HW2-CSCI544-RNN-Part 5

October 5, 2021

[2]:
```python
# import required libraries and methods from them

from platform import python_version

import pandas as pd
import numpy as np

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

import re

from bs4 import BeautifulSoup

import contractions

import gensim
import gensim.downloader as api

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset,DataLoader
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/mrinalkadam/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/mrinalkadam/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/mrinalkadam/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

[3]:
```python
# check the python version being used by the jupyter notebook

python_version()
```

[3]:
```
'3.8.5'
```

[4]:
```python
# read the input dataset into a dataframe

df = pd.read_csv("data.tsv", sep='\t', quoting=3)
df
```

[4]:
```
        marketplace  customer_id       review_id  product_id  product_parent  \
0                US     37000337  R3DT59XH7HXR9K  B00303FI0G       529320574
1                US     15272914  R1LFS11BNASSU8  B00JCZKZN6       274237558
2                US     36137863  R296RT05AG0AF6  B00JLIKA5C       544675303
3                US     43311049  R3V37XDZ7ZCI3L  B000GBNB8G       491599489
4                US     13763148  R14GU232NQFYX2  B00VJ5KX9S       353790155
...             ...          ...             ...         ...             ...
4880461          US     51094108  R22DLC2P26MUMR  B00004SBGS       732420532
4880462          US     50562512  R1N6KLTENLQOMT  B00004SBIA       261705371
4880463          US     52469742  R10TW4QXDV8KJC  B00004SPEF       191184892
4880464          US     51865238   R41RL2U1FSQ4V  B00004RHR6       912491903
4880465          US     52900320  R1NHMPKSJG2E37  B0000021VO        41913389

                                              product_title product_category  \
0                              Arthur Court Paper Towel Holder          Kitchen
1         Olde Thompson Bavaria Glass Salt and Pepper Mi...          Kitchen
2         Progressive International PL8 Professional Man...          Kitchen
3                                  Zyliss Jumbo Garlic Press          Kitchen
4         1 X Premier Pizza Cutter - Stainless Steel 14"...          Kitchen
...                                                     ...              ...
4880461   Le Creuset Enameled Cast-Iron 6-3/4-Quart Oval...          Kitchen
4880462   Le Creuset Enameled Cast-Iron 2-Quart Heart Ca...          Kitchen
4880463            Krups 358-70 La Glaciere Ice Cream Maker          Kitchen
4880464          Hoffritz Stainless-Steel Manual Can Opener          Kitchen
4880465                                        Tammy Rogers          Kitchen

         star_rating  helpful_votes  total_votes vine verified_purchase  \
0                  5              0            0    N                 Y
```

```
1                5           0           1    N             Y
2                5           0           0    N             Y
3                5           0           1    N             Y
4                5           0           0    N             Y
...            ...         ...         ...  ...           ...
4880461          4          30          41    N             N
4880462          5          84          92    N             N
4880463          4          55          60    N             N
4880464          4          30          42    N             N
4880465          5           5           5    N             N

                                review_headline  \
0              Beautiful. Looks great on counter
1                           Awesome & Self-ness
2                 Fabulous and worth every penny
3                                    Five Stars
4                               Better than sex
...                                         ...
4880461             Not as sturdy as you'd think.
4880462                      A Sweetheart of A Pan
4880463                     Ice Cream Like a Dream
4880464            Opens anything and everything
4880465  The more you listen, the more you hear...

                                               review_body review_date
0                        Beautiful.  Looks great on counter.  2015-08-31
1        I personally have 5 days sets and have also bo...  2015-08-31
2        Fabulous and worth every penny. Used for clean...  2015-08-31
3        A must if you love garlic on tomato marinara s...  2015-08-31
4        Worth every penny! Buy one now and be a pizza ...  2015-08-31
...                                                    ...         ...
4880461  After a month of heavy use, primarily as a chi...  2000-04-28
4880462  I've used my Le Creuset enameled cast iron coo...  2000-04-28
4880463  According to my wife, this is \\"the best birt...  2000-04-28
4880464  Hoffritz has a name of producing a trendy and ...  2000-04-24
4880465  OK. I was late to snap to the Dead Reckoners. ...  2000-01-20

[4880466 rows x 15 columns]
```

1 1. Dataset Generation

[5]:
```python
# keep only reviews and ratings columns

df = df[["review_body","star_rating"]]
df
```

[5]:
```
                                               review_body  star_rating
0                       Beautiful.  Looks great on counter.            5
1        I personally have 5 days sets and have also bo...            5
2        Fabulous and worth every penny. Used for clean...            5
3        A must if you love garlic on tomato marinara s...            5
4        Worth every penny! Buy one now and be a pizza ...            5
...                                                    ...          ...
4880461  After a month of heavy use, primarily as a chi...            4
4880462  I've used my Le Creuset enameled cast iron coo...            5
4880463  According to my wife, this is \\"the best birt...            4
4880464  Hoffritz has a name of producing a trendy and ...            4
4880465  OK. I was late to snap to the Dead Reckoners. ...            5

[4880466 rows x 2 columns]
```

[6]:
```python
# find out the number of reviews falling under each distinct rating

df['star_rating'].value_counts()
```

[6]:
```
5    3128564
4     732471
1     427306
3     349929
2     242196
Name: star_rating, dtype: int64
```

[7]:
```python
# check for null values in the reviews column

df['review_body'].isnull().sum()
```

[7]: 243

[8]:
```python
# check for null values in the ratings column

df['star_rating'].isnull().sum()
```

[8]: 0

[9]:
```python
# drop null value records from the dataframe

df.dropna(inplace=True)
```

```
<ipython-input-9-ba0c96652bb5>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.dropna(inplace=True)
```

[10]:
```
# find out records with star ratings 1,2,3,4 and 5 and select 50000 records␣
↪randomly per each rating score

df_1 = df[df['star_rating']==1].sample(n=50000, random_state=100)
df_2 = df[df['star_rating']==2].sample(n=50000, random_state=100)
df_3 = df[df['star_rating']==3].sample(n=50000, random_state=100)
df_4 = df[df['star_rating']==4].sample(n=50000, random_state=100)
df_5 = df[df['star_rating']==5].sample(n=50000, random_state=100)

# concat the above records together to get a sample of 250000 reviews

df = pd.concat([df_1,df_2,df_3,df_4,df_5]).reset_index()

# shuffle the dataset

df = df.sample(frac=1).reset_index()
df.drop(['index','level_0'],axis=1,inplace=True)
df
```

[10]:
```
                                              review_body  star_rating
0       Learning to use any new piece of technology ca...            3
1       Not exactly what expected, a little hard to as...            3
2       I followed all of the hints and the egg still ...            1
3       This electric kettle does what it's supposed t...            4
4       I kept hearing about the benefits of this kett...            5
...                                                   ...          ...
249995  Got for the ice tray. Not very good, turned brown            1
249996  These look lovely on my open shelves.  They ar...            4
249997  Broke down in less than two years. Not a cheap...            1
249998  Product was given as a gift but found out late...            1
249999  Perfect size and suitable to fry or make soup ...            5

[250000 rows x 2 columns]
```

[11]:
```
# find out the number of reviews falling under distinct ratings now

print("Positive, Negative, Neutral Reviews Count:")
print(df[((df['star_rating']==4.0) | (df['star_rating']==5.0))]['star_rating'].
↪count(),",",df[((df['star_rating']==1.0) | (df['star_rating']==2.
↪0))]['star_rating'].count(),",",df[df['star_rating']==3.0]['star_rating'].
↪count())
```

```
Positive, Negative, Neutral Reviews Count:
100000 , 100000 , 50000
```

[12]:
```
# label reviews falling under ratings 4 and 5 as 1(positive class), under␣
↪ratings 1 and 2 as 2(negative class), and under rating 3 as 3(neutral class)
```

```
df['class'] = np.where(((df['star_rating']==4) | (df['star_rating']==5)),1,0)
df['class'] = np.where(((df['star_rating']==1) |␣
↪(df['star_rating']==2)),2,df['class'])
df['class'] = np.where((df['star_rating']==3),3,df['class'])
df
```

[12]:

```
                                              review_body  star_rating  class
0       Learning to use any new piece of technology ca...            3      3
1       Not exactly what expected, a little hard to as...            3      3
2       I followed all of the hints and the egg still ...            1      2
3       This electric kettle does what it's supposed t...            4      1
4       I kept hearing about the benefits of this kett...            5      1
...                                                   ...          ...    ...
249995  Got for the ice tray. Not very good, turned brown            1      2
249996  These look lovely on my open shelves.  They ar...            4      1
249997  Broke down in less than two years. Not a cheap...            1      2
249998  Product was given as a gift but found out late...            1      2
249999  Perfect size and suitable to fry or make soup ...            5      1

[250000 rows x 3 columns]
```

[13]:

```
# drop the rating column once you have the label('class') column

df.drop(['star_rating'],axis=1,inplace=True)
df
```

[13]:

```
                                              review_body  class
0       Learning to use any new piece of technology ca...      3
1       Not exactly what expected, a little hard to as...      3
2       I followed all of the hints and the egg still ...      2
3       This electric kettle does what it's supposed t...      1
4       I kept hearing about the benefits of this kett...      1
...                                                   ...    ...
249995  Got for the ice tray. Not very good, turned brown      2
249996  These look lovely on my open shelves.  They ar...      1
249997  Broke down in less than two years. Not a cheap...      2
249998  Product was given as a gift but found out late...      2
249999  Perfect size and suitable to fry or make soup ...      1

[250000 rows x 2 columns]
```

[14]:

```
# make a copy of the original data frame(without any data cleaning)

df_uncleaned = df.copy(deep = True)
df_uncleaned
```

[14]:
```
                                               review_body  class
0       Learning to use any new piece of technology ca...      3
1       Not exactly what expected, a little hard to as...      3
2       I followed all of the hints and the egg still ...      2
3       This electric kettle does what it's supposed t...      1
4       I kept hearing about the benefits of this kett...      1
...                                                   ...    ...
249995  Got for the ice tray. Not very good, turned brown      2
249996  These look lovely on my open shelves.  They ar...      1
249997  Broke down in less than two years. Not a cheap...      2
249998  Product was given as a gift but found out late...      2
249999  Perfect size and suitable to fry or make soup ...      1

[250000 rows x 2 columns]
```

2 2. Word Embedding

3 (a)

[15]:
```python
# load the google news word2vec model

wv = api.load('word2vec-google-news-300')
```

4 (b)

[16]:
```python
##### REMOVE FROM COMMENT LATER

words = [row.split(' ') for row in df['review_body']]

# train your own word2vec model

model = gensim.models.Word2Vec(words, min_count=10,size=300,workers=3,␣
 ↪window=11, sg=1)

# summarize the loaded model

print(model)
```

[17]:
```python
# save model

model.save('model.bin')

# load saved model
```

```
final_model = gensim.models.Word2Vec.load('model.bin')
print(final_model)
```

```
Word2Vec(vocab=34607, size=300, alpha=0.025)
```

5 4. Feedforward Neural Networks

[18]:
```
# set hyperparameters for all the models

EPOCHS = 50
BATCH_SIZE = 20
LEARNING_RATE = 0.001
```

[19]:
```
## train data
class trainData(Dataset):

    def __init__(self, x_data, y_data):
        self.x_data = x_data
        self.y_data = y_data

    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

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

## test data
class testData(Dataset):

    def __init__(self, x_data):
        self.x_data = x_data

    def __getitem__(self, index):
        return self.x_data[index]

    def __len__ (self):
        return len(self.x_data)
```

6 (a)

7 Binary

[20]:
```
# function to find the accuracy of the binary model

def binary_acc(y_pred, y_test):
    y_pred_tag = torch.round(torch.sigmoid(y_pred))

    correct_results_sum = (y_pred_tag == y_test).sum().float()
    acc = correct_results_sum/y_test.shape[0]
    acc = torch.round(acc * 100)

    return acc
```

[21]:
```
# function to train the binary model and print results(loss & accuracy per epoch)

def train_model_binary():
    model.train()
    for e in range(1, EPOCHS+1):
        epoch_loss = 0
        epoch_acc = 0

        for x_batch, y_batch in train_loader:
            x_batch, y_batch = x_batch, y_batch
            optimizer.zero_grad()

            y_pred = model(x_batch)

            loss = criterion(y_pred, y_batch.unsqueeze(1))
            acc = binary_acc(y_pred, y_batch.unsqueeze(1))

            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()
            epoch_acc += acc.item()


        print(f'Epoch {e+0:03}: | Loss: {epoch_loss/len(train_loader):.5f} | Acc:
 ↪ {epoch_acc/len(train_loader):.3f}')
```

[22]:
```
def test_model_binary(y_test):
    model.eval()

    y_pred_list = []
```

```
    with torch.no_grad():
        for x_batch in test_loader:
            y_test_pred = model(x_batch)
            y_pred_list.append(y_test_pred)

    y_pred_list = torch.FloatTensor(y_pred_list)
    y_test = torch.FloatTensor(y_test.tolist())

    accuracy = binary_acc(y_pred_list, y_test)
    print("Accuracy:",accuracy.item())
```

8 Ternary

[22]:
```
# function to find the accuracy of the ternary model

def ternary_acc(y_pred, y_test):
    y_pred_softmax = torch.log_softmax(y_pred, dim = 1)
    y_pred_tags = torch.argmax(y_pred_softmax, dim = 1)

    correct_pred = (y_pred_tags == y_test).float()
    acc = correct_pred.sum() / len(correct_pred)

    acc = torch.round(acc * 100)

    return acc
```

[23]:
```
# function to train the ternary model and print results(loss & accuracy per␣
 ↪epoch)

def train_model_ternary():
    model.train()
    for e in range(1, EPOCHS+1):
        epoch_loss = 0
        epoch_acc = 0

        for x_batch, y_batch in train_loader:
            x_batch, y_batch = x_batch, y_batch
            optimizer.zero_grad()

            y_pred = model(x_batch)

            loss = criterion(y_pred, y_batch.type(torch.LongTensor))
            acc = ternary_acc(y_pred, y_batch.type(torch.LongTensor))

            loss.backward()
            optimizer.step()
```

```
            epoch_loss += loss.item()
            epoch_acc += acc.item()


        print(f'Epoch {e+0:03}: | Loss: {epoch_loss/len(train_loader):.5f} | Acc:
 ↪ {epoch_acc/len(train_loader):.3f}')
```

[24]:
```
def test_model_ternary(y_test):
    model.eval()

    y_pred_list = []

    with torch.no_grad():
        for x_batch in test_loader:
            y_test_pred = model(x_batch)
            y_pred_list.extend(y_test_pred.tolist())

    y_pred_list = torch.FloatTensor(y_pred_list)
    y_test = torch.FloatTensor(y_test.tolist())

    accuracy = ternary_acc(y_pred_list, y_test)
    print("Accuracy:",accuracy.item())
```

9 (b)

[24]:
```
# function to pad a list with a specific number of zeroes

def pad_or_truncate(some_list, target_len):
    return some_list[:target_len] + [0]*(target_len - len(some_list))
```

10 5 Recurrent Neural Networks

[25]:
```
# Use the dataframe without any data cleaning and subtract target class values␣
 ↪by 1 so that it becomes easier later on while comparison

df_uncleaned['class'] = df_uncleaned['class']-1
df_uncleaned
```

[25]:

	review_body	class
0	Learning to use any new piece of technology ca...	2
1	Not exactly what expected, a little hard to as...	2
2	I followed all of the hints and the egg still ...	1
3	This electric kettle does what it's supposed t...	0
4	I kept hearing about the benefits of this kett...	0

```
...                                                      ...    ...
249995  Got for the ice tray. Not very good, turned brown      1
249996  These look lovely on my open shelves.  They ar...      0
249997  Broke down in less than two years. Not a cheap...      1
249998  Product was given as a gift but found out late...      1
249999  Perfect size and suitable to fry or make soup ...      0

[250000 rows x 2 columns]
```

[26]:
```python
# function to concatenate vectors of first fifty words as your input feature

def concatenate_vectors_rnn(review,model_used):

    sentence_words = review.split(" ")

    sentence_vectors = []

    for i,word in enumerate(sentence_words):
        if i < 50:
            try:
                sentence_vectors.append(model_used[word])
            except:
                continue

    flattened_sentence_vector = np.array(sentence_vectors).flatten()

    if len(sentence_vectors)!=0:
        if len(flattened_sentence_vector) != 15000:
            flattened_sentence_vector =␣
 ↪pad_or_truncate(list(flattened_sentence_vector),15000)

    else:
        flattened_sentence_vector = np.zeros(15000,)

    return np.reshape(np.array(flattened_sentence_vector),(50,300))
```

[27]:
```python
# find input feature for google model

df_uncleaned['input_features_1'] = df_uncleaned['review_body'].apply(lambda x:␣
 ↪concatenate_vectors_rnn(x,wv))
df_uncleaned
```

[27]:
```
                                              review_body  class  \
0       Learning to use any new piece of technology ca...      2
1       Not exactly what expected, a little hard to as...      2
2       I followed all of the hints and the egg still ...      1
3       This electric kettle does what it's supposed t...      0
```

```
4       I kept hearing about the benefits of this kett...      0
...                                                   ...    ...
249995  Got for the ice tray. Not very good, turned brown      1
249996  These look lovely on my open shelves.  They ar...      0
249997  Broke down in less than two years. Not a cheap...      1
249998  Product was given as a gift but found out late...      1
249999  Perfect size and suitable to fry or make soup ...      0

                                         input_features_1
0       [[0.0252685546875, 0.0634765625, 0.1455078125,...
1       [[0.033447265625, 0.0019989013671875, 0.061279...
2       [[0.0791015625, -0.005035400390625, 0.11181640...
3       [[-0.2890625, 0.19921875, 0.16015625, 0.025268...
4       [[0.0791015625, -0.005035400390625, 0.11181640...
...                                                   ...
249995  [[0.10888671875, -0.1435546875, -0.10693359375...
249996  [[-0.45703125, 0.259765625, 0.279296875, -0.06...
249997  [[-0.0101318359375, -0.09228515625, -0.3476562...
249998  [[-0.040283203125, -0.2099609375, 0.068359375,...
249999  [[0.103515625, 0.01312255859375, -0.0825195312...

[250000 rows x 3 columns]
```

[28]:
```python
# find input feature for our model

df_uncleaned['input_features_2'] = df_uncleaned['review_body'].apply(lambda x:
    concatenate_vectors_rnn(x,final_model))
df_uncleaned
```

```
<ipython-input-26-f682be030278>:12: DeprecationWarning: Call to deprecated
`__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__()
instead).
  sentence_vectors.append(model_used[word])
```

[28]:
```
                                              review_body  class  \
0       Learning to use any new piece of technology ca...      2
1       Not exactly what expected, a little hard to as...      2
2       I followed all of the hints and the egg still ...      1
3       This electric kettle does what it's supposed t...      0
4       I kept hearing about the benefits of this kett...      0
...                                                   ...    ...
249995  Got for the ice tray. Not very good, turned brown      1
249996  These look lovely on my open shelves.  They ar...      0
249997  Broke down in less than two years. Not a cheap...      1
249998  Product was given as a gift but found out late...      1
249999  Perfect size and suitable to fry or make soup ...      0
```

```
                                       input_features_1  \
0       [[0.0252685546875, 0.0634765625, 0.1455078125,...
1       [[0.033447265625, 0.0019989013671875, 0.061279...
2       [[0.0791015625, -0.005035400390625, 0.11181640...
3       [[-0.2890625, 0.19921875, 0.16015625, 0.025268...
4       [[0.0791015625, -0.005035400390625, 0.11181640...
...                                                   ...
249995  [[0.10888671875, -0.1435546875, -0.10693359375...
249996  [[-0.45703125, 0.259765625, 0.279296875, -0.06...
249997  [[-0.0101318359375, -0.09228515625, -0.3476562...
249998  [[-0.040283203125, -0.2099609375, 0.068359375,...
249999  [[0.103515625, 0.01312255859375, -0.0825195312...

                                       input_features_2
0       [[0.09294697642326355, -0.10120201110839844, 0...
1       [[-0.09758312255144119, -0.2398706078529358, 0...
2       [[0.20600520074367523, -0.18501587212085724, -...
3       [[0.24736081, 0.060367156, 0.11369512, -0.1542...
4       [[0.2060052, -0.18501587, -0.0031185225, -0.02...
...                                                   ...
249995  [[0.39207589626312256, -0.2490065097808838, 0...
249996  [[0.21091242, -0.040659525, -0.061009485, 0.04...
249997  [[0.43620601296424866, -0.2176143229007721, -0...
249998  [[0.21832595765590668, 0.0003553759306669235, ...
249999  [[0.24905350804328918, -0.2616400420665741, -0...

[250000 rows x 4 columns]
```

11 (a)

12 Binary

[29]:
```
# binary classification dataframe

df_binary = df_uncleaned[((df_uncleaned['class'] == 0) | (df_uncleaned['class']␣
 ↪== 1))]
df_binary
```

[29]:
```
                                            review_body  class  \
2       I followed all of the hints and the egg still ...      1
3       This electric kettle does what it's supposed t...      0
4       I kept hearing about the benefits of this kett...      0
5       Its funny how an industry shapes itself -- \\"...      0
8                                    did not fit gave away      1
...                                                   ...    ...
```

```
249995  Got for the ice tray. Not very good, turned brown      1
249996  These look lovely on my open shelves.  They ar...      0
249997  Broke down in less than two years. Not a cheap...      1
249998  Product was given as a gift but found out late...      1
249999  Perfect size and suitable to fry or make soup ...      0

                                         input_features_1  \
2       [[0.0791015625, -0.005035400390625, 0.11181640...
3       [[-0.2890625, 0.19921875, 0.16015625, 0.025268...
4       [[0.0791015625, -0.005035400390625, 0.11181640...
5       [[-0.1826171875, 0.1357421875, 0.1728515625, 0...
8       [[0.2001953125, 0.154296875, 0.10302734375, 0...
...                                                   ...
249995  [[0.10888671875, -0.1435546875, -0.10693359375...
249996  [[-0.45703125, 0.259765625, 0.279296875, -0.06...
249997  [[-0.0101318359375, -0.09228515625, -0.3476562...
249998  [[-0.040283203125, -0.2099609375, 0.068359375,...
249999  [[0.103515625, 0.01312255859375, -0.0825195312...

                                         input_features_2
2       [[0.20600520074367523, -0.18501587212085724, -...
3       [[0.24736081, 0.060367156, 0.11369512, -0.1542...
4       [[0.2060052, -0.18501587, -0.0031185225, -0.02...
5       [[0.1838633418083191, 0.03691640868782997, 0.0...
8       [[-0.14633455872535706, -0.15258042514324188, ...
...                                                   ...
249995  [[0.39207589626312256, -0.2490065097808838, 0...
249996  [[0.21091242, -0.040659525, -0.061009485, 0.04...
249997  [[0.43620601296424866, -0.2176143229007721, -0...
249998  [[0.21832595765590668, 0.0003553759306669235, ...
249999  [[0.24905350804328918, -0.2616400420665741, -0...

[200000 rows x 4 columns]
```

[30]:
```python
# set parameters

batch_size = 20
input_size = 300  # input dimension
hidden_size = 50  # hidden layer dimension
output_size = 1   # output dimension
```

[31]:
```python
# Vanilla RNN model

class RNN(nn.Module):
    def __init__(self, batch_size, input_size, hidden_size, output_size):
        super(RNN, self).__init__()
```

```
        self.batch_size, self.input_size, self.hidden_size, self.output_size =␣
↪batch_size, input_size, hidden_size, output_size

        # RNN Layer
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True,␣
↪nonlinearity='relu')
        # Fully Connected Layer
        self.layer = nn.Linear(hidden_size, self.output_size)

    def forward(self, x):
        # Initialize hidden state with zeros
        hidden_state = torch.zeros(1,self.batch_size,hidden_size)

        # Creating RNN
        hidden_outputs, hidden_state = self.rnn(x, hidden_state)

        # Log probabilities
        out = self.layer(hidden_state)

        # Reshaped out
        out = out.view(-1, self.output_size)

        return out
```

[32]:
```
# print model

model = RNN(batch_size,input_size,hidden_size,output_size)
print(model)
```

```
RNN(
  (rnn): RNN(300, 50, batch_first=True)
  (layer): Linear(in_features=50, out_features=1, bias=True)
)
```

[33]:
```
# define loss function and optimizer

criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

13 Google model

[34]:
```
x = df_binary['input_features_1']
y = df_binary['class']

# Split the dataset into 80% training dataset and 20% testing dataset
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
↪random_state=100)
```

[35]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train.tolist()))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[36]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[37]:
```
train_model_binary()
```

```
Epoch 001: | Loss: 0.69241 | Acc: 50.574
Epoch 002: | Loss: 0.69935 | Acc: 49.968
Epoch 003: | Loss: 0.69318 | Acc: 50.159
Epoch 004: | Loss: 0.69320 | Acc: 49.983
Epoch 005: | Loss: 0.63707 | Acc: 60.428
Epoch 006: | Loss: 0.47207 | Acc: 78.104
Epoch 007: | Loss: 0.40853 | Acc: 81.868
Epoch 008: | Loss: 0.37352 | Acc: 83.752
Epoch 009: | Loss: 0.35640 | Acc: 84.647
Epoch 010: | Loss: 0.34603 | Acc: 85.248
Epoch 011: | Loss: 0.33797 | Acc: 85.771
Epoch 012: | Loss: 0.33470 | Acc: 85.973
Epoch 013: | Loss: 0.33386 | Acc: 86.058
Epoch 014: | Loss: 0.33803 | Acc: 85.594
Epoch 015: | Loss: 0.32323 | Acc: 86.489
Epoch 016: | Loss: 0.43339 | Acc: 79.429
Epoch 017: | Loss: 0.33484 | Acc: 85.734
Epoch 018: | Loss: 0.31647 | Acc: 86.812
Epoch 019: | Loss: 0.31546 | Acc: 86.944
Epoch 020: | Loss: 0.31335 | Acc: 87.121
Epoch 021: | Loss: 0.31632 | Acc: 86.948
Epoch 022: | Loss: 0.30611 | Acc: 87.309
Epoch 023: | Loss: 0.30016 | Acc: 87.589
Epoch 024: | Loss: 0.30472 | Acc: 87.677
Epoch 025: | Loss: 0.29316 | Acc: 87.895
Epoch 026: | Loss: 3.30128 | Acc: 87.959
Epoch 027: | Loss: 0.28801 | Acc: 88.248
Epoch 028: | Loss: 0.28410 | Acc: 88.356
Epoch 029: | Loss: 0.28711 | Acc: 88.152
Epoch 030: | Loss: 0.28632 | Acc: 88.358
Epoch 031: | Loss: 0.28668 | Acc: 88.213
Epoch 032: | Loss: 0.28225 | Acc: 88.494
```

```
Epoch 033: | Loss: 0.28051 | Acc: 88.539
Epoch 034: | Loss: 0.28442 | Acc: 88.319
Epoch 035: | Loss: 0.28805 | Acc: 88.640
Epoch 036: | Loss: 0.27685 | Acc: 88.669
Epoch 037: | Loss: 0.32449 | Acc: 88.692
Epoch 038: | Loss: 0.27371 | Acc: 88.847
Epoch 039: | Loss: 0.27699 | Acc: 88.731
Epoch 040: | Loss: 0.44259 | Acc: 88.753
Epoch 041: | Loss: 0.27265 | Acc: 88.826
Epoch 042: | Loss: 0.27442 | Acc: 88.877
Epoch 043: | Loss: 0.27566 | Acc: 88.879
Epoch 044: | Loss: 0.28053 | Acc: 88.939
Epoch 045: | Loss: 0.26808 | Acc: 89.104
Epoch 046: | Loss: 0.26830 | Acc: 89.087
Epoch 047: | Loss: 0.26794 | Acc: 89.133
Epoch 048: | Loss: 0.26776 | Acc: 89.196
Epoch 049: | Loss: 0.34596 | Acc: 87.409
Epoch 050: | Loss: 0.27889 | Acc: 88.651
```

[38]:
```
test_model_binary(y_test)
```

```
Accuracy: 78.0
```

14 Our model

[38]:
```
x = df_binary['input_features_2']
y = df_binary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
 ↪random_state=100)
```

[39]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train.tolist()))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[40]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[41]:
```
train_model_binary()
```

```
Epoch 001: | Loss: 1.16831 | Acc: 52.679
Epoch 002: | Loss: 0.71981 | Acc: 52.742
```

```
Epoch 003: | Loss: 0.64749 | Acc: 60.969
Epoch 004: | Loss: 0.46168 | Acc: 79.341
Epoch 005: | Loss: 90.01034 | Acc: 82.052
Epoch 006: | Loss: 0.38162 | Acc: 83.591
Epoch 007: | Loss: 0.37436 | Acc: 83.855
Epoch 008: | Loss: 0.37095 | Acc: 84.084
Epoch 009: | Loss: 0.35511 | Acc: 84.839
Epoch 010: | Loss: 0.34181 | Acc: 85.526
Epoch 011: | Loss: 0.32949 | Acc: 86.115
Epoch 012: | Loss: 0.32193 | Acc: 86.463
Epoch 013: | Loss: 0.32006 | Acc: 86.632
Epoch 014: | Loss: 0.32128 | Acc: 86.593
Epoch 015: | Loss: 0.31828 | Acc: 86.671
Epoch 016: | Loss: 0.31562 | Acc: 86.901
Epoch 017: | Loss: 0.32181 | Acc: 86.798
Epoch 018: | Loss: 0.32270 | Acc: 86.799
Epoch 019: | Loss: 6.40798 | Acc: 86.496
Epoch 020: | Loss: 1.92663 | Acc: 70.343
Epoch 021: | Loss: 0.92880 | Acc: 73.413
Epoch 022: | Loss: 0.83280 | Acc: 68.356
Epoch 023: | Loss: 0.65926 | Acc: 65.443
Epoch 024: | Loss: 0.48080 | Acc: 78.717
Epoch 025: | Loss: 0.35514 | Acc: 84.923
Epoch 026: | Loss: 0.37566 | Acc: 84.504
Epoch 027: | Loss: 0.44516 | Acc: 80.558
Epoch 028: | Loss: 0.42032 | Acc: 81.544
Epoch 029: | Loss: 0.38254 | Acc: 84.191
Epoch 030: | Loss: 0.35804 | Acc: 84.858
Epoch 031: | Loss: 4.59904 | Acc: 83.344
Epoch 032: | Loss: 894.50197 | Acc: 76.135
Epoch 033: | Loss: 0.74381 | Acc: 77.533
Epoch 034: | Loss: 0.80675 | Acc: 72.624
Epoch 035: | Loss: 0.74502 | Acc: 65.621
Epoch 036: | Loss: 0.64562 | Acc: 63.171
Epoch 037: | Loss: 0.54417 | Acc: 71.858
Epoch 038: | Loss: 0.38567 | Acc: 83.814
Epoch 039: | Loss: 0.57992 | Acc: 66.572
Epoch 040: | Loss: 0.57676 | Acc: 69.403
Epoch 041: | Loss: 0.64331 | Acc: 60.016
Epoch 042: | Loss: 0.64842 | Acc: 60.724
Epoch 043: | Loss: 0.43496 | Acc: 80.531
Epoch 044: | Loss: 0.39507 | Acc: 82.644
Epoch 045: | Loss: 0.35604 | Acc: 84.931
Epoch 046: | Loss: 0.34374 | Acc: 85.556
Epoch 047: | Loss: 0.33803 | Acc: 86.047
Epoch 048: | Loss: 0.32319 | Acc: 86.449
Epoch 049: | Loss: 0.31823 | Acc: 86.666
Epoch 050: | Loss: 0.31896 | Acc: 86.819
```

[42]:
```
test_model_binary(y_test)
```

```
Accuracy: 80.0
```

15 Ternary

[42]:
```
# ternary classification dataframe

df_ternary = df_uncleaned.copy(deep = True)
df_ternary
```

[42]:
```
                                              review_body  class  \
0       Learning to use any new piece of technology ca...      2
1       Not exactly what expected, a little hard to as...      2
2       I followed all of the hints and the egg still ...      1
3       This electric kettle does what it's supposed t...      0
4       I kept hearing about the benefits of this kett...      0
...                                                   ...    ...
249995  Got for the ice tray. Not very good, turned brown      1
249996  These look lovely on my open shelves.  They ar...      0
249997  Broke down in less than two years. Not a cheap...      1
249998  Product was given as a gift but found out late...      1
249999  Perfect size and suitable to fry or make soup ...      0

                                         input_features_1  \
0       [[0.0252685546875, 0.0634765625, 0.1455078125,...
1       [[0.033447265625, 0.0019989013671875, 0.061279...
2       [[0.0791015625, -0.005035400390625, 0.11181640...
3       [[-0.2890625, 0.19921875, 0.16015625, 0.025268...
4       [[0.0791015625, -0.005035400390625, 0.11181640...
...                                                   ...
249995  [[0.10888671875, -0.1435546875, -0.10693359375...
249996  [[-0.45703125, 0.259765625, 0.279296875, -0.06...
249997  [[-0.0101318359375, -0.09228515625, -0.3476562...
249998  [[-0.040283203125, -0.2099609375, 0.068359375,...
249999  [[0.103515625, 0.01312255859375, -0.0825195312...

                                         input_features_2
0       [[0.09294697642326355, -0.10120201110839844, 0...
1       [[-0.09758312255144119, -0.2398706078529358, 0...
2       [[0.20600520074367523, -0.18501587212085724, -...
3       [[0.24736081, 0.060367156, 0.11369512, -0.1542...
4       [[0.2060052, -0.18501587, -0.0031185225, -0.02...
...                                                   ...
249995  [[0.39207589626312256, -0.2490065097808838, 0...
249996  [[0.21091242, -0.040659525, -0.061009485, 0.04...
```

```
249997   [[0.43620601296424866, -0.2176143229007721, -0...
249998   [[0.21832595765590668, 0.0003553759306669235, ...
249999   [[0.24905350804328918, -0.2616400420665741, -0...

[250000 rows x 4 columns]
```

[43]:
```
# set parameters

batch_size = 20
input_size = 300    # input dimension
hidden_size = 50  # hidden layer dimension
output_size = 3   # output dimension
```

[44]:
```
# print model

model = RNN(batch_size,input_size,hidden_size,output_size)
print(model)
```

```
RNN(
  (rnn): RNN(300, 50, batch_first=True)
  (layer): Linear(in_features=50, out_features=3, bias=True)
)
```

[45]:
```
# define loss function and optimizer

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

16 Google model

[46]:
```
x = df_ternary['input_features_1']
y = df_ternary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,
 ↪random_state=100)
```

[47]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train.tolist()))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[48]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,
 ↪shuffle=True)
```

```
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[49]:
```
train_model_ternary()
```

```
Epoch 001: | Loss: 0.99511 | Acc: 50.870
Epoch 002: | Loss: 0.87176 | Acc: 62.533
Epoch 003: | Loss: 0.97284 | Acc: 62.941
Epoch 004: | Loss: 0.82779 | Acc: 63.697
Epoch 005: | Loss: 1.01240 | Acc: 48.918
Epoch 006: | Loss: 0.88264 | Acc: 61.607
Epoch 007: | Loss: 0.87748 | Acc: 62.410
Epoch 008: | Loss: 0.87180 | Acc: 62.118
Epoch 009: | Loss: 0.85649 | Acc: 62.981
Epoch 010: | Loss: 0.92392 | Acc: 58.730
Epoch 011: | Loss: 0.86826 | Acc: 62.160
Epoch 012: | Loss: 0.93015 | Acc: 56.507
Epoch 013: | Loss: 0.82117 | Acc: 65.431
Epoch 014: | Loss: 0.83949 | Acc: 63.980
Epoch 015: | Loss: 0.82458 | Acc: 64.406
Epoch 016: | Loss: 0.82352 | Acc: 65.181
Epoch 017: | Loss: 0.78507 | Acc: 66.546
Epoch 018: | Loss: 0.80989 | Acc: 65.432
Epoch 019: | Loss: 0.77869 | Acc: 66.772
Epoch 020: | Loss: 0.80582 | Acc: 65.859
Epoch 021: | Loss: 0.82965 | Acc: 64.469
Epoch 022: | Loss: 0.81963 | Acc: 65.105
Epoch 023: | Loss: 0.76734 | Acc: 67.459
Epoch 024: | Loss: 0.78420 | Acc: 66.919
Epoch 025: | Loss: 0.76714 | Acc: 67.436
Epoch 026: | Loss: 0.76500 | Acc: 67.575
Epoch 027: | Loss: 0.79151 | Acc: 66.268
Epoch 028: | Loss: 0.84510 | Acc: 63.587
Epoch 029: | Loss: 0.81674 | Acc: 65.089
Epoch 030: | Loss: 0.79094 | Acc: 66.409
Epoch 031: | Loss: 7.59040 | Acc: 67.368
Epoch 032: | Loss: 0.76442 | Acc: 67.428
Epoch 033: | Loss: 0.77320 | Acc: 67.057
Epoch 034: | Loss: 0.75427 | Acc: 67.974
Epoch 035: | Loss: 0.76113 | Acc: 67.731
Epoch 036: | Loss: 0.74276 | Acc: 68.258
Epoch 037: | Loss: 0.73107 | Acc: 68.710
Epoch 038: | Loss: 0.72385 | Acc: 69.011
Epoch 039: | Loss: 0.72510 | Acc: 69.022
Epoch 040: | Loss: 0.71762 | Acc: 69.234
Epoch 041: | Loss: 0.74872 | Acc: 69.442
Epoch 042: | Loss: 0.72207 | Acc: 69.400
Epoch 043: | Loss: 0.71516 | Acc: 69.504
```

```
Epoch 044: | Loss: 0.72358 | Acc: 69.434
Epoch 045: | Loss: 0.71594 | Acc: 69.537
Epoch 046: | Loss: 8346.04507 | Acc: 69.359
Epoch 047: | Loss: 0.71731 | Acc: 69.341
Epoch 048: | Loss: 0.71195 | Acc: 69.398
Epoch 049: | Loss: 0.71126 | Acc: 69.690
Epoch 050: | Loss: 0.70949 | Acc: 69.710
```

[50]:
```
test_model_ternary(y_test)
```

```
Accuracy: 66.0
```

17 Our model

[50]:
```
x = df_ternary['input_features_2']
y = df_ternary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
 ↪random_state=100)
```

[51]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train.tolist()))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[52]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[52]:
```
train_model_ternary()
```

```
Epoch 001: | Loss: 0.80116 | Acc: 65.451
Epoch 002: | Loss: 0.75433 | Acc: 68.079
Epoch 003: | Loss: 0.73258 | Acc: 68.737
Epoch 004: | Loss: 3773561203.03661 | Acc: 68.079
Epoch 005: | Loss: 0.72532 | Acc: 68.918
Epoch 006: | Loss: 0.72194 | Acc: 69.128
Epoch 007: | Loss: 0.71858 | Acc: 69.153
Epoch 008: | Loss: 0.71692 | Acc: 69.209
Epoch 009: | Loss: 0.72348 | Acc: 69.115
Epoch 010: | Loss: 0.71177 | Acc: 69.511
Epoch 011: | Loss: 0.71189 | Acc: 69.688
Epoch 012: | Loss: 16.33662 | Acc: 54.169
Epoch 013: | Loss: 3.17537 | Acc: 62.745
```

```
Epoch 014: | Loss: 0.88831 | Acc: 61.114
Epoch 015: | Loss: 1.05546 | Acc: 45.058
Epoch 016: | Loss: 1.17286 | Acc: 43.706
Epoch 017: | Loss: 1.06301 | Acc: 45.240
Epoch 018: | Loss: 0.94693 | Acc: 54.622
Epoch 019: | Loss: 1.03982 | Acc: 45.260
Epoch 020: | Loss: 0.76587 | Acc: 67.485
Epoch 021: | Loss: 0.79271 | Acc: 65.894
Epoch 022: | Loss: 1.00990 | Acc: 67.805
Epoch 023: | Loss: 0.73218 | Acc: 68.684
Epoch 024: | Loss: 0.71789 | Acc: 69.330
Epoch 025: | Loss: 0.71085 | Acc: 69.588
Epoch 026: | Loss: 0.71027 | Acc: 69.707
```

[54]:
```
test_model_ternary(y_test)
```

```
Accuracy: 69.0
```

18 Comments about this question

[3]:
```
d = {'Model': ['RNN', 'RNN', 'RNN', 'RNN'],
     'Word2Vec Model': ['Google News', 'Amazon Reviews(Our)', 'Google News',␣
 ↪'Amazon Reviews(Our)'],
     'Classification Type': ['Binary', 'Binary', 'Ternary', 'Ternary',],
     'Input Features Type': ['Concat_first_50' , 'Concat_first_50',␣
 ↪'Concat_first_50', 'Concat_first_50'],
     'Accuracy': ['0.78', '0.80', '0.66', '0.69']}

df_results_part_5_a = pd.DataFrame(data=d)
df_results_part_5_a
```

[3]:

	Model	Word2Vec Model	Classification Type	Input Features Type	Accuracy
0	RNN	Google News	Binary	Concat_first_50	0.78
1	RNN	Amazon Reviews(Our)	Binary	Concat_first_50	0.80
2	RNN	Google News	Ternary	Concat_first_50	0.66
3	RNN	Amazon Reviews(Our)	Ternary	Concat_first_50	0.69

19 (b)

20 Binary

[55]:
```
# set parameters

batch_size = 20
input_size = 300     # input dimension
```

```
hidden_size = 50  # hidden layer dimension
output_size = 1   # output dimension
```

[56]:
```
# GRU model

class GRU(nn.Module):
    def __init__(self, batch_size, input_size, hidden_size, output_size):
        super(GRU, self).__init__()
        self.batch_size, self.input_size, self.hidden_size, self.output_size =␣
 ↪batch_size, input_size, hidden_size, output_size

        # RNN Layer
        self.gru = nn.GRU(input_size, hidden_size, batch_first=True)
        # Fully Connected Layer
        self.layer = nn.Linear(hidden_size, self.output_size)

    def forward(self, x):

        # Initialize hidden state with zeros
        hidden_state = torch.zeros(1,self.batch_size,hidden_size)

        # Creating RNN
        hidden_outputs, hidden_state = self.gru(x, hidden_state)

        # Log probabilities
        out = self.layer(hidden_state)

        # Reshaped out
        out = out.view(-1, self.output_size)

        return out
```

[57]:
```
# print model

model = GRU(batch_size,input_size,hidden_size,output_size)
print(model)
```

[58]:
```
# define loss function and optimizer

criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

21 Google model

[59]:
```
x = df_binary['input_features_1']
y = df_binary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
↪random_state=100)
```

[60]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train.tolist()))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[61]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[62]:
```
train_model_binary()
```

[63]:
```
test_model_binary(y_test)
```

```
Accuracy: 82.0
```

22 Our model

[64]:
```
x = df_binary['input_features_2']
y = df_binary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
↪random_state=100)
```

[65]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train.tolist()))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[66]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[67]:
```
train_model_binary()
```

[68]:
```
test_model_binary(y_test)
```

```
Accuracy: 84.0
```

23 Ternary

[69]:
```
# set parameters

batch_size = 20
input_size = 300     # input dimension
hidden_size = 50   # hidden layer dimension
output_size = 3    # output dimension
```

[70]:
```
# print model

model = GRU(batch_size,input_size,hidden_size,output_size)
print(model)
```

[71]:
```
# define loss function and optimizer

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

24 Google model

[72]:
```
x = df_ternary['input_features_1']
y = df_ternary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
 ↪random_state=100)
```

[73]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train.tolist()))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[74]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

```
[75]: train_model_ternary()
```

```
[76]: test_model_ternary(y_test)
```

```
Accuracy: 58.0
```

25 Our model

```
[77]: x = df_ternary['input_features_2']
y = df_ternary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
 ↪random_state=100)
```

```
[78]: ## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train.tolist()))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

```
[79]: train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

```
[80]: train_model_ternary()
```

```
[81]: test_model_ternary(y_test)
```

```
Accuracy: 60.0
```

26 Comments about this question

```
[82]: d = {'Model': ['GRU', 'GRU', 'GRU', 'GRU'],
     'Word2Vec Model': ['Google News', 'Amazon Reviews(Our)', 'Google News',␣
 ↪'Amazon Reviews(Our)'],
     'Classification Type': ['Binary', 'Binary', 'Ternary', 'Ternary',],
     'Input Features Type': ['Concat_first_50' , 'Concat_first_50',␣
 ↪'Concat_first_50', 'Concat_first_50'],
     'Accuracy': ['0.82', '0.84', '0.58', '0.60']}

df_results_part_5_b = pd.DataFrame(data=d)
df_results_part_5_b
```

[82]:

```
  Model       Word2Vec Model Classification Type Input Features Type Accuracy
0   GRU          Google News              Binary     Concat_first_50     0.82
1   GRU  Amazon Reviews(Our)              Binary     Concat_first_50     0.84
2   GRU          Google News             Ternary     Concat_first_50     0.58
3   GRU  Amazon Reviews(Our)             Ternary     Concat_first_50     0.60
```

27 Comments

[84]:

```
df_results_part_5_a
```

[84]:

```
  Model       Word2Vec Model Classification Type Input Features Type Accuracy
0   RNN          Google News              Binary     Concat_first_50     0.78
1   RNN  Amazon Reviews(Our)              Binary     Concat_first_50     0.80
2   RNN          Google News             Ternary     Concat_first_50     0.66
3   RNN  Amazon Reviews(Our)             Ternary     Concat_first_50     0.69
```

[86]:

```
df_results_part_5_b
```

[86]:

```
  Model       Word2Vec Model Classification Type Input Features Type Accuracy
0   GRU          Google News              Binary     Concat_first_50     0.82
1   GRU  Amazon Reviews(Our)              Binary     Concat_first_50     0.84
2   GRU          Google News             Ternary     Concat_first_50     0.58
3   GRU  Amazon Reviews(Our)             Ternary     Concat_first_50     0.60
```

It can be seen from the above tables that the GRU model performs slightly better than the RNN at binary classification. This is because GRUs contained gated units that help the model to remember long-term dependencies between words and thus do a better job at predicting the sentiment. However RNN has no long-term memory and can only help in simple sequence prediction using it's short-term memory. But, for ternary classification, the GRU model performs worse than the RNN. Since there were limited resources and time, I could not run my GRU model for as many epochs as I wished to which is why I think that for ternary, the results for GRU are lower than RNN's. I am confident that if run for more epochs, the GRU will definitely do better.

28 All accuracies reported

29 Simple Models

[88]:

```
d = {'Model': ['Perceptron', 'SVM', 'Perceptron', 'SVM', 'Perceptron', 'SVM'],
     'Word2Vec Features/Other Features': ['Google News', 'Google News', 'Amazon␣
 ↪Reviews(Our)', 'Amazon Reviews(Our)', 'TF-IDF', 'TF-IDF'],
     'Accuracy': ['0.71', '0.82', '0.81', '0.85', '0.85', '0.81']}

df_results_part_3 = pd.DataFrame(data=d)
df_results_part_3
```

[88]:
```
        Model Word2Vec Features/Other Features Accuracy
0  Perceptron                      Google News     0.71
1         SVM                      Google News     0.82
2  Perceptron               Amazon Reviews(Our)     0.81
3         SVM               Amazon Reviews(Our)     0.85
4  Perceptron                           TF-IDF     0.85
5         SVM                           TF-IDF     0.81
```

30 FNN

[90]:
```
d = {'Model': ['FNN', 'FNN', 'FNN', 'FNN'],
     'Word2Vec Model': ['Google News', 'Amazon Reviews(Our)', 'Google News',␣
 ↪'Amazon Reviews(Our)'],
     'Classification Type': ['Binary', 'Binary', 'Ternary', 'Ternary',],
     'Input Features Type': ['Average' , 'Average', 'Average', 'Average'],
     'Accuracy': ['0.85', '0.87', '0.68', '0.71']}

df_results_part_4_a = pd.DataFrame(data=d)
df_results_part_4_a
```

[90]:
```
  Model       Word2Vec Model Classification Type Input Features Type Accuracy
0   FNN          Google News              Binary             Average     0.85
1   FNN  Amazon Reviews(Our)              Binary             Average     0.87
2   FNN          Google News             Ternary             Average     0.68
3   FNN  Amazon Reviews(Our)             Ternary             Average     0.71
```

[92]:
```
d = {'Model': ['FNN', 'FNN', 'FNN', 'FNN'],
     'Word2Vec Model': ['Google News', 'Amazon Reviews(Our)', 'Google News',␣
 ↪'Amazon Reviews(Our)'],
     'Classification Type': ['Binary', 'Binary', 'Ternary', 'Ternary',],
     'Input Features Type': ['Concat_first_10' , 'Concat_first_10',␣
 ↪'Concat_first_10', 'Concat_first_10'],
     'Accuracy': ['0.73', '0.75', '0.57', '0.59']}

df_results_part_4_b = pd.DataFrame(data=d)
df_results_part_4_b
```

[92]:
```
  Model       Word2Vec Model Classification Type Input Features Type Accuracy
0   FNN          Google News              Binary     Concat_first_10     0.73
1   FNN  Amazon Reviews(Our)              Binary     Concat_first_10     0.75
2   FNN          Google News             Ternary     Concat_first_10     0.57
3   FNN  Amazon Reviews(Our)             Ternary     Concat_first_10     0.59
```

31 RNN

[94]:
```
df_results_part_5_a
```

[94]:

	Model	Word2Vec Model	Classification Type	Input Features Type	Accuracy
0	RNN	Google News	Binary	Concat_first_50	0.78
1	RNN	Amazon Reviews(Our)	Binary	Concat_first_50	0.80
2	RNN	Google News	Ternary	Concat_first_50	0.66
3	RNN	Amazon Reviews(Our)	Ternary	Concat_first_50	0.69

[95]:
```
df_results_part_5_b
```

[95]:

	Model	Word2Vec Model	Classification Type	Input Features Type	Accuracy
0	GRU	Google News	Binary	Concat_first_50	0.82
1	GRU	Amazon Reviews(Our)	Binary	Concat_first_50	0.84
2	GRU	Google News	Ternary	Concat_first_50	0.58
3	GRU	Amazon Reviews(Our)	Ternary	Concat_first_50	0.60

32 Final results

[97]:
```
df_results_final_3_4_5 = pd.
 ↪concat([df_results_part_3,df_results_part_4_a,df_results_part_4_b,df_results_part_5_a,df_resu
df_results_final_3_4_5.fillna('-',inplace=True)
cols_at_end = ['Accuracy']
df_results_final_3_4_5 = df_results_final_3_4_5[[c for c in␣
 ↪df_results_final_3_4_5 if c not in cols_at_end]
        + [c for c in cols_at_end if c in df_results_final_3_4_5]]
df_results_final_3_4_5 = df_results_final_3_4_5.reset_index()
df_results_final_3_4_5.drop(['index'],axis=1,inplace=True)
df_results_final_3_4_5
```

[97]:

	Model	Word2Vec Features/Other Features	Word2Vec Model \
0	Perceptron	Google News	-
1	SVM	Google News	-
2	Perceptron	Amazon Reviews(Our)	-
3	SVM	Amazon Reviews(Our)	-
4	Perceptron	TF-IDF	-
5	SVM	TF-IDF	-
6	FNN	-	Google News
7	FNN	-	Amazon Reviews(Our)
8	FNN	-	Google News
9	FNN	-	Amazon Reviews(Our)
10	FNN	-	Google News
11	FNN	-	Amazon Reviews(Our)
12	FNN	-	Google News
13	FNN	-	Amazon Reviews(Our)

```
14          RNN                              -          Google News
15          RNN                              -  Amazon Reviews(Our)
16          RNN                              -          Google News
17          RNN                              -  Amazon Reviews(Our)
18          GRU                              -          Google News
19          GRU                              -  Amazon Reviews(Our)
20          GRU                              -          Google News
21          GRU                              -  Amazon Reviews(Our)

   Classification Type Input Features Type Accuracy
0                    -                   -     0.71
1                    -                   -     0.82
2                    -                   -     0.81
3                    -                   -     0.85
4                    -                   -     0.85
5                    -                   -     0.81
6               Binary             Average     0.85
7               Binary             Average     0.87
8              Ternary             Average     0.68
9              Ternary             Average     0.71
10              Binary     Concat_first_10     0.73
11              Binary     Concat_first_10     0.75
12             Ternary     Concat_first_10     0.57
13             Ternary     Concat_first_10     0.59
14              Binary     Concat_first_50     0.78
15              Binary     Concat_first_50     0.80
16             Ternary     Concat_first_50     0.66
17             Ternary     Concat_first_50     0.69
18              Binary     Concat_first_50     0.82
19              Binary     Concat_first_50     0.84
20             Ternary     Concat_first_50     0.58
21             Ternary     Concat_first_50     0.60
```

Approximate order of accuracy for binary classfication - FNN Average > GRU > SVM > Perceptron > RNN > FNN Concat

Approximate order of accuracy for ternary classfication - FNN Average > RNN > GRU > FNN Concat

This shows that it's not always the most complex models that work the best. We have to try all possible models, tweak them and then check which one works best for our given data.

```
[ ]:
```