Loss Function
```

```
    dual=False,                     #Selects the algorithm to either the dual or
)
```

```
CPU times: user 22 µs, sys: 2 µs, total: 24 µs
Wall time: 32.2 µs
```

In [70]:
```
%%time
svm_model_wv=svm_model_wv.fit(Xtrain_wv , ytrain_wv)
```

```
CPU times: user 1min 8s, sys: 410 ms, total: 1min 9s
Wall time: 1min 9s
```

In [71]:
```
%%time
pred_svm_wv=svm_model_wv.predict(Xtest_wv)
```

```
CPU times: user 12.1 ms, sys: 1.11 ms, total: 13.2 ms
Wall time: 16.6 ms
```

In [72]:
```
%%time
svm_result_wv=classification_report(ytest_wv, pred_svm_wv,output_dict=True)
print(svm_result_wv)
acc3_wv=accuracy_score(ytest_wv, pred_svm_wv)
```

```
{'1': {'precision': 0.6381241250583295, 'recall': 0.6920546558704453, 'f1-scor
e': 0.6639961155620296, 'support': 3952}, '2': {'precision': 0.596729490022172
9, 'recall': 0.5393286573146293, 'f1-score': 0.5665789473684212, 'support': 39
92}, '3': {'precision': 0.7289332683877253, 'recall': 0.7379191321499013, 'f1-
score': 0.7333986767949032, 'support': 4056}, 'accuracy': 0.65675, 'macro av
g': {'precision': 0.6545956278227426, 'recall': 0.6564341484449919, 'f1-scor
e': 0.6546579132417847, 'support': 12000}, 'weighted avg': {'precision': 0.655
0470002483039, 'recall': 0.65675, 'f1-score': 0.6550467366396672, 'support': 1
2000}}
CPU times: user 14.1 ms, sys: 1.11 ms, total: 15.2 ms
Wall time: 13.3 ms
```

In [73]:
```
i=1
for keys,values in svm_result_wv.items():
    if i==4:
        i=i+1
        continue
    else:
        print(keys,": ",values['precision'],",",values['recall'],",",values['f1
        i=i+1
```

```
1 :  0.6381241250583295 , 0.6920546558704453 , 0.6639961155620296
2 :  0.5967294900221729 , 0.5393286573146293 , 0.5665789473684212
3 :  0.7289332683877253 , 0.7379191321499013 , 0.7333986767949032
macro avg :  0.6545956278227426 , 0.6564341484449919 , 0.6546579132417847
weighted avg :  0.6550470002483039 , 0.65675 , 0.6550467366396672
```

In [74]:
```
print("SVM:TF-IDF",acc3_tfid)
print("SVM:W2V",acc3_wv)
```

```
SVM:TF-IDF 0.6749166666666667
SVM:W2V 0.65675
```

Creating Iterator for dataloader

In [75]:
```
class creating_iterator(torch.utils.data.Dataset):
```

```
    def __init__(self, review, class_rating):
        self.data = review
        self.labels = class_rating

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        dataset = self.data[index]
        class_label  = self.labels[index]

        return dataset, class_label
```

In [76]:
```
# number of subprocesses to use for data loading
num_workers = 0
# how many samples per batch to load
batch_size = 100
# percentage of training set to use as validation
valid_size = 0.2
```

In [77]:
```
def training_model(loader_train,loader_valid,loader_test,model,optimizer_model,
    # number of epochs to train the model
    n_epochs = 50
    graphs={}
    training_graph = []
    valid_graph=[]
    # initialize tracker for minimum validation loss
    valid_loss_min = np.Inf # set initial "min" to infinity
    for epoch in range(n_epochs):
        # monitor training loss
            train_loss = 0.0
            valid_loss = 0.0
            ###################
            # train the model #
            ###################
            model.train() # prep model for training
            for data, target in loader_train:
                # clear the gradients of all optimized variables
                optimizer_model.zero_grad()
                # forward pass: compute predicted outputs by passing inputs to
                output = model(data)
                # calculate the loss
                loss = criterion_model(output, target)
                # backward pass: compute gradient of the loss with respect to m
                loss.backward()
                # perform a single optimization step (parameter update)
                optimizer_model.step()
                # update running training loss
                train_loss += loss.item()*data.size(0)
            ######################
            # validate the model #
            ######################
            model.eval() # prep model for evaluation
            for data, target in loader_valid:
                # forward pass: compute predicted outputs by passing inputs to
                output = model(data)
                # calculate the loss
                loss = criterion2(output, target)
                # update running validation loss
```

```
            valid_loss += loss.item()*data.size(0)
        # print training/validation statistics
        # calculate average loss over an epoch
        train_loss = train_loss/(len(loader_train)*batch_size)
        valid_loss = valid_loss/(len(loader_test)*batch_size)
        training_graph.append(train_loss)
        valid_graph.append(valid_loss)
        print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'
            epoch+1,
            train_loss,
            valid_loss
            ))

        if valid_loss<valid_loss_min:
            if fnn_tag==True:
                 torch.save(model.state_dict(), 'fnn_sp.pt')
            elif fnn_concat_tag==True:
                 torch.save(model.state_dict(), 'fnn_concat.pt')
            elif rnn_tag==True:
                    torch.save(model.state_dict(), 'rnn_sp.pt')
            elif gru_tag==True:
                    torch.save(model.state_dict(), 'rnn_gru.pt')
            elif lstm_tag==True:
                torch.save(model.state_dict(), 'rnn_lstm.pt')
            valid_loss_min=valid_loss

correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for ou
with torch.no_grad():
  for data in loader_test:
    embeddings, labels = data
    outputs = model(embeddings)
    # the class with the highest score is what we choose as prediction
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()
acc=correct/total

graphs['training']=training_graph
graphs['valid']=valid_graph
graphs['accuracy']=acc

return graphs
```

Feedforward Neural Networks

Using the Word2Vec features, train a feedforward multilayer perceptron net- work for classification. Consider a network with two hidden layers, each with 100 and 10 nodes, respectively. You can use cross entropy loss and your own choice for other hyperparamters, e.g., nonlinearity, number of epochs, etc. Part of getting good results is to select suitable values for these hyper-paramters.

You can also refer to the following tutorial to familiarize yourself: https://www.kaggle.com/mishra1993/pytorch-multi-layer-perceptron-mnist Although the above tutorial is for image data but the concept of training an MLP is very similar to what we want to do.

(a) To generate the input features, use the average Word2Vec vectors similar to the "Simple models" section and train the neural network. Report accuracy values on the testing split for your MLP.

In [78]:
```
from torch.utils.data import DataLoader, Dataset
```

In [79]:
```
%%time
class FNN(nn.Module):
    def __init__(self, input_dim,output_dim):
        super(FNN, self).__init__()
        self.layer1 = nn.Linear(input_dim, 100)
        self.act_func_relu1 = nn.ReLU()
        self.layer2 = nn.Linear(100, 10)
        self.act_func_relu2 = nn.ReLU()
        self.layer3 = nn.Linear(10, output_dim)
        self.dropout = nn.Dropout(0.2)


    def forward(self, x):
        # add hidden layer, with relu activation function
        x = self.act_func_relu1(self.layer1(x))
        # add hidden layer, with relu activation function
        x = self.act_func_relu2(self.layer2(x))
        x = self.dropout(x)
        # add output layer
        x = self.layer3(x)
        return x
```

```
CPU times: user 57 µs, sys: 56 µs, total: 113 µs
Wall time: 120 µs
```

In [80]:
```
%%time
X_train_w2v = Xtrain_wv.astype(np.float32)
X_test_w2v = Xtest_wv.astype(np.float32)
```

```
CPU times: user 21.7 ms, sys: 27.5 ms, total: 49.2 ms
Wall time: 45.2 ms
```

In [81]:
```
X_train_w2v.shape,X_test_w2v.shape
```

Out[81]:
```
((48000, 300), (12000, 300))
```

In [82]:
```
ytrain2=ytrain.copy()
ytest2=ytest.copy()
ytrain2-=1
ytest2-=1
```

In [83]:
```
type(ytrain2)
```

Out[83]:
```
pandas.core.series.Series
```

In [84]:
```
train_dataset_fnn = creating_iterator(X_train_w2v, ytrain2.values)
test_dataset_fnn = creating_iterator(X_test_w2v, ytest2.values)
```

In [85]:
```
len(train_dataset_fnn)
```

Out[85]:
```
48000
```

In [86]:
```
%%time
fnn = FNN(300,3)
print(fnn)
```

```
FNN(
  (layer1): Linear(in_features=300, out_features=100, bias=True)
  (act_func_relu1): ReLU()
  (layer2): Linear(in_features=100, out_features=10, bias=True)
  (act_func_relu2): ReLU()
  (layer3): Linear(in_features=10, out_features=3, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
CPU times: user 3.26 ms, sys: 9.01 ms, total: 12.3 ms
Wall time: 29.5 ms
```

In [87]:
```
# obtain training indices that will be used for validation
num_train_fnn = len(train_dataset_fnn)
indices_fnn = list(range(num_train_fnn))
np.random.shuffle(indices_fnn)
split_fnn = int(np.floor(valid_size * num_train_fnn))
train_idx_fnn, valid_idx_fnn = indices_fnn[split_fnn:], indices_fnn[:split_fnn]
# define samples for obtaining training and validation batches
train_sampler_fnn = SubsetRandomSampler(train_idx_fnn)
valid_sampler_fnn = SubsetRandomSampler(valid_idx_fnn)
# prepare data loaders
train_loader_fnn = torch.utils.data.DataLoader(train_dataset_fnn, batch_size=ba
                                               sampler=train_sampler_fnn, num_worke
                                               =num_workers)
valid_loader_fnn = torch.utils.data.DataLoader(train_dataset_fnn, batch_size=ba
                                               sampler=valid_sampler_fnn, num_worke
                                               =num_workers)
test_loader_fnn = torch.utils.data.DataLoader(test_dataset_fnn, batch_size=batc
                                               num_workers=num_workers)
```

In [88]:
```
len(train_loader_fnn)
```

Out[88]:
```
384
```

In [89]:
```
len(valid_loader_fnn)
```

Out[89]:
```
96
```

In [90]:
```
len(valid_loader_fnn.dataset)
```

Out[90]:
```
48000
```

In [91]:
```
# Define the loss
criterion2 = nn.CrossEntropyLoss()
optimizer2 = Adam(fnn.parameters(), lr=1e-2)
```

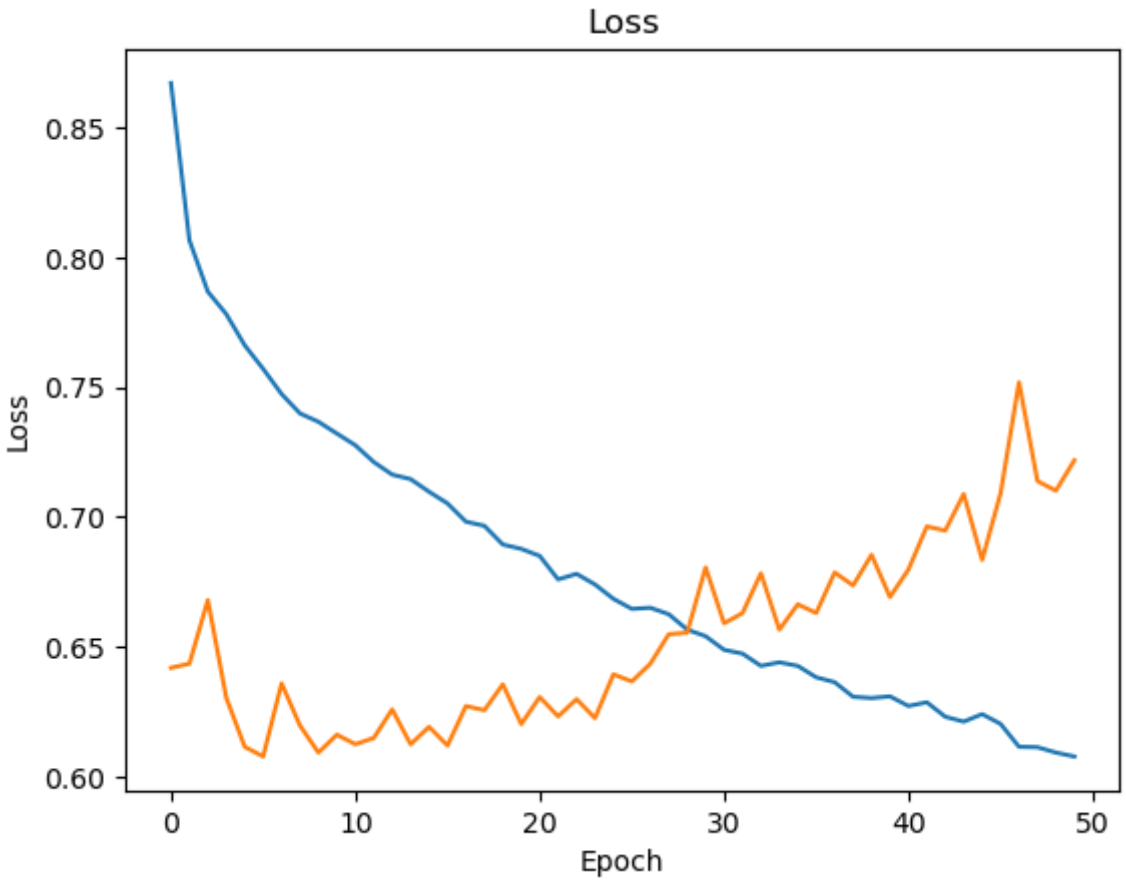
In [92]:
```
fnn_training= training_model(train_loader_fnn,valid_loader_fnn,test_loader_fnn,
```

```
Epoch: 1        Training Loss: 0.867081         Validation Loss: 0.641912
Epoch: 2        Training Loss: 0.806540         Validation Loss: 0.643512
Epoch: 3        Training Loss: 0.786754         Validation Loss: 0.668028
Epoch: 4        Training Loss: 0.778119         Validation Loss: 0.630390
Epoch: 5        Training Loss: 0.766102         Validation Loss: 0.611500
Epoch: 6        Training Loss: 0.757036         Validation Loss: 0.607796
Epoch: 7        Training Loss: 0.747344         Validation Loss: 0.635927
Epoch: 8        Training Loss: 0.739859         Validation Loss: 0.619654
Epoch: 9        Training Loss: 0.736651         Validation Loss: 0.609269
Epoch: 10       Training Loss: 0.732119         Validation Loss: 0.616176
Epoch: 11       Training Loss: 0.727626         Validation Loss: 0.612491
Epoch: 12       Training Loss: 0.721159         Validation Loss: 0.614901
Epoch: 13       Training Loss: 0.716318         Validation Loss: 0.625917
Epoch: 14       Training Loss: 0.714574         Validation Loss: 0.612465
Epoch: 15       Training Loss: 0.709711         Validation Loss: 0.619202
Epoch: 16       Training Loss: 0.705219         Validation Loss: 0.611985
Epoch: 17       Training Loss: 0.698193         Validation Loss: 0.627200
Epoch: 18       Training Loss: 0.696603         Validation Loss: 0.625594
Epoch: 19       Training Loss: 0.689400         Validation Loss: 0.635567
Epoch: 20       Training Loss: 0.687717         Validation Loss: 0.620248
Epoch: 21       Training Loss: 0.685016         Validation Loss: 0.630684
Epoch: 22       Training Loss: 0.675967         Validation Loss: 0.623213
Epoch: 23       Training Loss: 0.678107         Validation Loss: 0.629858
Epoch: 24       Training Loss: 0.674004         Validation Loss: 0.622548
Epoch: 25       Training Loss: 0.668478         Validation Loss: 0.639416
Epoch: 26       Training Loss: 0.664616         Validation Loss: 0.636674
Epoch: 27       Training Loss: 0.665042         Validation Loss: 0.643493
Epoch: 28       Training Loss: 0.662564         Validation Loss: 0.654856
Epoch: 29       Training Loss: 0.656628         Validation Loss: 0.655440
Epoch: 30       Training Loss: 0.654094         Validation Loss: 0.680582
Epoch: 31       Training Loss: 0.648845         Validation Loss: 0.659039
Epoch: 32       Training Loss: 0.647392         Validation Loss: 0.662981
Epoch: 33       Training Loss: 0.642666         Validation Loss: 0.678301
Epoch: 34       Training Loss: 0.644032         Validation Loss: 0.656603
Epoch: 35       Training Loss: 0.642657         Validation Loss: 0.666376
Epoch: 36       Training Loss: 0.638260         Validation Loss: 0.663031
Epoch: 37       Training Loss: 0.636379         Validation Loss: 0.678614
Epoch: 38       Training Loss: 0.630765         Validation Loss: 0.673648
Epoch: 39       Training Loss: 0.630286         Validation Loss: 0.685422
Epoch: 40       Training Loss: 0.630921         Validation Loss: 0.669170
Epoch: 41       Training Loss: 0.627215         Validation Loss: 0.679740
Epoch: 42       Training Loss: 0.628649         Validation Loss: 0.696363
Epoch: 43       Training Loss: 0.623073         Validation Loss: 0.694765
Epoch: 44       Training Loss: 0.621192         Validation Loss: 0.708861
Epoch: 45       Training Loss: 0.624083         Validation Loss: 0.683476
Epoch: 46       Training Loss: 0.620317         Validation Loss: 0.709118
Epoch: 47       Training Loss: 0.611511         Validation Loss: 0.751876
Epoch: 48       Training Loss: 0.611357         Validation Loss: 0.713829
Epoch: 49       Training Loss: 0.609281         Validation Loss: 0.710131
Epoch: 50       Training Loss: 0.607766         Validation Loss: 0.721818
```

In [93]:
```
print("Accuracy of Simple FNN(a):", fnn_training['accuracy']*100, "%")
```

```
Accuracy of Simple FNN(a): 64.25833333333333 %
```

In [94]:
```
plt.plot(fnn_training['training'])
plt.plot(fnn_training['valid'])
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```



(b) To generate the input features, concatenate the first 10 Word2Vec vectors for each review as the input feature (x = [W T , ..., W T]) and train the neural network. Report the accuracy value on the testing split for your MLP model. What do you conclude by comparing accuracy values you obtain with those obtained in the "'Simple Models" section.

In [95]:
```
%%time
class FNN2(nn.Module):
    def __init__(self, input_dim,output_dim):
        super(FNN2, self).__init__()
        self.layer1 = nn.Linear(input_dim, 300)
        self.act_func_relu1 = nn.ReLU()
        self.layer2 = nn.Linear(300, 100)
        self.act_func_relu2 = nn.ReLU()
        self.layer3 = nn.Linear(100, output_dim)


    def forward(self, x):
        # add hidden layer, with relu activation function
        x = self.act_func_relu1(self.layer1(x))
```

```
        # add hidden layer, with relu activation function
        x = self.act_func_relu2(self.layer2(x))
        # add output layer
        x = self.layer3(x)
        return x
```

```
CPU times: user 74 µs, sys: 2 µs, total: 76 µs
Wall time: 85.8 µs
```

In [96]:
```
Xtrain_wv_concat = concat_embedding_creation(Xtrain)
Xtest_wv_concat = concat_embedding_creation(Xtest)
```

In [97]:
```
Xtrain_wv_concat.shape
```

Out[97]:
```
(48000, 3000)
```

In [98]:
```
%%time
X_train_w2v_concat = Xtrain_wv_concat.astype(np.float32)
X_test_w2v_concat = Xtest_wv_concat.astype(np.float32)
```

```
CPU times: user 145 ms, sys: 258 ms, total: 403 ms
Wall time: 499 ms
```

In [99]:
```
ytrain_cc=ytrain.copy()
ytest_cc=ytest.copy()
ytrain_cc-=1
ytest_cc-=1
```

In [100…
```
train_iter_concat = creating_iterator(X_train_w2v_concat, ytrain_cc.values)
test_iter_concat = creating_iterator(X_test_w2v_concat, ytest_cc.values)
```

In [101…
```
# obtain training indices that will be used for validation
num_train_fnn2 = len(train_iter_concat)
indices_fnn2 = list(range(num_train_fnn2))
np.random.shuffle(indices_fnn2)
split_fnn2 = int(np.floor(valid_size * num_train_fnn2))
train_idx_fnn2, valid_idx_fnn2 = indices_fnn2[split_fnn2:], indices_fnn2[:split
# define samples for obtaining training and validation batches
train_sampler_fnn2 = SubsetRandomSampler(train_idx_fnn2)
valid_sampler_fnn2 = SubsetRandomSampler(valid_idx_fnn2)
# prepare data loaders
train_loader_fnn2 = torch.utils.data.DataLoader(train_iter_concat, batch_size=b
                                                sampler=train_sampler_fnn2, num_work
                                                =num_workers)
valid_loader_fnn2 = torch.utils.data.DataLoader(train_iter_concat, batch_size=b
                                                sampler=valid_sampler_fnn2, num_work
                                                =num_workers)
test_loader_fnn2 = torch.utils.data.DataLoader(test_iter_concat, batch_size=bat
                                                num_workers=num_workers)
```

In [102…
```
fnn_concat=FNN2(3000,3)
```

In [103…
```
print(fnn_concat)
```

```
FNN2(
  (layer1): Linear(in_features=3000, out_features=300, bias=True)
  (act_func_relu1): ReLU()
  (layer2): Linear(in_features=300, out_features=100, bias=True)
  (act_func_relu2): ReLU()
  (layer3): Linear(in_features=100, out_features=3, bias=True)
)
```

In [104…

```
# Define the loss
criterion2_concat = nn.CrossEntropyLoss()

# Optimizers require the parameters to optimize and a learning rate
optimizer2_concat = SGD(fnn_concat.parameters(),lr=0.01)
```

In [105…

```
fnn_concat_training= training_model(train_loader_fnn2,valid_loader_fnn2,test_lo
```

```
Epoch: 1        Training Loss: 1.098284         Validation Loss: 0.877949
Epoch: 2        Training Loss: 1.096443         Validation Loss: 0.876501
Epoch: 3        Training Loss: 1.094012         Validation Loss: 0.874071
Epoch: 4        Training Loss: 1.089573         Validation Loss: 0.869267
Epoch: 5        Training Loss: 1.080790         Validation Loss: 0.859711
Epoch: 6        Training Loss: 1.063322         Validation Loss: 0.841251
Epoch: 7        Training Loss: 1.032254         Validation Loss: 0.812581
Epoch: 8        Training Loss: 0.993642         Validation Loss: 0.785019
Epoch: 9        Training Loss: 0.962896         Validation Loss: 0.767103
Epoch: 10       Training Loss: 0.942668         Validation Loss: 0.756266
Epoch: 11       Training Loss: 0.928341         Validation Loss: 0.748751
Epoch: 12       Training Loss: 0.916863         Validation Loss: 0.741636
Epoch: 13       Training Loss: 0.906708         Validation Loss: 0.736608
Epoch: 14       Training Loss: 0.897011         Validation Loss: 0.731720
Epoch: 15       Training Loss: 0.887718         Validation Loss: 0.727283
Epoch: 16       Training Loss: 0.878928         Validation Loss: 0.723259
Epoch: 17       Training Loss: 0.870146         Validation Loss: 0.719158
Epoch: 18       Training Loss: 0.861838         Validation Loss: 0.716203
Epoch: 19       Training Loss: 0.854146         Validation Loss: 0.715347
Epoch: 20       Training Loss: 0.846685         Validation Loss: 0.711592
Epoch: 21       Training Loss: 0.839480         Validation Loss: 0.709746
Epoch: 22       Training Loss: 0.833334         Validation Loss: 0.710371
Epoch: 23       Training Loss: 0.827162         Validation Loss: 0.706970
Epoch: 24       Training Loss: 0.820915         Validation Loss: 0.707345
Epoch: 25       Training Loss: 0.815063         Validation Loss: 0.707149
Epoch: 26       Training Loss: 0.809386         Validation Loss: 0.706467
Epoch: 27       Training Loss: 0.803440         Validation Loss: 0.705945
Epoch: 28       Training Loss: 0.798235         Validation Loss: 0.707195
Epoch: 29       Training Loss: 0.792376         Validation Loss: 0.706874
Epoch: 30       Training Loss: 0.787041         Validation Loss: 0.707389
Epoch: 31       Training Loss: 0.781157         Validation Loss: 0.709354
Epoch: 32       Training Loss: 0.775364         Validation Loss: 0.709890
Epoch: 33       Training Loss: 0.769385         Validation Loss: 0.710758
Epoch: 34       Training Loss: 0.763183         Validation Loss: 0.711552
Epoch: 35       Training Loss: 0.756283         Validation Loss: 0.711355
Epoch: 36       Training Loss: 0.749813         Validation Loss: 0.712571
Epoch: 37       Training Loss: 0.742764         Validation Loss: 0.713159
Epoch: 38       Training Loss: 0.734759         Validation Loss: 0.714658
Epoch: 39       Training Loss: 0.727686         Validation Loss: 0.716457
Epoch: 40       Training Loss: 0.718536         Validation Loss: 0.716423
Epoch: 41       Training Loss: 0.709323         Validation Loss: 0.722736
Epoch: 42       Training Loss: 0.700427         Validation Loss: 0.722546
Epoch: 43       Training Loss: 0.689519         Validation Loss: 0.723724
Epoch: 44       Training Loss: 0.679131         Validation Loss: 0.725134
Epoch: 45       Training Loss: 0.667083         Validation Loss: 0.726468
Epoch: 46       Training Loss: 0.654328         Validation Loss: 0.730252
Epoch: 47       Training Loss: 0.641276         Validation Loss: 0.732476
Epoch: 48       Training Loss: 0.626772         Validation Loss: 0.735234
Epoch: 49       Training Loss: 0.612172         Validation Loss: 0.744841
Epoch: 50       Training Loss: 0.596077         Validation Loss: 0.743507
```

In [106…

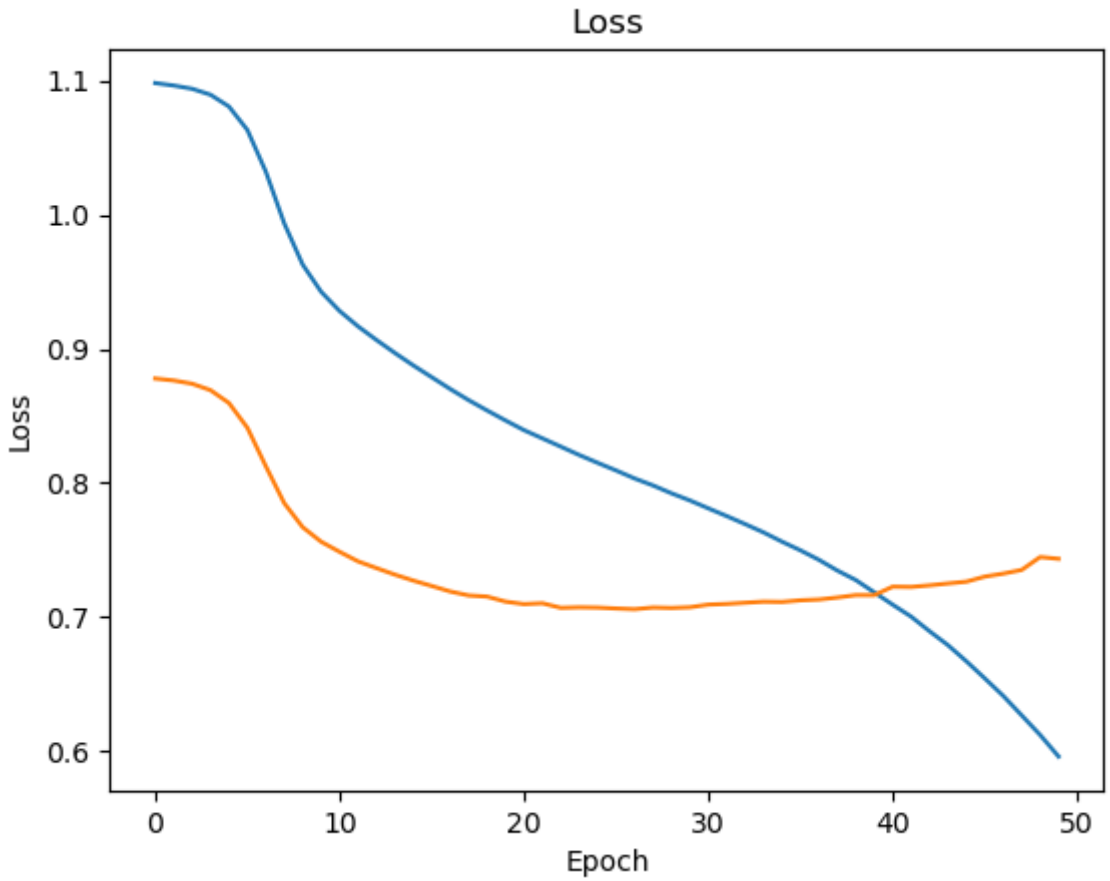
```
print("Accuracy of FNN CONCAT(b):", fnn_concat_training['accuracy']*100, "%")
```

```
Accuracy of FNN CONCAT(b): 56.78333333333333 %
```

In [107…

```
plt.plot(fnn_concat_training['training'])
plt.plot(fnn_concat_training['valid'])
plt.title('Loss')
plt.xlabel('Epoch')
```

```
plt.ylabel('Loss')
plt.show()
```



Recurrent Neural Networks

Using the Word2Vec features, train a recurrent neural network (RNN) for classification. You can refer to the following tutorial to familiarize yourself: https://pytorch.org/tutorials/intermediate/char_rnn_classification tutorial.html

(a) Train a simple RNN for sentiment analysis. You can consider an RNN cell with the hidden state size of 20. To feed your data into our RNN, limit the maximum review length to 20 by truncating longer reviews and padding shorter reviews with a null value (0). Report accuracy values on the testing split for your RNN model. What do you conclude by comparing accuracy values you obtain with those obtained with feedforward neural network models.

In [108…

```
class RNN(nn.Module):

    def __init__(self, input_size, output_size, n_layers):
        super(RNN, self).__init__()
        self.layer_dim = n_layers
```

```
        self.rnn = nn.RNN(input_size, 20, n_layers,batch_first=True,nonlinearit
        self.linear = nn.Linear(20, output_size)


    def forward(self, x):
        h0 = torch.zeros(self.layer_dim, x.size(0), 20)
        out, hn = self.rnn(x, h0)
        return self.linear(out[:, -1, :])
```

In [109…
```
Xtrain_wv_rnn = rnn_embedding_creation(Xtrain,20)
Xtest_wv_rnn = rnn_embedding_creation(Xtest,20)
```

In [110…
```
Xtrain_wv_rnn.shape,Xtest_wv_rnn.shape
```

Out[110]:
```
((48000, 20, 1, 300), (12000, 20, 1, 300))
```

In [111…
```
Xtrain_wv_rnn=Xtrain_wv_rnn.reshape(Xtrain_wv_rnn.shape[0], Xtrain_wv_rnn.shape
Xtest_wv_rnn=Xtest_wv_rnn.reshape(Xtest_wv_rnn.shape[0], Xtest_wv_rnn.shape[1],
```

In [112…
```
Xtrain_wv_rnn.shape,Xtest_wv_rnn.shape
```

Out[112]:
```
((48000, 20, 300), (12000, 20, 300))
```

In [113…
```
%%time
Xtrain_w2v_rnn = Xtrain_wv_rnn.astype(np.float32)
Xtest_w2v_rnn = Xtest_wv_rnn.astype(np.float32)
```

```
CPU times: user 392 ms, sys: 1.12 s, total: 1.52 s
Wall time: 2.05 s
```

In [114…
```
ytrain_rnn=ytrain.copy()
ytest_rnn=ytest.copy()
ytrain_rnn-=1
ytest_rnn-=1
```

In [115…
```
train_iter_rnn = creating_iterator(Xtrain_w2v_rnn, ytrain_rnn.values)
test_iter_rnn = creating_iterator(Xtest_w2v_rnn, ytest_rnn.values)
```

In [116…
```
batch_size=128
```

In [117…
```
# obtain training indices that will be used for validation
num_train_rnn = len(train_iter_rnn)
indices_rnn = list(range(num_train_rnn))
np.random.shuffle(indices_rnn)
split_rnn = int(np.floor(valid_size * num_train_rnn))
train_idx_rnn, valid_idx_rnn = indices_rnn[split_rnn:], indices_rnn[:split_rnn]
# define samples for obtaining training and validation batches
train_sampler_rnn = SubsetRandomSampler(train_idx_rnn)
valid_sampler_rnn = SubsetRandomSampler(valid_idx_rnn)
# prepare data loaders
train_loader_rnn = torch.utils.data.DataLoader(train_iter_rnn, batch_size=batch
                                               sampler=train_sampler_rnn, num_worke
                                               =num_workers)
valid_loader_rnn = torch.utils.data.DataLoader(train_iter_rnn, batch_size=batch
                                               sampler=valid_sampler_rnn, num_worke
                                               =num_workers)
```

```
test_loader_rnn = torch.utils.data.DataLoader(test_iter_rnn, batch_size=batch_s
                                              num_workers=num_workers)
```

In [118…
```
rnn = RNN(300,3, 2)
```

In [119…
```
print(rnn)
```

```
RNN(
  (rnn): RNN(300, 20, num_layers=2, batch_first=True)
  (linear): Linear(in_features=20, out_features=3, bias=True)
)
```

In [120…
```
%%time
# Define the loss
criterion_rnn = nn.CrossEntropyLoss()
optimizer_rnn = Adam(rnn.parameters(), lr=0.001)
```

```
CPU times: user 204 µs, sys: 353 µs, total: 557 µs
Wall time: 1.23 ms
```

In [121…
```
rnn_training= training_model(train_loader_rnn,valid_loader_rnn,test_loader_rnn,
```

```
Epoch: 1        Training Loss: 1.038148         Validation Loss: 0.750417
Epoch: 2        Training Loss: 0.910006         Validation Loss: 0.718806
Epoch: 3        Training Loss: 0.885437         Validation Loss: 0.701595
Epoch: 4        Training Loss: 0.868465         Validation Loss: 0.697088
Epoch: 5        Training Loss: 0.854279         Validation Loss: 0.687160
Epoch: 6        Training Loss: 0.837399         Validation Loss: 0.669486
Epoch: 7        Training Loss: 0.825702         Validation Loss: 0.661943
Epoch: 8        Training Loss: 0.814410         Validation Loss: 0.649913
Epoch: 9        Training Loss: 0.798486         Validation Loss: 0.648665
Epoch: 10       Training Loss: 0.789032         Validation Loss: 0.649274
Epoch: 11       Training Loss: 0.781514         Validation Loss: 0.638348
Epoch: 12       Training Loss: 0.772689         Validation Loss: 0.650237
Epoch: 13       Training Loss: 0.767457         Validation Loss: 0.635193
Epoch: 14       Training Loss: 0.762820         Validation Loss: 0.638569
Epoch: 15       Training Loss: 0.758486         Validation Loss: 0.637563
Epoch: 16       Training Loss: 0.750956         Validation Loss: 0.635531
Epoch: 17       Training Loss: 0.750315         Validation Loss: 0.632561
Epoch: 18       Training Loss: 0.745433         Validation Loss: 0.635775
Epoch: 19       Training Loss: 0.741177         Validation Loss: 0.635691
Epoch: 20       Training Loss: 0.738590         Validation Loss: 0.634802
Epoch: 21       Training Loss: 0.737025         Validation Loss: 0.629275
Epoch: 22       Training Loss: 0.731387         Validation Loss: 0.631895
Epoch: 23       Training Loss: 0.733199         Validation Loss: 0.631754
Epoch: 24       Training Loss: 0.731042         Validation Loss: 0.631616
Epoch: 25       Training Loss: 0.723972         Validation Loss: 0.634543
Epoch: 26       Training Loss: 0.720706         Validation Loss: 0.632828
Epoch: 27       Training Loss: 0.719077         Validation Loss: 0.636120
Epoch: 28       Training Loss: 0.721736         Validation Loss: 0.639137
Epoch: 29       Training Loss: 0.715786         Validation Loss: 0.638989
Epoch: 30       Training Loss: 0.710998         Validation Loss: 0.636480
Epoch: 31       Training Loss: 0.709780         Validation Loss: 0.642488
Epoch: 32       Training Loss: 0.708967         Validation Loss: 0.634287
Epoch: 33       Training Loss: 0.706072         Validation Loss: 0.634023
Epoch: 34       Training Loss: 0.705707         Validation Loss: 0.640249
Epoch: 35       Training Loss: 0.702905         Validation Loss: 0.645203
Epoch: 36       Training Loss: 0.701250         Validation Loss: 0.643901
Epoch: 37       Training Loss: 0.697063         Validation Loss: 0.650431
Epoch: 38       Training Loss: 0.697273         Validation Loss: 0.636596
Epoch: 39       Training Loss: 0.697918         Validation Loss: 0.634059
Epoch: 40       Training Loss: 0.692007         Validation Loss: 0.646634
Epoch: 41       Training Loss: 0.691046         Validation Loss: 0.640724
Epoch: 42       Training Loss: 0.696693         Validation Loss: 0.639060
Epoch: 43       Training Loss: 0.687067         Validation Loss: 0.660488
Epoch: 44       Training Loss: 0.685686         Validation Loss: 0.641817
Epoch: 45       Training Loss: 0.683178         Validation Loss: 0.645093
Epoch: 46       Training Loss: 0.687520         Validation Loss: 0.653528
Epoch: 47       Training Loss: 0.685251         Validation Loss: 0.661264
Epoch: 48       Training Loss: 0.684189         Validation Loss: 0.643561
Epoch: 49       Training Loss: 0.678391         Validation Loss: 0.659635
Epoch: 50       Training Loss: 0.683223         Validation Loss: 0.649098
```

In [122…

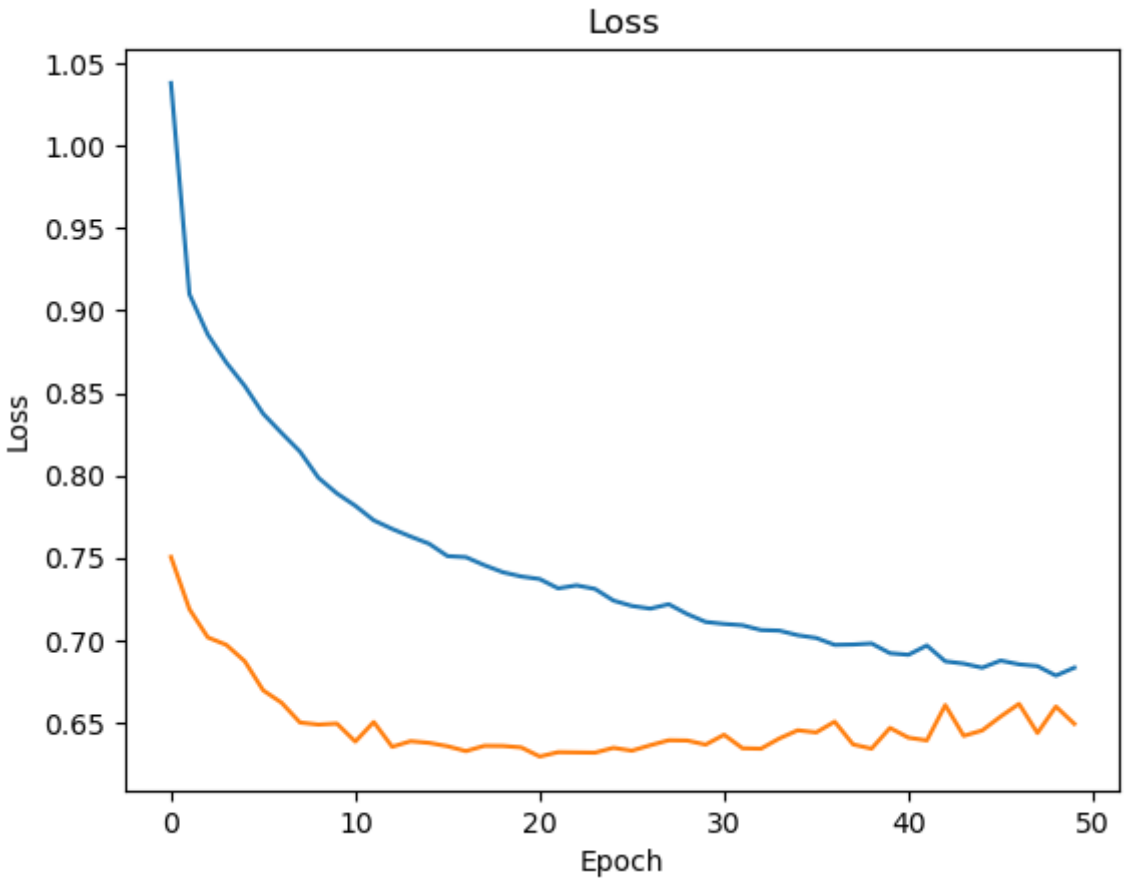
```
print("Accuracy of Simple RNN:", rnn_training['accuracy']*100, "%")
```

```
Accuracy of Simple RNN: 64.35833333333333 %
```

In [123…

```
plt.plot(rnn_training['training'])
plt.plot(rnn_training['valid'])
plt.title('Loss')
plt.xlabel('Epoch')
```

```
plt.ylabel('Loss')
plt.show()
```



(b) Repeat part (a) by considering a gated recurrent unit cell.

In [124…
```
class GRU(nn.Module):

    def __init__(self, input_size, output_size, n_layers):
        super(GRU, self).__init__()
        self.layer_dim = n_layers
        self.gru = nn.GRU(input_size, 20, n_layers,batch_first=True)
        self.linear = nn.Linear(20, output_size)


    def forward(self, x):
        h0 = torch.zeros(self.layer_dim, x.size(0), 20)
        out, hn = self.gru(x, h0)
        return self.linear(out[:, -1, :])
```

In [125…
```
Xtrain_wv_gru = rnn_embedding_creation(Xtrain,20)
Xtest_wv_gru = rnn_embedding_creation(Xtest,20)
```

In [126…
```
Xtrain_wv_gru.shape,Xtest_wv_gru.shape
```

Out[126]:
```
((48000, 20, 1, 300), (12000, 20, 1, 300))
```

In [127…
```
Xtrain_wv_gru=Xtrain_wv_gru.reshape(Xtrain_wv_gru.shape[0], Xtrain_wv_gru.shape
Xtest_wv_gru=Xtest_wv_gru.reshape(Xtest_wv_gru.shape[0], Xtest_wv_gru.shape[1],
```

In [128…
```
Xtrain_wv_rnn.shape,Xtest_wv_rnn.shape
```

Out[128]:
```
((48000, 20, 300), (12000, 20, 300))
```

In [129…
```
%%time
Xtrain_w2v_gru = Xtrain_wv_gru.astype(np.float32)
Xtest_w2v_gru = Xtest_wv_gru.astype(np.float32)
```

```
CPU times: user 399 ms, sys: 1.09 s, total: 1.49 s
Wall time: 2.06 s
```

In [130…
```
ytrain_gru=ytrain.copy()
ytest_gru=ytest.copy()
ytrain_gru-=1
ytest_gru-=1
```

In [131…
```
train_iter_gru = creating_iterator(Xtrain_w2v_gru, ytrain_gru.values)
test_iter_gru = creating_iterator(Xtest_w2v_gru, ytest_gru.values)
```

In [132…
```
# obtain training indices that will be used for validation
num_train_gru = len(train_iter_gru)
indices_gru = list(range(num_train_gru))
np.random.shuffle(indices_gru)
split_gru = int(np.floor(valid_size * num_train_gru))
train_idx_gru, valid_idx_gru = indices_gru[split_gru:], indices_gru[:split_gru]
# define samples for obtaining training and validation batches
train_sampler_gru = SubsetRandomSampler(train_idx_gru)
valid_sampler_gru = SubsetRandomSampler(valid_idx_gru)
# prepare data loaders
train_loader_gru = torch.utils.data.DataLoader(train_iter_gru, batch_size=batch
                                               sampler=train_sampler_gru, num_worke
                                               =num_workers)
valid_loader_gru = torch.utils.data.DataLoader(train_iter_gru, batch_size=batch
                                               sampler=valid_sampler_gru, num_worke
                                               =num_workers)
test_loader_gru = torch.utils.data.DataLoader(test_iter_gru, batch_size=batch_s
                                               num_workers=num_workers)
```

In [133…
```
gru = GRU(300,3,1)
print(gru)
```

```
GRU(
  (gru): GRU(300, 20, batch_first=True)
  (linear): Linear(in_features=20, out_features=3, bias=True)
)
```

In [134…
```
%%time
# Define the loss
criterion_gru = nn.CrossEntropyLoss()
optimizer_gru = Adam(gru.parameters(), lr=0.001)
```

```
CPU times: user 191 µs, sys: 541 µs, total: 732 µs
Wall time: 1.2 ms
```

In [135…
```
gru_training= training_model(train_loader_gru,valid_loader_gru,test_loader_gru,
```

```
Epoch: 1        Training Loss: 1.022177         Validation Loss: 0.722165
Epoch: 2        Training Loss: 0.851648         Validation Loss: 0.662355
Epoch: 3        Training Loss: 0.801167         Validation Loss: 0.643314
Epoch: 4        Training Loss: 0.778774         Validation Loss: 0.634348
Epoch: 5        Training Loss: 0.764040         Validation Loss: 0.627755
Epoch: 6        Training Loss: 0.751524         Validation Loss: 0.622646
Epoch: 7        Training Loss: 0.742109         Validation Loss: 0.619686
Epoch: 8        Training Loss: 0.732592         Validation Loss: 0.628114
Epoch: 9        Training Loss: 0.726363         Validation Loss: 0.613076
Epoch: 10       Training Loss: 0.718723         Validation Loss: 0.612470
Epoch: 11       Training Loss: 0.711153         Validation Loss: 0.611610
Epoch: 12       Training Loss: 0.705909         Validation Loss: 0.609304
Epoch: 13       Training Loss: 0.698711         Validation Loss: 0.617331
Epoch: 14       Training Loss: 0.693084         Validation Loss: 0.609956
Epoch: 15       Training Loss: 0.687433         Validation Loss: 0.624194
Epoch: 16       Training Loss: 0.682538         Validation Loss: 0.612453
Epoch: 17       Training Loss: 0.678467         Validation Loss: 0.612712
Epoch: 18       Training Loss: 0.672568         Validation Loss: 0.613405
Epoch: 19       Training Loss: 0.669101         Validation Loss: 0.614900
Epoch: 20       Training Loss: 0.663659         Validation Loss: 0.612826
Epoch: 21       Training Loss: 0.658181         Validation Loss: 0.615661
Epoch: 22       Training Loss: 0.654292         Validation Loss: 0.618871
Epoch: 23       Training Loss: 0.650905         Validation Loss: 0.613207
Epoch: 24       Training Loss: 0.647777         Validation Loss: 0.623629
Epoch: 25       Training Loss: 0.643337         Validation Loss: 0.623633
Epoch: 26       Training Loss: 0.640449         Validation Loss: 0.623937
Epoch: 27       Training Loss: 0.634949         Validation Loss: 0.623114
Epoch: 28       Training Loss: 0.630227         Validation Loss: 0.622955
Epoch: 29       Training Loss: 0.626617         Validation Loss: 0.626996
Epoch: 30       Training Loss: 0.624232         Validation Loss: 0.631953
Epoch: 31       Training Loss: 0.620168         Validation Loss: 0.628998
Epoch: 32       Training Loss: 0.618185         Validation Loss: 0.628164
Epoch: 33       Training Loss: 0.612322         Validation Loss: 0.632371
Epoch: 34       Training Loss: 0.609466         Validation Loss: 0.640622
Epoch: 35       Training Loss: 0.608117         Validation Loss: 0.643133
Epoch: 36       Training Loss: 0.605594         Validation Loss: 0.640290
Epoch: 37       Training Loss: 0.598768         Validation Loss: 0.654089
Epoch: 38       Training Loss: 0.595464         Validation Loss: 0.652403
Epoch: 39       Training Loss: 0.594467         Validation Loss: 0.663945
Epoch: 40       Training Loss: 0.588580         Validation Loss: 0.669450
Epoch: 41       Training Loss: 0.587467         Validation Loss: 0.669607
Epoch: 42       Training Loss: 0.581874         Validation Loss: 0.658865
Epoch: 43       Training Loss: 0.580531         Validation Loss: 0.660543
Epoch: 44       Training Loss: 0.577829         Validation Loss: 0.656413
Epoch: 45       Training Loss: 0.575292         Validation Loss: 0.664162
Epoch: 46       Training Loss: 0.571220         Validation Loss: 0.663289
Epoch: 47       Training Loss: 0.568214         Validation Loss: 0.665069
Epoch: 48       Training Loss: 0.563560         Validation Loss: 0.675971
Epoch: 49       Training Loss: 0.562096         Validation Loss: 0.687775
Epoch: 50       Training Loss: 0.558843         Validation Loss: 0.682648
```

In [136…

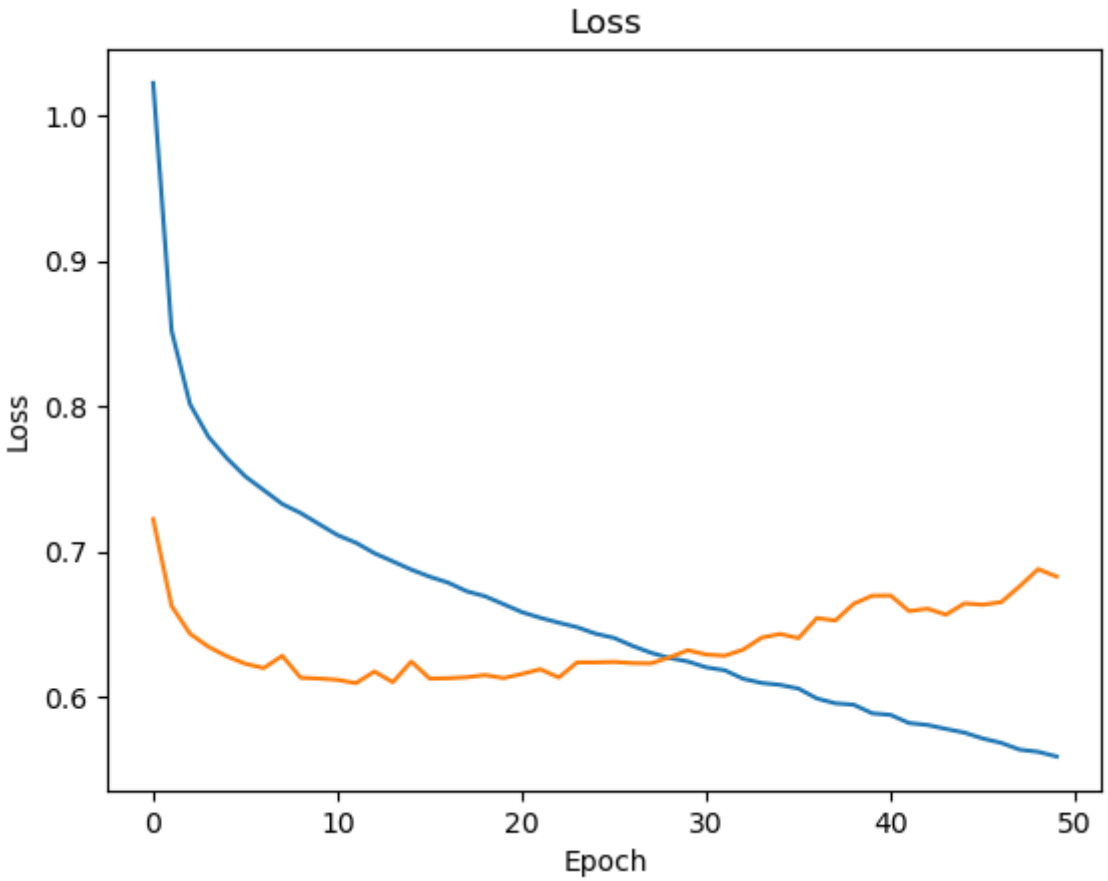
```
print("Accuracy of Gated RNN(GRU):", gru_training['accuracy']*100, "%")
```

```
Accuracy of Gated RNN(GRU): 64.80833333333334 %
```

In [137…

```
plt.plot(gru_training['training'])
plt.plot(gru_training['valid'])
plt.title('Loss')
plt.xlabel('Epoch')
```

```
plt.ylabel('Loss')
plt.show()
```



(c) Repeat part (a) by considering an LSTM unit cell. What do you conclude by comparing accuracy values you obtain by GRU, LSTM, and simple RNN.

https://cnvrg.io/pytorch-lstm/

In [138…
```
from torch.autograd import Variable
```

In [139…
```
class LSTM1(nn.Module):
    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(LSTM1, self).__init__()
        self.num_classes = num_classes #number of classes
        self.num_layers = num_layers #number of layers
        self.input_size = input_size #input size
        self.hidden_size = hidden_size #hidden state

        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True) #lstm
        self.fc_1 =  nn.Linear(hidden_size, 20) #fully connected 1
        self.fc = nn.Linear(20, num_classes) #fully connected last layer

        self.relu = nn.ReLU()

    def forward(self,x):
        h_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size
```

```
        c_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size
        # Propagate input through LSTM
        output, (hn, cn) = self.lstm(x, (h_0, c_0)) #lstm with input, hidden, a
        hn = hn.view(-1, self.hidden_size) #reshaping the data for Dense layer
        out = self.relu(hn)
        out = self.fc_1(out) #first Dense
        out = self.relu(out) #relu
        out = self.fc(out) #Final Output
        return out
```

In [ ]:

In [140…
```
Xtrain_wv_lstm = rnn_embedding_creation(Xtrain,20)
Xtest_wv_lstm = rnn_embedding_creation(Xtest,20)
```

In [141…
```
Xtrain_wv_lstm.shape,Xtest_wv_lstm.shape
```

Out[141]: ((48000, 20, 1, 300), (12000, 20, 1, 300))

In [142…
```
Xtrain_wv_lstm=Xtrain_wv_lstm.reshape(Xtrain_wv_lstm.shape[0], Xtrain_wv_lstm.s
Xtest_wv_lstm=Xtest_wv_lstm.reshape(Xtest_wv_lstm.shape[0], Xtest_wv_lstm.shape
```

In [143…
```
Xtrain_wv_lstm.shape,Xtest_wv_lstm.shape
```

Out[143]: ((48000, 20, 300), (12000, 20, 300))

In [144…
```
%%time
Xtrain_w2v_lstm = Xtrain_wv_lstm.astype(np.float32)
Xtest_w2v_lstm = Xtest_wv_lstm.astype(np.float32)
```

```
CPU times: user 395 ms, sys: 1.1 s, total: 1.49 s
Wall time: 2.07 s
```

In [145…
```
ytrain_lstm=ytrain.copy()
ytest_lstm=ytest.copy()
ytrain_lstm-=1
ytest_lstm-=1
```

In [146…
```
train_iter_lstm = creating_iterator(Xtrain_w2v_lstm, ytrain_lstm.values)
test_iter_lstm = creating_iterator(Xtest_w2v_lstm, ytest_lstm.values)
```

In [147…
```
# obtain training indices that will be used for validation
num_train_lstm = len(train_iter_lstm )
indices_lstm  = list(range(num_train_lstm ))
np.random.shuffle(indices_lstm)
split_lstm  = int(np.floor(valid_size * num_train_lstm))
train_idx_lstm , valid_idx_lstm  = indices_lstm [split_lstm :], indices_lstm [:
# define samples for obtaining training and validation batches
train_sampler_lstm  = SubsetRandomSampler(train_idx_lstm )
valid_sampler_lstm  = SubsetRandomSampler(valid_idx_lstm )
# prepare data loaders
train_loader_lstm  = torch.utils.data.DataLoader(train_iter_lstm , batch_size=b
                                                 sampler=train_sampler_lstm , num_wor
                                                 =num_workers)
valid_loader_lstm  = torch.utils.data.DataLoader(train_iter_lstm , batch_size=b
                                                 sampler=valid_sampler_lstm , num_wor
                                                 =num_workers)
```

```
test_loader_lstm  = torch.utils.data.DataLoader(test_iter_lstm , batch_size=bat
                                                num_workers=num_workers)
```

In [148…
```
lstm=LSTM1(3, 300, 100, 1)
print(lstm)
```

```
LSTM1(
  (lstm): LSTM(300, 100, batch_first=True)
  (fc_1): Linear(in_features=100, out_features=20, bias=True)
  (fc): Linear(in_features=20, out_features=3, bias=True)
  (relu): ReLU()
)
```

In [149…
```
%%time
# Define the loss
criterion_lstm = nn.CrossEntropyLoss()
optimizer_lstm = Adam(lstm.parameters(), lr=0.001)
```

```
CPU times: user 297 µs, sys: 382 µs, total: 679 µs
Wall time: 686 µs
```

In [150…
```
lstm_training= training_model(train_loader_lstm,valid_loader_lstm,test_loader_l
```

```
Epoch: 1        Training Loss: 1.041656         Validation Loss: 0.769787
Epoch: 2        Training Loss: 0.901610         Validation Loss: 0.702990
Epoch: 3        Training Loss: 0.840728         Validation Loss: 0.678224
Epoch: 4        Training Loss: 0.803132         Validation Loss: 0.647252
Epoch: 5        Training Loss: 0.778518         Validation Loss: 0.632175
Epoch: 6        Training Loss: 0.755769         Validation Loss: 0.625676
Epoch: 7        Training Loss: 0.735035         Validation Loss: 0.626346
Epoch: 8        Training Loss: 0.718143         Validation Loss: 0.618812
Epoch: 9        Training Loss: 0.705130         Validation Loss: 0.619269
Epoch: 10       Training Loss: 0.684753         Validation Loss: 0.648606
Epoch: 11       Training Loss: 0.669900         Validation Loss: 0.631620
Epoch: 12       Training Loss: 0.649312         Validation Loss: 0.633651
Epoch: 13       Training Loss: 0.633205         Validation Loss: 0.620956
Epoch: 14       Training Loss: 0.615123         Validation Loss: 0.637743
Epoch: 15       Training Loss: 0.597723         Validation Loss: 0.635925
Epoch: 16       Training Loss: 0.575642         Validation Loss: 0.655939
Epoch: 17       Training Loss: 0.558401         Validation Loss: 0.656022
Epoch: 18       Training Loss: 0.532063         Validation Loss: 0.678698
Epoch: 19       Training Loss: 0.504497         Validation Loss: 0.684345
Epoch: 20       Training Loss: 0.479033         Validation Loss: 0.723727
Epoch: 21       Training Loss: 0.448287         Validation Loss: 0.747542
Epoch: 22       Training Loss: 0.426741         Validation Loss: 0.792411
Epoch: 23       Training Loss: 0.397710         Validation Loss: 0.859475
Epoch: 24       Training Loss: 0.371949         Validation Loss: 0.869129
Epoch: 25       Training Loss: 0.343949         Validation Loss: 0.909657
Epoch: 26       Training Loss: 0.309259         Validation Loss: 0.986532
Epoch: 27       Training Loss: 0.294537         Validation Loss: 1.053719
Epoch: 28       Training Loss: 0.267174         Validation Loss: 1.096772
Epoch: 29       Training Loss: 0.239945         Validation Loss: 1.164962
Epoch: 30       Training Loss: 0.226401         Validation Loss: 1.235370
Epoch: 31       Training Loss: 0.206715         Validation Loss: 1.287979
Epoch: 32       Training Loss: 0.191126         Validation Loss: 1.362047
Epoch: 33       Training Loss: 0.172187         Validation Loss: 1.428923
Epoch: 34       Training Loss: 0.160762         Validation Loss: 1.465843
Epoch: 35       Training Loss: 0.150532         Validation Loss: 1.567807
Epoch: 36       Training Loss: 0.135930         Validation Loss: 1.617378
Epoch: 37       Training Loss: 0.130844         Validation Loss: 1.717212
Epoch: 38       Training Loss: 0.119326         Validation Loss: 1.687678
Epoch: 39       Training Loss: 0.113869         Validation Loss: 1.896215
Epoch: 40       Training Loss: 0.114741         Validation Loss: 1.925643
Epoch: 41       Training Loss: 0.107272         Validation Loss: 1.917114
Epoch: 42       Training Loss: 0.103590         Validation Loss: 1.916996
Epoch: 43       Training Loss: 0.091597         Validation Loss: 1.986938
Epoch: 44       Training Loss: 0.096153         Validation Loss: 2.009193
Epoch: 45       Training Loss: 0.107831         Validation Loss: 1.960078
Epoch: 46       Training Loss: 0.093044         Validation Loss: 2.093747
Epoch: 47       Training Loss: 0.082525         Validation Loss: 2.186416
Epoch: 48       Training Loss: 0.085747         Validation Loss: 2.163405
Epoch: 49       Training Loss: 0.078839         Validation Loss: 2.176399
Epoch: 50       Training Loss: 0.073805         Validation Loss: 2.267855
```

In [151…

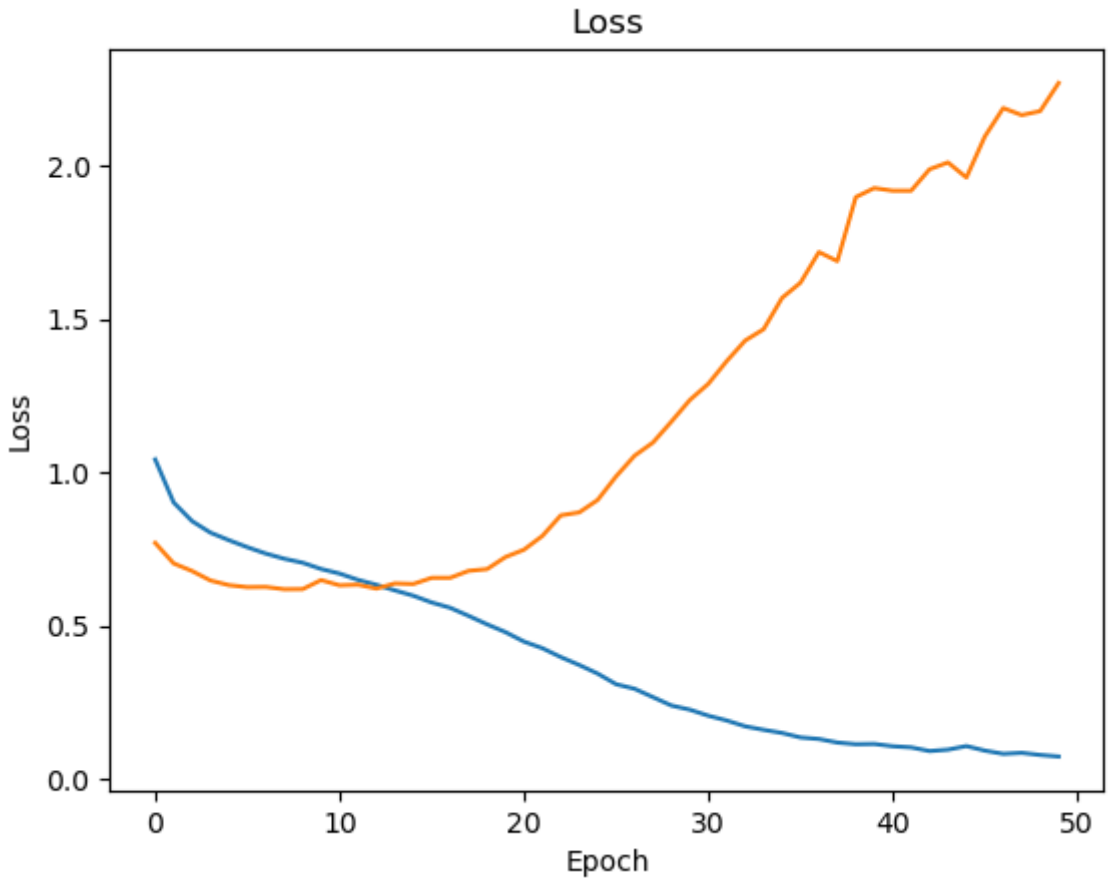
```
print("Accuracy of LSTM:", lstm_training['accuracy']*100, "%")
```

```
Accuracy of LSTM: 61.49166666666667 %
```

In [152…

```
plt.plot(lstm_training['training'])
plt.plot(lstm_training['valid'])
plt.title('Loss')
plt.xlabel('Epoch')
```

```
plt.ylabel('Loss')
plt.show()
```



Trial 2-LSTM

In [153…
```
class LSTM(nn.Module):
    def __init__(self, num_classes, layers,hidden):
        super(LSTM, self).__init__()
        self.lstm = nn.LSTM(300, hidden, layers, batch_first=True)
        self.linear = nn.Linear(hidden, num_classes)

    def forward(self, x):
        return self.linear(self.lstm(x)[0][:, -1])
```

In [154…
```
lstm_tr=LSTM(3, 1, 100)
print(lstm_tr)
```

```
LSTM(
  (lstm): LSTM(300, 100, batch_first=True)
  (linear): Linear(in_features=100, out_features=3, bias=True)
)
```

In [155…
```
%%time
# Define the loss
criterion_lstm_tr = nn.CrossEntropyLoss()
optimizer_lstm_tr = Adam(lstm_tr.parameters(), lr=0.001)
```

```
CPU times: user 120 µs, sys: 7 µs, total: 127 µs
Wall time: 133 µs
```

In [156…
```
lstm_tr_training= training_model(train_loader_lstm,valid_loader_lstm,test_loade
```

```
Epoch: 1        Training Loss: 0.937277         Validation Loss: 0.703239
Epoch: 2        Training Loss: 0.835579         Validation Loss: 0.656238
Epoch: 3        Training Loss: 0.797458         Validation Loss: 0.643381
Epoch: 4        Training Loss: 0.774061         Validation Loss: 0.655957
Epoch: 5        Training Loss: 0.755125         Validation Loss: 0.630401
Epoch: 6        Training Loss: 0.733704         Validation Loss: 0.623344
Epoch: 7        Training Loss: 0.718144         Validation Loss: 0.612256
Epoch: 8        Training Loss: 0.701914         Validation Loss: 0.608945
Epoch: 9        Training Loss: 0.686249         Validation Loss: 0.610177
Epoch: 10       Training Loss: 0.669696         Validation Loss: 0.619321
Epoch: 11       Training Loss: 0.648644         Validation Loss: 0.610671
Epoch: 12       Training Loss: 0.633079         Validation Loss: 0.615332
Epoch: 13       Training Loss: 0.618180         Validation Loss: 0.634767
Epoch: 14       Training Loss: 0.600233         Validation Loss: 0.623061
Epoch: 15       Training Loss: 0.582290         Validation Loss: 0.649076
Epoch: 16       Training Loss: 0.562837         Validation Loss: 0.645566
Epoch: 17       Training Loss: 0.544740         Validation Loss: 0.680291
Epoch: 18       Training Loss: 0.525433         Validation Loss: 0.699376
Epoch: 19       Training Loss: 0.505495         Validation Loss: 0.674257
Epoch: 20       Training Loss: 0.480430         Validation Loss: 0.699437
Epoch: 21       Training Loss: 0.459226         Validation Loss: 0.729265
Epoch: 22       Training Loss: 0.438418         Validation Loss: 0.743538
Epoch: 23       Training Loss: 0.411005         Validation Loss: 0.824777
Epoch: 24       Training Loss: 0.388613         Validation Loss: 0.831074
Epoch: 25       Training Loss: 0.362120         Validation Loss: 0.901657
Epoch: 26       Training Loss: 0.339139         Validation Loss: 0.930574
Epoch: 27       Training Loss: 0.314702         Validation Loss: 0.944591
Epoch: 28       Training Loss: 0.300459         Validation Loss: 1.016028
Epoch: 29       Training Loss: 0.270958         Validation Loss: 1.061831
Epoch: 30       Training Loss: 0.252244         Validation Loss: 1.091642
Epoch: 31       Training Loss: 0.226339         Validation Loss: 1.186979
Epoch: 32       Training Loss: 0.216500         Validation Loss: 1.236558
Epoch: 33       Training Loss: 0.205673         Validation Loss: 1.215405
Epoch: 34       Training Loss: 0.196783         Validation Loss: 1.275524
Epoch: 35       Training Loss: 0.168830         Validation Loss: 1.381594
Epoch: 36       Training Loss: 0.159146         Validation Loss: 1.386130
Epoch: 37       Training Loss: 0.146305         Validation Loss: 1.426149
Epoch: 38       Training Loss: 0.143633         Validation Loss: 1.512832
Epoch: 39       Training Loss: 0.121372         Validation Loss: 1.546126
Epoch: 40       Training Loss: 0.134398         Validation Loss: 1.628262
Epoch: 41       Training Loss: 0.124853         Validation Loss: 1.524572
Epoch: 42       Training Loss: 0.110429         Validation Loss: 1.690084
Epoch: 43       Training Loss: 0.102784         Validation Loss: 1.777067
Epoch: 44       Training Loss: 0.101351         Validation Loss: 1.778308
Epoch: 45       Training Loss: 0.096919         Validation Loss: 1.874336
Epoch: 46       Training Loss: 0.095084         Validation Loss: 1.864301
Epoch: 47       Training Loss: 0.077412         Validation Loss: 1.975696
Epoch: 48       Training Loss: 0.092432         Validation Loss: 1.920643
Epoch: 49       Training Loss: 0.091005         Validation Loss: 1.883733
Epoch: 50       Training Loss: 0.095988         Validation Loss: 1.913565
```

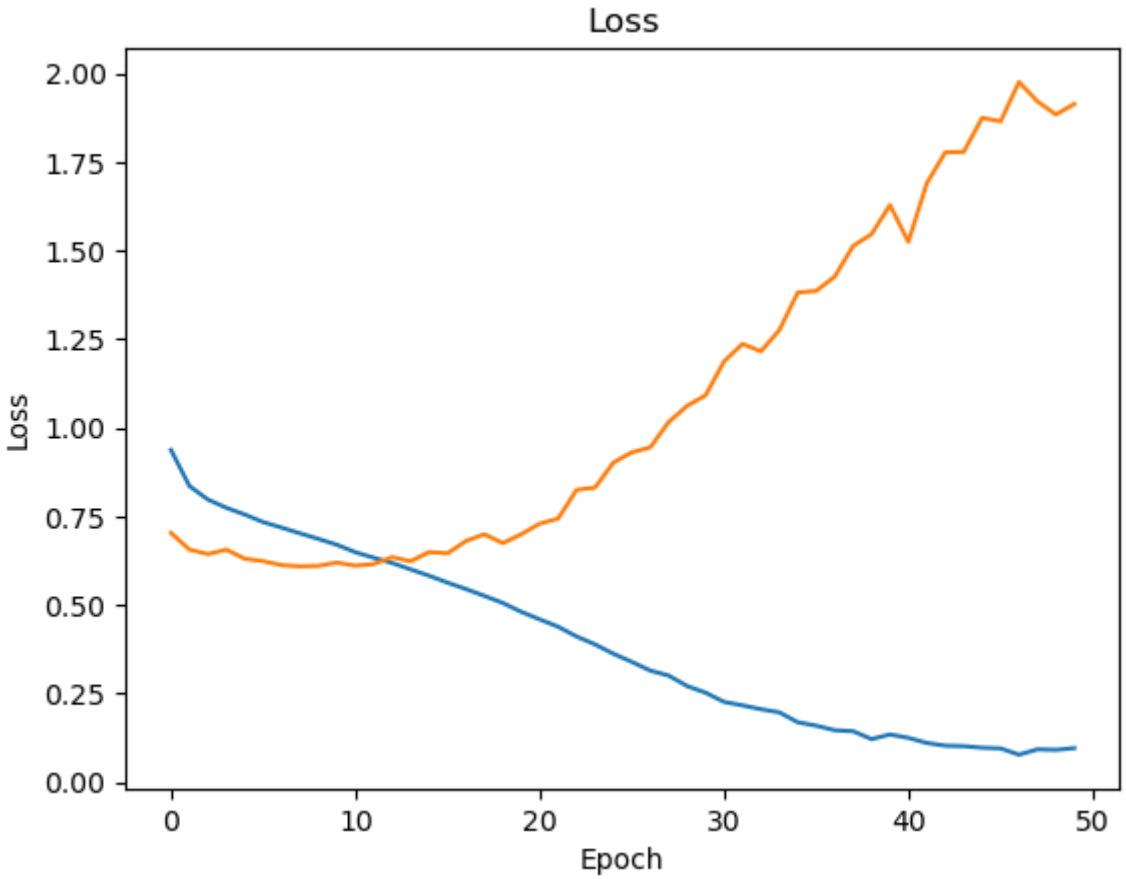
In [157…
```
lstm_tr_training['accuracy']*100
```

Out[157]:
```
61.88333333333333
```

In [158…
```
print("Accuracy of LSTM:", lstm_tr_training['accuracy']*100, "%")
```

```
Accuracy of LSTM: 61.88333333333333 %
```

In [159…
```
plt.plot(lstm_tr_training['training'])
plt.plot(lstm_tr_training['valid'])
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```



References

In [ ]:
```
# https://www.kaggle.com/code/mishra1993/pytorch-multi-layer-perceptron-mnist/r
# https://builtin.com/machine-learning/nlp-word2vec-python
# https://python-bloggers.com/2022/05/building-a-pytorch-binary-classification-
# https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with
# https://machinelearningmastery.com/pytorch-tutorial-develop-deep-learning-moc
# https://stackoverflow.com/questions/70804697/why-is-my-pytorch-classificatior
# https://deborahmesquita.com/2017-11-05/how-pytorch-gives-the-big-picture-with
# https://medium.com/swlh/text-classification-using-scikit-learn-pytorch-and-te
# https://www.projectpro.io/recipes/develop-mlp-for-multiclass-classification-p
# https://www.analyticsvidhya.com/blog/2020/01/first-text-classification-in-pyt
# https://galhever.medium.com/sentiment-analysis-with-pytorch-part-5-mlp-model-
# https://bhadreshpsavani.medium.com/tutorial-on-sentimental-analysis-using-pyt
# https://dipikabaad.medium.com/finding-the-hidden-sentiments-using-rnns-in-pyt
# https://medium.com/analytics-vidhya/creating-a-custom-dataset-and-dataloader-
# https://pytorch.org/tutorials/recipes/recipes/custom_dataset_transforms_loade
```

```
# https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
# https://pytorch.org/tutorials/recipes/recipes/custom_dataset_transforms_loade
# https://towardsdatascience.com/custom-datasets-in-pytorch-part-2-text-machine
# https://towardsdatascience.com/how-to-use-datasets-and-dataloader-in-pytorch-
# https://cnvrg.io/pytorch-lstm/
# https://blog.paperspace.com/dataloaders-abstractions-pytorch/
# https://www.kaggle.com/code/pierremegret/gensim-word2vec-tutorial/notebook
#
```

In [ ]:
```
# https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html
# https://www.guru99.com/word-embedding-word2vec.html
# https://radimrehurek.com/gensim/models/word2vec.html
# https://machinelearningmastery.com/develop-word-embedding-model-predicting-mo
# https://machinelearningmastery.com/develop-word-embedding-model-predicting-mo
# https://medium.com/data-science-lab-spring-2021/amazon-review-rating-predicti
# http://yaronvazana.com/2018/09/20/average-word-vectors-generate-document-para
# https://medium.com/analytics-vidhya/i-strongly-recommend-to-first-know-how-rn
# https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_fe
# https://www.kaggle.com/code/mehmetlaudatekman/lstm-text-classification-pytorc
# https://towardsdatascience.com/building-a-lstm-by-hand-on-pytorch-59c02a4ec09
# https://selbydavid.com/scrooge/reference/cosine_similarity.html
```

In [ ]:
```

```

In [ ]:
```

```