Github:
https://github.com/AKSHITHREDDI1205/Neural_networks_ICP_7/blob/main/ICP-7.ipynb

Video link:
https://drive.google.com/file/d/1rZJstcfmLbsa7SDGNylVv2khCvYXf08N/view?usp=sharing

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

import re

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from keras.utils import to_categorical

data = pd.read_csv('Sentiment.csv')
# Keeping only the neccessary columns
data = data[['text','sentiment']]

data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '',
x)))

for idx, row in data.iterrows():
    row[0] = row[0].replace('rt', ' ')

max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)

X = pad_sequences(X)

embed_dim = 128
lstm_out = 196
def createmodel():
    model = Sequential()
```

```
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3,activation='softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics
= ['accuracy'])
    return model
# print(model.summary())

labelencoder = LabelEncoder()
integer_encoded = labelencoder.fit_transform(data['sentiment'])
y = to_categorical(integer_encoded)
X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size = 0.33,
random_state = 42)

batch_size = 32
model = createmodel()
model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2)
score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size)
print(score)
print(acc)
print(model.metrics_names)
```

```
291/291 - 50s - loss: 0.8292 - accuracy: 0.6393 - 50s/epoch - 171ms/step
144/144 - 5s - loss: 0.7593 - accuracy: 0.6662 - 5s/epoch - 34ms/step
0.7592564821243286
0.666229784488678
['loss', 'accuracy']
```

```
model.save('sentimentAnalysis.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the native
Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
from keras.models import load_model
model= load_model('sentimentAnalysis.h5')
```

```
print(integer_encoded)
print(data['sentiment'])
```

```
[1 2 1 ... 2 0 2]
0         Neutral
1        Positive
2         Neutral
3        Positive
4        Positive
           ...
13866    Negative
13867    Positive
```

```
13868    Positive
13869    Negative
13870    Positive
Name: sentiment, Length: 13871, dtype: object
```

```
sentence = ['A lot of good things are happening. We are respected again
throughout the world, and that is a great thing.@realDonaldTrump']
sentence = tokenizer.texts_to_sequences(sentence)
sentence = pad_sequences(sentence, maxlen=28, dtype='int32', value=0)
sentiment_probs = model.predict(sentence, batch_size=1, verbose=2)[0]
sentiment = np.argmax(sentiment_probs)

print(sentiment_probs)
if sentiment == 0:
    print("Neutral")
elif sentiment < 0:
    print("Negative")
elif sentiment > 0:
    print("Positive")
else:
    print("Cannot be determined")
```

```
1/1 - 0s - 314ms/epoch - 314ms/step
[0.46129748 0.12149629 0.41720623]
Neutral
```

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

model = KerasClassifier(build_fn=createmodel,verbose=2)
batch_size= [10, 20, 40]
epochs = [1, 2]
param_grid= {'batch_size':batch_size, 'epochs':epochs}
grid  = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result= grid.fit(X_train,Y_train)

print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
```

```
<ipython-input-7-5b4d4aa083bb>:4: DeprecationWarning: KerasClassifier is
deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  model = KerasClassifier(build_fn=createmodel,verbose=2)

744/744 - 99s - loss: 0.8293 - accuracy: 0.6480 - 99s/epoch - 133ms/step
186/186 - 3s - loss: 0.7483 - accuracy: 0.6697 - 3s/epoch - 17ms/step
744/744 - 100s - loss: 0.8195 - accuracy: 0.6492 - 100s/epoch - 135ms/step
186/186 - 4s - loss: 0.7616 - accuracy: 0.6708 - 4s/epoch - 20ms/step
744/744 - 100s - loss: 0.8219 - accuracy: 0.6464 - 100s/epoch - 134ms/step
186/186 - 3s - loss: 0.7582 - accuracy: 0.6740 - 3s/epoch - 14ms/step
744/744 - 100s - loss: 0.8261 - accuracy: 0.6465 - 100s/epoch - 134ms/step
```

```
186/186 - 3s - loss: 0.7562 - accuracy: 0.6690 - 3s/epoch - 14ms/step
744/744 - 99s - loss: 0.8213 - accuracy: 0.6437 - 99s/epoch - 133ms/step
186/186 - 3s - loss: 0.7806 - accuracy: 0.6706 - 3s/epoch - 14ms/step
Epoch 1/2
744/744 - 100s - loss: 0.8243 - accuracy: 0.6470 - 100s/epoch - 134ms/step
Epoch 2/2
744/744 - 96s - loss: 0.6831 - accuracy: 0.7092 - 96s/epoch - 129ms/step
186/186 - 3s - loss: 0.7271 - accuracy: 0.6934 - 3s/epoch - 14ms/step
Epoch 1/2
744/744 - 99s - loss: 0.8190 - accuracy: 0.6499 - 99s/epoch - 133ms/step
Epoch 2/2
744/744 - 94s - loss: 0.6837 - accuracy: 0.7082 - 94s/epoch - 126ms/step
186/186 - 3s - loss: 0.7593 - accuracy: 0.6826 - 3s/epoch - 16ms/step
Epoch 1/2
744/744 - 99s - loss: 0.8202 - accuracy: 0.6485 - 99s/epoch - 133ms/step
Epoch 2/2
744/744 - 93s - loss: 0.6719 - accuracy: 0.7163 - 93s/epoch - 126ms/step
186/186 - 3s - loss: 0.7635 - accuracy: 0.6778 - 3s/epoch - 14ms/step
Epoch 1/2
744/744 - 96s - loss: 0.8266 - accuracy: 0.6436 - 96s/epoch - 129ms/step
Epoch 2/2
744/744 - 95s - loss: 0.6682 - accuracy: 0.7138 - 95s/epoch - 128ms/step
186/186 - 3s - loss: 0.7689 - accuracy: 0.6889 - 3s/epoch - 14ms/step
Epoch 1/2
744/744 - 98s - loss: 0.8209 - accuracy: 0.6491 - 98s/epoch - 132ms/step
Epoch 2/2
744/744 - 95s - loss: 0.6662 - accuracy: 0.7146 - 95s/epoch - 128ms/step
186/186 - 3s - loss: 0.7963 - accuracy: 0.6561 - 3s/epoch - 14ms/step
372/372 - 58s - loss: 0.8376 - accuracy: 0.6433 - 58s/epoch - 155ms/step
93/93 - 2s - loss: 0.7685 - accuracy: 0.6799 - 2s/epoch - 25ms/step
372/372 - 54s - loss: 0.8344 - accuracy: 0.6433 - 54s/epoch - 146ms/step
93/93 - 2s - loss: 0.7547 - accuracy: 0.6713 - 2s/epoch - 19ms/step
372/372 - 55s - loss: 0.8289 - accuracy: 0.6430 - 55s/epoch - 149ms/step
93/93 - 2s - loss: 0.7497 - accuracy: 0.6837 - 2s/epoch - 19ms/step
372/372 - 58s - loss: 0.8297 - accuracy: 0.6428 - 58s/epoch - 156ms/step
93/93 - 3s - loss: 0.7288 - accuracy: 0.6841 - 3s/epoch - 27ms/step
372/372 - 55s - loss: 0.8254 - accuracy: 0.6438 - 55s/epoch - 148ms/step
93/93 - 3s - loss: 0.8447 - accuracy: 0.6577 - 3s/epoch - 33ms/step
Epoch 1/2
372/372 - 55s - loss: 0.8389 - accuracy: 0.6406 - 55s/epoch - 149ms/step
Epoch 2/2
372/372 - 54s - loss: 0.6864 - accuracy: 0.7072 - 54s/epoch - 145ms/step
93/93 - 2s - loss: 0.7341 - accuracy: 0.6864 - 2s/epoch - 19ms/step
Epoch 1/2
372/372 - 57s - loss: 0.8337 - accuracy: 0.6478 - 57s/epoch - 153ms/step
Epoch 2/2
372/372 - 52s - loss: 0.6827 - accuracy: 0.7121 - 52s/epoch - 139ms/step
93/93 - 2s - loss: 0.7413 - accuracy: 0.6756 - 2s/epoch - 20ms/step
Epoch 1/2
372/372 - 58s - loss: 0.8431 - accuracy: 0.6367 - 58s/epoch - 155ms/step
```

```
Epoch 2/2
372/372 - 52s - loss: 0.6796 - accuracy: 0.7147 - 52s/epoch - 139ms/step
93/93 - 2s - loss: 0.7404 - accuracy: 0.6923 - 2s/epoch - 19ms/step
Epoch 1/2
372/372 - 57s - loss: 0.8332 - accuracy: 0.6459 - 57s/epoch - 153ms/step
Epoch 2/2
372/372 - 52s - loss: 0.6757 - accuracy: 0.7173 - 52s/epoch - 139ms/step
93/93 - 2s - loss: 0.7407 - accuracy: 0.6841 - 2s/epoch - 22ms/step
Epoch 1/2
372/372 - 55s - loss: 0.8297 - accuracy: 0.6421 - 55s/epoch - 147ms/step
Epoch 2/2
372/372 - 54s - loss: 0.6725 - accuracy: 0.7163 - 54s/epoch - 144ms/step
93/93 - 2s - loss: 0.7687 - accuracy: 0.6744 - 2s/epoch - 19ms/step
186/186 - 36s - loss: 0.8419 - accuracy: 0.6361 - 36s/epoch - 196ms/step
47/47 - 1s - loss: 0.7497 - accuracy: 0.6568 - 1s/epoch - 28ms/step
186/186 - 35s - loss: 0.8436 - accuracy: 0.6351 - 35s/epoch - 187ms/step
47/47 - 2s - loss: 0.7758 - accuracy: 0.6600 - 2s/epoch - 35ms/step
186/186 - 37s - loss: 0.8510 - accuracy: 0.6314 - 37s/epoch - 197ms/step
47/47 - 1s - loss: 0.7785 - accuracy: 0.6708 - 1s/epoch - 31ms/step
186/186 - 36s - loss: 0.8472 - accuracy: 0.6340 - 36s/epoch - 194ms/step
47/47 - 1s - loss: 0.7531 - accuracy: 0.6744 - 1s/epoch - 28ms/step
186/186 - 36s - loss: 0.8355 - accuracy: 0.6348 - 36s/epoch - 195ms/step
47/47 - 2s - loss: 0.8012 - accuracy: 0.6685 - 2s/epoch - 53ms/step
Epoch 1/2
186/186 - 35s - loss: 0.8490 - accuracy: 0.6341 - 35s/epoch - 189ms/step
Epoch 2/2
186/186 - 33s - loss: 0.6952 - accuracy: 0.7037 - 33s/epoch - 179ms/step
47/47 - 1s - loss: 0.7331 - accuracy: 0.6686 - 1s/epoch - 28ms/step
Epoch 1/2
186/186 - 39s - loss: 0.8398 - accuracy: 0.6383 - 39s/epoch - 210ms/step
Epoch 2/2
186/186 - 31s - loss: 0.6906 - accuracy: 0.7109 - 31s/epoch - 169ms/step
47/47 - 2s - loss: 0.7313 - accuracy: 0.6885 - 2s/epoch - 47ms/step
Epoch 1/2
186/186 - 37s - loss: 0.8461 - accuracy: 0.6308 - 37s/epoch - 197ms/step
Epoch 2/2
186/186 - 33s - loss: 0.6937 - accuracy: 0.7019 - 33s/epoch - 180ms/step
47/47 - 1s - loss: 0.7497 - accuracy: 0.6853 - 1s/epoch - 28ms/step
Epoch 1/2
186/186 - 36s - loss: 0.8429 - accuracy: 0.6362 - 36s/epoch - 195ms/step
Epoch 2/2
186/186 - 33s - loss: 0.6909 - accuracy: 0.7050 - 33s/epoch - 179ms/step
47/47 - 2s - loss: 0.7457 - accuracy: 0.6868 - 2s/epoch - 47ms/step
Epoch 1/2
186/186 - 35s - loss: 0.8433 - accuracy: 0.6358 - 35s/epoch - 186ms/step
Epoch 2/2
186/186 - 33s - loss: 0.6797 - accuracy: 0.7110 - 33s/epoch - 178ms/step
47/47 - 1s - loss: 0.7839 - accuracy: 0.6679 - 1s/epoch - 31ms/step
Epoch 1/2
465/465 - 70s - loss: 0.8184 - accuracy: 0.6461 - 70s/epoch - 151ms/step
```

```
Epoch 2/2
465/465 - 67s - loss: 0.6756 - accuracy: 0.7132 - 67s/epoch - 143ms/step
Best: 0.682556 using {'batch_size': 20, 'epochs': 2}
```