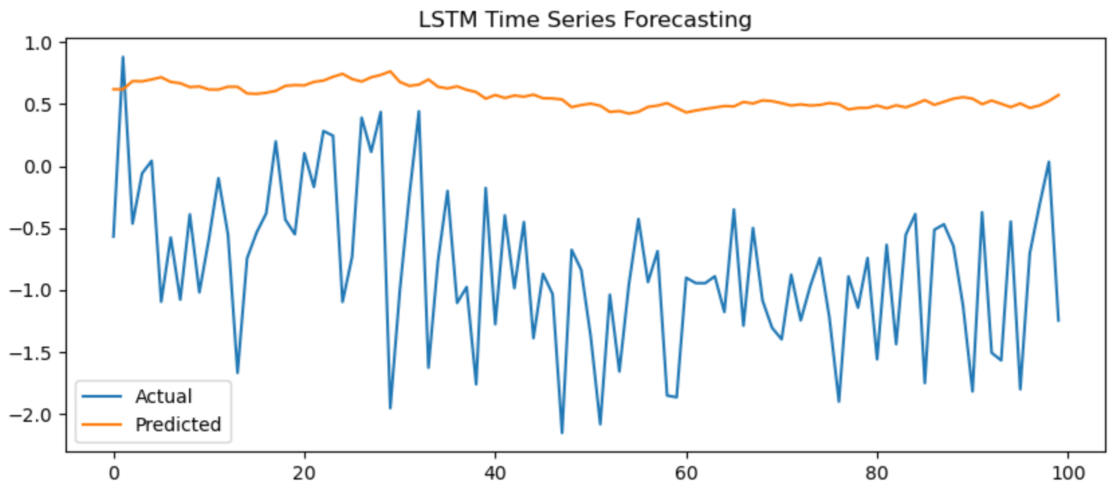
Forecasting_timeseries_Data_p6

October 26, 2025

Design and implement a deep learning network for forecasting time series data.

[4]:
```python
import numpy as np
import torch
import torch.nn as nn
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math
import matplotlib.pyplot as plt

# ----- 1) Generate synthetic time series -----
n = 1000
t = np.arange(n)
data = np.sin(0.02 * t) + 0.5 * np.random.randn(n)

# ----- 2) Scale data -----
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data.reshape(-1, 1))

# ----- 3) Create input-output sequences -----
def create_sequences(data, window=20):
    X, y = [], []
    for i in range(len(data) - window):
        X.append(data[i:i+window])
        y.append(data[i+window])
    return np.array(X), np.array(y)

X, y = create_sequences(data_scaled, 20)
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.float32)

# Train-test split
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# ----- 4) Define LSTM Model -----
class LSTMForecaster(nn.Module):
    def __init__(self, input_size=1, hidden_size=64):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, 1)
    def forward(self, x):
        out, _ = self.lstm(x)
        return self.fc(out[:, -1, :])

model = LSTMForecaster()
loss_fn = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# ----- 5) Train Model -----
for epoch in range(20):
    model.train()
    optimizer.zero_grad()
    output = model(X_train)
    loss = loss_fn(output, y_train)
    loss.backward()
    optimizer.step()
    if (epoch+1) % 5 == 0:
        print(f"Epoch {epoch+1}/20, Loss: {loss.item():.6f}")

# ----- 6) Evaluate -----
model.eval()
with torch.no_grad():
    preds = model(X_test).numpy()
    y_true = y_test.numpy()

# Inverse scale
preds_unscaled = scaler.inverse_transform(preds)
y_true_unscaled = scaler.inverse_transform(y_true)

mae = mean_absolute_error(y_true_unscaled, preds_unscaled)
rmse = math.sqrt(mean_squared_error(y_true_unscaled, preds_unscaled))
print(f"\nTest MAE: {mae:.4f}, RMSE: {rmse:.4f}")

# ----- 7) Plot -----
plt.figure(figsize=(10,4))
plt.plot(y_true_unscaled[:100], label="Actual")
plt.plot(preds_unscaled[:100], label="Predicted")
plt.legend()
plt.title("LSTM Time Series Forecasting")
plt.show()
```

```
Epoch 5/20, Loss: 0.197546
Epoch 10/20, Loss: 0.112522
Epoch 15/20, Loss: 0.039078
Epoch 20/20, Loss: 0.046198

Test MAE: 1.0682, RMSE: 1.2440
```



```
[ ]:
```