University of Tehran
Department of ECE
Neural Networks & Deep Learning
MP2

Members:	Milad Mohammadi & Tohid Abdi
IDs:	810100462 & 810100410
Date:	May 25, 2022

Questions List Table

1 Stock Market Prediction

1.1

Import essential libraries:

[1]:
```
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, GRU, SimpleRNN
from keras import optimizers
from tensorflow import keras
import math
from sklearn.metrics import mean_squared_error
```

Import Datasets:

[2]:
```
df1 = pd.read_csv('AAPL.csv' , usecols=[1,2,3,4,5,6], sep='\t')
df2 = pd.read_csv('GOOG.csv' , usecols=[1,2,3,4,5,6], sep='\t')
```

[3]:
```
df1 = df1.rename(columns={'High': 'AAPL_High', 'Low': 'AAPL_Low', 'Open':␣
↪'AAPL_Open', 'Close': 'AAPL_Close', 'Volume': 'AAPL_Volume', 'Adj Close':␣
↪'AAPL_Adj_Close'})
df2 = df2.rename(columns={'High': 'GOOG_High', 'Low': 'GOOG_Low', 'Open':␣
↪'GOOG_Open', 'Close': 'GOOG_Close', 'Volume': 'GOOG_Volume', 'Adj Close':␣
↪'GOOG_Adj_Close'})
```

Dataset:

[4]:
```
df = pd.concat([df1,df2], axis=1)
```

[5]:
```
df
```

[5]:
```
        AAPL_High    AAPL_Low   AAPL_Open  AAPL_Close  AAPL_Volume  \
0       30.642857   30.340000   30.490000   30.572857  123432400.0
1       30.798571   30.464285   30.657143   30.625713  150476200.0
2       30.747143   30.107143   30.625713   30.138571  138040000.0
3       30.285715   29.864286   30.250000   30.082857  119282800.0
4       30.285715   29.865715   30.042856   30.282858  111902700.0
...           ...         ...         ...         ...          ...
2259   151.550003  146.589996  148.149994  146.830002   37169200.0
2260   157.229996  146.720001  148.300003  157.169998   58582500.0
2261   156.770004  150.070007  155.839996  156.149994   53117100.0
2262   158.520004  154.550003  157.500000  156.229996   42291400.0
2263   159.360001  156.479996  158.529999  157.740005   35003500.0
```

```
      AAPL_Adj_Close    GOOG_High     GOOG_Low    GOOG_Open   GOOG_Close  \
0          26.601469   313.579620   310.954468   312.304413   312.204773
1          26.647457   312.747742   309.609497   312.418976   310.829926
2          26.223597   311.761444   302.047852   311.761444   302.994293
3          26.175119   303.861053   295.218445   303.562164   295.940735
4          26.349140   300.498657   293.455048   294.894653   299.885956
...              ...          ...          ...          ...          ...
2259      144.656540  1003.539978   970.109985   973.900024   976.219971
2260      154.843475  1040.000000   983.000000   989.010010  1039.459961
2261      153.838562  1043.890015   997.000000  1017.150024  1043.880005
2262      153.917389  1055.560059  1033.099976  1049.619995  1037.079956
2263      155.405045  1052.699951  1023.590027  1050.959961  1035.609985

      GOOG_Volume  GOOG_Adj_Close
0       3927000.0      312.204773
1       6031900.0      310.829926
2       7987100.0      302.994293
3      12876600.0      295.940735
4       9483900.0      299.885956
...           ...             ...
2259    1590300.0      976.219971
2260    2373300.0     1039.459961
2261    2109800.0     1043.880005
2262    1414800.0     1037.079956
2263    1493300.0     1035.609985

[2264 rows x 12 columns]
```

Close column:

[6]:
```python
close = df[['AAPL_Close', 'GOOG_Close']]
```

Preproccess data:

[7]:
```python
dataset = df.values
close = close.values

dataset = dataset.astype('float32')
close = close.astype('float32')
```

Data normalization:

[8]:
```python
scaler = MinMaxScaler(feature_range=(0, 1))
datasetscaled = scaler.fit_transform(dataset)

closescaler = MinMaxScaler(feature_range=(0, 1))
closescaled = closescaler.fit_transform(close)
```

train-test split:

```
[9]: train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = datasetscaled[0:train_size,:], datasetscaled[train_size:
 ↪len(dataset),:]
y_train, y_test = closescaled[0:train_size,:], closescaled[train_size:
 ↪len(dataset),:]
print(len(train), len(test))
```

```
1516 748
```

Getting data ready:

```
[10]: def create_dataset(dataset, y, look_back):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
                a = dataset[i:(i+look_back)]
                dataX.append(a)
                dataY.append(y[i + look_back])
        return numpy.array(dataX), numpy.array(dataY)
```

```
[11]: look_back = 30
trainX, trainY = create_dataset(train, y_train, look_back)
testX, testY = create_dataset(test, y_test, look_back)
```

early-stop callback:

```
[12]: callback = keras.callbacks.EarlyStopping(monitor='loss', patience=10,␣
 ↪restore_best_weights=True)
```

Define LSTM model and fit data:

```
[13]: lstm = Sequential()
lstm.add(LSTM(64, input_shape=(look_back, len(df.columns)),␣
 ↪return_sequences=True))
lstm.add(LSTM(64))
lstm.add(Dense(2))
lstm.compile(loss='MSE', optimizer = keras.optimizers.Adam())
history_lstm = lstm.fit(trainX, trainY, epochs=4000, batch_size=16, verbose=2,␣
 ↪callbacks = [callback])
```

```
Epoch 1/4000
93/93 - 5s - loss: 0.0029 - 5s/epoch - 56ms/step
...
Epoch 83/4000
93/93 - 2s - loss: 5.6303e-05 - 2s/epoch - 18ms/step
```

Predict by model:

```
[14]: testPredict = lstm.predict(testX)
# invert predictions
testPredict = closescaler.inverse_transform(testPredict)
recovered_testY = closescaler.inverse_transform(testY)
```

```
[15]: testPredict_AAPL, testPredict_GOOG, recovered_testY_APPL, recovered_testY_GOOG = [], [], [], []

for i in range(len(testPredict)):
    testPredict_AAPL.append(testPredict[i,0])
    testPredict_GOOG.append(testPredict[i,1])

    recovered_testY_APPL.append(recovered_testY[i,0])
    recovered_testY_GOOG.append(recovered_testY[i,1])


testPredict_AAPL = numpy.asanyarray(testPredict_AAPL)
testPredict_GOOG = numpy.asanyarray(testPredict_GOOG)
recovered_testY_APPL = numpy.asarray(recovered_testY_APPL)
recovered_testY_GOOG = numpy.asarray(recovered_testY_GOOG)
```

Results:

```
[40]: plt.figure(figsize=[25,10])
plt.plot(recovered_testY_APPL)
plt.plot(testPredict_AAPL)
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```

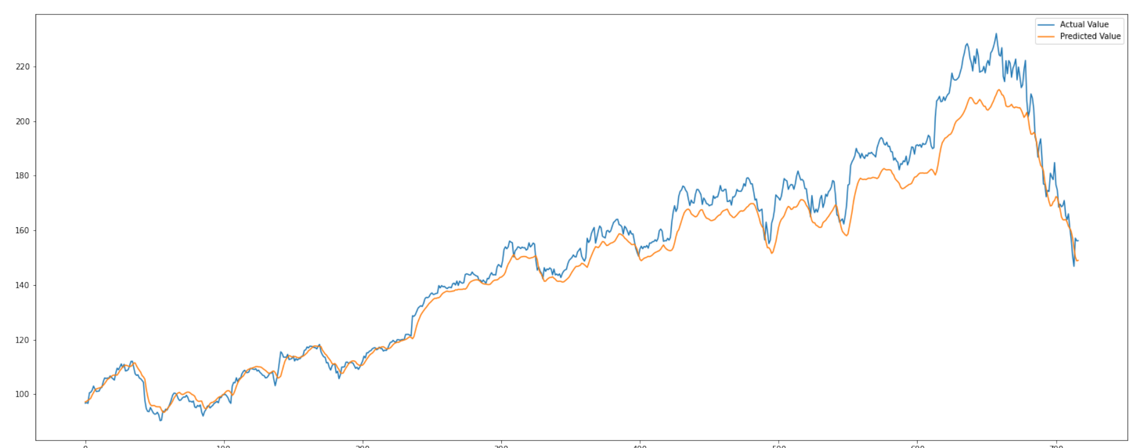

Figure 1: AAPL prediction with LSTM model

```
[17]: plt.figure(figsize=[25,10])
      plt.plot(recovered_testY_GOOG)
      plt.plot(testPredict_GOOG)
      plt.legend(['Actual Value', 'Predicted Value'])
      plt.show()
```

Figure 2: GOOG prediction with LSTM model

Loss Function:

```
[18]: plt.plot(history_lstm.history['loss'])
      plt.show()
```

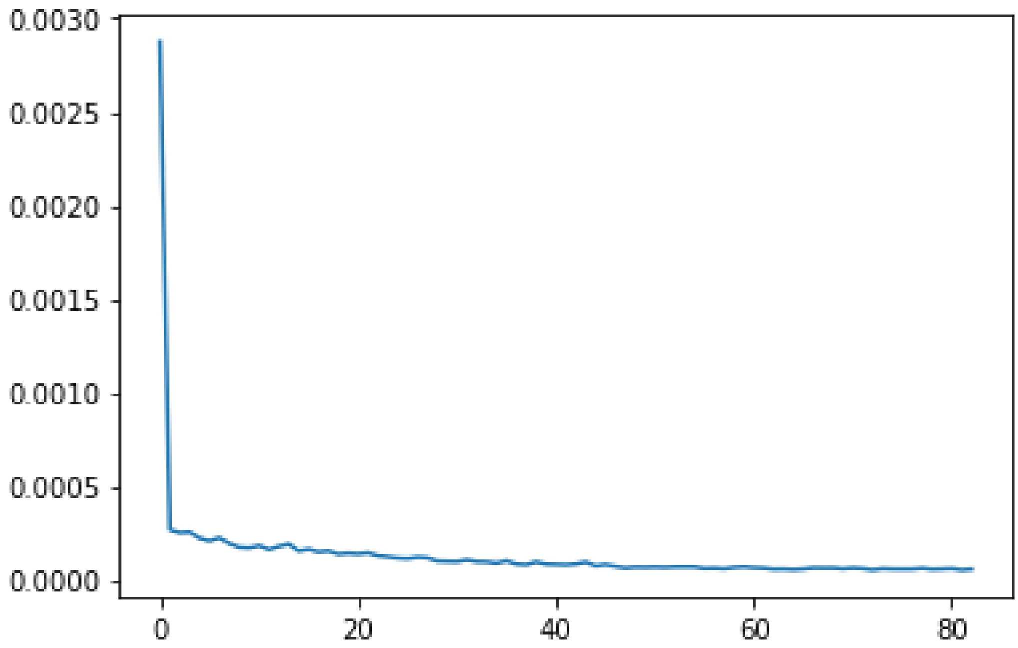

Figure 3: Loss of LSTM model

GRU Model:

[19]:
```
gru = Sequential()
gru.add(GRU(64, input_shape=(look_back, len(df.columns)), return_sequences=True))
gru.add(GRU(64))
gru.add(Dense(2))
gru.compile(loss='MSE', optimizer = keras.optimizers.Adam())
history_gru = gru.fit(trainX, trainY, epochs=4000, batch_size=16, verbose=2,␣
 ↪callbacks = [callback])
```

```
Epoch 1/4000
93/93 - 6s - loss: 0.0019 - 6s/epoch - 64ms/step
...
Epoch 34/4000
93/93 - 2s - loss: 5.8971e-05 - 2s/epoch - 19ms/step
```

Predict by model:

[20]:
```
testPredict = gru.predict(testX)
# invert predictions
testPredict = closescaler.inverse_transform(testPredict)
```

[21]:
```
testPredict_AAPL, testPredict_GOOG = [], []

for i in range(len(testPredict)):
    testPredict_AAPL.append(testPredict[i,0])
    testPredict_GOOG.append(testPredict[i,1])



testPredict_AAPL = numpy.asanyarray(testPredict_AAPL)
testPredict_GOOG = numpy.asanyarray(testPredict_GOOG)
```

Results:

[22]:
```
plt.figure(figsize=[25,10])
plt.plot(recovered_testY_APPL)
plt.plot(testPredict_AAPL)
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```

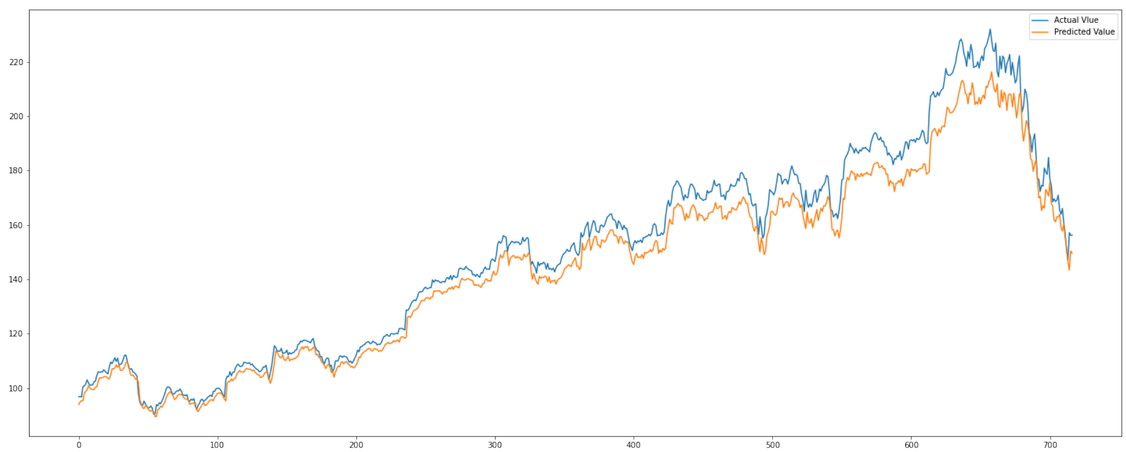

Figure 4: AAPL prediction with GRU model

[23]:
```
plt.figure(figsize=[25,10])
plt.plot(recovered_testY_GOOG)
plt.plot(testPredict_GOOG)
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```

Figure 5: GOOG prediction with GRU model

Loss Function:

[24]:
```
plt.plot(history_gru.history['loss'])
plt.show()
```

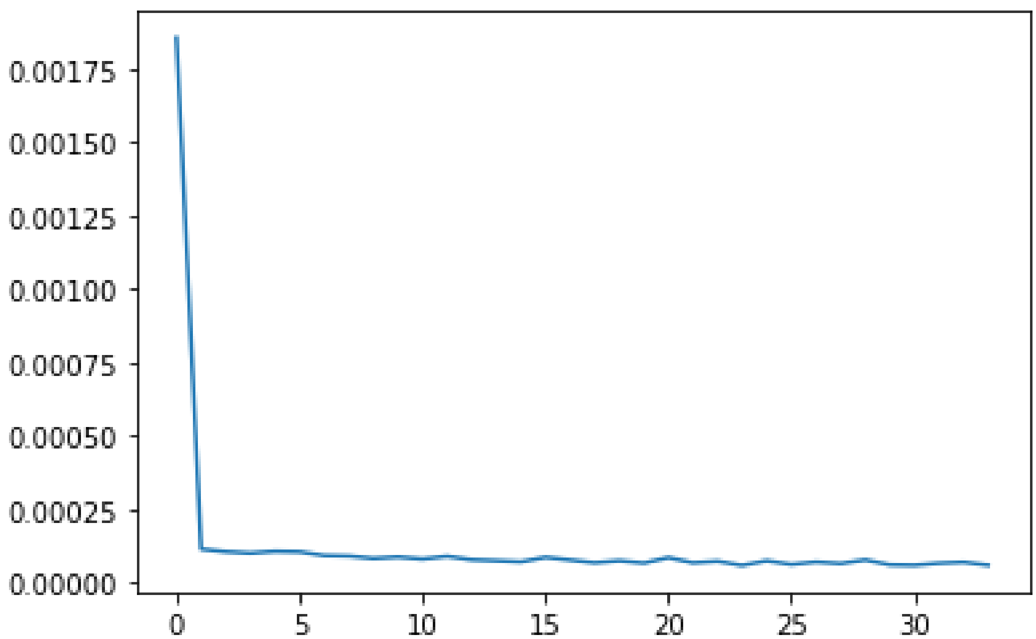

Figure 6: Loss of GRU model

RNN Model:

[25]:
```
rnn = Sequential()
rnn.add(SimpleRNN(64, input_shape=(look_back, len(df.columns)),␣
 ↪return_sequences=True))
rnn.add(SimpleRNN(64))
rnn.add(Dense(2))
rnn.compile(loss='MSE', optimizer = keras.optimizers.Adam())
history_rnn = rnn.fit(trainX, trainY, epochs=4000, batch_size=16, verbose=2,␣
 ↪callbacks = [callback])
```

```
Epoch 1/4000
93/93 - 2s - loss: 0.0119 - 2s/epoch - 18ms/step
...
Epoch 44/4000
93/93 - 1s - loss: 1.6388e-04 - 853ms/epoch - 9ms/step
```

Predict by model:

[26]:
```
testPredict = rnn.predict(testX)
# invert predictions
testPredict = closescaler.inverse_transform(testPredict)
```

[27]:
```
testPredict_AAPL, testPredict_GOOG = [], []

for i in range(len(testPredict)):
    testPredict_AAPL.append(testPredict[i,0])
    testPredict_GOOG.append(testPredict[i,1])



testPredict_AAPL = numpy.asanyarray(testPredict_AAPL)
testPredict_GOOG = numpy.asanyarray(testPredict_GOOG)
```

Results:

[28]:
```
plt.figure(figsize=[25,10])
plt.plot(recovered_testY_APPL)
plt.plot(testPredict_AAPL)
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```

Figure 7: AAPL prediction with Simple RNN model

[39]:
```
plt.figure(figsize=[25,10])
plt.plot(recovered_testY_GOOG)
plt.plot(testPredict_GOOG)
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```

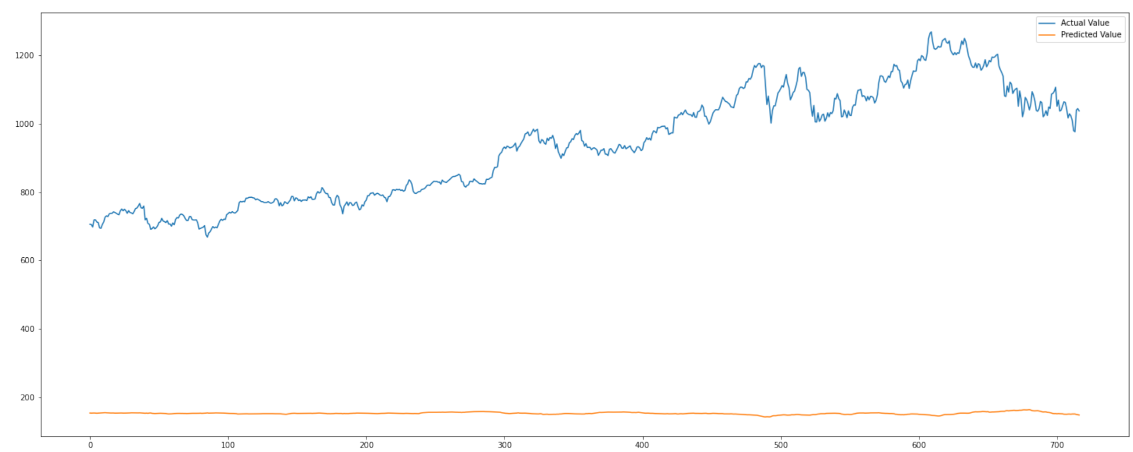

Figure 8: GOOG prediction with Simple RNN model

Loss Function:

```
[30]: plt.plot(history_rnn.history['loss'])
      plt.show()
```

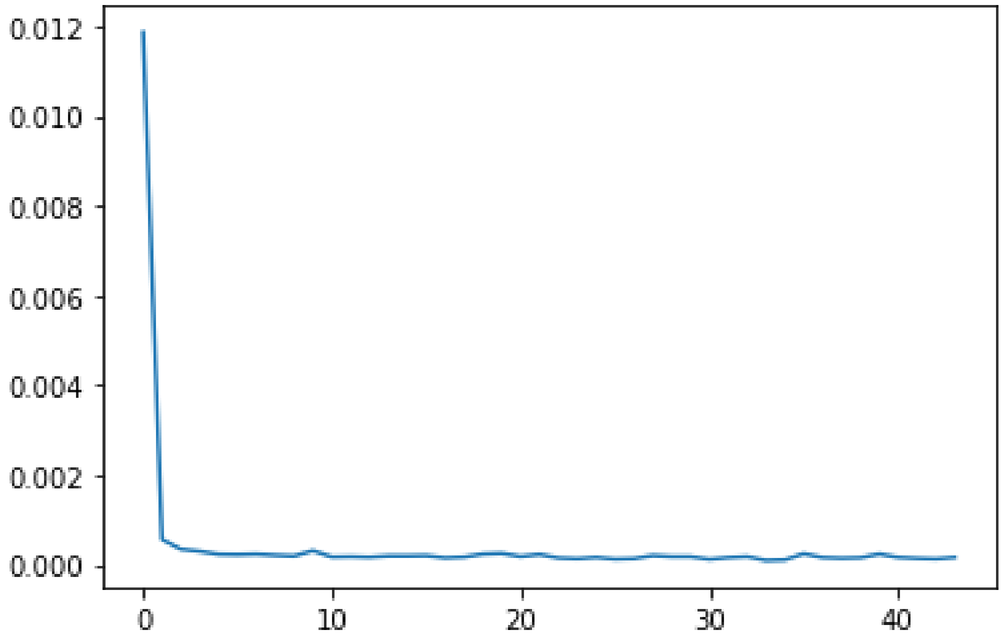

Figure 9: Loss of Simple RNN model

Simple RNN cell has lower accuracy comparing with GRU and LSTM, but it's Faster than them. The difference betwwen training time of these architectures rises from their different number of 'gates', in other words number of trainable matrices.

GRU model is faster than LSTM and it has nearly same accuracy.

1.2

GRU Model:

```
[31]: gru_mape = Sequential()
      gru_mape.add(GRU(64, input_shape=(look_back, len(df.columns)),␣
       ↪return_sequences=True))
      gru_mape.add(GRU(64))
      gru_mape.add(Dense(2))
      gru_mape.compile(loss='MAPE', optimizer = keras.optimizers.Adam())
      history_gru_mape = gru_mape.fit(trainX, trainY, epochs=4000, batch_size=16,␣
       ↪verbose=2, callbacks = [callback])
```

```
Epoch 1/4000
93/93 - 5s - loss: 23154.2383 - 5s/epoch - 49ms/step
...
Epoch 43/4000
93/93 - 2s - loss: 8575.6016 - 2s/epoch - 25ms/step
```

Predict by model:

```
[32]: testPredict = gru_mape.predict(testX)
      # invert predictions
      testPredict = closescaler.inverse_transform(testPredict)
```

```
[33]: testPredict_AAPL, testPredict_GOOG = [], []

      for i in range(len(testPredict)):
          testPredict_AAPL.append(testPredict[i,0])
          testPredict_GOOG.append(testPredict[i,1])



      testPredict_AAPL = numpy.asanyarray(testPredict_AAPL)
      testPredict_GOOG = numpy.asanyarray(testPredict_GOOG)
```

Results:

```
[35]: plt.figure(figsize=[25,10])
      plt.plot(recovered_testY_APPL)
      plt.plot(testPredict_AAPL)
      plt.legend(['Actual Value', 'Predicted Value'])
      plt.show()
```

Figure 10: AAPL prediction with GRU model with MAPE loss function

```
[38]: plt.figure(figsize=[25,10])
      plt.plot(recovered_testY_GOOG)
      plt.plot(testPredict_GOOG)
      plt.legend(['Actual Value', 'Predicted Value'])
      plt.show()
```

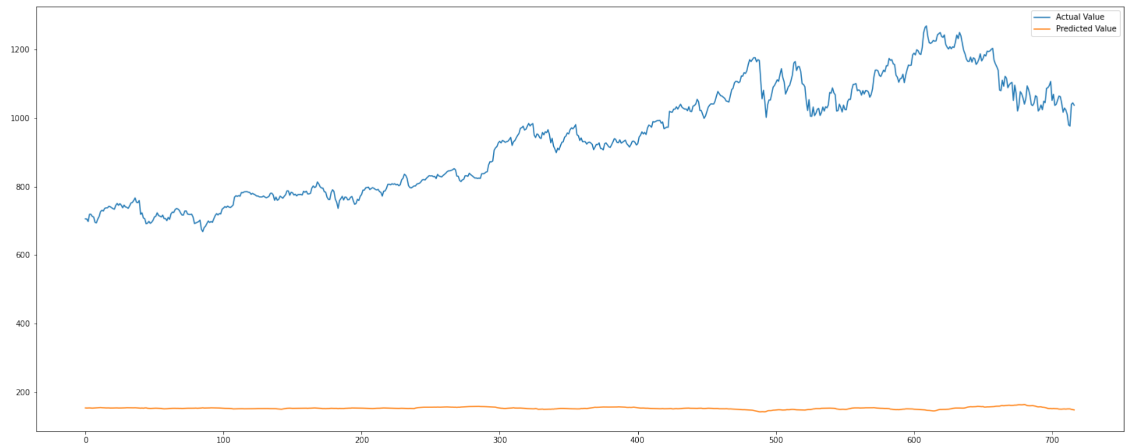

Figure 11: GOOG prediction with GRU model with MAPE loss function

Loss Function:

```
[37]: plt.plot(history_gru_mape.history['loss'])
      plt.show()
```

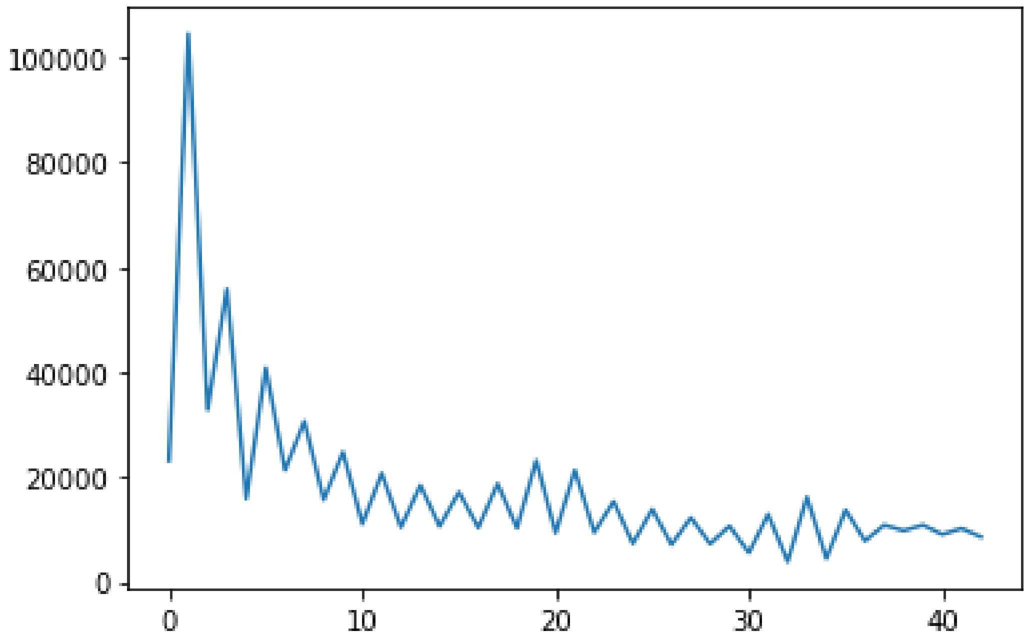

Figure 12: Loss of model with MAPE loss function

Model with MSE loss function has more accuracy in compare with model with MAPE loss function.

$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \qquad MAPE = \frac{1}{n}\sum_{i=1}^{n}|\frac{Y_i - \hat{Y}_i}{Y_i}|$

1.3

LSTM model with ADAgrad optimizer:

[43]:
```
lstm_adagrad = Sequential()
lstm_adagrad.add(LSTM(64, input_shape=(look_back, len(df.columns)),␣
↪return_sequences=True))
lstm_adagrad.add(LSTM(64))
lstm_adagrad.add(Dense(2))
lstm_adagrad.compile(loss='MSE', optimizer = keras.optimizers.Adagrad())
history_lstm_adagrad = lstm_adagrad.fit(trainX, trainY, epochs=400,␣
↪batch_size=16, verbose=2, callbacks = [callback])
```

```
Epoch 1/400
93/93 - 7s - loss: 0.0376 - 7s/epoch - 77ms/step
...
Epoch 400/400
93/93 - 2s - loss: 2.2792e-04 - 2s/epoch - 20ms/step
```

Predict by model:

```
[44]: testPredict = lstm_adagrad.predict(testX)
      # invert predictions
      testPredict = closescaler.inverse_transform(testPredict)
      recovered_testY = closescaler.inverse_transform(testY)
```

```
[45]: testPredict_AAPL, testPredict_GOOG, recovered_testY_APPL, recovered_testY_GOOG =␣
       ↪[], [], [], []

      for i in range(len(testPredict)):
          testPredict_AAPL.append(testPredict[i,0])
          testPredict_GOOG.append(testPredict[i,1])

          recovered_testY_APPL.append(recovered_testY[i,0])
          recovered_testY_GOOG.append(recovered_testY[i,1])


      testPredict_AAPL = numpy.asanyarray(testPredict_AAPL)
      testPredict_GOOG = numpy.asanyarray(testPredict_GOOG)
      recovered_testY_APPL = numpy.asarray(recovered_testY_APPL)
      recovered_testY_GOOG = numpy.asarray(recovered_testY_GOOG)
```

Results:

```
[46]: plt.figure(figsize=[25,10])
      plt.plot(recovered_testY_APPL)
      plt.plot(testPredict_AAPL)
      plt.legend(['Actual Value', 'Predicted Value'])
      plt.show()
```

Figure 13: AAPL prediction with LSTM model with Adagrad optimizer

[47]:
```
plt.figure(figsize=[25,10])
plt.plot(recovered_testY_GOOG)
plt.plot(testPredict_GOOG)
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```

Figure 14: GOOG prediction with LSTM model with Adagrad optimizer

Loss Function:

[56]:
```
plt.plot(history_lstm_adagrad.history['loss'])
plt.show()
```

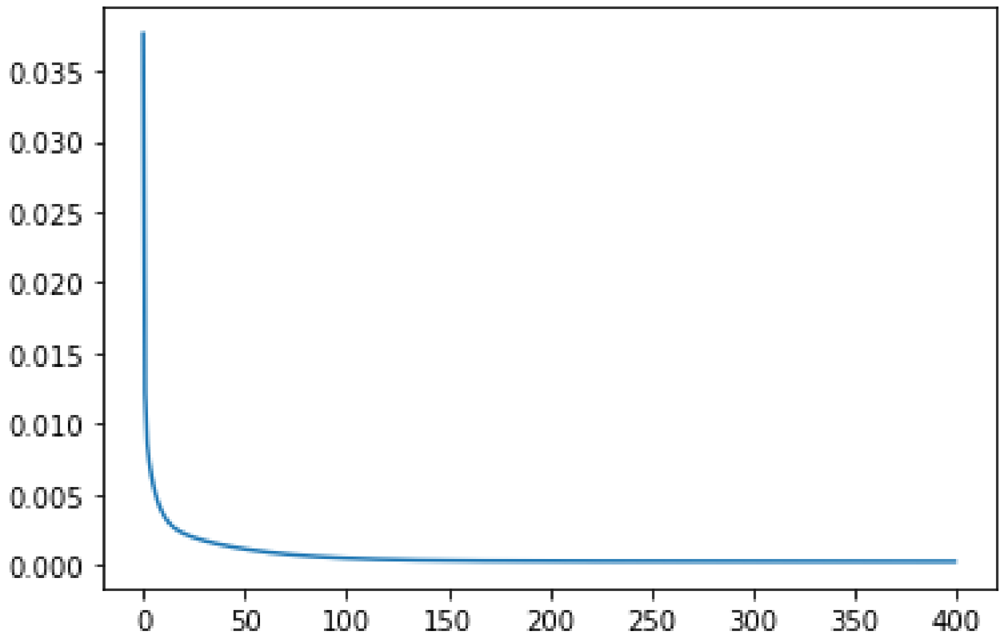

Figure 15: Loss of model with Adagrad optimizer

LSTM model with RMSprop optimizer:

[50]:
```
lstm_rmsprop = Sequential()
lstm_rmsprop.add(LSTM(64, input_shape=(look_back, len(df.columns)),␣
 ↪return_sequences=True))
lstm_rmsprop.add(LSTM(64))
lstm_rmsprop.add(Dense(2))
lstm_rmsprop.compile(loss='MSE', optimizer = keras.optimizers.RMSprop())
history_lstm_rmsprop = lstm_rmsprop.fit(trainX, trainY, epochs=400,␣
 ↪batch_size=16, verbose=2, callbacks = [callback])
```

```
Epoch 1/400
93/93 - 8s - loss: 0.0025 - 8s/epoch - 86ms/step
...
Epoch 211/400
93/93 - 2s - loss: 5.9057e-05 - 2s/epoch - 22ms/step
```

Predict by model:

[51]:
```
testPredict = lstm_rmsprop.predict(testX)
# invert predictions
testPredict = closescaler.inverse_transform(testPredict)
recovered_testY = closescaler.inverse_transform(testY)
```

[52]:
```
testPredict_AAPL, testPredict_GOOG, recovered_testY_APPL, recovered_testY_GOOG =␣
↪[], [], [], []

for i in range(len(testPredict)):
    testPredict_AAPL.append(testPredict[i,0])
    testPredict_GOOG.append(testPredict[i,1])

    recovered_testY_APPL.append(recovered_testY[i,0])
    recovered_testY_GOOG.append(recovered_testY[i,1])


testPredict_AAPL = numpy.asanyarray(testPredict_AAPL)
testPredict_GOOG = numpy.asanyarray(testPredict_GOOG)
recovered_testY_APPL = numpy.asarray(recovered_testY_APPL)
recovered_testY_GOOG = numpy.asarray(recovered_testY_GOOG)
```

Results:

[53]:
```
plt.figure(figsize=[25,10])
plt.plot(recovered_testY_APPL)
plt.plot(testPredict_AAPL)
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```

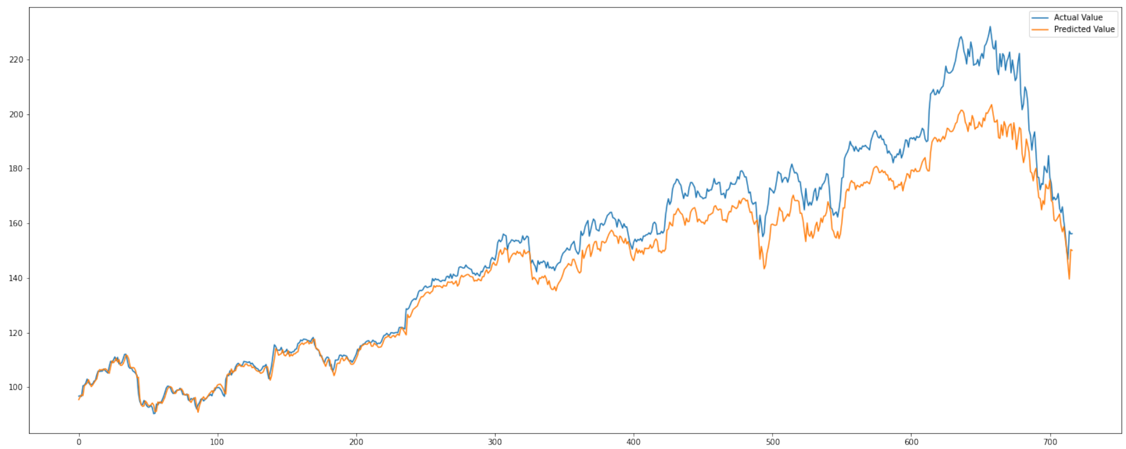

Figure 16: AAPL prediction with LSTM model with RMSprop optimizer

[54]:
```
plt.figure(figsize=[25,10])
plt.plot(recovered_testY_GOOG)
plt.plot(testPredict_GOOG)
plt.legend(['Actual Value', 'Predicted Value'])
```

```
plt.show()
```

Figure 17: GOOG prediction with LSTM model with RMSprop optimizer

Loss Function:

[55]:
```
plt.plot(history_lstm_rmsprop.history['loss'])
plt.show()
```

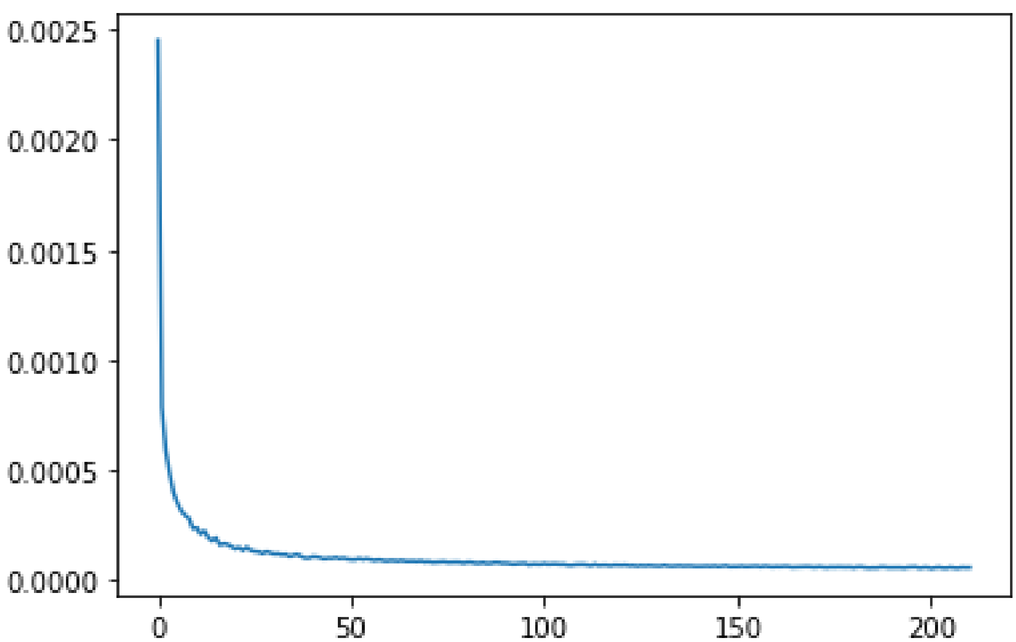

Figure 18: Loss of LSTM model with RMSprop optimizer

Adam is best optimizer and produces best results. Convergence speed of Adagrad is low and it reduces loss value with low speed. RMSprop converges to expected result but it's convergence speed is so low in compare with Adam.

1.4

Define LSTM model with dropout:

[65]:
```
lstm_dropout = Sequential()
lstm_dropout.add(LSTM(64, input_shape=(look_back, len(df.columns)),␣
 ↪return_sequences=True, recurrent_dropout=0.2))
lstm_dropout.add(LSTM(64))
lstm_dropout.add(Dense(2))
lstm_dropout.compile(loss='MSE', optimizer = keras.optimizers.Adam())
history_lstm_dropout = lstm_dropout.fit(trainX, trainY, epochs=4000,␣
 ↪batch_size=16, verbose=2, callbacks = [callback])
```

```
Epoch 1/4000
93/93 - 8s - loss: 0.0032 - 8s/epoch - 88ms/step
...
Epoch 102/4000
93/93 - 3s - loss: 6.1983e-05 - 3s/epoch - 27ms/step
```

Predict by model:

[66]:
```
testPredict = lstm_dropout.predict(testX)
# invert predictions
testPredict = closescaler.inverse_transform(testPredict)
recovered_testY = closescaler.inverse_transform(testY)
```

[67]:
```
testPredict_AAPL, testPredict_GOOG, recovered_testY_APPL, recovered_testY_GOOG =␣
 ↪[], [], [], []

for i in range(len(testPredict)):
    testPredict_AAPL.append(testPredict[i,0])
    testPredict_GOOG.append(testPredict[i,1])

    recovered_testY_APPL.append(recovered_testY[i,0])
    recovered_testY_GOOG.append(recovered_testY[i,1])


testPredict_AAPL = numpy.asanyarray(testPredict_AAPL)
testPredict_GOOG = numpy.asanyarray(testPredict_GOOG)
recovered_testY_APPL = numpy.asarray(recovered_testY_APPL)
```

```
recovered_testY_GOOG = numpy.asarray(recovered_testY_GOOG)
```

Results:

[68]:
```
plt.figure(figsize=[25,10])
plt.plot(recovered_testY_APPL)
plt.plot(testPredict_AAPL)
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```

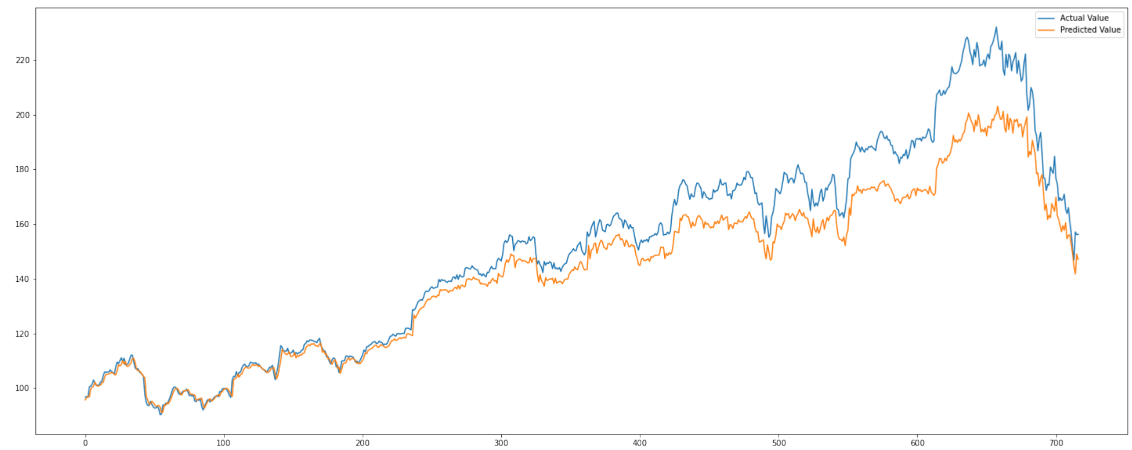

Figure 19: AAPL prediction with LSTM model with recurrent dropout

[69]:
```
plt.figure(figsize=[25,10])
plt.plot(recovered_testY_GOOG)
plt.plot(testPredict_GOOG)
plt.legend(['Actual Value', 'Predicted Value'])
plt.show()
```

Figure 20: GOOG prediction with LSTM model with recurrent dropout

Loss Function:

[70]:
```
plt.plot(history_lstm_dropout.history['loss'])
plt.show()
```

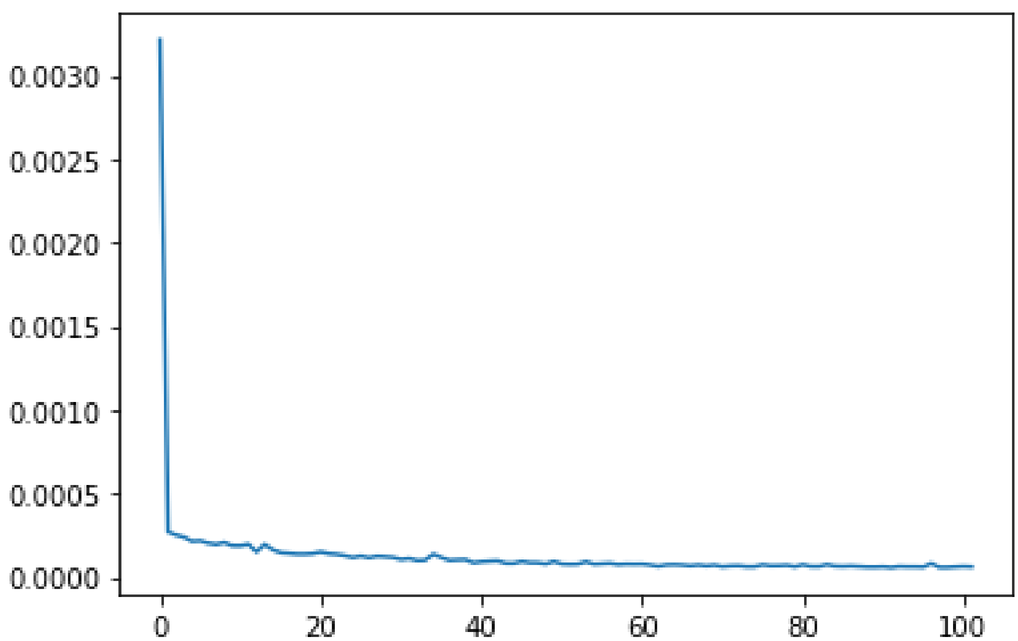

Figure 21: Loss of LSTM model with recurrent dropout

Using normal dropout increased model loss and recurrent dropout has not significant effect on model performance.

2 Text Generation

2.1

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

Importing Dependencies

[2]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM, GRU
from keras.utils import np_utils
```

Reading dataset.txt

[3]:
```
path_to_file = "/content/drive/MyDrive/dataset.txt"
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
print ('{} characters'.format(len(text)))
```

```
1107542 characters
```

Getting a list of unique characters

[4]:
```
vocab = sorted(set(text))
print ('{} unique characters'.format(len(vocab)))
```

```
80 unique characters
```

Pre-processing

[5]:
```
char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)

text_as_int = np.array([char2idx[c] for c in text])
```

[6]:
```
for char,_ in zip(char2idx, range(20)):
    print('  {:4s}: {:3d},'.format(repr(char), char2idx[char]))
```

```
  '\t':   0,
  '\n':   1,
  ' ' :   2,
  '!' :   3,
  '"' :   4,
```

```
  "'" :   5,
  '(' :   6,
  ')' :   7,
  ',' :   8,
  '-' :   9,
  '.' :  10,
  '/' :  11,
  '0' :  12,
  '1' :  13,
  '2' :  14,
  '3' :  15,
  '4' :  16,
  '6' :  17,
  '7' :  18,
  '9' :  19,
```

Creating training examples and targets

[7]:
```
seq_length = 200
examples_per_epoch = len(text)//(seq_length+1)

char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)

for i in char_dataset.take(5):
  print(idx2char[i.numpy()] , end = "")
```

```
HARRY
```

[8]:
```
sequences = char_dataset.batch(seq_length+1, drop_remainder=True)

for item in sequences.take(5):
  print(repr(''.join(idx2char[item.numpy()])))
```

```
'HARRY POTTER AND THE GOBLET OF FIRE\n\nCHAPTER ONE - THE RIDDLE HOUSE\n\n\tThe
villagers of Little Hangleron still called it "the Riddle House," even though it
had been many years since the Riddle family ha'
'd lived there.  It stood on a hill overlooking the village, some of its windows
boarded, tiles missing from its roof, and ivy spreading unchecked over its face.
Once a fine-looking manor, and easily t'
'he largest and grandest building for miles around, the Riddle House was now
damp, derelict, and unoccupied.\n\tThe Little Hagletons all agreed that the old
house was "creepy."  Half a century ago, someth'
'ing strange and horrible had happened there, something that the older
inhabitants of the village still liked to discuss when topics for gossip were
scarce.  The story had been picked over so many times'
', and had been embroidered in so many places, that nobody was quite sure what
the truth was anymore.  Every version of the tale, however, started in the same
place:  Fifty years before, at daybreak on '
```

Mapping function

[9]:
```
def split_input_target(chunk):
    input_text = chunk[:-1]
    target_text = chunk[1:]
    return input_text, target_text

dataset = sequences.map(split_input_target)
```

Creating training batches

[10]:
```
BATCH_SIZE = 64
BUFFER_SIZE = 10000
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)
dataset
```

[10]:
```
<BatchDataset element_spec=(TensorSpec(shape=(64, 200), dtype=tf.int64,
name=None), TensorSpec(shape=(64, 200), dtype=tf.int64, name=None))>
```

[11]:
```
vocab_size = len(vocab)

embedding_dim = 256

rnn_units = 1500
```

Define the model

1. Embedding layer : The input layer. A trainable lookup table that will map the numbers of each character to a vector with embedding_dim dimensions.

2. LSTM layer : A type of RNN with size units=rnn_units.

3. Dense layer : The output layer, with vocab_size outputs and 'RELU' as the activation fuction.

4. Dropout layer : Benifits regularisation and prevents overfitting

[12]:
```
def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
  model = tf.keras.Sequential([

    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                              batch_input_shape=[batch_size, None]),

    tf.keras.layers.LSTM(rnn_units,
                        return_sequences=True,
                        stateful=True,
                        recurrent_initializer='glorot_uniform'),

    tf.keras.layers.Dense(vocab_size,activation='relu'),

    tf.keras.layers.Dropout(0.2)
```

```
  ])
  return model
```

[13]:
```
model = build_model(
  vocab_size = len(vocab),
  embedding_dim=embedding_dim,
  rnn_units=rnn_units,
  batch_size=BATCH_SIZE)
```

[14]:
```
for input_example_batch, target_example_batch in dataset.take(1):
  example_batch_predictions = model(input_example_batch)
  print(example_batch_predictions.shape, "(batch_size, sequence_length,␣
 ↪vocab_size)")
```

```
(64, 200, 80) (batch_size, sequence_length, vocab_size)
```

[15]:
```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (64, None, 256)           20480

 lstm (LSTM)                 (64, None, 1500)          10542000

 dense (Dense)               (64, None, 80)            120080

 dropout (Dropout)           (64, None, 80)            0

=================================================================
Total params: 10,682,560
Trainable params: 10,682,560
Non-trainable params: 0
_________________________________________________________________
```

[16]:
```
def loss(labels, logits):
  return tf.keras.losses.sparse_categorical_crossentropy(labels, logits,␣
 ↪from_logits=True)

example_batch_loss  = loss(target_example_batch, example_batch_predictions)
print("Prediction shape: ", example_batch_predictions.shape, "(batch_size,␣
 ↪sequence_length, vocab_size)")
print("scalar_loss:      ", example_batch_loss.numpy().mean())
```

```
Prediction shape:  (64, 200, 80) (batch_size, sequence_length, vocab_size)
scalar_loss:       4.381857
```

```
[17]: callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
      model.compile(optimizer='adam', loss=loss, metrics = 'accuracy')
```

```
[19]: history = model.fit(dataset, epochs= 50, callbacks=[callback])
```

```
Epoch 1/50
86/86 [==============================] - 26s 259ms/step - loss: 3.9377 -
accuracy: 0.1216
...
Epoch 50/50
86/86 [==============================] - 27s 297ms/step - loss: 1.2430 -
accuracy: 0.7377
```

```
[20]: plt.plot(history.history['loss'])
      plt.title('Loss per epoch')
      plt.ylabel('Loss')
      plt.xlabel('epoch')
      plt.legend(['train'], loc='upper right')
      plt.show()
```

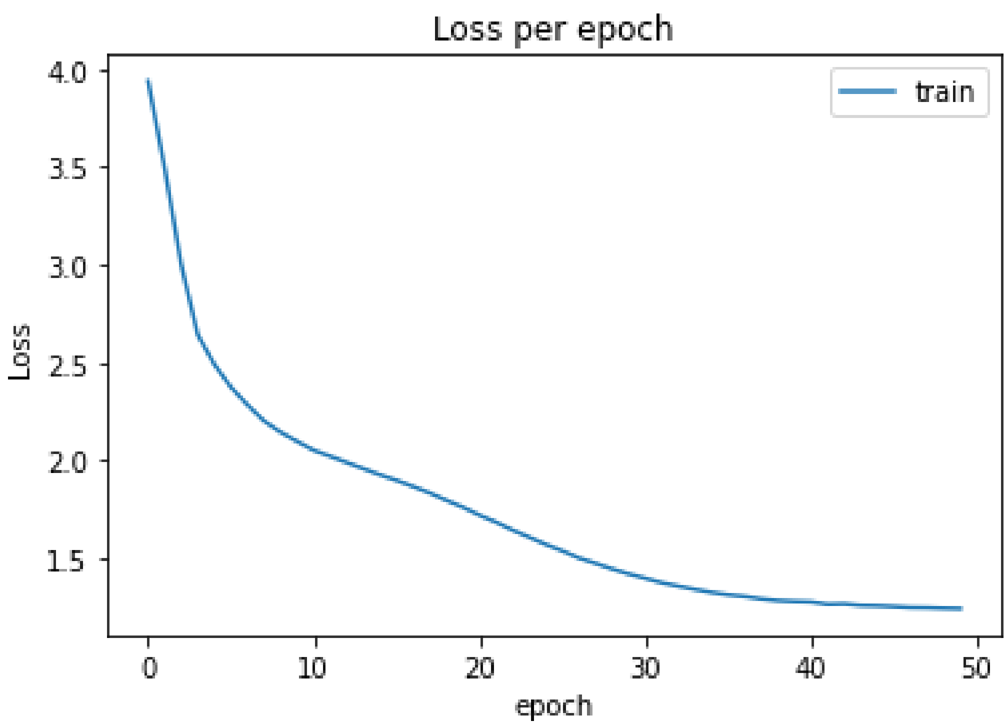

Figure 22: Loss of model

[21]:
```
plt.plot(history.history['accuracy'])
plt.title('accuracy per epoch')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper right')
plt.show()
```

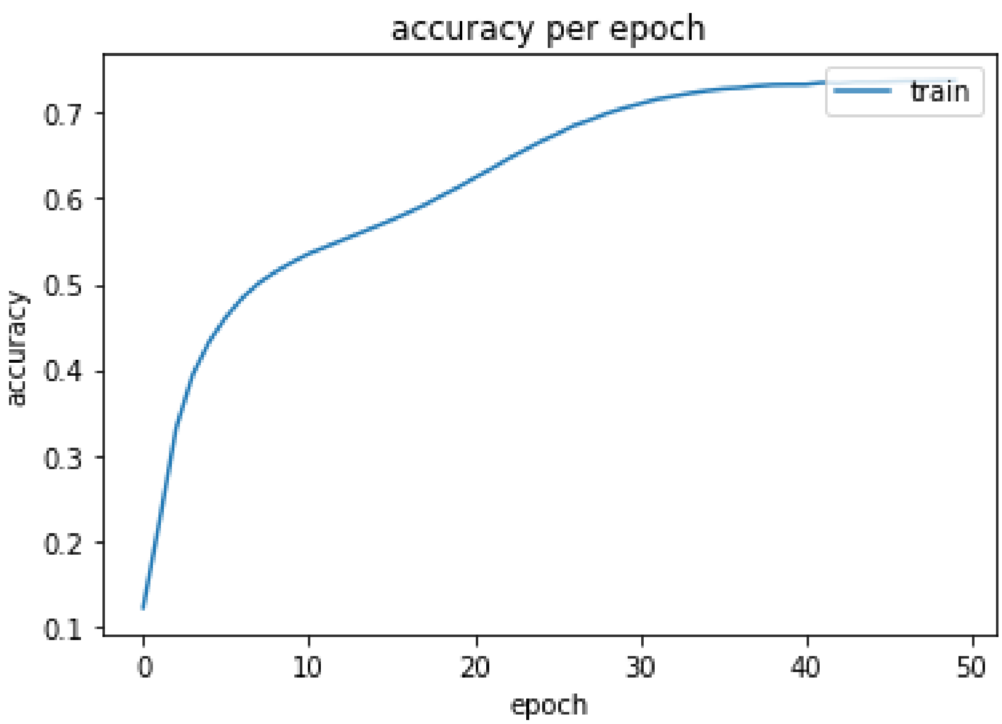

Figure 23: Accuracy of model

Generating Words

[22]:
```
model.reset_states()
sample = 'HARRY POTTER AND THE GOBLET OF FIRE'
sample_vector = [char2idx[s] for s in sample]
predicted = sample_vector
sample_tensor = tf.expand_dims(sample_vector, 0)
sample_tensor = tf.repeat(sample_tensor, 64, axis=0)


temperature = 0.6
for i in range(1000):
```

```
    pred = model(sample_tensor)
    pred = pred[0].numpy()/temperature
    pred = tf.random.categorical(pred, num_samples=1)[-1,0].numpy()
    predicted.append(pred)
    sample_tensor = predicted[-99:]
    sample_tensor = tf.expand_dims([pred],0)
    sample_tensor = tf.repeat(sample_tensor, 64, axis=0)

pred_char = [vocab[i] for i in predicted]
generated = ''.join(pred_char)
print(generated)
```

```
HARRY POTTER AND THE GOBLET OF FIRE

EA:

A Barty Quidditch team, the Neville g_t involved with him when he zouched ?MO to
HhpiüNe.  The surface of the water was almost voice.
"XI - 7(could he ^:E Xver tjudü anX /(could have easSnelnuster, Neville XSne"
- Harry sMi•gs down the stone steps, out into the clouB7?  at Voldemort's father
nervously as possible toward him.  Harry could see Madam Pomfrey fussing over
Hermione, •zeF, B_/r?  Wurry didN't seem to be very difficult to see out of
them.  They s'queJ6 the third task, My zoo   use TheUrt; it f•yVoldemort stood
on tip of it and pointing to the kitchenKN.
"The "Th2Karkaroff's supporterpfkinnroI4 to rest on de
"That's not the point musttince 1Hen," she said. "1f Ju) a Jre7?"
"Yeago," said Harry.
"I see," said xising like zeFQthe look on Harry's face.  "Ministry was pleased.
Ind Quidditch team Sirius ha
"There's a wayzPoff tallJY - you will come and have a look," said Harry,
pointing to a large patH  XJown quickly from the stairs Wormtail had done when
his hand had been cut
```

1. Sparse Categorical Crossentropy

In part 1, we use sparse_categorical_crossentropy as loss function.

loss = 1.2430

2.2

2. Mean Squared Error(MSE)

loss = 7.05

[36]:
```
model = build_model(
  vocab_size = len(vocab),
  embedding_dim=embedding_dim,
  rnn_units=rnn_units,
  batch_size=BATCH_SIZE)
```

[37]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
model.compile(optimizer='adam', loss='mean_squared_error')
```

[26]:
```
history = model.fit(dataset, epochs= 10)
```

```
Epoch 1/10
86/86 [==============================] - 28s 304ms/step - loss: 7.0297
...
Epoch 10/10
86/86 [==============================] - 24s 262ms/step - loss: 7.0535
```

3. Mean Absolute Error(MAE)

loss = 7.0047

[29]:
```
model = build_model(
  vocab_size = len(vocab),
  embedding_dim=embedding_dim,
  rnn_units=rnn_units,
  batch_size=BATCH_SIZE)
```

[40]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
model.compile(optimizer='adam', loss='mean_absolute_error')
```

[24]:
```
history = model.fit(dataset, epochs= 10, callbacks=[callback])
```

```
Epoch 1/10
86/86 [==============================] - 29s 303ms/step - loss: 6.0270
...
Epoch 5/10
86/86 [==============================] - 24s 264ms/step - loss: 7.0047
```

2.3

In part 1, we use Adam as optimizer.

loss = 1.2430, accuracy = 0.7377

2. SGD

loss = 7.69

[19]:
```
model = build_model(
  vocab_size = len(vocab),
  embedding_dim=embedding_dim,
  rnn_units=rnn_units,
  batch_size=BATCH_SIZE)
```

[20]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
model.compile(optimizer='SGD', loss='sparse_categorical_crossentropy')
```

[20]:
```
history = model.fit(dataset, epochs= 10, callbacks=[callback])
```

```
Epoch 1/10
86/86 [==============================] - 28s 294ms/step - loss: 7.6880
Epoch 2/10
86/86 [==============================] - 27s 300ms/step - loss: 7.6908
Epoch 3/10
86/86 [==============================] - 26s 291ms/step - loss: 7.6924
Epoch 4/10
86/86 [==============================] - 28s 307ms/step - loss: 7.6788
Epoch 5/10
86/86 [==============================] - 26s 289ms/step - loss: 7.6894
Epoch 6/10
86/86 [==============================] - 27s 304ms/step - loss: 7.6873
Epoch 7/10
86/86 [==============================] - 26s 290ms/step - loss: 7.6922
```

3. Nadam

loss = 7.03

[22]:
```
model = build_model(
  vocab_size = len(vocab),
  embedding_dim=embedding_dim,
  rnn_units=rnn_units,
  batch_size=BATCH_SIZE)
```

[23]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
model.compile(optimizer='Nadam', loss='sparse_categorical_crossentropy')
```

[23]:
```
history = model.fit(dataset, epochs= 10, callbacks=[callback])
```

```
Epoch 1/10
86/86 [==============================] - 29s 303ms/step - loss: 6.1675
Epoch 2/10
86/86 [==============================] - 28s 307ms/step - loss: 5.7675
Epoch 3/10
86/86 [==============================] - 27s 301ms/step - loss: 5.8680
Epoch 4/10
86/86 [==============================] - 28s 306ms/step - loss: 6.7698
Epoch 5/10
86/86 [==============================] - 26s 290ms/step - loss: 7.0355
```

Conclusion : The best loss function for this problem is Sparse Categorical Crossentropy and best optimizer is Adam

Table 1: Loss value for different loss functions

Loss Function	**Loss**
Sparse Categorical Crossentropy	**1.2430**
Mean Squared Error(MSE)	**7.05**
Mean Absolute Error(MAE)	**7.0047**

Table 2: Loss value for different optimizers

Optimizer	**Loss**
Adam	**1.2430**
SGD	**7.69**
NAdam)	**7.03**

2.4

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn! For predicting data in sequence we can use deep learning models like RNN or LSTM. LSTM can be used to predict the next word. The neural network take sequence of words as input and output will be a matrix of probability for each word from dictionary to be next of given sequence.

3 Contextual Embedding + RNNs

3.1

preprocessing includes:

1. replaced username with $< user >$ **tag**

2. replaced numbers with $< number >$ **tag**

3. removed RT-tags

4. Converted upper-case letters to lower-case

5. replaced many exclamation marks to one

6. replaced links with $< url >$ **tag**

7. replaced hashtags with $< emotion >$ **tag**

8. removed quotation mark

9. replaced the don't with (do not) and didn't with (did not)

3.2

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[2]:
```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils import data
import re
from termcolor import colored
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
import torch.optim as optim
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix, plot_confusion_matrix
from sklearn.model_selection import train_test_split
```

[3]:
```
PreDataFrame = pd.read_csv('/content/drive/MyDrive/pre_main_data.csv')
```

[4]:
```
sample_text = PreDataFrame['tweet'][0]
sample_text
```

[4]:
```
'! <user> as a woman you should not complain about cleaning up your house and as
a man you should always take the trash out'
```

[5]:
```python
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
```

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

One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words. Stop Words: A stop word is a commonly used word (such as "the", "a", "an", "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

[6]:
```python
def remove_stop_words(text):
    stop_words = set(stopwords.words('english'))
    pstem = PorterStemmer()
    lem = WordNetLemmatizer()
    text_words = [ word for word in text.split() if (word not in stop_words or word in ['not', 'can'])]
    text = ' '.join(text_words)
    return text
```

[7]:
```python
print(colored('before removing the stopwords \n', 'red'), sample_text)
sample_text = remove_stop_words(sample_text)
print(colored('after removing the stopwords \n', 'red'), sample_text)
```

```
before removing the stopwords

 ! <user> as a woman you should not complain about cleaning up your house
and as a man you should always take the trash out
after removing the stopwords

 ! <user> woman not complain cleaning house man always take trash
```

[8]:
```python
all_texts = PreDataFrame['tweet'].values
all_labels = PreDataFrame['label'].values
all_texts = [
             remove_stop_words(text)
             for text in tqdm(all_texts)
]
```

```
0%|          | 0/24433 [00:00<?, ?it/s]
```

[9]:
```
Hate_text = ''
Offensive_text = ''
Normal_text = ''
for i in range(len(all_labels)):
    if all_labels[i] == 0:
        Hate_text += all_texts[i]
    elif all_labels[i] == 1:
        Offensive_text += all_texts[i]
    elif all_labels[i] == 2:
        Normal_text += all_texts[i]
from wordcloud import WordCloud
list_text = [Hate_text, Offensive_text, Normal_text]
for txt in list_text:
    word_cloud = WordCloud(width = 600,height = 600,max_font_size = 200).
 ↪generate(txt)
    plt.figure(figsize=(12,10))# create a new figure
    plt.imshow(word_cloud,interpolation="bilinear")
    plt.axis("off")
    plt.show()
```

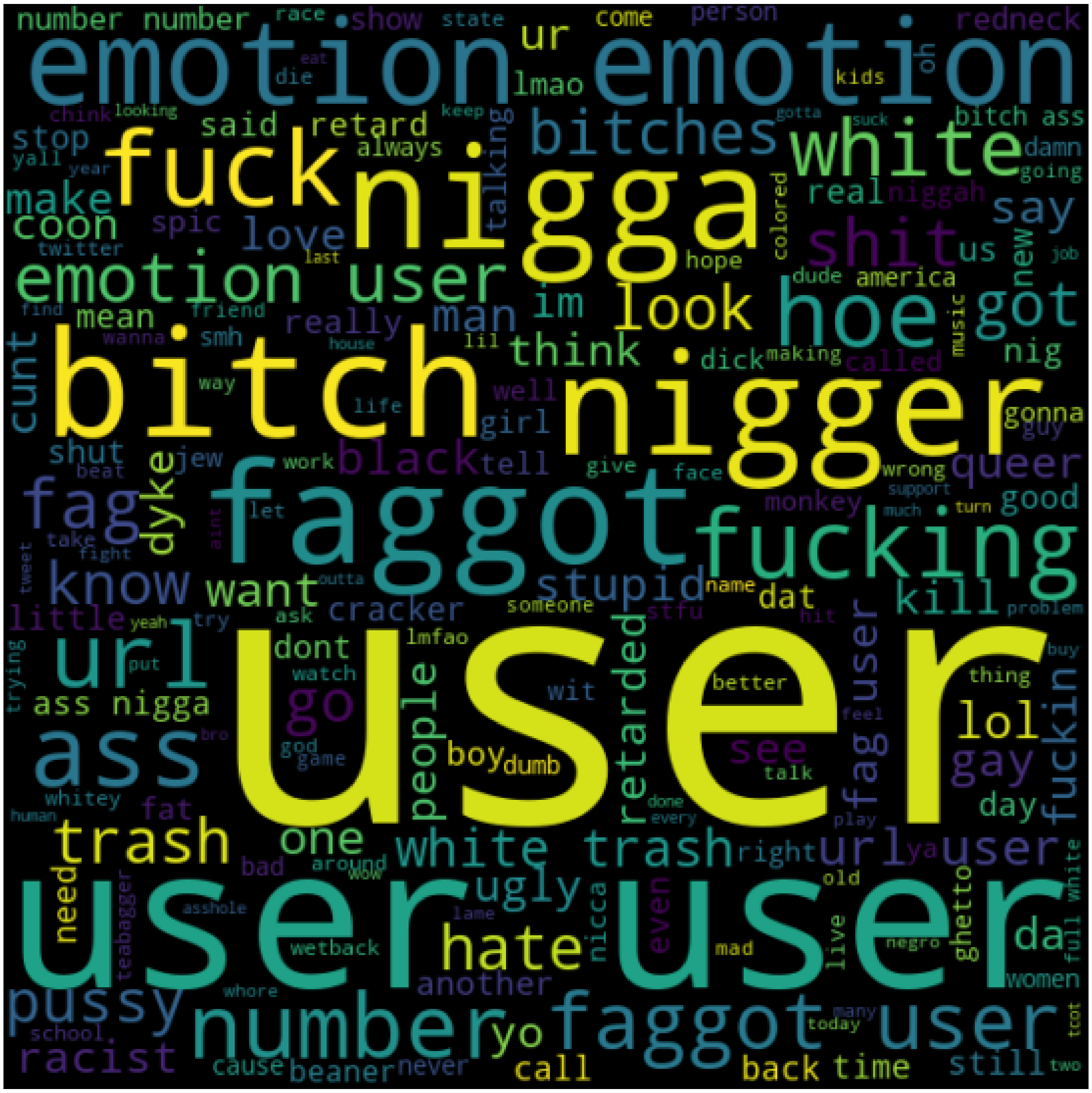

Figure 24: Most frequently used words in class 'Hate'

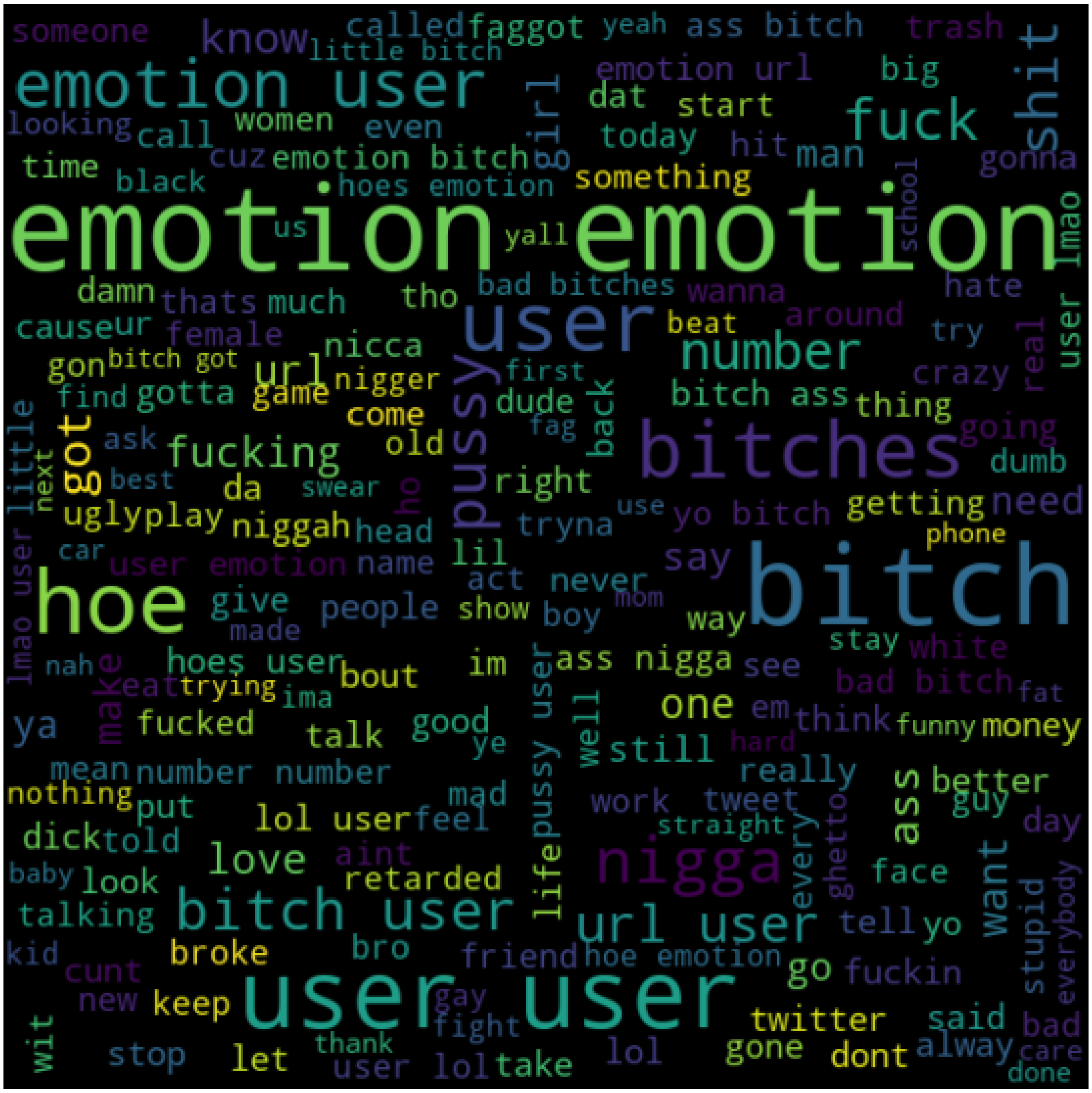

Figure 25: Most frequently used words in class 'Offensive'

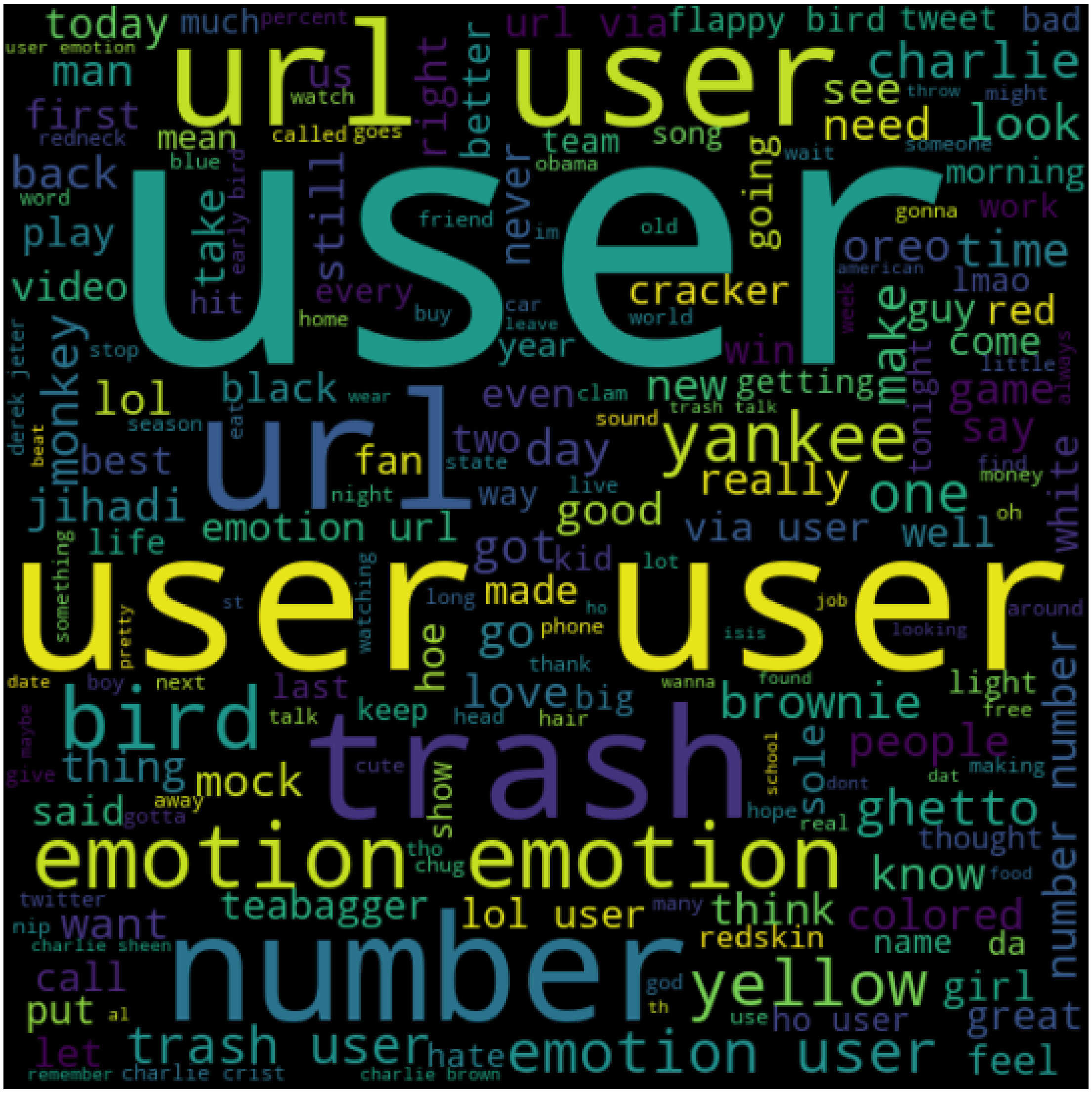

Figure 26: Most frequently used words in class 'Normal'

[11]:
```python
from transformers import BertTokenizer, BertForSequenceClassification

# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

# Create a function to tokenize a set of texts
def preprocessing_for_bert(data):
    """Perform required preprocessing steps for pretrained BERT.
```

```
    @param    data (np.array): Array of texts to be processed.
    @return   input_ids (torch.Tensor): Tensor of token ids to be fed to a model.
    @return   attention_masks (torch.Tensor): Tensor of indices specifying which
                  tokens should be attended to by the model.
    """

    input_ids = []
    attention_masks = []

    # For every sentence...
    for sent in data:
        encoded_sent = tokenizer.encode_plus(
            text=sent,  # Preprocess sentence
            add_special_tokens=True,        # Add `[CLS]` and `[SEP]`
            max_length=MAX_LEN,                  # Max length to truncate/pad
            padding='max_length',
            truncation=True,
            return_attention_mask=True      # Return attention mask
            )

        # Add the outputs to the lists
        input_ids.append(encoded_sent.get('input_ids'))
        attention_masks.append(encoded_sent.get('attention_mask'))

    # Convert lists to tensors
    input_ids = torch.tensor(input_ids)
    attention_masks = torch.tensor(attention_masks)

    return input_ids, attention_masks
```

```
Downloading:   0%|          | 0.00/226k [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/28.0 [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/570 [00:00<?, ?B/s]
```

[12]:
```
encoded_texts = [tokenizer.encode(sent, add_special_tokens=True) for sent in␣
 ↪all_texts]

# Find the maximum length
max_len = max([len(sent) for sent in encoded_texts])
print('Max length: ', max_len)
```

```
Max length:  241
```

[13]:
```
MAX_LEN = max_len

# Print sentence 0 and its encoded token ids
```

```
token_ids = list(preprocessing_for_bert([all_texts[0]])[0].squeeze().numpy())
print('Original: ', all_texts[0])
print('Token IDs: ', token_ids)
```

```
Original:  ! <user> woman not complain cleaning house man always take trash
Token IDs:  [101, 999, 1026, 5310, 1028, 2450, 2025, 17612, 9344, 2160, 2158,
2467, 2202, 11669, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[14]:
```
sns.countplot(all_labels)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff1e2521510>

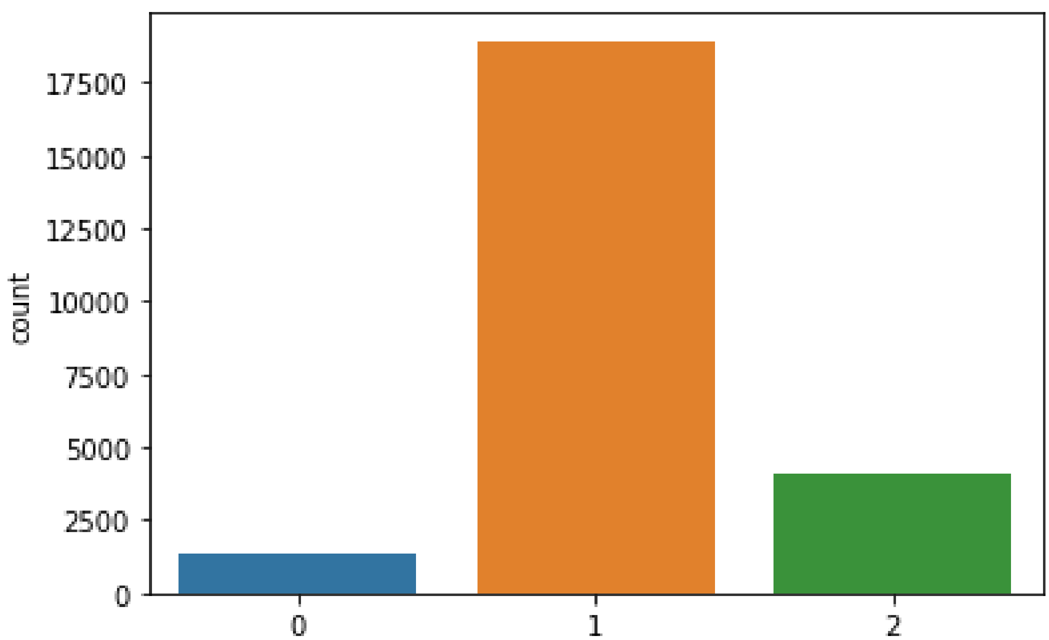

Figure 27: Countplot of data distribution in three classess

[15]:
```
X_train, X_test, y_train, y_test = train_test_split(all_texts, all_labels,␣
↪test_size=0.2, random_state=42, stratify=all_labels)
```

[16]:
```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)

X_train = np.array(X_train).reshape(-1, 1)

X_train, y_train = ros.fit_resample(X_train, y_train)

X_train = X_train.squeeze(1)
```

[17]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
↪1, random_state=42, stratify=y_train)
```

[18]:
```
print('the shape of the train split: {}'.format(len(X_train)))
print('the shape of the validation split: {}'.format(len(X_val)))
print('the shape of the test split: {}'.format(len(X_test)))
```

```
the shape of the train split: 40902
the shape of the validation split: 4545
the shape of the test split: 4887
```

[19]:
```
print('Tokenizing data...')
train_inputs, train_masks = preprocessing_for_bert(X_train)
val_inputs, val_masks = preprocessing_for_bert(X_val)
```

```
Tokenizing data...
```

[21]:
```
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,␣
↪SequentialSampler

# Convert other data types to torch.Tensor
train_labels = torch.tensor(y_train)
val_labels = torch.tensor(y_val)

batch_size = 32

# Create the DataLoader for our training set
train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler,␣
↪batch_size=batch_size)
```

```python
# Create the DataLoader for our validation set
val_data = TensorDataset(val_inputs, val_masks, val_labels)
val_sampler = SequentialSampler(val_data)
val_dataloader = DataLoader(val_data, sampler=val_sampler, batch_size=batch_size)
```

[22]:
```python
%%time
import torch
import torch.nn as nn
from transformers import BertModel

# Create the BertClassfier class
class BertClassifier(nn.Module):
    """Bert Model for Classification Tasks.
    """
    def __init__(self, freeze_bert=False):
        """
        @param    bert: a BertModel object
        @param    classifier: a torch.nn.Module classifier
        @param    freeze_bert (bool): Set `False` to fine-tune the BERT model
        """
        super(BertClassifier, self).__init__()

        D_in, H, D_out = 768, 50, 3

        self.bert = BertModel.from_pretrained('bert-base-uncased')

        self.classifier = nn.Sequential(
            nn.Linear(D_in, H),
            nn.ReLU(),
            # nn.Dropout(0.5),
            nn.Linear(H, D_out)
        )

        # Freeze the BERT model
        if freeze_bert:
            for param in self.bert.parameters():
                param.requires_grad = False

    def forward(self, input_ids, attention_mask):
        """
        Feed input to BERT and the classifier to compute logits.
        @param    input_ids (torch.Tensor): an input tensor with shape␣
↪(batch_size,
                      max_length)
        @param    attention_mask (torch.Tensor): a tensor that hold attention␣
↪mask
```

```
                    information with shape (batch_size, max_length)
        @return   logits (torch.Tensor): an output tensor with shape (batch_size,
                    num_labels)
        """
        # Feed input to BERT
        outputs = self.bert(input_ids=input_ids,
                            attention_mask=attention_mask)


        # Extract the last hidden state of the token `[CLS]` for classification␣
 ↪task
        last_hidden_state_cls = outputs[0][:, 0, :]

        # Feed input to classifier to compute logits
        logits = self.classifier(last_hidden_state_cls)

        return logits
```

```
CPU times: user 121 µs, sys: 5 µs, total: 126 µs
Wall time: 130 µs
```

[23]:
```
import torch

if torch.cuda.is_available():
    device = torch.device("cuda")
    print(f'There are {torch.cuda.device_count()} GPU(s) available.')
    print('Device name:', torch.cuda.get_device_name(0))

else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

```
There are 1 GPU(s) available.
Device name: Tesla T4
```

[24]:
```
from transformers import AdamW, get_linear_schedule_with_warmup

def initialize_model(epochs=4):
    """Initialize the Bert Classifier, the optimizer and the learning rate␣
 ↪scheduler.
    """
    # Instantiate Bert Classifier
    bert_classifier = BertClassifier(freeze_bert=False)

    # Tell PyTorch to run the model on GPU
    bert_classifier.to(device)
```

```
    # Create the optimizer
    optimizer = AdamW(bert_classifier.parameters(),
                      lr=5e-5,    # Default learning rate
                      eps=1e-8    # Default epsilon value
                      )

    # Total number of training steps
    total_steps = len(train_dataloader) * epochs

    # Set up the learning rate scheduler
    scheduler = get_linear_schedule_with_warmup(optimizer,
                                                num_warmup_steps=0, # Default␣
 ↪value
                                                num_training_steps=total_steps)
    return bert_classifier, optimizer, scheduler
```

[25]:
```
import random
import time

# Specify loss function
loss_fn = nn.CrossEntropyLoss()

def set_seed(seed_value=42):
    """Set seed for reproducibility.
    """
    random.seed(seed_value)
    np.random.seed(seed_value)
    torch.manual_seed(seed_value)
    torch.cuda.manual_seed_all(seed_value)
```

[26]:
```
def train(model, train_dataloader, val_dataloader=None, epochs=4,␣
 ↪evaluation=False):
    """Train the BertClassifier model.
    """

    val_accuracy_list = []
    train_accuracy_list = []
    val_loss_list = []
    train_loss_list = []

    # Start training loop
    print("Start training...\n")
    for epoch_i in range(epochs):

        train_epoch_accuracy_list = []
        train_epoch_loss_list = []
```

```
    print(f"{'Epoch':^7} | {'Batch':^7} | {'Train Loss':^12} | {'Val Loss':
↪^10} | {'Val Acc':^9} | {'Elapsed':^9}")
    print("-"*70)

    # Measure the elapsed time of each epoch
    t0_epoch, t0_batch = time.time(), time.time()

    # Reset tracking variables at the beginning of each epoch
    total_loss, batch_loss, batch_counts = 0, 0, 0

    # Put the model into the training mode
    model.train()

    # For each batch of training data...
    for step, batch in enumerate(train_dataloader):
        batch_counts +=1

        b_input_ids, b_attn_mask, b_labels = tuple(t.to(device) for t in␣
↪batch)

        model.zero_grad()


        logits = model(b_input_ids, b_attn_mask)

        loss = loss_fn(logits, b_labels)
        batch_loss += loss.item()
        total_loss += loss.item()

        preds = torch.argmax(logits, dim=1).flatten()

        accuracy = (preds == b_labels).cpu().numpy().mean() * 100
        train_epoch_accuracy_list.append(accuracy)
        train_epoch_loss_list.append(loss.item())


        # Perform a backward pass to calculate gradients
        loss.backward()

        # Clip the norm of the gradients to 1.0 to prevent "exploding␣
↪gradients"
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        # Update parameters and the learning rate
        optimizer.step()
        scheduler.step()
```

```
            # Print the loss values and time elapsed for every 20 batches
            if (step % 20 == 0 and step != 0) or (step == len(train_dataloader)␣
↪- 1):
                # Calculate time elapsed for 20 batches
                time_elapsed = time.time() - t0_batch

                # Print training results
                print(f"{epoch_i + 1:^7} | {step:^7} | {batch_loss /␣
↪batch_counts:^12.6f} | {'-':^10} | {'-':^9} | {time_elapsed:^9.2f}")

                # Reset batch tracking variables
                batch_loss, batch_counts = 0, 0
                t0_batch = time.time()

        # Calculate the average loss over the entire training data
        avg_train_loss = total_loss / len(train_dataloader)
        train_accuracy_list.append(np.mean(train_epoch_accuracy_list))
        train_loss_list.append(np.mean(train_epoch_loss_list))

        print("-"*70)

        if evaluation == True:
            # After the completion of each training epoch, measure the model's␣
↪performance
            # on our validation set.
            val_loss, val_accuracy = evaluate(model, val_dataloader)
            val_accuracy_list.append(val_accuracy)
            val_loss_list.append(val_loss)

            # Print performance over the entire training data
            time_elapsed = time.time() - t0_epoch

            print(f"{epoch_i + 1:^7} | {'-':^7} | {avg_train_loss:^12.6f} |␣
↪{val_loss:^10.6f} | {val_accuracy:^9.2f} | {time_elapsed:^9.2f}")
            print("-"*70)
        print("\n")

    print("Training complete!")
    return train_accuracy_list, train_loss_list, val_accuracy_list, val_loss_list
```

[27]:
```
def evaluate(model, val_dataloader):
    """After the completion of each training epoch, measure the model's␣
↪performance
    on our validation set.
    """
    # Put the model into the evaluation mode. The dropout layers are disabled␣
↪during
```

```
    # the test time.
    model.eval()

    # Tracking variables
    val_accuracy = []
    val_loss = []

    # For each batch in our validation set...
    for batch in val_dataloader:
        # Load batch to GPU
        b_input_ids, b_attn_mask, b_labels = tuple(t.to(device) for t in batch)

        # Compute logits
        with torch.no_grad():
            logits = model(b_input_ids, b_attn_mask)

        # Compute loss
        loss = loss_fn(logits, b_labels)
        val_loss.append(loss.item())

        # Get the predictions
        preds = torch.argmax(logits, dim=1).flatten()

        # Calculate the accuracy rate
        accuracy = (preds == b_labels).cpu().numpy().mean() * 100
        val_accuracy.append(accuracy)

    # Compute the average accuracy and loss over the validation set.
    val_loss = np.mean(val_loss)
    val_accuracy = np.mean(val_accuracy)

    return val_loss, val_accuracy
```

[29]:
```
set_seed(42)
bert_classifier, optimizer, scheduler = initialize_model(epochs=3)
train_accuracy_list, train_loss_list, val_accuracy_list, val_loss_list =␣
 ↪train(bert_classifier, train_dataloader, val_dataloader, epochs=3,␣
 ↪evaluation=True)
```

```
Downloading:   0%|          | 0.00/420M [00:00<?, ?B/s]
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when
initializing BertModel: ['cls.seq_relationship.weight',
'cls.predictions.decoder.weight', 'cls.predictions.transform.LayerNorm.bias',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.bias',
'cls.predictions.transform.dense.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a
```

```
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of
a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
/usr/local/lib/python3.7/dist-packages/transformers/optimization.py:309:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  FutureWarning,

Start training...

 Epoch  |  Batch  |  Train Loss  |  Val Loss  |  Val Acc  |  Elapsed
----------------------------------------------------------------------
   1    |   20    |   0.996747   |     -      |     -     |   24.69
   1    |   40    |   0.646411   |     -      |     -     |   24.37
   1    |   60    |   0.555916   |     -      |     -     |   25.28
   1    |   80    |   0.515594   |     -      |     -     |   26.18
   1    |   100   |   0.478272   |     -      |     -     |   27.41
   1    |   120   |   0.424866   |     -      |     -     |   27.41
   1    |   140   |   0.447459   |     -      |     -     |   26.98
   1    |   160   |   0.452878   |     -      |     -     |   27.25
   1    |   180   |   0.439995   |     -      |     -     |   27.21
   1    |   200   |   0.417585   |     -      |     -     |   27.13
   1    |   220   |   0.423836   |     -      |     -     |   27.07
   1    |   240   |   0.415922   |     -      |     -     |   27.09
   1    |   260   |   0.377614   |     -      |     -     |   27.21
   1    |   280   |   0.362014   |     -      |     -     |   27.21
   1    |   300   |   0.338250   |     -      |     -     |   27.13
   1    |   320   |   0.384805   |     -      |     -     |   27.03
   1    |   340   |   0.428619   |     -      |     -     |   27.04
   1    |   360   |   0.355779   |     -      |     -     |   27.15
   1    |   380   |   0.366752   |     -      |     -     |   27.20
   1    |   400   |   0.325294   |     -      |     -     |   27.09
   1    |   420   |   0.333883   |     -      |     -     |   27.05
   1    |   440   |   0.339145   |     -      |     -     |   27.04
   1    |   460   |   0.359476   |     -      |     -     |   27.19
   1    |   480   |   0.293419   |     -      |     -     |   27.19
   1    |   500   |   0.389113   |     -      |     -     |   27.17
   1    |   520   |   0.303409   |     -      |     -     |   27.19
   1    |   540   |   0.264825   |     -      |     -     |   27.17
   1    |   560   |   0.288023   |     -      |     -     |   27.14
   1    |   580   |   0.269139   |     -      |     -     |   27.18
   1    |   600   |   0.287805   |     -      |     -     |   27.15
   1    |   620   |   0.325858   |     -      |     -     |   27.17
   1    |   640   |   0.279732   |     -      |     -     |   27.17
   1    |   660   |   0.282266   |     -      |     -     |   27.16
```

1	680	0.257314	-	-	27.14
1	700	0.246090	-	-	27.11
1	720	0.268035	-	-	27.16
1	740	0.239145	-	-	27.14
1	760	0.316495	-	-	27.16
1	780	0.241679	-	-	27.15
1	800	0.275521	-	-	27.14
1	820	0.204975	-	-	27.17
1	840	0.208426	-	-	27.16
1	860	0.235175	-	-	27.11
1	880	0.163076	-	-	27.17
1	900	0.201489	-	-	27.12
1	920	0.238443	-	-	27.13
1	940	0.218483	-	-	27.15
1	960	0.143447	-	-	27.10
1	980	0.214860	-	-	27.16
1	1000	0.194434	-	-	27.11
1	1020	0.241123	-	-	27.14
1	1040	0.199650	-	-	27.16
1	1060	0.211746	-	-	27.12
1	1080	0.179543	-	-	27.13
1	1100	0.120439	-	-	27.12
1	1120	0.167094	-	-	27.14
1	1140	0.122781	-	-	27.14
1	1160	0.143319	-	-	27.11
1	1180	0.194827	-	-	27.14
1	1200	0.150943	-	-	27.13
1	1220	0.167907	-	-	27.14
1	1240	0.127340	-	-	27.15
1	1260	0.177321	-	-	27.14
1	1278	0.149138	-	-	23.36
1	-	0.303767	0.158377	95.61	1806.47

Epoch	Batch	Train Loss	Val Loss	Val Acc	Elapsed
2	20	0.137117	-	-	28.48
2	40	0.133996	-	-	27.16
2	60	0.092514	-	-	27.14
2	80	0.125131	-	-	27.15
2	100	0.066786	-	-	27.16
2	120	0.116204	-	-	27.14
2	140	0.080077	-	-	27.14
2	160	0.063756	-	-	27.15
2	180	0.120881	-	-	27.15
2	200	0.111340	-	-	27.12

2	220	0.115245	-	-	27.13
2	240	0.107565	-	-	27.15
2	260	0.088122	-	-	27.14
2	280	0.125623	-	-	27.12
2	300	0.138723	-	-	27.13
2	320	0.095544	-	-	27.13
2	340	0.099597	-	-	27.13
2	360	0.090261	-	-	27.14
2	380	0.110188	-	-	27.15
2	400	0.088834	-	-	27.11
2	420	0.126565	-	-	27.15
2	440	0.077358	-	-	27.14
2	460	0.121551	-	-	27.10
2	480	0.112420	-	-	27.13
2	500	0.096967	-	-	27.11
2	520	0.082370	-	-	27.13
2	540	0.073203	-	-	27.14
2	560	0.067093	-	-	27.05
2	580	0.111559	-	-	27.02
2	600	0.073098	-	-	27.18
2	620	0.119348	-	-	27.20
2	640	0.105500	-	-	27.16
2	660	0.096469	-	-	27.12
2	680	0.069706	-	-	27.14
2	700	0.064163	-	-	27.07
2	720	0.078143	-	-	27.07
2	740	0.110736	-	-	27.03
2	760	0.075678	-	-	27.05
2	780	0.101792	-	-	27.06
2	800	0.098647	-	-	27.08
2	820	0.068547	-	-	27.07
2	840	0.079753	-	-	27.08
2	860	0.079431	-	-	27.08
2	880	0.075087	-	-	27.07
2	900	0.091613	-	-	27.05
2	920	0.070594	-	-	27.05
2	940	0.076625	-	-	27.03
2	960	0.090343	-	-	27.01
2	980	0.062902	-	-	27.03
2	1000	0.088362	-	-	27.07
2	1020	0.076612	-	-	27.15
2	1040	0.072062	-	-	27.19
2	1060	0.042243	-	-	27.18
2	1080	0.063079	-	-	27.15
2	1100	0.054001	-	-	27.13
2	1120	0.059686	-	-	27.13
2	1140	0.096434	-	-	27.10
2	1160	0.059437	-	-	27.08

2	1180	0.044441	-	-	27.08
2	1200	0.034562	-	-	27.07
2	1220	0.042263	-	-	27.09
2	1240	0.105150	-	-	27.13
2	1260	0.056231	-	-	27.10
2	1278	0.069251	-	-	23.30
2	-	0.088014	0.095323	97.71	1813.67

Epoch	Batch	Train Loss	Val Loss	Val Acc	Elapsed
3	20	0.036287	-	-	28.54
3	40	0.041991	-	-	27.18
3	60	0.065149	-	-	27.15
3	80	0.039897	-	-	27.14
3	100	0.034615	-	-	27.12
3	120	0.037805	-	-	27.10
3	140	0.038546	-	-	27.09
3	160	0.034308	-	-	27.07
3	180	0.020614	-	-	27.07
3	200	0.010977	-	-	27.09
3	220	0.036150	-	-	27.11
3	240	0.040152	-	-	27.11
3	260	0.023429	-	-	27.12
3	280	0.009737	-	-	27.10
3	300	0.023500	-	-	27.07
3	320	0.058050	-	-	27.06
3	340	0.003162	-	-	27.09
3	360	0.061921	-	-	27.10
3	380	0.024907	-	-	27.09
3	400	0.044794	-	-	27.08
3	420	0.019773	-	-	27.05
3	440	0.019471	-	-	27.07
3	460	0.028582	-	-	27.06
3	480	0.051904	-	-	27.11
3	500	0.025780	-	-	27.08
3	520	0.033977	-	-	27.09
3	540	0.034615	-	-	27.09
3	560	0.019735	-	-	27.08
3	580	0.037006	-	-	27.07
3	600	0.029785	-	-	27.07
3	620	0.043323	-	-	27.09
3	640	0.067604	-	-	27.07
3	660	0.022829	-	-	27.10
3	680	0.023095	-	-	27.12
3	700	0.027819	-	-	27.10

```
    3     |    720    |    0.030802   |       -       |       -      |    27.08
    3     |    740    |    0.036834   |       -       |       -      |    27.10
    3     |    760    |    0.049956   |       -       |       -      |    27.07
    3     |    780    |    0.064659   |       -       |       -      |    27.08
    3     |    800    |    0.019458   |       -       |       -      |    27.09
    3     |    820    |    0.024605   |       -       |       -      |    27.10
    3     |    840    |    0.040575   |       -       |       -      |    27.13
    3     |    860    |    0.030176   |       -       |       -      |    27.10
    3     |    880    |    0.037373   |       -       |       -      |    27.11
    3     |    900    |    0.031026   |       -       |       -      |    27.17
    3     |    920    |    0.004772   |       -       |       -      |    27.20
    3     |    940    |    0.022867   |       -       |       -      |    27.19
    3     |    960    |    0.027840   |       -       |       -      |    27.16
    3     |    980    |    0.022078   |       -       |       -      |    27.11
    3     |   1000    |    0.041967   |       -       |       -      |    27.12
    3     |   1020    |    0.058163   |       -       |       -      |    27.10
    3     |   1040    |    0.009130   |       -       |       -      |    27.11
    3     |   1060    |    0.053224   |       -       |       -      |    27.08
    3     |   1080    |    0.027609   |       -       |       -      |    27.09
    3     |   1100    |    0.013194   |       -       |       -      |    27.15
    3     |   1120    |    0.010884   |       -       |       -      |    27.20
    3     |   1140    |    0.018413   |       -       |       -      |    27.18
    3     |   1160    |    0.015547   |       -       |       -      |    27.20
    3     |   1180    |    0.009515   |       -       |       -      |    27.22
    3     |   1200    |    0.021760   |       -       |       -      |    27.18
    3     |   1220    |    0.010094   |       -       |       -      |    27.17
    3     |   1240    |    0.028345   |       -       |       -      |    27.12
    3     |   1260    |    0.004983   |       -       |       -      |    27.07
    3     |   1278    |    0.031245   |       -       |       -      |    23.28
----------------------------------------------------------------------
    3     |     -     |    0.030760   |   0.079825    |    98.43     |   1813.36
----------------------------------------------------------------------


Training complete!
```

[30]:
```python
def plot_train_test_metric(train_acc, val_acc, xlabel, ylabel, title, metric):
    fig = plt.figure(figsize=(10, 6))
    plt.plot(train_acc, color='salmon', label='train {}'.format(metric),␣
 ↪marker='o', linewidth=1)
    plt.annotate(str(np.round(train_acc[-1], 1)) + '%', (len(train_acc) - .8,␣
 ↪train_acc[-1]), color='salmon')
    plt.plot(val_acc, color='skyblue', label='test {}'.format(metric),␣
 ↪marker='o', linewidth=1)
    plt.annotate(str(np.round(val_acc[-1], 1)) + '%', (len(val_acc) - .8,␣
 ↪val_acc[-1]), color='skyblue')
    plt.gca().spines['top'].set_visible(False)
```

```
    plt.gca().spines['right'].set_visible(False)
    plt.legend()
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.grid(axis='y')
```

[31]:
```
plot_train_test_metric(train_accuracy_list, val_accuracy_list, "epochs",␣
 ↪"accuracy", "Accuracy for different epochs", "accuracy")
```

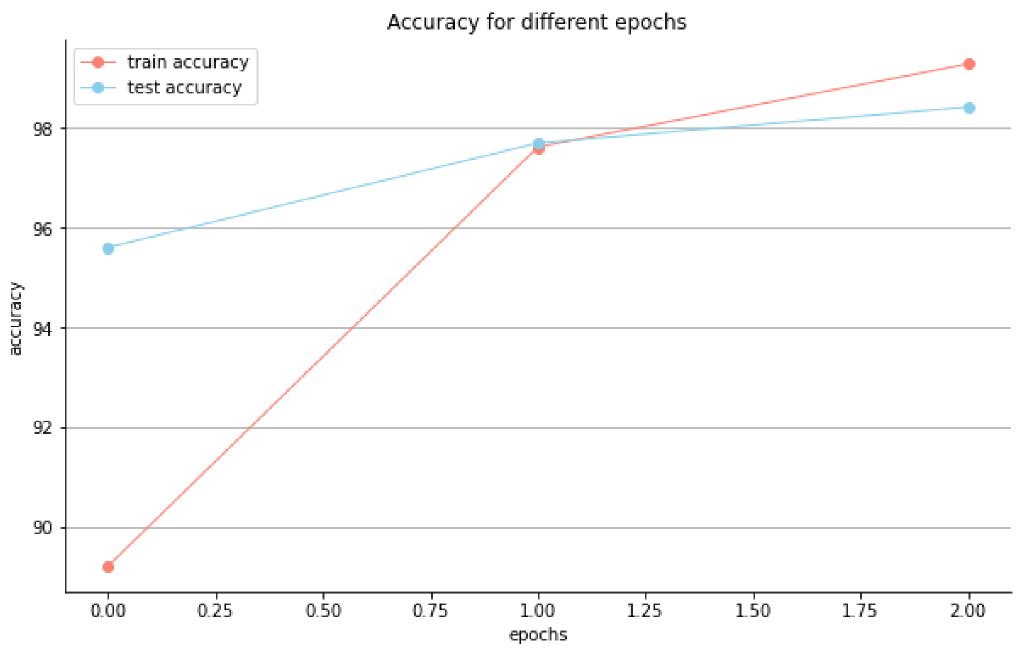

Figure 28: Accuracy and validation accuracy for Bert Model

[32]:
```
plot_train_test_metric(train_loss_list, val_loss_list, "epochs", "loss", "loss␣
 ↪for different epochs", "loss")
```

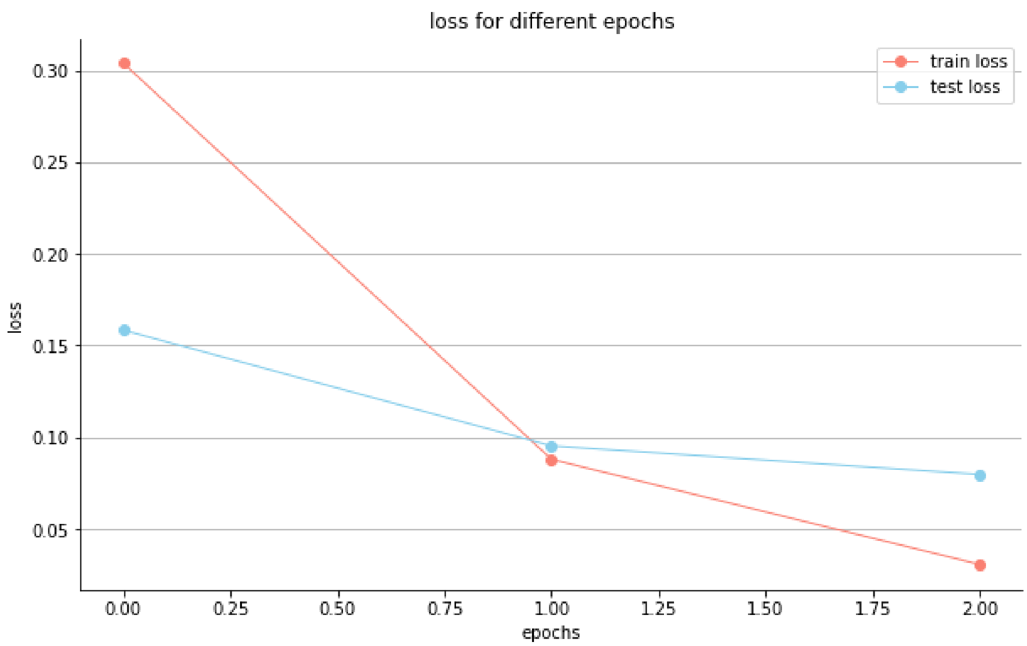

Figure 29: Loss and validation loss for Bert Model

[38]:
```
print('Tokenizing data...')
test_inputs, test_masks = preprocessing_for_bert(X_test)

# Create the DataLoader for our test set
test_dataset = TensorDataset(test_inputs, test_masks)
test_sampler = SequentialSampler(test_dataset)
test_dataloader = DataLoader(test_dataset, sampler=test_sampler, batch_size=32)
```

```
Tokenizing data...
```

[39]:
```
import torch.nn.functional as F

def bert_predict(model, test_dataloader):
    """Perform a forward pass on the trained BERT model to predict probabilities
    on the test set.
    """
    # Put the model into the evaluation mode. The dropout layers are disabled␣
 ↪during
    # the test time.
    model.eval()

    all_logits = []
```

```
    # For each batch in our test set...
    for batch in test_dataloader:
        # Load batch to GPU
        b_input_ids, b_attn_mask = tuple(t.to(device) for t in batch)[:2]

        # Compute logits
        with torch.no_grad():
            logits = model(b_input_ids, b_attn_mask)
        all_logits.append(logits)

    # Concatenate logits from each batch
    all_logits = torch.cat(all_logits, dim=0)

    # Apply softmax to calculate probabilities
    probs = F.softmax(all_logits, dim=1).cpu().numpy()

    return probs
```

[108]:
```
probs = bert_predict(bert_classifier, test_dataloader)
```

[150]:
```
classes = []
for i in range(len(probs)):
  if np.argmax(probs[i]) == 0 :
    classes.append(0)
  elif np.argmax(probs[i]) == 1:
    classes.append(1)
  else:
    classes.append(2)
```

[151]:
```
sns.heatmap(confusion_matrix(y_test, classes), annot=True)
plt.ylabel('target class')
plt.xlabel('output class')
plt.title('confusion matrix representing the performance of the model')
```

[151]: Text(0.5, 1.0, 'confusion matrix representing the performance of the model')

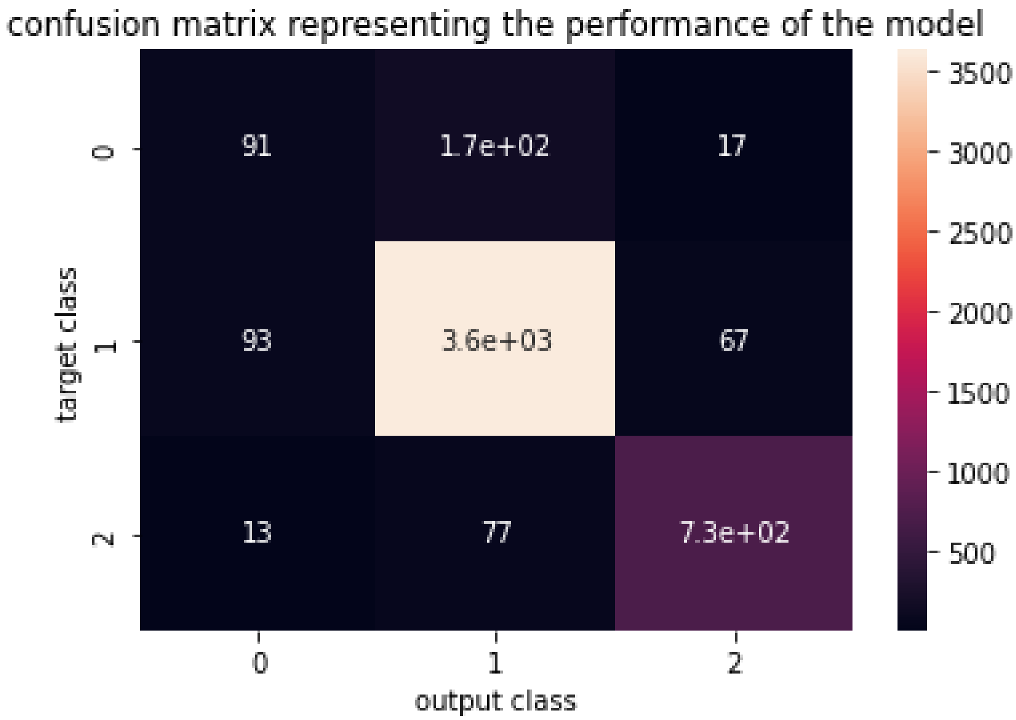

Figure 30: Confusion matrix heatmap

[155]:
```
acc = accuracy_score(y_test, classes)
print(acc)
```

```
0.910273924
```

[156]:
```
if acc > 0.91:
  print('Excellent job, You get the bonous!')
```

```
Excellent job, You get the bonous!
```