LSTM_Tweets_Data

April 14, 2024

[1]:
```
import json
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.regularizers import L1, L2
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.callbacks import EarlyStopping

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
```

[2]:
```
tf.config.list_physical_devices('GPU')
```

[2]:
```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

[3]:
```
df = pd.read_csv('emotion_dataset.csv')
df = df.drop_duplicates()
df = df.dropna()
df.head()
```

[3]:

	text	Emotion
0	awww bummer shoulda got david carr third day	0
1	upset update facebook texting might cry result…	0
2	dived many times ball managed save 50 rest go …	0
3	whole body feels itchy like fire	0
4	behaving mad see	0

[4]:
```
df
```

[4]:
```
                                                     text  Emotion
0                  awww bummer shoulda got david carr third day        0
1             upset update facebook texting might cry result…        0
2             dived many times ball managed save 50 rest go …        0
3                               whole body feels itchy like fire        0
4                                            behaving mad see        0
…                                                          …      …
1599995                        woke school best feeling ever        4
1599996           thewdb com cool hear old walt interviews        4
1599997                    ready mojo makeover ask details        4
1599998  happy 38th birthday boo alll time tupac amaru …        4
1599999  happy charitytuesday thenspcc sparkscharity sp…        4

[1489715 rows x 2 columns]
```

[5]:
```
def safe_lower(input_text):
    if pd.isnull(input_text):
        return None  # Choose how to handle nulls based on your context
    return str(input_text).lower()

df['text'] = df['text'].apply(safe_lower)
```

[6]:
```
print(df['text'][0])
```

```
awww bummer shoulda got david carr third day
```

[7]:
```
print(f"Unique emotions: {df['Emotion'].unique()}\n Number of unique emotions:␣
↪{df['Emotion'].nunique()}")
```

```
Unique emotions: [0 4]
 Number of unique emotions: 2
```

We need to convert our labels into numeric values so the model can work with them

[8]:
```
df['Label'], Emotions = pd.factorize(df['Emotion'])
df['Label']
```

[8]:
```
0          0
1          0
2          0
3          0
4          0
          ..
1599995    1
1599996    1
1599997    1
1599998    1
1599999    1
```

```
Name: Label, Length: 1489715, dtype: int64
```

[9]:
```
X = df['text']
y = df['Label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=37)
```

Before we can feed our inputs into the model, we need to convert them into a form the computer can read: numbers. Tensorflow has a tokenizer method built in, which essentially creates a dictionary of words from a series of texts.

Once we have all of our tokens, we can transform our lines of text into sequences of tokens which can be fed into the model.

[10]:
```
tokenizer = Tokenizer(num_words = 60000, oov_token = "<OOV>")
tokenizer.fit_on_texts(X_train)
tokenizer.fit_on_texts(X_test)

X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_test_sequences = tokenizer.texts_to_sequences(X_test)

X_train_sequences[0]
```

[10]:
```
[308, 1583, 115, 157, 4, 2697, 339, 67]
```

Our model expects a uniform input size. To accomplish this we pad the sequences with 0's so that each input is equal in size to our largest input.

[11]:
```
maxlen = max(len(tokens) for tokens in X_train_sequences)
print("Maximum sequence length (maxlen):", maxlen)

X_train_padded = pad_sequences(X_train_sequences, maxlen=maxlen, padding='post')
X_test_padded = pad_sequences(X_test_sequences, maxlen=maxlen, padding='post')
```

```
Maximum sequence length (maxlen): 50
```

[12]:
```
X_train_padded[0]
```

[12]:
```
array([ 308, 1583,  115,  157,    4, 2697,  339,   67,    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], dtype=int32)
```

[13]:
```
# Define the model
model = Sequential()
model.add(Embedding(input_dim=np.max(X_train_padded)+1,␣
↪output_dim=100,input_shape=(maxlen,)))
```

```
model.add(Bidirectional(LSTM(128)))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu',
                kernel_regularizer=L1(0.01),
                activity_regularizer=L2(0.01)))
model.add(Dropout(0.5))
model.add(Dense(13, activation='softmax'))
```

[14]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 100)           6000000

 bidirectional (Bidirection  (None, 256)               234496
 al)

 batch_normalization (Batch  (None, 256)               1024
 Normalization)

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 64)                16448

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 13)                845

=================================================================
Total params: 6252813 (23.85 MB)
Trainable params: 6252301 (23.85 MB)
Non-trainable params: 512 (2.00 KB)
_________________________________________________________________
```

[15]:
```
history = model.fit(X_train_padded, y_train, epochs=5, batch_size=32,
                    validation_data=(X_test_padded, y_test))
```

```
Epoch 1/5
37243/37243 [==============================] - 478s 13ms/step - loss: 0.6191 -
accuracy: 0.7634 - val_loss: 0.5391 - val_accuracy: 0.7824
Epoch 2/5
37243/37243 [==============================] - 442s 12ms/step - loss: 0.5414 -
```

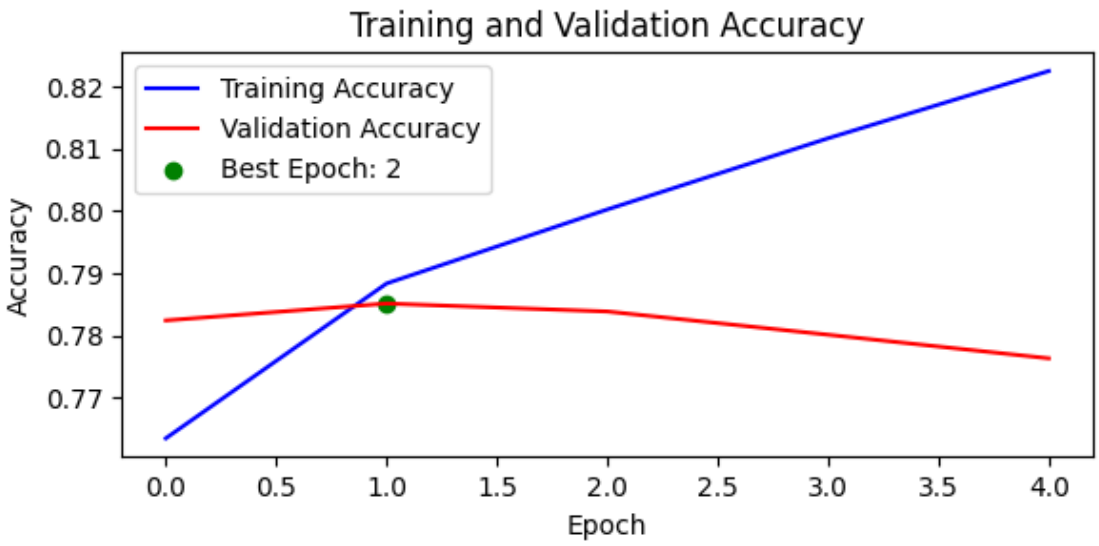
```
accuracy: 0.7883 - val_loss: 0.5237 - val_accuracy: 0.7851
Epoch 3/5
37243/37243 [==============================] - 434s 12ms/step - loss: 0.5176 -
accuracy: 0.8003 - val_loss: 0.5433 - val_accuracy: 0.7838
Epoch 4/5
37243/37243 [==============================] - 431s 12ms/step - loss: 0.4977 -
accuracy: 0.8118 - val_loss: 0.5498 - val_accuracy: 0.7801
Epoch 5/5
37243/37243 [==============================] - 472s 13ms/step - loss: 0.4789 -
accuracy: 0.8226 - val_loss: 0.5557 - val_accuracy: 0.7763
```

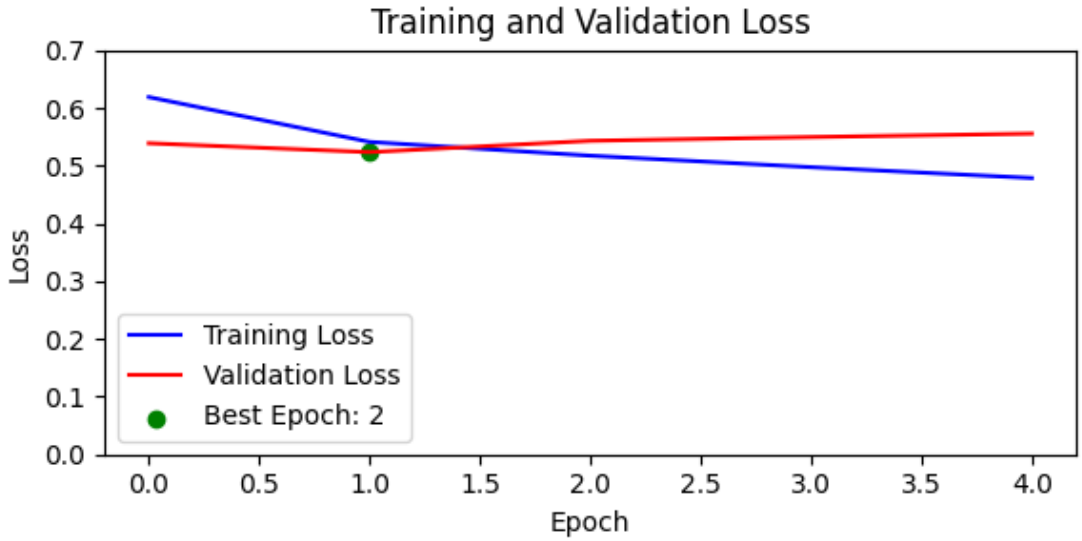
[16]:
```
history_dict = history.history
json.dump(history_dict, open('LSTM_history','w'))
```

[17]:
```
model.save('LSTM_Model.keras')
```

[20]:
```
history = json.load(open("LSTM_history",'r'))
```

[21]:
```
best_epoch = history['val_accuracy'].index(max(history['val_accuracy'])) + 1

fig, axs = plt.subplots(2, 1, figsize=(6, 6))

# Plot training and validation accuracy
axs[0].plot(history['accuracy'], label='Training Accuracy', color='blue')
axs[0].plot(history['val_accuracy'], label='Validation Accuracy', color='red')
axs[0].scatter(best_epoch - 1, history['val_accuracy'][best_epoch - 1],␣
 ↪color='green', label=f'Best Epoch: {best_epoch}')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Accuracy')
axs[0].set_title('Training and Validation Accuracy')
axs[0].legend()

# Plot training and validation loss
axs[1].plot(history['loss'], label='Training Loss', color='blue')
axs[1].plot(history['val_loss'], label='Validation Loss', color='red')
axs[1].scatter(best_epoch - 1, history['val_loss'][best_epoch - 1],␣
 ↪color='green',label=f'Best Epoch: {best_epoch}')
axs[1].set_ylim([0,.7])
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Loss')
axs[1].set_title('Training and Validation Loss')
axs[1].legend()

plt.tight_layout()
plt.show()
```





[22]:
```
model = tf.keras.models.load_model("LSTM_Model.keras")
```

[23]:
```
y_pred = model.predict(X_test_padded)
y_pred = np.argmax(y_pred, axis=1)
```

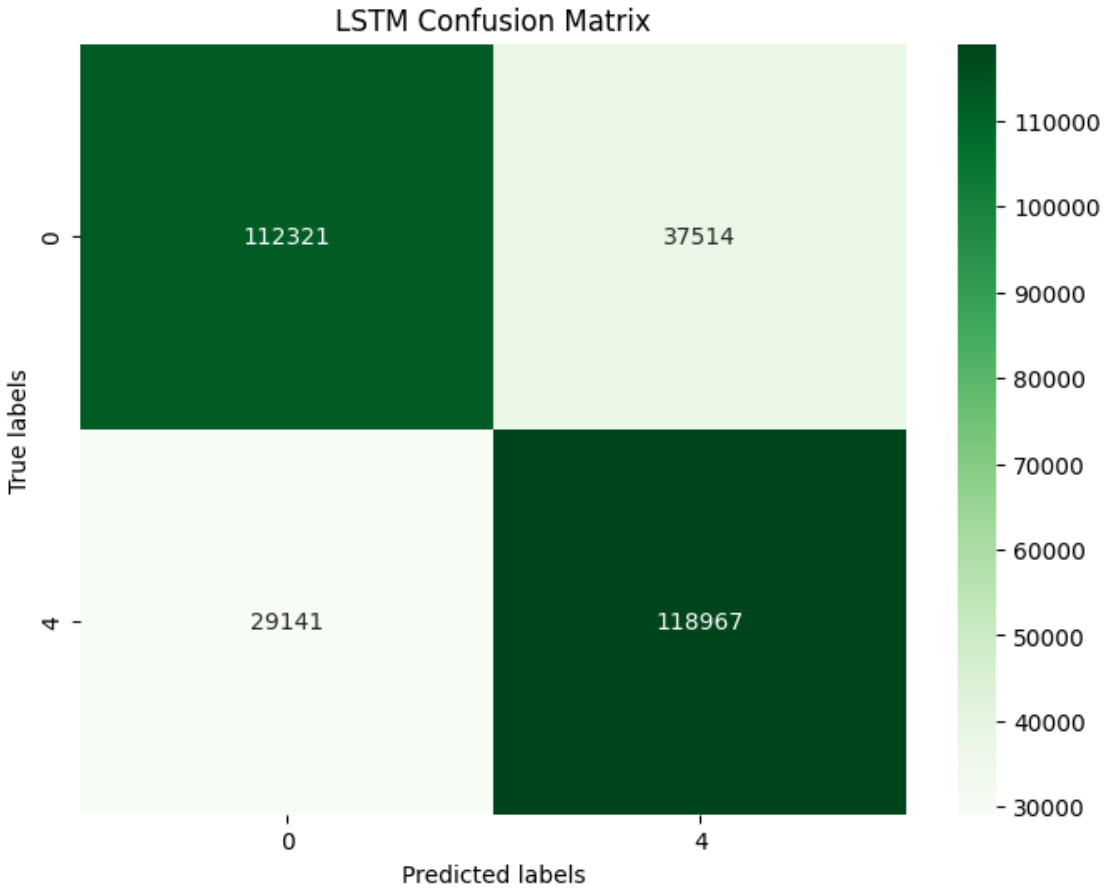
```
9311/9311 [==============================] - 33s 4ms/step
```

[24]:
```
print(classification_report(y_test,y_pred,labels = list(np.
↪arange(2)),target_names = ['0', '4'], zero_division = 0.0))
```

	precision	recall	f1-score	support
0	0.79	0.75	0.77	149835

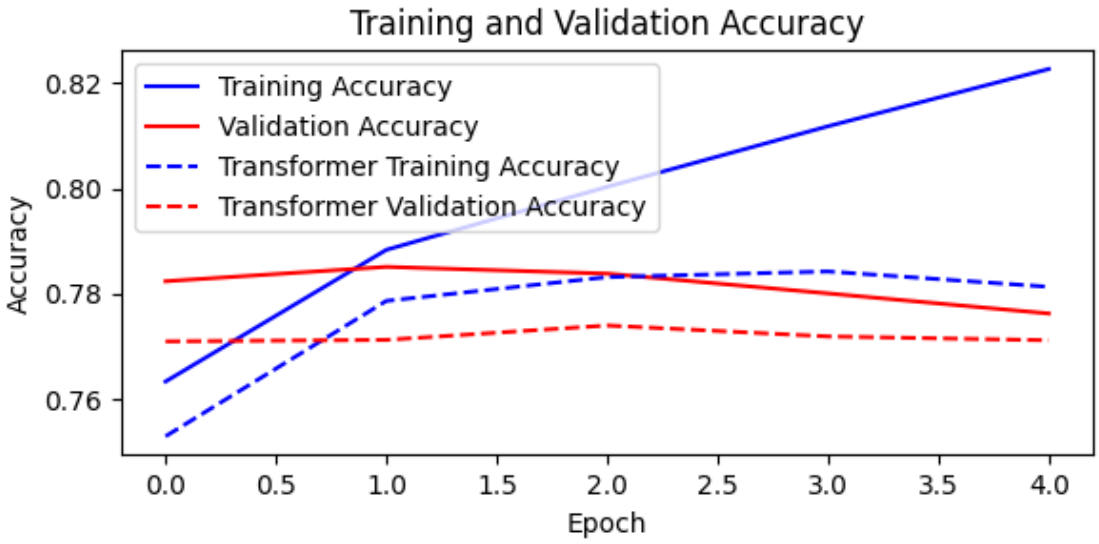
```
              4       0.76      0.80      0.78    148108

       accuracy                           0.78    297943
      macro avg       0.78      0.78      0.78    297943
   weighted avg       0.78      0.78      0.78    297943
```

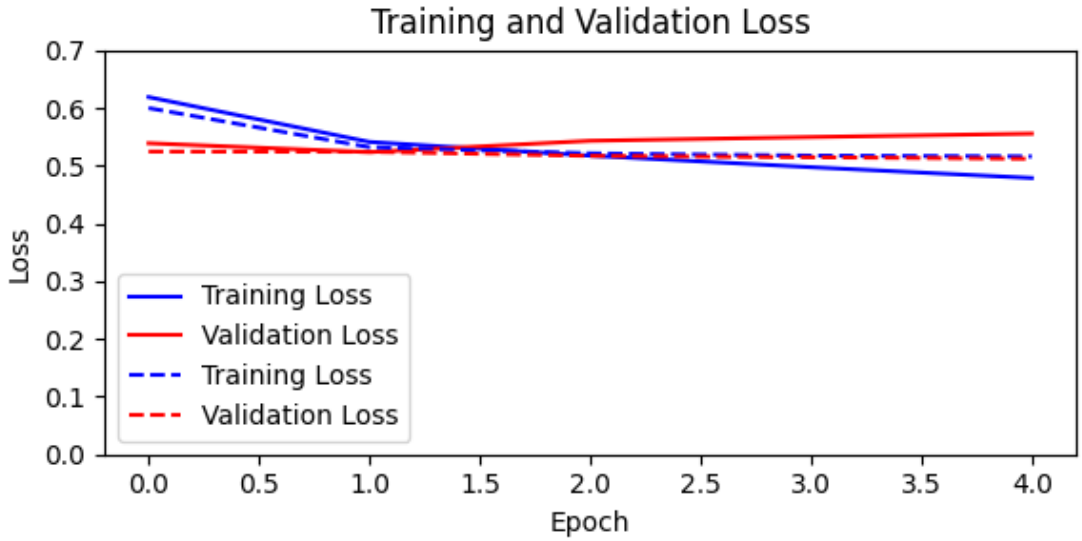
[26]:
```
# y_test and y_pred are your true and predicted labels
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot = True, fmt='d', cmap = 'Greens',
            xticklabels = ['0','4'],
            yticklabels = ['0','4'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('LSTM Confusion Matrix')
plt.show()
```



[27]:
```
T_history = json.load(open("transformer_history",'r'))
```

[28]:
```
best_epoch = history['val_accuracy'].index(max(history['val_accuracy'])) + 1

fig, axs = plt.subplots(2, 1, figsize=(6, 6))

# Plot training and validation accuracy
axs[0].plot(history['accuracy'], label='Training Accuracy', color='blue')
axs[0].plot(history['val_accuracy'], label='Validation Accuracy', color='red')
axs[0].plot(T_history['accuracy'], label='Transformer Training Accuracy',␣
 ↪color='blue', linestyle = 'dashed')
axs[0].plot(T_history['val_accuracy'], label='Transformer Validation Accuracy',␣
 ↪color='red', linestyle = 'dashed')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Accuracy')
axs[0].set_title('Training and Validation Accuracy')
axs[0].legend()

# Plot training and validation loss
axs[1].plot(history['loss'], label='Training Loss', color='blue')
axs[1].plot(history['val_loss'], label='Validation Loss', color='red')
axs[1].plot(T_history['loss'], label='Training Loss', color='blue', linestyle =␣
 ↪'dashed')
axs[1].plot(T_history['val_loss'], label='Validation Loss', color='red',␣
 ↪linestyle = 'dashed')
axs[1].set_ylim([0,.7])
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Loss')
axs[1].set_title('Training and Validation Loss')
axs[1].legend()

plt.tight_layout()
plt.show()
```





0.0.1 Generative Text

[ ]:
```
songs = pd.read_excel(r"English_Lyrics.xlsx")
songs.head()
```

[ ]:

	Unnamed: 0	Lyric
0	69	I feel so unsure\nAs I take your hand and lead…
1	86	Don't let them fool, ya\nOr even try to school…
2	88	Baby, let's cruise, away from here\nDon't be c…
3	111	Know it sounds funny\nBut, I just can't stand …
4	140	You've got that look again\nThe one I hoped I …

[ ]:
```
songs_en = songs[songs.language == 'en']
```

[ ]:
```
songs_en['Lyric'].to_excel("English_Lyrics.xlsx", sheet_name = 'Lyrics')
```

[ ]:
```
# select subset of songs cause I'm not Amazon
songs_en = songs[:250]
```

[ ]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(songs_en['Lyric'].astype(str).str.lower())

total_words = len(tokenizer.word_index)+1
tokenized_sentences = tokenizer.texts_to_sequences(songs_en['Lyric'].
 ↪astype(str))
```

[ ]:
```
# Create n-gram sequences
input_sequences = list()
for i in tokenized_sentences:
    for t in range(1, len(i)):
        n_gram_sequence = i[:t+1]
        input_sequences.append(n_gram_sequence)

# Pre-padding
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences,␣
 ↪maxlen=max_sequence_len, padding='pre'))
```

[ ]:
```
# Creating predictors and labels
X, labels = input_sequences[:,:-1],input_sequences[:,-1]
y = tf.keras.utils.to_categorical(labels, num_classes=total_words)
```

[ ]:
```
model = Sequential() # Initialization
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dropout(0.1))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
earlystop = EarlyStopping(monitor='loss', min_delta=0, patience=3, verbose=0,␣
 ↪mode='auto')
history = model.fit(X, y, epochs=10, verbose=1, callbacks=[earlystop])
```

```
Epoch 1/10
2972/2972 [==============================] - 372s 123ms/step - loss: 5.6778 -
accuracy: 0.0916
Epoch 2/10
2972/2972 [==============================] - 269s 90ms/step - loss: 4.5900 -
accuracy: 0.1980
Epoch 3/10
2972/2972 [==============================] - 262s 88ms/step - loss: 3.8806 -
```

```
accuracy: 0.2855
Epoch 4/10
2972/2972 [==============================] - 259s 87ms/step - loss: 3.3586 -
accuracy: 0.3570
Epoch 5/10
2972/2972 [==============================] - 258s 87ms/step - loss: 2.9527 -
accuracy: 0.4185
Epoch 6/10
2972/2972 [==============================] - 257s 86ms/step - loss: 2.6298 -
accuracy: 0.4673
Epoch 7/10
2972/2972 [==============================] - 257s 86ms/step - loss: 2.3559 -
accuracy: 0.5128
Epoch 8/10
2972/2972 [==============================] - 258s 87ms/step - loss: 2.1320 -
accuracy: 0.5516
Epoch 9/10
2972/2972 [==============================] - 258s 87ms/step - loss: 1.9374 -
accuracy: 0.5879
Epoch 10/10
2972/2972 [==============================] - 265s 89ms/step - loss: 1.7723 -
accuracy: 0.6161
```

[ ]:
```
model.save('LSTM_Gen_Model')
```

```
/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(
```

[ ]:
```
history_dict = history.history
json.dump(history_dict, open('LSTM_Gen_history2','w'))
```

[ ]:
```
del model
model = tf.keras.models.load_model(r"C:
↪\Users\danfe\OneDrive\Desktop\School_Work\MATH␣
↪6373\Project\LSTM\LSTM_Gen_Model.keras")
```

[ ]:
```
def generate_lyrics(seed_text, next_words):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_sequence_len-1,␣
↪padding='pre')
        predictions = model.predict(token_list, verbose=0)
        choice = np.random.choice([1,2,3])
        # sort ascending and select from 3 highest probabilities
```

```
        predicted = np.argsort(predictions)[0][-choice]

        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text
```

[ ]:
```
seed_text = 'Lie'
generate_lyrics(seed_text, 20)
```

[ ]:
```
"Lie baby i know you want my baby baby i'm gonna be the only girl i see and you
and i"
```

[ ]:
```
history_dict = history.history
json.dump(history_dict, open('LSTM_Gen_history','w'))
```

[ ]:
```
new_model = tf.keras.models.load_model(r"C:
 ↪\Users\danfe\Downloads\LSTM_Gen_Model.keras")
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Cell In[5], line 1
----> 1 new_model =␣
 ↪tf.keras.models.load_model(r"C:\Users\danfe\Downloads\LSTM_Gen_Model.keras")

File ~\anaconda3\envs\Tensor_gpu\lib\site-packages\keras\utils\traceback_utils.
 ↪py:70, in filter_traceback.<locals>.error_handler(*args, **kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb

File ~\anaconda3\envs\Tensor_gpu\lib\site-packages\h5py\_hl\files.py:562, in␣
 ↪File.__init__(self, name, mode, driver, libver, userblock_size, swmr,␣
 ↪rdcc_nslots, rdcc_nbytes, rdcc_w0, track_order, fs_strategy, fs_persist,␣
 ↪fs_threshold, fs_page_size, page_buf_size, min_meta_keep, min_raw_keep,␣
 ↪locking, alignment_threshold, alignment_interval, meta_block_size, **kwds)
    553     fapl = make_fapl(driver, libver, rdcc_nslots, rdcc_nbytes, rdcc_w0,
    554                      locking, page_buf_size, min_meta_keep, min_raw_kee ,
    555                      alignment_threshold=alignment_threshold,
    556                      alignment_interval=alignment_interval,
    557                      meta_block_size=meta_block_size,
    558                      **kwds)
```

```
    559     fcpl = make_fcpl(track_order=track_order, fs_strategy=fs_strategy,
    560                      fs_persist=fs_persist, fs_threshold=fs_threshold,
    561                      fs_page_size=fs_page_size)
--> 562     fid = make_fid(name, mode, userblock_size, fapl, fcpl, swmr=swmr)
    564 if isinstance(libver, tuple):
    565     self._libver = libver

File ~\anaconda3\envs\Tensor_gpu\lib\site-packages\h5py\_hl\files.py:235, in␣
 ↪make_fid(name, mode, userblock_size, fapl, fcpl, swmr)
    233     if swmr and swmr_support:
    234         flags |= h5f.ACC_SWMR_READ
--> 235     fid = h5f.open(name, flags, fapl=fapl)
    236 elif mode == 'r+':
    237     fid = h5f.open(name, h5f.ACC_RDWR, fapl=fapl)

File h5py\_objects.pyx:54, in h5py._objects.with_phil.wrapper()

File h5py\_objects.pyx:55, in h5py._objects.with_phil.wrapper()

File h5py\h5f.pyx:102, in h5py.h5f.open()

OSError: Unable to synchronously open file (file signature not found)
```

[ ]: