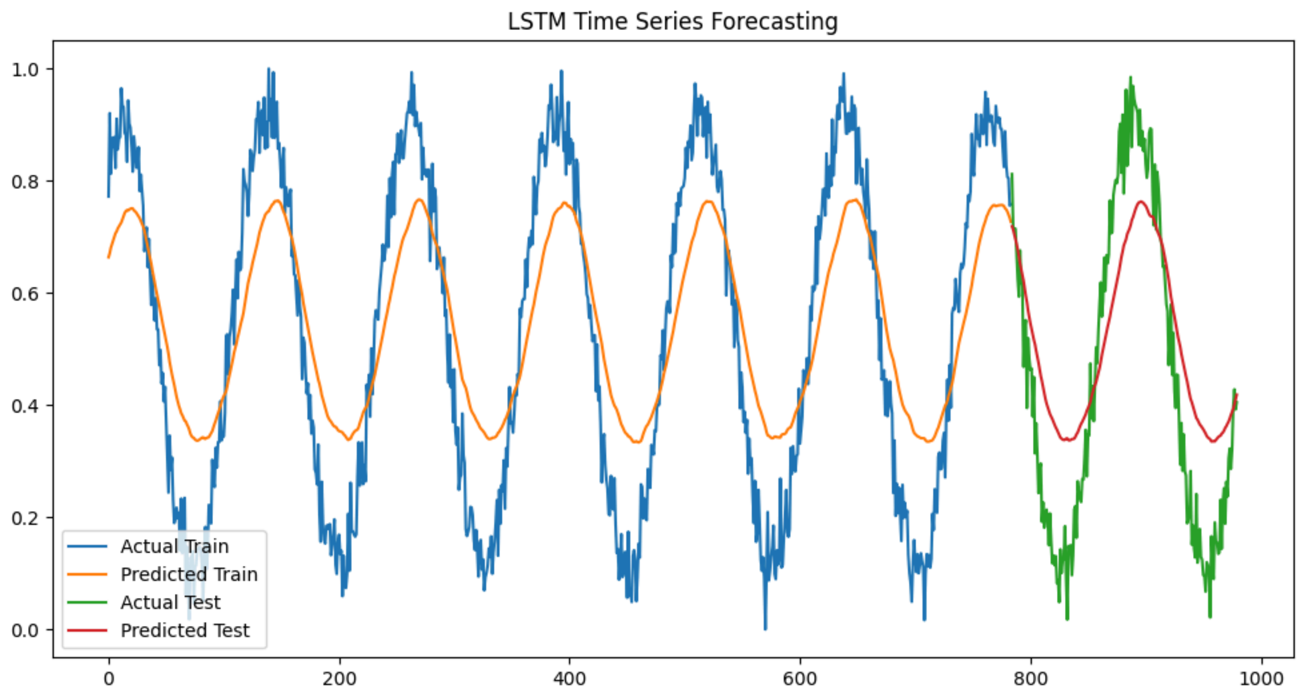
Pratham Nagar - 64 Experiment 07 ~ Design and implement LSTM model for time series forecasting.

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

# Generate synthetic sine wave data
t = np.linspace(0, 50, 1000)
data = np.sin(t) + 0.1 * np.random.randn(1000)

# Normalize data
data = (data - data.min()) / (data.max() - data.min())

# Create sequences
def create_sequences(data, seq_length=20):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return torch.FloatTensor(X), torch.FloatTensor(y)

X, y = create_sequences(data)
X = X.unsqueeze(-1)  # Add feature dimension

# Split data
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# LSTM Model
class TimeSeriesLSTM(nn.Module):
    def __init__(self):
        super().__init__()
        self.lstm = nn.LSTM(input_size=1, hidden_size=50, num_layers=2, batch_first=True, dropout=0.2)
        self.linear = nn.Linear(50, 1)

    def forward(self, x):
        x, _ = self.lstm(x)
        return self.linear(x[:, -1, :])

# Training
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = TimeSeriesLSTM().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Move data to device
X_train, y_train = X_train.to(device), y_train.to(device)
X_test, y_test = X_test.to(device), y_test.to(device)

# Train
epochs = 50
for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs.squeeze(), y_train)
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# Prediction
model.eval()
with torch.no_grad():
    train_pred = model(X_train)
    test_pred = model(X_test)

# Plot results
plt.figure(figsize=(12, 6))
plt.plot(y_train.cpu().numpy(), label='Actual Train')
plt.plot(train_pred.cpu().numpy(), label='Predicted Train')
plt.plot(range(len(y_train), len(y_train) + len(y_test)), y_test.cpu().numpy(), label='Actual Test')
plt.plot(range(len(y_train), len(y_train) + len(y_test)), test_pred.cpu().numpy(), label='Predicted Test')
plt.legend()
plt.title('LSTM Time Series Forecasting')
plt.show()
```

```
/tmp/ipython-input-3630769786.py:20: UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely slow. Please conside
  return torch.FloatTensor(X), torch.FloatTensor(y)
Epoch [10/50], Loss: 0.2049
Epoch [20/50], Loss: 0.0798
Epoch [30/50], Loss: 0.0615
Epoch [40/50], Loss: 0.0452
Epoch [50/50], Loss: 0.0272
```


LSTM Time Series Forecasting
Actual Train
Predicted Train
Actual Test
Predicted Test