LSTM Forecasting with PyTorch

Saif Ullah

September 1, 2025

1 Introduction

This document demonstrates a simple implementation of a Long Short-Term Memory (LSTM) network in PyTorch to predict the values of a noisy sine wave. The LSTM is well-suited for sequence modeling tasks due to its ability to capture long-term dependencies.

2 Python Implementation

The following Python script was used for dataset creation, model definition, training, and evaluation:

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

# 1. Create synthetic sine wave data
T = 50  # sequence length (window)
L = 1000
x = np.linspace(0, 40*np.pi, L)
y = np.sin(x) + 0.1*np.random.randn(L)  # sine wave with noise

# Prepare dataset: sliding windows
def create_dataset(series, window_size):
    X, Y = [], []
    for i in range(len(series) - window_size):
        X.append(series[i:i+window_size])
        Y.append(series[i+window_size])
    return np.array(X, dtype=np.float32), np.array(Y, dtype=np.float32)

X, Y = create_dataset(y, T)

X = torch.from_numpy(X).unsqueeze(-1)   # (samples, seq, feature)
Y = torch.from_numpy(Y).unsqueeze(-1)   # (samples, 1)

# 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:]

# 2. Define LSTM model
class LSTMRegressor(nn.Module):
    def __init__(self, input_size=1, hidden_size=32, num_layers=1):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
           batch_first=True)
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        out, _ = self.lstm(x)             # out: (batch, seq, hidden)
        last = out[:, -1, :]              # take last timestep
        return self.fc(last)

model = LSTMRegressor()
loss_fn = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

# 3. Train loop
epochs = 20
for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    y_pred = model(X_train)
    loss = loss_fn(y_pred, Y_train)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 5 == 0:
        print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")

# 4. Evaluate
model.eval()
with torch.no_grad():
    preds = model(X_test).squeeze().numpy()
    true = Y_test.squeeze().numpy()

plt.plot(true, label="True")
plt.plot(preds, label="Predicted")
plt.legend()
plt.title("LSTM Forecasting Example")
plt.show()
```

3 Results

After training, the model predictions were compared with the ground truth sine wave values. The results are shown in Figure 1.

```
Epoch 5, Loss: 0.2871
Epoch 10, Loss: 0.0887
Epoch 15, Loss: 0.0366
Epoch 20, Loss: 0.0197
```

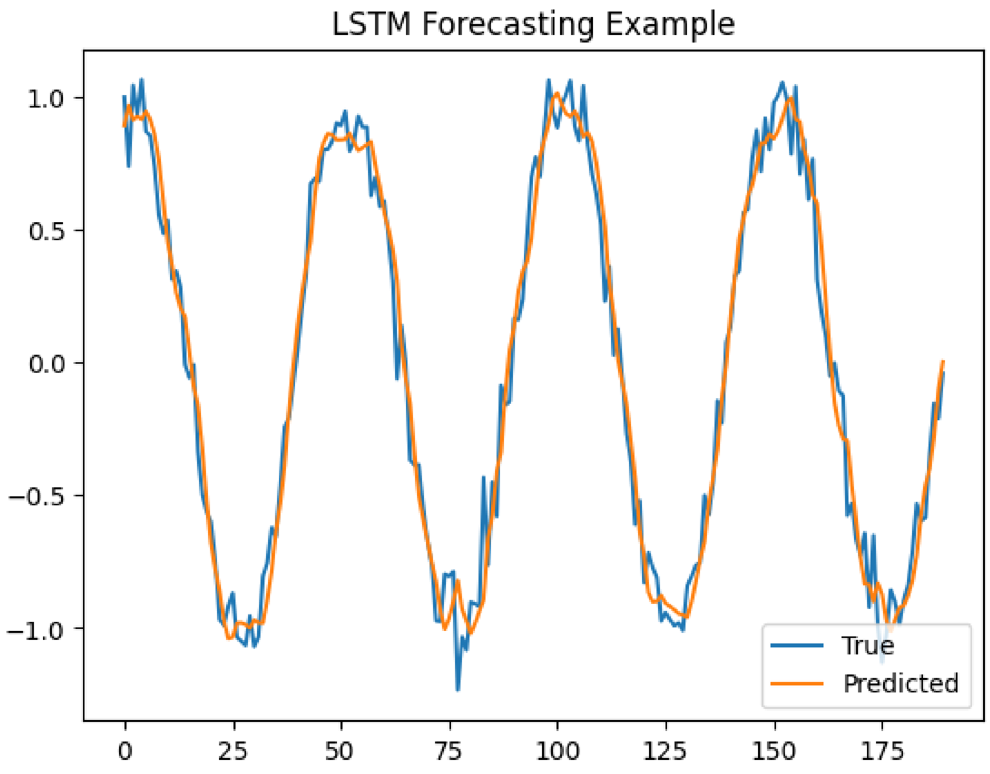

Figure 1: Predicted vs. true sine wave values using LSTM.