Name: **B. SAI CHARAN**

Roll No: **2203A51L72**

Batch No: **21CSBTB12**

ASSIGNMENT – 4

Question:

Implement Recurrent Neural network.

Answer:

- Recurrent Neural Network (RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step.
- In traditional neural networks, all the inputs and outputs are independent of each other.
- Implementing a Recurrent Neural Network (RNN) for Natural Language Processing (NLP) involves several steps. Here's a detailed guide to build a simple RNN model using Python and TensorFlow/Keras for text classification.

RNN Code Implementation:

Imported libraries:

- Imported some necessary libraries such as numpy, tensorflow for numerical calculation an model building.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
```

Input Generation:

Generated some example data using text.

```
text = "This is GeeksforGeeks a software training institute"
chars = sorted(list(set(text)))
char_to_index = {char: i for i, char in enumerate(chars)}
index_to_char = {i: char for i, char in enumerate(chars)}
```

Created input sequences and corresponding labels for further implementation.

```
seq_length = 3
sequences = []
labels = []

for i in range(len(text) - seq_length):
    seq = text[i:i+seq_length]
    label = text[i+seq_length]
    sequences.append([char_to_index[char] for char in seq])
    labels.append(char_to_index[label])
```

Converted sequences and labels into numpy arrays and used one-hot encoding to convert text into vector.

```
X = np.array(sequences)
y = np.array(labels)

X_one_hot = tf.one_hot(X, len(chars))
y_one_hot = tf.one_hot(y, len(chars))
```

Model Building:

Build RNN Model using 'relu' and 'softmax' activation function.

```
model = Sequential()
model.add(SimpleRNN(50, input_shape=(seq_length, len(chars)), activation='relu'))
model.add(Dense(len(chars), activation='softmax'))
```

Model Compilation:

The model.compile line builds the neural network for training by specifying the optimizer (Adam), the loss function (categorical crossentropy), and the training metric (accuracy).

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

Model Training:

Using the input sequences (X_one_hot) and corresponding labels (y_one_hot) for 100 epochs, the model is trained using the model.fit line, which optimises the model parameters to minimise the categorical crossentropy loss.

```
model.fit(X_one_hot, y_one_hot, epochs=100)
```

```
Epoch 1/100
2/2 [==============================] - 2s 54ms/step - loss: 2.8327 - accuracy: 0.0000e+00
Epoch 2/100
2/2 [==============================] - 0s 16ms/step - loss: 2.8121 - accuracy: 0.0000e+00
Epoch 3/100
2/2 [==============================] - 0s 16ms/step - loss: 2.7944 - accuracy: 0.0208
Epoch 4/100
2/2 [==============================] - 0s 16ms/step - loss: 2.7766 - accuracy: 0.0208
Epoch 5/100
2/2 [==============================] - 0s 15ms/step - loss: 2.7596 - accuracy: 0.0625
Epoch 6/100
2/2 [==============================] - 0s 13ms/step - loss: 2.7424 - accuracy: 0.0833
Epoch 7/100
2/2 [==============================] - 0s 13ms/step - loss: 2.7254 - accuracy: 0.1042
Epoch 8/100
2/2 [==============================] - 0s 12ms/step - loss: 2.7092 - accuracy: 0.1042
Epoch 9/100
2/2 [==============================] - 0s 11ms/step - loss: 2.6917 - accuracy: 0.1458
Epoch 10/100
2/2 [==============================] - 0s 12ms/step - loss: 2.6742 - accuracy: 0.1667
Epoch 11/100
2/2 [==============================] - 0s 10ms/step - loss: 2.6555 - accuracy: 0.1667
Epoch 12/100
2/2 [==============================] - 0s 16ms/step - loss: 2.6369 - accuracy: 0.1667
Epoch 13/100
```

Model Prediction:
Generated text using pre-trained model.

```
start_seq = "This is G"
generated_text = start_seq

for i in range(50):
    x = np.array([[char_to_index[char] for char in generated_text[-seq_length:]]])
    x_one_hot = tf.one_hot(x, len(chars))
    prediction = model.predict(x_one_hot)
    next_index = np.argmax(prediction)
    next_char = index_to_char[next_index]
    generated_text += next_char

print("Generated Text:")
print(generated_text)
```

output:

```
1/1 [==============================] - 1s 517ms/step
1/1 [==============================] - 0s 75ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 140ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 32ms/step
```

```
Generated Text:
This is Geeks a software training instituteais is is is is
```