In [2]:
```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
from sklearn.metrics import f1_score,recall_score,precision_score

import contractions
from bs4 import BeautifulSoup
import nltk

import gensim.downloader as api
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, Embedding, GRU, LSTM,BatchNormalization

#these are all the import statements for the dependencies
#gensim verison is 4.3.0 and that needs to be used to run this code
#DO NO RUN THIS ON COLAB THE CODE IS DIFFERENT DUE TO GENSIM VERSION
# THERE IS A POSSIBILITY OF 2 NOT WORKING AS EACH TIME 60K REVIEWS MIGHT BE DIFFERENT
# REST OF THE CODE SHOULD RUN JUST FINE IN ONE GO
#SINCE I AM NOT ABLE TO RUN THIS ON MY PC DUE TO RAM ISSUES THE OUTPUTS DISPLAYED HERE ARE NOT THE VALID ONES
```

In [3]:
```
# Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Beauty_v1_00.tsv.gz
data=pd.read_csv("amazon_reviews_us_Beauty_v1_00.tsv",sep='\t',on_bad_lines='skip')
df=data.loc[:,["review_body","star_rating"]]
```

Dataset Generation

In [4]:
```
def classfunc(star_): # defines func to cluster reviwes based on rating
    if star_=='5'or star_=='4':
        return 3
    elif star_=='3':
        return 2
    elif star_ =='2'or star_ =='1':
        return 1
    else:
        return 0

df['star_rating'] = df['star_rating'].astype(str) # converting the column's entries to string
df['class'] = df['star_rating'].apply(lambda x: classfunc(x[0])) # applying the clustering custom function on
df.drop(df[(df['class'] == 0)].index, inplace=True) #dropping all entries with incorrect/invalid data
df = df.dropna(subset=['review_body']) #removing all entries with NA
#df = df[df['review_body'].apply(lambda x: len(x.split())>= 15)]



df_balanced = pd.DataFrame()
df_balanced = df.groupby(["class"]).apply(lambda grp: grp.sample(n=20000)) #selecting 2k entries from all thr
```

In [5]:
```
df_balanced['review_body'] = df_balanced['review_body'].str.lower() #converting all reviews to lower case
df_balanced['review_body'] = df_balanced['review_body'].str.replace('http\S+|www.\S+', '', case=False) # remo
df_balanced['review_body'] = df_balanced['review_body'].str.replace('[^a-zA-Z ]', '') #removing non alphabeti
X=df_balanced.review_body
# removing html tags from review text
df_balanced['review_body'] = [BeautifulSoup(X).get_text() for X in df_balanced['review_body'].astype(str) ]
df_balanced['review_body'] = df_balanced['review_body'].str.strip()

df_balanced['review_body'] = df_balanced['review_body'].apply(lambda x: [contractions.fix(word) for word in x
#df_balanced['review_body'] = [' '.join(map(str, word)) for word in df_balanced['review_body']]
```

In [5]:
```
#df_balanced['review_body'] = [' '.join(map(str, word)) for word in df_balanced['review_body']]
```

In [5]:
```
# # Save the DataFrame to a CSV file with the header and without the index
#df_balanced.to_csv('balanceddata.csv', index=False)
```

In [16]:
```
#df_balanced1=pd.read_csv("balanceddata.csv")
```

In [17]:
```
#df_balanced1.head()
```

Out[17]:

	review_body	star_rating	class
0	it just like you spray the white powder on you...	1.0	1
1	it was a mixture of liquid soap and something ...	1.0	1
2	it did not do anything for me except make my c...	1.0	1
3	i am sitting here with a half shaven beard as ...	1.0	1
4	i have tried many products most do a little to...	1.0	1

In [12]:
```
#from nltk.corpus import stopwords

#stop words removal
#english_stopwords = set(stopwords.words('english')) - set(['not', 'no'])
# print(english_stopwords)

#df_balanced['review_body']= df_balanced['review_body'].apply(lambda x: [item for item in x.split() if item n
#df_balanced['review_body'] = [' '.join(map(str, word)) for word in df_balanced['review_body']]
```

Word Embedding

In [12]:
```
model = api.load('word2vec-google-news-300')
# using gensim's wv and loading it using downloader
```

In [ ]:

```
```

In [ ]:

```
result = model.most_similar(positive=["water", "thick"], negative=["thin"])

result1 = model.most_similar("amazing")
result2 = model.similarity("good","money")
result3 = model.similarity("good","great")


print("water+thick-thin = ",result[0][0])# this gives multiple words ans its score so printing highest
print("most similar word to amazing = ",result1[0][0], "with similarity score of ", result1[0][1])
print("similarity between good and money and good and great ",result2,result3)
```

In [6]:

```
import multiprocessing
import gensim
cores = multiprocessing.cpu_count() # finds no of core so that we can use i to mention workers in mymodel

#defining my own model as hyperparams mentioned in pdf
mymodel = gensim.models.Word2Vec(
            window = 13,
            vector_size = 300,
            min_count = 9,
            workers = cores-1
        )
```

In [7]:

```
#building vocab using predefined method
#training model on my reviews

mymodel.build_vocab(df_balanced.review_body, progress_per = 10000)
mymodel.train(df_balanced.review_body, total_examples = mymodel.corpus_count, epochs = mymodel.epochs)
```

Out[7]:

```
(10398497, 14985240)
```

In [8]:
```
## we can conclude that pretrained model has better vocab so its vectors capture more dependencies in them
## we can also see that similarity values ans the most similar words in case of pretrained make more sense as
## so we say tht pretrained vectors is better
```

Out[8]:

class		review_body	star_rating	class
3	4965110	[lightweight, rubberized, which, makes, it, ea...	5	3
	2352657	[it, does not, take, up, much, room, on, the, ...	5.0	3
	3961515	[i have, used, other, products, in, the, past,...	3	2
2	3671506	[was, thicker, than, i, expected, but, i, thin...	3	2
	3018456	[i, have, been, using, this, for, about, two, ...	3	2
3	3809434	[this, is, my, favorite, perfume, for, my, boy...	5	3
2	2440435	[my, wife, got, stretch, mark, and, used, it, ...	3	2
	1482067	[this, one, disappointed, me, with, it, uncomf...	3	2
1	1474646	[it, shipped, pretty, fast, however, it, says,...	1	1
2	4915382	[i, was, not, too, crazy, about, this, it, sti...	3	2
1	1203675	[fake, bake, is, the, way, to, go, you, can, b...	1	1
2	2858872	[its, a, very, cute, purse, not, what, i, expe...	3	2
3	1356431	[very, good]	5	3
1	4940376	[i, was, very, disappointed, with, this, produ...	1	1
	4560545	[i, purchased, this, hair, from, my, local, bs...	1	1
3	1320184	[great, product, really, does, its, job]	5	3
2	3056282	[i, have, seen, rave, reviews, of, this, shade...	3	2
1	4023929	[it, really, did not, fit, my, skin, although,...	1	1
2	2045484	[made, my, skin, even, more, oily]	3.0	2
3	3331465	[fresh, clean, scent, my, absolute, favorite, ...	5	3
1	524855	[this, product, does, not, hold, my, hair, at,...	1	1
2	2698763	[this, kit, came, in, handy, for, me, to, do, ...	3	2
1	4404898	[i, have, used, this, product, exactly, as, su...	1	1
3	378889	[love, this, sunscreenmy, dermatologist, recom...	4	3
	1625896	[seems, to, working, well]	4	3
1	2371644	[bought, it, for, a, friend, it, was not, as, ...	2	1

class		review_body	star_rating	class
	1959772	[the, one, i, got, was, not, white, at, all, i...	3	2
2	2463573	[works, fine, but, the, scent, is, off, puttin...	3	2
	2616583	[it, worked, great, on, your, hair, but, stopp...	3	2
3	1262422	[mamas, favorite]	5	3
	4345794	[if, you, saw, videos, on, youtube, about, thi...	3	2
2	3825864	[pros, love, having, fun, hairbr, cons, gets, ...	3	2
	1042337	[i, cannot, comment, on, the, purity, of, this...	3	2
1	3370612	[this, scraper, is, so, flimsy, that, it, does...	1	1
	2511825	[dried, out, and, irritated, my, skin, for, ab...	1	1
	3009666	[this, is, a, great, product, and, works, well...	5	3
3	2088176	[excellent]	5.0	3
	1804461	[awesome, thank, you]	5	3
	238081	[good, cream, but, expensive]	4	3
2	919825	[was, a, gift, and, she, loved, it]	3	2
1	4560892	[i, swear, by, this, stuff, if, you, use, it, ...	2	1
3	3002900	[my, almost, year, old, has, sucked, his, fore...	5	3
2	4122756	[it, took, me, several, tries, to, finally, ge...	3	2
1	4159987	[i, used, this, a, few, times, now, as, a, top...	2	1
	433593	[these, are, the, worst, clippers, on, the, pl...	1	1
2	1431315	[this, looks, like, a, barbie, comb, its, real...	3.0	2
3	3263247	[this, creme, worked, very, well, it, helped, ...	4	3
2	3437952	[a, little, pricy, but, sometimes, you, just, ...	3	2
	4680704	[nice, bubbles, lavender, water, but, not, muc...	3	2
3	4285369	[i, bought, this, for, my, husband, he, has, b...	5	3

In [11]:

```
result = mymodel.wv.most_similar(positive=["water", "thick"], negative=["thin"])
result1 = mymodel.wv.most_similar("amazing")
result2 = mymodel.wv.similarity("good","money")
result3 = mymodel.wv.similarity("good","great")
result3 = mymodel.wv.similarity("thick","thin")



print("water - thin + thick = ",result[0][0])
print("most similar word to amazing = ",result1[0][0], "with similarity score of ", result1[0][1])
print("similarity between good and money and good and great ",result2,result3)
print("similarity between thick and thin", result3)
```

```
King - M an + W oman =  rinse
most similar word to amazing =  awesome with similarity score of  0.8327875733375549
similarity between good and money and good and great  -0.015617784 0.7993416
similarity between thick and thin 0.7993416
```

Simple models

In [31]:
```
##In this cell, for each entry in review, first i am checing if that word is present in the vocabulary is yes
## and later making sure that there are valid entries if not appending vectors as entries zeros, and if yes t

vectorized_x = []
for review in df_balanced['review_body']:
    vectors = []
    for word in review:
        if word in model.key_to_index:
            vectors.append(model.get_vector(word))
    if len(vectors) > 0:
        vectorized_x.append(np.mean(vectors, axis=0))
    else:
        vectorized_x.append(np.zeros(300))
```

In [32]:
```
y=df_balanced['class']

# using the train test split function with stratify so that we get balanced data
X_train, X_test,y_train, y_test = train_test_split(vectorized_x,y ,
                                    random_state=46,
                                    test_size=0.20,
                                     stratify = y,
                                    shuffle=True)
```

Perceptron

In [24]:
```
from sklearn.utils import multiclass
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report


# Used sklearn module for inbuilt model of perceptron with random state as 50
# (could be any feasible value). This was done using
# Perceptron(random_state=50)
#  Fit the model on training data and later predicted the y_hat using the .predict()
# function.
#  Used the actual y values and yhat (predicted) values to calculate recall precision
# and f1 scores using the sklearn.metrics.
# Extracted the per class values from the generated metrics printed those and for
# the average calculated the metric again with parameter specifying that average
# should be weighted.


model_perceptron = Perceptron(random_state=50)
model_perceptron.fit(X_train, y_train)

y_hat = model_perceptron.predict(X_test)

precision_perceptron = precision_score(y_test,y_hat,average=None)
recall_perceptron = recall_score(y_test,y_hat,average=None)
f1_perceptron = f1_score(y_test,y_hat,average=None)

print("For the current assignment with word2vec features")

print("precision, recall, f1score for class 1: ", precision_perceptron[0], ", ", recall_perceptron[0], ", ",
print("precision, recall, f1score for class 2: ", precision_perceptron[1], ", ", recall_perceptron[1], ", ",
print("precision, recall, f1score for class 3: ", precision_perceptron[2], ", ", recall_perceptron[2], ", ",
print("precision, recall, f1score   average  : ", precision_score(y_test,y_hat,average='weighted'), ", ", rec

print("for HW1 where we used tfidf features (avg of all classes's) precision was 0.651, recall was 0.64, f1 s


#print(classification_report(model_perceptron.predict(X_test), y_test))
```

```
precision, recall, f1score for class 1:  0.4356101674497049 ,  0.904 ,  0.5879196813267215
precision, recall, f1score for class 2:  0.5544440169295883 ,  0.36025 ,  0.43673283830883475
precision, recall, f1score for class 3:  0.9154545454545454 ,  0.25175 ,  0.39490196078431367
precision, recall, f1score   average  :  0.6351695766112795 ,  0.5053333333333333 ,  0.47318482680662327
```

In [ ]:

SVM

In [25]:
```
from sklearn.svm import LinearSVC


# Used sklearn module for inbuilt model of SVM with random state as 4 (could be
# any feasible value). This was done using LinearSVC(random_state=4)
# • Fit the model on training data and later predicted the y_hat using the .predict()
# function.
# • Used the actual y values and yhat (predicted) values to calculate recall precision
# and f1 scores using the sklearn.metrics.
# • Extracted the per class values from the generated metrics printed those and for
# the average calculated the metric again with parameter specifying that the
# average should be weighted.

model_svc = LinearSVC(random_state=4)
model_svc.fit(X_train, y_train)

y_hat = model_svc.predict(X_test)

precision_svc = precision_score(y_test,y_hat,average=None)
recall_svc = recall_score(y_test,y_hat,average=None)
f1_svc = f1_score(y_test,y_hat,average=None)

print("For the current assignment with word2vec features")
print("precision, recall, f1score for class 1: ", precision_svc[0], ", ", recall_svc[0], ", ", f1_svc[0])
print("precision, recall, f1score for class 2: ", precision_svc[1], ", ", recall_svc[1], ", ", f1_svc[1])
print("precision, recall, f1score for class 3: ", precision_svc[2], ", ", recall_svc[2], ", ", f1_svc[2])
print("precision, recall, f1score   average  : ", precision_score(y_test,y_hat,average='weighted'), ", ", rec

print("for HW1 where we used tfidf features (avg of all classes's) precision was 0.687, recall was 0.689, f1

#We can conclude that tfidf features give better results as they use local data to find features and that qua
# so we can conclude that SVM and perceptron work better when features represnt local data as in case of tfid
```

```
precision, recall, f1score for class 1:  0.6559314179796107 ,  0.70775 ,  0.6808561808561809
precision, recall, f1score for class 2:  0.6003143006809848 ,  0.573 ,  0.5863392171910975
precision, recall, f1score for class 3:  0.7382307294361097 ,  0.7135 ,  0.7256547165013985
precision, recall, f1score   average  :  0.6648254826989017 ,  0.66475 ,  0.6642833715162257
```

Feedforward Neural Networks (4a)

In [27]:

```
# Define the multilayer perceptron network with activations as relu for first 2 layers and then set the final
#using relu so that model is non linear and learns complex relations
#
model_fnn = tf.keras.models.Sequential([
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Compile the model
#used categorical_crossentropy for faster convergence and high penalty for mistakes
#used adams for adaptive learning rate, fater concergence and as it requires less memory
model_fnn.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])


X_trainfnn = np.array(X_train)
X_testfnn = np.array(X_test)
y_trainfnn = np.array(y_train)
y_testfnn = np.array(y_test)

y_trainfnn = to_categorical(y_trainfnn-1,num_classes = 3)
y_testfnn = to_categorical(y_testfnn-1,num_classes = 3)

#creating the checkpoint to save the best epocs model
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy', save_best_only=True)

#fitting the model with hyperparameters that gave decent scores
history = model_fnn.fit(X_trainfnn, y_trainfnn, epochs=10, batch_size=32, shuffle=True, validation_data=(X_te
best_model = load_model('best_model.h5')

# Evaluate the performance of the trained model
test_loss, test_acc = best_model.evaluate(X_testfnn, y_testfnn)

print('Test accuracy of fnn:', test_acc*100)
```

```
Epoch 1/10
1500/1500 [==============================] - 7s 4ms/step - loss: 0.8430 - accuracy: 0.6143 - val_loss: 0.782
4 - val_accuracy: 0.6536
Epoch 2/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.7804 - accuracy: 0.6523 - val_loss: 0.814
3 - val_accuracy: 0.6261
Epoch 3/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.7624 - accuracy: 0.6607 - val_loss: 0.796
4 - val_accuracy: 0.6424
Epoch 4/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.7477 - accuracy: 0.6684 - val_loss: 0.751
9 - val_accuracy: 0.6678
Epoch 5/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7391 - accuracy: 0.6734 - val_loss: 0.741
0 - val_accuracy: 0.6747
Epoch 6/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.7300 - accuracy: 0.6774 - val_loss: 0.747
1 - val_accuracy: 0.6695
Epoch 7/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.7218 - accuracy: 0.6826 - val_loss: 0.736
1 - val_accuracy: 0.6765
Epoch 8/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.7157 - accuracy: 0.6850 - val_loss: 0.733
5 - val_accuracy: 0.6749
Epoch 9/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.7085 - accuracy: 0.6875 - val_loss: 0.731
1 - val_accuracy: 0.6816
Epoch 10/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7014 - accuracy: 0.6912 - val_loss: 0.749
3 - val_accuracy: 0.6671
375/375 [==============================] - 2s 3ms/step - loss: 0.7311 - accuracy: 0.6816
Test accuracy of fnn: 68.15833449363708
```

Feedforward Neural Networks (4b)

In [41]:
```
vec_x = []

#since we need vectors to be concatenated vectors of first 10 words in each review, writing this code to achi


def vectorizex(review):
#for review in df_balanced['review_body']:
    vectors = []
    for word in review:
        if word in model.key_to_index:
            vectors.append(model.get_vector(word))
    if len(vectors)<10:
        while(len(vectors)!=10):
            vectors.append(np.zeros(300))
    #print(len(vectors))
    return np.concatenate(vectors[:10])

for review in df_balanced['review_body']:
    vec_x.append(vectorizex(review))
```

In [44]:
```

y=df_balanced['class']

# using the train test split function
X_train_1, X_test_1,y_train_1, y_test_1 = train_test_split(vec_x,y ,
                                    random_state=46,
                                    test_size=0.20,
                                     stratify = y,
                                    shuffle=True)
```

In [45]:

```
#using relu so that model is non linear and learns complex relations
#used categorical_crossentropy for faster convergence and high penalty for mistakes
#used adams for adaptive learning rate, faster concergence and as it requires less memory

# Define the multilayer perceptron network
model_fnn_4b = tf.keras.models.Sequential([
    tf.keras.layers.Dense(100, activation='relu',input_dim=3000),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Compile the model
model_fnn_4b.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
X_trainfnn_1 = np.array(X_train_1)
X_testfnn_1 = np.array(X_test_1)
y_trainfnn_1 = np.array(y_train_1)
y_testfnn_1 = np.array(y_test_1)

y_trainfnn_1 = to_categorical(y_trainfnn_1-1,num_classes = 3)
y_testfnn_1 = to_categorical(y_testfnn_1-1,num_classes = 3)

# X_trainfnn_1 = X_trainfnn_1.reshape(48000,3000)
# X_testfnn_1 = X_testfnn_1.reshape(12000,3000)


checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy', save_best_only=True)


history = model_fnn_4b.fit(X_trainfnn_1, y_trainfnn_1, epochs=10, batch_size=32, shuffle=True, validation_dat
best_model = load_model('best_model.h5')

# Evaluate the performance of the trained model
#test_loss, test_acc = model_fnn.evaluate(X_testfnn, y_testfnn)
test_loss, test_acc = best_model.evaluate(X_testfnn_1, y_testfnn_1)

print('Test accuracy of fnn(4b):', test_acc*100)

##What do you conclude by comparing accuracy values you obtain with
##those obtained in the “’Simple Models” section.
```

```
##Accuracy for both cases of fnn is better as fnn perform better on non linear relationships which is the cas
##fnn seems to capture distributed relationships better
##FNNs are more flexible than SVM or Perceptron
##word2vec has more dimentionality as it represents more data which is used better by fnn than simple models
```

```
Epoch 5/10
1500/1500 [==============================] - 14s 9ms/step - loss: 0.5257 - accuracy: 0.7762 - val_loss:
1.1189 - val_accuracy: 0.5549
Epoch 6/10
1500/1500 [==============================] - 13s 9ms/step - loss: 0.4216 - accuracy: 0.8271 - val_loss:
1.3032 - val_accuracy: 0.5452
Epoch 7/10
1500/1500 [==============================] - 13s 9ms/step - loss: 0.3321 - accuracy: 0.8675 - val_loss:
1.4645 - val_accuracy: 0.5446
Epoch 8/10
1500/1500 [==============================] - 13s 9ms/step - loss: 0.2660 - accuracy: 0.8961 - val_loss:
1.7260 - val_accuracy: 0.5378
Epoch 9/10
1500/1500 [==============================] - 13s 9ms/step - loss: 0.2187 - accuracy: 0.9149 - val_loss:
1.9333 - val_accuracy: 0.5343
Epoch 10/10
1500/1500 [==============================] - 13s 9ms/step - loss: 0.1828 - accuracy: 0.9302 - val_loss:
2.2429 - val_accuracy: 0.5387
375/375 [==============================] - 2s 4ms/step - loss: 0.8869 - accuracy: 0.5768
Test accuracy of fnn(4b): 57.68333077430725
```

RNN (5a)

In [6]:
```
idx_x = []

## since we wanted to train rnn insted of using feature vectors we use the vectors representing the indices o
## also since we want the length to be 20 we are padding if length is less.

def idxx(review):
    vectors = []
    for word in review:
        if word in model.key_to_index:
            vectors.append( model.key_to_index[word])
    if len(vectors)<20:
        while(len(vectors)!=20):
            vectors.append(0)

    return vectors[:20]

for review in df_balanced['review_body']:
    idx_x.append(idxx(review))
```

In [7]:
```
y=df_balanced['class']

# using the train test split function
X_train_2, X_test_2,y_train_2, y_test_2 = train_test_split(idx_x,y ,
                                    random_state=42,
                                    test_size=0.20,
                                     stratify = y,
                                    shuffle=True)
```

In [56]:

```
#using relu so that model is non linear and learns complex relations
#used categorical_crossentropy for faster convergence and high penalty for mistakes
#used adams for adaptive learning rate, faster concergence and as it requires less memory

model_rnn = Sequential()
model_rnn.add(Embedding(input_dim=len(model.key_to_index), output_dim=300, weights=[model.vectors], input_len
model_rnn.add(SimpleRNN(20))
model_rnn.add(Dense(3, activation='softmax'))

# Compile the model
model_rnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy', save_best_only=True)

X_trainrnn_2 = np.array(X_train_2)
X_testrnn_2 = np.array(X_test_2)
y_trainrnn_2 = np.array(y_train_2)
y_testrnn_2 = np.array(y_test_2)

y_trainrnn_2 = to_categorical(y_trainrnn_2-1,num_classes = 3)
y_testrnn_2 = to_categorical(y_testrnn_2-1,num_classes = 3)


# Fit the model
model_rnn.fit(X_trainrnn_2, y_trainrnn_2, batch_size=32, epochs=15, validation_data=(X_testrnn_2, y_testrnn_2
best_model = load_model('best_model.h5')


# Evaluate the model on the testing set
loss, accuracy = best_model.evaluate(X_testrnn_2, y_testrnn_2)
print('Test loss:', loss)
print('Test accuracy:', accuracy)



# What do you conclude by comparing accuracy values you obtain with
# those obtained with feedforward neural network models.

##RNN is performing better in this case as our data is sort of sequential in nature so rnns are known to perf
#fnns are not as efficient in capturing long term dependencies than rnn so that might be one of the reasons b
## rnns are maybe performing better as context of word is better captured in even limited vocabulary due to t
##also due to large size of data and rnns being more complax that could be one of the reasons
```

```
## so we conclude that rnn is better in all the above things than fnn by seeing the results
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[56], line 2
      1 model_rnn = Sequential()
----> 2 model_rnn.add(Embedding(input_dim=len(model.vocab), output_dim=300, weights=[model.vectors], input_l
ength=20, trainable=False))
      3 model_rnn.add(SimpleRNN(20))
      4 model_rnn.add(Dense(3, activation='softmax'))

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\gensim\models\keyedvectors.py:734, in Keyed
Vectors.vocab(self)
    732 @property
    733 def vocab(self):
--> 734     raise AttributeError(
    735         "The vocab attribute was removed from KeyedVector in Gensim 4.0.0.\n"
    736         "Use KeyedVector's .key_to_index dict, .index_to_key list, and methods "
    737         ".get_vecattr(key, attr) and .set_vecattr(key, attr, new_val) instead.\n"
    738         "See https://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-Gensim-3.x-to-4 (http
s://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-Gensim-3.x-to-4)"
    739     )

AttributeError: The vocab attribute was removed from KeyedVector in Gensim 4.0.0.
Use KeyedVector's .key_to_index dict, .index_to_key list, and methods .get_vecattr(key, attr) and .set_vecat
tr(key, attr, new_val) instead.
See https://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-Gensim-3.x-to-4 (https://github.com/RaRe
-Technologies/gensim/wiki/Migrating-from-Gensim-3.x-to-4)
```

RNN (5B)

In [ ]:

```
#using relu so that model is non linear and learns complex relations
#used categorical_crossentropy for faster convergence and high penalty for mistakes
#used adams for adaptive learning rate, faster concergence and as it requires less memory

model_rnn = Sequential()
model_rnn.add(Embedding(input_dim=len(model.key_to_index), output_dim=300, weights=[model.vectors], input_len
#model_rnn.add(Embedding(input_dim=len(model.vocab), output_dim=300, weights=[model.vectors], input_length=20

model_rnn.add(GRU(units=20,activation='relu'))
model_rnn.add(Dense(3, activation='softmax'))

# Compile the model
model_rnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

checkpoint = ModelCheckpoint('best_model_1.h5', monitor='val_accuracy', save_best_only=True)

# X_trainrnn_2 = np.array(X_train_2)
# X_testrnn_2 = np.array(X_test_2)
# y_trainrnn_2 = np.array(y_train_2)
# y_testrnn_2 = np.array(y_test_2)

# y_trainrnn_2 = to_categorical(y_trainrnn_2-1,num_classes = 3)
# y_testrnn_2 = to_categorical(y_testrnn_2-1,num_classes = 3)


# Fit the model
model_rnn.fit(X_trainrnn_2, y_trainrnn_2, batch_size=32, epochs=15, validation_data=(X_testrnn_2, y_testrnn_2
best_model = load_model('best_model_1.h5')


# Evaluate the model on the testing set
loss, accuracy = best_model.evaluate(X_testrnn_2, y_testrnn_2)
print('Test loss for rnn in 5b:', loss)
print('Test accuracy for rnn in 5b:', accuracy)
```

RNN(5c)

In [ ]:
```
#using relu so that model is non linear and learns complex relations
#used categorical_crossentropy for faster convergence and high penalty for mistakes
#used adams for adaptive learning rate, faster concergence and as it requires less memory

model_rnn = Sequential()
model_rnn.add(Embedding(input_dim=len(model.key_to_index), output_dim=300, weights=[model.vectors], input_len
#model_rnn.add(Embedding(input_dim=len(model.vocab), output_dim=300, weights=[model.vectors], input_length=20

model_rnn.add(LSTM(units=20,activation='relu'))
model_rnn.add(Dense(3, activation='softmax'))

# Compile the model
model_rnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

checkpoint = ModelCheckpoint('best_model_2.h5', monitor='val_accuracy', save_best_only=True)

# X_trainrnn_2 = np.array(X_train_2)
# X_testrnn_2 = np.array(X_test_2)
# y_trainrnn_2 = np.array(y_train_2)
# y_testrnn_2 = np.array(y_test_2)

# y_trainrnn_2 = to_categorical(y_trainrnn_2-1,num_classes = 3)
# y_testrnn_2 = to_categorical(y_testrnn_2-1,num_classes = 3)


# Fit the model with hyper
model_rnn.fit(X_trainrnn_2, y_trainrnn_2, batch_size=32, epochs=10, validation_data=(X_testrnn_2, y_testrnn_2
best_model = load_model('best_model_2.h5')


# Evaluate the model on the testing set
loss, accuracy = best_model.evaluate(X_testrnn_2, y_testrnn_2)
print('Test loss for rnn in 5c:', loss)
print('Test accuracyfor rnn in 5c:', accuracy)
```

In [ ]:
```
##What do you conclude by comparing accuracy values you obtain by GRU,
#LSTM, and simple RNN.


###Observation: the best performing rnn out of all three is LSTM, then GRU and then simple rnn

##inferences/justification:
## maybe simple is suffering from vanishing gradients
## LSTM due to its better and complex architecture is capturing context in words better. also it seems it is
## gru seems to  be better in generalizing over data that is not seen before than simple rnn
```