LSTM Implementation

Text Preprocessing

In [1]:
```
!pip install tensorflow
import os
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
import nltk
import re
```

```
Requirement already satisfied: tensorflow in c:\users\asus\anaconda3\lib\site
-packages (2.17.0)
Requirement already satisfied: tensorflow-intel==2.17.0 in c:\users\asus\anac
onda3\lib\site-packages (from tensorflow) (2.17.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\asus\anaconda3\lib
\site-packages (from tensorflow-intel==2.17.0->tensorflow) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\asus\anaconda3\l
ib\site-packages (from tensorflow-intel==2.17.0->tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in c:\users\asus\anaconda
3\lib\site-packages (from tensorflow-intel==2.17.0->tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in c:\user
s\asus\anaconda3\lib\site-packages (from tensorflow-intel==2.17.0->tensorflo
w) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\asus\anaconda3
\lib\site-packages (from tensorflow-intel==2.17.0->tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in c:\users\asus\anaconda3\lib\si
te-packages (from tensorflow-intel==2.17.0->tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\asus\anaconda3\li
b\site-packages (from tensorflow-intel==2.17.0->tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in c:\users\asus\anaco
nda3\lib\site-packages (from tensorflow-intel==2.17.0->tensorflow) (0.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\asus\anaconda3\l
ib\site-packages (from tensorflow-intel==2.17.0->tensorflow) (3.3.0)
Requirement already satisfied: packaging in c:\users\asus\anaconda3\lib\site-
packages (from tensorflow-intel==2.17.0->tensorflow) (23.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=
4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in c:\users\asus\anaconda3\lib\site-packag
es (from tensorflow-intel==2.17.0->tensorflow) (3.20.3)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\asus\anaconda3
\lib\site-packages (from tensorflow-intel==2.17.0->tensorflow) (2.31.0)
Requirement already satisfied: setuptools in c:\users\asus\anaconda3\lib\site
-packages (from tensorflow-intel==2.17.0->tensorflow) (68.0.0)
Requirement already satisfied: six>=1.12.0 in c:\users\asus\anaconda3\lib\sit
e-packages (from tensorflow-intel==2.17.0->tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\asus\anaconda3\li
b\site-packages (from tensorflow-intel==2.17.0->tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\asus\anac
onda3\lib\site-packages (from tensorflow-intel==2.17.0->tensorflow) (4.7.1)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\asus\anaconda3\lib\s
ite-packages (from tensorflow-intel==2.17.0->tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\asus\anaconda3
\lib\site-packages (from tensorflow-intel==2.17.0->tensorflow) (1.65.4)
Requirement already satisfied: tensorboard<2.18,>=2.17 in c:\users\asus\anaco
nda3\lib\site-packages (from tensorflow-intel==2.17.0->tensorflow) (2.17.0)
Requirement already satisfied: keras>=3.2.0 in c:\users\asus\anaconda3\lib\si
te-packages (from tensorflow-intel==2.17.0->tensorflow) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\use
rs\asus\anaconda3\lib\site-packages (from tensorflow-intel==2.17.0->tensorflo
w) (0.31.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in c:\users\asus\anaconda
3\lib\site-packages (from tensorflow-intel==2.17.0->tensorflow) (1.24.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\asus\anaconda3
\lib\site-packages (from astunparse>=1.6.0->tensorflow-intel==2.17.0->tensorf
low) (0.38.4)
Requirement already satisfied: rich in c:\users\asus\anaconda3\lib\site-packa
ges (from keras>=3.2.0->tensorflow-intel==2.17.0->tensorflow) (13.7.1)
Requirement already satisfied: namex in c:\users\asus\anaconda3\lib\site-pack
```

```
ages (from keras>=3.2.0->tensorflow-intel==2.17.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in c:\users\asus\anaconda3\lib\site-pac
kages (from keras>=3.2.0->tensorflow-intel==2.17.0->tensorflow) (0.12.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\asus\anac
onda3\lib\site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.17.0->
tensorflow) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\asus\anaconda3\lib\si
te-packages (from requests<3,>=2.21.0->tensorflow-intel==2.17.0->tensorflow)
(3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\asus\anaconda3
\lib\site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.17.0->tenso
rflow) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\asus\anaconda3
\lib\site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.17.0->tenso
rflow) (2023.7.22)
Requirement already satisfied: markdown>=2.6.8 in c:\users\asus\anaconda3\lib
\site-packages (from tensorboard<2.18,>=2.17->tensorflow-intel==2.17.0->tenso
rflow) (3.4.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\us
ers\asus\anaconda3\lib\site-packages (from tensorboard<2.18,>=2.17->tensorflo
w-intel==2.17.0->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\asus\anaconda3\lib
\site-packages (from tensorboard<2.18,>=2.17->tensorflow-intel==2.17.0->tenso
rflow) (2.2.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\asus\anaconda3\l
ib\site-packages (from werkzeug>=1.0.1->tensorboard<2.18,>=2.17->tensorflow-i
ntel==2.17.0->tensorflow) (2.1.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\users\asus\anacond
a3\lib\site-packages (from rich->keras>=3.2.0->tensorflow-intel==2.17.0->tens
orflow) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\asus\anaco
nda3\lib\site-packages (from rich->keras>=3.2.0->tensorflow-intel==2.17.0->te
nsorflow) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in c:\users\asus\anaconda3\lib\site
-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow-intel==
2.17.0->tensorflow) (0.1.0)
```

In [7]:

```python
# Specify the filename
input_file = 'holmes.txt'

# Read the contents of the file
with open(input_file, 'r', encoding='utf-8') as infile:
    data = infile.read()
```

In [8]:

```python
data[:100] # view first few characters
```

Out[8]:

```
"*Project Gutenberg's Etext of Tom Swift And His Submarine Boat*\n\n#4 in the
Victor Appleton's Tom Swi"
```

In [9]:

```python
# Limit data to 500000 characters
data = data[:500000]
```

Clean Text

In [25]:

```
# Function to remove emojis and special characters from text
def remove_emojis_and_special_characters(text):
    # Remove emojis
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictograp
                               u"\U0001F680-\U0001F6FF"  # transport & map sym
                               u"\U0001F700-\U0001F77F"  # alchemical symbols
                               u"\U0001F780-\U0001F7FF"  # Geometric Shapes Ex
                               u"\U0001F800-\U0001F8FF"  # Supplemental Arrows
                               u"\U0001F900-\U0001F9FF"  # Supplemental Symbol
                               u"\U0001FA00-\U0001FA6F"  # Chess Symbols
                               u"\U0001FA70-\U0001FAFF"  # Symbols and Pictogr
                               u"\U00002702-\U000027B0"  # Dingbats
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)

    # Remove special characters
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)

    # Remove extra spaces
    text = re.sub(' +', ' ', text)

    return text
```

In [26]:

```
# Preprocessing pipeline
def preprocess_pipeline(data) -> 'list':
    # Split by newline character
    sentences = data.split('\n')
    for i in range(len(sentences)):
        sentences[i] = remove_emojis_and_special_characters(sentences[i])
    # Remove leading and trailing spaces
    sentences = [s.strip() for s in sentences]
    # Drop empty sentences
    sentences = [s for s in sentences if len(s) > 0]
    # Tokenization
    tokenized = []
    for sentence in sentences:
        # Convert to lowercase
        sentence = sentence.lower()
        tokenized.append(sentence)
    return tokenized

# Tokenize sentences
tokenized_sentences = preprocess_pipeline(data)
```

In [50]:

```
"""
What is an OOV Token?
An out-of-vocabulary (OOV) token is a special token used in natural language p
are not present in the vocabulary of the model or tokenizer. When a word that
tokenization or text processing, it is replaced with the OOV token.

Why Use an OOV Token?
Using an OOV token helps handle unseen or unknown words during the training or
Instead of encountering errors or issues when encountering unknown words, the
representing them with the OOV token. This is particularly useful when working
of the model may not cover all possible words.
"""
# Tokenize words
tokenizer = Tokenizer(oov_token='<oov>')
tokenizer.fit_on_texts(tokenized_sentences)
total_words = len(tokenizer.word_index) + 1
# tokenizer.word_counts
# tokenizer.word_index
"""
n_gram example:
[3, 15, 8, 7, 20, 12, 6]

For the above sentece sentence, the code would generate the following n-gram s

[3, 15]
[3, 15, 8]
[3, 15, 8, 7]
[3, 15, 8, 7, 20]
[3, 15, 8, 7, 20, 12]
[3, 15, 8, 7, 20, 12, 6]
"""

# Generate input sequences
input_sequences = []
for line in tokenized_sentences:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i + 1]
        input_sequences.append(n_gram_sequence)

# Pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_
```

In [51]:

```
# Creates labels with input sequences
X,labels = input_sequences[:,:-1],input_sequences[:,-1]
ys = tf.keras.utils.to_categorical(labels, num_classes=total_words)
```

In [52]:

```
# Split data into training, validation, and test sets
from sklearn.model_selection import train_test_split
X_train_temp, X_val_test, y_train_temp, y_val_test = train_test_split(X, ys, t
X_val, X_test, y_val, y_test = train_test_split(X_val_test, y_val_test, test_s
```

Train LSTM Model

In [63]:

```
# Define your model
model = Sequential()
model.add(Embedding(total_words, 100))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))

adam = Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accur

# Train the model
history = model.fit(X_train_temp, y_train_temp, epochs=50, validation_data=(X_
Epoch 1/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 36s 17ms/step - accuracy: 0.0740 - loss: 6.637
6 - val_accuracy: 0.1045 - val_loss: 6.1685
Epoch 2/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.1195 - loss: 5.611
3 - val_accuracy: 0.1190 - val_loss: 6.2526
Epoch 3/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.1415 - loss: 5.027
7 - val_accuracy: 0.1121 - val_loss: 6.4698
Epoch 4/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.1687 - loss: 4.520
6 - val_accuracy: 0.1167 - val_loss: 6.8007
Epoch 5/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2009 - loss: 4.126
7 - val_accuracy: 0.1157 - val_loss: 7.1744
Epoch 6/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2167 - loss: 4.024
0 - val_accuracy: 0.1053 - val_loss: 7.5095
Epoch 7/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2338 - loss: 3.785
9 - val_accuracy: 0.1054 - val_loss: 7.8422
Epoch 8/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2510 - loss: 3.618
1 - val_accuracy: 0.1066 - val_loss: 8.1853
Epoch 9/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2565 - loss: 3.549
1 - val_accuracy: 0.1071 - val_loss: 8.4493
Epoch 10/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2684 - loss: 3.456
0 - val_accuracy: 0.1068 - val_loss: 8.7272
Epoch 11/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2752 - loss: 3.393
6 - val_accuracy: 0.1029 - val_loss: 8.9280
Epoch 12/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2777 - loss: 3.370
0 - val_accuracy: 0.1058 - val_loss: 9.1781
Epoch 13/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2842 - loss: 3.338
4 - val_accuracy: 0.1011 - val_loss: 9.4172
Epoch 14/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2889 - loss: 3.275
6 - val_accuracy: 0.1053 - val_loss: 9.6058
Epoch 15/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2910 - loss: 3.266
0 - val_accuracy: 0.1027 - val_loss: 9.7615
Epoch 16/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2915 - loss: 3.269
1 - val_accuracy: 0.1033 - val_loss: 9.9533
Epoch 17/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2905 - loss: 3.260
7 - val_accuracy: 0.0980 - val_loss: 10.1505
Epoch 18/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2927 - loss: 3.253
5 - val_accuracy: 0.0992 - val_loss: 10.2983
Epoch 19/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2940 - loss: 3.257
4 - val_accuracy: 0.1030 - val_loss: 10.4274
```

```
Epoch 20/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2991 - loss: 3.216
7 - val_accuracy: 0.1002 - val_loss: 10.5672
Epoch 21/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2978 - loss: 3.231
6 - val_accuracy: 0.1000 - val_loss: 10.7582
Epoch 22/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2975 - loss: 3.251
1 - val_accuracy: 0.0991 - val_loss: 10.9165
Epoch 23/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3013 - loss: 3.235
0 - val_accuracy: 0.1035 - val_loss: 11.0819
Epoch 24/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2974 - loss: 3.242
2 - val_accuracy: 0.0988 - val_loss: 11.2693
Epoch 25/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3063 - loss: 3.191
7 - val_accuracy: 0.0988 - val_loss: 11.3572
Epoch 26/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3034 - loss: 3.196
6 - val_accuracy: 0.1040 - val_loss: 11.4403
Epoch 27/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3008 - loss: 3.240
8 - val_accuracy: 0.0990 - val_loss: 11.5304
Epoch 28/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.2980 - loss: 3.309
7 - val_accuracy: 0.1030 - val_loss: 11.6778
Epoch 29/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3028 - loss: 3.244
9 - val_accuracy: 0.0990 - val_loss: 11.8247
Epoch 30/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3041 - loss: 3.250
5 - val_accuracy: 0.0973 - val_loss: 11.9230
Epoch 31/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3042 - loss: 3.240
6 - val_accuracy: 0.0982 - val_loss: 12.0616
Epoch 32/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3069 - loss: 3.212
5 - val_accuracy: 0.1019 - val_loss: 12.1948
Epoch 33/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3078 - loss: 3.221
2 - val_accuracy: 0.1007 - val_loss: 12.1885
Epoch 34/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3041 - loss: 3.233
1 - val_accuracy: 0.1027 - val_loss: 12.2329
Epoch 35/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3065 - loss: 3.209
2 - val_accuracy: 0.0993 - val_loss: 12.3775
Epoch 36/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3086 - loss: 3.216
2 - val_accuracy: 0.1043 - val_loss: 12.5220
Epoch 37/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3071 - loss: 3.236
4 - val_accuracy: 0.1016 - val_loss: 12.5817
Epoch 38/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3083 - loss: 3.224
5 - val_accuracy: 0.1030 - val_loss: 12.7685
```

```
Epoch 39/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 33s 16ms/step - accuracy: 0.3102 - loss: 3.219
6 - val_accuracy: 0.1011 - val_loss: 12.8408
Epoch 40/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3063 - loss: 3.237
3 - val_accuracy: 0.1024 - val_loss: 12.8162
Epoch 41/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3099 - loss: 3.230
7 - val_accuracy: 0.0993 - val_loss: 12.9878
Epoch 42/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3113 - loss: 3.232
3 - val_accuracy: 0.0981 - val_loss: 13.0761
Epoch 43/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3060 - loss: 3.229
6 - val_accuracy: 0.1007 - val_loss: 13.1570
Epoch 44/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3155 - loss: 3.185
0 - val_accuracy: 0.1003 - val_loss: 13.3580
Epoch 45/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3074 - loss: 3.237
4 - val_accuracy: 0.0956 - val_loss: 13.2644
Epoch 46/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3046 - loss: 3.260
7 - val_accuracy: 0.0976 - val_loss: 13.5457
Epoch 47/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3134 - loss: 3.211
9 - val_accuracy: 0.1033 - val_loss: 13.4333
Epoch 48/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3050 - loss: 3.286
6 - val_accuracy: 0.0998 - val_loss: 13.5238
Epoch 49/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3082 - loss: 3.254
9 - val_accuracy: 0.0971 - val_loss: 13.6948
Epoch 50/50
2019/2019 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3080 - loss: 3.253
7 - val_accuracy: 0.0985 - val_loss: 13.7639
```

Save Models (Weights and biases)

In [55]:

```
# # Save model architecture as JSON file
# from tensorflow.keras.models import model_from_json

# model_json = model.to_json()
# with open("lstm_model.json", "w") as json_file:
#     json_file.write(model_json)
```

In [56]:

```
# # Load model architecture from JSON file
# from tensorflow.keras.models import model_from_json

# with open("lstm_model.json", "r") as json_file:
#     loaded_model_json = json_file.read()

# # Create model from loaded architecture
# loaded_model = model_from_json(loaded_model_json)

# print("Model architecture loaded successfully from JSON file.")
```

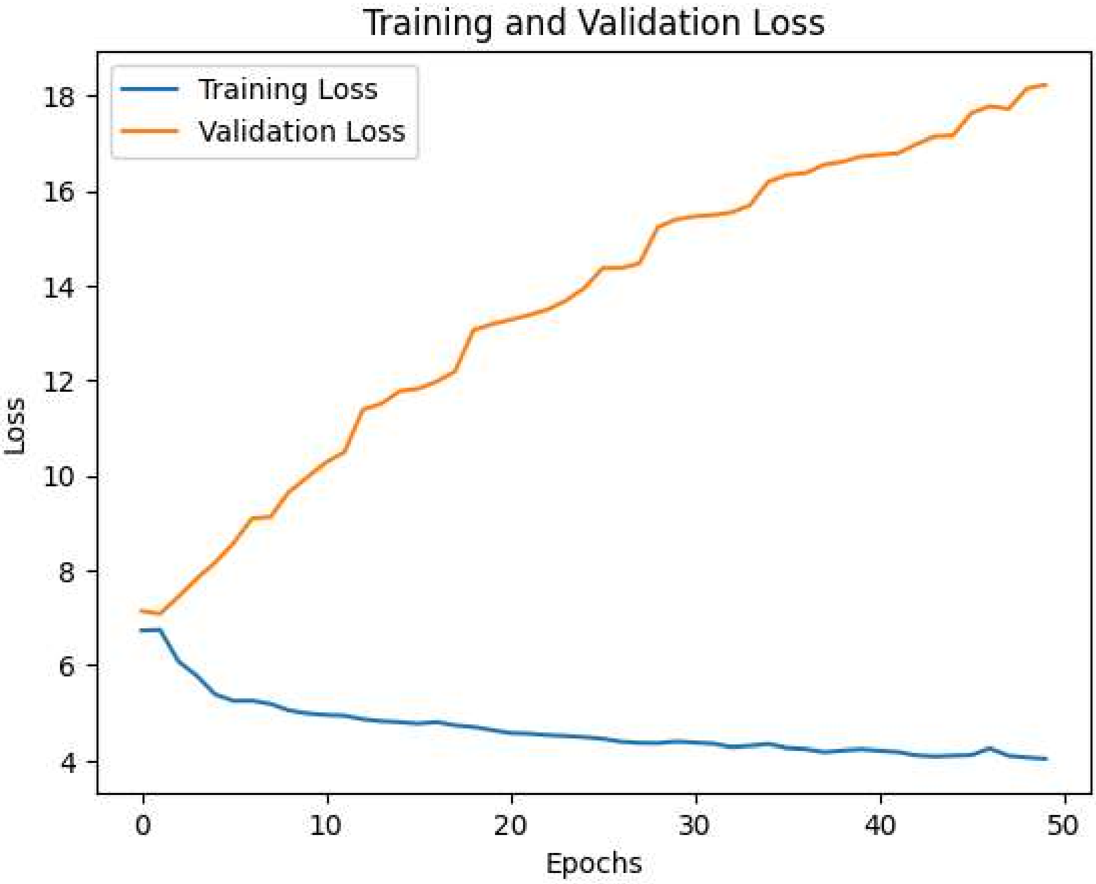
In [9]:

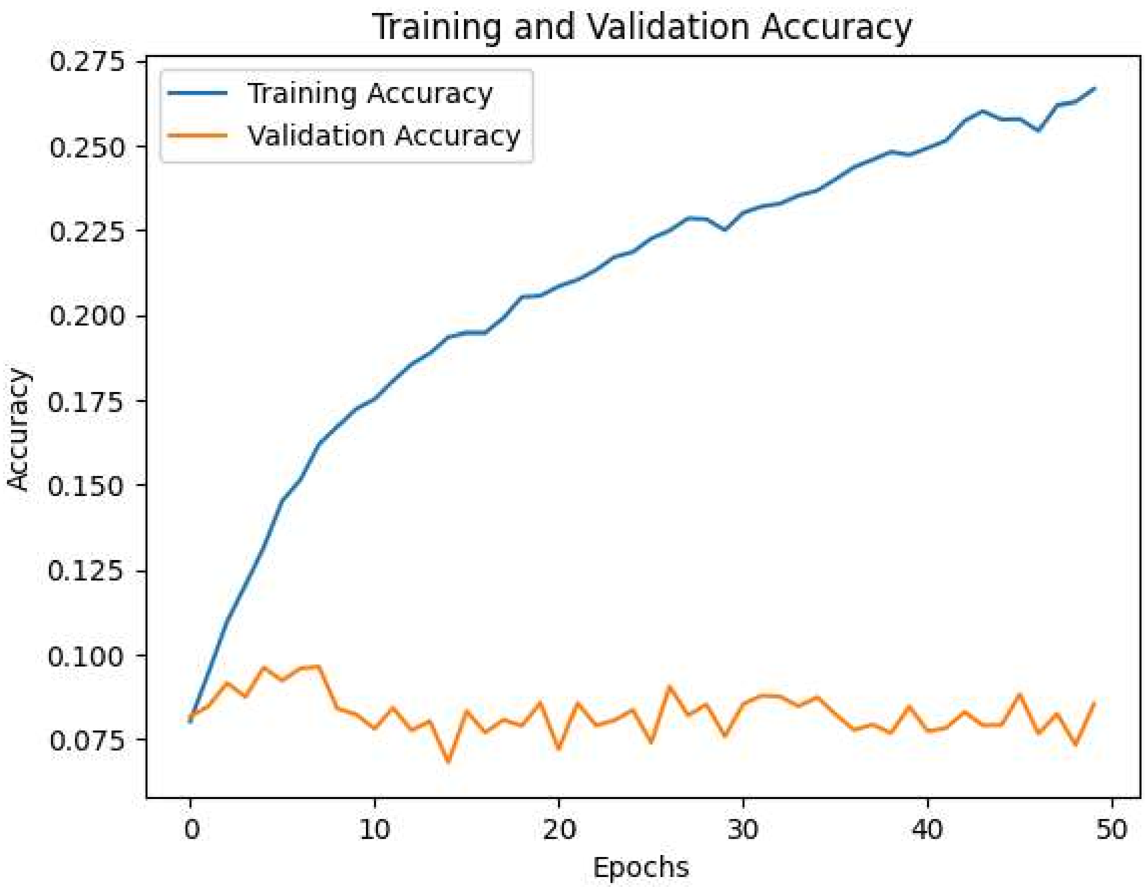
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

In [10]:

```
# Plot Loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot Accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





Inferences

In [31]:
```
def predict_top_five_words(model, tokenizer, seed_text):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, paddir
    predicted = model.predict(token_list, verbose=0)
    top_five_indexes = np.argsort(predicted[0])[::-1][:5]
    top_five_words = []
    for index in top_five_indexes:
        for word, idx in tokenizer.word_index.items():
            if idx == index:
                top_five_words.append(word)
                break
    return top_five_words
```

In [68]:

```
from IPython.display import HTML

def predict_top_five_words(model, tokenizer, seed_text):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, paddir
    predicted = model.predict(token_list, verbose=0)
    top_five_indexes = np.argsort(predicted[0])[::-1][:5]
    top_five_words = []
    for index in top_five_indexes:
        for word, idx in tokenizer.word_index.items():
            if idx == index:
                top_five_words.append(word)
                break
    return top_five_words

def predict_and_display_top_five_words(seed_text, model, tokenizer):
    top_five_words = predict_top_five_words(model, tokenizer, seed_text)
    heading_app = f"<h1>Sentence AutoCompletion App With Five Outputs</h1>"
    output_text = f"<ul>{''.join([f'<li>{seed_text} {word}</li>' for word in t
    javascript_code = f"""
    <script>
        var newWindow = window.open("", "_blank");
        newWindow.document.write('<html><head><title>Top Five Words</title></h
    </script>
    """
    return HTML(javascript_code)
```

Out[68]:

In [69]:

```
# Test the function
seed_text = "She is my"
predict_and_display_top_five_words(seed_text, loaded_model, tokenizer)
```

Out[69]:

In [49]:

```
# Test 2:
# Test the function
seed_text = "I have"
predict_and_display_top_five_words(seed_text, loaded_model, tokenizer)
```

Out[49]:

In [70]:

```
# Test 3:
# Test the function
seed_text = "We love"
predict_and_display_top_five_words(seed_text, loaded_model, tokenizer)
```

Out[70]:

In [52]:

```
# Test 3:
seed_text = "How are"
predict_and_display_top_five_words(seed_text, loaded_model, tokenizer)
```

Out[52]: