HW2-Ashwin-Chafale

September 27, 2022

1 CSCI-544 Homework Assignment No. 2

1.0.1 Name : Ashwin Chafale

1.0.2 USC ID : 1990624801

[1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

1.1 1. Dataset Generation

- Amazon reviews dataset

[2]:
```
df = pd.read_csv("amazon_reviews_us_Jewelry_v1_00.tsv", sep='\t', header=0,␣
↪on_bad_lines='skip')
df = df[['review_body','star_rating']]
df = df.dropna()
df = df.reset_index(drop=True)
df['star_rating'] = df['star_rating'].astype(int)
df.shape
```

[2]:
```
(1766748, 2)
```

[3]:
```
df['star_rating'].value_counts()
```

[3]:
```
5    1080871
4     270424
3     159654
1     155002
2     100797
Name: star_rating, dtype: int64
```

1.1.1 i. Down-sample 5-star & 4-star reviews, Up-sample 3-star, 2-star, 1-star reviews to get 100K balance dataset

Reference : https://elitedatascience.com/imbalanced-classes

[4]:
```python
from sklearn.utils import resample
# separating reviews
five_star =  df.loc[ df['star_rating'] == 5]
four_star =  df.loc[ df['star_rating'] == 4]
three_star =  df.loc[ df['star_rating'] == 3]
two_star =  df.loc[ df['star_rating'] == 2]
one_star =  df.loc[ df['star_rating'] == 1]

# Downsample 5-star class
five_star_downsampled = resample(five_star,
                                 replace=False,    # sample without replacement
                                 n_samples=20000,     # to match minority class
                                 random_state=123) # reproducible results

# Downsample 4-star class
four_star_downsampled = resample(four_star,
                                 replace=False,    # sample without replacement
                                 n_samples=20000,     # to match minority class
                                 random_state=123) # reproducible results

# Upsample 3-star class
three_star_upsampled = resample(three_star,
                                replace=True,     # sample with replacement
                                n_samples=20000,    # to match majority class
                                random_state=123) # reproducible results

# Upsample 2-star class
two_star_upsampled = resample(two_star,
                              replace=True,     # sample with replacement
                              n_samples=20000,    # to match majority class
                              random_state=123) # reproducible results

# Upsample 1-star class
one_star_upsampled = resample(one_star,
                              replace=True,     # sample with replacement
                              n_samples=20000,    # to match majority class
                              random_state=123) # reproducible results

balanced_data = pd.concat([five_star_downsampled, four_star_downsampled,␣
 ↪three_star_upsampled, two_star_upsampled, one_star_upsampled], axis=0)
balanced_data["star_rating"].value_counts()
```

[4]:
```
5    20000
4    20000
3    20000
2    20000
1    20000
```

```
Name: star_rating, dtype: int64
```

1.1.2 ii. Test-train split

[5]:
```
# Train - test split
from sklearn.model_selection import train_test_split

five_star_X_train, five_star_X_test, five_star_Y_train, five_star_Y_test = \
    train_test_split(balanced_data[balanced_data["star_rating"] ==␣
 ↪5]["review_body"],
                     balanced_data[balanced_data["star_rating"] ==␣
 ↪5]["star_rating"], test_size=0.2, random_state=30)

four_star_X_train, four_star_X_test, four_star_Y_train, four_star_Y_test = \
    train_test_split(balanced_data[balanced_data["star_rating"] ==␣
 ↪4]["review_body"],
                     balanced_data[balanced_data["star_rating"] ==␣
 ↪4]["star_rating"], test_size=0.2, random_state=30)

three_star_X_train, three_star_X_test, three_star_Y_train, three_star_Y_test = \
    train_test_split(balanced_data[balanced_data["star_rating"] ==␣
 ↪3]["review_body"],
                     balanced_data[balanced_data["star_rating"] ==␣
 ↪3]["star_rating"], test_size=0.2, random_state=30)

two_star_X_train, two_star_X_test, two_star_Y_train, two_star_Y_test = \
    train_test_split(balanced_data[balanced_data["star_rating"] ==␣
 ↪2]["review_body"],
                     balanced_data[balanced_data["star_rating"] ==␣
 ↪2]["star_rating"], test_size=0.2, random_state=30)

one_star_X_train, one_star_X_test, one_star_Y_train, one_star_Y_test = \
    train_test_split(balanced_data[balanced_data["star_rating"] ==␣
 ↪1]["review_body"],
                     balanced_data[balanced_data["star_rating"] ==␣
 ↪1]["star_rating"], test_size=0.2, random_state=30)

X_train = pd.concat([five_star_X_train, four_star_X_train, three_star_X_train,␣
 ↪two_star_X_train, one_star_X_train])
X_test = pd.concat([five_star_X_test, four_star_X_test, three_star_X_test,␣
 ↪two_star_X_test, one_star_X_test])
Y_train = pd.concat([five_star_Y_train, four_star_Y_train, three_star_Y_train,␣
 ↪two_star_Y_train, one_star_Y_train])
Y_test = pd.concat([five_star_Y_test, four_star_Y_test, three_star_Y_test,␣
 ↪two_star_Y_test, one_star_Y_test])
```

```
print("Train: ", X_train.shape, Y_train.shape, "Test: ", (X_test.shape, Y_test.
↪shape))
```

```
Train:  (80000,) (80000,) Test:  ((20000,), (20000,))
```

1.1.3 iii. Data Preprocessing

[6]:
```
from bs4 import BeautifulSoup
import re
import contractions
import nltk
from nltk.stem import WordNetLemmatizer

def data_preprocessing(data):
    # convert all reviews to lower case
    data = data.apply(lambda x: " ".join(x.lower() for x in str(x).split()))

    # remove HTML tags as well as URLs from reviews.
    data = data.apply(lambda x: BeautifulSoup(x).get_text())
    data = data.apply(lambda x: re.sub(r'https?://\S+|www\.\S+', "", x))

    # contractions
    data = data.apply(lambda x:contractions.fix(x))

    # remove the non-alpha characters
    data = data.apply(lambda x: " ".join([re.sub("[^A-Za-z]+","", x) for x in␣
↪nltk.word_tokenize(x)]))

    # remove extra spaces among the words
    data = data.apply(lambda x: re.sub(' +', ' ', x))

    # removing stop words
    stop_words=['the', 'a', 'and', 'is', 'be', 'will', 'are']
    data = data.apply(lambda x: " ".join([x for x in x.split() if x not in␣
↪stop_words]))

    lemmatizer = WordNetLemmatizer()
    data = data.apply(lambda x: " ".join([lemmatizer.lemmatize(w) for w in nltk.
↪word_tokenize(x)]))

    return data
```

[7]:
```
X_train = data_preprocessing(X_train)
X_test = data_preprocessing(X_test)
```

1.2 2. Word Embedding

Reference : https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html

1.2.1 a) Exploring pretrained "word2vec-google-news-300"

[8]:
```
# Loading 'word2vec-google-news-300' model
import gensim.downloader as api
wv_google = api.load('word2vec-google-news-300')
```

[9]:
```
# checking semantic similarities
# Example 1
result = wv_google.most_similar(positive=['woman', 'king'], negative=['man'])
print("{}: {:.4f}".format(*result[0]))
```

```
queen: 0.7118
```

[10]:
```
# Example 2
wv_google.similarity('excellent', 'outstanding')
```

[10]:
```
0.55674857
```

[11]:
```
# Example 3
wv_google.doesnt_match(['fire', 'water', 'land', 'sea', 'air', 'car'])
```

[11]:
```
'car'
```

1.2.2 b) Train a Word2Vec model using your own dataset.

Reference : https://www.kaggle.com/code/chewzy/tutorial-how-to-train-your-custom-word-embedding

[16]:
```
from gensim.models import Word2Vec
full_dataset = pd.concat([X_train, X_test],axis=0)
sentences = []
for review in full_dataset:
  tokens = review.split()
  sentences.append(tokens)
```

[17]:
```
custom_wv_model = Word2Vec(sentences=sentences, size=300, window=11,␣
↪min_count=10)
```

[19]:
```
result = custom_wv_model.most_similar(positive=['woman', 'king'],␣
↪negative=['man'])
print("{}: {:.4f}".format(*result[0]))
```

```
avenue: 0.5790
```

[20]:
```python
# Example 1
custom_wv_model.similarity('excellent', 'outstanding')
```

[20]: 0.7977613

[21]:
```python
# Example 2
custom_wv_model.most_similar("good")
```

[21]:
```
[('decent', 0.8035435676574707),
 ('great', 0.7997811436653137),
 ('nice', 0.6498663425445557),
 ('high', 0.6413455009460449),
 ('excellent', 0.6235317587852478),
 ('ok', 0.6038389205932617),
 ('fantastic', 0.5990501046180725),
 ('poor', 0.5846014022827148),
 ('bad', 0.5676741003990173),
 ('control', 0.5650346875190735)]
```

Question : What do you conclude from comparing vectors generated by yourself and the pretrained model? Which of the Word2Vec models seems to encode semantic similarities between words better? Answer : Pretrained Google word2vec model have diverse variety of words in its vocabulary and therefore is able to capture semantic similarities of diverse set of words better. For our own custom build model we need to a large and diverse corpus to train to get the desired results. Hence, Pretrained google word2vec is better than our own custom build.

1.3 3. Simple models

1.3.1 Average Word2Vec vectors for each review

[22]:
```python
# average word2vec
def get_avg_wor2vec(_reviews):
    word_list = _reviews.split()
    words_cnt = 0
    word_vector = np.zeros(300)
    for word in word_list:
        if word in wv_google:
            word_vector += wv_google[word]
            words_cnt += 1
    if words_cnt != 0:
        word_vector /= words_cnt
    return word_vector
```

[23]:
```python
train_vec = []
for reviews in X_train:
    train_vec.append(get_avg_wor2vec(reviews))
train_vec = np.array(train_vec)
```

```
test_vec = []
for reviews in X_test:
    test_vec.append(get_avg_wor2vec(reviews))
test_vec = np.array(test_vec)
```

1.3.2 a) Perceptron

[24]:
```
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
perceptron = Perceptron(max_iter=1000, random_state=0)
perceptron.fit(train_vec,Y_train)
y_test_predicted = perceptron.predict(test_vec)

report = classification_report(Y_test, y_test_predicted, output_dict=True)
pd.DataFrame.from_dict(report)
```

[24]:

	1	2	…	macro avg	weighted avg
precision	0.469344	0.220267	…	0.429588	0.429588
recall	0.549250	0.704250	…	0.333650	0.333650
f1-score	0.506163	0.335577	…	0.279520	0.279520
support	4000.000000	4000.000000	…	20000.000000	20000.000000

[4 rows x 8 columns]

1.3.3 b) SVM

[25]:
```
from sklearn.svm import LinearSVC
svm = LinearSVC(multi_class="ovr", random_state=0)
svm.fit(train_vec,Y_train)
y_test_predicted = svm.predict(test_vec)

report = classification_report(Y_test, y_test_predicted, output_dict=True)
pd.DataFrame.from_dict(report)
```

[25]:

	1	2	…	macro avg	weighted avg
precision	0.511508	0.399172	…	0.466722	0.466722
recall	0.716750	0.265250	…	0.485100	0.485100
f1-score	0.596981	0.318714	…	0.464614	0.464614
support	4000.000000	4000.000000	…	20000.000000	20000.000000

[4 rows x 8 columns]

Comparing performance of Perceptron & SVM model trained using TF-IDF and Word2Vec features Reading accuracy values of Perceptron and SVM model from HW1

```
[ ]: perceptron_using_tfidf = pd.read_csv("perceptron.csv")
perceptron_using_tfidf
```

```
[ ]:   Unnamed: 0           1           2  …  accuracy     macro avg  weighted
avg
0  precision     0.529361     0.302142  …     0.414      0.425204
0.425204
1     recall     0.462000     0.469000  …     0.414      0.414000
0.414000
2   f1-score     0.493392     0.367519  …     0.414      0.413541
0.413541
3    support  4000.000000  4000.000000  …     0.414  20000.000000
20000.000000

[4 rows x 9 columns]
```

```
[ ]: svm_using_tfidf = pd.read_csv("svm.csv")
svm_using_tfidf
```

```
[ ]:   Unnamed: 0           1           2  …  accuracy     macro avg  weighted
avg
0  precision     0.563424     0.404890  …   0.51355      0.500972
0.500972
1     recall     0.676250     0.339500  …   0.51355      0.513550
0.513550
2   f1-score     0.614703     0.369323  …   0.51355      0.504211
0.504211
3    support  4000.000000  4000.000000  …   0.51355  20000.000000
20000.000000

[4 rows x 9 columns]
```

Question : What do you conclude from comparing performances for the models trained using the two different feature types (TF-IDF and your trained Word2Vec features)?
Answer : Simple model (Perceptron & SVM) trained using TF-IDF has better accuracy as compared to model trained using Word2Vec.

1.4 4. Feedforward Neural Networks

1.4.1 a) Train using average Word2Vec

```
[26]: import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.optimizers import SGD
```

[27]:
```
model = Sequential()
model.add(Dense(50, input_shape = (300,),activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(5, activation='softmax'))
sgd = SGD(0.01)
model.compile(loss="sparse_categorical_crossentropy", optimizer=sgd,␣
 ↪metrics=["accuracy"])
model.summary()
```

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

 dense_1 (Dense)             (None, 10)                510

 dropout (Dropout)           (None, 10)                0

 dense_2 (Dense)             (None, 5)                 55

=================================================================
Total params: 15,615
Trainable params: 15,615
Non-trainable params: 0
_________________________________________________________________
```

[28]:
```
X_train_vec = X_train.apply(lambda x: get_avg_wor2vec(x)).to_numpy()
X_test_vec = X_test.apply(lambda x: get_avg_wor2vec(x)).to_numpy()
X_train_vec = np.concatenate([np.concatenate(X_train_vec, axis=0)], axis=0).
 ↪reshape(-1, 300)
X_test_vec = np.concatenate([np.concatenate(X_test_vec, axis=0)], axis=0).
 ↪reshape(-1, 300)
```

[29]:
```
Y_train_np = Y_train.apply(lambda x : x - 1)
Y_train_np = Y_train_np.to_numpy()
Y_test_np = Y_test.apply(lambda x : x - 1)
Y_test_np = Y_test_np.to_numpy()
```

[30]:
```
model.fit(X_train_vec, Y_train_np, epochs=100)
```

```
Epoch 1/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.5815 -
accuracy: 0.2971
Epoch 2/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.4592 -
```

```
accuracy: 0.3608
Epoch 3/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.3608 -
accuracy: 0.3911
Epoch 4/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.3119 -
accuracy: 0.4102
Epoch 5/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2834 -
accuracy: 0.4262
Epoch 6/100
2500/2500 [==============================] - 4s 2ms/step - loss: 1.2645 -
accuracy: 0.4386
Epoch 7/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2511 -
accuracy: 0.4466
Epoch 8/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2426 -
accuracy: 0.4523
Epoch 9/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2345 -
accuracy: 0.4576
Epoch 10/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2268 -
accuracy: 0.4587
Epoch 11/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2214 -
accuracy: 0.4635
Epoch 12/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2169 -
accuracy: 0.4652
Epoch 13/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2132 -
accuracy: 0.4655
Epoch 14/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2098 -
accuracy: 0.4675
Epoch 15/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2063 -
accuracy: 0.4706
Epoch 16/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2042 -
accuracy: 0.4719
Epoch 17/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.2005 -
accuracy: 0.4729
Epoch 18/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1992 -
```

```
accuracy: 0.4741
Epoch 19/100
2500/2500 [==============================] - 4s 2ms/step - loss: 1.1988 -
accuracy: 0.4735
Epoch 20/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1962 -
accuracy: 0.4730
Epoch 21/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1920 -
accuracy: 0.4764
Epoch 22/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1932 -
accuracy: 0.4763
Epoch 23/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1902 -
accuracy: 0.4774
Epoch 24/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1878 -
accuracy: 0.4784
Epoch 25/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1857 -
accuracy: 0.4779
Epoch 26/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1846 -
accuracy: 0.4804
Epoch 27/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1823 -
accuracy: 0.4799
Epoch 28/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1838 -
accuracy: 0.4785
Epoch 29/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1807 -
accuracy: 0.4820
Epoch 30/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1786 -
accuracy: 0.4829
Epoch 31/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1785 -
accuracy: 0.4818
Epoch 32/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1770 -
accuracy: 0.4818
Epoch 33/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1763 -
accuracy: 0.4808
Epoch 34/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1745 -
```

```
accuracy: 0.4807
Epoch 35/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1736 -
accuracy: 0.4824
Epoch 36/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1718 -
accuracy: 0.4828
Epoch 37/100
2500/2500 [==============================] - 4s 2ms/step - loss: 1.1723 -
accuracy: 0.4843
Epoch 38/100
2500/2500 [==============================] - 4s 2ms/step - loss: 1.1703 -
accuracy: 0.4850
Epoch 39/100
2500/2500 [==============================] - 4s 2ms/step - loss: 1.1690 -
accuracy: 0.4867
Epoch 40/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1675 -
accuracy: 0.4855
Epoch 41/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1664 -
accuracy: 0.4882
Epoch 42/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1659 -
accuracy: 0.4856
Epoch 43/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1661 -
accuracy: 0.4877
Epoch 44/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1636 -
accuracy: 0.4851
Epoch 45/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1627 -
accuracy: 0.4884
Epoch 46/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1611 -
accuracy: 0.4862
Epoch 47/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1617 -
accuracy: 0.4879
Epoch 48/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1596 -
accuracy: 0.4893
Epoch 49/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1576 -
accuracy: 0.4875
Epoch 50/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1578 -
```

```
accuracy: 0.4899
Epoch 51/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1572 -
accuracy: 0.4890
Epoch 52/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1564 -
accuracy: 0.4883
Epoch 53/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1538 -
accuracy: 0.4917
Epoch 54/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1537 -
accuracy: 0.4898
Epoch 55/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1529 -
accuracy: 0.4897
Epoch 56/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1542 -
accuracy: 0.4895
Epoch 57/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1516 -
accuracy: 0.4909
Epoch 58/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1514 -
accuracy: 0.4918
Epoch 59/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1518 -
accuracy: 0.4904
Epoch 60/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1499 -
accuracy: 0.4940
Epoch 61/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1489 -
accuracy: 0.4934
Epoch 62/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1482 -
accuracy: 0.4923
Epoch 63/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1463 -
accuracy: 0.4928
Epoch 64/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1461 -
accuracy: 0.4942
Epoch 65/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1454 -
accuracy: 0.4928
Epoch 66/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1451 -
```

```
accuracy: 0.4935
Epoch 67/100
2500/2500 [==============================] - 4s 2ms/step - loss: 1.1442 -
accuracy: 0.4943
Epoch 68/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1434 -
accuracy: 0.4945
Epoch 69/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1439 -
accuracy: 0.4941
Epoch 70/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1429 -
accuracy: 0.4939
Epoch 71/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1401 -
accuracy: 0.4968
Epoch 72/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1397 -
accuracy: 0.4959
Epoch 73/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1383 -
accuracy: 0.4962
Epoch 74/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1402 -
accuracy: 0.4961
Epoch 75/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1396 -
accuracy: 0.4972
Epoch 76/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1383 -
accuracy: 0.4953
Epoch 77/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1363 -
accuracy: 0.4986
Epoch 78/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1359 -
accuracy: 0.4949
Epoch 79/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1363 -
accuracy: 0.4938
Epoch 80/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1355 -
accuracy: 0.4975
Epoch 81/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1338 -
accuracy: 0.4990
Epoch 82/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1336 -
```

```
accuracy: 0.4979
Epoch 83/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1335 -
accuracy: 0.4970
Epoch 84/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1334 -
accuracy: 0.4991
Epoch 85/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1314 -
accuracy: 0.4981
Epoch 86/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1331 -
accuracy: 0.4969
Epoch 87/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1301 -
accuracy: 0.5007
Epoch 88/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1314 -
accuracy: 0.4996
Epoch 89/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1277 -
accuracy: 0.4992
Epoch 90/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1298 -
accuracy: 0.4981
Epoch 91/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1295 -
accuracy: 0.4990
Epoch 92/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1267 -
accuracy: 0.5011
Epoch 93/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1279 -
accuracy: 0.5005
Epoch 94/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1268 -
accuracy: 0.5013
Epoch 95/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1279 -
accuracy: 0.4993
Epoch 96/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1254 -
accuracy: 0.5006
Epoch 97/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1243 -
accuracy: 0.5016
Epoch 98/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1239 -
```

```
accuracy: 0.5017
Epoch 99/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1243 -
accuracy: 0.5005
Epoch 100/100
2500/2500 [==============================] - 4s 1ms/step - loss: 1.1237 -
accuracy: 0.5005
```

[30]:
```
<keras.callbacks.History at 0x7f7ab2d54f50>
```

[31]:
```
test_loss, test_acc = model.evaluate(X_test_vec, Y_test_np)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

```
625/625 [==============================] - 1s 1ms/step - loss: 1.1291 -
accuracy: 0.5060
Test Loss: 1.129124641418457
Test Accuracy: 0.5059999823570251
```

1.4.2 b) Generate the input feature by concatenating the first 10 Word2Vec vectors for each review as the input feature

[32]:
```
def get_concatenated_first10_feature_vector(dataset):
  feature_10_word2vec = []
  for reviews in dataset:
    words = reviews.split()
    max_words = 10
    review_embedding = []
    for word in words:
      if len(review_embedding) < max_words:
        word_vec = np.zeros(300)
        if word in wv_google:
          word_vec += wv_google[word]
        review_embedding.append(word_vec)
    if len(review_embedding) < max_words:
      while len(review_embedding) != max_words:
        review_embedding.append(np.zeros(300))
    review_embedding = np.concatenate(review_embedding)
    feature_10_word2vec.append(review_embedding)
  feature_10_word2vec = np.array(feature_10_word2vec)
  return feature_10_word2vec
```

[33]:
```
model = Sequential()
model.add(Dense(50, input_shape = (3000,),activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dropout(0.2))
```

```
model.add(Dense(5, activation='softmax'))
sgd = SGD(0.01)
model.compile(loss="sparse_categorical_crossentropy", optimizer=sgd,␣
 ↪metrics=["accuracy"])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 50)                150050

 dense_4 (Dense)             (None, 10)                510

 dropout_1 (Dropout)         (None, 10)                0

 dense_5 (Dense)             (None, 5)                 55

=================================================================
Total params: 150,615
Trainable params: 150,615
Non-trainable params: 0
_________________________________________________________________
```

[34]:
```
X_train_10_word2vec = get_concatenated_first10_feature_vector(X_train)
X_test_10_word2vec = get_concatenated_first10_feature_vector(X_test)
```

[35]:
```
model.fit(X_train_10_word2vec, Y_train_np, epochs=100)
```

```
Epoch 1/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.5196 -
accuracy: 0.2954
Epoch 2/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.3897 -
accuracy: 0.3737
Epoch 3/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.3467 -
accuracy: 0.4001
Epoch 4/100
2500/2500 [==============================] - 6s 2ms/step - loss: 1.3251 -
accuracy: 0.4128
Epoch 5/100
2500/2500 [==============================] - 6s 2ms/step - loss: 1.3092 -
accuracy: 0.4217
Epoch 6/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.2962 -
accuracy: 0.4297
```

```
Epoch 7/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.2845 -
accuracy: 0.4346
Epoch 8/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.2743 -
accuracy: 0.4410
Epoch 9/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.2620 -
accuracy: 0.4441
Epoch 10/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.2529 -
accuracy: 0.4513
Epoch 11/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.2432 -
accuracy: 0.4537
Epoch 12/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.2347 -
accuracy: 0.4599
Epoch 13/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.2225 -
accuracy: 0.4679
Epoch 14/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.2101 -
accuracy: 0.4720
Epoch 15/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.2001 -
accuracy: 0.4759
Epoch 16/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.1882 -
accuracy: 0.4828
Epoch 17/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.1756 -
accuracy: 0.4897
Epoch 18/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.1647 -
accuracy: 0.4930
Epoch 19/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.1518 -
accuracy: 0.4995
Epoch 20/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.1395 -
accuracy: 0.5057
Epoch 21/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.1255 -
accuracy: 0.5117
Epoch 22/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.1129 -
accuracy: 0.5162
```

```
Epoch 23/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.0989 -
accuracy: 0.5218
Epoch 24/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.0865 -
accuracy: 0.5291
Epoch 25/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.0729 -
accuracy: 0.5355
Epoch 26/100
2500/2500 [==============================] - 6s 2ms/step - loss: 1.0574 -
accuracy: 0.5416
Epoch 27/100
2500/2500 [==============================] - 6s 2ms/step - loss: 1.0434 -
accuracy: 0.5459
Epoch 28/100
2500/2500 [==============================] - 5s 2ms/step - loss: 1.0294 -
accuracy: 0.5529
Epoch 29/100
2500/2500 [==============================] - 6s 2ms/step - loss: 1.0151 -
accuracy: 0.5593
Epoch 30/100
2500/2500 [==============================] - 6s 2ms/step - loss: 1.0029 -
accuracy: 0.5636
Epoch 31/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.9907 -
accuracy: 0.5718
Epoch 32/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.9771 -
accuracy: 0.5756
Epoch 33/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.9637 -
accuracy: 0.5794
Epoch 34/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.9494 -
accuracy: 0.5861
Epoch 35/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.9372 -
accuracy: 0.5915
Epoch 36/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.9253 -
accuracy: 0.5958
Epoch 37/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.9127 -
accuracy: 0.6021
Epoch 38/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.9017 -
accuracy: 0.6076
```

```
Epoch 39/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.8903 -
accuracy: 0.6136
Epoch 40/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.8746 -
accuracy: 0.6175
Epoch 41/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.8676 -
accuracy: 0.6229
Epoch 42/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.8551 -
accuracy: 0.6273
Epoch 43/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.8407 -
accuracy: 0.6319
Epoch 44/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.8361 -
accuracy: 0.6345
Epoch 45/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.8215 -
accuracy: 0.6424
Epoch 46/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.8114 -
accuracy: 0.6469
Epoch 47/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7997 -
accuracy: 0.6517
Epoch 48/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7932 -
accuracy: 0.6544
Epoch 49/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7869 -
accuracy: 0.6567
Epoch 50/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7791 -
accuracy: 0.6583
Epoch 51/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7669 -
accuracy: 0.6663
Epoch 52/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7585 -
accuracy: 0.6684
Epoch 53/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7488 -
accuracy: 0.6715
Epoch 54/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7426 -
accuracy: 0.6745
```

```
Epoch 55/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7315 -
accuracy: 0.6807
Epoch 56/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7270 -
accuracy: 0.6825
Epoch 57/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7207 -
accuracy: 0.6859
Epoch 58/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7110 -
accuracy: 0.6899
Epoch 59/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.7042 -
accuracy: 0.6912
Epoch 60/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6973 -
accuracy: 0.6959
Epoch 61/100
2500/2500 [==============================] - 6s 2ms/step - loss: 0.6890 -
accuracy: 0.6967
Epoch 62/100
2500/2500 [==============================] - 6s 2ms/step - loss: 0.6842 -
accuracy: 0.6993
Epoch 63/100
2500/2500 [==============================] - 6s 2ms/step - loss: 0.6781 -
accuracy: 0.7028
Epoch 64/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6689 -
accuracy: 0.7082
Epoch 65/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6662 -
accuracy: 0.7085
Epoch 66/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6600 -
accuracy: 0.7106
Epoch 67/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6497 -
accuracy: 0.7155
Epoch 68/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6491 -
accuracy: 0.7146
Epoch 69/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6429 -
accuracy: 0.7186
Epoch 70/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6352 -
accuracy: 0.7209
```

```
Epoch 71/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6289 -
accuracy: 0.7239
Epoch 72/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6241 -
accuracy: 0.7262
Epoch 73/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6199 -
accuracy: 0.7300
Epoch 74/100
2500/2500 [==============================] - 6s 2ms/step - loss: 0.6135 -
accuracy: 0.7304
Epoch 75/100
2500/2500 [==============================] - 6s 2ms/step - loss: 0.6074 -
accuracy: 0.7343
Epoch 76/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6060 -
accuracy: 0.7341
Epoch 77/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.6006 -
accuracy: 0.7355
Epoch 78/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5972 -
accuracy: 0.7378
Epoch 79/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5900 -
accuracy: 0.7423
Epoch 80/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5855 -
accuracy: 0.7417
Epoch 81/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5824 -
accuracy: 0.7446
Epoch 82/100
2500/2500 [==============================] - 6s 2ms/step - loss: 0.5760 -
accuracy: 0.7484
Epoch 83/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5710 -
accuracy: 0.7518
Epoch 84/100
2500/2500 [==============================] - 6s 2ms/step - loss: 0.5691 -
accuracy: 0.7511
Epoch 85/100
2500/2500 [==============================] - 6s 2ms/step - loss: 0.5657 -
accuracy: 0.7518
Epoch 86/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5601 -
accuracy: 0.7558
```

```
Epoch 87/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5561 -
accuracy: 0.7576
Epoch 88/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5524 -
accuracy: 0.7576
Epoch 89/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5508 -
accuracy: 0.7578
Epoch 90/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5446 -
accuracy: 0.7627
Epoch 91/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5386 -
accuracy: 0.7642
Epoch 92/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5334 -
accuracy: 0.7657
Epoch 93/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5340 -
accuracy: 0.7661
Epoch 94/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5331 -
accuracy: 0.7683
Epoch 95/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5277 -
accuracy: 0.7703
Epoch 96/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5254 -
accuracy: 0.7710
Epoch 97/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5235 -
accuracy: 0.7717
Epoch 98/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5250 -
accuracy: 0.7696
Epoch 99/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5149 -
accuracy: 0.7756
Epoch 100/100
2500/2500 [==============================] - 5s 2ms/step - loss: 0.5087 -
accuracy: 0.7777
```

[35]:
```
<keras.callbacks.History at 0x7f7ab25b4ed0>
```

[36]:
```
test_loss, test_acc = model.evaluate(X_test_10_word2vec, Y_test_np)
```

```
print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

```
625/625 [==============================] - 1s 2ms/step - loss: 3.7096 -
accuracy: 0.4054
Test Loss: 3.709632396697998
Test Accuracy: 0.4054499864578247
```

1.4.3 Question : What do you conclude by comparing accuracy values you obtain with those obtained in the "'Simple Models" section?

Answer : As compared to simple model (Perceptron test accuracy = 33.365% & SVM test accuacy = 48.51%) first version of FNN (trained on complete word2vec, test accuracy = 50.60%) performed better than the simple model.

Where as the second version of FNN (10 word2vec concatenated, test accuracy = 40.54%) performed better than the perceptron however Simple model SVM accuracy (SVM test accuacy = 48.51%) is better in this case.

1.5 5. Recurrent Neural Networks

1.5.1 a) Simple RNN

[37]:
```
def get_first20_feature_embedding(dataset):
  feature_vec_embedding = []
  for reviews in dataset:
    words = reviews.split()
    max_vocab = 20
    review_embedding = []
    for word in words:
      if len(review_embedding) < max_vocab:
        word_embedd = np.zeros(300)
        if word in wv_google:
          word_embedd += wv_google[word]
          review_embedding.append(word_embedd)
      else:
        break
    if len(review_embedding) < max_vocab:
      while len(review_embedding) != max_vocab:
        review_embedding.append(np.zeros(300))
    feature_vec_embedding.append(review_embedding)
  feature_vec_embedding = np.array(feature_vec_embedding)
  return feature_vec_embedding
```

[38]:
```
X_train_vec_embedding = get_first20_feature_embedding(X_train)
X_test_vec_embedding = get_first20_feature_embedding(X_test)
```

[39]:
```
# building RNN model
from keras.layers import SimpleRNN
model = keras.Sequential()
model.add(SimpleRNN(20, activation='relu'))
model.add(Dense(5, activation='softmax'))
sgd = SGD(0.001)
model.compile(loss="sparse_categorical_crossentropy", optimizer=sgd,␣
↪metrics=["accuracy"])
model.build(input_shape=(None, 20, 300))
model.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 20)                6420

 dense_6 (Dense)             (None, 5)                 105

=================================================================
Total params: 6,525
Trainable params: 6,525
Non-trainable params: 0
_________________________________________________________________
```

[40]:
```
model.fit(X_train_vec_embedding, Y_train_np, epochs=100)
```

```
Epoch 1/100
2500/2500 [==============================] - 12s 4ms/step - loss: 1.6167 -
accuracy: 0.2150
Epoch 2/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.6135 -
accuracy: 0.2167
Epoch 3/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.6124 -
accuracy: 0.2160
Epoch 4/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.6114 -
accuracy: 0.2178
Epoch 5/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.6104 -
accuracy: 0.2190
Epoch 6/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.6095 -
accuracy: 0.2200
Epoch 7/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.6084 -
```

```
accuracy: 0.2214
Epoch 8/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.6071 -
accuracy: 0.2234
Epoch 9/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.6048 -
accuracy: 0.2284
Epoch 10/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.5978 -
accuracy: 0.2396
Epoch 11/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.5581 -
accuracy: 0.2785
Epoch 12/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.4989 -
accuracy: 0.3213
Epoch 13/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.4469 -
accuracy: 0.3476
Epoch 14/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.4062 -
accuracy: 0.3637
Epoch 15/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.3728 -
accuracy: 0.3735
Epoch 16/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.3510 -
accuracy: 0.3809
Epoch 17/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.3373 -
accuracy: 0.3880
Epoch 18/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.3282 -
accuracy: 0.3917
Epoch 19/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.3211 -
accuracy: 0.3953
Epoch 20/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.3154 -
accuracy: 0.3990
Epoch 21/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.3105 -
accuracy: 0.4011
Epoch 22/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.3067 -
accuracy: 0.4038
Epoch 23/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.3025 -
```

```
accuracy: 0.4053
Epoch 24/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2987 -
accuracy: 0.4075
Epoch 25/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2954 -
accuracy: 0.4103
Epoch 26/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2918 -
accuracy: 0.4109
Epoch 27/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2888 -
accuracy: 0.4124
Epoch 28/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2857 -
accuracy: 0.4159
Epoch 29/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2828 -
accuracy: 0.4161
Epoch 30/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2793 -
accuracy: 0.4208
Epoch 31/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2765 -
accuracy: 0.4226
Epoch 32/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2740 -
accuracy: 0.4228
Epoch 33/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2709 -
accuracy: 0.4250
Epoch 34/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.2661 -
accuracy: 0.4301
Epoch 35/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.2620 -
accuracy: 0.4322
Epoch 36/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2579 -
accuracy: 0.4352
Epoch 37/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2540 -
accuracy: 0.4368
Epoch 38/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2505 -
accuracy: 0.4401
Epoch 39/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.2469 -
```

```
accuracy: 0.4421
Epoch 40/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2438 -
accuracy: 0.4446
Epoch 41/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.2404 -
accuracy: 0.4472
Epoch 42/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.2372 -
accuracy: 0.4470
Epoch 43/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.2345 -
accuracy: 0.4515
Epoch 44/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2321 -
accuracy: 0.4528
Epoch 45/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2294 -
accuracy: 0.4538
Epoch 46/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2259 -
accuracy: 0.4562
Epoch 47/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.2246 -
accuracy: 0.4558
Epoch 48/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.2219 -
accuracy: 0.4594
Epoch 49/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2196 -
accuracy: 0.4590
Epoch 50/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.2177 -
accuracy: 0.4606
Epoch 51/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2159 -
accuracy: 0.4602
Epoch 52/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.2139 -
accuracy: 0.4618
Epoch 53/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2117 -
accuracy: 0.4637
Epoch 54/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2103 -
accuracy: 0.4644
Epoch 55/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2090 -
```

```
accuracy: 0.4645
Epoch 56/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2075 -
accuracy: 0.4658
Epoch 57/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2052 -
accuracy: 0.4660
Epoch 58/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2040 -
accuracy: 0.4667
Epoch 59/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2023 -
accuracy: 0.4688
Epoch 60/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.2008 -
accuracy: 0.4703
Epoch 61/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1994 -
accuracy: 0.4700
Epoch 62/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.1980 -
accuracy: 0.4694
Epoch 63/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1971 -
accuracy: 0.4707
Epoch 64/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1958 -
accuracy: 0.4720
Epoch 65/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1947 -
accuracy: 0.4724
Epoch 66/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1927 -
accuracy: 0.4732
Epoch 67/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.1920 -
accuracy: 0.4735
Epoch 68/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.1906 -
accuracy: 0.4731
Epoch 69/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.1893 -
accuracy: 0.4754
Epoch 70/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1878 -
accuracy: 0.4758
Epoch 71/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1871 -
```

```
accuracy: 0.4757
Epoch 72/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1857 -
accuracy: 0.4747
Epoch 73/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1847 -
accuracy: 0.4765
Epoch 74/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1832 -
accuracy: 0.4772
Epoch 75/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1824 -
accuracy: 0.4766
Epoch 76/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1809 -
accuracy: 0.4772
Epoch 77/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1799 -
accuracy: 0.4784
Epoch 78/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1788 -
accuracy: 0.4797
Epoch 79/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1778 -
accuracy: 0.4790
Epoch 80/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1767 -
accuracy: 0.4794
Epoch 81/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1753 -
accuracy: 0.4793
Epoch 82/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1749 -
accuracy: 0.4800
Epoch 83/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1732 -
accuracy: 0.4813
Epoch 84/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1722 -
accuracy: 0.4815
Epoch 85/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1718 -
accuracy: 0.4821
Epoch 86/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1709 -
accuracy: 0.4819
Epoch 87/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1692 -
```

```
accuracy: 0.4827
Epoch 88/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1682 -
accuracy: 0.4859
Epoch 89/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1681 -
accuracy: 0.4837
Epoch 90/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1670 -
accuracy: 0.4844
Epoch 91/100
2500/2500 [==============================] - 11s 4ms/step - loss: 1.1654 -
accuracy: 0.4857
Epoch 92/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1649 -
accuracy: 0.4855
Epoch 93/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1637 -
accuracy: 0.4863
Epoch 94/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1632 -
accuracy: 0.4857
Epoch 95/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1619 -
accuracy: 0.4884
Epoch 96/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1614 -
accuracy: 0.4863
Epoch 97/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1607 -
accuracy: 0.4872
Epoch 98/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1600 -
accuracy: 0.4877
Epoch 99/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1595 -
accuracy: 0.4869
Epoch 100/100
2500/2500 [==============================] - 10s 4ms/step - loss: 1.1581 -
accuracy: 0.4883
```

[40]:
```
<keras.callbacks.History at 0x7f7ab25734d0>
```

[41]:
```python
test_loss, test_acc = model.evaluate(X_test_vec_embedding, Y_test_np)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

```
625/625 [==============================] - 2s 2ms/step - loss: 1.1830 -
accuracy: 0.4791
Test Loss: 1.1829966306686401
Test Accuracy: 0.4791499972343445
```

1.5.2 Question : What do you conclude by comparing accuracy values you obtain with those obtained with feedforward neural network models?

Answer => 1. Part a) FNN model (test accuracy = 50.60%) performed slightly better than RNN (test accuracy = 47.91%) 2. Part b) FNN model (test accuracy = 40.54%) performed was not good, RNN accuracy is better

1.5.3 b) GRU

[42]:
```
# building RNN model
from keras.layers import GRU
model = keras.Sequential()
model.add(GRU(20, activation='relu'))
model.add(Dense(5, activation='softmax'))
sgd = SGD(0.001)
model.compile(loss="sparse_categorical_crossentropy", optimizer=sgd,␣
 ↪metrics=["accuracy"])
model.build(input_shape=(None, 20, 300))
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 20)                19320

 dense_7 (Dense)             (None, 5)                 105

=================================================================
Total params: 19,425
Trainable params: 19,425
Non-trainable params: 0
_________________________________________________________________
```

[43]:
```
model.fit(X_train_vec_embedding, Y_train_np, epochs=100)
```

```
Epoch 1/100
2500/2500 [==============================] - 15s 5ms/step - loss: 1.6116 -
accuracy: 0.1971
Epoch 2/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.6098 -
accuracy: 0.2077
Epoch 3/100
```

```
2500/2500 [==============================] - 14s 5ms/step - loss: 1.6090 -
accuracy: 0.2227
Epoch 4/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.6085 -
accuracy: 0.2233
Epoch 5/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.6081 -
accuracy: 0.2237
Epoch 6/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.6077 -
accuracy: 0.2243
Epoch 7/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.6073 -
accuracy: 0.2246
Epoch 8/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.6069 -
accuracy: 0.2260
Epoch 9/100
2500/2500 [==============================] - 16s 6ms/step - loss: 1.6065 -
accuracy: 0.2257
Epoch 10/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.6061 -
accuracy: 0.2263
Epoch 11/100
2500/2500 [==============================] - 16s 6ms/step - loss: 1.6057 -
accuracy: 0.2274
Epoch 12/100
2500/2500 [==============================] - 16s 6ms/step - loss: 1.6053 -
accuracy: 0.2279
Epoch 13/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.6049 -
accuracy: 0.2284
Epoch 14/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.6045 -
accuracy: 0.2290
Epoch 15/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.6041 -
accuracy: 0.2293
Epoch 16/100
2500/2500 [==============================] - 16s 6ms/step - loss: 1.6037 -
accuracy: 0.2307
Epoch 17/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.6033 -
accuracy: 0.2309
Epoch 18/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.6028 -
accuracy: 0.2312
Epoch 19/100
```

```
2500/2500 [==============================] - 15s 6ms/step - loss: 1.6024 -
accuracy: 0.2323
Epoch 20/100
2500/2500 [==============================] - 16s 6ms/step - loss: 1.6019 -
accuracy: 0.2329
Epoch 21/100
2500/2500 [==============================] - 16s 6ms/step - loss: 1.6014 -
accuracy: 0.2338
Epoch 22/100
2500/2500 [==============================] - 19s 8ms/step - loss: 1.6009 -
accuracy: 0.2345
Epoch 23/100
2500/2500 [==============================] - 16s 6ms/step - loss: 1.6004 -
accuracy: 0.2357
Epoch 24/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5999 -
accuracy: 0.2358
Epoch 25/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5994 -
accuracy: 0.2359
Epoch 26/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5989 -
accuracy: 0.2366
Epoch 27/100
2500/2500 [==============================] - 16s 6ms/step - loss: 1.5984 -
accuracy: 0.2374
Epoch 28/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5978 -
accuracy: 0.2386
Epoch 29/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5973 -
accuracy: 0.2383
Epoch 30/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5967 -
accuracy: 0.2398
Epoch 31/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5961 -
accuracy: 0.2403
Epoch 32/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5955 -
accuracy: 0.2410
Epoch 33/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5948 -
accuracy: 0.2420
Epoch 34/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5941 -
accuracy: 0.2430
Epoch 35/100
```

```
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5934 -
accuracy: 0.2440
Epoch 36/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5926 -
accuracy: 0.2455
Epoch 37/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5918 -
accuracy: 0.2459
Epoch 38/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5910 -
accuracy: 0.2472
Epoch 39/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5900 -
accuracy: 0.2484
Epoch 40/100
2500/2500 [==============================] - 16s 6ms/step - loss: 1.5890 -
accuracy: 0.2506
Epoch 41/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5879 -
accuracy: 0.2512
Epoch 42/100
2500/2500 [==============================] - 16s 6ms/step - loss: 1.5867 -
accuracy: 0.2527
Epoch 43/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.5854 -
accuracy: 0.2542
Epoch 44/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5838 -
accuracy: 0.2561
Epoch 45/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5820 -
accuracy: 0.2583
Epoch 46/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5798 -
accuracy: 0.2605
Epoch 47/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5772 -
accuracy: 0.2627
Epoch 48/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5736 -
accuracy: 0.2661
Epoch 49/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5683 -
accuracy: 0.2706
Epoch 50/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5585 -
accuracy: 0.2754
Epoch 51/100
```

```
2500/2500 [==============================] - 14s 6ms/step - loss: 1.5232 -
accuracy: 0.2939
Epoch 52/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.4260 -
accuracy: 0.3551
Epoch 53/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3925 -
accuracy: 0.3714
Epoch 54/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3725 -
accuracy: 0.3805
Epoch 55/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3572 -
accuracy: 0.3859
Epoch 56/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3449 -
accuracy: 0.3897
Epoch 57/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3347 -
accuracy: 0.3950
Epoch 58/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3264 -
accuracy: 0.3984
Epoch 59/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3199 -
accuracy: 0.4010
Epoch 60/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3143 -
accuracy: 0.4029
Epoch 61/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3096 -
accuracy: 0.4050
Epoch 62/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3055 -
accuracy: 0.4081
Epoch 63/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.3019 -
accuracy: 0.4090
Epoch 64/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2988 -
accuracy: 0.4120
Epoch 65/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2957 -
accuracy: 0.4140
Epoch 66/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2928 -
accuracy: 0.4139
Epoch 67/100
```

```
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2905 -
accuracy: 0.4162
Epoch 68/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2881 -
accuracy: 0.4180
Epoch 69/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2857 -
accuracy: 0.4193
Epoch 70/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2833 -
accuracy: 0.4207
Epoch 71/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2814 -
accuracy: 0.4220
Epoch 72/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2791 -
accuracy: 0.4226
Epoch 73/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.2773 -
accuracy: 0.4245
Epoch 74/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2751 -
accuracy: 0.4255
Epoch 75/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2731 -
accuracy: 0.4265
Epoch 76/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2710 -
accuracy: 0.4286
Epoch 77/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2689 -
accuracy: 0.4291
Epoch 78/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.2668 -
accuracy: 0.4309
Epoch 79/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2644 -
accuracy: 0.4330
Epoch 80/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2620 -
accuracy: 0.4344
Epoch 81/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2595 -
accuracy: 0.4350
Epoch 82/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2570 -
accuracy: 0.4359
Epoch 83/100
```

```
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2541 -
accuracy: 0.4392
Epoch 84/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2510 -
accuracy: 0.4411
Epoch 85/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2475 -
accuracy: 0.4419
Epoch 86/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2441 -
accuracy: 0.4450
Epoch 87/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2409 -
accuracy: 0.4471
Epoch 88/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2378 -
accuracy: 0.4484
Epoch 89/100
2500/2500 [==============================] - 18s 7ms/step - loss: 1.2352 -
accuracy: 0.4484
Epoch 90/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2327 -
accuracy: 0.4509
Epoch 91/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2303 -
accuracy: 0.4525
Epoch 92/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2281 -
accuracy: 0.4541
Epoch 93/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2261 -
accuracy: 0.4543
Epoch 94/100
2500/2500 [==============================] - 15s 6ms/step - loss: 1.2242 -
accuracy: 0.4560
Epoch 95/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2222 -
accuracy: 0.4579
Epoch 96/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2204 -
accuracy: 0.4595
Epoch 97/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2186 -
accuracy: 0.4607
Epoch 98/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2169 -
accuracy: 0.4615
Epoch 99/100
```

```
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2156 -
accuracy: 0.4606
Epoch 100/100
2500/2500 [==============================] - 14s 6ms/step - loss: 1.2142 -
accuracy: 0.4622
```

[43]: <keras.callbacks.History at 0x7f7aa1c46a50>

[44]:
```
test_loss, test_acc = model.evaluate(X_test_vec_embedding, Y_test_np)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

```
625/625 [==============================] - 2s 3ms/step - loss: 1.2230 -
accuracy: 0.4593
Test Loss: 1.223007082939148
Test Accuracy: 0.4593000113964081
```

1.5.4 Question: What do you conclude by comparing accuracy values you obtain with those obtained using simple RNN ?

Answer => Simple RNN (test accuracy = 47.91%) performed better on unseen test data than GRU (test accuracy = 45.93%)

[ ]:
```
```