In [1]:
```python
#Libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, LSTM, Dropout, GRU, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.sequence import pad_sequences
import nltk
import re
from nltk.stem import PorterStemmer
import pandas as pd
import numpy as np
from keras.preprocessing.text import Tokenizer
!pip install textblob
from textblob import TextBlob
from tensorflow.keras.optimizers import Adam
```

```
Requirement already satisfied: textblob in c:\users\user\anaconda3\envs\newtf\lib\site-packages (0.18.0.post0)
Requirement already satisfied: nltk>=3.8 in c:\users\user\anaconda3\envs\newtf\lib\site-packages (from textblob) (3.9.1)
Requirement already satisfied: click in c:\users\user\anaconda3\envs\newtf\lib\site-packages (from nltk>=3.8->textblob) (8.1.7)
Requirement already satisfied: joblib in c:\users\user\anaconda3\envs\newtf\lib\site-packages (from nltk>=3.8->textblob) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in c:\users\user\anaconda3\envs\newtf\lib\site-packages (from nltk>=3.8->textblob) (2024.11.6)
Requirement already satisfied: tqdm in c:\users\user\anaconda3\envs\newtf\lib\site-packages (from nltk>=3.8->textblob) (4.67.0)
Requirement already satisfied: colorama in c:\users\user\anaconda3\envs\newtf\lib\site-packages (from click->nltk>=3.8->textblob) (0.4.6)
```

In [2]:
```python
#Load the dataset
df = pd.read_csv("next_word_predictor.txt", delimiter='\t')  # Replace '\t' with the correct delimiter.
#first five rows of the data
df.head()
```

Out[2]:

	The sun was shining brightly in the clear blue sky, and a gentle breeze rustled the leaves of the tall trees. People were out enjoying the beautiful weather, some sitting in the park, others taking a leisurely stroll along the riverbank. Children were playing games, and laughter filled the air.
0	As the day turned into evening, the temperatur...
1	In the distance, you could hear the sound of l...
2	As the stars began to twinkle in the night sky...
3	The ancient castle stood on a hill, its toweri...
4	Inside the castle, you could find grand halls ...

In [3]:
```python
#how many columns and rows are in the data
df.shape
```

Out[3]: (1544, 1)

In [4]:
```python
#name of the columns in the dataset
print(df.columns)
```

```
Index(['The sun was shining brightly in the clear blue sky, and a gentle breeze rustled the leaves of the tall trees. People were out enjoying the beautiful weather, some sitting in the park, others taking
a leisurely stroll along the riverbank. Children were playing games, and laughter filled the air.'], dtype='object')
```

In [5]:
```python
#rename the columns
df = df.rename(columns={'The sun was shining brightly in the clear blue sky, and a gentle breeze rustled the leaves of the tall trees. People were out enjoying the beautiful weather, some sitting in the pa
```

In [6]:
```python
#conver all string into a lower case
df['noman'] = df['noman'].str.lower()
df.head()
```

Out[6]:

	noman
0	as the day turned into evening, the temperatur...
1	in the distance, you could hear the sound of l...
2	as the stars began to twinkle in the night sky...
3	the ancient castle stood on a hill, its toweri...
4	inside the castle, you could find grand halls ...

In [7]:
```python
#data preprocessing
def remove(text):
    # Regular expression to match HTML tags, URLs, emojis, and punctuation
     pattern = r"(<[^>]+>)|(\b(?:https?://|www\.)[^\s]+)|([\U0001F600-\U0001F64F\u2700-\u27BF\u2600-\u26FF\u2B50\u2B06\u2194]+)|([.,!?;:(){}[\]\"'`<>])"
       # Replacing matches with an empty string
     cleaned_text = re.sub(pattern, '', text)

     return cleaned_text
df['noman'] = df['noman'].apply(remove)
```

In [8]:
```python
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt_tab')

def remove_stopwords(text):

    stop_words = set(stopwords.words('english'))
    words = nltk.word_tokenize(text)
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)
df['noman'] = df['noman'].apply(remove_stopwords)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

In [9]:
```python
from nltk.tokenize import word_tokenize
```

In [10]:
```python
def stem_text(text):
    # Initialize the PorterStemmer
    stemmer = PorterStemmer()

    # Tokenize the text into words
    words = word_tokenize(text)

    # Stem each word in the tokenized text
    stemmed_words = [stemmer.stem(word) for word in words]

    # Return the stemmed text as a string
    return ' '.join(stemmed_words)
df['noman'] = df['noman'].apply(stem_text)
```

In [11]:
```python
df.head
```

Out[11]:
```
<bound method NDFrame.head of                                                    noman
0     day turn even temperatur start drop sky transf...
1     distanc could hear sound live music come local...
2     star began twinkl night sky crowd grew even la...
3     ancient castl stood hill tower spire reach tow...
4     insid castl could find grand hall adorn magnif...
...                                                 ...
1539  germani spirit tapestri woven thread histori c...
1540  sun-bak sand sahara verdant rainforest congo b...
1541  across vast expans savanna icon silhouett acac...
1542  beyond natur wonder africa pulsat energi peopl...
1543  spirit africa one resili hope ancient pyramid ...

[1544 rows x 1 columns]>
```

In [32]:
```python
token = Tokenizer()
token.fit_on_texts(df['noman'])

#token.word_index
```

In [13]:
```python
input = []
for sentence in df['noman'].apply(lambda x: str(x).split('\n') if x is not None else []):
    for sub_sentence in sentence:
        tokenized = token.texts_to_sequences([sub_sentence])[0]
        for i in range(1, len(tokenized)):
            input.append(tokenized[:i+1])
```

In [33]:
```python
#input
```

In [15]:
```python
#find the maximum len in one line
max_len = max([len(x) for x in input])
max_len
```

Out[15]: 189

In [16]:
```python
#add padding to equal the len of all lines
padded = pad_sequences(input, maxlen = max_len, padding = 'pre')
padded[10]
```

Out[16]:
```
array([   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,  143,  168,  116, 1369,   51,  398,  136,  541,  836,   48,
        117,  399])
```

In [17]:
```python
x = padded[:,:-1]
```

In [18]:
```python
# Check the maximum value in the input data
print(np.max(x))
```

```
3701
```

In [19]:
```python
x
```

Out[19]:
```
array([[   0,    0,    0, ...,    0,    0,  143],
       [   0,    0,    0, ...,    0,  143,  168],
       [   0,    0,    0, ...,  143,  168,  116],
       ...,
       [   0,    0,    0, ...,  160,  206,   81],
       [   0,    0,    0, ...,  206,   81, 1972],
       [   0,    0,    0, ...,   81, 1972,  260]])
```

In [20]:
```python
x.shape
```

Out[20]: (15056, 188)

In [21]:
```python
y = padded[:,-1]
```

In [22]:
```python
y.shape
```

Out[22]: (15056,)

In [23]:
```python
print(y.max())
```

```
3701
```

In [24]:
```python
print(y.shape)
print(np.unique(y))  # Check the unique values in y
```

```
(15056,)
[   1    2    3 ... 3699 3700 3701]
```

In [25]:
```python
import numpy as np
from tensorflow.keras.utils import to_categorical
y = to_categorical(y, num_classes=3702)
```

In [26]:
```python
y.shape
```

Out[26]: (15056, 3702)

In [27]:
```python
#apply LSTM model
model = Sequential()
model.add(Embedding(input_dim=3702, output_dim=100, input_length=188))
model.add(LSTM(100))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(3702, activation='softmax'))
```

In [28]:
```python
#compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x, y, epochs=70, batch_size=32, verbose=1)
```

```
Epoch 1/70
471/471 [==============================] - 91s 187ms/step - loss: 7.6674 - accuracy: 0.0078
Epoch 2/70
471/471 [==============================] - 95s 202ms/step - loss: 7.1668 - accuracy: 0.0136
Epoch 3/70
471/471 [==============================] - 113s 240ms/step - loss: 6.9250 - accuracy: 0.0162
Epoch 4/70
471/471 [==============================] - 104s 221ms/step - loss: 6.7300 - accuracy: 0.0198
Epoch 5/70
471/471 [==============================] - 96s 205ms/step - loss: 6.5391 - accuracy: 0.0247
Epoch 6/70
471/471 [==============================] - 92s 194ms/step - loss: 6.3379 - accuracy: 0.0299
Epoch 7/70
471/471 [==============================] - 91s 192ms/step - loss: 6.1336 - accuracy: 0.0361
Epoch 8/70
471/471 [==============================] - 89s 189ms/step - loss: 5.9101 - accuracy: 0.0468
Epoch 9/70
471/471 [==============================] - 87s 185ms/step - loss: 5.6575 - accuracy: 0.0592
Epoch 10/70
471/471 [==============================] - 86s 183ms/step - loss: 5.3766 - accuracy: 0.0751
Epoch 11/70
471/471 [==============================] - 119s 252ms/step - loss: 5.0849 - accuracy: 0.0985
Epoch 12/70
471/471 [==============================] - 101s 215ms/step - loss: 4.7629 - accuracy: 0.1232
Epoch 13/70
471/471 [==============================] - 85s 181ms/step - loss: 4.4687 - accuracy: 0.1522
Epoch 14/70
471/471 [==============================] - 86s 182ms/step - loss: 4.1577 - accuracy: 0.1872
Epoch 15/70
471/471 [==============================] - 88s 188ms/step - loss: 3.8550 - accuracy: 0.2216
Epoch 16/70
471/471 [==============================] - 99s 210ms/step - loss: 3.5630 - accuracy: 0.2638
Epoch 17/70
471/471 [==============================] - 107s 228ms/step - loss: 3.2769 - accuracy: 0.3080
Epoch 18/70
471/471 [==============================] - 102s 218ms/step - loss: 3.0254 - accuracy: 0.3519
Epoch 19/70
471/471 [==============================] - 86s 183ms/step - loss: 2.7939 - accuracy: 0.3887
Epoch 20/70
471/471 [==============================] - 85s 181ms/step - loss: 2.5786 - accuracy: 0.4305
Epoch 21/70
471/471 [==============================] - 87s 184ms/step - loss: 2.3975 - accuracy: 0.4649
Epoch 22/70
471/471 [==============================] - 87s 185ms/step - loss: 2.2259 - accuracy: 0.4993
Epoch 23/70
471/471 [==============================] - 86s 182ms/step - loss: 2.0792 - accuracy: 0.5299
Epoch 24/70
471/471 [==============================] - 86s 183ms/step - loss: 1.9420 - accuracy: 0.5527
Epoch 25/70
471/471 [==============================] - 87s 185ms/step - loss: 1.8219 - accuracy: 0.5793
Epoch 26/70
471/471 [==============================] - 86s 183ms/step - loss: 1.7255 - accuracy: 0.6007
Epoch 27/70
471/471 [==============================] - 87s 184ms/step - loss: 1.6303 - accuracy: 0.6175
Epoch 28/70
471/471 [==============================] - 131s 278ms/step - loss: 1.5468 - accuracy: 0.6338
Epoch 29/70
471/471 [==============================] - 83s 176ms/step - loss: 1.4724 - accuracy: 0.6554
Epoch 30/70
471/471 [==============================] - 86s 182ms/step - loss: 1.4007 - accuracy: 0.6666
Epoch 31/70
471/471 [==============================] - 83s 175ms/step - loss: 1.3459 - accuracy: 0.6763
Epoch 32/70
471/471 [==============================] - 83s 176ms/step - loss: 1.3029 - accuracy: 0.6912
Epoch 33/70
471/471 [==============================] - 83s 176ms/step - loss: 1.2356 - accuracy: 0.7061
Epoch 34/70
471/471 [==============================] - 83s 176ms/step - loss: 1.2114 - accuracy: 0.7075
Epoch 35/70
471/471 [==============================] - 83s 176ms/step - loss: 1.1588 - accuracy: 0.7210
Epoch 36/70
471/471 [==============================] - 83s 176ms/step - loss: 1.1064 - accuracy: 0.7293
Epoch 37/70
471/471 [==============================] - 83s 176ms/step - loss: 1.0848 - accuracy: 0.7354
Epoch 38/70
471/471 [==============================] - 83s 176ms/step - loss: 1.0509 - accuracy: 0.7424
Epoch 39/70
471/471 [==============================] - 82s 174ms/step - loss: 1.0249 - accuracy: 0.7471
Epoch 40/70
471/471 [==============================] - 88s 187ms/step - loss: 0.9660 - accuracy: 0.7581
Epoch 42/70
471/471 [==============================] - 84s 179ms/step - loss: 0.9355 - accuracy: 0.7692
Epoch 43/70
471/471 [==============================] - 86s 182ms/step - loss: 0.9178 - accuracy: 0.7718
Epoch 44/70
471/471 [==============================] - 83s 177ms/step - loss: 0.8930 - accuracy: 0.7766
Epoch 45/70
471/471 [==============================] - 83s 175ms/step - loss: 0.8771 - accuracy: 0.7804
Epoch 46/70
471/471 [==============================] - 82s 173ms/step - loss: 0.8768 - accuracy: 0.7768
Epoch 47/70
471/471 [==============================] - 82s 174ms/step - loss: 0.8466 - accuracy: 0.7863
Epoch 48/70
471/471 [==============================] - 84s 177ms/step - loss: 0.8319 - accuracy: 0.7913
Epoch 49/70
471/471 [==============================] - 102s 217ms/step - loss: 0.7972 - accuracy: 0.8003
Epoch 51/70
471/471 [==============================] - 85s 180ms/step - loss: 0.7787 - accuracy: 0.8042
Epoch 53/70
471/471 [==============================] - 86s 183ms/step - loss: 0.7572 - accuracy: 0.8058
Epoch 54/70
471/471 [==============================] - 82s 175ms/step - loss: 0.7346 - accuracy: 0.8145
Epoch 55/70
471/471 [==============================] - 82s 175ms/step - loss: 0.7366 - accuracy: 0.8152
Epoch 56/70
471/471 [==============================] - 83s 177ms/step - loss: 0.7265 - accuracy: 0.8144
Epoch 57/70
471/471 [==============================] - 83s 176ms/step - loss: 0.7191 - accuracy: 0.8172
Epoch 58/70
471/471 [==============================] - 82s 174ms/step - loss: 0.7183 - accuracy: 0.8162
Epoch 59/70
471/471 [==============================] - 83s 175ms/step - loss: 0.7131 - accuracy: 0.8169
Epoch 60/70
471/471 [==============================] - 82s 174ms/step - loss: 0.6925 - accuracy: 0.8257
Epoch 61/70
471/471 [==============================] - 98s 208ms/step - loss: 0.6791 - accuracy: 0.8257
Epoch 63/70
471/471 [==============================] - 97s 207ms/step - loss: 0.6758 - accuracy: 0.8253
Epoch 64/70
471/471 [==============================] - 92s 194ms/step - loss: 0.6664 - accuracy: 0.8296
Epoch 65/70
471/471 [==============================] - 89s 188ms/step - loss: 0.6703 - accuracy: 0.8288
Epoch 66/70
471/471 [==============================] - 83s 175ms/step - loss: 0.6489 - accuracy: 0.8334
Epoch 67/70
471/471 [==============================] - 82s 174ms/step - loss: 0.6464 - accuracy: 0.8342
Epoch 68/70
471/471 [==============================] - 82s 175ms/step - loss: 0.6471 - accuracy: 0.8331
Epoch 69/70
471/471 [==============================] - 82s 175ms/step - loss: 0.6318 - accuracy: 0.8392
Epoch 70/70
471/471 [==============================] - 82s 174ms/step - loss: 0.6206 - accuracy: 0.8406
```

Out[28]: <keras.callbacks.History at 0x299262b9730>

In [31]:
```python
#result
text = 'heart bustl citi street aliv'

for i in range(10):
    # Assuming `token` is a properly instantiated tokenizer
    tokeniz = token.texts_to_sequences([text])[0]  # Get the first sequence

    # Padding the sequence (ensure the padding length is appropriate)
    padd = pad_sequences([tokeniz], maxlen=188, padding='pre')

    # Predict using the model
    pos = np.argmax(model.predict(padd))

    # Find the predicted word using the token's word_index
    for word, index in token.word_index.items():
        if index == pos:
            # Append the predicted word to the text and print
            text = text + ' ' + word  # Update the text with the predicted word
            print(text)  # Print the updated text
            break  # Exit the loop after printing the predicted word
```

```
1/1 [==============================] - 0s 60ms/step
heart bustl citi street aliv sound
1/1 [==============================] - 0s 64ms/step
heart bustl citi street aliv sound traffic
1/1 [==============================] - 0s 63ms/step
heart bustl citi street aliv sound traffic chatter
1/1 [==============================] - 0s 77ms/step
heart bustl citi street aliv sound traffic chatter peopl
1/1 [==============================] - 0s 63ms/step
heart bustl citi street aliv sound traffic chatter peopl go
1/1 [==============================] - 0s 69ms/step
heart bustl citi street aliv sound traffic chatter peopl go daili
1/1 [==============================] - 0s 60ms/step
heart bustl citi street aliv sound traffic chatter peopl go daili live
1/1 [==============================] - 0s 63ms/step
heart bustl citi street aliv sound traffic chatter peopl go daili live skyscrap
1/1 [==============================] - 0s 55ms/step
heart bustl citi street aliv sound traffic chatter peopl go daili live skyscrap reach
1/1 [==============================] - 0s 78ms/step
heart bustl citi street aliv sound traffic chatter peopl go daili live skyscrap reach toward
```