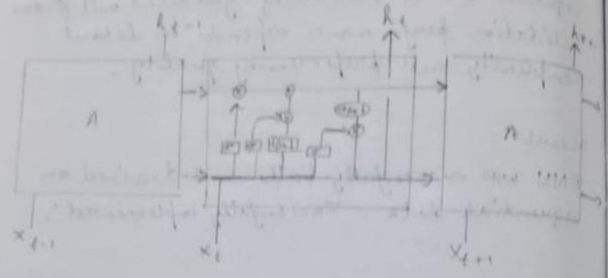
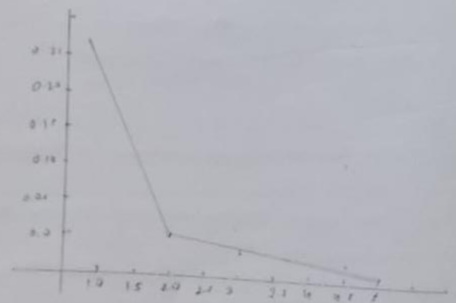
LSTM architecture



Output

Epoch 1: Loss = 0.277
Epoch 2: Loss = 0.023
Epoch 3: Loss = 0.009
Epoch 4: Loss = 0.005
Epoch 5: Loss = 0.002



AIM: To build and implement a L.S.T.M

Pseudocode: Import req libraries
Load and preprocess the sequential dataset
Normalize the data
Create input - output pairs
Reshape x into samples
· Define LSTM model.
Initialize seq. model
Add LSTM layer with req. unit
Add Dense output layer
· Compile the model with optimizer and loss
· Train the model using model fit
· Evaluate model performance on test data
· Predict future or test sample
· Visualize predicted vs actual output.

Observation

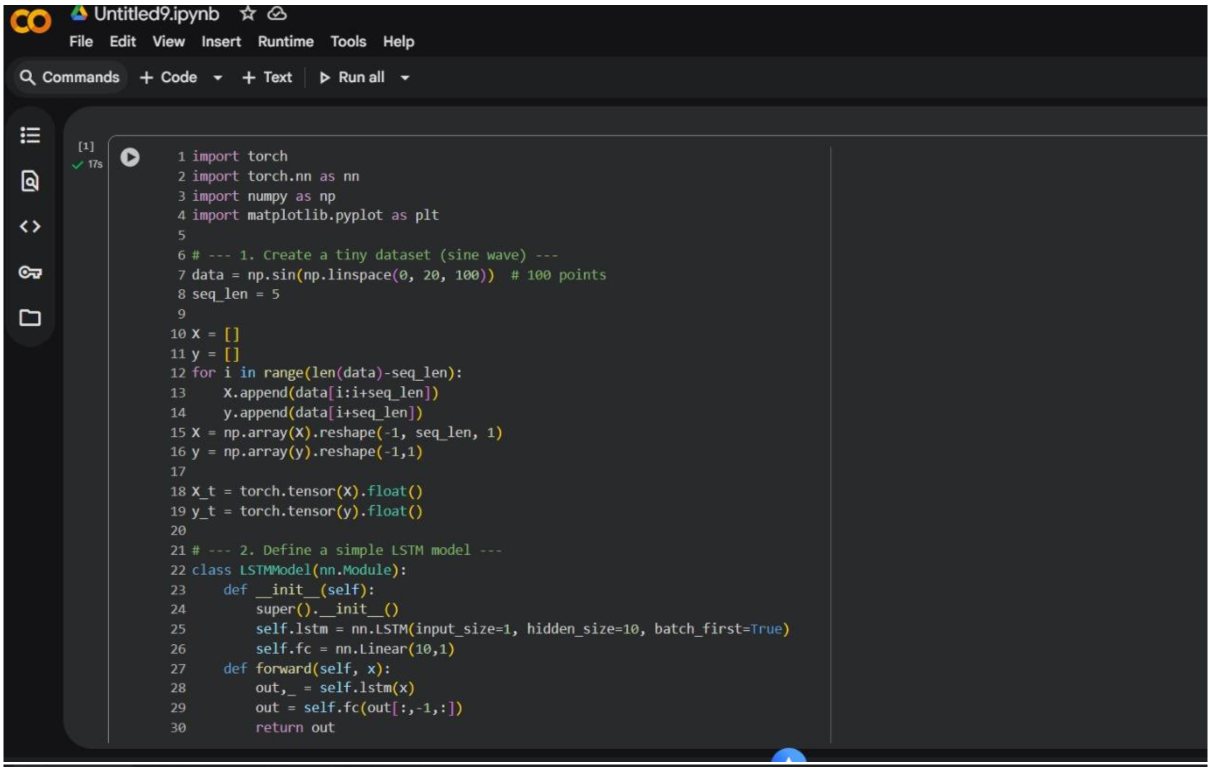
The training loss decreases gradually with each epoch, indicating that the model is learning the sequence pattern

LSTM performs better than simple RNN when dealing with long-term.

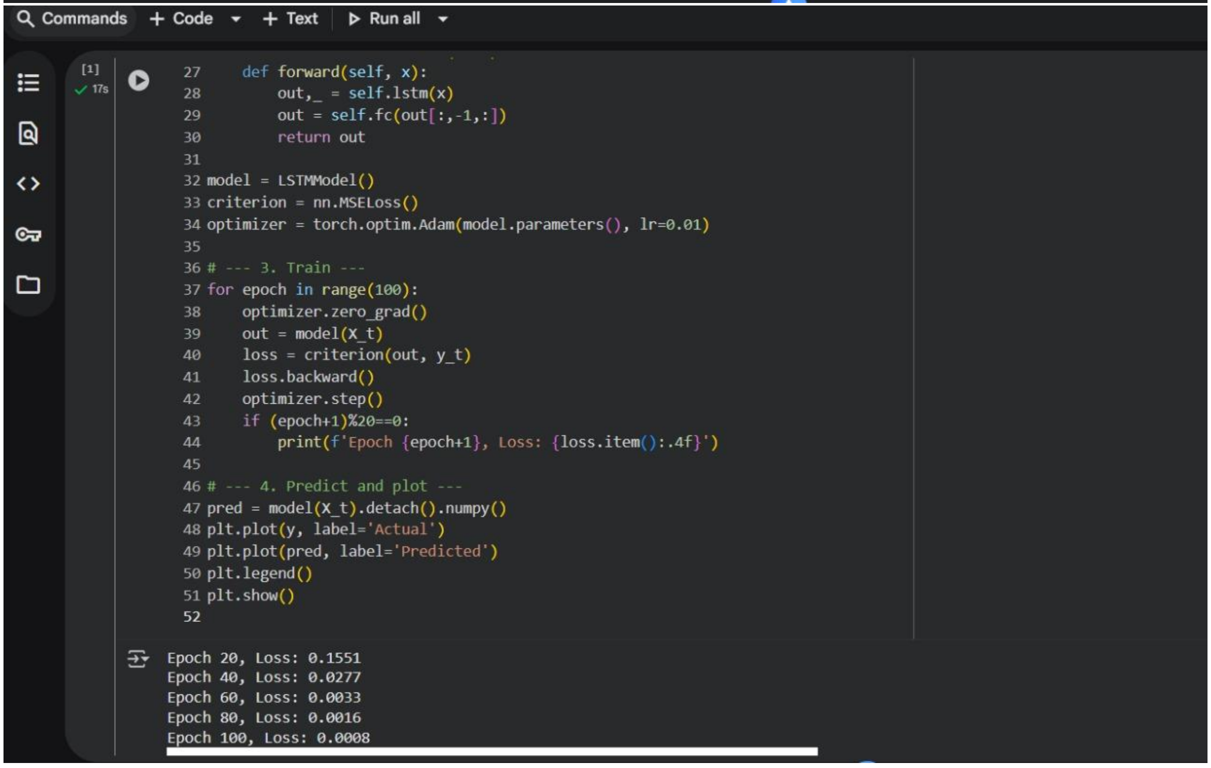
The predicted output closely follows the trend of a actual data, demonstrating the model's ability to remember previous content.

However, training time is higher compared to standard RNN due to more complex computation.

Result:
The experiment was successfully carried out and LSTM model was implemented to learn



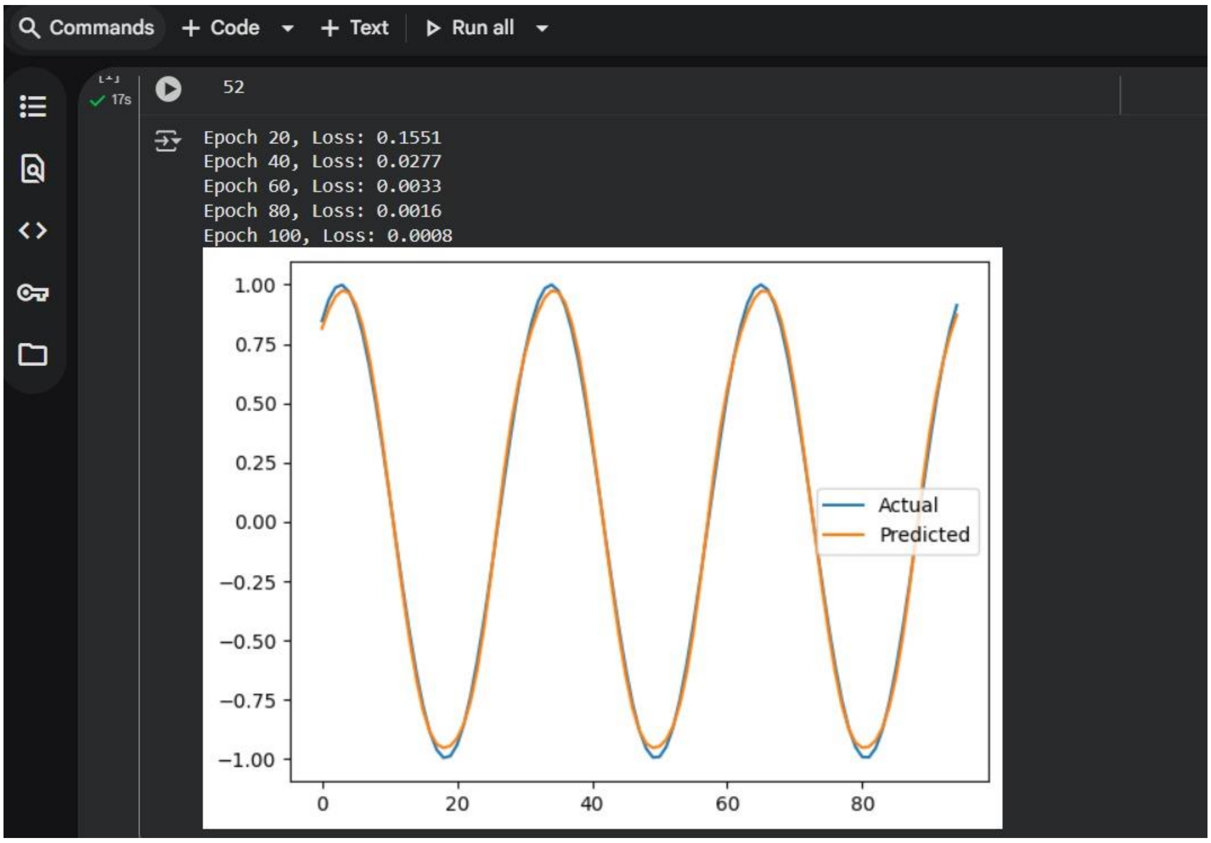
```python
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt

# --- 1. Create a tiny dataset (sine wave) ---
data = np.sin(np.linspace(0, 20, 100))  # 100 points
seq_len = 5

X = []
y = []
for i in range(len(data)-seq_len):
    X.append(data[i:i+seq_len])
    y.append(data[i+seq_len])
X = np.array(X).reshape(-1, seq_len, 1)
y = np.array(y).reshape(-1,1)

X_t = torch.tensor(X).float()
y_t = torch.tensor(y).float()

# --- 2. Define a simple LSTM model ---
class LSTMModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.lstm = nn.LSTM(input_size=1, hidden_size=10, batch_first=True)
        self.fc = nn.Linear(10,1)
    def forward(self, x):
        out,_ = self.lstm(x)
        out = self.fc(out[:,-1,:])
        return out
```



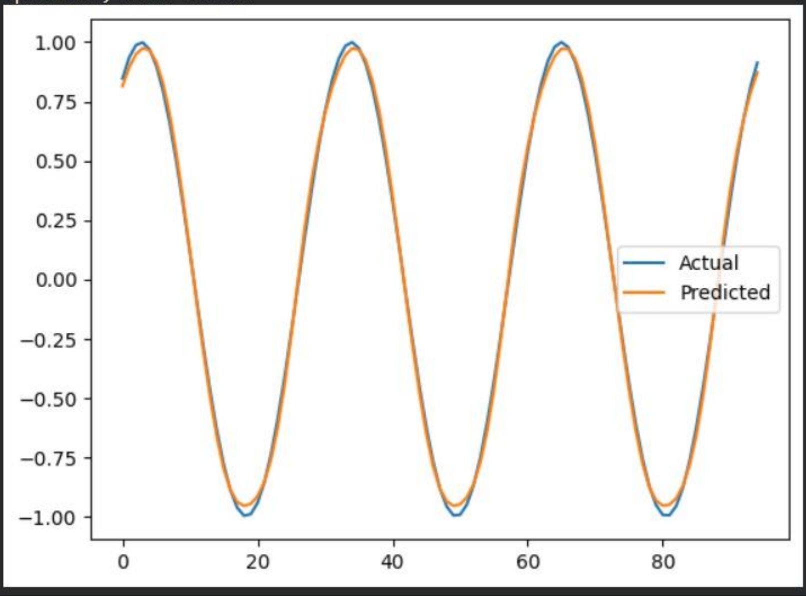
```python
    def forward(self, x):
        out,_ = self.lstm(x)
        out = self.fc(out[:,-1,:])
        return out
```

```python

model = LSTMModel()
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

# --- 3. Train ---
for epoch in range(100):
    optimizer.zero_grad()
    out = model(X_t)
    loss = criterion(out, y_t)
    loss.backward()
    optimizer.step()
    if (epoch+1)%20==0:
        print(f'Epoch {epoch+1}, Loss: {loss.item():.4f}')

# --- 4. Predict and plot ---
pred = model(X_t).detach().numpy()
plt.plot(y, label='Actual')
plt.plot(pred, label='Predicted')
plt.legend()
plt.show()

```

```
Epoch 20, Loss: 0.1551
Epoch 40, Loss: 0.0277
Epoch 60, Loss: 0.0033
Epoch 80, Loss: 0.0016
Epoch 100, Loss: 0.0008
```



```
Epoch 20, Loss: 0.1551
Epoch 40, Loss: 0.0277
Epoch 60, Loss: 0.0033
Epoch 80, Loss: 0.0016
Epoch 100, Loss: 0.0008
```


Actual
Predicted