In [1]:

```python
# Submission Details- Assignment 4 NLP
# Submitted by: Abhinav Pathak-2403res02,Kundan Kumar Sharma-2403res42, Amisha-2403res89,
# Namrata Chakraborty-2403res49
```

In [2]:

```python
# Imports

import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical

# Sample training data ()

train_data = '''The Chief Minister of Madhya Pradesh, Shivraj Singh Chouhan named Indore as th
It is not the first time that the city has achieved this feat.
In fact, Indore has been named the cleanest city in India for the sixth time in a row.
The results of the Swachh Surbekshan Awards 2022 were declared on Saturday.
The annual cleanliness survey of the Union government declared Indore as the cleanest city, wi
securing the second and the third positions respectively.
Chouhan expressed in a press release that it is actually a matter of great pride that Indore h
title of the cleanest city in the country for the sixth time in a row.
The man also appreciated the waste collection and disposal mechanism of Indore.
He also praised the sanitation workers (safai mitras), elected representatives, officials,
and citizens for their exemplary performance for their cleanliness performance. '''

train_data=train_data.lower()

# Unique Characters: We create a sorted list of unique characters in the training data.
# Character Mapping: Two dictionaries are created:
# Integer Encoding: Maps each character to a unique integer.
# char_to_int: Maps each character to a unique integer.
# int_to_char: Maps each integer back to its corresponding character.

# Create a sorted list of unique characters
chars = sorted(set(train_data))
# Create a mapping from characters to integers
char_to_int = {c: i for i, c in enumerate(chars)}
int_to_char = {i: c for i, c in enumerate(chars)}

print('Char_to_int','\n',char_to_int)
print('-----')
print('Int_to_char','\n',int_to_char)
```

```
Char_to_int
 {'\n': 0, ' ': 1, '(': 2, ')': 3, ',': 4, '.': 5, '0': 6, '2': 7, 'a': 8, 'b': 9, 'c': 10, 'd':
11, 'e': 12, 'f': 13, 'g': 14, 'h': 15, 'i': 16, 'j': 17, 'k': 18, 'l': 19, 'm': 20, 'n': 21,
'o': 22, 'p': 23, 'r': 24, 's': 25, 't': 26, 'u': 27, 'v': 28, 'w': 29, 'x': 30, 'y': 31, 'z': 3
2}
-----
Int_to_char
 {0: '\n', 1: ' ', 2: '(', 3: ')', 4: ',', 5: '.', 6: '0', 7: '2', 8: 'a', 9: 'b', 10: 'c', 11:
'd', 12: 'e', 13: 'f', 14: 'g', 15: 'h', 16: 'i', 17: 'j', 18: 'k', 19: 'l', 20: 'm', 21: 'n', 2
2: 'o', 23: 'p', 24: 'r', 25: 's', 26: 't', 27: 'u', 28: 'v', 29: 'w', 30: 'x', 31: 'y', 32: 'z'}
```

In [3]:

```
# Prepare the input and output sequences
# Input and Output Arrays: X will hold sequences of characters
# Y will hold the corresponding next character for each sequence.

X = []
Y = []
seq_length = 5  # Length of the input sequences

for i in range(len(train_data) - seq_length):
    seq_in = train_data[i:i + seq_length]
    seq_out = train_data[i + seq_length]
    X.append([char_to_int[char] for char in seq_in])
    Y.append(char_to_int[seq_out])

print('Training set independent variable','\n',X)
print('-----')
print('Training set target variable','\n',Y)
```

```
0, 25, 26], [21, 16, 25, 26, 1], [16, 25, 26, 1, 22], [25, 26, 1, 22, 13], [26, 1, 22, 13, 1],
[1, 22, 13, 1, 16], [22, 13, 1, 16, 21], [13, 1, 16, 21, 11], [1, 16, 21, 11, 22], [16, 21, 1
1, 22, 24], [21, 11, 22, 24, 12], [11, 22, 24, 12, 5], [22, 24, 12, 5, 0], [24, 12, 5, 0, 15],
[12, 5, 0, 15, 12], [5, 0, 15, 12, 1], [0, 15, 12, 1, 8], [15, 12, 1, 8, 19], [12, 1, 8, 19, 2
5], [1, 8, 19, 25, 22], [8, 19, 25, 22, 1], [19, 25, 22, 1, 23], [25, 22, 1, 23, 24], [22, 1,
23, 24, 8], [1, 23, 24, 8, 16], [23, 24, 8, 16, 25], [24, 8, 16, 25, 12], [8, 16, 25, 12, 11],
[16, 25, 12, 11, 1], [25, 12, 11, 1, 26], [12, 11, 1, 26, 15], [11, 1, 26, 15, 12], [1, 26, 1
5, 12, 1], [26, 15, 12, 1, 25], [15, 12, 1, 25, 8], [12, 1, 25, 8, 21], [1, 25, 8, 21, 16], [2
5, 8, 21, 16, 26], [8, 21, 16, 26, 8], [21, 16, 26, 8, 26], [16, 26, 8, 26, 16], [26, 8, 26, 1
6, 22], [8, 26, 16, 22, 21], [26, 16, 22, 21, 1], [16, 22, 21, 1, 29], [22, 21, 1, 29, 22], [2
1, 1, 29, 22, 24], [1, 29, 22, 24, 18], [29, 22, 24, 18, 12], [22, 24, 18, 12, 24], [24, 18, 1
2, 24, 25], [18, 12, 24, 25, 1], [12, 24, 25, 1, 2], [24, 25, 1, 2, 25], [25, 1, 2, 25, 8],
[1, 2, 25, 8, 13], [2, 25, 8, 13, 8], [25, 8, 13, 8, 16], [8, 13, 8, 16, 1], [13, 8, 16, 1, 2
0], [8, 16, 1, 20, 16], [16, 1, 20, 16, 26], [1, 20, 16, 26, 24], [20, 16, 26, 24, 8], [16, 2
6, 24, 8, 25], [26, 24, 8, 25, 3], [24, 8, 25, 3, 4], [8, 25, 3, 4, 1], [25, 3, 4, 1, 12], [3,
4, 1, 12, 19], [4, 1, 12, 19, 12], [1, 12, 19, 12, 10], [12, 19, 12, 10, 26], [19, 12, 10, 26,
12], [12, 10, 26, 12, 11], [10, 26, 12, 11, 1], [26, 12, 11, 1, 24], [12, 11, 1, 24, 12], [11,
1, 24, 12, 23], [1, 24, 12, 23, 24], [24, 12, 23, 24, 12], [12, 23, 24, 12, 25], [23, 24, 12,
25, 12], [24, 12, 25, 12, 21], [12, 25, 12, 21, 26], [25, 12, 21, 26, 8], [12, 21, 26, 8, 26],
[21, 26, 8, 26, 16], [26, 8, 26, 16, 28], [8, 26, 16, 28, 12], [26, 16, 28, 12, 25], [16, 28,
12, 25, 4], [28, 12, 25, 4, 1], [12, 25, 4, 1, 22], [25, 4, 1, 22, 13], [4, 1, 22, 13, 13]
```

In [4]:

```
# Convert to numpy arrays
X = np.array(X)
Y = np.array(Y)
```

In [5]:

```
# One-hot encode the output variable
Y = to_categorical(Y, num_classes=len(chars))

# Define the model
model = Sequential()
model.add(Embedding(input_dim=len(chars), output_dim=10, input_length=seq_length))
model.add(LSTM(50))
model.add(Dense(len(chars), activation='softmax'))

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

# Fit the model
model.fit(X, Y, epochs=500, verbose=2)
```

```
Epoch 492/500
30/30 - 0s - 5ms/step - accuracy: 0.9345 - loss: 0.1227
Epoch 493/500
30/30 - 0s - 10ms/step - accuracy: 0.9250 - loss: 0.1230
Epoch 494/500
30/30 - 0s - 9ms/step - accuracy: 0.9335 - loss: 0.1219
Epoch 495/500
30/30 - 0s - 10ms/step - accuracy: 0.9356 - loss: 0.1222
Epoch 496/500
30/30 - 0s - 10ms/step - accuracy: 0.9324 - loss: 0.1221
Epoch 497/500
30/30 - 0s - 10ms/step - accuracy: 0.9271 - loss: 0.1224
Epoch 498/500
30/30 - 0s - 5ms/step - accuracy: 0.9314 - loss: 0.1206
Epoch 499/500
30/30 - 0s - 5ms/step - accuracy: 0.9335 - loss: 0.1213
Epoch 500/500
30/30 - 0s - 5ms/step - accuracy: 0.9377 - loss: 0.1223
```

Out[5]:

```
<keras.src.callbacks.history.History at 0x7a6441ffbca0>
```

In [6]:

```
# Function to generate text

# Function Definition: Takes the trained model, a starting string, and the desired length of g

# Input Preparation: Converts the starting string into integer format and pads it

# Text Generation Loop:

# Predicts the next character using the model.

# Appends the predicted character to the output string.

# Updates the input to include the new character for the next prediction.


def generate_text(model, start_string, generation_length=50):
    input_eval = [char_to_int[s] for s in start_string]
    input_eval = pad_sequences([input_eval], maxlen=seq_length, truncating='pre')

    # Generate text
    predicted_text = start_string

    for _ in range(generation_length):
        predictions = model.predict(input_eval, verbose=0)
        next_char_index = np.argmax(predictions[-1])  # Get the index of the highest probabili
        next_char = int_to_char[next_char_index]

        predicted_text += next_char

        # Update the input for the next prediction
        input_eval = np.append(input_eval[:, 1:], [[next_char_index]], axis=1)

    return predicted_text

# Test the model with random input text
start_string = 'indore'
generated_text = generate_text(model, start_string, generation_length=50)
print("Generated text:", generated_text)
```

```
Generated text: indore has managed to secure the cleanest city in india.
```

In [7]:

```
# The character level LSTM cell model is build with the training dataset.

# Each character is represented by a unique integer intially.

# For sequence of charaacters, the training data -'X' and testing data-'Y' are created.

# It is converted to a sequential model with an embedding layer, LSTM layer, and a dense layer

# Model is trained with 500 epochs and verbose=2 and achieved a loss of 0.1223
# and accuracy of 93.77 percent.

# The model is tested with a random input text and
# the generated text is printed upto next 50 characters.
```