Task-4 Natural Language Processing(NLP) For Text Generation

Importing Libraries

In [49]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding, Dropout
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [50]:

```
text = """Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do:
once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it,
'and what is the use of a book,' thought Alice 'without pictures or conversations?' So she was considering in her own mi
(as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-cha
would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close
"""
```

Tokenization of Text

In [51]:

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
total_words = len(tokenizer.word_index) + 1
```

In [52]:

```
input_sequences = []
for line in text.split('.'):
    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)
```

In [53]:

```
max_sequence_len = max([len(seq) for seq in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
```

In [54]:

```
X, y = input_sequences[:, :-1], input_sequences[:, -1]
```

In [55]:

```
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
```

Model Training

In [56]:

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(LSTM(150))
model.add(Dropout(0.2))
model.add(Dense(total_words, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [57]:

```
epochs = 50
history = model.fit(X, y, epochs=epochs, verbose=1)
```

```
Epoch 1/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 131ms/step - accuracy: 0.0150 - loss: 4.3694
Epoch 2/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 138ms/step - accuracy: 0.0649 - loss: 4.3575
Epoch 3/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 141ms/step - accuracy: 0.0889 - loss: 4.3422
Epoch 4/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 141ms/step - accuracy: 0.0760 - loss: 4.3132
Epoch 5/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 143ms/step - accuracy: 0.0714 - loss: 4.2163
Epoch 6/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 132ms/step - accuracy: 0.0708 - loss: 4.2431
Epoch 7/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.0562 - loss: 4.2108
Epoch 8/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 135ms/step - accuracy: 0.0735 - loss: 4.1760
Epoch 9/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.0579 - loss: 4.1503
Epoch 10/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 143ms/step - accuracy: 0.0662 - loss: 4.0711
Epoch 11/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 138ms/step - accuracy: 0.0994 - loss: 4.0676
Epoch 12/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 133ms/step - accuracy: 0.1101 - loss: 3.9604
Epoch 13/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.1084 - loss: 3.9374
Epoch 14/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 139ms/step - accuracy: 0.0751 - loss: 3.9348
Epoch 15/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 138ms/step - accuracy: 0.0774 - loss: 3.7881
Epoch 16/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 132ms/step - accuracy: 0.0740 - loss: 3.6425
Epoch 17/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 133ms/step - accuracy: 0.0879 - loss: 3.4994
Epoch 18/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 138ms/step - accuracy: 0.1144 - loss: 3.3571
Epoch 19/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.0778 - loss: 3.3392
Epoch 20/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 135ms/step - accuracy: 0.1150 - loss: 3.1442
Epoch 21/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.1046 - loss: 3.1906
Epoch 22/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 145ms/step - accuracy: 0.1533 - loss: 2.9589
Epoch 23/50
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.1060 - loss: 2.9718
Epoch 24/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 138ms/step - accuracy: 0.1997 - loss: 2.7889
Epoch 25/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 132ms/step - accuracy: 0.1854 - loss: 2.7079
Epoch 26/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 137ms/step - accuracy: 0.1827 - loss: 2.6522
Epoch 27/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 139ms/step - accuracy: 0.2106 - loss: 2.5626
Epoch 28/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.2096 - loss: 2.4650
Epoch 29/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 130ms/step - accuracy: 0.2826 - loss: 2.4433
Epoch 30/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 135ms/step - accuracy: 0.2667 - loss: 2.4121
Epoch 31/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 139ms/step - accuracy: 0.3280 - loss: 2.2216
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 134ms/step - accuracy: 0.2747 - loss: 2.2665
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 138ms/step - accuracy: 0.2901 - loss: 2.1971
Epoch 34/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.3124 - loss: 2.2002
Epoch 35/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.2787 - loss: 2.1501
Epoch 36/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 130ms/step - accuracy: 0.3427 - loss: 2.0680
Epoch 37/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 135ms/step - accuracy: 0.3724 - loss: 2.0029
Epoch 38/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 141ms/step - accuracy: 0.3323 - loss: 1.9873
Epoch 39/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 135ms/step - accuracy: 0.3524 - loss: 1.9660
Epoch 40/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 133ms/step - accuracy: 0.4323 - loss: 1.9126
Epoch 41/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.4706 - loss: 1.8486
Epoch 42/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 132ms/step - accuracy: 0.4231 - loss: 1.7748
Epoch 43/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 133ms/step - accuracy: 0.3552 - loss: 1.8910
Epoch 44/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 132ms/step - accuracy: 0.4598 - loss: 1.7789
Epoch 45/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 142ms/step - accuracy: 0.3950 - loss: 1.7776
```

```
Epoch 46/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 131ms/step - accuracy: 0.3980 - loss: 1.7299
Epoch 47/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 134ms/step - accuracy: 0.4201 - loss: 1.6700
Epoch 48/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 132ms/step - accuracy: 0.4689 - loss: 1.6764
Epoch 49/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 141ms/step - accuracy: 0.5042 - loss: 1.6455
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 135ms/step - accuracy: 0.4744 - loss: 1.5950
```

Text Generation Function

In [60]:

```python
def generate_text(seed_text, next_words, max_sequence_len):
    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')
        predicted = model.predict(token_list, verbose=0)
        predicted_word_index = np.argmax(predicted, axis=1)[0]

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

Example Usage

In [62]:

```python
seed_text = "There was a boy named kevin"
generated_text = generate_text(seed_text, 10, max_sequence_len)
print(f"Generated text: {generated_text}")
```

```
Generated text: There was a boy named kevin was to get tired tired of sitting by her sister
```

In [63]:

```python
model.save('text_generation_lstm_model.h5')
import joblib
joblib.dump(tokenizer, 'tokenizer.pkl')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This fil
e format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
```

```
Out[63]: ['tokenizer.pkl']
```

```
In [ ]:
```