```
import codecs, os, numpy as np, pandas as pd
pd.set_option('display.max_colwidth', -1) #to prevent cell display truncation

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model
from keras import initializers, regularizers, constraints, optimizers, layers
#initializers, regularizers, constraints are not explicitly called

from sklearn.utils import shuffle
from sklearn import preprocessing

from keras.callbacks import EarlyStopping, ModelCheckpoint#options for when to stop trainin
#and to save only the best performing iteration

from keras.callbacks import TensorBoard#visualize the neural network's training
from time import time#needed for Tensorboard config

import re#for text preprocessing (checking for non words)
```

```
Using TensorFlow backend.
```

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947

Enter your authorization code:
··········
Mounted at /content/gdrive
```

```
temp_1 = open("/content/gdrive/My Drive/Colab Notebooks/WhoseLineIsItAnywayTRAIN.csv", 'r',
WhoseLineData = pd.read_csv(temp_1)
WhoseLineData = shuffle(WhoseLineData)
print(WhoseLineData.shape)
temp_1 = open("/content/gdrive/My Drive/Colab Notebooks/WhoseLineIsItAnywayTEST.csv", 'r',
WhoseLineDataTest = pd.read_csv(temp_1)
print(WhoseLineDataTest.shape)
```

```
(18977, 2)
(6326, 1)
```

```
#knock out the many white white spaces & line breaks
WhoseLineData['text'].replace(r'\s+', ' ', regex=True, inplace=True)
WhoseLineDataTest['text'].replace(r'\s+', ' ', regex=True, inplace=True)

WhoseLineData['text'] = WhoseLineData['text'].str.lower().str.replace("[^a-zA-Z]", " ")
WhoseLineDataTest['text'] = WhoseLineDataTest['text'].str.lower().str.replace("[^a-zA-Z]",

#WhoseLineDataTest['text']= re.sub("[^a-zA-Z]+"  " "  WhoseLineDataTest['text'])
```

```
#whoselinedatatest['text']= re.sub('[^a-zA-Z]+', ' ', whoselinedatatest['text'])
#checks for non words
WhoseLineDataText = ' '.join(WhoseLineData['text'].str.lower())
WhoseLineDataText1 = re.sub("[^a-zA-Z]+", " ", WhoseLineDataText)
print(len(WhoseLineDataText))
print(len(WhoseLineDataText1))
```

```
42965369
39669858
```

```
#unique word count, needed to set the max_features
uniqueWords = set()
WhoseLineData['text'].str.split().apply(uniqueWords.update)
print('# of unique words',len(uniqueWords))
print('Sample of unique words',list(uniqueWords)[1070:1100])
```

```
# of unique words 65924
Sample of unique words ['barrancas', 'outwards', 'buglos', 'envelop', 'mikado', 'aud
```

```
#to get an idea for the typical length of each text, to set the maxlen parameter later
WhoseLineData['textLength'] = WhoseLineData['text'].str.split().str.len()
```

```
WhoseLineData.describe()
```

	author	textLength
count	18977.000000	18977.000000
mean	3.667176	400.118617
std	2.678708	312.926616
min	0.000000	23.000000
25%	2.000000	251.000000
50%	4.000000	339.000000
75%	5.000000	465.000000
max	9.000000	5908.000000

```
#this is needed to define the number of nodes in the final layer of the neural network
WhoseLineData['author'].unique()
```

```
array([8, 0, 2, 5, 3, 4, 1, 9, 6, 7])
```

```
#finding the shortest text, displaying the row + splitting & unique words count (set)
print('Row with the shortest text is:', WhoseLineData['textLength'].idxmin())
print('Shortest text:', '\n',WhoseLineData['text'][135])
print('List of unique words:', '\n', WhoseLineData['text'][135].split())
print('#of unique words:', len(set(WhoseLineData['text'][135].split())))
```

```
Row with the shortest text is: 135
```

```
Shortest text:
 but what if my heart had failed me  or i had shrunk from making up my mind      no
List of unique words:
 ['but', 'what', 'if', 'my', 'heart', 'had', 'failed', 'me', 'or', 'i', 'had', 'shru
```

```
#finding the longest text, splitting & unique words count (set), and display some of the
#unique words. Note the presence of special characters despite running regex above
print('Row with the longest text is:', WhoseLineData['textLength'].idxmax())
print('#of unique words:', len(set(WhoseLineData['text'][16960].split())))
print('Sample of unique words:', '\n', list(set(WhoseLineData['text'][16960].split()))[0:50
```

```
Row with the longest text is: 16960
#of unique words: 1110
Sample of unique words:
 ['bones', 'risk', 'scarce', 'fast', 'pocket', 'talk', 'tell', 'mate', 'awhile', 'cl
```

```
EMBEDDING_FILE='/content/gdrive/My Drive/Colab Notebooks/glove.6B.50d.txt'#word emdedding f
embed_size = 50 # word vector/ embedding size
max_features = 20000 # number unique words to use (i.e num rows in embedding vector)
maxlen = 200 # max number of words used, from each row of text
```

```
#splitting train and test data
train = WhoseLineData.iloc[0:18970,]
#test = WhoseLineData.iloc[18970:18977,]
test = WhoseLineDataTest

list_sentences_train = train["text"]
list_sentences_test = test["text"]

#one hot encoding the labels
y = train['author']
y=y.values.reshape(-1,1)
enc = preprocessing.OneHotEncoder(categorical_features = [0])
y1 = enc.fit_transform(y).toarray()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/_encoders.py:368: Futur
If you want the future behaviour and silence this warning, you can specify "categori
In case you used a LabelEncoder before this OneHotEncoder to convert the categories
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/_encoders.py:390: Depre
  "use the ColumnTransformer instead.", DeprecationWarning)
```

```
len(list_sentences_test)#confirming test data size
```

```
6326
```

```
#checking the transformed labels
print(train['author'][0:5])
print(y1[0:5,:])
```

```
8105     8
```

```
10826    0
10251    2
5388     5
15103    3
Name: author, dtype: int64
[[0. 0. 0. 0. 0. 0. 0. 1. 0.]
```

```
#convert texts to lists of word indices + padding lists so they are all of equal length
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(list_sentences_train))
list_tokenized_train = tokenizer.texts_to_sequences(list_sentences_train)
list_tokenized_test = tokenizer.texts_to_sequences(list_sentences_test)
X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
```

```
#loading word embedding
def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.strip().split()) for o in open(EMBEDDING_FILE))
```

```
#this and the next cell convert word indices to word embeddings, where available. use rando
all_embs = np.stack(embeddings_index.values())
emb_mean,emb_std = all_embs.mean(), all_embs.std()
emb_mean,emb_std
```

```
(0.020940498, 0.6441043)
```

```
word_index = tokenizer.word_index
nb_words = min(max_features, len(word_index)+1)
embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector
```

```
#definining a bi-directional LSTM
inp = Input(shape=(maxlen,))
x = Embedding(max_features, embed_size, weights=[embedding_matrix])(inp)
#x = Embedding(nb_words, embed_size, weights=[embedding_matrix])(inp)
x = Bidirectional(LSTM(50, return_sequences=True, dropout=0.1, recurrent_dropout=0.1))(x)
x = Bidirectional(LSTM(50, return_sequences=True, dropout=0.1, recurrent_dropout=0.1))(x)
x = Bidirectional(LSTM(50, return_sequences=True, dropout=0.1, recurrent_dropout=0.1))(x)
x = GlobalMaxPool1D()(x)
x = Dense(50, activation="relu")(x)
x = Dropout(0.1)(x)
x = Dense(10, activation="sigmoid")(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
#the tensorboard worked while training on the local machine, but not on colab
tensorboard = TensorBoard(log_dir="logs/{}".format(time()))
!pip install tensorboardcolab
from tensorboardcolab import TensorBoardColab, TensorBoardColabCallback
tbc=TensorBoardColab()
```

```
#training for more epochs seemed to help. the callback helps prevents over doing it. probab
#checkpoint as well to save the best performing iteration. checkpoint = [ModelCheckpoint(fi
```

```
callbacks = [EarlyStopping(monitor='val_loss', patience=2)]
```

```
#model.fit(X_t, y1, batch_size=1000, epochs=50, callbacks=[TensorBoardColabCallback(tbc)],
model.fit(X_t, y1, batch_size=1000, epochs=50, callbacks=callbacks, validation_split=0.1)
```

```
Train on 17073 samples, validate on 1897 samples
```

```
Epoch 1/50
17073/17073 [==============================] - 71s 4ms/step - loss: 2.2102 - acc: 0.
Epoch 2/50
17073/17073 [==============================] - 64s 4ms/step - loss: 2.1125 - acc: 0.
Epoch 3/50
```

```
y_test = model.predict([X_te], batch_size=1000, verbose=1)
```

```
6326/6326 [==============================] - 10s 2ms/step
```

```
17073/17073 [==============================] - 64s 4ms/step - loss: 1.9028 - acc: 0.
```

```
test.reset_index(drop=True, inplace=True)
testPredict = pd.DataFrame(y_test)
PredAuthor = testPredict.idxmax(axis=1)
PredAuthor.columns = ['author']
```

```
Epoch 8/50
```

```
!pip install openpyxl
writer = pd.ExcelWriter('/content/gdrive/My Drive/Colab Notebooks/WhoseLinePredAuthor201812
PredAuthor.to_excel(writer,'Sheet1')
writer.save()
```

```
Requirement already satisfied: openpyxl in /usr/local/lib/python3.6/dist-packages (2
Requirement already satisfied: jdcal in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: et_xmlfile in /usr/local/lib/python3.6/dist-packages
```

```
Epoch 14/50
17073/17073 [==============================] - 64s 4ms/step - loss: 1.4949 - acc: 0.
Epoch 15/50
17073/17073 [==============================] - 64s 4ms/step - loss: 1.4220 - acc: 0.
Epoch 16/50
17073/17073 [==============================] - 64s 4ms/step - loss: 1.3349 - acc: 0.
Epoch 17/50
17073/17073 [==============================] - 64s 4ms/step - loss: 1.2535 - acc: 0.
Epoch 18/50
17073/17073 [==============================] - 64s 4ms/step - loss: 1.1711 - acc: 0.
Epoch 19/50
17073/17073 [==============================] - 64s 4ms/step - loss: 1.0862 - acc: 0.
Epoch 20/50
17073/17073 [==============================] - 64s 4ms/step - loss: 0.9849 - acc: 0.
Epoch 21/50
17073/17073 [==============================] - 64s 4ms/step - loss: 0.9023 - acc: 0.
Epoch 22/50
17073/17073 [==============================] - 65s 4ms/step - loss: 0.8287 - acc: 0.
Epoch 23/50
17073/17073 [==============================] - 64s 4ms/step - loss: 0.7695 - acc: 0.
Epoch 24/50
17073/17073 [==============================] - 65s 4ms/step - loss: 0.7087 - acc: 0.
Epoch 25/50
17073/17073 [==============================] - 64s 4ms/step - loss: 0.6814 - acc: 0.
Epoch 26/50
17073/17073 [==============================] - 64s 4ms/step - loss: 0.6172 - acc: 0.
```