course evng.ipynb

File Edit View Insert Runtime Tools Help

Share

Commands + Code + Text Run all Reconnect

[ ]

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

# Step 1: Generate sine wave data
np.random.seed(42)
x = np.linspace(0, 100, 1000)
data = np.sin(x)

# Function to create sequences
def create_dataset(seq, look_back=20):
    X, Y = [], []
    for i in range(len(seq) - look_back):
        X.append(seq[i:i+look_back])
        Y.append(seq[i+look_back])
    return np.array(X), np.array(Y)

look_back = 20
```

Variables Terminal

5:56 PM

```
look_back = 20
X, Y = create_dataset(data, look_back)

# Split into train and test sets
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = Y[:train_size], Y[train_size:]

# Convert to tensors
X_train = torch.tensor(X_train, dtype=torch.float32).unsqueeze(-1)
y_train = torch.tensor(y_train, dtype=torch.float32).unsqueeze(-1)
X_test = torch.tensor(X_test, dtype=torch.float32).unsqueeze(-1)
y_test = torch.tensor(y_test, dtype=torch.float32).unsqueeze(-1)

# Step 2: Define the LSTM Model
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
```

[ ]

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

# Step 3: Initialize model, loss, optimizer
model = LSTMModel(input_size=1, hidden_size=50, output_size=1)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Step 4: Train the model
epochs = 200
for epoch in range(epochs):
    model.train()
    output = model(X_train)
    loss = criterion(output, y_train)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

{} Variables   Terminal

✓ 5:56 PM

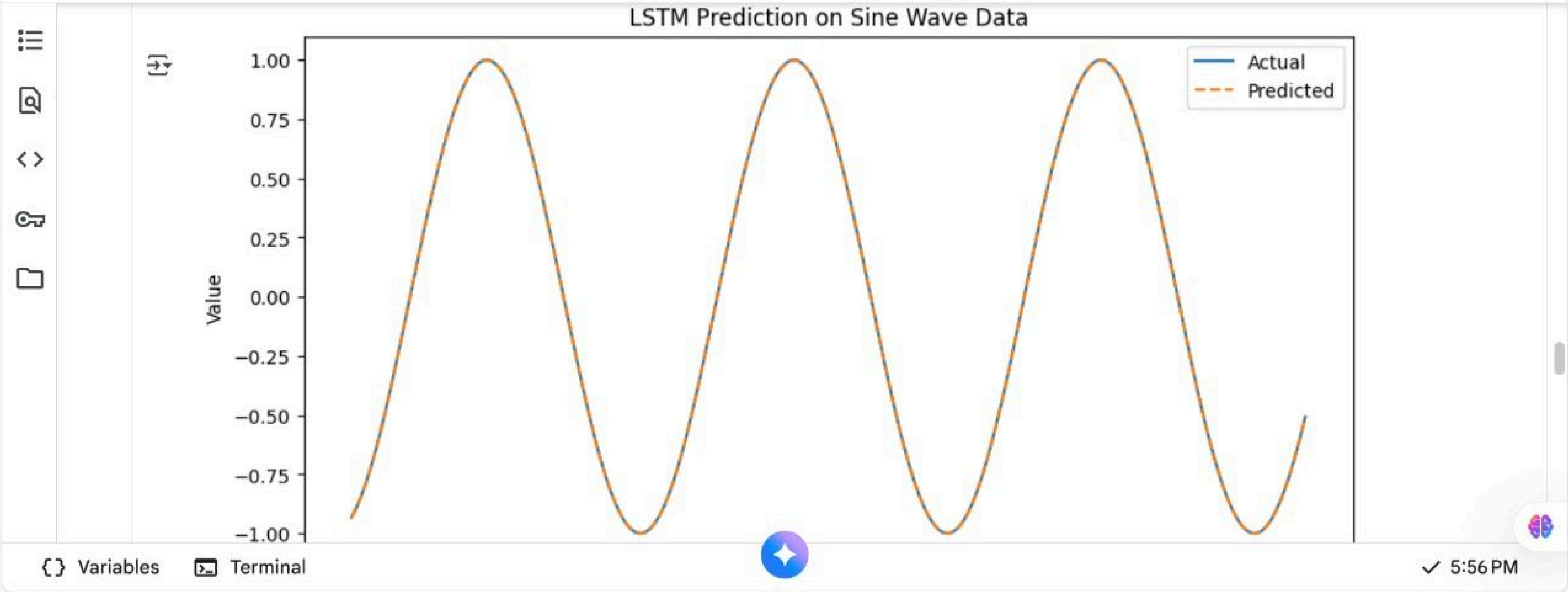
[ ]

```
        if (epoch+1) % 20 == 0:
            print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.6f}")

# Step 5: Evaluate
model.eval()
predicted = model(X_test).detach().numpy()

# Step 6: Plot
plt.figure(figsize=(10,5))
plt.plot(range(len(y_test)), y_test, label='Actual')
plt.plot(range(len(predicted)), predicted, label='Predicted', linestyle='--')
plt.title("LSTM Prediction on Sine Wave Data")
plt.xlabel("Time Step")
plt.ylabel("Value")
plt.legend()
plt.show()
```

```
Epoch [20/200], Loss: 0.004067
Epoch [40/200], Loss: 0.000231
Epoch [60/200], Loss: 0.000075
Epoch [80/200], Loss: 0.000019
Epoch [100/200], Loss: 0.000013
Epoch [120/200], Loss: 0.000007
Epoch [140/200], Loss: 0.000004
Epoch [160/200], Loss: 0.000003
Epoch [180/200], Loss: 0.000002
Epoch [200/200], Loss: 0.000001
```


LSTM Prediction on Sine Wave Data
Actual
Predicted
Value
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Variables
Terminal
5:56 PM

9/10/25 8. Experiment using LSTM

Aim :

TO implement and train a LSTM network using PyTorch to learn temporal dependency and predict future values

Objective :

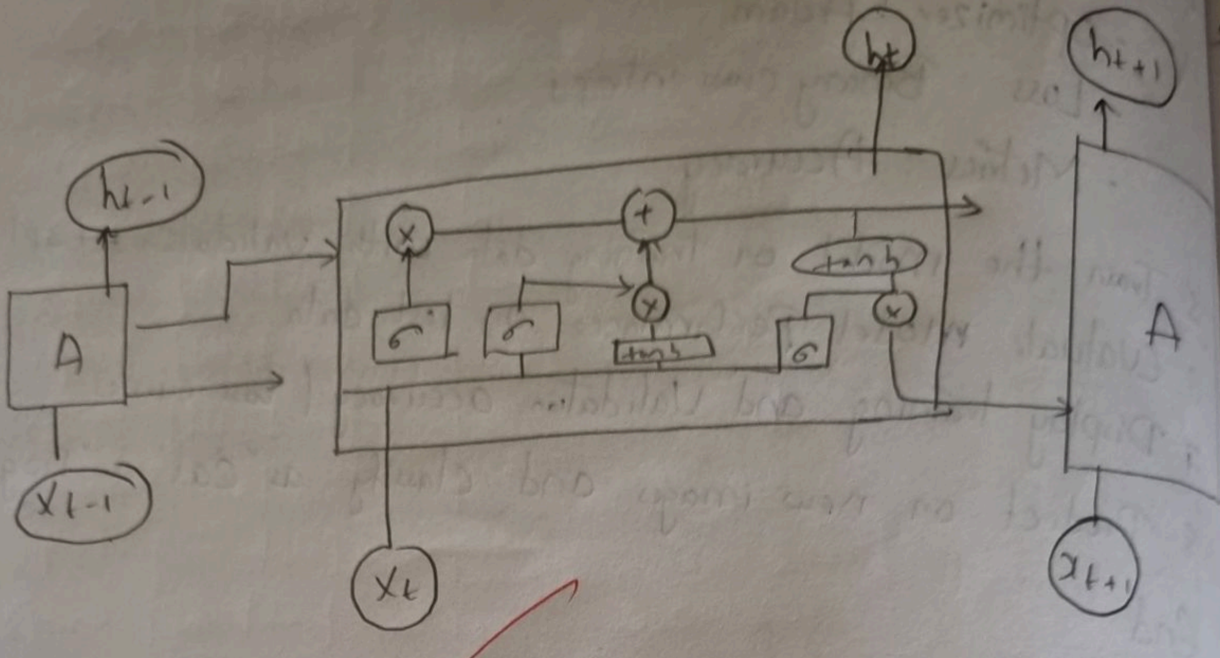
1. To understand structure and working of the LSTM model
2. To implement an LSTM model using PyTorch
3. To train and evaluate the model for sequence prediction tasks
4. To Compare the performance of LSTM with a simple RNN in capturing long term dependencies

Pseudo code

1. Import all the necessary libraries.
2. Prepare dataset
   - Generate sine wave data
   - Convert into input output pairs
   - split into training and testing sets
   - Convert to Pytorch tensors

3. Define the LSTM model
4. Initealize model, define loss function and optimizer

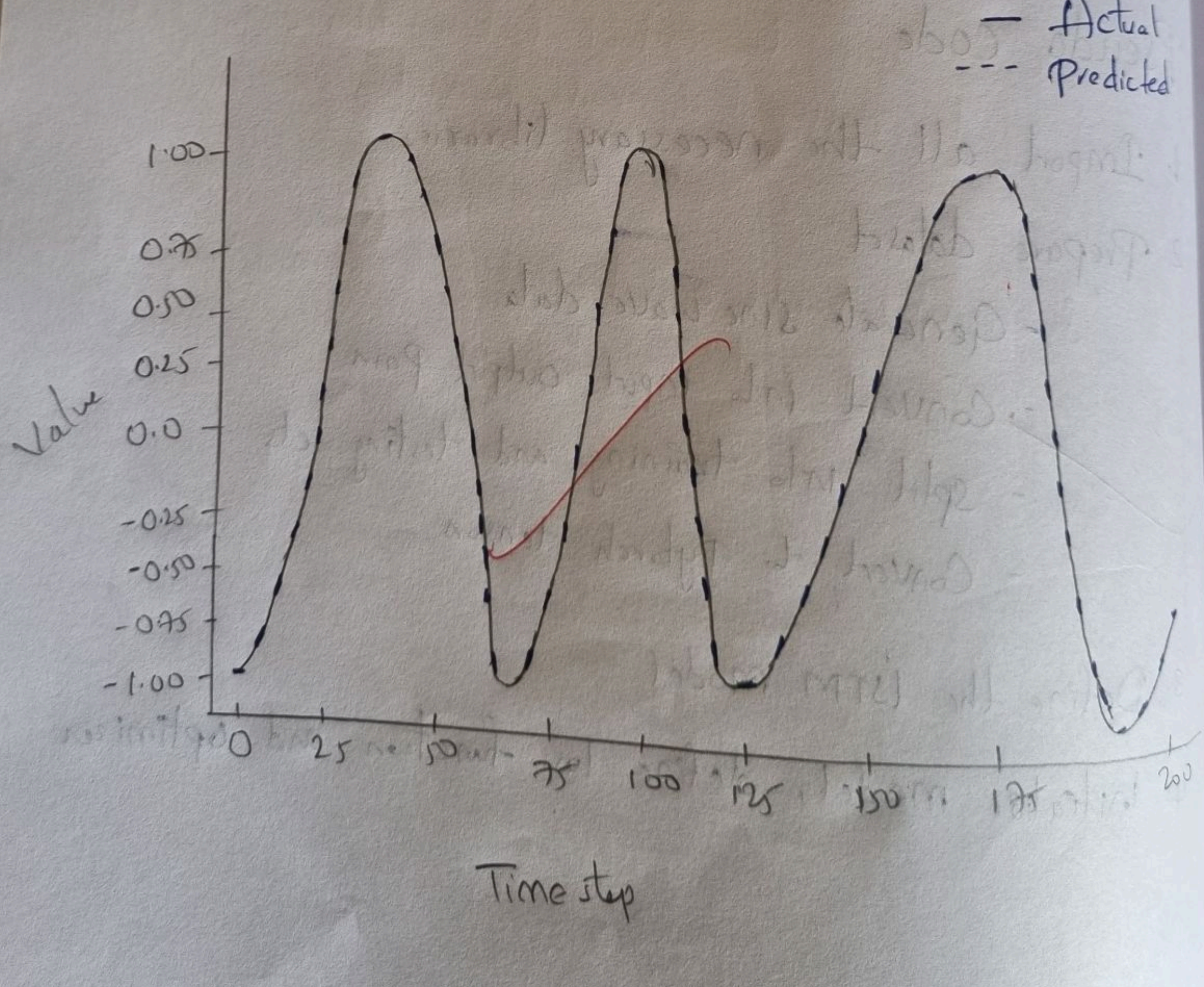
LSTM



O/P

Epoch [20|200] , loss : 0.004067
Epoch [40|200] , loss : 0.000231
Epoch [60|200] , loss : 0.000075
Epoch [80|200] , loss : 0.000019
Epoch [100|200] , loss : 0.000013
Epoch [120|200] , loss : 0.000007
Epoch [140|200] , loss : 0.000004
Epoch [160|200] , loss : 0.000003
Epoch [180|200] , loss : 0.000002
Epoch [200|200] , loss : 0.000001

LSTM prediction on sinewave data



5. Train the model
   - Forward pass
   - Compute loss
   - Backpropagate
   - Update weights
6. Evaluate on test data
   - Use model.eval()
   - Compare predicted vs actual
7. Visualize the result

Observation :

1. The training loss decreases gradually and stabilized after multiple epochs
2. LSTM model effectively captured the periodic pattern of sine wave
3. Compared to simple RNN, LSTM showed smoother predictions
4. The predicted sine curve closely follows the actual one.

Result :

Successfully implemented the experiment using LSTM.