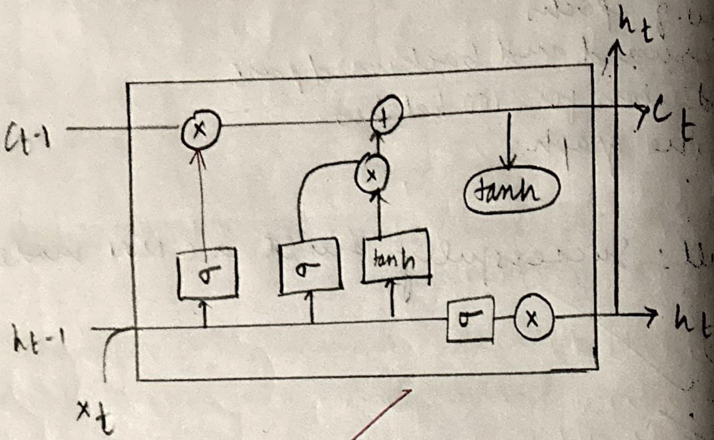
LSTM Architecture



Experiment 8
23/9/25

Build AN LSTM Model

Aim: To build an train a Long Short Term Memory (LSTM) model using PyTorch for time series forecasting

Objectives:

- To generate or load a numerical time series dataset.
- To process data and create sequential input samples for the LSTM
- To evaluate model performance on unseen test data

Pseudocode:

- Start
- Import necessary libraries
- Genrate or LOAD the time series data
- Preprocess the data
  - Define sequence_length.
  - For each i in range (len(data) - sequence_len)
    x[i] = data [i : i + sequence_length]
    y[i] = data [i + sequence_length]
  - split data into train and test set
- Define the LSTM model
  - Input size = 1
  - Hidden size = 64
  - Number of layer = 2
  - Output layer = 1 neuron

  Forward pass:
  out, _ = LSTM (x)
  output = FullyConnected (out [: , -1 , :])
- Train (Predict , output , compute loss, update weights)

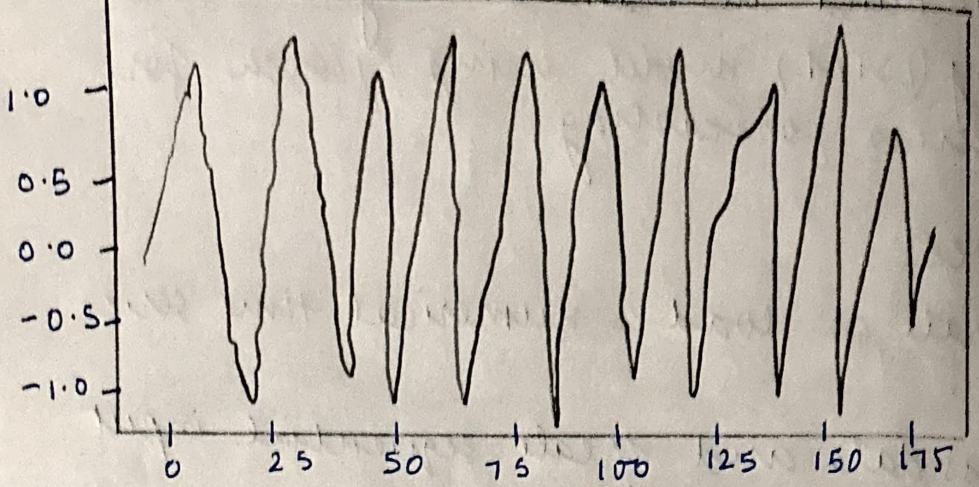
→ Forecast future ~~ttt~~ values after testing the model

→ Plot results.

→ End.

Result: Successfully built LSTM model

NOISY SINE WAVE TIME SERIES



epoch [1/25] : loss : 0.244074

Epoch [2/25] : loss : 0.018688

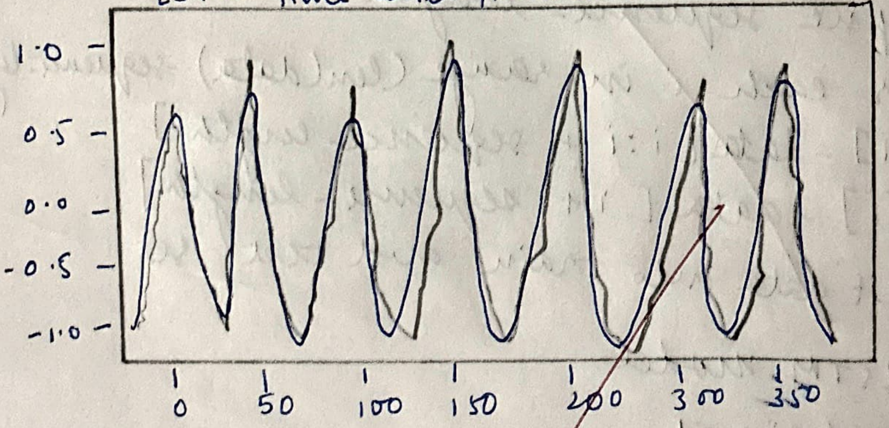
epoch [3/25] : loss : 0.014108

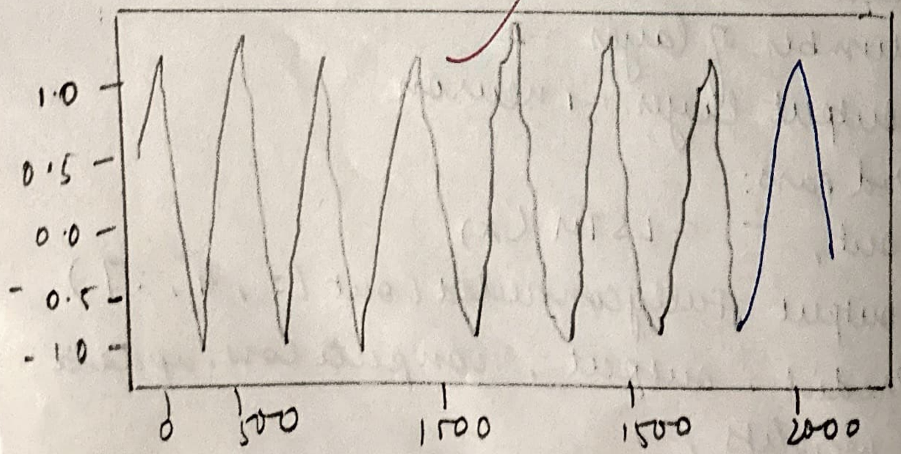
epoch [4/25] : loss : 0.013592

:

epoch [25/25] : loss 0.011999

LSTM Time series Prediction





```
Successfully installed torchtext-0.18.0
```

In [ ]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader, TensorDataset

np.random.seed(42)
time = np.arange(0, 200, 0.1)
data = np.sin(time) + 0.1 * np.random.randn(len(time))

plt.figure(figsize=(8,3))
plt.plot(time, data)
plt.title("Noisy Sine Wave Time Series")
plt.show()

def create_sequences(data, seq_length):
    xs, ys = [], []
    for i in range(len(data) - seq_length):
        x = data[i:i+seq_length]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 50
X, y = create_sequences(data, seq_length)

X = torch.tensor(X, dtype=torch.float32).unsqueeze(-1)  # (samples, seq_len, 1)
y = torch.tensor(y, dtype=torch.float32).unsqueeze(-1)  # (samples, 1)
```

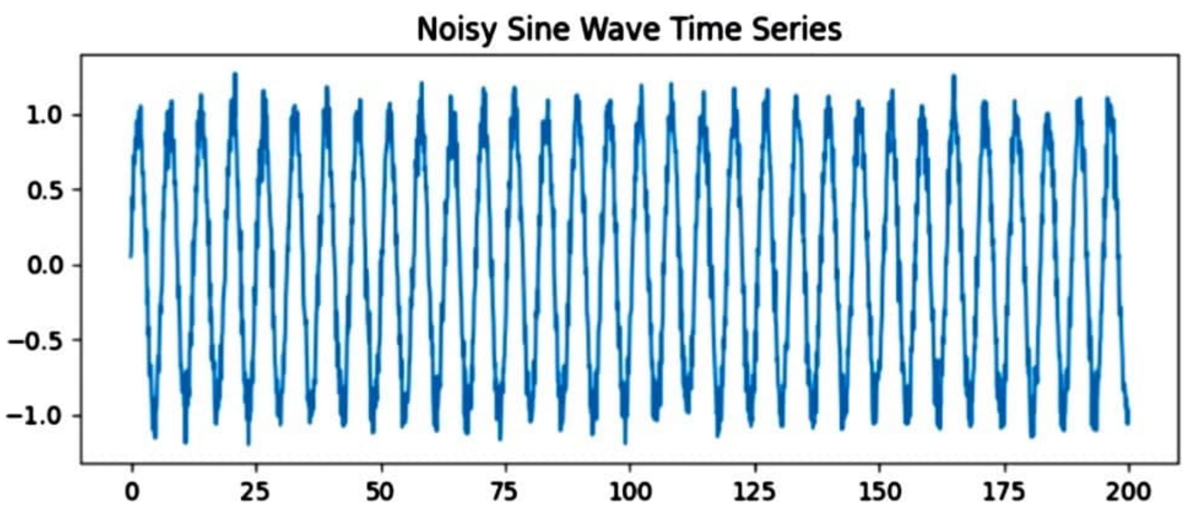
10:10 PM

```python
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:]

train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=32, shuffle=True)


class LSTMRegressor(nn.Module):
    def __init__(self, input_size=1, hidden_size=64, num_layers=2):
        super(LSTMRegressor, self).__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 = out[:, -1, :]   # use last time step
        out = self.fc(out)
        return out

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = LSTMRegressor().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 25
for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    for batch_X, batch_y in train_loader:
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        optimizer.zero_grad()
        output = model(batch_X)
        loss = criterion(output, batch_y)
        loss.backward()
```

10:10 PM

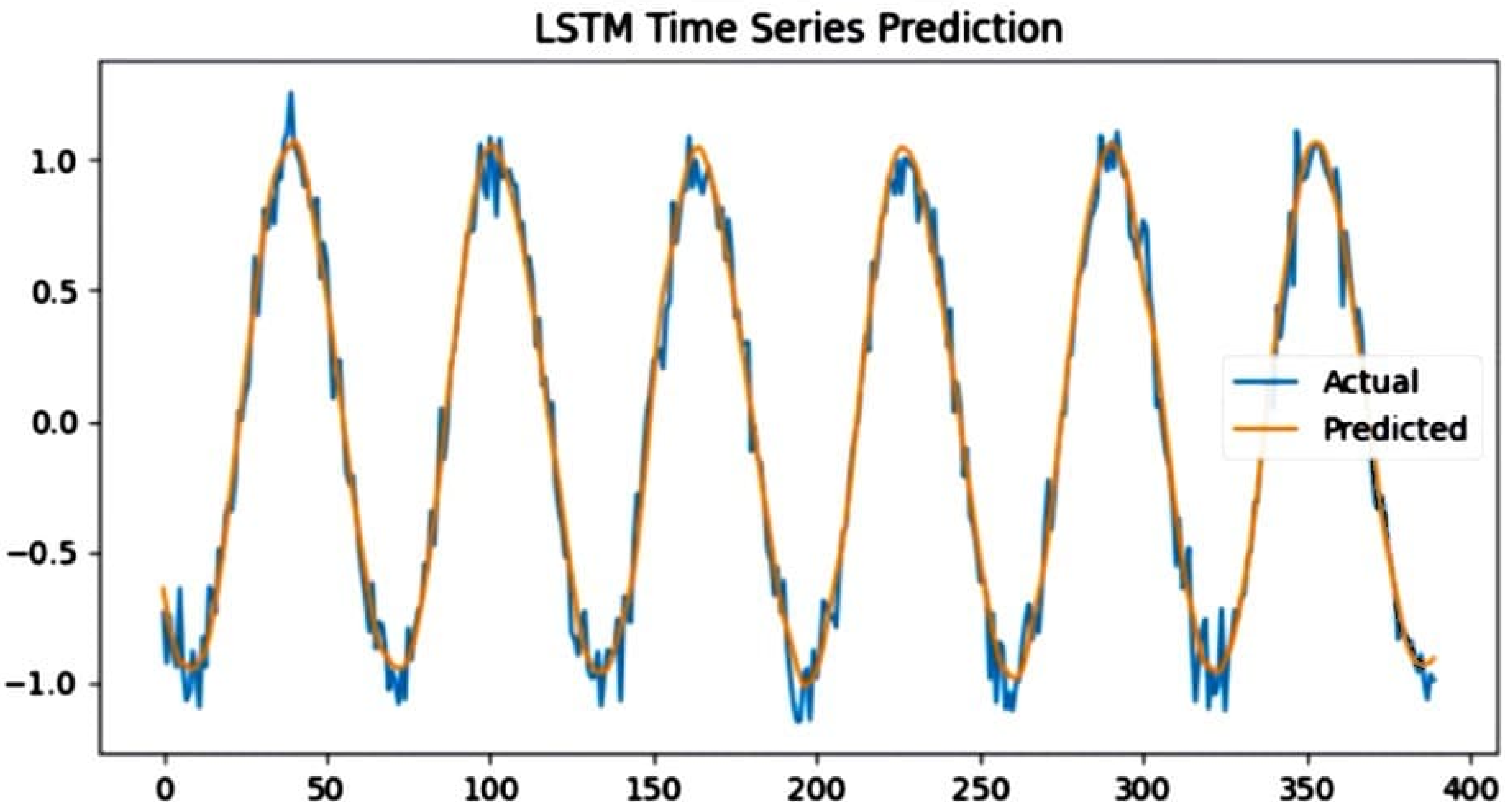
```python
        total_loss += loss.item()
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {total_loss/len(train_loader):.6f}")


model.eval()
with torch.no_grad():
    preds = model(X_test.to(device)).cpu().numpy()
    actual = y_test.cpu().numpy()

plt.figure(figsize=(8,4))
plt.plot(range(len(actual)), actual, label='Actual')
plt.plot(range(len(preds)), preds, label='Predicted')
plt.legend()
plt.title("LSTM Time Series Prediction")
plt.show()


with torch.no_grad():
    seq = X_test[-1].unsqueeze(0).to(device)
    future_preds = []
    for _ in range(50):
        pred = model(seq)
        future_preds.append(pred.item())
        seq = torch.cat([seq[:, 1:, :], pred.unsqueeze(1)], dim=1)


plt.figure(figsize=(8,3))
plt.plot(range(len(data)), data, label='Original Data')
plt.plot(range(len(data), len(data)+50), future_preds, label='Future Prediction')
plt.legend()
plt.title("Future Forecasting with LSTM")
plt.show()
```

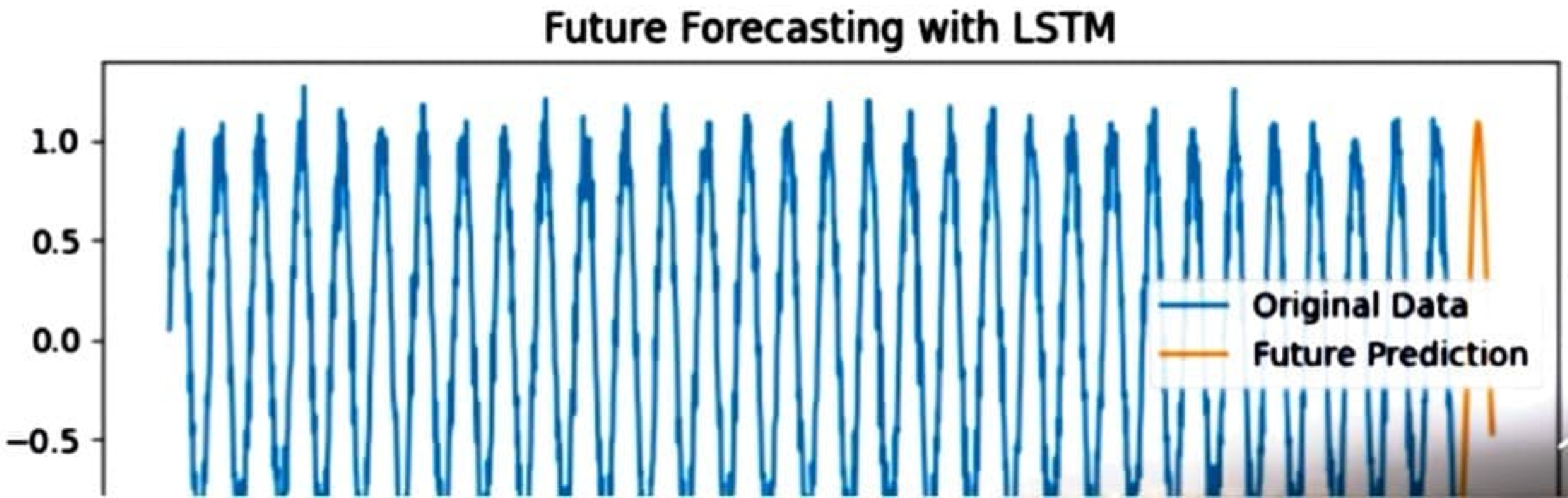
10:11 PM ✓




```
Epoch [1/25], Loss: 0.244074
Epoch [2/25], Loss: 0.018688
Epoch [3/25], Loss: 0.014108
Epoch [4/25], Loss: 0.013900
Epoch [5/25], Loss: 0.013592
Epoch [6/25], Loss: 0.012818
Epoch [7/25], Loss: 0.012583
Epoch [8/25], Loss: 0.012952
Epoch [9/25], Loss: 0.013004
Epoch [10/25], Loss: 0.013327
Epoch [11/25], Loss: 0.013180
Epoch [12/25], Loss: 0.012638
Epoch [13/25], Loss: 0.012094
Epoch [14/25], Loss: 0.011980
Epoch [15/25], Loss: 0.012476
Epoch [16/25], Loss: 0.012440
Epoch [17/25], Loss: 0.011978
Epoch [18/25], Loss: 0.011622
```

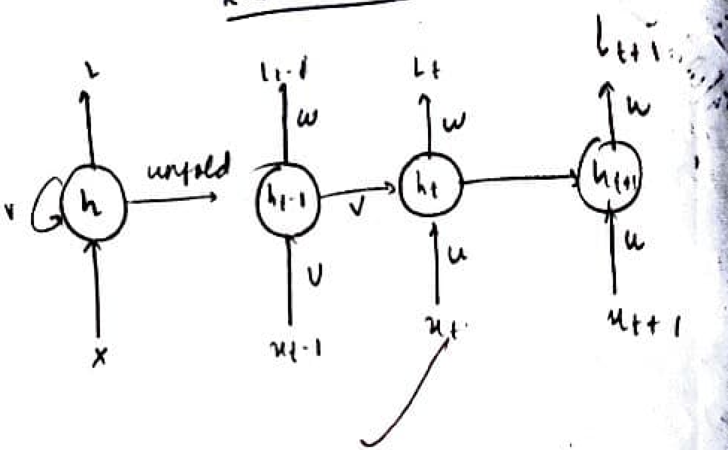
10:11 PM ✓


LSTM Time Series Prediction
1.0
0.5
0.0
−0.5
−1.0
0
50
100
150
200
250
300
350
400
Actual
Predicted


Future Forecasting with LSTM
1.0
0.5
0.0
−0.5
Original Data
Future Prediction

10:11 PM ✓

RNN structure



xperiment 9 — Build an RNN Model
0/9/25

Aim: To develop and train a Recurrent Neural Network

Objective:

- Generate sequential data
- Build an RNN model
- Train the model
- Visualize predicted vs Actual values

Pseudocode:

```
Start
Import required libraries
Set hyperparameters
    TIMESTEPS =10
    RNN_UNITS =32
    EPOCHS =100
    BATCH-SIZE =16
Generate synthetic sequential data
Create dataset for RNN:
Reshape input data
Split data into training & testing data
Build RNN Model :
    model= sequential( )
    Add SimpleRNN layer
    Add Dense layer
Train the model
    history =model.fit (x_train, ytrain,
                 epoch= EPOCHS , validation_split=0.1)
Evaluate model on test data
loss , mae = model.evaluate(x-test, y-test )
rmse = sqrt (loss)
```

redict using trained model
isualize predictions:
end.

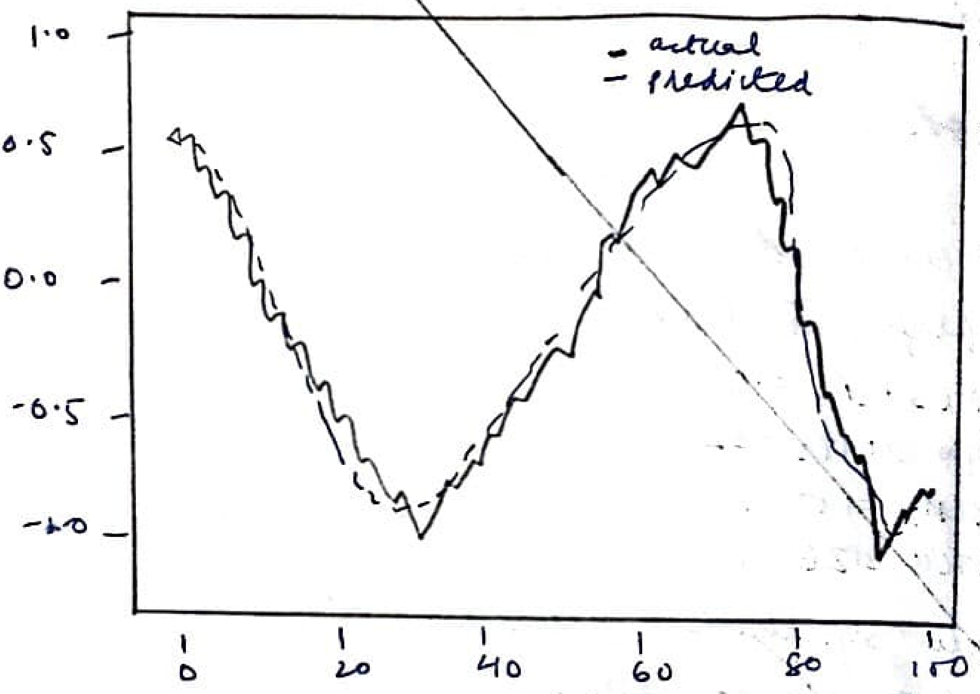
esult: Successfully built RNN model.

1.9.2025

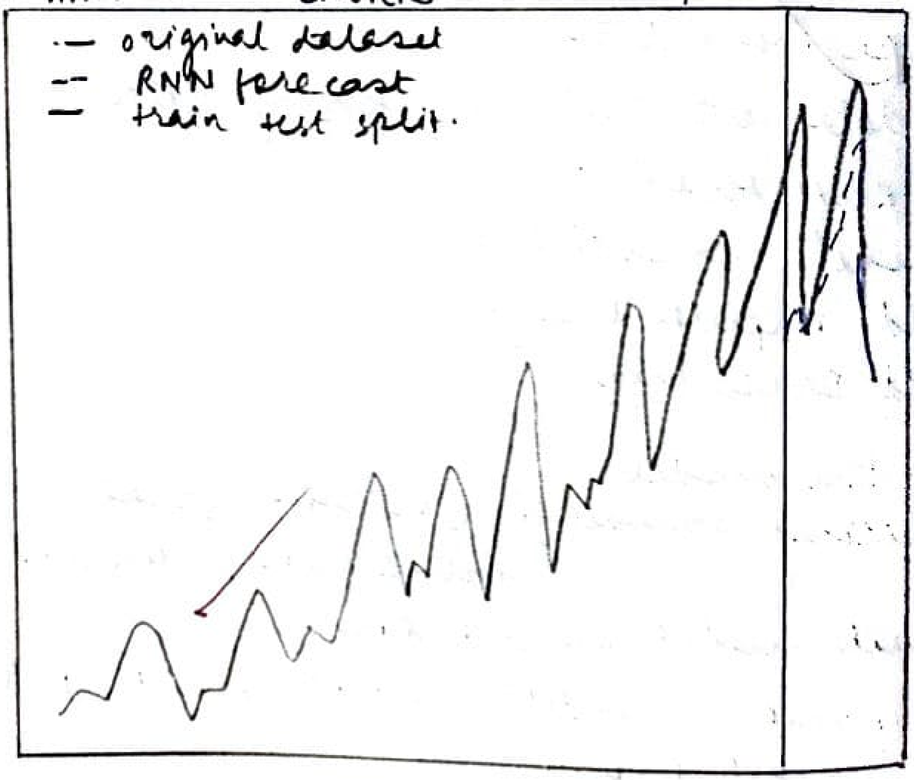
Test loss (MSE) = 0.0169

Test MAE : 0.1070

Test RMSE : 0.1299



AIRLINE PASSENGERS FORECASTING



```python
import torch
import torch.nn as nn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from torch.utils.data import Dataset, DataLoader
import seaborn as sns

data = sns.load_dataset("flights")
all_data = data['passengers'].values.astype(float)

test_data_size = 12
train_data = all_data[:-test_data_size]
test_data = all_data[-test_data_size:]

scaler = MinMaxScaler(feature_range=(-1, 1))
train_data_normalized = scaler.fit_transform(train_data.reshape(-1, 1))
train_data_normalized = torch.FloatTensor(train_data_normalized).view(-1)

def create_inout_sequences(input_data, tw):
    inout_seq = []
    L = len(input_data)
    for i in range(L - tw):
        train_seq = input_data[i:i + tw]
        train_label = input_data[i + tw:i + tw + 1]
        inout_seq.append((train_seq, train_label))
    return inout_seq

train_window = 12
train_inout_seq = create_inout_sequences(train_data_normalized, train_window)

class TimeSeriesDataset(Dataset):
    def __init__(self, sequences):
```

10:14 PM


```python
class TimeSeriesDataset(Dataset):
    def __init__(self, sequences):
```

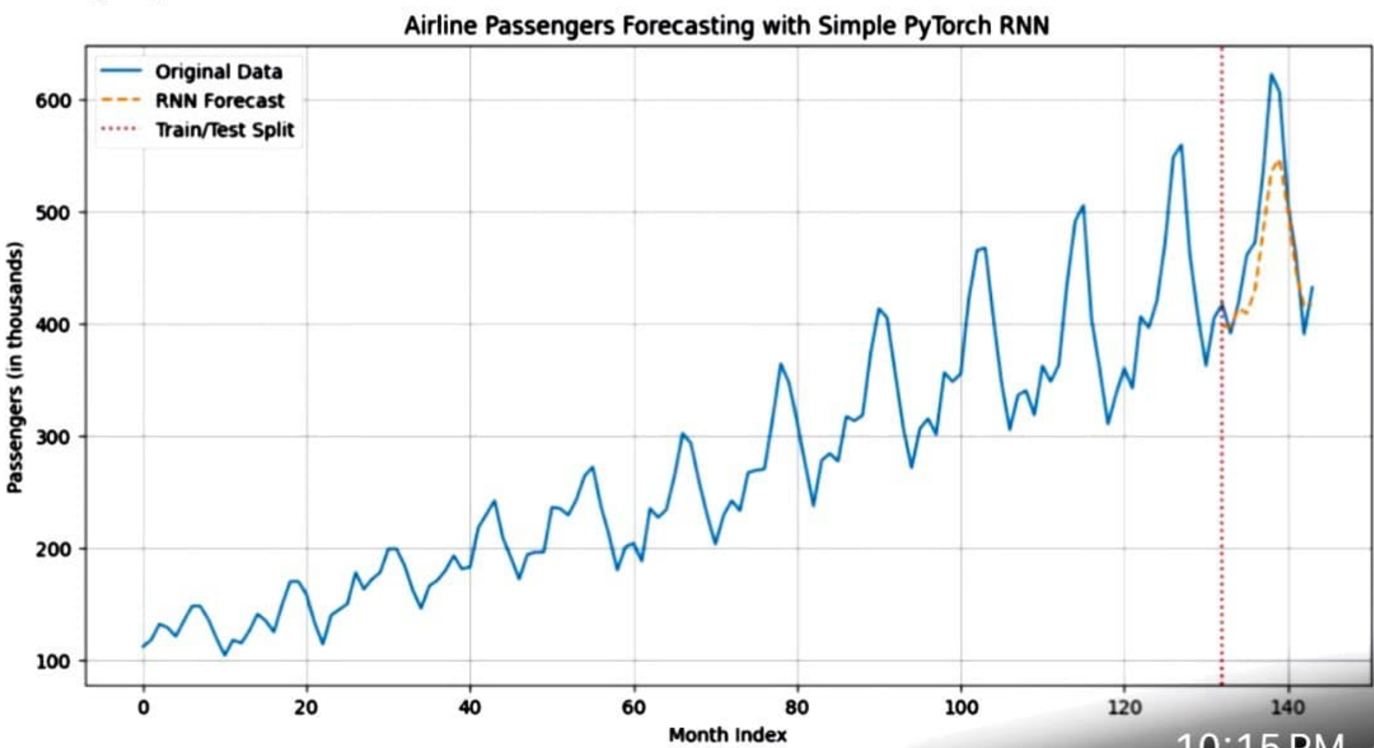

```python
        super().__init__()
        self.sequences = sequences
    def __len__(self):
        return len(self.sequences)
    def __getitem__(self, idx):
        return self.sequences[idx][0], self.sequences[idx][1]

train_dataset = TimeSeriesDataset(train_inout_seq)
batch_size = 10
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

class SimpleRNN(nn.Module):
    def __init__(self, input_size=1, hidden_layer_size=100, output_size=1):
        super().__init__()
        self.hidden_layer_size = hidden_layer_size

        self.rnn = nn.RNN(input_size, hidden_layer_size, batch_first=True)

        self.linear = nn.Linear(hidden_layer_size, output_size)

    def forward(self, input_seq):

        rnn_input = input_seq.unsqueeze(-1)

        h0 = torch.zeros(1, rnn_input.size(0), self.hidden_layer_size).to(input_seq.device)

        rnn_out, h_n = self.rnn(rnn_input, h0)

        predictions = self.linear(rnn_out[:, -1, :])
        return predictions

input_dim = 1
```

10:14 PM

```python
        optimizer.zero_grad()

        y_pred = model(seq)

        single_loss = loss_function(y_pred, labels)

        single_loss.backward()
        optimizer.step()

    if epoch % 20 == 0:
        print(f'Epoch {epoch:3} Loss: {single_loss.item():10.8f}')

print(f'Final Loss: {single_loss.item():10.8f}')
print("Training complete!")

model.eval()

test_input = scaler.transform(test_data.reshape(-1, 1))
test_input = torch.FloatTensor(test_input).view(-1)

fut_pred = 12
test_inputs = train_data_normalized[-train_window:].tolist()

for i in range(fut_pred):
    seq = torch.FloatTensor(test_inputs[-train_window:])

    with torch.no_grad():

        y_pred = model(seq.unsqueeze(0)).squeeze()

    test_inputs.append(y_pred.item())

actual_predictions = test_inputs[-fut_pred:]
```

10:15 PM

```
Starting training SimpleRNN on 120 sequences...
Epoch   0 Loss: 0.11418664
Epoch  20 Loss: 0.00629161
Epoch  40 Loss: 0.00531148
Epoch  60 Loss: 0.01036970
Epoch  80 Loss: 0.00373696
Final Loss: 0.00587350
Training complete!
```



10:15 PM