HW2-CSCI544

October 5, 2021

Name: Nidhi Rajkumar Saini
USC ID: 3730422872

0.1 Import required libraries

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.utils import resample
import nltk
import re
from bs4 import BeautifulSoup

import contractions
from nltk.corpus import stopwords
nltk.download('stopwords', quiet = True)
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics.pairwise import cosine_similarity

from sklearn.linear_model import Perceptron
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

import tensorflow as tf
import random

random.seed(42)

import warnings
```

```
warnings.filterwarnings('ignore')
```

[ ]:
```
#! pip install bs4 # in case you don't have it installed

# Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/
 ↪amazon_reviews_us_Kitchen_v1_00.tsv.gz
```

1 Dataset Generation

- Used pandas read_csv method to read the Amazon reviews dataset .gz file into a pandas dataframe. The compression parameter for this method is set to infer by default, which can automatically infer the kind of files i.e gzip , zip , bz2 , xz from the file extension.
- error_bad_lines is used to drop lines with too many fields (e.g. a csv line with too many commas).
- warn_bad_lines is used to supress the logs showing skipped lines.
- pandas.DataFrame.head method returns the first n rows of the dataframe.

[ ]:
```
df = pd.read_csv('https://s3.amazonaws.com/amazon-reviews-pds/tsv/
 ↪amazon_reviews_us_Kitchen_v1_00.tsv.gz', sep="\t", error_bad_lines = False,␣
 ↪warn_bad_lines = False)
df.head(3)
```

[ ]:
```
  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

                                     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

   star_rating  helpful_votes  total_votes vine verified_purchase  \
0          5.0            0.0          0.0    N                 Y
1          5.0            0.0          1.0    N                 Y
2          5.0            0.0          0.0    N                 Y

                     review_headline  \
0  Beautiful. Looks great on counter
1                 Awesome & Self-ness
2     Fabulous and worth every penny

                                         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
```

- Selected only the required Reviews and Ratings fields from the input dataframe and renamed them as 'reviews' and 'ratings' respectively.
- Used pandas.DataFrame.sample method to return a random sample of n rows of the dataframe.

[ ]:
```
selected_df = df[['star_rating', 'review_body']].rename(columns={'star_rating':␣
    ↪'ratings', 'review_body':'reviews'})
selected_df.sample(n = 3)
```

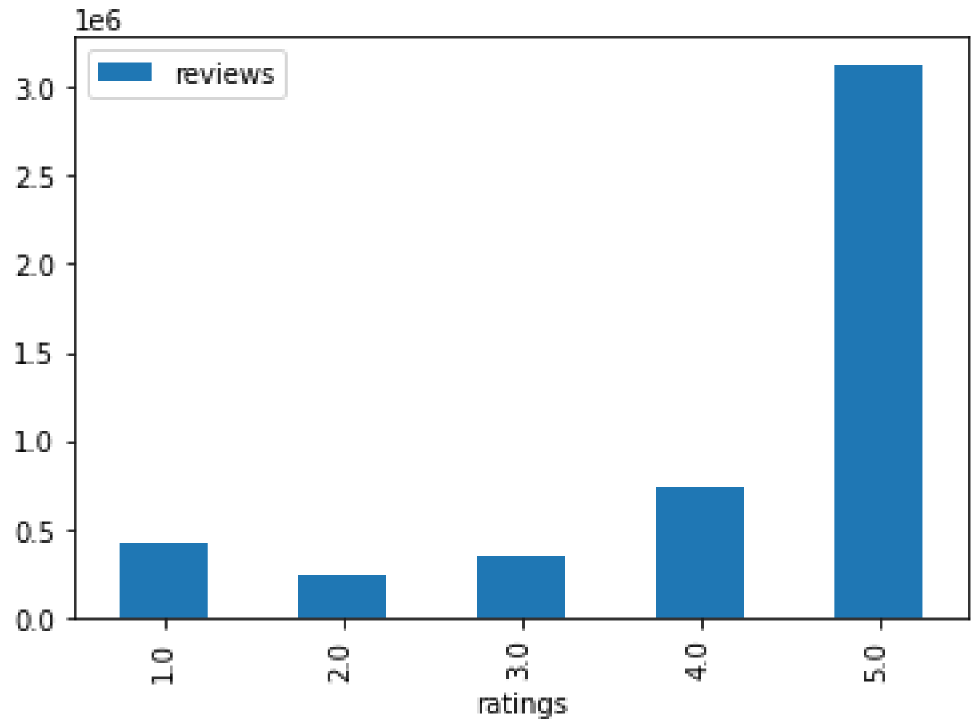
[ ]:
```
         ratings                                            reviews
1262296      1.0  Don't do it! there are other options that don'...
3383550      5.0  This bag is so spacious that I easily fits all...
4715509      2.0  It makes a big mess in the kitchen, spitting w...
```

- Calculated statistics of ratings by taking an aggregated count after grouping the dataframe by column 'ratings'.
- Plotted a bar chart to visualize the results.

[ ]:
```
# Reporting the statistics of the ratings, i.e., how many reviews received 1␣
    ↪ratings, etc.
stats_df = selected_df.groupby(selected_df['ratings']).agg('count')
print(stats_df)
stats_df.plot.bar(y='reviews')
```

```
         reviews
ratings
1.0       426870
2.0       241939
3.0       349539
4.0       731701
5.0      3124595
```

[ ]:
```
<AxesSubplot:xlabel='ratings'>
```



1.1 Build a balanced dataset

- Used sklearn.utils.resample method to keep only 50K instances of each ratings from 1-5.

[ ]:
```python
# Build a balanced dataset of 250K reviews along with their ratings (50K␣
↪instances per each rating score) through random selection.
balanced_df = []
for i in range(1,6):
    updated_df = resample(selected_df[selected_df['ratings'] == i],
                          replace = True,
                          n_samples = 50000,
                          random_state = 42)
    balanced_df.append(updated_df)
resampled_df = pd.concat(balanced_df, ignore_index = True)
resampled_df
```

[ ]:
```
        ratings                                            reviews
0           1.0  I had this for about a year and it stopped wor...
1           1.0  Cant give it any stars... just got mine yeste...
2           1.0  I just got this cup in the mail for my dad's 5...
3           1.0  I got this as a wedding gift. We used it once ...
4           1.0  We purchased this item less than two months ag...
...         ...                                                ...
249995      5.0  Awesome aerator.  It works fabulously.  The pr...
```

```
249996      5.0  Excellent, versatile strainer. Sits on top of ...
249997      5.0                                              fits
249998      5.0  I usually use Ceramic kitchen knives, but this...
249999      5.0  Have used them for years. Good way to tie chic...

[250000 rows x 2 columns]
```

- Used pandas.DataFrame.value_counts() to verify the count of reviews for each rating.

[ ]:
```
resampled_df.ratings.value_counts()
```

[ ]:
```
1.0    50000
2.0    50000
3.0    50000
4.0    50000
5.0    50000
Name: ratings, dtype: int64
```

1.2 Labelling Reviews:

Create ternary labels using the ratings.

- Created function label_review to return labels according to the ratings as asked by the problem statement.
- Ratings 1 & 2 = Label 0, Ratings 4 & 5 = Label 1, Rating 3 = Label 2
- Used pandas.DataFrame.apply method to apply label_review function to each row of the 'ratings' column and get labels.
- Used pandas.DataFrame.assign method to assign new column 'label' to the dataframe with values returned by the pandas.DataFrame.apply method.

[ ]:
```
def label_review(row):
    if row['ratings'] == 1 or row['ratings'] == 2:
        return 0
    elif row['ratings'] == 4 or row['ratings'] == 5:
        return 1
    else:
        return 2
```

[ ]:
```
col = resampled_df.apply(label_review, axis = 1) # get column data with an␣
 ↪index
resampled_df = resampled_df.assign(label = col.values) # assign values to␣
 ↪column 'label'
resampled_df
```

[ ]:
```
        ratings                                            reviews  label
0           1.0  I had this for about a year and it stopped wor...      0
1           1.0  Cant give it any stars... just got mine yeste...      0
2           1.0  I just got this cup in the mail for my dad's 5...      0
```

```
3           1.0  I got this as a wedding gift. We used it once ...      0
4           1.0  We purchased this item less than two months ag...      0
...         ...                                                ...    ...
249995      5.0  Awesome aerator.  It works fabulously.  The pr...      1
249996      5.0  Excellent, versatile strainer. Sits on top of ...      1
249997      5.0                                               fits      1
249998      5.0  I usually use Ceramic kitchen knives, but this...      1
249999      5.0  Have used them for years. Good way to tie chic...      1

[250000 rows x 3 columns]
```

- Selected only required two columns 'reviews' and 'label' from resampled_df.

[ ]:
```
resampled_df = resampled_df[['reviews','label']]
```

- Stored dataset after generation to reduce the computational load.

[ ]:
```
resampled_df.to_csv('resampled_review_data.csv', index = False)
```

[ ]:
```
resampled_df = pd.read_csv('resampled_review_data.csv')
```

1.3 Average length of reviews (Before Cleaning)

- Calculated the average length of reviews by using the len method to get the character length of each review and then taking the mean of the same, by using the pandas.DataFrame.mean method.
- Used pandas.DataFrame.apply method to apply above function to each row of the 'reviews' column.

[ ]:
```
# printing the average length of the reviews in terms of character length in␣
↪the dataset before cleaning
resampled_df['reviews'].apply(lambda x: len(str(x))).mean()
```

[ ]:
```
340.40554
```

1.4 3 sample reviews (Before Data Cleaning + Preprocessing)

- Used pandas.DataFrame.sample method to return a random sample of n rows of the dataframe.

[ ]:
```
# Printing three sample reviews before data cleaning + preprocessing
resampled_df.sample(n = 3)
```

[ ]:
```
                                                  reviews  label
84878   CR rated this one highly so we went with it. I...      0
249903  My grandson is a type 1 diabetic and we have t...      1
137845  I needed a larger skillet and boy, is this pan...      2
```

1.5 Data Cleaning

1.5.1 Convert all the reviews into lower case.

- Used str.lower() method to convert all reviews into lower case.
- Used pandas.DataFrame.apply method to apply above function to each row of the 'reviews' column.

[ ]:
```
resampled_df['reviews'] = resampled_df['reviews'].apply(lambda x: str(x).
 ↪lower())
# Dataframe after converting to lower case
resampled_df
```

[ ]:
```
                                                  reviews  label
0       i had this for about a year and it stopped wor...      0
1       cant give it any stars... just got mine yeste...      0
2       i just got this cup in the mail for my dad's 5...      0
3       i got this as a wedding gift. we used it once ...      0
4       we purchased this item less than two months ag...      0
...                                                   ...    ...
249995  awesome aerator.  it works fabulously.  the pr...      1
249996  excellent, versatile strainer. sits on top of ...      1
249997                                               fits      1
249998  i usually use ceramic kitchen knives, but this...      1
249999  have used them for years. good way to tie chic...      1

[250000 rows x 2 columns]
```

1.5.2 Remove the HTML and URLs from the reviews.

- Used BeautifulSoup to remove the HTML tags as it is a Python package that creates a parse tree for parsed pages that can be used to extract data from HTML.
- Used regular expressions to identify and remove any URLs from the reviews.
- Used pandas.DataFrame.apply method to apply above functions to each row of the 'reviews' column.

[ ]:
```
def remove_html(string):
    soup = BeautifulSoup(string, "html.parser")
    return soup.get_text()
```

[ ]:
```
def remove_urls(string):
    result = re.sub(r'http\S+', '', string)
    return result
```

[ ]:
```
# removing the HTML and URLs from the reviews
resampled_df['reviews'] = resampled_df['reviews'].apply(lambda x:␣
 ↪remove_html(str(x)))
resampled_df['reviews'] = resampled_df['reviews'].apply(lambda x:␣
 ↪remove_urls(x))
```

1.5.3 Remove non-alphabetical characters

- Used regular expressions to identify and remove non-alphabetical characters from the reviews(except spaces so that we can differentiate between words).
- Used pandas.DataFrame.apply method to apply above function to each row of the 'reviews' column in the dataframe.

[ ]:
```
def remove_nonalphachars(string):
    result = re.sub(r'[^a-zA-Z\s]','',string)
    return result
```

[ ]:
```
# Identifying rows with non-alphabetical characters in reviews
df_check = resampled_df[resampled_df.reviews.str.contains('[^a-zA-Z\s]', regex␣
 ↪= True, na = False)]
idx_nonalpha = df_check.index.tolist()
df_check
```

[ ]:
```
                                                 reviews  label
0       i had this for about a year and it stopped wor...      0
1       cant give it any stars... just got mine yeste...      0
2       i just got this cup in the mail for my dad's 5...      0
3       i got this as a wedding gift. we used it once ...      0
4       we purchased this item less than two months ag...      0
...                                                   ...    ...
249994  trays were awesome!  the molds were detailed a...      1
249995  awesome aerator.  it works fabulously.  the pr...      1
249996  excellent, versatile strainer. sits on top of ...      1
249998  i usually use ceramic kitchen knives, but this...      1
249999  have used them for years. good way to tie chic...      1

[236953 rows x 2 columns]
```

[ ]:
```
# Saving indices of reviews having contractions to test later
df_contractions = resampled_df[resampled_df.reviews.str.contains('\'', regex =␣
 ↪True, na = False)]
idx_contractions = df_check[:10].index.tolist()
```

[ ]:
```
# removing the non-alphabetical characters from the reviews
resampled_df['reviews'] = resampled_df['reviews'].apply(remove_nonalphachars)
resampled_df.loc[idx_nonalpha]
```

[ ]:
```
                                                 reviews  label
0       i had this for about a year and it stopped wor...      0
1       cant give it any stars just got mine yesterday...      0
2       i just got this cup in the mail for my dads th...      0
3       i got this as a wedding gift we used it once i...      0
4       we purchased this item less than two months ag...      0
...                                                   ...    ...
249994  trays were awesome  the molds were detailed an...      1
249995  awesome aerator  it works fabulously  the pric...      1
```

```
249996  excellent versatile strainer sits on top of mi...      1
249998  i usually use ceramic kitchen knives but this ...      1
249999  have used them for years good way to tie chick...      1

[236953 rows x 2 columns]
```

1.5.4 Remove the extra spaces between the words

- Used regular expressions to identify and remove extra spaces between the words.
- Used pandas.DataFrame.apply method to apply above function to each row of the 'reviews' column in the dataframe.

[ ]:
```python
def remove_extra_space(string):
    result = re.sub(r'\s+',' ',string)
    return result
```

[ ]:
```python
# removing extra spaces from the reviews
resampled_df['reviews'] = resampled_df['reviews'].apply(remove_extra_space)
```

1.5.5 Perform contractions on the reviews.

- Used contractions library to perform contractions on the reviews and then expand them by applying contractions.fix method to each word in the review sentence.
- Used pandas.DataFrame.apply method to apply above function to each row of the 'reviews' column in the dataframe.

[ ]:
```python
def contraction_function(review):
    contraction_str = []
    for word in review.split(' '):
        contraction_str.append(contractions.fix(word))
    return ' '.join(contraction_str)
```

[ ]:
```python
# Before removing contractions from the reviews
df_contractions[:10]
```

[ ]:
```
                                              reviews  label
1   cant give it any stars... just got mine yeste...      0
2   i just got this cup in the mail for my dad's 5...      0
3   i got this as a wedding gift. we used it once ...      0
4   we purchased this item less than two months ag...      0
6   bought from another site, and only used a hand...      0
7   first time i tried blending my bpc it spewed a...      0
10   it's bull**** it's only 0.1g not 0.01g don't buy      0
11  i tried to grind up duck.  what meat did come ...      0
12  update: after two years and maybe 4 uses, it b...      0
14  too difficult to clean the cap interior.  i've...      0
```

[ ]:
```python
# After removing contractions from the reviews
resampled_df['reviews'] = resampled_df['reviews'].apply(contraction_function)
```

```
resampled_df.loc[idx_contractions]
```

[ ]:
```
                                              reviews  label
0  i had this for about a year and it stopped wor...      0
1  cannot give it any stars just got mine yesterd...      0
2  i just got this cup in the mail for my dads th...      0
3  i got this as a wedding gift we used it once i...      0
4  we purchased this item less than two months ag...      0
5  arrived broken very badly made what i should h...      0
6  bought from another site and only used a handf...      0
7  first time i tried blending my bpc it spewed a...      0
8  not worth the effort if you are going to buy a...      0
9  the meat cutting shipped to me is not from xtr...      0
```

1.5.6 Average length of reviews (After Cleaning)

- Calculated the average length of reviews by using the len method to get the character length of each review and then taking the mean of the same, by using the pandas.DataFrame.mean method.
- Used pandas.DataFrame.apply method to apply above function to each row of the 'reviews' column.

[ ]:
```
# print the average length of the reviews in terms of character length in your␣
↪dataset after cleaning
resampled_df['reviews'].apply(lambda x: len(str(x))).mean()
```

[ ]:
```
324.165856
```

1.6 Pre-processing

1.6.1 Remove the stop words

- Used nltk.corpus.stopwords package to remove all stop words from the reviews.
- Used nltk.tokenize.word_tokenize to tokenize the sentence into words and then checked if each word was a stop word and removed the stop words.

[ ]:
```
stopword = set(stopwords.words('english'))
def remove_stop_words(string):
    text_tokens = word_tokenize(string)
    tokens_without_sw = [word for word in text_tokens if not word in stopword]
    filtered_sentence = (" ").join(tokens_without_sw)
    return filtered_sentence
```

[ ]:
```
# Identifying stop words
resampled_df.head(3)
```

[ ]:
```
                                              reviews  label
0  i had this for about a year and it stopped wor...      0
1  cannot give it any stars just got mine yesterd...      0
2  i just got this cup in the mail for my dads th...      0
```

[ ]:
```
# Removing stop words from the reviews
resampled_df['reviews'] = resampled_df['reviews'].apply(remove_stop_words)
resampled_df.head(3)
```

[ ]:
```
                                             reviews  label
0  year stopped working gentle use fresh batterie...      0
1  give stars got mine yesterday go plug tried di...      0
2  got cup mail dads th birthday next month defin...      0
```

1.6.2 3 sample reviews (After Data Cleaning + Preprocessing)

[ ]:
```
# Printing three sample reviews after data cleaning + preprocessing
resampled_df.sample(n = 3)
```

[ ]:
```
                                  reviews  label
164556   saves money buy pods constantly      1
64992       stains always looks dirty fan      0
217706  easy works great easy clean love      1
```

1.7 Splitting the dataset into Train and Test

- Used sklearn.model_selection.train_test_split to split the dataset into training and testing dataset.

[ ]:
```
# Split your dataset into 80% training dataset and 20% testing dataset.
train_df,test_df = train_test_split(resampled_df, train_size = 0.8,␣
 ↪random_state = 42)
```

2 Word Embedding

2.1 TF-IDF Feature Extraction

- Used sklearn.feature_extraction.text.TfidfVectorizer to extract TF-IDF features from the dataframe.

[ ]:
```
vectorizer = TfidfVectorizer()
vectorizer.fit(train_df['reviews'].tolist())
```

[ ]:
```
TfidfVectorizer()
```

2.2 Word2Vec

[ ]:
```
# Importing required libraries to load word2vec models

from gensim.models import Word2Vec
from gensim import models
import gensim.downloader as api
```

2.2.1 Load the pretrained "word2vec-google-news-300" Word2Vec model.

[ ]:
```
# google_model = models.KeyedVectors.load_word2vec_format('data/
↪GoogleNews-vectors-negative300.bin', binary=True)
google_model = api.load('word2vec-google-news-300')
```

[ ]:
```
google_model
```

[ ]:
```
<gensim.models.keyedvectors.KeyedVectors at 0x20600393488>
```

Check semantic similarities of the generated vectors using two examples of your own, e.g., King Man + Woman = Queen or excellent outstanding.

- Used sklearn.metrics.pairwise.cosine_similarity metric to check semantic similarities between two vectors. Cosine similarity returns the cosine of the angle between the two vectors. Therefore, the higher the cosine similarity value, the lower the angle between the vectors and thus, the closer the vectors to each other.
- Ideally, any two semantically same vectors should have cosine similarity score of 1.

[ ]:
```
a = google_model['husband'] - google_model['man'] + google_model['woman']
b = google_model['wife']
cosine_similarity(a.reshape(1,-1),b.reshape(1,-1))
```

[ ]:
```
array([[0.73326993]], dtype=float32)
```

[ ]:
```
a = google_model['tiny']
b = google_model['small']
cosine_similarity(a.reshape(1,-1),b.reshape(1,-1))
```

[ ]:
```
array([[0.7187928]], dtype=float32)
```

2.2.2 Train a Word2Vec model using your own dataset. Set the embedding size to be 300 and the window size to be 11. You can also consider a minimum word count of 10.

- I split each review into its constituent words using the split function in python.
- Then, I trained a Word2Vec model on the above defined data with a vector size of 300, window size of 11 and minimum word count of 10 as asked by the problem statement.
- Finally, after the model is trained, I save it in a .bin file for faster loading.

[ ]:
```
sentences = [x.split(' ') for x in resampled_df['reviews'].tolist()]

w2v_model = Word2Vec(sentences=sentences,vector_size=300,window=11,min_count=10)
w2v_model.save('data/my_w2v_model.bin')
```

[ ]:
```
w2v_model = Word2Vec.load('data/my_w2v_model.bin')
```

[ ]:
```
w2v_model
```

[ ]:
```
<gensim.models.word2vec.Word2Vec at 0x2075df2e608>
```

Check the semantic similarities for the same two examples in part (a).

- Used sklearn.metrics.pairwise.cosine_similarity metric to check semantic similarities between two vectors. Cosine similarity returns the cosine of the angle between the two vectors. Therefore, the higher the cosine similarity value, the lower the angle between the vectors and thus, the closer the vectors to each other.
- Ideally, any two semantically same vectors should have cosine similarity score of 1.

[ ]:
```
a = w2v_model.wv['husband'] - w2v_model.wv['man'] + w2v_model.wv['woman']
b = w2v_model.wv['wife']
cosine_similarity(a.reshape(1,-1),b.reshape(1,-1))
```

[ ]:
```
array([[0.7663224]], dtype=float32)
```

[ ]:
```
a = w2v_model.wv['tiny']
b = w2v_model.wv['small']
cosine_similarity(a.reshape(1,-1),b.reshape(1,-1))
```

[ ]:
```
array([[0.7121456]], dtype=float32)
```

2.2.3 What do you conclude from comparing vectors generated by yourself and the pretrained model?

- Both my word2vec and the pretrained word2vec models were able to capture the semantic similarities correctly for the above examples.
- This is evident by the high cosine similarity between vectors a and b taken in the examples.
- In conclusion, the performance of the models trained using both the above embeddings should not vary by a lot.

2.2.4 Which of the Word2Vec models seems to encode semantic similarities between words better?

- Although both the word2vec models have very similar cosine similarity scores on the above examples, my word2vec model (approx 76%) performs slightly better than the pretrained one (approx 73%) on the first example.

2.3 Function to compute metrics

- Created function that uses accuracy_score, precision_score, recall_score and f1_score methods from sklearn.metrics module to get the metrics.

[ ]:
```
def compute_scores(text, y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1score = f1_score(y_true, y_pred)
    print("{} Metrics:\nAccuracy = {}\nPrecision = {}\nRecall = {}\nF1-score = {}\n".format(text, accuracy, precision, recall, f1score))
```

3 Simple models

3.1 Perceptron

- Used sklearn.linear_model.Perceptron to train a Perceptron model on the training dataset.

```
[ ]: def train_test_perceptron(X_train, y_train, X_test, y_test):
         model = Perceptron()
         model.fit(X_train, y_train.ravel())

         # make a prediction
         ytrain_pred = model.predict(X_train)
         ytest_pred = model.predict(X_test)

         #calculate metrics
         compute_scores("Train", y_train, ytrain_pred)
         compute_scores("Test", y_test, ytest_pred)
```

3.1.1 TFIDF: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Then retrieved the TFIDF vectors for each review using the TFIDF vectorizer's transform function.
- Finally, performed training and testing of the perceptron model and reported the metrics.

```
[ ]: new_train_df = pd.concat([train_df[train_df['label'] ==␣
      ↪0],train_df[train_df['label'] == 1]])

     new_test_df = pd.concat([test_df[test_df['label'] ==␣
      ↪0],test_df[test_df['label'] == 1]])

     X_train = vectorizer.transform(new_train_df['reviews'].tolist())
     X_test = vectorizer.transform(new_test_df['reviews'].tolist())
     y_train = new_train_df['label']
     y_test = new_test_df['label']
     train_test_perceptron(X_train,y_train,X_test,y_test)
```

```
Train Metrics:
Accuracy = 0.9070584118573322
Precision = 0.8874558977782016
Recall = 0.932003104423804
F1-score = 0.9091841594314394

Test Metrics:
Accuracy = 0.8387298568711841
Precision = 0.8229762298568121
```

```
Recall = 0.8658148553246495
F1-score = 0.8438522107813446
```

3.1.2 Pretrained Google Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the Google word2vec vocabulary.
- Then retrieved the vectors for each review by averaging the Google word2vec vector for each word in the review.
- Finally, performed training and testing of the perceptron model and reported the metrics.

[ ]:
```
new_train_df = pd.concat([train_df[train_df['label'] ==␣
↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==␣
↪0],test_df[test_df['label'] == 1]])

DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.mean([google_model[word] if word in google_model else␣
↪DEFAULT_VECTOR for word in sentence.split(' ') ],axis=0) for sentence in␣
↪new_train_df['reviews'].tolist()]
X_test = [np.mean([google_model[word] if word in google_model else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')],axis=0) for sentence in␣
↪new_test_df['reviews'].tolist()]

y_train = new_train_df['label']
y_test = new_test_df['label']

train_test_perceptron(X_train,y_train,X_test,y_test)
```

```
Train Metrics:
Accuracy = 0.7398022944837411
Precision = 0.6732345880327943
Recall = 0.9302631249530581
F1-score = 0.7811489927524424

Test Metrics:
Accuracy = 0.7439695726153538
Precision = 0.678866203301477
Recall = 0.9323356865864572
F1-score = 0.7856634127948384
```

3.1.3 Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the word2vec vocabulary.
- Then retrieved the vectors for each review by averaging my word2vec vector for each word in the review.
- Finally, performed training and testing of the perceptron model and reported the metrics.

[ ]:
```
new_train_df = pd.concat([train_df[train_df['label'] ==␣
 ↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==␣
 ↪0],test_df[test_df['label'] == 1]])

DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.mean([w2v_model.wv[word] if word in w2v_model.wv else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ') ],axis=0) for sentence in␣
 ↪new_train_df['reviews'].tolist()]
X_test = [np.mean([w2v_model.wv[word] if word in w2v_model.wv else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')],axis=0) for sentence in␣
 ↪new_test_df['reviews'].tolist()]

y_train = new_train_df['label']
y_test = new_test_df['label']

train_test_perceptron(X_train,y_train,X_test,y_test)
```

```
Train Metrics:
Accuracy = 0.7818740783323752
Precision = 0.7693616001916397
Recall = 0.80407080089127
F1-score = 0.7863333659778671

Test Metrics:
Accuracy = 0.7825793213892503
Precision = 0.7717779152195633
Recall = 0.8065029332803023
F1-score = 0.7887584178153794
```

3.2 SVM

- Used sklearn.svm.LinearSVC to train a SVM model on the training dataset with max_iter set to 10000 i.e. increased the number of iterations to help the model to converge.

- Used sklearn.preprocessing.StandardScaler to standardize features by removing the mean and scaling to unit variance, with option with_mean = False as we are dealing with features of type sparse matrices.
- Used sklearn.pipeline.make_pipeline to construct a pipeline for the given estimators.

[ ]:
```
def train_test_svm(X_train,y_train,X_test,y_test):
    clf = make_pipeline(StandardScaler(with_mean=False),␣
 ↪LinearSVC(max_iter=10000))
    clf.fit(X_train, y_train.ravel())

    # make a prediction
    ytrain_pred = clf.predict(X_train)
    ytest_pred = clf.predict(X_test)

    #calculate metrics
    compute_scores("Train", y_train, ytrain_pred)
    compute_scores("Test", y_test, ytest_pred)
```

3.2.1 TFIDF: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Then retrieved the TFIDF vectors for each review using the TFIDF vectorizer's transform function.
- Finally, performed training and testing of the SVM model and reported the metrics.

[ ]:
```
#TFIDF: Binary classification
new_train_df = pd.concat([train_df[train_df['label'] ==␣
 ↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==␣
 ↪0],test_df[test_df['label'] == 1]])

X_train = vectorizer.transform(new_train_df['reviews'].tolist())
X_test = vectorizer.transform(new_test_df['reviews'].tolist())
y_train = new_train_df['label']
y_test = new_test_df['label']
train_test_svm(X_train,y_train,X_test,y_test)
```

```
Train Metrics:
Accuracy = 0.9440563373241021
Precision = 0.9421209439160299
Recall = 0.9460481185689608
F1-score = 0.9440804472065208

Test Metrics:
Accuracy = 0.8262185967370633
```

```
Precision = 0.8295710496020822
Recall = 0.8240031818633787
F1-score = 0.8267777417504303
```

3.2.2 Pretrained Google Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the Google word2vec vocabulary.
- Then retrieved the vectors for each review by averaging the Google word2vec vector for each word in the review.
- Finally, performed training and testing of the SVM model and reported the metrics.

[ ]:
```
#Word2Vec - Google: Binary classification
new_train_df = pd.concat([train_df[train_df['label'] ==␣
 ↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==␣
 ↪0],test_df[test_df['label'] == 1]])

DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.mean([google_model[word] if word in google_model else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ') ],axis=0) for sentence in␣
 ↪new_train_df['reviews'].tolist()]
X_test = [np.mean([google_model[word] if word in google_model else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')],axis=0) for sentence in␣
 ↪new_test_df['reviews'].tolist()]

y_train = new_train_df['label']
y_test = new_test_df['label']

train_test_svm(X_train,y_train,X_test,y_test)
```

```
C:\Users\saini\anaconda3\lib\site-packages\sklearn\svm\_base.py:985:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "

Train Metrics:
Accuracy = 0.8204841410682596
Precision = 0.8345562749329671
Recall = 0.7987131662619232
F1-score = 0.8162414209964118
```

```
Test Metrics:
Accuracy = 0.8209388449604644
Precision = 0.8360128617363344
Recall = 0.8014318385204335
F1-score = 0.8183571936237181
```

3.2.3 Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the word2vec vocabulary.
- Then retrieved the vectors for each review by averaging my word2vec vector for each word in the review.
- Finally, performed training and testing of the SVM model and reported the metrics.

[ ]:
```
#Word2Vec: Binary classification
new_train_df = pd.concat([train_df[train_df['label'] ==␣
 ↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==␣
 ↪0],test_df[test_df['label'] == 1]])

DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.mean([w2v_model.wv[word] if word in w2v_model.wv else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ') ],axis=0) for sentence in␣
 ↪new_train_df['reviews'].tolist()]
X_test = [np.mean([w2v_model.wv[word] if word in w2v_model.wv else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')],axis=0) for sentence in␣
 ↪new_test_df['reviews'].tolist()]

y_train = new_train_df['label']
y_test = new_test_df['label']

train_test_svm(X_train,y_train,X_test,y_test)
```

```
C:\Users\saini\anaconda3\lib\site-packages\sklearn\svm\_base.py:985:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "

Train Metrics:
Accuracy = 0.8506523532205253
Precision = 0.8582856557744244
Recall = 0.8394086573367048
```

```
F1-score = 0.8487422080182261

Test Metrics:
Accuracy = 0.8518416574917426
Precision = 0.8608695652173913
Recall = 0.8416525802923337
F1-score = 0.8511526182156415
```

3.3 What do you conclude from comparing performances for the models trained using the three different feature types (TF-IDF, pretrained Word2Vec, your trained Word2Vec)?

For Perceptron - The TFIDF had test accuracy of approx 84%, google word2vec of approx 74% and my Word2Vec of approx 78%. - Therefore the TFIDF features had the best performance for perceptron model.

For SVM - The TFIDF had test accuracy of 82.6%, google word2vec of approx 82% and my Word2Vec of approx 85%. - Therefore my Word2Vec features had the best performance for SVM model.

Overall, SVM Model performed better than Perceptron model for the given data.

4 FFNN

4.1 To generate the input features, use the average Word2Vec vectors similar to the "Simple models" section and train the neural network. Train a network for binary classification using class 1 and class 2 and also a ternary model for the three classes. Report accuracy values on the testing split for your MLP model for each of the binary and ternary classification cases.

- Created function to load the saved model checkpoint, running it on Test data and reporting the loss and accuracy.

[ ]:
```
def load_and_test_model_weights(model,model_name,X_test,y_test):
    model.load_weights('models/{model_name}/variables/variables'.
 ↪format(model_name=model_name))
    loss,acc = model.evaluate(np.array(X_test),y_test)
    print("Loss = {loss} Accuracy = {acc}".format(loss=loss,acc=acc))
```

4.1.1 Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the word2vec vocabulary.
- Then retrieved the vectors for each review by averaging my word2vec vector for each word in the review.

[ ]:
```
new_train_df = pd.concat([train_df[train_df['label'] ==␣
↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==␣
↪0],test_df[test_df['label'] == 1]])

DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.mean([w2v_model.wv[word] if word in w2v_model.wv else␣
↪DEFAULT_VECTOR for word in sentence.split(' ') ],axis=0) for sentence in␣
↪new_train_df['reviews'].tolist()]
X_test = [np.mean([w2v_model.wv[word] if word in w2v_model.wv else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')],axis=0) for sentence in␣
↪new_test_df['reviews'].tolist()]

y_train = new_train_df['label']
y_test = new_test_df['label']
```

- Below, I described the FFNN model consisting of two hidden layers with ReLU activation with 50 and 10 neurons respectively for the binary classification task.
- I used the tanh non-linearity in the output layer with 1 neuron which gives me an output in the range [-1,1], therefore for outputs <0, we predict class 0 (negative) and for outputs >0, we predict class 1 (positive).
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_1 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((300,)),
    tf.keras.layers.Dense(50,activation='relu'),
    tf.keras.layers.Dense(10,activation='relu'),
    tf.keras.layers.Dense(1,activation='tanh')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_1', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
```

```
)

model_1.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='binary_crossentropy',metrics=['accuracy'])
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 50)                15050
_________________________________________________________________
dense_1 (Dense)              (None, 10)                510
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 11
=================================================================
Total params: 15,571
Trainable params: 15,571
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the FFNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_1.fit(np.array(X_train),y_train,validation_data=(np.
 ↪array(X_test),y_test),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
5002/5002 [==============================] - 10s 2ms/step - loss: 0.3752 -
accuracy: 0.8400 - val_loss: 0.3370 - val_accuracy: 0.8587
INFO:tensorflow:Assets written to: models\model_1\assets
Epoch 2/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.3405 -
accuracy: 0.8558 - val_loss: 0.3448 - val_accuracy: 0.8612
INFO:tensorflow:Assets written to: models\model_1\assets
Epoch 3/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.3279 -
accuracy: 0.8614 - val_loss: 0.3253 - val_accuracy: 0.8613
INFO:tensorflow:Assets written to: models\model_1\assets
Epoch 4/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.3228 -
accuracy: 0.8639 - val_loss: 0.3290 - val_accuracy: 0.8637
INFO:tensorflow:Assets written to: models\model_1\assets
Epoch 5/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.3132 -
```

```
accuracy: 0.8673 - val_loss: 0.3199 - val_accuracy: 0.8653
INFO:tensorflow:Assets written to: models\model_1\assets
Epoch 6/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.3094 -
accuracy: 0.8685 - val_loss: 0.3199 - val_accuracy: 0.8639
Epoch 7/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.3008 -
accuracy: 0.8726 - val_loss: 0.3201 - val_accuracy: 0.8682
INFO:tensorflow:Assets written to: models\model_1\assets
Epoch 8/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.3042 -
accuracy: 0.8736 - val_loss: 0.3197 - val_accuracy: 0.8685
INFO:tensorflow:Assets written to: models\model_1\assets
Epoch 9/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2946 -
accuracy: 0.8758 - val_loss: 0.3200 - val_accuracy: 0.8646
Epoch 10/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2954 -
accuracy: 0.8773 - val_loss: 0.3241 - val_accuracy: 0.8661
Epoch 11/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2921 -
accuracy: 0.8784 - val_loss: 0.3237 - val_accuracy: 0.8688
INFO:tensorflow:Assets written to: models\model_1\assets
Epoch 12/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2984 -
accuracy: 0.8779 - val_loss: 0.3159 - val_accuracy: 0.8675
Epoch 13/50
5002/5002 [==============================] - 9s 2ms/step - loss: 0.2841 -
accuracy: 0.8819 - val_loss: 0.3272 - val_accuracy: 0.8680
Epoch 14/50
5002/5002 [==============================] - 9s 2ms/step - loss: 0.2799 -
accuracy: 0.8830 - val_loss: 0.3242 - val_accuracy: 0.8675
Epoch 15/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2786 -
accuracy: 0.8841 - val_loss: 0.3144 - val_accuracy: 0.8686
Epoch 16/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2771 -
accuracy: 0.8851 - val_loss: 0.3184 - val_accuracy: 0.8679
Epoch 17/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2717 -
accuracy: 0.8868 - val_loss: 0.3210 - val_accuracy: 0.8676
Epoch 18/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2710 -
accuracy: 0.8873 - val_loss: 0.3156 - val_accuracy: 0.8700
INFO:tensorflow:Assets written to: models\model_1\assets
Epoch 19/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2665 -
accuracy: 0.8892 - val_loss: 0.3307 - val_accuracy: 0.8658
```

```
Epoch 20/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2660 -
accuracy: 0.8898 - val_loss: 0.3155 - val_accuracy: 0.8674
Epoch 21/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2656 -
accuracy: 0.8907 - val_loss: 0.3156 - val_accuracy: 0.8677
Epoch 22/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2613 -
accuracy: 0.8916 - val_loss: 0.3233 - val_accuracy: 0.8682
Epoch 23/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2608 -
accuracy: 0.8924 - val_loss: 0.3244 - val_accuracy: 0.8693
Epoch 24/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2583 -
accuracy: 0.8936 - val_loss: 0.3193 - val_accuracy: 0.8692
Epoch 25/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2581 -
accuracy: 0.8935 - val_loss: 0.3221 - val_accuracy: 0.8679
Epoch 26/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2553 -
accuracy: 0.8942 - val_loss: 0.3270 - val_accuracy: 0.8678
Epoch 27/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2544 -
accuracy: 0.8953 - val_loss: 0.3239 - val_accuracy: 0.8667
Epoch 28/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2521 -
accuracy: 0.8956 - val_loss: 0.3335 - val_accuracy: 0.8676
Epoch 29/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2509 -
accuracy: 0.8967 - val_loss: 0.3323 - val_accuracy: 0.8681
Epoch 30/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2503 -
accuracy: 0.8973 - val_loss: 0.3314 - val_accuracy: 0.8669
Epoch 31/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2485 -
accuracy: 0.8977 - val_loss: 0.3309 - val_accuracy: 0.8662
Epoch 32/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2475 -
accuracy: 0.8975 - val_loss: 0.3337 - val_accuracy: 0.8667
Epoch 33/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2476 -
accuracy: 0.8989 - val_loss: 0.3316 - val_accuracy: 0.8674
Epoch 34/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2456 -
accuracy: 0.8987 - val_loss: 0.3286 - val_accuracy: 0.8669
Epoch 35/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2447 -
accuracy: 0.8991 - val_loss: 0.3330 - val_accuracy: 0.8659
```

```
Epoch 36/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2435 -
accuracy: 0.8998 - val_loss: 0.3309 - val_accuracy: 0.8676
Epoch 37/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2428 -
accuracy: 0.9005 - val_loss: 0.3353 - val_accuracy: 0.8672
Epoch 38/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2423 -
accuracy: 0.9006 - val_loss: 0.3348 - val_accuracy: 0.8668
Epoch 39/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2411 -
accuracy: 0.9009 - val_loss: 0.3387 - val_accuracy: 0.8655
Epoch 40/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2403 -
accuracy: 0.9013 - val_loss: 0.3359 - val_accuracy: 0.8667
Epoch 41/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2427 -
accuracy: 0.9014 - val_loss: 0.3348 - val_accuracy: 0.8674
Epoch 42/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2387 -
accuracy: 0.9021 - val_loss: 0.3345 - val_accuracy: 0.8672
Epoch 43/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2385 -
accuracy: 0.9024 - val_loss: 0.3353 - val_accuracy: 0.8671
Epoch 44/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2374 -
accuracy: 0.9028 - val_loss: 0.3354 - val_accuracy: 0.8674
Epoch 45/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2368 -
accuracy: 0.9030 - val_loss: 0.3400 - val_accuracy: 0.8675
Epoch 46/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2365 -
accuracy: 0.9033 - val_loss: 0.3407 - val_accuracy: 0.8660
Epoch 47/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2359 -
accuracy: 0.9030 - val_loss: 0.3394 - val_accuracy: 0.8675
Epoch 48/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2350 -
accuracy: 0.9040 - val_loss: 0.3424 - val_accuracy: 0.8662
Epoch 49/50
5002/5002 [==============================] - 7s 1ms/step - loss: 0.2346 -
accuracy: 0.9043 - val_loss: 0.3387 - val_accuracy: 0.8667
Epoch 50/50
5002/5002 [==============================] - 8s 2ms/step - loss: 0.2341 -
accuracy: 0.9043 - val_loss: 0.3441 - val_accuracy: 0.8659
```

[ ]: <tensorflow.python.keras.callbacks.History at 0x23cc9a36940>

- Ran the load_and_test_model_weights function to load the saved best model as well as report accuracy and loss for best model.

[ ]:
```
load_and_test_model_weights(model_1,'model_1',X_test,y_test)
```

```
1249/1249 [==============================] - 2s 2ms/step - loss: 0.3156 -
accuracy: 0.8700
Loss = 0.31557077169418335 Accuracy = 0.8699829578399658
```

4.1.2 Google Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the google word2vec vocabulary.
- Then retrieved the vectors for each review by averaging the google word2vec vector for each word in the review.

[ ]:
```
new_train_df = pd.concat([train_df[train_df['label'] ==␣
↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==␣
↪0],test_df[test_df['label'] == 1]])

DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.mean([google_model[word] if word in google_model else␣
↪DEFAULT_VECTOR for word in sentence.split(' ') ],axis=0) for sentence in␣
↪new_train_df['reviews'].tolist()]
X_test = [np.mean([google_model[word] if word in google_model else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')],axis=0) for sentence in␣
↪new_test_df['reviews'].tolist()]

y_train = new_train_df['label']
y_test = new_test_df['label']
```

- Below, I described the FFNN model consisting of two hidden layers with ReLU activation with 50 and 10 neurons respectively for the binary classification task.
- I used the tanh non-linearity in the output layer with 1 neuron which gives me an output in the range [-1,1], therefore for outputs <0, we predict class 0 (negative) and for outputs >0, we predict class 1 (positive).
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_2 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((300,)),
    tf.keras.layers.Dense(50,activation='relu'),
    tf.keras.layers.Dense(10,activation='relu'),
    tf.keras.layers.Dense(1,activation='tanh')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_2', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_2.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='binary_crossentropy',metrics=['accuracy'])
model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 50)                15050
_________________________________________________________________
dense_4 (Dense)              (None, 10)                510
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 11
=================================================================
Total params: 15,571
Trainable params: 15,571
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the FFNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_2.fit(np.array(X_train),y_train,
            validation_data=(np.array(X_test),y_test),
```

```
            batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
5002/5002 [==============================] - 15s 3ms/step - loss: 0.4424 -
accuracy: 0.8038 - val_loss: 0.3985 - val_accuracy: 0.8233
INFO:tensorflow:Assets written to: models\model_2\assets
Epoch 2/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3941 -
accuracy: 0.8258 - val_loss: 0.3799 - val_accuracy: 0.8319
INFO:tensorflow:Assets written to: models\model_2\assets
Epoch 3/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.3812 -
accuracy: 0.8327 - val_loss: 0.3776 - val_accuracy: 0.8336
INFO:tensorflow:Assets written to: models\model_2\assets
Epoch 4/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3708 -
accuracy: 0.8381 - val_loss: 0.3853 - val_accuracy: 0.8309
Epoch 5/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3663 -
accuracy: 0.8407 - val_loss: 0.3793 - val_accuracy: 0.8313
Epoch 6/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3597 -
accuracy: 0.8441 - val_loss: 0.3797 - val_accuracy: 0.8392
INFO:tensorflow:Assets written to: models\model_2\assets
Epoch 7/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3550 -
accuracy: 0.8458 - val_loss: 0.3658 - val_accuracy: 0.8406
INFO:tensorflow:Assets written to: models\model_2\assets
Epoch 8/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3518 -
accuracy: 0.8478 - val_loss: 0.3767 - val_accuracy: 0.8374
Epoch 9/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3472 -
accuracy: 0.8508 - val_loss: 0.3660 - val_accuracy: 0.8449
INFO:tensorflow:Assets written to: models\model_2\assets
Epoch 10/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3445 -
accuracy: 0.8521 - val_loss: 0.3602 - val_accuracy: 0.8440
Epoch 11/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.3384 -
accuracy: 0.8539 - val_loss: 0.3704 - val_accuracy: 0.8427
Epoch 12/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3348 -
accuracy: 0.8554 - val_loss: 0.3682 - val_accuracy: 0.8424
Epoch 13/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3362 -
accuracy: 0.8564 - val_loss: 0.3629 - val_accuracy: 0.8443
```

```
Epoch 14/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3300 -
accuracy: 0.8584 - val_loss: 0.3586 - val_accuracy: 0.8468
INFO:tensorflow:Assets written to: models\model_2\assets
Epoch 15/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3290 -
accuracy: 0.8594 - val_loss: 0.3700 - val_accuracy: 0.8430
Epoch 16/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3300 -
accuracy: 0.8600 - val_loss: 0.3898 - val_accuracy: 0.8318
Epoch 17/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3265 -
accuracy: 0.8600 - val_loss: 0.3645 - val_accuracy: 0.8444
Epoch 18/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3208 -
accuracy: 0.8611 - val_loss: 0.3661 - val_accuracy: 0.8461
Epoch 19/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3169 -
accuracy: 0.8634 - val_loss: 0.3709 - val_accuracy: 0.8443
Epoch 20/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3200 -
accuracy: 0.8641 - val_loss: 0.3728 - val_accuracy: 0.8468
Epoch 21/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3171 -
accuracy: 0.8653 - val_loss: 0.3811 - val_accuracy: 0.8442
Epoch 22/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3163 -
accuracy: 0.8651 - val_loss: 0.3792 - val_accuracy: 0.8465
Epoch 23/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3135 -
accuracy: 0.8668 - val_loss: 0.3885 - val_accuracy: 0.8427
Epoch 24/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3124 -
accuracy: 0.8673 - val_loss: 0.3980 - val_accuracy: 0.8467
Epoch 25/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3084 -
accuracy: 0.8682 - val_loss: 0.3888 - val_accuracy: 0.8458
Epoch 26/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3076 -
accuracy: 0.8691 - val_loss: 0.3946 - val_accuracy: 0.8437
Epoch 27/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3088 -
accuracy: 0.8695 - val_loss: 0.3782 - val_accuracy: 0.8388
Epoch 28/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3050 -
accuracy: 0.8716 - val_loss: 0.3965 - val_accuracy: 0.8461
Epoch 29/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3041 -
```

```
accuracy: 0.8716 - val_loss: 0.3984 - val_accuracy: 0.8425
Epoch 30/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3045 -
accuracy: 0.8723 - val_loss: 0.3876 - val_accuracy: 0.8366
Epoch 31/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3035 -
accuracy: 0.8716 - val_loss: 0.4040 - val_accuracy: 0.8440
Epoch 32/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2998 -
accuracy: 0.8738 - val_loss: 0.3919 - val_accuracy: 0.8437
Epoch 33/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2994 -
accuracy: 0.8736 - val_loss: 0.3931 - val_accuracy: 0.8415
Epoch 34/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3011 -
accuracy: 0.8739 - val_loss: 0.4132 - val_accuracy: 0.8251
Epoch 35/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3080 -
accuracy: 0.8703 - val_loss: 0.3836 - val_accuracy: 0.8428
Epoch 36/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2919 -
accuracy: 0.8745 - val_loss: 0.3932 - val_accuracy: 0.8444
Epoch 37/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2927 -
accuracy: 0.8756 - val_loss: 0.3987 - val_accuracy: 0.8453
Epoch 38/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2911 -
accuracy: 0.8761 - val_loss: 0.3957 - val_accuracy: 0.8462
Epoch 39/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2894 -
accuracy: 0.8765 - val_loss: 0.3969 - val_accuracy: 0.8448
Epoch 40/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2884 -
accuracy: 0.8771 - val_loss: 0.3991 - val_accuracy: 0.8446
Epoch 41/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2908 -
accuracy: 0.8768 - val_loss: 0.3945 - val_accuracy: 0.8429
Epoch 42/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2894 -
accuracy: 0.8779 - val_loss: 0.3978 - val_accuracy: 0.8439
Epoch 43/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2887 -
accuracy: 0.8783 - val_loss: 0.4034 - val_accuracy: 0.8454
Epoch 44/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2878 -
accuracy: 0.8785 - val_loss: 0.4122 - val_accuracy: 0.8455
Epoch 45/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2883 -
```

```
accuracy: 0.8789 - val_loss: 0.4136 - val_accuracy: 0.8452
Epoch 46/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.2875 -
accuracy: 0.8792 - val_loss: 0.4093 - val_accuracy: 0.8443
Epoch 47/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.2857 -
accuracy: 0.8792 - val_loss: 0.4110 - val_accuracy: 0.8439
Epoch 48/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.2861 -
accuracy: 0.8798 - val_loss: 0.4234 - val_accuracy: 0.8452
Epoch 49/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.2869 -
accuracy: 0.8801 - val_loss: 0.4295 - val_accuracy: 0.8459
Epoch 50/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2850 -
accuracy: 0.8801 - val_loss: 0.4220 - val_accuracy: 0.8450
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x1c5f24c4be0>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_2,'model_2',X_test,y_test)
```

```
1249/1249 [==============================] - 2s 2ms/step - loss: 0.3586 -
accuracy: 0.8468
Loss = 0.3585948348045349 Accuracy = 0.8468371629714966
```

4.1.3 Word2Vec: Ternary classification

- For ternary classification, I created default vector using the np.zeros() function in case a word is missing from the word2vec vocabulary.
- Then retrieved the vectors for each review by averaging my word2vec vector for each word in the review.

[ ]:
```
DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.mean([w2v_model.wv[word] if word in w2v_model.wv else␣
↪DEFAULT_VECTOR for word in sentence.split(' ') ],axis=0) for sentence in␣
↪train_df['reviews'].tolist()]
X_test = [np.mean([w2v_model.wv[word] if word in w2v_model.wv else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')],axis=0) for sentence in␣
↪test_df['reviews'].tolist()]

y_train = train_df['label']
y_test = test_df['label']
```

- Below, I described the FFNN model consisting of two hidden layers with ReLU activation with 50 and 10 neurons respectively for the ternary classification task.
- I used the softmax non-linearity in the output layer with 3 neurons which gives an output probability distribution with the probability for each class, therefore the final predicted output is the class with maximum probability.
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_3 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((300,)),
    tf.keras.layers.Dense(50,activation='relu'),
    tf.keras.layers.Dense(10,activation='relu'),
    tf.keras.layers.Dense(3,activation='softmax')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_3', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_3.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='sparse_categorical_crossentropy',metrics=['accuracy']
model_3.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 50)                15050

_________________________________________________________________
dense_7 (Dense)              (None, 10)                510

_________________________________________________________________
dense_8 (Dense)              (None, 3)                 33
=================================================================
Total params: 15,593
Trainable params: 15,593
```

```
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the FFNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

```
[ ]: model_3.fit(np.array(X_train),y_train,validation_data=(np.
      ↪array(X_test),y_test),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
6250/6250 [==============================] - 16s 2ms/step - loss: 0.7374 -
accuracy: 0.6855 - val_loss: 0.7122 - val_accuracy: 0.6945
INFO:tensorflow:Assets written to: models\model_3\assets
Epoch 2/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.7083 -
accuracy: 0.6964 - val_loss: 0.7095 - val_accuracy: 0.6979
INFO:tensorflow:Assets written to: models\model_3\assets
Epoch 3/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6970 -
accuracy: 0.7022 - val_loss: 0.7050 - val_accuracy: 0.6921
Epoch 4/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6881 -
accuracy: 0.7058 - val_loss: 0.6993 - val_accuracy: 0.7010
INFO:tensorflow:Assets written to: models\model_3\assets
Epoch 5/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6807 -
accuracy: 0.7085 - val_loss: 0.6972 - val_accuracy: 0.7006
Epoch 6/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6756 -
accuracy: 0.7096 - val_loss: 0.6920 - val_accuracy: 0.7025
INFO:tensorflow:Assets written to: models\model_3\assets
Epoch 7/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6710 -
accuracy: 0.7122 - val_loss: 0.6894 - val_accuracy: 0.7063
INFO:tensorflow:Assets written to: models\model_3\assets
Epoch 8/50
6250/6250 [==============================] - 16s 2ms/step - loss: 0.6661 -
accuracy: 0.7137 - val_loss: 0.6942 - val_accuracy: 0.7017
Epoch 9/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6618 -
accuracy: 0.7162 - val_loss: 0.6957 - val_accuracy: 0.7050
Epoch 10/50
6250/6250 [==============================] - 16s 2ms/step - loss: 0.6590 -
accuracy: 0.7166 - val_loss: 0.6909 - val_accuracy: 0.7037
Epoch 11/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6555 -
```

```
accuracy: 0.7182 - val_loss: 0.7053 - val_accuracy: 0.7013
Epoch 12/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6529 -
accuracy: 0.7190 - val_loss: 0.7031 - val_accuracy: 0.7022
Epoch 13/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6501 -
accuracy: 0.7208 - val_loss: 0.6929 - val_accuracy: 0.7038
Epoch 14/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6474 -
accuracy: 0.7216 - val_loss: 0.7000 - val_accuracy: 0.7048
Epoch 15/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6449 -
accuracy: 0.7225 - val_loss: 0.7111 - val_accuracy: 0.6984
Epoch 16/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6426 -
accuracy: 0.7228 - val_loss: 0.6944 - val_accuracy: 0.7062
Epoch 17/50
6250/6250 [==============================] - 20s 3ms/step - loss: 0.6404 -
accuracy: 0.7243 - val_loss: 0.6952 - val_accuracy: 0.7054
Epoch 18/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6384 -
accuracy: 0.7248 - val_loss: 0.6944 - val_accuracy: 0.7055
Epoch 19/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6367 -
accuracy: 0.7253 - val_loss: 0.6943 - val_accuracy: 0.7054
Epoch 20/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6348 -
accuracy: 0.7266 - val_loss: 0.6944 - val_accuracy: 0.7061
Epoch 21/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6337 -
accuracy: 0.7273 - val_loss: 0.6932 - val_accuracy: 0.7054
Epoch 22/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6319 -
accuracy: 0.7287 - val_loss: 0.6990 - val_accuracy: 0.7046
Epoch 23/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6301 -
accuracy: 0.7292 - val_loss: 0.6951 - val_accuracy: 0.7051
Epoch 24/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6285 -
accuracy: 0.7299 - val_loss: 0.6978 - val_accuracy: 0.7043
Epoch 25/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6276 -
accuracy: 0.7305 - val_loss: 0.6945 - val_accuracy: 0.7045
Epoch 26/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6262 -
accuracy: 0.7308 - val_loss: 0.6986 - val_accuracy: 0.7043
Epoch 27/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6248 -
```

```
accuracy: 0.7309 - val_loss: 0.7000 - val_accuracy: 0.7020
Epoch 28/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6239 -
accuracy: 0.7308 - val_loss: 0.7054 - val_accuracy: 0.7039
Epoch 29/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6227 -
accuracy: 0.7319 - val_loss: 0.7018 - val_accuracy: 0.7056
Epoch 30/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6214 -
accuracy: 0.7324 - val_loss: 0.7004 - val_accuracy: 0.7048
Epoch 31/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6207 -
accuracy: 0.7328 - val_loss: 0.6990 - val_accuracy: 0.7038
Epoch 32/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6197 -
accuracy: 0.7328 - val_loss: 0.7030 - val_accuracy: 0.7042
Epoch 33/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6188 -
accuracy: 0.7339 - val_loss: 0.7011 - val_accuracy: 0.7049
Epoch 34/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6180 -
accuracy: 0.7338 - val_loss: 0.7034 - val_accuracy: 0.7048
Epoch 35/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6171 -
accuracy: 0.7337 - val_loss: 0.7030 - val_accuracy: 0.7027
Epoch 36/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6160 -
accuracy: 0.7346 - val_loss: 0.7061 - val_accuracy: 0.7045
Epoch 37/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6155 -
accuracy: 0.7350 - val_loss: 0.7053 - val_accuracy: 0.7043
Epoch 38/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6148 -
accuracy: 0.7354 - val_loss: 0.7048 - val_accuracy: 0.7041
Epoch 39/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6140 -
accuracy: 0.7358 - val_loss: 0.7045 - val_accuracy: 0.7044
Epoch 40/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6135 -
accuracy: 0.7352 - val_loss: 0.7051 - val_accuracy: 0.7035
Epoch 41/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6126 -
accuracy: 0.7362 - val_loss: 0.7069 - val_accuracy: 0.7019
Epoch 42/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6122 -
accuracy: 0.7363 - val_loss: 0.7078 - val_accuracy: 0.7036
Epoch 43/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6118 -
```

```
accuracy: 0.7359 - val_loss: 0.7125 - val_accuracy: 0.7033
Epoch 44/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6110 -
accuracy: 0.7367 - val_loss: 0.7108 - val_accuracy: 0.7033
Epoch 45/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6104 -
accuracy: 0.7367 - val_loss: 0.7096 - val_accuracy: 0.7032
Epoch 46/50
6250/6250 [==============================] - 18s 3ms/step - loss: 0.6100 -
accuracy: 0.7375 - val_loss: 0.7135 - val_accuracy: 0.7038
Epoch 47/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6096 -
accuracy: 0.7370 - val_loss: 0.7099 - val_accuracy: 0.7024
Epoch 48/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6090 -
accuracy: 0.7375 - val_loss: 0.7133 - val_accuracy: 0.7026
Epoch 49/50
6250/6250 [==============================] - 16s 2ms/step - loss: 0.6087 -
accuracy: 0.7378 - val_loss: 0.7136 - val_accuracy: 0.7028
Epoch 50/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6083 -
accuracy: 0.7376 - val_loss: 0.7141 - val_accuracy: 0.7035
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x1c5f28387f0>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_3,'model_3',X_test,y_test)
```

```
1563/1563 [==============================] - 3s 2ms/step - loss: 0.6894 -
accuracy: 0.7063
Loss = 0.6894176006317139 Accuracy = 0.7062600255012512
```

4.1.4 Google Word2Vec: Ternary classification

- For ternary classification, I created default vector using the np.zeros() function in case a word is missing from the google word2vec vocabulary.
- Then retrieved the vectors for each review by averaging the google word2vec vector for each word in the review.

[ ]:
```
DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.mean([google_model[word] if word in google_model else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ') ],axis=0) for sentence in␣
 ↪train_df['reviews'].tolist()]
```

```
X_test = [np.mean([google_model[word] if word in google_model else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')],axis=0) for sentence in␣
↪test_df['reviews'].tolist()]

y_train = train_df['label']
y_test = test_df['label']
```

- Below, I described the FFNN model consisting of two hidden layers with ReLU activation with 50 and 10 neurons respectively for the ternary classification task.
- I used the softmax non-linearity in the output layer with 3 neurons which gives an output probability distribution with the probability for each class, therefore the final predicted output is the class with maximum probability.
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_4 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((300,)),
    tf.keras.layers.Dense(50,activation='relu'),
    tf.keras.layers.Dense(10,activation='relu'),
    tf.keras.layers.Dense(3,activation='softmax')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_4', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_4.compile(optimizer=tf.keras.optimizers.
↪Adam(learning_rate=lr_schedule),loss='sparse_categorical_crossentropy',metrics=['accuracy']
model_4.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 50)                15050
```

```
_________________________________________________________________
dense_13 (Dense)             (None, 10)                510
_________________________________________________________________
dense_14 (Dense)             (None, 3)                 33
=================================================================
Total params: 15,593
Trainable params: 15,593
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the FFNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_4.fit(np.array(X_train),y_train,validation_data=(np.
 ↪array(X_test),y_test),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.7848 -
accuracy: 0.6599 - val_loss: 0.7566 - val_accuracy: 0.6731
INFO:tensorflow:Assets written to: models\model_4\assets
Epoch 2/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.7534 -
accuracy: 0.6744 - val_loss: 0.7452 - val_accuracy: 0.6790
INFO:tensorflow:Assets written to: models\model_4\assets
Epoch 3/50
6250/6250 [==============================] - 14s 2ms/step - loss: 0.7389 -
accuracy: 0.6811 - val_loss: 0.7392 - val_accuracy: 0.6801
INFO:tensorflow:Assets written to: models\model_4\assets
Epoch 4/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.7300 -
accuracy: 0.6844 - val_loss: 0.7359 - val_accuracy: 0.6811
INFO:tensorflow:Assets written to: models\model_4\assets
Epoch 5/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.7227 -
accuracy: 0.6883 - val_loss: 0.7412 - val_accuracy: 0.6787
Epoch 6/50
6250/6250 [==============================] - 18s 3ms/step - loss: 0.7160 -
accuracy: 0.6917 - val_loss: 0.7365 - val_accuracy: 0.6827
INFO:tensorflow:Assets written to: models\model_4\assets
Epoch 7/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.7110 -
accuracy: 0.6936 - val_loss: 0.7308 - val_accuracy: 0.6866
INFO:tensorflow:Assets written to: models\model_4\assets
Epoch 8/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.7061 -
accuracy: 0.6964 - val_loss: 0.7460 - val_accuracy: 0.6788
```

```
Epoch 9/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.7018 -
accuracy: 0.6973 - val_loss: 0.7378 - val_accuracy: 0.6864
Epoch 10/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6977 -
accuracy: 0.7002 - val_loss: 0.7325 - val_accuracy: 0.6870
INFO:tensorflow:Assets written to: models\model_4\assets
Epoch 11/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6946 -
accuracy: 0.7017 - val_loss: 0.7320 - val_accuracy: 0.6852
Epoch 12/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6911 -
accuracy: 0.7032 - val_loss: 0.7343 - val_accuracy: 0.6866
Epoch 13/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6887 -
accuracy: 0.7046 - val_loss: 0.7299 - val_accuracy: 0.6880
INFO:tensorflow:Assets written to: models\model_4\assets
Epoch 14/50
6250/6250 [==============================] - 14s 2ms/step - loss: 0.6854 -
accuracy: 0.7053 - val_loss: 0.7372 - val_accuracy: 0.6865
Epoch 15/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6830 -
accuracy: 0.7067 - val_loss: 0.7376 - val_accuracy: 0.6842
Epoch 16/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6811 -
accuracy: 0.7071 - val_loss: 0.7346 - val_accuracy: 0.6855
Epoch 17/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6782 -
accuracy: 0.7091 - val_loss: 0.7392 - val_accuracy: 0.6877
Epoch 18/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6762 -
accuracy: 0.7095 - val_loss: 0.7366 - val_accuracy: 0.6851
Epoch 19/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6741 -
accuracy: 0.7109 - val_loss: 0.7341 - val_accuracy: 0.6865
Epoch 20/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6725 -
accuracy: 0.7123 - val_loss: 0.7369 - val_accuracy: 0.6853
Epoch 21/50
6250/6250 [==============================] - 16s 2ms/step - loss: 0.6707 -
accuracy: 0.7119 - val_loss: 0.7366 - val_accuracy: 0.6867
Epoch 22/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6691 -
accuracy: 0.7131 - val_loss: 0.7456 - val_accuracy: 0.6853
Epoch 23/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6674 -
accuracy: 0.7136 - val_loss: 0.7426 - val_accuracy: 0.6848
Epoch 24/50
```

```
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6662 -
accuracy: 0.7143 - val_loss: 0.7453 - val_accuracy: 0.6818
Epoch 25/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6645 -
accuracy: 0.7148 - val_loss: 0.7466 - val_accuracy: 0.6832
Epoch 26/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6633 -
accuracy: 0.7156 - val_loss: 0.7421 - val_accuracy: 0.6839
Epoch 27/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6622 -
accuracy: 0.7161 - val_loss: 0.7440 - val_accuracy: 0.6853
Epoch 28/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6611 -
accuracy: 0.7164 - val_loss: 0.7444 - val_accuracy: 0.6849
Epoch 29/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6599 -
accuracy: 0.7169 - val_loss: 0.7454 - val_accuracy: 0.6843
Epoch 30/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6587 -
accuracy: 0.7182 - val_loss: 0.7481 - val_accuracy: 0.6836
Epoch 31/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6578 -
accuracy: 0.7182 - val_loss: 0.7469 - val_accuracy: 0.6825
Epoch 32/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6566 -
accuracy: 0.7193 - val_loss: 0.7521 - val_accuracy: 0.6852
Epoch 33/50
6250/6250 [==============================] - 14s 2ms/step - loss: 0.6558 -
accuracy: 0.7191 - val_loss: 0.7494 - val_accuracy: 0.6856
Epoch 34/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6548 -
accuracy: 0.7199 - val_loss: 0.7499 - val_accuracy: 0.6835
Epoch 35/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6542 -
accuracy: 0.7201 - val_loss: 0.7500 - val_accuracy: 0.6840
Epoch 36/50
6250/6250 [==============================] - 14s 2ms/step - loss: 0.6532 -
accuracy: 0.7205 - val_loss: 0.7529 - val_accuracy: 0.6839
Epoch 37/50
6250/6250 [==============================] - 14s 2ms/step - loss: 0.6524 -
accuracy: 0.7213 - val_loss: 0.7526 - val_accuracy: 0.6828
Epoch 38/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6516 -
accuracy: 0.7214 - val_loss: 0.7530 - val_accuracy: 0.6832
Epoch 39/50
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6512 -
accuracy: 0.7220 - val_loss: 0.7561 - val_accuracy: 0.6825
Epoch 40/50
```

```
6250/6250 [==============================] - 17s 3ms/step - loss: 0.6505 -
accuracy: 0.7216 - val_loss: 0.7589 - val_accuracy: 0.6827
Epoch 41/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6498 -
accuracy: 0.7222 - val_loss: 0.7543 - val_accuracy: 0.6832
Epoch 42/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6494 -
accuracy: 0.7223 - val_loss: 0.7545 - val_accuracy: 0.6824
Epoch 43/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6487 -
accuracy: 0.7225 - val_loss: 0.7570 - val_accuracy: 0.6817
Epoch 44/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6483 -
accuracy: 0.7224 - val_loss: 0.7574 - val_accuracy: 0.6821
Epoch 45/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6476 -
accuracy: 0.7231 - val_loss: 0.7594 - val_accuracy: 0.6836
Epoch 46/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6470 -
accuracy: 0.7234 - val_loss: 0.7621 - val_accuracy: 0.6799
Epoch 47/50
6250/6250 [==============================] - 15s 2ms/step - loss: 0.6465 -
accuracy: 0.7233 - val_loss: 0.7587 - val_accuracy: 0.6827
Epoch 48/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6463 -
accuracy: 0.7235 - val_loss: 0.7596 - val_accuracy: 0.6823
Epoch 49/50
6250/6250 [==============================] - 14s 2ms/step - loss: 0.6458 -
accuracy: 0.7242 - val_loss: 0.7611 - val_accuracy: 0.6827
Epoch 50/50
6250/6250 [==============================] - 16s 3ms/step - loss: 0.6455 -
accuracy: 0.7239 - val_loss: 0.7610 - val_accuracy: 0.6821
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x1c72ef06b50>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_4,'model_4',X_test,y_test)
```

```
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7299 -
accuracy: 0.6880
Loss = 0.7298944592475891 Accuracy = 0.688040018081665
```

4.2 To generate the input features, concatenate the first 10 Word2Vec vectors for each review as the input feature (x = [WT1, ..., WT10]) and train the neural network. Report the accuracy value on the testing split for your MLP model for each of the binary and ternary classification cases.

4.2.1 Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the word2vec vocabulary.
- Then retrieved the vectors for each review by concatenating my word2vec vector for the first 10 words in the review.

[ ]:
```
new_train_df = pd.concat([train_df[train_df['label'] ==␣
 ↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==␣
 ↪0],test_df[test_df['label'] == 1]])

DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.concatenate([w2v_model.wv[word] if word in w2v_model.wv else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:10] +␣
 ↪['<oov>']*(10-min(10,len(sentence.split(' '))))],axis=0) for sentence in␣
 ↪new_train_df['reviews'].tolist()]
X_test = [np.concatenate([w2v_model.wv[word] if word in w2v_model.wv else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:10] +␣
 ↪['<oov>']*(10-min(10,len(sentence.split(' '))))],axis=0) for sentence in␣
 ↪new_test_df['reviews'].tolist()]

y_train = new_train_df['label']
y_test = new_test_df['label']
```

- Below, I described the FFNN model consisting of two hidden layers with ReLU activation with 50 and 10 neurons respectively for the binary classification task.
- I used the tanh non-linearity in the output layer with 1 neuron which gives me an output in the range [-1,1], therefore for outputs <0, we predict class 0 (negative) and for outputs >0, we predict class 1 (positive).
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_5 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((3000,)),
    tf.keras.layers.Dense(50,activation='relu'),
    tf.keras.layers.Dense(10,activation='relu'),
```

```
    tf.keras.layers.Dense(1,activation='tanh')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_5', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_5.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='binary_crossentropy',metrics=['accuracy'])
model_5.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 50)                150050
_________________________________________________________________
dense_16 (Dense)             (None, 10)                510
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 11
=================================================================
Total params: 150,571
Trainable params: 150,571
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the FFNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_5.fit(np.array(X_train),y_train,validation_data=(np.
 ↪array(X_test),y_test),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
5002/5002 [==============================] - 43s 8ms/step - loss: 0.6678 -
accuracy: 0.7473 - val_loss: 0.4866 - val_accuracy: 0.7720
```

```
INFO:tensorflow:Assets written to: models\model_5\assets
Epoch 2/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.4670 -
accuracy: 0.7827 - val_loss: 1.1751 - val_accuracy: 0.7699
Epoch 3/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.4527 -
accuracy: 0.7952 - val_loss: 0.4531 - val_accuracy: 0.7877
INFO:tensorflow:Assets written to: models\model_5\assets
Epoch 4/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.4368 -
accuracy: 0.8054 - val_loss: 0.4767 - val_accuracy: 0.7835
Epoch 5/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.4056 -
accuracy: 0.8231 - val_loss: 0.4873 - val_accuracy: 0.7906
INFO:tensorflow:Assets written to: models\model_5\assets
Epoch 6/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.3972 -
accuracy: 0.8355 - val_loss: 0.5556 - val_accuracy: 0.7918
INFO:tensorflow:Assets written to: models\model_5\assets
Epoch 7/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.3973 -
accuracy: 0.8454 - val_loss: 0.4919 - val_accuracy: 0.7896
Epoch 8/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.3512 -
accuracy: 0.8591 - val_loss: 0.4895 - val_accuracy: 0.7878
Epoch 9/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.3291 -
accuracy: 0.8695 - val_loss: 0.5371 - val_accuracy: 0.7874
Epoch 10/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.3545 -
accuracy: 0.8786 - val_loss: 0.6032 - val_accuracy: 0.7894
Epoch 11/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.3433 -
accuracy: 0.8906 - val_loss: 0.5744 - val_accuracy: 0.7868
Epoch 12/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.3080 -
accuracy: 0.8978 - val_loss: 0.6566 - val_accuracy: 0.7884
Epoch 13/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.3243 -
accuracy: 0.9045 - val_loss: 0.6523 - val_accuracy: 0.7879
Epoch 14/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.2996 -
accuracy: 0.9075 - val_loss: 0.6353 - val_accuracy: 0.7867
Epoch 15/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.2928 -
accuracy: 0.9139 - val_loss: 0.6332 - val_accuracy: 0.7759
Epoch 16/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.2973 -
```

```
accuracy: 0.9197 - val_loss: 0.6608 - val_accuracy: 0.7859
Epoch 17/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.2842 -
accuracy: 0.9259 - val_loss: 0.6453 - val_accuracy: 0.7816
Epoch 18/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.2344 -
accuracy: 0.9297 - val_loss: 0.6980 - val_accuracy: 0.7837
Epoch 19/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.2278 -
accuracy: 0.9337 - val_loss: 0.7964 - val_accuracy: 0.7872
Epoch 20/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.2230 -
accuracy: 0.9360 - val_loss: 0.6830 - val_accuracy: 0.7833
Epoch 21/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1970 -
accuracy: 0.9415 - val_loss: 0.7177 - val_accuracy: 0.7803
Epoch 22/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.1816 -
accuracy: 0.9445 - val_loss: 0.7260 - val_accuracy: 0.7828
Epoch 23/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1720 -
accuracy: 0.9493 - val_loss: 0.8151 - val_accuracy: 0.7845
Epoch 24/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1947 -
accuracy: 0.9508 - val_loss: 0.8875 - val_accuracy: 0.7822
Epoch 25/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1972 -
accuracy: 0.9540 - val_loss: 0.9092 - val_accuracy: 0.7788
Epoch 26/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1913 -
accuracy: 0.9576 - val_loss: 0.9204 - val_accuracy: 0.7805
Epoch 27/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.2026 -
accuracy: 0.9593 - val_loss: 1.1694 - val_accuracy: 0.7813
Epoch 28/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.2149 -
accuracy: 0.9564 - val_loss: 1.0254 - val_accuracy: 0.7801
Epoch 29/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.1916 -
accuracy: 0.9604 - val_loss: 1.0017 - val_accuracy: 0.7777
Epoch 30/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.1611 -
accuracy: 0.9623 - val_loss: 0.9607 - val_accuracy: 0.7792
Epoch 31/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1484 -
accuracy: 0.9661 - val_loss: 0.9551 - val_accuracy: 0.7804
Epoch 32/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1448 -
```

```
accuracy: 0.9665 - val_loss: 0.9806 - val_accuracy: 0.7778
Epoch 33/50
5002/5002 [==============================] - 27s 5ms/step - loss: 0.1578 -
accuracy: 0.9680 - val_loss: 1.1397 - val_accuracy: 0.7793
Epoch 34/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1549 -
accuracy: 0.9682 - val_loss: 1.1656 - val_accuracy: 0.7800
Epoch 35/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1692 -
accuracy: 0.9708 - val_loss: 1.3722 - val_accuracy: 0.7791
Epoch 36/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1673 -
accuracy: 0.9712 - val_loss: 1.2184 - val_accuracy: 0.7796
Epoch 37/50
5002/5002 [==============================] - 27s 5ms/step - loss: 0.1632 -
accuracy: 0.9713 - val_loss: 1.3013 - val_accuracy: 0.7798
Epoch 38/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.1640 -
accuracy: 0.9726 - val_loss: 1.3158 - val_accuracy: 0.7767
Epoch 39/50
5002/5002 [==============================] - 27s 5ms/step - loss: 0.1665 -
accuracy: 0.9714 - val_loss: 1.5405 - val_accuracy: 0.7765
Epoch 40/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.1666 -
accuracy: 0.9730 - val_loss: 1.3412 - val_accuracy: 0.7779
Epoch 41/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1327 -
accuracy: 0.9754 - val_loss: 1.3184 - val_accuracy: 0.7784
Epoch 42/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1137 -
accuracy: 0.9746 - val_loss: 1.2540 - val_accuracy: 0.7791
Epoch 43/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1249 -
accuracy: 0.9757 - val_loss: 1.4900 - val_accuracy: 0.7800
Epoch 44/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.1380 -
accuracy: 0.9749 - val_loss: 1.5893 - val_accuracy: 0.7792
Epoch 45/50
5002/5002 [==============================] - 27s 5ms/step - loss: 0.1565 -
accuracy: 0.9765 - val_loss: 1.5472 - val_accuracy: 0.7788
Epoch 46/50
5002/5002 [==============================] - 26s 5ms/step - loss: 0.1379 -
accuracy: 0.9755 - val_loss: 1.4670 - val_accuracy: 0.7780
Epoch 47/50
5002/5002 [==============================] - 27s 5ms/step - loss: 0.1473 -
accuracy: 0.9762 - val_loss: 1.5219 - val_accuracy: 0.7769
Epoch 48/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.1299 -
```

```
accuracy: 0.9764 - val_loss: 1.4529 - val_accuracy: 0.7777
Epoch 49/50
5002/5002 [==============================] - 27s 5ms/step - loss: 0.1215 -
accuracy: 0.9782 - val_loss: 1.4543 - val_accuracy: 0.7764
Epoch 50/50
5002/5002 [==============================] - 27s 5ms/step - loss: 0.1300 -
accuracy: 0.9777 - val_loss: 1.5571 - val_accuracy: 0.7771
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x1c6f89e72b0>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_5,'model_5',X_test,y_test)
```

```
1249/1249 [==============================] - 5s 4ms/step - loss: 0.5556 -
accuracy: 0.7918
Loss = 0.5555978417396545 Accuracy = 0.7918376326560974
```

4.2.2 Google Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the google word2vec vocabulary.
- Then retrieved the vectors for each review by concatenating the google word2vec vector for the first 10 words in the review.

[ ]:
```
new_train_df = pd.concat([train_df[train_df['label'] == 0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] == 0],test_df[test_df['label'] == 1]])

DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.concatenate([google_model[word] if word in google_model else DEFAULT_VECTOR for word in sentence.split(' ')[:10] + ['<oov>']*(10-min(10,len(sentence.split(' '))))],axis=0) for sentence in new_train_df['reviews'].tolist()]
X_test = [np.concatenate([google_model[word] if word in google_model else DEFAULT_VECTOR for word in sentence.split(' ')[:10] + ['<oov>']*(10-min(10,len(sentence.split(' '))))],axis=0) for sentence in new_test_df['reviews'].tolist()]

y_train = new_train_df['label']
y_test = new_test_df['label']
```

- Below, I described the FFNN model consisting of two hidden layers with ReLU activation with 50 and 10 neurons respectively for the binary classification task.
- I used the tanh non-linearity in the output layer with 1 neuron which gives me an output in the range [-1,1], therefore for outputs <0, we predict class 0 (negative) and for outputs >0, we predict class 1 (positive).
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.01), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_6 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((3000,)),
    tf.keras.layers.Dense(50,activation='relu'),
    tf.keras.layers.Dense(10,activation='relu'),
    tf.keras.layers.Dense(1,activation='tanh')
])

initial_learning_rate = 0.01 #since this model was training slowly, kept␣
 ↪initial learning rate as 0.01

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_6', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_6.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='binary_crossentropy',metrics=['accuracy'])
model_6.summary()
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
dense (Dense)               (None, 50)                150050

_________________________________________________________________
dense_1 (Dense)             (None, 10)                510

_________________________________________________________________
dense_2 (Dense)             (None, 1)                 11
=================================================================
Total params: 150,571
```

```
Trainable params: 150,571
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the FFNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_6.fit(np.array(X_train),y_train,validation_data=(np.
 ↪array(X_test),y_test),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
5002/5002 [==============================] - 28s 5ms/step - loss: 0.5102 -
accuracy: 0.7494 - val_loss: 0.4951 - val_accuracy: 0.7582
INFO:tensorflow:Assets written to: models\model_6\assets
Epoch 2/50
5002/5002 [==============================] - 15s 3ms/step - loss: 0.4959 -
accuracy: 0.7730 - val_loss: 0.4727 - val_accuracy: 0.7721
INFO:tensorflow:Assets written to: models\model_6\assets
Epoch 3/50
5002/5002 [==============================] - 15s 3ms/step - loss: 0.4962 -
accuracy: 0.7914 - val_loss: 1.0316 - val_accuracy: 0.7694
Epoch 4/50
5002/5002 [==============================] - 16s 3ms/step - loss: 0.4614 -
accuracy: 0.8060 - val_loss: 0.5162 - val_accuracy: 0.7706
Epoch 5/50
5002/5002 [==============================] - 16s 3ms/step - loss: 0.4190 -
accuracy: 0.8212 - val_loss: 0.5279 - val_accuracy: 0.7724
INFO:tensorflow:Assets written to: models\model_6\assets
Epoch 6/50
5002/5002 [==============================] - 18s 4ms/step - loss: 0.4138 -
accuracy: 0.8353 - val_loss: 0.6066 - val_accuracy: 0.7693
Epoch 7/50
5002/5002 [==============================] - 15s 3ms/step - loss: 0.3981 -
accuracy: 0.8438 - val_loss: 0.5168 - val_accuracy: 0.7717
Epoch 8/50
5002/5002 [==============================] - 15s 3ms/step - loss: 0.3493 -
accuracy: 0.8577 - val_loss: 0.5386 - val_accuracy: 0.7693
Epoch 9/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.3503 -
accuracy: 0.8672 - val_loss: 0.6790 - val_accuracy: 0.7703
Epoch 10/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.3400 -
accuracy: 0.8764 - val_loss: 0.6184 - val_accuracy: 0.7693
Epoch 11/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.3376 -
accuracy: 0.8804 - val_loss: 0.5715 - val_accuracy: 0.7684
```

```
Epoch 12/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.3100 -
accuracy: 0.8896 - val_loss: 0.7087 - val_accuracy: 0.7703
Epoch 13/50
5002/5002 [==============================] - 16s 3ms/step - loss: 0.3273 -
accuracy: 0.8960 - val_loss: 0.7643 - val_accuracy: 0.7650
Epoch 14/50
5002/5002 [==============================] - 15s 3ms/step - loss: 0.3114 -
accuracy: 0.8998 - val_loss: 0.6459 - val_accuracy: 0.7656
Epoch 15/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.2648 -
accuracy: 0.9056 - val_loss: 0.6567 - val_accuracy: 0.7651
Epoch 16/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2596 -
accuracy: 0.9106 - val_loss: 0.7547 - val_accuracy: 0.7621
Epoch 17/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.2608 -
accuracy: 0.9153 - val_loss: 0.7184 - val_accuracy: 0.7575
Epoch 18/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2423 -
accuracy: 0.9154 - val_loss: 0.7439 - val_accuracy: 0.7611
Epoch 19/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.2162 -
accuracy: 0.9227 - val_loss: 0.7408 - val_accuracy: 0.7628
Epoch 20/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.2065 -
accuracy: 0.9264 - val_loss: 0.7785 - val_accuracy: 0.7616
Epoch 21/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.1986 -
accuracy: 0.9297 - val_loss: 0.7358 - val_accuracy: 0.7615
Epoch 22/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1915 -
accuracy: 0.9336 - val_loss: 0.8484 - val_accuracy: 0.7618
Epoch 23/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.1917 -
accuracy: 0.9356 - val_loss: 0.8404 - val_accuracy: 0.7597
Epoch 24/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.1939 -
accuracy: 0.9383 - val_loss: 0.9449 - val_accuracy: 0.7591
Epoch 25/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.1935 -
accuracy: 0.9418 - val_loss: 1.0125 - val_accuracy: 0.7607
Epoch 26/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.2022 -
accuracy: 0.9428 - val_loss: 0.9913 - val_accuracy: 0.7588
Epoch 27/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1943 -
accuracy: 0.9445 - val_loss: 0.9339 - val_accuracy: 0.7583
```

```
Epoch 28/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.1790 -
accuracy: 0.9461 - val_loss: 0.9921 - val_accuracy: 0.7583
Epoch 29/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.1780 -
accuracy: 0.9478 - val_loss: 1.0821 - val_accuracy: 0.7599
Epoch 30/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.1783 -
accuracy: 0.9494 - val_loss: 1.1425 - val_accuracy: 0.7595
Epoch 31/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.1885 -
accuracy: 0.9496 - val_loss: 1.1741 - val_accuracy: 0.7575
Epoch 32/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1756 -
accuracy: 0.9508 - val_loss: 1.0551 - val_accuracy: 0.7564
Epoch 33/50
5002/5002 [==============================] - 11s 2ms/step - loss: 0.1661 -
accuracy: 0.9527 - val_loss: 1.1429 - val_accuracy: 0.7570
Epoch 34/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.1660 -
accuracy: 0.9540 - val_loss: 1.1639 - val_accuracy: 0.7564
Epoch 35/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1802 -
accuracy: 0.9546 - val_loss: 1.3748 - val_accuracy: 0.7558
Epoch 36/50
5002/5002 [==============================] - 15s 3ms/step - loss: 0.1625 -
accuracy: 0.9552 - val_loss: 1.1759 - val_accuracy: 0.7572
Epoch 37/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1526 -
accuracy: 0.9568 - val_loss: 1.1870 - val_accuracy: 0.7558
Epoch 38/50
5002/5002 [==============================] - 16s 3ms/step - loss: 0.1629 -
accuracy: 0.9574 - val_loss: 1.3147 - val_accuracy: 0.7567
Epoch 39/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1696 -
accuracy: 0.9584 - val_loss: 1.3244 - val_accuracy: 0.7572
Epoch 40/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1681 -
accuracy: 0.9592 - val_loss: 1.3141 - val_accuracy: 0.7567
Epoch 41/50
5002/5002 [==============================] - 13s 3ms/step - loss: 0.1638 -
accuracy: 0.9600 - val_loss: 1.3668 - val_accuracy: 0.7570
Epoch 42/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1609 -
accuracy: 0.9605 - val_loss: 1.3989 - val_accuracy: 0.7561
Epoch 43/50
5002/5002 [==============================] - 15s 3ms/step - loss: 0.1558 -
accuracy: 0.9613 - val_loss: 1.3359 - val_accuracy: 0.7549
```

```
Epoch 44/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1548 -
accuracy: 0.9612 - val_loss: 1.3785 - val_accuracy: 0.7560
Epoch 45/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.1571 -
accuracy: 0.9619 - val_loss: 1.3646 - val_accuracy: 0.7548
Epoch 46/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1598 -
accuracy: 0.9627 - val_loss: 1.4697 - val_accuracy: 0.7564
Epoch 47/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.1581 -
accuracy: 0.9630 - val_loss: 1.4470 - val_accuracy: 0.7560
Epoch 48/50
5002/5002 [==============================] - 12s 2ms/step - loss: 0.1540 -
accuracy: 0.9633 - val_loss: 1.4935 - val_accuracy: 0.7557
Epoch 49/50
5002/5002 [==============================] - 14s 3ms/step - loss: 0.1520 -
accuracy: 0.9638 - val_loss: 1.4976 - val_accuracy: 0.7552
Epoch 50/50
5002/5002 [==============================] - 15s 3ms/step - loss: 0.1465 -
accuracy: 0.9639 - val_loss: 1.4845 - val_accuracy: 0.7547
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x1d54b93e760>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_6,'model_6',X_test,y_test)
```

```
1249/1249 [==============================] - 2s 2ms/step - loss: 0.5279 -
accuracy: 0.7724
Loss = 0.5279173851013184 Accuracy = 0.7724452018737793
```

4.2.3 Word2Vec: Ternary classification

- For ternary classification, I created default vector using the np.zeros() function in case a word is missing from the word2vec vocabulary.
- Then retrieved the vectors for each review by concatenating my word2vec vector for the first 10 words in the review.

[ ]:
```
DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.concatenate([w2v_model.wv[word] if word in w2v_model.wv else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:10] +␣
 ↪['<oov>']*(10-min(10,len(sentence.split(' '))))],axis=0) for sentence in␣
 ↪train_df['reviews'].tolist()]
```

```
X_test = [np.concatenate([w2v_model.wv[word] if word in w2v_model.wv else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')[:10] +␣
↪['<oov>']*(10-min(10,len(sentence.split(' '))))],axis=0) for sentence in␣
↪test_df['reviews'].tolist()]

y_train = train_df['label']
y_test = test_df['label']
```

- Below, I described the FFNN model consisting of two hidden layers with ReLU activation with 50 and 10 neurons respectively for the ternary classification task.
- I used the softmax non-linearity in the output layer with 3 neurons which gives an output probability distribution with the probability for each class, therefore the final predicted output is the class with maximum probability.
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_7 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((3000,)),
    tf.keras.layers.Dense(50,activation='relu'),
    tf.keras.layers.Dense(10,activation='relu'),
    tf.keras.layers.Dense(3,activation='softmax')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_7', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_7.compile(optimizer=tf.keras.optimizers.
↪Adam(learning_rate=lr_schedule),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
model_7.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
dense_6 (Dense)              (None, 50)                150050
_________________________________________________________________
dense_7 (Dense)              (None, 10)                510
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 33
=================================================================
Total params: 150,593
Trainable params: 150,593
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the FFNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_7.fit(np.array(X_train),y_train,validation_data=(np.
 ↪array(X_test),y_test),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
6250/6250 [==============================] - 47s 7ms/step - loss: 0.8598 -
accuracy: 0.6191 - val_loss: 0.8429 - val_accuracy: 0.6260
INFO:tensorflow:Assets written to: models\model_7\assets
Epoch 2/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.8180 -
accuracy: 0.6404 - val_loss: 0.8336 - val_accuracy: 0.6370
INFO:tensorflow:Assets written to: models\model_7\assets
Epoch 3/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.7881 -
accuracy: 0.6544 - val_loss: 0.8321 - val_accuracy: 0.6343
Epoch 4/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.7576 -
accuracy: 0.6684 - val_loss: 0.8349 - val_accuracy: 0.6373
INFO:tensorflow:Assets written to: models\model_7\assets
Epoch 5/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.7265 -
accuracy: 0.6814 - val_loss: 0.8381 - val_accuracy: 0.6315
Epoch 6/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.6976 -
accuracy: 0.6926 - val_loss: 0.8625 - val_accuracy: 0.6336
Epoch 7/50
6250/6250 [==============================] - 33s 5ms/step - loss: 0.6705 -
accuracy: 0.7055 - val_loss: 0.8709 - val_accuracy: 0.6298
Epoch 8/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.6448 -
accuracy: 0.7160 - val_loss: 0.9045 - val_accuracy: 0.6209
Epoch 9/50
6250/6250 [==============================] - 33s 5ms/step - loss: 0.6213 -
```

```
accuracy: 0.7262 - val_loss: 0.9466 - val_accuracy: 0.6241
Epoch 10/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.5994 -
accuracy: 0.7362 - val_loss: 1.0039 - val_accuracy: 0.6169
Epoch 11/50
6250/6250 [==============================] - 33s 5ms/step - loss: 0.5793 -
accuracy: 0.7445 - val_loss: 1.0430 - val_accuracy: 0.6163
Epoch 12/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.5610 -
accuracy: 0.7526 - val_loss: 1.0624 - val_accuracy: 0.6125
Epoch 13/50
6250/6250 [==============================] - 33s 5ms/step - loss: 0.5422 -
accuracy: 0.7613 - val_loss: 1.1017 - val_accuracy: 0.6115
Epoch 14/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.5273 -
accuracy: 0.7680 - val_loss: 1.1539 - val_accuracy: 0.6074
Epoch 15/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.5129 -
accuracy: 0.7746 - val_loss: 1.1456 - val_accuracy: 0.6091
Epoch 16/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.4975 -
accuracy: 0.7818 - val_loss: 1.2135 - val_accuracy: 0.6068
Epoch 17/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.4836 -
accuracy: 0.7879 - val_loss: 1.2781 - val_accuracy: 0.5969
Epoch 18/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.4716 -
accuracy: 0.7940 - val_loss: 1.2792 - val_accuracy: 0.6017
Epoch 19/50
6250/6250 [==============================] - 32s 5ms/step - loss: 0.4592 -
accuracy: 0.7992 - val_loss: 1.4173 - val_accuracy: 0.5988
Epoch 20/50
6250/6250 [==============================] - 38s 6ms/step - loss: 0.4488 -
accuracy: 0.8044 - val_loss: 1.3901 - val_accuracy: 0.6013
Epoch 21/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.4369 -
accuracy: 0.8098 - val_loss: 1.4512 - val_accuracy: 0.5968
Epoch 22/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.4276 -
accuracy: 0.8146 - val_loss: 1.6057 - val_accuracy: 0.5993
Epoch 23/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.4179 -
accuracy: 0.8189 - val_loss: 1.6039 - val_accuracy: 0.5942
Epoch 24/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.4091 -
accuracy: 0.8238 - val_loss: 1.6640 - val_accuracy: 0.5943
Epoch 25/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.4005 -
```

```
accuracy: 0.8272 - val_loss: 1.7555 - val_accuracy: 0.5987
Epoch 26/50
6250/6250 [==============================] - 34s 6ms/step - loss: 0.3914 -
accuracy: 0.8311 - val_loss: 1.8958 - val_accuracy: 0.6003
Epoch 27/50
6250/6250 [==============================] - 38s 6ms/step - loss: 0.3841 -
accuracy: 0.8349 - val_loss: 1.8408 - val_accuracy: 0.5941
Epoch 28/50
6250/6250 [==============================] - 47s 7ms/step - loss: 0.3771 -
accuracy: 0.8392 - val_loss: 1.9794 - val_accuracy: 0.5936
Epoch 29/50
6250/6250 [==============================] - 51s 8ms/step - loss: 0.3698 -
accuracy: 0.8423 - val_loss: 2.0198 - val_accuracy: 0.5965
Epoch 30/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.3630 -
accuracy: 0.8456 - val_loss: 2.1697 - val_accuracy: 0.5951
Epoch 31/50
6250/6250 [==============================] - 51s 8ms/step - loss: 0.3572 -
accuracy: 0.8489 - val_loss: 2.1927 - val_accuracy: 0.5931
Epoch 32/50
6250/6250 [==============================] - 53s 8ms/step - loss: 0.3512 -
accuracy: 0.8518 - val_loss: 2.2423 - val_accuracy: 0.5954
Epoch 33/50
6250/6250 [==============================] - 51s 8ms/step - loss: 0.3448 -
accuracy: 0.8553 - val_loss: 2.3756 - val_accuracy: 0.5956
Epoch 34/50
6250/6250 [==============================] - 48s 8ms/step - loss: 0.3397 -
accuracy: 0.8570 - val_loss: 2.4543 - val_accuracy: 0.5914
Epoch 35/50
6250/6250 [==============================] - 51s 8ms/step - loss: 0.3338 -
accuracy: 0.8604 - val_loss: 2.5372 - val_accuracy: 0.5893
Epoch 36/50
6250/6250 [==============================] - 53s 9ms/step - loss: 0.3300 -
accuracy: 0.8623 - val_loss: 2.6978 - val_accuracy: 0.5927
Epoch 37/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.3244 -
accuracy: 0.8649 - val_loss: 2.6556 - val_accuracy: 0.5891
Epoch 38/50
6250/6250 [==============================] - 49s 8ms/step - loss: 0.3203 -
accuracy: 0.8672 - val_loss: 2.8169 - val_accuracy: 0.5926
Epoch 39/50
6250/6250 [==============================] - 48s 8ms/step - loss: 0.3161 -
accuracy: 0.8697 - val_loss: 2.8477 - val_accuracy: 0.5900
Epoch 40/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.3119 -
accuracy: 0.8710 - val_loss: 3.0645 - val_accuracy: 0.5927
Epoch 41/50
6250/6250 [==============================] - 47s 7ms/step - loss: 0.3079 -
```

```
accuracy: 0.8735 - val_loss: 3.0826 - val_accuracy: 0.5934
Epoch 42/50
6250/6250 [==============================] - 48s 8ms/step - loss: 0.3043 -
accuracy: 0.8748 - val_loss: 3.1919 - val_accuracy: 0.5935
Epoch 43/50
6250/6250 [==============================] - 55s 9ms/step - loss: 0.3014 -
accuracy: 0.8765 - val_loss: 3.2761 - val_accuracy: 0.5929
Epoch 44/50
6250/6250 [==============================] - 54s 9ms/step - loss: 0.2975 -
accuracy: 0.8782 - val_loss: 3.3734 - val_accuracy: 0.5923
Epoch 45/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.2947 -
accuracy: 0.8798 - val_loss: 3.4754 - val_accuracy: 0.5914
Epoch 46/50
6250/6250 [==============================] - 48s 8ms/step - loss: 0.2916 -
accuracy: 0.8816 - val_loss: 3.5694 - val_accuracy: 0.5913
Epoch 47/50
6250/6250 [==============================] - 5854s 937ms/step - loss: 0.2886 -
accuracy: 0.8828 - val_loss: 3.6919 - val_accuracy: 0.5934
Epoch 48/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.2863 -
accuracy: 0.8838 - val_loss: 3.7648 - val_accuracy: 0.5927
Epoch 49/50
6250/6250 [==============================] - 43s 7ms/step - loss: 0.2831 -
accuracy: 0.8854 - val_loss: 3.8358 - val_accuracy: 0.5906
Epoch 50/50
6250/6250 [==============================] - 65s 10ms/step - loss: 0.2811 -
accuracy: 0.8868 - val_loss: 3.9118 - val_accuracy: 0.5896
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x1d24c46d0a0>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_7,'model_7',X_test,y_test)
```

```
1563/1563 [==============================] - 11s 7ms/step - loss: 0.8349 -
accuracy: 0.6373
Loss = 0.8349009156227112 Accuracy = 0.6373000144958496
```

4.2.4 Google Word2Vec: Ternary classification

- For ternary classification, I created default vector using the np.zeros() function in case a word is missing from the google word2vec vocabulary.
- Then retrieved the vectors for each review by concatenating the google word2vec vector for the first 10 words in the review.

[ ]:
```
DEFAULT_VECTOR = np.zeros((300,))

X_train = [np.concatenate([google_model[word] if word in google_model else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')[:10] +␣
↪['<oov>']*(10-min(10,len(sentence.split(' '))))],axis=0) for sentence in␣
↪train_df['reviews'].tolist()]
X_test = [np.concatenate([google_model[word] if word in google_model else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')[:10] +␣
↪['<oov>']*(10-min(10,len(sentence.split(' '))))],axis=0) for sentence in␣
↪test_df['reviews'].tolist()]

y_train = train_df['label']
y_test = test_df['label']
```

- Below, I described the FFNN model consisting of two hidden layers with ReLU activation with 50 and 10 neurons respectively for the ternary classification task.
- I used the softmax non-linearity in the output layer with 3 neurons which gives an output probability distribution with the probability for each class, therefore the final predicted output is the class with maximum probability.
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_8 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((3000,)),
    tf.keras.layers.Dense(50,activation='relu'),
    tf.keras.layers.Dense(10,activation='relu'),
    tf.keras.layers.Dense(3,activation='softmax')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_8', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_8.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
                loss='sparse_categorical_crossentropy',
```

```
                 metrics=['accuracy'])

model_8.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 50)                150050
_________________________________________________________________
dense_4 (Dense)              (None, 10)                510
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 33
=================================================================
Total params: 150,593
Trainable params: 150,593
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the FFNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_8.fit(np.array(X_train),y_train,validation_data=(np.array(X_test),y_test),
            batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
6250/6250 [==============================] - 16s 2ms/step - loss: 0.8638 -
accuracy: 0.6133 - val_loss: 0.8473 - val_accuracy: 0.6243
INFO:tensorflow:Assets written to: models\model_8\assets
Epoch 2/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.8123 -
accuracy: 0.6411 - val_loss: 0.8374 - val_accuracy: 0.6290
INFO:tensorflow:Assets written to: models\model_8\assets
Epoch 3/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.7717 -
accuracy: 0.6619 - val_loss: 0.8439 - val_accuracy: 0.6279
Epoch 4/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.7318 -
accuracy: 0.6816 - val_loss: 0.8655 - val_accuracy: 0.6281
Epoch 5/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.6948 -
accuracy: 0.6989 - val_loss: 0.8832 - val_accuracy: 0.6248
Epoch 6/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.6613 -
accuracy: 0.7141 - val_loss: 0.9150 - val_accuracy: 0.6228
```

```
Epoch 7/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.6307 -
accuracy: 0.7275 - val_loss: 0.9714 - val_accuracy: 0.6167
Epoch 8/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.6029 -
accuracy: 0.7401 - val_loss: 1.0179 - val_accuracy: 0.6091
Epoch 9/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.5781 -
accuracy: 0.7496 - val_loss: 1.0567 - val_accuracy: 0.6098
Epoch 10/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.5560 -
accuracy: 0.7607 - val_loss: 1.1010 - val_accuracy: 0.6067
Epoch 11/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.5356 -
accuracy: 0.7692 - val_loss: 1.1371 - val_accuracy: 0.5997
Epoch 12/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.5170 -
accuracy: 0.7771 - val_loss: 1.2009 - val_accuracy: 0.6026
Epoch 13/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.5004 -
accuracy: 0.7857 - val_loss: 1.2442 - val_accuracy: 0.6011
Epoch 14/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.4837 -
accuracy: 0.7920 - val_loss: 1.3179 - val_accuracy: 0.6031
Epoch 15/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.4695 -
accuracy: 0.7987 - val_loss: 1.3836 - val_accuracy: 0.5932
Epoch 16/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.4554 -
accuracy: 0.8050 - val_loss: 1.5196 - val_accuracy: 0.5969
Epoch 17/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.4413 -
accuracy: 0.8111 - val_loss: 1.5595 - val_accuracy: 0.5937
Epoch 18/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.4294 -
accuracy: 0.8160 - val_loss: 1.6527 - val_accuracy: 0.5884
Epoch 19/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.4177 -
accuracy: 0.8211 - val_loss: 1.7398 - val_accuracy: 0.5936
Epoch 20/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.4060 -
accuracy: 0.8257 - val_loss: 1.9274 - val_accuracy: 0.5875
Epoch 21/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.3957 -
accuracy: 0.8308 - val_loss: 1.9636 - val_accuracy: 0.5869
Epoch 22/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.3863 -
accuracy: 0.8353 - val_loss: 2.0661 - val_accuracy: 0.5870
```

```
Epoch 23/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.3767 -
accuracy: 0.8388 - val_loss: 2.2017 - val_accuracy: 0.5891
Epoch 24/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.3679 -
accuracy: 0.8427 - val_loss: 2.3546 - val_accuracy: 0.5800
Epoch 25/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.3592 -
accuracy: 0.8467 - val_loss: 2.4751 - val_accuracy: 0.5847
Epoch 26/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.3504 -
accuracy: 0.8504 - val_loss: 2.6392 - val_accuracy: 0.5808
Epoch 27/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.3430 -
accuracy: 0.8542 - val_loss: 2.7024 - val_accuracy: 0.5807
Epoch 28/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.3353 -
accuracy: 0.8576 - val_loss: 2.8697 - val_accuracy: 0.5795
Epoch 29/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.3286 -
accuracy: 0.8605 - val_loss: 3.0476 - val_accuracy: 0.5802
Epoch 30/50
6250/6250 [==============================] - 12s 2ms/step - loss: 0.3217 -
accuracy: 0.8633 - val_loss: 3.2016 - val_accuracy: 0.5829
Epoch 31/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.3160 -
accuracy: 0.8655 - val_loss: 3.2635 - val_accuracy: 0.5836
Epoch 32/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.3099 -
accuracy: 0.8679 - val_loss: 3.4983 - val_accuracy: 0.5818
Epoch 33/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.3037 -
accuracy: 0.8714 - val_loss: 3.6985 - val_accuracy: 0.5838
Epoch 34/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2988 -
accuracy: 0.8738 - val_loss: 3.7760 - val_accuracy: 0.5828
Epoch 35/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2939 -
accuracy: 0.8754 - val_loss: 3.9542 - val_accuracy: 0.5835
Epoch 36/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2883 -
accuracy: 0.8785 - val_loss: 4.1343 - val_accuracy: 0.5835
Epoch 37/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2842 -
accuracy: 0.8804 - val_loss: 4.3080 - val_accuracy: 0.5837
Epoch 38/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2796 -
accuracy: 0.8826 - val_loss: 4.5024 - val_accuracy: 0.5832
```

```
Epoch 39/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2764 -
accuracy: 0.8840 - val_loss: 4.5733 - val_accuracy: 0.5818
Epoch 40/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2720 -
accuracy: 0.8856 - val_loss: 4.7879 - val_accuracy: 0.5809
Epoch 41/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2684 -
accuracy: 0.8874 - val_loss: 4.9503 - val_accuracy: 0.5823
Epoch 42/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2654 -
accuracy: 0.8887 - val_loss: 5.0712 - val_accuracy: 0.5824
Epoch 43/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2618 -
accuracy: 0.8902 - val_loss: 5.2184 - val_accuracy: 0.5814
Epoch 44/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2585 -
accuracy: 0.8918 - val_loss: 5.3656 - val_accuracy: 0.5820
Epoch 45/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2558 -
accuracy: 0.8929 - val_loss: 5.5282 - val_accuracy: 0.5811
Epoch 46/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2528 -
accuracy: 0.8944 - val_loss: 5.6798 - val_accuracy: 0.5814
Epoch 47/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2503 -
accuracy: 0.8951 - val_loss: 5.7871 - val_accuracy: 0.5805
Epoch 48/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2479 -
accuracy: 0.8962 - val_loss: 5.9369 - val_accuracy: 0.5817
Epoch 49/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2452 -
accuracy: 0.8974 - val_loss: 6.1104 - val_accuracy: 0.5822
Epoch 50/50
6250/6250 [==============================] - 13s 2ms/step - loss: 0.2434 -
accuracy: 0.8981 - val_loss: 6.2760 - val_accuracy: 0.5818
```

[ ]:
```
<keras.callbacks.History at 0x26cef4fc348>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_8,'model_8',X_test,y_test)
```

```
1563/1563 [==============================] - 4s 2ms/step - loss: 0.8374 -
accuracy: 0.6290
Loss = 0.8373900651931763 Accuracy = 0.6289799809455872
```

4.3 What do you conclude by comparing accuracy values you obtain with those obtained in the "Simple Models" section (note you can compare the accuracy values for binary classification).

Simple Models - The simple models achieved an overall accuracy of approx 85% for binary classification task.

FFNN - The FFNN achieved the best accuracy of approx 87% for the binary classification task. - The models achieved a best accuracy of approx 70.6% on the ternary classification task.

In conclusion, FFNN performed slightly better than Simple Models on binary classification task. This can be attributed to the ability of neural networks to model more complex functions than simple models.

5 RNN

5.1 Train a simple RNN for sentiment analysis. You can consider an RNN cell with the hidden state size of 50. To feed your data into our RNN, limit the maximum review length to 50 by truncating longer reviews and padding shorter reviews with a null value (0). Train the RNN network for binary classification using class 1 and class 2 and also a ternary model for the three classes. Report accuracy values on the testing split for your RNN model.

5.1.1 Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the word2vec vocabulary.
- Then retrieved the tensor for each review by concatenating my word2vec vector for the first 20 words in the review. The tensor shape for 1 review comes out to be (20,300).
- Although the assignment specified us to take the first 50 words, I took the first 20 words as suggested by Prof. Rostami due to lack of computational resources.

[ ]:
```
new_train_df = pd.concat([train_df[train_df['label'] ==␣
↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==␣
↪0],test_df[test_df['label'] == 1]])

DEFAULT_VECTOR = np.zeros((300,))

X_train = np.array([[w2v_model.wv[word] if word in w2v_model.wv else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
↪new_train_df['reviews'].tolist()])
X_test = np.array([[w2v_model.wv[word] if word in w2v_model.wv else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
↪new_test_df['reviews'].tolist()])
```

```
y_train = new_train_df['label'].tolist()
y_test = new_test_df['label'].tolist()
```

- Below, I described the RNN model consisting of one RNN layer with 50 neurons for the binary classification task. By default, the RNN's return_sequences parameter is False, therefore it only returns a vector of size 50 at the last timestep.
- I used the tanh non-linearity in the output layer with 1 neuron which gives me an output in the range [-1,1], therefore for outputs <0, we predict class 0 (negative) and for outputs >0, we predict class 1 (positive).
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_9 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((20,300)),
    tf.keras.layers.SimpleRNN(50),
    tf.keras.layers.Dense(1,activation='tanh')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_9', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_9.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='binary_crossentropy',metrics=['accuracy'])
model_9.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_3 (SimpleRNN)     (None, 50)                17550
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 51
=================================================================
```

```
Total params: 17,601
Trainable params: 17,601
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the RNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_9.fit(np.array(X_train),np.array(y_train),validation_data=(np.
↪array(X_test),np.
↪array(y_test)),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
5002/5002 [==============================] - 32s 6ms/step - loss: 1.4929 -
accuracy: 0.5407 - val_loss: 1.5336 - val_accuracy: 0.5340
INFO:tensorflow:Assets written to: models\model_9\assets
Epoch 2/50
5002/5002 [==============================] - 27s 5ms/step - loss: 0.7798 -
accuracy: 0.5527 - val_loss: 0.6695 - val_accuracy: 0.5933
INFO:tensorflow:Assets written to: models\model_9\assets
Epoch 3/50
5002/5002 [==============================] - 28s 6ms/step - loss: 1.3360 -
accuracy: 0.5205 - val_loss: 1.5673 - val_accuracy: 0.5186
Epoch 4/50
5002/5002 [==============================] - 27s 5ms/step - loss: 1.1751 -
accuracy: 0.5245 - val_loss: 0.8972 - val_accuracy: 0.5456
Epoch 5/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.7495 -
accuracy: 0.5518 - val_loss: 0.6807 - val_accuracy: 0.5604
Epoch 6/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6803 -
accuracy: 0.5732 - val_loss: 0.6659 - val_accuracy: 0.5962
INFO:tensorflow:Assets written to: models\model_9\assets
Epoch 7/50
5002/5002 [==============================] - 26s 5ms/step - loss: 0.6656 -
accuracy: 0.6091 - val_loss: 0.6574 - val_accuracy: 0.6127
INFO:tensorflow:Assets written to: models\model_9\assets
Epoch 8/50
5002/5002 [==============================] - 27s 5ms/step - loss: 0.6598 -
accuracy: 0.6155 - val_loss: 0.6524 - val_accuracy: 0.6299
INFO:tensorflow:Assets written to: models\model_9\assets
Epoch 9/50
5002/5002 [==============================] - 31s 6ms/step - loss: 0.7423 -
accuracy: 0.6195 - val_loss: 0.6379 - val_accuracy: 0.6417
INFO:tensorflow:Assets written to: models\model_9\assets
Epoch 10/50
```

```
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6436 -
accuracy: 0.6387 - val_loss: 0.6476 - val_accuracy: 0.6377
Epoch 11/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6643 -
accuracy: 0.6249 - val_loss: 0.6492 - val_accuracy: 0.6232
Epoch 12/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6634 -
accuracy: 0.6066 - val_loss: 0.6677 - val_accuracy: 0.6076
Epoch 13/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6735 -
accuracy: 0.5889 - val_loss: 0.6787 - val_accuracy: 0.5783
Epoch 14/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6614 -
accuracy: 0.6081 - val_loss: 0.6526 - val_accuracy: 0.6155
Epoch 15/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6603 -
accuracy: 0.6162 - val_loss: 0.6525 - val_accuracy: 0.6230
Epoch 16/50
5002/5002 [==============================] - 31s 6ms/step - loss: 0.7219 -
accuracy: 0.5990 - val_loss: 0.7024 - val_accuracy: 0.5771
Epoch 17/50
5002/5002 [==============================] - 31s 6ms/step - loss: 0.6523 -
accuracy: 0.6221 - val_loss: 0.6489 - val_accuracy: 0.6317
Epoch 18/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6504 -
accuracy: 0.6271 - val_loss: 0.6500 - val_accuracy: 0.6257
Epoch 19/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6527 -
accuracy: 0.6244 - val_loss: 0.6499 - val_accuracy: 0.6239
Epoch 20/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6519 -
accuracy: 0.6239 - val_loss: 0.6585 - val_accuracy: 0.6080
Epoch 21/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.6507 -
accuracy: 0.6263 - val_loss: 0.6486 - val_accuracy: 0.6298
Epoch 22/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6510 -
accuracy: 0.6262 - val_loss: 0.6460 - val_accuracy: 0.6329
Epoch 23/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6497 -
accuracy: 0.6272 - val_loss: 0.6476 - val_accuracy: 0.6308
Epoch 24/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6553 -
accuracy: 0.6204 - val_loss: 0.6570 - val_accuracy: 0.6164
Epoch 25/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6518 -
accuracy: 0.6206 - val_loss: 0.6515 - val_accuracy: 0.6225
Epoch 26/50
```

```
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6546 -
accuracy: 0.6180 - val_loss: 0.6537 - val_accuracy: 0.6217
Epoch 27/50
5002/5002 [==============================] - 31s 6ms/step - loss: 0.6557 -
accuracy: 0.6179 - val_loss: 0.6540 - val_accuracy: 0.6247
Epoch 28/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6583 -
accuracy: 0.6146 - val_loss: 0.6530 - val_accuracy: 0.6220
Epoch 29/50
5002/5002 [==============================] - 31s 6ms/step - loss: 0.6603 -
accuracy: 0.6099 - val_loss: 0.6741 - val_accuracy: 0.6051
Epoch 30/50
5002/5002 [==============================] - 31s 6ms/step - loss: 0.6647 -
accuracy: 0.6028 - val_loss: 0.6645 - val_accuracy: 0.5999
Epoch 31/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6638 -
accuracy: 0.6034 - val_loss: 0.6642 - val_accuracy: 0.6050
Epoch 32/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6636 -
accuracy: 0.6054 - val_loss: 0.6593 - val_accuracy: 0.6130
Epoch 33/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6620 -
accuracy: 0.6089 - val_loss: 0.6645 - val_accuracy: 0.5956
Epoch 34/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6649 -
accuracy: 0.6027 - val_loss: 0.6642 - val_accuracy: 0.6009
Epoch 35/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6653 -
accuracy: 0.6007 - val_loss: 0.6681 - val_accuracy: 0.6011
Epoch 36/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6680 -
accuracy: 0.5970 - val_loss: 0.6733 - val_accuracy: 0.5902
Epoch 37/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6710 -
accuracy: 0.5925 - val_loss: 0.6625 - val_accuracy: 0.6059
Epoch 38/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6657 -
accuracy: 0.5986 - val_loss: 0.6616 - val_accuracy: 0.6056
Epoch 39/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6670 -
accuracy: 0.5949 - val_loss: 0.6694 - val_accuracy: 0.5992
Epoch 40/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6716 -
accuracy: 0.5881 - val_loss: 0.6720 - val_accuracy: 0.5823
Epoch 41/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6704 -
accuracy: 0.5901 - val_loss: 0.6663 - val_accuracy: 0.5943
Epoch 42/50
```

```
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6691 -
accuracy: 0.5905 - val_loss: 0.6687 - val_accuracy: 0.5978
Epoch 43/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6729 -
accuracy: 0.5853 - val_loss: 0.6706 - val_accuracy: 0.5885
Epoch 44/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6705 -
accuracy: 0.5894 - val_loss: 0.6627 - val_accuracy: 0.6071
Epoch 45/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6645 -
accuracy: 0.6035 - val_loss: 0.6595 - val_accuracy: 0.6114
Epoch 46/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6645 -
accuracy: 0.6019 - val_loss: 0.6619 - val_accuracy: 0.6065
Epoch 47/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6624 -
accuracy: 0.6086 - val_loss: 0.6601 - val_accuracy: 0.6175
Epoch 48/50
5002/5002 [==============================] - 32s 6ms/step - loss: 0.6599 -
accuracy: 0.6135 - val_loss: 0.6575 - val_accuracy: 0.6168
Epoch 49/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6596 -
accuracy: 0.6138 - val_loss: 0.6578 - val_accuracy: 0.6149
Epoch 50/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6617 -
accuracy: 0.6100 - val_loss: 0.6598 - val_accuracy: 0.6125
```

[ ]: `<keras.callbacks.History at 0x26f0a7fddc8>`

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_9,'model_9',X_test,np.array(y_test))
```

```
1249/1249 [==============================] - 5s 3ms/step - loss: 0.6379 -
accuracy: 0.6417
Loss = 0.6379140615463257 Accuracy = 0.6416774988174438
```

5.1.2 Google Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the google word2vec vocabulary.
- Then retrieved the tensor for each review by concatenating the google word2vec vector for the first 20 words in the review. The tensor shape for 1 review comes out to be (20,300).
- Although the assignment specified us to take the first 50 words, I took the first 20 words as suggested by Prof. Rostami due to lack of computational resources.

```
new_train_df = pd.concat([train_df[train_df['label'] ==
 ↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==
 ↪0],test_df[test_df['label'] == 1]])

DEFAULT_VECTOR = np.zeros((300,))

X_train = np.array([[google_model[word] if word in google_model else
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +
 ↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in
 ↪new_train_df['reviews'].tolist()])
X_test = np.array([[google_model[word] if word in google_model else
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +
 ↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in
 ↪new_test_df['reviews'].tolist()])

y_train = new_train_df['label'].tolist()
y_test = new_test_df['label'].tolist()
```

- Below, I described the RNN model consisting of one RNN layer with 50 neurons for the binary classification task. By default, the RNN's return_sequences parameter is False, therefore it only returns a vector of size 50 at the last timestep.
- I used the tanh non-linearity in the output layer with 1 neuron which gives me an output in the range [-1,1], therefore for outputs <0, we predict class 0 (negative) and for outputs >0, we predict class 1 (positive).
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

```
model_10 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((20,300)),
    tf.keras.layers.SimpleRNN(50),
    tf.keras.layers.Dense(1,activation='tanh')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
```

```
    'models/model_10', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_10.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='binary_crossentropy',metrics=['accuracy'])
model_10.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_4 (SimpleRNN)     (None, 50)                17550
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 51
=================================================================
Total params: 17,601
Trainable params: 17,601
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the RNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_10.fit(np.array(X_train),np.array(y_train),validation_data=(np.
 ↪array(X_test),np.
 ↪array(y_test)),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
5002/5002 [==============================] - 40s 8ms/step - loss: 4.2494 -
accuracy: 0.5248 - val_loss: 0.7546 - val_accuracy: 0.5092
INFO:tensorflow:Assets written to: models\model_10\assets
Epoch 2/50
5002/5002 [==============================] - 32s 6ms/step - loss: 5.6529 -
accuracy: 0.5268 - val_loss: 6.2822 - val_accuracy: 0.5705
INFO:tensorflow:Assets written to: models\model_10\assets
Epoch 3/50
5002/5002 [==============================] - 32s 6ms/step - loss: 5.7106 -
accuracy: 0.5977 - val_loss: 5.3080 - val_accuracy: 0.6134
INFO:tensorflow:Assets written to: models\model_10\assets
Epoch 4/50
5002/5002 [==============================] - 29s 6ms/step - loss: 5.2828 -
accuracy: 0.5667 - val_loss: 0.6952 - val_accuracy: 0.5116
Epoch 5/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.7526 -
```

```
accuracy: 0.5427 - val_loss: 0.6952 - val_accuracy: 0.5119
Epoch 6/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6892 -
accuracy: 0.5578 - val_loss: 0.6492 - val_accuracy: 0.6584
INFO:tensorflow:Assets written to: models\model_10\assets
Epoch 7/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6758 -
accuracy: 0.5866 - val_loss: 0.6894 - val_accuracy: 0.5584
Epoch 8/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6298 -
accuracy: 0.6538 - val_loss: 0.6006 - val_accuracy: 0.6919
INFO:tensorflow:Assets written to: models\model_10\assets
Epoch 9/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6401 -
accuracy: 0.6278 - val_loss: 0.6334 - val_accuracy: 0.6459
Epoch 10/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6230 -
accuracy: 0.6549 - val_loss: 0.6368 - val_accuracy: 0.5893
Epoch 11/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6319 -
accuracy: 0.6507 - val_loss: 0.5871 - val_accuracy: 0.7057
INFO:tensorflow:Assets written to: models\model_10\assets
Epoch 12/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6446 -
accuracy: 0.6812 - val_loss: 0.6030 - val_accuracy: 0.6913
Epoch 13/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6342 -
accuracy: 0.6757 - val_loss: 0.5799 - val_accuracy: 0.6980
Epoch 14/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6026 -
accuracy: 0.6860 - val_loss: 0.5824 - val_accuracy: 0.6979
Epoch 15/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5975 -
accuracy: 0.7053 - val_loss: 0.5509 - val_accuracy: 0.7412
INFO:tensorflow:Assets written to: models\model_10\assets
Epoch 16/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5920 -
accuracy: 0.6942 - val_loss: 0.6307 - val_accuracy: 0.6786
Epoch 17/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6729 -
accuracy: 0.6772 - val_loss: 0.5609 - val_accuracy: 0.7343
Epoch 18/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5803 -
accuracy: 0.7071 - val_loss: 0.5687 - val_accuracy: 0.7292
Epoch 19/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5776 -
accuracy: 0.7127 - val_loss: 0.7285 - val_accuracy: 0.5076
Epoch 20/50
```

```
5002/5002 [==============================] - 29s 6ms/step - loss: 0.6293 -
accuracy: 0.6595 - val_loss: 0.5852 - val_accuracy: 0.7118
Epoch 21/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.6058 -
accuracy: 0.7008 - val_loss: 0.5543 - val_accuracy: 0.7235
Epoch 22/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5594 -
accuracy: 0.7292 - val_loss: 0.5609 - val_accuracy: 0.7393
Epoch 23/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5611 -
accuracy: 0.7322 - val_loss: 0.5563 - val_accuracy: 0.7222
Epoch 24/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5845 -
accuracy: 0.7116 - val_loss: 0.5689 - val_accuracy: 0.7026
Epoch 25/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5578 -
accuracy: 0.7271 - val_loss: 0.5528 - val_accuracy: 0.7339
Epoch 26/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5528 -
accuracy: 0.7358 - val_loss: 0.5548 - val_accuracy: 0.7271
Epoch 27/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.5778 -
accuracy: 0.7170 - val_loss: 0.5711 - val_accuracy: 0.7267
Epoch 28/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5611 -
accuracy: 0.7334 - val_loss: 0.5755 - val_accuracy: 0.7243
Epoch 29/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5724 -
accuracy: 0.7225 - val_loss: 0.6300 - val_accuracy: 0.6947
Epoch 30/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5837 -
accuracy: 0.7106 - val_loss: 0.5720 - val_accuracy: 0.7262
Epoch 31/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5883 -
accuracy: 0.7312 - val_loss: 0.5646 - val_accuracy: 0.7235
Epoch 32/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5891 -
accuracy: 0.7176 - val_loss: 0.5651 - val_accuracy: 0.7090
Epoch 33/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5690 -
accuracy: 0.7185 - val_loss: 0.5616 - val_accuracy: 0.7128
Epoch 34/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5592 -
accuracy: 0.7190 - val_loss: 0.5626 - val_accuracy: 0.7108
Epoch 35/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5643 -
accuracy: 0.7203 - val_loss: 0.5806 - val_accuracy: 0.7055
Epoch 36/50
```

```
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5729 -
accuracy: 0.7170 - val_loss: 0.5641 - val_accuracy: 0.7279
Epoch 37/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5648 -
accuracy: 0.7215 - val_loss: 0.5675 - val_accuracy: 0.7307
Epoch 38/50
5002/5002 [==============================] - 29s 6ms/step - loss: 0.5467 -
accuracy: 0.7334 - val_loss: 0.5523 - val_accuracy: 0.7110
Epoch 39/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5435 -
accuracy: 0.7365 - val_loss: 0.5481 - val_accuracy: 0.7283
Epoch 40/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5453 -
accuracy: 0.7283 - val_loss: 0.5422 - val_accuracy: 0.7254
Epoch 41/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5425 -
accuracy: 0.7253 - val_loss: 0.5333 - val_accuracy: 0.7399
Epoch 42/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5419 -
accuracy: 0.7364 - val_loss: 0.5438 - val_accuracy: 0.7396
Epoch 43/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5326 -
accuracy: 0.7465 - val_loss: 0.5442 - val_accuracy: 0.7219
Epoch 44/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5395 -
accuracy: 0.7340 - val_loss: 0.5487 - val_accuracy: 0.7276
Epoch 45/50
5002/5002 [==============================] - 28s 6ms/step - loss: 0.5376 -
accuracy: 0.7379 - val_loss: 0.5375 - val_accuracy: 0.7422
INFO:tensorflow:Assets written to: models\model_10\assets
Epoch 46/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.5355 -
accuracy: 0.7433 - val_loss: 0.5393 - val_accuracy: 0.7393
Epoch 47/50
5002/5002 [==============================] - 30s 6ms/step - loss: 0.5361 -
accuracy: 0.7441 - val_loss: 0.5346 - val_accuracy: 0.7432
INFO:tensorflow:Assets written to: models\model_10\assets
Epoch 48/50
5002/5002 [==============================] - 31s 6ms/step - loss: 0.5390 -
accuracy: 0.7424 - val_loss: 0.5448 - val_accuracy: 0.7385
Epoch 49/50
5002/5002 [==============================] - 31s 6ms/step - loss: 0.5423 -
accuracy: 0.7403 - val_loss: 0.5425 - val_accuracy: 0.7392
Epoch 50/50
5002/5002 [==============================] - 31s 6ms/step - loss: 0.5376 -
accuracy: 0.7394 - val_loss: 0.5451 - val_accuracy: 0.7395
```

[ ]: <keras.callbacks.History at 0x26ee409bac8>

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_10,'model_10',X_test,np.array(y_test))
```

```
1249/1249 [==============================] - 5s 4ms/step - loss: 0.5346 -
accuracy: 0.7432
Loss = 0.534572422504425 Accuracy = 0.743243932723999
```

5.1.3 Word2Vec: Ternary classification

- For ternary classification, I created default vector using the np.zeros() function in case a word is missing from the word2vec vocabulary.
- Then retrieved the tensor for each review by concatenating the word2vec vector for the first 20 words in the review. The tensor shape for 1 review comes out to be (20,300).
- Although the assignment specified us to take the first 50 words, I took the first 20 words as suggested by Prof. Rostami due to lack of computational resources.

[ ]:
```
DEFAULT_VECTOR = np.zeros((300,))

X_train = np.array([[w2v_model.wv[word] if word in w2v_model.wv else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
↪train_df['reviews'].tolist()])
X_test = np.array([[w2v_model.wv[word] if word in w2v_model.wv else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
↪test_df['reviews'].tolist()])

y_train = train_df['label'].tolist()
y_test = test_df['label'].tolist()
```

- Below, I described the RNN model consisting of one RNN layer with 50 neurons for the ternary classification task. By default, the RNN's return_sequences parameter is False, therefore it only returns a vector of size 50 at the last timestep.
- I used the softmax non-linearity in the output layer with 3 neurons which gives an output probability distribution with the probability for each class, therefore the final predicted output is the class with maximum probability.
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_11 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((20,300)),
```

```
    tf.keras.layers.SimpleRNN(50),
    tf.keras.layers.Dense(3,activation='softmax')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_11', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_11.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='sparse_categorical_crossentropy',metrics=['accuracy']
model_11.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_5 (SimpleRNN)     (None, 50)                17550
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 153
=================================================================
Total params: 17,703
Trainable params: 17,703
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the RNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_11.fit(np.array(X_train),np.array(y_train),validation_data=(np.
 ↪array(X_test),np.
 ↪array(y_test)),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
6250/6250 [==============================] - 52s 8ms/step - loss: 1.0226 -
accuracy: 0.5061 - val_loss: 1.0362 - val_accuracy: 0.4929
```

```
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 2/50
6250/6250 [==============================] - 35s 6ms/step - loss: 1.0128 -
accuracy: 0.5148 - val_loss: 0.9963 - val_accuracy: 0.5476
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 3/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9953 -
accuracy: 0.5375 - val_loss: 0.9971 - val_accuracy: 0.5255
Epoch 4/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9983 -
accuracy: 0.5358 - val_loss: 0.9838 - val_accuracy: 0.5456
Epoch 5/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9847 -
accuracy: 0.5415 - val_loss: 0.9978 - val_accuracy: 0.5226
Epoch 6/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9905 -
accuracy: 0.5382 - val_loss: 0.9820 - val_accuracy: 0.5484
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 7/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9833 -
accuracy: 0.5543 - val_loss: 0.9863 - val_accuracy: 0.5499
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 8/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9903 -
accuracy: 0.5500 - val_loss: 0.9878 - val_accuracy: 0.5524
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 9/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9832 -
accuracy: 0.5559 - val_loss: 0.9825 - val_accuracy: 0.5542
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 10/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9790 -
accuracy: 0.5593 - val_loss: 0.9786 - val_accuracy: 0.5567
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 11/50
6250/6250 [==============================] - 32s 5ms/step - loss: 0.9787 -
accuracy: 0.5618 - val_loss: 0.9709 - val_accuracy: 0.5653
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 12/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9656 -
accuracy: 0.5739 - val_loss: 0.9790 - val_accuracy: 0.5678
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 13/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9661 -
accuracy: 0.5721 - val_loss: 0.9682 - val_accuracy: 0.5698
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 14/50
6250/6250 [==============================] - 34s 6ms/step - loss: 0.9616 -
```

```
accuracy: 0.5802 - val_loss: 0.9613 - val_accuracy: 0.5801
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 15/50
6250/6250 [==============================] - 38s 6ms/step - loss: 0.9605 -
accuracy: 0.5821 - val_loss: 0.9586 - val_accuracy: 0.5824
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 16/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9577 -
accuracy: 0.5847 - val_loss: 0.9599 - val_accuracy: 0.5808
Epoch 17/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9606 -
accuracy: 0.5795 - val_loss: 0.9689 - val_accuracy: 0.5706
Epoch 18/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9604 -
accuracy: 0.5781 - val_loss: 0.9614 - val_accuracy: 0.5752
Epoch 19/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9536 -
accuracy: 0.5753 - val_loss: 0.9518 - val_accuracy: 0.5745
Epoch 20/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9508 -
accuracy: 0.5803 - val_loss: 0.9538 - val_accuracy: 0.5789
Epoch 21/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9486 -
accuracy: 0.5795 - val_loss: 0.9512 - val_accuracy: 0.5788
Epoch 22/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9473 -
accuracy: 0.5806 - val_loss: 0.9468 - val_accuracy: 0.5803
Epoch 23/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9451 -
accuracy: 0.5796 - val_loss: 0.9485 - val_accuracy: 0.5804
Epoch 24/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9408 -
accuracy: 0.5839 - val_loss: 0.9407 - val_accuracy: 0.5839
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 25/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9394 -
accuracy: 0.5836 - val_loss: 0.9488 - val_accuracy: 0.5798
Epoch 26/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9419 -
accuracy: 0.5826 - val_loss: 0.9451 - val_accuracy: 0.5785
Epoch 27/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9427 -
accuracy: 0.5840 - val_loss: 0.9429 - val_accuracy: 0.5827
Epoch 28/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9381 -
accuracy: 0.5868 - val_loss: 0.9370 - val_accuracy: 0.5868
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 29/50
```

```
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9377 -
accuracy: 0.5880 - val_loss: 0.9453 - val_accuracy: 0.5830
Epoch 30/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9359 -
accuracy: 0.5881 - val_loss: 0.9351 - val_accuracy: 0.5893
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 31/50
6250/6250 [==============================] - 37s 6ms/step - loss: 0.9294 -
accuracy: 0.5947 - val_loss: 0.9297 - val_accuracy: 0.5952
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 32/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9290 -
accuracy: 0.5971 - val_loss: 0.9335 - val_accuracy: 0.5939
Epoch 33/50
6250/6250 [==============================] - 40s 6ms/step - loss: 0.9274 -
accuracy: 0.5978 - val_loss: 0.9306 - val_accuracy: 0.5944
Epoch 34/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9262 -
accuracy: 0.5954 - val_loss: 0.9275 - val_accuracy: 0.5927
Epoch 35/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9227 -
accuracy: 0.5973 - val_loss: 0.9302 - val_accuracy: 0.5888
Epoch 36/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9229 -
accuracy: 0.5935 - val_loss: 0.9244 - val_accuracy: 0.5951
Epoch 37/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9222 -
accuracy: 0.5992 - val_loss: 0.9259 - val_accuracy: 0.5972
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 38/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9226 -
accuracy: 0.5994 - val_loss: 0.9261 - val_accuracy: 0.5975
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 39/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9231 -
accuracy: 0.5998 - val_loss: 0.9298 - val_accuracy: 0.5974
Epoch 40/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9237 -
accuracy: 0.5981 - val_loss: 0.9272 - val_accuracy: 0.5943
Epoch 41/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9231 -
accuracy: 0.5997 - val_loss: 0.9277 - val_accuracy: 0.5984
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 42/50
6250/6250 [==============================] - 38s 6ms/step - loss: 0.9238 -
accuracy: 0.6008 - val_loss: 0.9270 - val_accuracy: 0.5975
Epoch 43/50
6250/6250 [==============================] - 39s 6ms/step - loss: 0.9243 -
```

```
accuracy: 0.6000 - val_loss: 0.9281 - val_accuracy: 0.5972
Epoch 44/50
6250/6250 [==============================] - 40s 6ms/step - loss: 0.9237 -
accuracy: 0.6010 - val_loss: 0.9269 - val_accuracy: 0.5988
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 45/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9235 -
accuracy: 0.6015 - val_loss: 0.9296 - val_accuracy: 0.5979
Epoch 46/50
6250/6250 [==============================] - 37s 6ms/step - loss: 0.9242 -
accuracy: 0.6008 - val_loss: 0.9284 - val_accuracy: 0.5980
Epoch 47/50
6250/6250 [==============================] - 38s 6ms/step - loss: 0.9240 -
accuracy: 0.6011 - val_loss: 0.9291 - val_accuracy: 0.5972
Epoch 48/50
6250/6250 [==============================] - 37s 6ms/step - loss: 0.9227 -
accuracy: 0.6019 - val_loss: 0.9266 - val_accuracy: 0.5994
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 49/50
6250/6250 [==============================] - 34s 6ms/step - loss: 0.9233 -
accuracy: 0.6024 - val_loss: 0.9274 - val_accuracy: 0.6005
INFO:tensorflow:Assets written to: models\model_11\assets
Epoch 50/50
6250/6250 [==============================] - 31s 5ms/step - loss: 0.9250 -
accuracy: 0.6004 - val_loss: 0.9289 - val_accuracy: 0.5993
```

[ ]:
```
<keras.callbacks.History at 0x26cf007d3c8>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_11,'model_11',X_test,np.array(y_test))
```

```
1563/1563 [==============================] - 5s 3ms/step - loss: 0.9274 -
accuracy: 0.6005
Loss = 0.927401065826416 Accuracy = 0.6005200147628784
```

5.1.4 Google Word2Vec: Ternary classification

- For ternary classification, I created default vector using the np.zeros() function in case a word is missing from the google word2vec vocabulary.
- Then retrieved the tensor for each review by concatenating the google word2vec vector for the first 20 words in the review. The tensor shape for 1 review comes out to be (20,300).
- Although the assignment specified us to take the first 50 words, I took the first 20 words as suggested by Prof. Rostami due to lack of computational resources.

[ ]:
```
DEFAULT_VECTOR = np.zeros((300,))

X_train = np.array([[google_model[word] if word in google_model else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
↪train_df['reviews'].tolist()])
X_test = np.array([[google_model[word] if word in google_model else␣
↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
↪test_df['reviews'].tolist()])

y_train = train_df['label'].tolist()
y_test = test_df['label'].tolist()
```

- Below, I described the RNN model consisting of one RNN layer with 50 neurons for the ternary classification task. By default, the RNN's return_sequences parameter is False, therefore it only returns a vector of size 50 at the last timestep.
- I used the softmax non-linearity in the output layer with 3 neurons which gives an output probability distribution with the probability for each class, therefore the final predicted output is the class with maximum probability.
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_12 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((20,300)),
    tf.keras.layers.SimpleRNN(50),
    tf.keras.layers.Dense(3,activation='softmax')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_12', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_12.compile(optimizer=tf.keras.optimizers.
↪Adam(learning_rate=lr_schedule),loss='sparse_categorical_crossentropy',metrics=['accuracy']
```

```
model_12.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_6 (SimpleRNN)     (None, 50)                17550
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 153
=================================================================
Total params: 17,703
Trainable params: 17,703
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the RNN for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_12.fit(np.array(X_train),np.array(y_train),validation_data=(np.
↪array(X_test),np.
↪array(y_test)),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
6250/6250 [==============================] - 49s 8ms/step - loss: 0.9926 -
accuracy: 0.5363 - val_loss: 0.9790 - val_accuracy: 0.5438
INFO:tensorflow:Assets written to: models\model_12\assets
Epoch 2/50
6250/6250 [==============================] - 34s 5ms/step - loss: 1.0178 -
accuracy: 0.5049 - val_loss: 0.9971 - val_accuracy: 0.5310
Epoch 3/50
6250/6250 [==============================] - 33s 5ms/step - loss: 1.0147 -
accuracy: 0.5128 - val_loss: 1.0463 - val_accuracy: 0.4526
Epoch 4/50
6250/6250 [==============================] - 34s 5ms/step - loss: 1.0157 -
accuracy: 0.5076 - val_loss: 1.0110 - val_accuracy: 0.5162
Epoch 5/50
6250/6250 [==============================] - 33s 5ms/step - loss: 1.0132 -
accuracy: 0.5047 - val_loss: 0.9993 - val_accuracy: 0.5320
Epoch 6/50
6250/6250 [==============================] - 33s 5ms/step - loss: 1.0333 -
accuracy: 0.4755 - val_loss: 1.0481 - val_accuracy: 0.4631
Epoch 7/50
6250/6250 [==============================] - 33s 5ms/step - loss: 1.0366 -
accuracy: 0.4680 - val_loss: 1.0598 - val_accuracy: 0.4097
Epoch 8/50
```

```
6250/6250 [==============================] - 33s 5ms/step - loss: 1.0138 -
accuracy: 0.5027 - val_loss: 0.9929 - val_accuracy: 0.5580
INFO:tensorflow:Assets written to: models\model_12\assets
Epoch 9/50
6250/6250 [==============================] - 32s 5ms/step - loss: 0.9845 -
accuracy: 0.5506 - val_loss: 0.9998 - val_accuracy: 0.5301
Epoch 10/50
6250/6250 [==============================] - 32s 5ms/step - loss: 1.0129 -
accuracy: 0.5107 - val_loss: 1.0198 - val_accuracy: 0.5186
Epoch 11/50
6250/6250 [==============================] - 32s 5ms/step - loss: 0.9991 -
accuracy: 0.5334 - val_loss: 0.9996 - val_accuracy: 0.5396
Epoch 12/50
6250/6250 [==============================] - 32s 5ms/step - loss: 1.0038 -
accuracy: 0.5304 - val_loss: 0.9869 - val_accuracy: 0.5440
Epoch 13/50
6250/6250 [==============================] - 32s 5ms/step - loss: 1.0023 -
accuracy: 0.5344 - val_loss: 0.9963 - val_accuracy: 0.5358
Epoch 14/50
6250/6250 [==============================] - 32s 5ms/step - loss: 0.9934 -
accuracy: 0.5473 - val_loss: 0.9741 - val_accuracy: 0.5655
INFO:tensorflow:Assets written to: models\model_12\assets
Epoch 15/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9974 -
accuracy: 0.5392 - val_loss: 0.9818 - val_accuracy: 0.5580
Epoch 16/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9948 -
accuracy: 0.5271 - val_loss: 1.0085 - val_accuracy: 0.5060
Epoch 17/50
6250/6250 [==============================] - 33s 5ms/step - loss: 1.0080 -
accuracy: 0.4955 - val_loss: 1.0218 - val_accuracy: 0.4865
Epoch 18/50
6250/6250 [==============================] - 33s 5ms/step - loss: 1.0035 -
accuracy: 0.5063 - val_loss: 0.9797 - val_accuracy: 0.5536
Epoch 19/50
6250/6250 [==============================] - 33s 5ms/step - loss: 0.9808 -
accuracy: 0.5420 - val_loss: 0.9678 - val_accuracy: 0.5695
INFO:tensorflow:Assets written to: models\model_12\assets
Epoch 20/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9708 -
accuracy: 0.5531 - val_loss: 0.9975 - val_accuracy: 0.5253
Epoch 21/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9943 -
accuracy: 0.5196 - val_loss: 1.0049 - val_accuracy: 0.4953
Epoch 22/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9787 -
accuracy: 0.5454 - val_loss: 0.9574 - val_accuracy: 0.5720
INFO:tensorflow:Assets written to: models\model_12\assets
```

```
Epoch 23/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9748 -
accuracy: 0.5508 - val_loss: 0.9718 - val_accuracy: 0.5643
Epoch 24/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9614 -
accuracy: 0.5707 - val_loss: 0.9692 - val_accuracy: 0.5650
Epoch 25/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9754 -
accuracy: 0.5534 - val_loss: 0.9910 - val_accuracy: 0.5485
Epoch 26/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9733 -
accuracy: 0.5575 - val_loss: 0.9741 - val_accuracy: 0.5581
Epoch 27/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9583 -
accuracy: 0.5670 - val_loss: 0.9530 - val_accuracy: 0.5764
INFO:tensorflow:Assets written to: models\model_12\assets
Epoch 28/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9577 -
accuracy: 0.5728 - val_loss: 0.9500 - val_accuracy: 0.5768
INFO:tensorflow:Assets written to: models\model_12\assets
Epoch 29/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9513 -
accuracy: 0.5753 - val_loss: 0.9517 - val_accuracy: 0.5685
Epoch 30/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9474 -
accuracy: 0.5730 - val_loss: 0.9491 - val_accuracy: 0.5615
Epoch 31/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9449 -
accuracy: 0.5749 - val_loss: 0.9569 - val_accuracy: 0.5645
Epoch 32/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9474 -
accuracy: 0.5677 - val_loss: 0.9435 - val_accuracy: 0.5791
INFO:tensorflow:Assets written to: models\model_12\assets
Epoch 33/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9421 -
accuracy: 0.5843 - val_loss: 0.9574 - val_accuracy: 0.5787
Epoch 34/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9435 -
accuracy: 0.5742 - val_loss: 0.9460 - val_accuracy: 0.5859
INFO:tensorflow:Assets written to: models\model_12\assets
Epoch 35/50
6250/6250 [==============================] - 33s 5ms/step - loss: 0.9503 -
accuracy: 0.5784 - val_loss: 0.9466 - val_accuracy: 0.5848
Epoch 36/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9460 -
accuracy: 0.5815 - val_loss: 0.9436 - val_accuracy: 0.5831
Epoch 37/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9451 -
```

```
accuracy: 0.5805 - val_loss: 0.9399 - val_accuracy: 0.5769
Epoch 38/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9487 -
accuracy: 0.5730 - val_loss: 0.9431 - val_accuracy: 0.5673
Epoch 39/50
6250/6250 [==============================] - 33s 5ms/step - loss: 0.9519 -
accuracy: 0.5630 - val_loss: 0.9511 - val_accuracy: 0.5618
Epoch 40/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9454 -
accuracy: 0.5783 - val_loss: 0.9526 - val_accuracy: 0.5779
Epoch 41/50
6250/6250 [==============================] - 34s 5ms/step - loss: 0.9478 -
accuracy: 0.5800 - val_loss: 0.9465 - val_accuracy: 0.5771
Epoch 42/50
6250/6250 [==============================] - 33s 5ms/step - loss: 0.9448 -
accuracy: 0.5812 - val_loss: 0.9358 - val_accuracy: 0.5905
INFO:tensorflow:Assets written to: models\model_12\assets
Epoch 43/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9405 -
accuracy: 0.5866 - val_loss: 0.9511 - val_accuracy: 0.5742
Epoch 44/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9483 -
accuracy: 0.5816 - val_loss: 0.9492 - val_accuracy: 0.5810
Epoch 45/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9466 -
accuracy: 0.5833 - val_loss: 0.9475 - val_accuracy: 0.5775
Epoch 46/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9405 -
accuracy: 0.5846 - val_loss: 0.9388 - val_accuracy: 0.5855
Epoch 47/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9371 -
accuracy: 0.5850 - val_loss: 0.9402 - val_accuracy: 0.5847
Epoch 48/50
6250/6250 [==============================] - 37s 6ms/step - loss: 0.9375 -
accuracy: 0.5832 - val_loss: 0.9381 - val_accuracy: 0.5833
Epoch 49/50
6250/6250 [==============================] - 35s 6ms/step - loss: 0.9355 -
accuracy: 0.5786 - val_loss: 0.9297 - val_accuracy: 0.5844
Epoch 50/50
6250/6250 [==============================] - 36s 6ms/step - loss: 0.9324 -
accuracy: 0.5826 - val_loss: 0.9365 - val_accuracy: 0.5862
```

[ ]:
```
<keras.callbacks.History at 0x26cefe5b388>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_12,'model_12',X_test,np.array(y_test))
```

```
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9358 -
accuracy: 0.5905
Loss = 0.9357994794845581 Accuracy = 0.590499997138977
```

5.2 GRU

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

5.2.1 Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the word2vec vocabulary.
- Then retrieved the tensor for each review by concatenating my word2vec vector for the first 20 words in the review. The tensor shape for 1 review comes out to be (20,300).
- Although the assignment specified us to take the first 50 words, I took the first 20 words as suggested by Prof. Rostami due to lack of computational resources.

```
[ ]: new_train_df = pd.concat([train_df[train_df['label'] ==␣
      ↪0],train_df[train_df['label'] == 1]])

     new_test_df = pd.concat([test_df[test_df['label'] ==␣
      ↪0],test_df[test_df['label'] == 1]])



     DEFAULT_VECTOR = np.zeros((300,))

     X_train = np.array([[w2v_model.wv[word] if word in w2v_model.wv else␣
      ↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
      ↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
      ↪new_train_df['reviews'].tolist()])
     X_test = np.array([[w2v_model.wv[word] if word in w2v_model.wv else␣
      ↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
      ↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
      ↪new_test_df['reviews'].tolist()])

     y_train = new_train_df['label'].tolist()
     y_test = new_test_df['label'].tolist()
```

- Below, I described the GRU model consisting of one GRU layer with 50 neurons for the binary classification task. By default, the GRU's return_sequences parameter is False, therefore it only returns a vector of size 50 at the last timestep.
- I used the tanh non-linearity in the output layer with 1 neuron which gives me an output in the range [-1,1], therefore for outputs <0, we predict class 0 (negative) and for outputs >0, we predict class 1 (positive).
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.

- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_13 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((20,300)),
    tf.keras.layers.GRU(50),
    tf.keras.layers.Dense(1,activation='tanh')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_13', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)

model_13.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='binary_crossentropy',metrics=['accuracy'])
model_13.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 50)                52800
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 51
=================================================================
Total params: 52,851
Trainable params: 52,851
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the GRU for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:

```
model_13.fit(np.array(X_train),np.array(y_train),validation_data=(np.
 ↪array(X_test),np.
 ↪array(y_test)),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
5002/5002 [==============================] - 76s 15ms/step - loss: 0.7413 -
accuracy: 0.7321 - val_loss: 0.7387 - val_accuracy: 0.6983

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 2/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.6764 -
accuracy: 0.7344 - val_loss: 0.5139 - val_accuracy: 0.7829

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 3/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.6688 -
accuracy: 0.7110 - val_loss: 0.6472 - val_accuracy: 0.7390
Epoch 4/50
5002/5002 [==============================] - 75s 15ms/step - loss: 0.5596 -
accuracy: 0.7523 - val_loss: 0.5957 - val_accuracy: 0.7744
Epoch 5/50
5002/5002 [==============================] - 75s 15ms/step - loss: 0.7261 -
accuracy: 0.7219 - val_loss: 0.7360 - val_accuracy: 0.7103
Epoch 6/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.5966 -
accuracy: 0.7459 - val_loss: 0.5351 - val_accuracy: 0.7440
Epoch 7/50
5002/5002 [==============================] - 73s 15ms/step - loss: 0.9215 -
accuracy: 0.6736 - val_loss: 0.8897 - val_accuracy: 0.7100
Epoch 8/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.7412 -
```

```
accuracy: 0.7062 - val_loss: 0.9108 - val_accuracy: 0.5089
Epoch 9/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.6543 -
accuracy: 0.7136 - val_loss: 0.9101 - val_accuracy: 0.6779
Epoch 10/50
5002/5002 [==============================] - 73s 15ms/step - loss: 0.6686 -
accuracy: 0.7212 - val_loss: 0.5611 - val_accuracy: 0.7754
Epoch 11/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.5370 -
accuracy: 0.7544 - val_loss: 0.6376 - val_accuracy: 0.7405
Epoch 12/50
5002/5002 [==============================] - 73s 15ms/step - loss: 0.5809 -
accuracy: 0.7434 - val_loss: 0.7363 - val_accuracy: 0.6825
Epoch 13/50
5002/5002 [==============================] - 73s 15ms/step - loss: 0.5683 -
accuracy: 0.7487 - val_loss: 0.4606 - val_accuracy: 0.7931

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 14/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.5919 -
accuracy: 0.7377 - val_loss: 0.5301 - val_accuracy: 0.7657
Epoch 15/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.5112 -
accuracy: 0.7684 - val_loss: 0.6067 - val_accuracy: 0.7392
Epoch 16/50
5002/5002 [==============================] - 73s 15ms/step - loss: 0.5372 -
accuracy: 0.7602 - val_loss: 0.5660 - val_accuracy: 0.7889
Epoch 17/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.5610 -
accuracy: 0.7527 - val_loss: 0.6229 - val_accuracy: 0.6496
Epoch 18/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.4945 -
accuracy: 0.7833 - val_loss: 0.5784 - val_accuracy: 0.7846
Epoch 19/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.4882 -
accuracy: 0.7839 - val_loss: 0.4757 - val_accuracy: 0.7959

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
```

```
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 20/50
5002/5002 [==============================] - 76s 15ms/step - loss: 0.5036 -
accuracy: 0.7729 - val_loss: 0.5851 - val_accuracy: 0.7197
Epoch 21/50
5002/5002 [==============================] - 76s 15ms/step - loss: 0.4819 -
accuracy: 0.7845 - val_loss: 0.7252 - val_accuracy: 0.5314
Epoch 22/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.4746 -
accuracy: 0.7904 - val_loss: 0.4315 - val_accuracy: 0.8056

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 23/50
5002/5002 [==============================] - 76s 15ms/step - loss: 0.4974 -
accuracy: 0.7739 - val_loss: 0.4980 - val_accuracy: 0.7819
Epoch 24/50
5002/5002 [==============================] - 76s 15ms/step - loss: 0.4542 -
accuracy: 0.8013 - val_loss: 0.4638 - val_accuracy: 0.8077

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 25/50
5002/5002 [==============================] - 75s 15ms/step - loss: 0.4648 -
accuracy: 0.7981 - val_loss: 0.4592 - val_accuracy: 0.8018
Epoch 26/50
5002/5002 [==============================] - 75s 15ms/step - loss: 0.4753 -
accuracy: 0.7936 - val_loss: 0.4825 - val_accuracy: 0.7974
Epoch 27/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.4500 -
accuracy: 0.8012 - val_loss: 0.4657 - val_accuracy: 0.8081
```

```
WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 28/50
5002/5002 [==============================] - 75s 15ms/step - loss: 0.5015 -
accuracy: 0.7566 - val_loss: 0.5009 - val_accuracy: 0.7766
Epoch 29/50
5002/5002 [==============================] - 74s 15ms/step - loss: 0.4696 -
accuracy: 0.7907 - val_loss: 0.4305 - val_accuracy: 0.8106

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 30/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.4749 -
accuracy: 0.7824 - val_loss: 0.4786 - val_accuracy: 0.8017
Epoch 31/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.4462 -
accuracy: 0.8015 - val_loss: 0.4382 - val_accuracy: 0.8123

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 32/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.4437 -
accuracy: 0.8038 - val_loss: 0.4582 - val_accuracy: 0.7931
Epoch 33/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.4832 -
accuracy: 0.7669 - val_loss: 0.4595 - val_accuracy: 0.7943
Epoch 34/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.4321 -
accuracy: 0.8096 - val_loss: 0.4269 - val_accuracy: 0.8139
```

```
WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 35/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.4333 -
accuracy: 0.8119 - val_loss: 0.4246 - val_accuracy: 0.8156

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 36/50
5002/5002 [==============================] - 76s 15ms/step - loss: 0.4263 -
accuracy: 0.8132 - val_loss: 0.4301 - val_accuracy: 0.8145
Epoch 37/50
5002/5002 [==============================] - 76s 15ms/step - loss: 0.4396 -
accuracy: 0.8000 - val_loss: 0.4776 - val_accuracy: 0.7922
Epoch 38/50
5002/5002 [==============================] - 76s 15ms/step - loss: 0.4991 -
accuracy: 0.7468 - val_loss: 0.4937 - val_accuracy: 0.7492
Epoch 39/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.4391 -
accuracy: 0.8028 - val_loss: 0.4311 - val_accuracy: 0.8133
Epoch 40/50
5002/5002 [==============================] - 80s 16ms/step - loss: 0.4091 -
accuracy: 0.8230 - val_loss: 0.4238 - val_accuracy: 0.8149
Epoch 41/50
5002/5002 [==============================] - 78s 16ms/step - loss: 0.4150 -
accuracy: 0.8202 - val_loss: 0.4241 - val_accuracy: 0.8166

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets
```

```
INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 42/50
5002/5002 [==============================] - 79s 16ms/step - loss: 0.4154 -
accuracy: 0.8177 - val_loss: 0.4426 - val_accuracy: 0.8143
Epoch 43/50
5002/5002 [==============================] - 78s 16ms/step - loss: 0.4098 -
accuracy: 0.8212 - val_loss: 0.4228 - val_accuracy: 0.8203

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 44/50
5002/5002 [==============================] - 78s 16ms/step - loss: 0.4029 -
accuracy: 0.8265 - val_loss: 0.4101 - val_accuracy: 0.8197
Epoch 45/50
5002/5002 [==============================] - 78s 16ms/step - loss: 0.4013 -
accuracy: 0.8270 - val_loss: 0.4134 - val_accuracy: 0.8212

WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses,
gru_cell_layer_call_and_return_conditional_losses while saving (showing 5 of 5).
These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_13\assets

INFO:tensorflow:Assets written to: models\model_13\assets

Epoch 46/50
5002/5002 [==============================] - 78s 16ms/step - loss: 0.3984 -
accuracy: 0.8305 - val_loss: 0.4410 - val_accuracy: 0.8170
Epoch 47/50
5002/5002 [==============================] - 78s 16ms/step - loss: 0.4026 -
accuracy: 0.8295 - val_loss: 0.4308 - val_accuracy: 0.8196
Epoch 48/50
5002/5002 [==============================] - 78s 16ms/step - loss: 0.3918 -
accuracy: 0.8317 - val_loss: 0.4107 - val_accuracy: 0.8204
Epoch 49/50
5002/5002 [==============================] - 80s 16ms/step - loss: 0.3923 -
accuracy: 0.8327 - val_loss: 0.4093 - val_accuracy: 0.8202
Epoch 50/50
5002/5002 [==============================] - 80s 16ms/step - loss: 0.4056 -
accuracy: 0.8284 - val_loss: 0.4232 - val_accuracy: 0.8211
```

[ ]: <keras.callbacks.History at 0x26cf3fb3e88>

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_13,'model_13',X_test,np.array(y_test))
```

```
1249/1249 [==============================] - 9s 7ms/step - loss: 0.4134 -
accuracy: 0.8212
Loss = 0.4133811593055725 Accuracy = 0.8212140798568726
```

5.2.2 Google Word2Vec: Binary classification

- For binary classification, I selected reviews with label 0 (negative) and 1 (positive). Then concatenated the dataframes to a single dataframe using pandas.concat() method.
- Created default vector using the np.zeros() function in case a word is missing from the google word2vec vocabulary.
- Then retrieved the tensor for each review by concatenating the google word2vec vector for the first 20 words in the review. The tensor shape for 1 review comes out to be (20,300).
- Although the assignment specified us to take the first 50 words, I took the first 20 words as suggested by Prof. Rostami due to lack of computational resources.

[ ]:
```
new_train_df = pd.concat([train_df[train_df['label'] ==␣
 ↪0],train_df[train_df['label'] == 1]])

new_test_df = pd.concat([test_df[test_df['label'] ==␣
 ↪0],test_df[test_df['label'] == 1]])


DEFAULT_VECTOR = np.zeros((300,))

X_train = np.array([[google_model[word] if word in google_model else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
 ↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
 ↪new_train_df['reviews'].tolist()])
X_test = np.array([[google_model[word] if word in google_model else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
 ↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
 ↪new_test_df['reviews'].tolist()])

y_train = new_train_df['label'].tolist()
y_test = new_test_df['label'].tolist()
```

- Below, I described the GRU model consisting of one GRU layer with 50 neurons for the binary classification task. By default, the GRU's return_sequences parameter is False, therefore it only returns a vector of size 50 at the last timestep.

- I used the tanh non-linearity in the output layer with 1 neuron which gives me an output in the range [-1,1], therefore for outputs <0, we predict class 0 (negative) and for outputs >0, we predict class 1 (positive).
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_14 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((20,300)),
    tf.keras.layers.GRU(50),
    tf.keras.layers.Dense(1,activation='tanh')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_14', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)


model_14.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='binary_crossentropy',metrics=['accuracy'])
model_14.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_1 (GRU)                  (None, 50)                52800
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 51
=================================================================
Total params: 52,851
Trainable params: 52,851
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the GRU for 50 epochs and passed the model checkpointing callback to the callbacks parameter.

- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_14.fit(np.array(X_train),np.array(y_train),validation_data=(np.
↪array(X_test),np.
↪array(y_test)),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
5002/5002 [==============================] - 101s 19ms/step - loss: 0.4937 -
accuracy: 0.7781 - val_loss: 0.4355 - val_accuracy: 0.8170

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_14\assets

INFO:tensorflow:Assets written to: models\model_14\assets

Epoch 2/50
5002/5002 [==============================] - 82s 16ms/step - loss: 0.4368 -
accuracy: 0.8110 - val_loss: 0.3981 - val_accuracy: 0.8353

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_14\assets

INFO:tensorflow:Assets written to: models\model_14\assets

Epoch 3/50
5002/5002 [==============================] - 81s 16ms/step - loss: 0.4130 -
accuracy: 0.8262 - val_loss: 0.4173 - val_accuracy: 0.8305
Epoch 4/50
5002/5002 [==============================] - 82s 16ms/step - loss: 0.4100 -
accuracy: 0.8294 - val_loss: 0.4037 - val_accuracy: 0.8372

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_14\assets
```

```
INFO:tensorflow:Assets written to: models\model_14\assets

Epoch 5/50
5002/5002 [==============================] - 83s 17ms/step - loss: 0.3958 -
accuracy: 0.8371 - val_loss: 0.4090 - val_accuracy: 0.8336
Epoch 6/50
5002/5002 [==============================] - 83s 17ms/step - loss: 0.4108 -
accuracy: 0.8321 - val_loss: 0.4041 - val_accuracy: 0.8209
Epoch 7/50
5002/5002 [==============================] - 84s 17ms/step - loss: 0.3846 -
accuracy: 0.8453 - val_loss: 0.4598 - val_accuracy: 0.8368
Epoch 8/50
5002/5002 [==============================] - 83s 17ms/step - loss: 0.3943 -
accuracy: 0.8396 - val_loss: 0.4191 - val_accuracy: 0.8379

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_14\assets

INFO:tensorflow:Assets written to: models\model_14\assets

Epoch 9/50
5002/5002 [==============================] - 86s 17ms/step - loss: 0.3833 -
accuracy: 0.8463 - val_loss: 0.4202 - val_accuracy: 0.8172
Epoch 10/50
5002/5002 [==============================] - 88s 17ms/step - loss: 0.3918 -
accuracy: 0.8424 - val_loss: 0.4012 - val_accuracy: 0.8431

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_14\assets

INFO:tensorflow:Assets written to: models\model_14\assets

Epoch 11/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.3825 -
accuracy: 0.8422 - val_loss: 0.3983 - val_accuracy: 0.8361
Epoch 12/50
5002/5002 [==============================] - 79s 16ms/step - loss: 0.3673 -
accuracy: 0.8501 - val_loss: 0.4214 - val_accuracy: 0.8169
Epoch 13/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.3523 -
accuracy: 0.8614 - val_loss: 0.4286 - val_accuracy: 0.8482
```

```
WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_14\assets

INFO:tensorflow:Assets written to: models\model_14\assets

Epoch 14/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.3611 -
accuracy: 0.8587 - val_loss: 0.4089 - val_accuracy: 0.8470
Epoch 15/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.4121 -
accuracy: 0.8186 - val_loss: 0.4809 - val_accuracy: 0.7746
Epoch 16/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.4015 -
accuracy: 0.8340 - val_loss: 0.4235 - val_accuracy: 0.8406
Epoch 17/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.3597 -
accuracy: 0.8536 - val_loss: 0.4040 - val_accuracy: 0.8446
Epoch 18/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.3502 -
accuracy: 0.8583 - val_loss: 0.4263 - val_accuracy: 0.8460
Epoch 19/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.3442 -
accuracy: 0.8636 - val_loss: 0.4640 - val_accuracy: 0.8472
Epoch 20/50
5002/5002 [==============================] - 78s 16ms/step - loss: 0.3675 -
accuracy: 0.8515 - val_loss: 0.4033 - val_accuracy: 0.8309
Epoch 21/50
5002/5002 [==============================] - 76s 15ms/step - loss: 0.3548 -
accuracy: 0.8587 - val_loss: 0.4621 - val_accuracy: 0.8370
Epoch 22/50
5002/5002 [==============================] - 77s 15ms/step - loss: 0.3357 -
accuracy: 0.8687 - val_loss: 0.4177 - val_accuracy: 0.8476
Epoch 23/50
5002/5002 [==============================] - 78s 16ms/step - loss: 0.3306 -
accuracy: 0.8704 - val_loss: 0.4400 - val_accuracy: 0.8475
Epoch 24/50
5002/5002 [==============================] - 79s 16ms/step - loss: 0.3308 -
accuracy: 0.8687 - val_loss: 0.4404 - val_accuracy: 0.8486

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: models\model_14\assets

INFO:tensorflow:Assets written to: models\model_14\assets

Epoch 25/50
5002/5002 [==============================] - 81s 16ms/step - loss: 0.3377 -
accuracy: 0.8644 - val_loss: 0.4419 - val_accuracy: 0.8463
Epoch 26/50
5002/5002 [==============================] - 81s 16ms/step - loss: 0.3140 -
accuracy: 0.8766 - val_loss: 0.4153 - val_accuracy: 0.8484
Epoch 27/50
5002/5002 [==============================] - 81s 16ms/step - loss: 0.3158 -
accuracy: 0.8771 - val_loss: 0.4960 - val_accuracy: 0.8486
Epoch 28/50
5002/5002 [==============================] - 81s 16ms/step - loss: 0.3318 -
accuracy: 0.8682 - val_loss: 0.4761 - val_accuracy: 0.8427
Epoch 29/50
5002/5002 [==============================] - 80s 16ms/step - loss: 0.3355 -
accuracy: 0.8681 - val_loss: 0.4999 - val_accuracy: 0.8415
Epoch 30/50
5002/5002 [==============================] - 81s 16ms/step - loss: 0.3196 -
accuracy: 0.8738 - val_loss: 0.4895 - val_accuracy: 0.8465
Epoch 31/50
5002/5002 [==============================] - 82s 16ms/step - loss: 0.3023 -
accuracy: 0.8826 - val_loss: 0.4383 - val_accuracy: 0.8447
Epoch 32/50
5002/5002 [==============================] - 81s 16ms/step - loss: 0.2985 -
accuracy: 0.8834 - val_loss: 0.5086 - val_accuracy: 0.8462
Epoch 33/50
5002/5002 [==============================] - 81s 16ms/step - loss: 0.2935 -
accuracy: 0.8859 - val_loss: 0.5102 - val_accuracy: 0.8486

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_14\assets

INFO:tensorflow:Assets written to: models\model_14\assets

Epoch 34/50
5002/5002 [==============================] - 90s 18ms/step - loss: 0.2857 -
accuracy: 0.8893 - val_loss: 0.5188 - val_accuracy: 0.8493

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: models\model_14\assets

INFO:tensorflow:Assets written to: models\model_14\assets

Epoch 35/50
5002/5002 [==============================] - 91s 18ms/step - loss: 0.2830 -
accuracy: 0.8899 - val_loss: 0.4875 - val_accuracy: 0.8472
Epoch 36/50
5002/5002 [==============================] - 93s 18ms/step - loss: 0.2757 -
accuracy: 0.8935 - val_loss: 0.5141 - val_accuracy: 0.8497

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_14\assets

INFO:tensorflow:Assets written to: models\model_14\assets

Epoch 37/50
5002/5002 [==============================] - 94s 19ms/step - loss: 0.2776 -
accuracy: 0.8917 - val_loss: 0.5088 - val_accuracy: 0.8395
Epoch 38/50
5002/5002 [==============================] - 94s 19ms/step - loss: 0.2722 -
accuracy: 0.8953 - val_loss: 0.5678 - val_accuracy: 0.8491
Epoch 39/50
5002/5002 [==============================] - 94s 19ms/step - loss: 0.2714 -
accuracy: 0.8937 - val_loss: 0.5253 - val_accuracy: 0.8472
Epoch 40/50
5002/5002 [==============================] - 95s 19ms/step - loss: 0.2651 -
accuracy: 0.8977 - val_loss: 0.5335 - val_accuracy: 0.8471
Epoch 41/50
5002/5002 [==============================] - 94s 19ms/step - loss: 0.2613 -
accuracy: 0.8994 - val_loss: 0.5686 - val_accuracy: 0.8469
Epoch 42/50
5002/5002 [==============================] - 95s 19ms/step - loss: 0.2582 -
accuracy: 0.9012 - val_loss: 0.6107 - val_accuracy: 0.8479
Epoch 43/50
5002/5002 [==============================] - 93s 19ms/step - loss: 0.2573 -
accuracy: 0.9016 - val_loss: 0.5649 - val_accuracy: 0.8469
Epoch 44/50
5002/5002 [==============================] - 95s 19ms/step - loss: 0.2547 -
accuracy: 0.9028 - val_loss: 0.5744 - val_accuracy: 0.8455
Epoch 45/50
5002/5002 [==============================] - 96s 19ms/step - loss: 0.2479 -
accuracy: 0.9050 - val_loss: 0.6125 - val_accuracy: 0.8478
Epoch 46/50
```

```
5002/5002 [==============================] - 97s 19ms/step - loss: 0.2413 -
accuracy: 0.9082 - val_loss: 0.6163 - val_accuracy: 0.8468
Epoch 47/50
5002/5002 [==============================] - 95s 19ms/step - loss: 0.2467 -
accuracy: 0.9059 - val_loss: 0.6097 - val_accuracy: 0.8460
Epoch 48/50
5002/5002 [==============================] - 120s 24ms/step - loss: 0.2366 -
accuracy: 0.9096 - val_loss: 0.6088 - val_accuracy: 0.8471
Epoch 49/50
5002/5002 [==============================] - 153s 31ms/step - loss: 0.2350 -
accuracy: 0.9107 - val_loss: 0.6241 - val_accuracy: 0.8454
Epoch 50/50
5002/5002 [==============================] - 155s 31ms/step - loss: 0.2314 -
accuracy: 0.9121 - val_loss: 0.6554 - val_accuracy: 0.8466
```

[ ]: <keras.callbacks.History at 0x26cf0fbf688>

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

[ ]:
```
load_and_test_model_weights(model_14,'model_14',X_test,np.array(y_test))
```

```
1249/1249 [==============================] - 17s 14ms/step - loss: 0.5141 -
accuracy: 0.8497
Loss = 0.5141297578811646 Accuracy = 0.8497397899627686
```

5.2.3 Word2Vec: Ternary classification

- For ternary classification, I created default vector using the np.zeros() function in case a word is missing from the word2vec vocabulary.
- Then retrieved the tensor for each review by concatenating the word2vec vector for the first 20 words in the review. The tensor shape for 1 review comes out to be (20,300).
- Although the assignment specified us to take the first 50 words, I took the first 20 words as suggested by Prof. Rostami due to lack of computational resources.

[ ]:
```
DEFAULT_VECTOR = np.zeros((300,))

X_train = np.array([[w2v_model.wv[word] if word in w2v_model.wv else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
 ↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
 ↪train_df['reviews'].tolist()])
X_test = np.array([[w2v_model.wv[word] if word in w2v_model.wv else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
 ↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
 ↪test_df['reviews'].tolist()])

y_train = train_df['label'].tolist()
```

```
y_test = test_df['label'].tolist()
```

- Below, I described the GRU model consisting of one GRU layer with 50 neurons for the ternary classification task. By default, the GRU's return_sequences parameter is False, therefore it only returns a vector of size 50 at the last timestep.
- I used the softmax non-linearity in the output layer with 3 neurons which gives an output probability distribution with the probability for each class, therefore the final predicted output is the class with maximum probability.
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_15 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((20,300)),
    tf.keras.layers.GRU(50),
    tf.keras.layers.Dense(3,activation='softmax')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_15', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)


model_15.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='sparse_categorical_crossentropy',metrics=['accuracy']
model_15.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_3 (GRU)                  (None, 50)                52800
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 153
=================================================================
Total params: 52,953
```

```
Trainable params: 52,953
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the GRU for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_15.fit(np.array(X_train),np.array(y_train),validation_data=(np.
↪array(X_test),np.
↪array(y_test)),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
6250/6250 [==============================] - 109s 17ms/step - loss: 0.7897 -
accuracy: 0.6575 - val_loss: 0.7864 - val_accuracy: 0.6621

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 2/50
6250/6250 [==============================] - 86s 14ms/step - loss: 0.7693 -
accuracy: 0.6678 - val_loss: 0.7766 - val_accuracy: 0.6655

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 3/50
6250/6250 [==============================] - 88s 14ms/step - loss: 0.7626 -
accuracy: 0.6709 - val_loss: 0.7702 - val_accuracy: 0.6682

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 4/50
6250/6250 [==============================] - 92s 15ms/step - loss: 0.7611 -
accuracy: 0.6720 - val_loss: 0.7689 - val_accuracy: 0.6664
Epoch 5/50
6250/6250 [==============================] - 90s 14ms/step - loss: 0.7585 -
accuracy: 0.6718 - val_loss: 0.7707 - val_accuracy: 0.6644
Epoch 6/50
6250/6250 [==============================] - 90s 14ms/step - loss: 0.7527 -
accuracy: 0.6751 - val_loss: 0.7667 - val_accuracy: 0.6668
Epoch 7/50
6250/6250 [==============================] - 92s 15ms/step - loss: 0.7478 -
accuracy: 0.6779 - val_loss: 0.7636 - val_accuracy: 0.6706

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 8/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.7469 -
accuracy: 0.6782 - val_loss: 0.7705 - val_accuracy: 0.6681
Epoch 9/50
6250/6250 [==============================] - 94s 15ms/step - loss: 0.7450 -
accuracy: 0.6793 - val_loss: 0.7608 - val_accuracy: 0.6707

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 10/50
6250/6250 [==============================] - 94s 15ms/step - loss: 0.7423 -
accuracy: 0.6812 - val_loss: 0.7600 - val_accuracy: 0.6722
```

```
WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 11/50
6250/6250 [==============================] - 101s 16ms/step - loss: 0.7386 -
accuracy: 0.6831 - val_loss: 0.7602 - val_accuracy: 0.6727

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 12/50
6250/6250 [==============================] - 101s 16ms/step - loss: 0.7373 -
accuracy: 0.6837 - val_loss: 0.7604 - val_accuracy: 0.6716
Epoch 13/50
6250/6250 [==============================] - 98s 16ms/step - loss: 0.7357 -
accuracy: 0.6842 - val_loss: 0.7568 - val_accuracy: 0.6739

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 14/50
6250/6250 [==============================] - 106s 17ms/step - loss: 0.7328 -
accuracy: 0.6847 - val_loss: 0.7567 - val_accuracy: 0.6752

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 15/50
6250/6250 [==============================] - 98s 16ms/step - loss: 0.7295 -
accuracy: 0.6868 - val_loss: 0.7574 - val_accuracy: 0.6734
Epoch 16/50
6250/6250 [==============================] - 93s 15ms/step - loss: 0.7278 -
accuracy: 0.6878 - val_loss: 0.7573 - val_accuracy: 0.6719
Epoch 17/50
6250/6250 [==============================] - 94s 15ms/step - loss: 0.7260 -
accuracy: 0.6883 - val_loss: 0.7548 - val_accuracy: 0.6748
Epoch 18/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.7235 -
accuracy: 0.6895 - val_loss: 0.7562 - val_accuracy: 0.6763

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 19/50
6250/6250 [==============================] - 94s 15ms/step - loss: 0.7219 -
accuracy: 0.6907 - val_loss: 0.7511 - val_accuracy: 0.6774

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 20/50
6250/6250 [==============================] - 98s 16ms/step - loss: 0.7205 -
accuracy: 0.6915 - val_loss: 0.7504 - val_accuracy: 0.6766
Epoch 21/50
6250/6250 [==============================] - 99s 16ms/step - loss: 0.7165 -
accuracy: 0.6934 - val_loss: 0.7528 - val_accuracy: 0.6757
Epoch 22/50
6250/6250 [==============================] - 100s 16ms/step - loss: 0.7152 -
```

```
accuracy: 0.6946 - val_loss: 0.7504 - val_accuracy: 0.6772
Epoch 23/50
6250/6250 [==============================] - 100s 16ms/step - loss: 0.7127 -
accuracy: 0.6964 - val_loss: 0.7499 - val_accuracy: 0.6766
Epoch 24/50
6250/6250 [==============================] - 98s 16ms/step - loss: 0.7102 -
accuracy: 0.6967 - val_loss: 0.7498 - val_accuracy: 0.6786

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 25/50
6250/6250 [==============================] - 99s 16ms/step - loss: 0.7084 -
accuracy: 0.6967 - val_loss: 0.7479 - val_accuracy: 0.6804

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 26/50
6250/6250 [==============================] - 97s 16ms/step - loss: 0.7054 -
accuracy: 0.6981 - val_loss: 0.7499 - val_accuracy: 0.6782
Epoch 27/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.7038 -
accuracy: 0.6991 - val_loss: 0.7499 - val_accuracy: 0.6796
Epoch 28/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.7019 -
accuracy: 0.6999 - val_loss: 0.7494 - val_accuracy: 0.6776
Epoch 29/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.6994 -
accuracy: 0.7007 - val_loss: 0.7446 - val_accuracy: 0.6792
Epoch 30/50
6250/6250 [==============================] - 92s 15ms/step - loss: 0.6976 -
accuracy: 0.7020 - val_loss: 0.7477 - val_accuracy: 0.6782
Epoch 31/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.6952 -
```

```
accuracy: 0.7031 - val_loss: 0.7458 - val_accuracy: 0.6781
Epoch 32/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.6937 -
accuracy: 0.7034 - val_loss: 0.7473 - val_accuracy: 0.6798
Epoch 33/50
6250/6250 [==============================] - 97s 15ms/step - loss: 0.6904 -
accuracy: 0.7050 - val_loss: 0.7432 - val_accuracy: 0.6805

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 34/50
6250/6250 [==============================] - 97s 16ms/step - loss: 0.6888 -
accuracy: 0.7060 - val_loss: 0.7446 - val_accuracy: 0.6809

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 35/50
6250/6250 [==============================] - 98s 16ms/step - loss: 0.6871 -
accuracy: 0.7060 - val_loss: 0.7460 - val_accuracy: 0.6800
Epoch 36/50
6250/6250 [==============================] - 98s 16ms/step - loss: 0.6854 -
accuracy: 0.7079 - val_loss: 0.7443 - val_accuracy: 0.6816

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets
```

```
Epoch 37/50
6250/6250 [==============================] - 100s 16ms/step - loss: 0.6833 -
accuracy: 0.7084 - val_loss: 0.7458 - val_accuracy: 0.6825

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets

INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 38/50
6250/6250 [==============================] - 94s 15ms/step - loss: 0.6811 -
accuracy: 0.7093 - val_loss: 0.7447 - val_accuracy: 0.6818
Epoch 39/50
6250/6250 [==============================] - 93s 15ms/step - loss: 0.6790 -
accuracy: 0.7106 - val_loss: 0.7435 - val_accuracy: 0.6791
Epoch 40/50
6250/6250 [==============================] - 93s 15ms/step - loss: 0.6767 -
accuracy: 0.7118 - val_loss: 0.7446 - val_accuracy: 0.6809
Epoch 41/50
6250/6250 [==============================] - 96s 15ms/step - loss: 0.6752 -
accuracy: 0.7119 - val_loss: 0.7438 - val_accuracy: 0.6799
Epoch 42/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.6732 -
accuracy: 0.7134 - val_loss: 0.7486 - val_accuracy: 0.6794
Epoch 43/50
6250/6250 [==============================] - 94s 15ms/step - loss: 0.6709 -
accuracy: 0.7147 - val_loss: 0.7467 - val_accuracy: 0.6814
Epoch 44/50
6250/6250 [==============================] - 93s 15ms/step - loss: 0.6693 -
accuracy: 0.7154 - val_loss: 0.7437 - val_accuracy: 0.6811
Epoch 45/50
6250/6250 [==============================] - 93s 15ms/step - loss: 0.6674 -
accuracy: 0.7161 - val_loss: 0.7454 - val_accuracy: 0.6805
Epoch 46/50
6250/6250 [==============================] - 94s 15ms/step - loss: 0.6656 -
accuracy: 0.7170 - val_loss: 0.7454 - val_accuracy: 0.6827

WARNING:absl:Found untraced functions such as gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_15\assets
```

```
INFO:tensorflow:Assets written to: models\model_15\assets

Epoch 47/50
6250/6250 [==============================] - 97s 16ms/step - loss: 0.6637 -
accuracy: 0.7178 - val_loss: 0.7465 - val_accuracy: 0.6815
Epoch 48/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.6618 -
accuracy: 0.7188 - val_loss: 0.7463 - val_accuracy: 0.6812
Epoch 49/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.6603 -
accuracy: 0.7200 - val_loss: 0.7478 - val_accuracy: 0.6815
Epoch 50/50
6250/6250 [==============================] - 95s 15ms/step - loss: 0.6588 -
accuracy: 0.7200 - val_loss: 0.7477 - val_accuracy: 0.6823
```

```
[ ]: <keras.callbacks.History at 0x207759fc6c8>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

```
[ ]: load_and_test_model_weights(model_15,'model_15',X_test,np.array(y_test))
```

```
1563/1563 [==============================] - 21s 11ms/step - loss: 0.7454 -
accuracy: 0.6827
Loss = 0.7454344630241394 Accuracy = 0.682699978351593
```

5.2.4 Google Word2Vec: Ternary classification

- For ternary classification, I created default vector using the np.zeros() function in case a word is missing from the google word2vec vocabulary.
- Then retrieved the tensor for each review by concatenating the google word2vec vector for the first 20 words in the review. The tensor shape for 1 review comes out to be (20,300).
- Although the assignment specified us to take the first 50 words, I took the first 20 words as suggested by Prof. Rostami due to lack of computational resources.

```
[ ]: DEFAULT_VECTOR = np.zeros((300,))

X_train = np.array([[google_model[word] if word in google_model else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
 ↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
 ↪train_df['reviews'].tolist()])
X_test = np.array([[google_model[word] if word in google_model else␣
 ↪DEFAULT_VECTOR for word in sentence.split(' ')[:20] +␣
 ↪['<oov>']*(20-min(20,len(sentence.split(' '))))] for sentence in␣
 ↪test_df['reviews'].tolist()])

y_train = train_df['label'].tolist()
```

```
y_test = test_df['label'].tolist()
```

- Below, I described the GRU model consisting of one GRU layer with 50 neurons for the ternary classification task. By default, the GRU's return_sequences parameter is False, therefore it only returns a vector of size 50 at the last timestep.
- I used the softmax non-linearity in the output layer with 3 neurons which gives an output probability distribution with the probability for each class, therefore the final predicted output is the class with maximum probability.
- Furthermore, I used an exponential decay learning rate schedule to decrease my learning rate (initially 0.005), by a factor of 0.96 at every 5000 steps.
- Also, I defined a model checkpointing callback to save the best model based on validation accuracy, which checks model performance after each epoch.

[ ]:
```
model_16 = tf.keras.Sequential([
    tf.keras.layers.InputLayer((20,300)),
    tf.keras.layers.GRU(50),
    tf.keras.layers.Dense(3,activation='softmax')
])

initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps = 5000,
                decay_rate = 0.96,
                staircase = True
                )

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    'models/model_16', monitor='val_accuracy', verbose=0, save_best_only=True,
    save_weights_only=False, mode='auto', save_freq='epoch',
)


model_16.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=lr_schedule),loss='sparse_categorical_crossentropy',metrics=['accuracy']
model_16.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_2 (GRU)                  (None, 50)                52800
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 153
=================================================================
Total params: 52,953
```

```
Trainable params: 52,953
Non-trainable params: 0
_________________________________________________________________
```

- I used the fit function to train the GRU for 50 epochs and passed the model checkpointing callback to the callbacks parameter.
- The model trained for 50 epochs with a batch size of 32 while continuously reporting the train/validation loss as well as train/validation accuracy.

[ ]:
```
model_16.fit(np.array(X_train),np.array(y_train),validation_data=(np.
↪array(X_test),np.
↪array(y_test)),batch_size=32,epochs=50,callbacks=[checkpointer])
```

```
Epoch 1/50
6250/6250 [==============================] - 71s 11ms/step - loss: 0.7462 -
accuracy: 0.6773 - val_loss: 0.7169 - val_accuracy: 0.6902

WARNING:absl:Found untraced functions such as gru_cell_2_layer_call_fn,
gru_cell_2_layer_call_and_return_conditional_losses, gru_cell_2_layer_call_fn,
gru_cell_2_layer_call_and_return_conditional_losses,
gru_cell_2_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_16\assets

INFO:tensorflow:Assets written to: models\model_16\assets

Epoch 2/50
6250/6250 [==============================] - 47s 8ms/step - loss: 0.6957 -
accuracy: 0.7021 - val_loss: 0.7141 - val_accuracy: 0.6930

WARNING:absl:Found untraced functions such as gru_cell_2_layer_call_fn,
gru_cell_2_layer_call_and_return_conditional_losses, gru_cell_2_layer_call_fn,
gru_cell_2_layer_call_and_return_conditional_losses,
gru_cell_2_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_16\assets

INFO:tensorflow:Assets written to: models\model_16\assets

Epoch 3/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.6756 -
accuracy: 0.7120 - val_loss: 0.7116 - val_accuracy: 0.6948

WARNING:absl:Found untraced functions such as gru_cell_2_layer_call_fn,
gru_cell_2_layer_call_and_return_conditional_losses, gru_cell_2_layer_call_fn,
gru_cell_2_layer_call_and_return_conditional_losses,
gru_cell_2_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: models\model_16\assets

INFO:tensorflow:Assets written to: models\model_16\assets

Epoch 4/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.6599 -
accuracy: 0.7190 - val_loss: 0.7092 - val_accuracy: 0.6949

WARNING:absl:Found untraced functions such as gru_cell_2_layer_call_fn,
gru_cell_2_layer_call_and_return_conditional_losses, gru_cell_2_layer_call_fn,
gru_cell_2_layer_call_and_return_conditional_losses,
gru_cell_2_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_16\assets

INFO:tensorflow:Assets written to: models\model_16\assets

Epoch 5/50
6250/6250 [==============================] - 46s 7ms/step - loss: 0.6466 -
accuracy: 0.7263 - val_loss: 0.7158 - val_accuracy: 0.6964

WARNING:absl:Found untraced functions such as gru_cell_2_layer_call_fn,
gru_cell_2_layer_call_and_return_conditional_losses, gru_cell_2_layer_call_fn,
gru_cell_2_layer_call_and_return_conditional_losses,
gru_cell_2_layer_call_and_return_conditional_losses while saving (showing 5 of
5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\model_16\assets

INFO:tensorflow:Assets written to: models\model_16\assets

Epoch 6/50
6250/6250 [==============================] - 48s 8ms/step - loss: 0.6344 -
accuracy: 0.7309 - val_loss: 0.7215 - val_accuracy: 0.6909
Epoch 7/50
6250/6250 [==============================] - 49s 8ms/step - loss: 0.6240 -
accuracy: 0.7357 - val_loss: 0.7269 - val_accuracy: 0.6927
Epoch 8/50
6250/6250 [==============================] - 49s 8ms/step - loss: 0.6144 -
accuracy: 0.7403 - val_loss: 0.7301 - val_accuracy: 0.6933
Epoch 9/50
6250/6250 [==============================] - 49s 8ms/step - loss: 0.6042 -
accuracy: 0.7453 - val_loss: 0.7292 - val_accuracy: 0.6916
Epoch 10/50
6250/6250 [==============================] - 49s 8ms/step - loss: 0.5958 -
accuracy: 0.7490 - val_loss: 0.7429 - val_accuracy: 0.6925
Epoch 11/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.5877 -
```

```
accuracy: 0.7532 - val_loss: 0.7392 - val_accuracy: 0.6882
Epoch 12/50
6250/6250 [==============================] - 49s 8ms/step - loss: 0.5791 -
accuracy: 0.7567 - val_loss: 0.7404 - val_accuracy: 0.6893
Epoch 13/50
6250/6250 [==============================] - 51s 8ms/step - loss: 0.5705 -
accuracy: 0.7609 - val_loss: 0.7573 - val_accuracy: 0.6881
Epoch 14/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.5637 -
accuracy: 0.7642 - val_loss: 0.7635 - val_accuracy: 0.6890
Epoch 15/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.5562 -
accuracy: 0.7676 - val_loss: 0.7679 - val_accuracy: 0.6848
Epoch 16/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.5498 -
accuracy: 0.7702 - val_loss: 0.7739 - val_accuracy: 0.6867
Epoch 17/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.5426 -
accuracy: 0.7736 - val_loss: 0.7754 - val_accuracy: 0.6859
Epoch 18/50
6250/6250 [==============================] - 50s 8ms/step - loss: 0.5362 -
accuracy: 0.7766 - val_loss: 0.7900 - val_accuracy: 0.6860
Epoch 19/50
6250/6250 [==============================] - 51s 8ms/step - loss: 0.5297 -
accuracy: 0.7799 - val_loss: 0.7881 - val_accuracy: 0.6860
Epoch 20/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.5235 -
accuracy: 0.7834 - val_loss: 0.8004 - val_accuracy: 0.6818
Epoch 21/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.5178 -
accuracy: 0.7854 - val_loss: 0.8031 - val_accuracy: 0.6824
Epoch 22/50
6250/6250 [==============================] - 51s 8ms/step - loss: 0.5112 -
accuracy: 0.7892 - val_loss: 0.8171 - val_accuracy: 0.6830
Epoch 23/50
6250/6250 [==============================] - 51s 8ms/step - loss: 0.5064 -
accuracy: 0.7910 - val_loss: 0.8287 - val_accuracy: 0.6825
Epoch 24/50
6250/6250 [==============================] - 51s 8ms/step - loss: 0.5003 -
accuracy: 0.7933 - val_loss: 0.8273 - val_accuracy: 0.6823
Epoch 25/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4950 -
accuracy: 0.7970 - val_loss: 0.8360 - val_accuracy: 0.6829
Epoch 26/50
6250/6250 [==============================] - 51s 8ms/step - loss: 0.4901 -
accuracy: 0.7990 - val_loss: 0.8432 - val_accuracy: 0.6797
Epoch 27/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4848 -
```

```
accuracy: 0.8020 - val_loss: 0.8540 - val_accuracy: 0.6790
Epoch 28/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4801 -
accuracy: 0.8037 - val_loss: 0.8638 - val_accuracy: 0.6805
Epoch 29/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4754 -
accuracy: 0.8059 - val_loss: 0.8668 - val_accuracy: 0.6783
Epoch 30/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4706 -
accuracy: 0.8084 - val_loss: 0.8722 - val_accuracy: 0.6782
Epoch 31/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4667 -
accuracy: 0.8103 - val_loss: 0.8773 - val_accuracy: 0.6802
Epoch 32/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4618 -
accuracy: 0.8126 - val_loss: 0.8920 - val_accuracy: 0.6800
Epoch 33/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4579 -
accuracy: 0.8147 - val_loss: 0.9016 - val_accuracy: 0.6780
Epoch 34/50
6250/6250 [==============================] - 53s 9ms/step - loss: 0.4540 -
accuracy: 0.8164 - val_loss: 0.9022 - val_accuracy: 0.6765
Epoch 35/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4508 -
accuracy: 0.8179 - val_loss: 0.9073 - val_accuracy: 0.6776
Epoch 36/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4466 -
accuracy: 0.8197 - val_loss: 0.9111 - val_accuracy: 0.6764
Epoch 37/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4430 -
accuracy: 0.8223 - val_loss: 0.9195 - val_accuracy: 0.6769
Epoch 38/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4397 -
accuracy: 0.8235 - val_loss: 0.9319 - val_accuracy: 0.6768
Epoch 39/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4361 -
accuracy: 0.8253 - val_loss: 0.9384 - val_accuracy: 0.6778
Epoch 40/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4335 -
accuracy: 0.8262 - val_loss: 0.9393 - val_accuracy: 0.6749
Epoch 41/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4300 -
accuracy: 0.8280 - val_loss: 0.9463 - val_accuracy: 0.6755
Epoch 42/50
6250/6250 [==============================] - 53s 8ms/step - loss: 0.4275 -
accuracy: 0.8293 - val_loss: 0.9532 - val_accuracy: 0.6762
Epoch 43/50
6250/6250 [==============================] - 53s 8ms/step - loss: 0.4245 -
```

```
accuracy: 0.8310 - val_loss: 0.9590 - val_accuracy: 0.6752
Epoch 44/50
6250/6250 [==============================] - 53s 8ms/step - loss: 0.4217 -
accuracy: 0.8325 - val_loss: 0.9630 - val_accuracy: 0.6739
Epoch 45/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4189 -
accuracy: 0.8338 - val_loss: 0.9676 - val_accuracy: 0.6736
Epoch 46/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4162 -
accuracy: 0.8354 - val_loss: 0.9747 - val_accuracy: 0.6735
Epoch 47/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4143 -
accuracy: 0.8358 - val_loss: 0.9840 - val_accuracy: 0.6730
Epoch 48/50
6250/6250 [==============================] - 53s 9ms/step - loss: 0.4115 -
accuracy: 0.8375 - val_loss: 0.9922 - val_accuracy: 0.6756
Epoch 49/50
6250/6250 [==============================] - 52s 8ms/step - loss: 0.4089 -
accuracy: 0.8394 - val_loss: 0.9988 - val_accuracy: 0.6739
Epoch 50/50
6250/6250 [==============================] - 53s 8ms/step - loss: 0.4069 -
accuracy: 0.8400 - val_loss: 1.0010 - val_accuracy: 0.6747
```

```
[ ]: <keras.callbacks.History at 0x20760f436c8>
```

- Ran the load_and_test_model_weights function to report accuracy and loss for trained model.

```
[ ]: load_and_test_model_weights(model_16,'model_16',X_test,np.array(y_test))
```

```
1563/1563 [==============================] - 7s 4ms/step - loss: 0.7158 -
accuracy: 0.6964
Loss = 0.7158205509185791 Accuracy = 0.6963800191879272
```

5.3 Observations

FFNN - The FFNN achieved the best accuracy of approx 87% for the binary classification task. - The models achieved a best accuracy of approx 70.6% on the ternary classification task.

RNN - The RNN achieved the best accuracy of approx 74% for the binary classification task with pretrained word2vec. - The models achieved the best accuracy of approx 60% for the ternary classification task.

GRU - The GRU model achieved the best accuracy of approx 85% for the binary classification task with pretrained word2vec. - The models achieved the best accuracy of approx 70% for the ternary classification task.

In conclusion, the FFNN Models were able to outperform the RNN Models in both tasks, the reason for such an observation can be that while the FFNNs which outperformed the RNNs were taking an average vector of the entire sentence, the RNN are only taking the first 20 words of the sentence as input

Furthermore, it can be noted that in the FFNN examples, performance dropped significantly when I used only the first 10 words of a sentence to make the input the FFNN. Therefore, we can conclude that providing a neural network with whole information can have a significant impact on its performance.

Also, GRU Models perform better than RNN Models because GRUs are able to maintain context for longer sequences while RNNs suffer at the same task.

6 References

https://radimrehurek.com/gensim/models/word2vec.html

https://www.tensorflow.org/neural_structured_learning

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules/ExponentialDecay

[ ]:
```
!apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-latex-recommended
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

[3]:
```
!jupyter nbconvert --Application.log_level=CRITICAL --to pdf "/content/drive/
↪MyDrive/Colab Notebooks/HW2-CSCI544.ipynb"
```