Lab Work 5: Time Series Forecasting with LSTM

Name: Al Amin Hossain Nayem

Date: April 6, 2025

Objective

The goal of this lab work is to implement a Long Short-Term Memory (LSTM) model to forecast SO_2 (Sulfur Dioxide) concentrations 1 to 4 hours ahead based on past observations. The model is trained on January 2018 data and tested on February 2018 data.

Tools and Libraries

- Python
- PyTorch
- NumPy
- Matplotlib

Dataset

- **Training set:** 2018_01_so2.npy
- **Testing set:** 2018_02_so2.npy

Code Overview

Dataset Preparation

The TimeSeriesDataset class is used to generate sequences:

- Input window size: 24 hours
- Output window size: 4 hours

Model Definition

An LSTM network with:

- Input size = 1
- Hidden size = 128
- 2 layers
- Dropout = 0.2

- Fully connected (FC) layer outputs 4 values for next 4 hours.

Normalization

The data is normalized using the training set's mean and standard deviation.

Training

- Optimizer: Adam
- Learning rate: 0.001
- Loss function: Mean Squared Error (MSE)
- Epochs: 150
- Batch size: 64

Testing and Evaluation

- Total test loss is computed.
- Individual loss is computed for each of the 4 prediction hours.
- True and predicted values are plotted for each forecast horizon.

Results

Loss during Training

Loss steadily decreased across 150 epochs, indicating successful model convergence.

Testing Results

- **Total Test Loss (MSE):** (Shown in the console)
- **Individual Losses:**
  - 1 hour ahead: (Shown in the console)
  - 2 hours ahead: (Shown in the console)
  - 3 hours ahead: (Shown in the console)
  - 4 hours ahead: (Shown in the console)

Plots

Graphs were generated comparing true vs. predicted SO_2 concentrations for:

- 1 hour ahead

- 2 hours ahead
- 3 hours ahead
- 4 hours ahead

These plots demonstrate that the model captures the temporal pattern well but with increasing error as prediction horizon extends.

Conclusion

An LSTM-based model was successfully built and trained to forecast short-term SO_2 concentrations. The model shows good predictive performance for immediate future steps but degrades slightly for farther steps. The performance could be further improved by:

- Tuning hyperparameters (e.g., hidden size, learning rate)
- Using more complex architectures (e.g., attention mechanisms)
- Training on more data.

Figures:

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 6 10:49:57 2025

@author: ginta
"""


import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
import matplotlib.pyplot as plt


# Define the Dataset
```

```
class TimeSeriesDataset(Dataset):
  def __init__(self, series, input_window=24, output_window=4):
    self.series = series
    self.input_window = input_window
    self.output_window = output_window
    self.samples = []
    for i in range(len(series) - input_window - output_window + 1):
      x = series[i : i + input_window]
      y = series[i + input_window : i + input_window + output_window]
      self.samples.append((x, y))

  def __len__(self):
    return len(self.samples)

  def __getitem__(self, index):
    x, y = self.samples[index]
    return torch.tensor(x, dtype=torch.float32).unsqueeze(-1), torch.tensor(y,
dtype=torch.float32)

# Define the Model
class LSTMModel(nn.Module):
  def __init__(self, input_size=1, hidden_size=128, num_layers=2, output_size=4):
    super(LSTMModel, self).__init__()
    self.hidden_size = hidden_size
    self.num_layers = num_layers
```

```
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True,
dropout=0.2)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.lstm(x, (h0, c0))
        out = out[:, -1, :]
        out = self.fc(out)
        return out

# Load and normalize the data
train = np.load('2018_01_so2.npy')
test = np.load('2018_02_so2.npy')

# Normalize
train_mean = train.mean()
train_std = train.std()

train = (train - train_mean) / train_std
test = (test - train_mean) / train_std

# Create Dataset and DataLoader
dataset = TimeSeriesDataset(train, input_window=24, output_window=4)
dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
```

```
# Hyperparameters
num_epochs = 150
learning_rate = 0.001

# Model, Loss, Optimizer
model = LSTMModel(input_size=1, hidden_size=128, num_layers=2, output_size=4)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Training loop
for epoch in range(num_epochs):
  model.train()
  epoch_loss = 0.0
  for x_batch, y_batch in dataloader:
    optimizer.zero_grad()
    predictions = model(x_batch)
    loss = criterion(predictions, y_batch)
    loss.backward()
    optimizer.step()
    epoch_loss += loss.item()

  avg_loss = epoch_loss / len(dataloader)
  print(f"Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.6f}")

# Testing
```

```
model.eval()

dataset_test = TimeSeriesDataset(test, input_window=24, output_window=4)

full_batch_loader = DataLoader(dataset_test, batch_size=len(dataset_test),
shuffle=False)


for x_full, y_full in full_batch_loader:

    with torch.no_grad():

        predictions = model(x_full)


# Calculate total test loss

total_loss = criterion(predictions, y_full)

print(f"\nTotal Test Loss (MSE): {total_loss:.6f}")


# Calculate individual losses for each of the 4 output steps

for i in range(4):

    step_loss = criterion(predictions[:,i], y_full[:,i])

    print(f"Test Loss for {i+1} hour(s) ahead: {step_loss:.6f}")


# Convert predictions and targets back to numpy (and denormalize for plotting if
needed)

Predictions = predictions.cpu().numpy() * train_std + train_mean

Targets = y_full.cpu().numpy() * train_std + train_mean


# Plot true vs predicted for each forecast step

for i in range(4):

    plt.figure(figsize=(10, 4))

    plt.plot(Targets[:,i], label="True Values")
```

```
plt.plot(Predictions[:,i], label="Predicted Values")
plt.title(f"SO₂ Concentration Prediction {i+1} hour(s) ahead")
plt.xlabel("Sample Index")
plt.ylabel("SO₂ Concentration")
plt.legend()
plt.grid(True)
plt.show()
```

Console 1/A

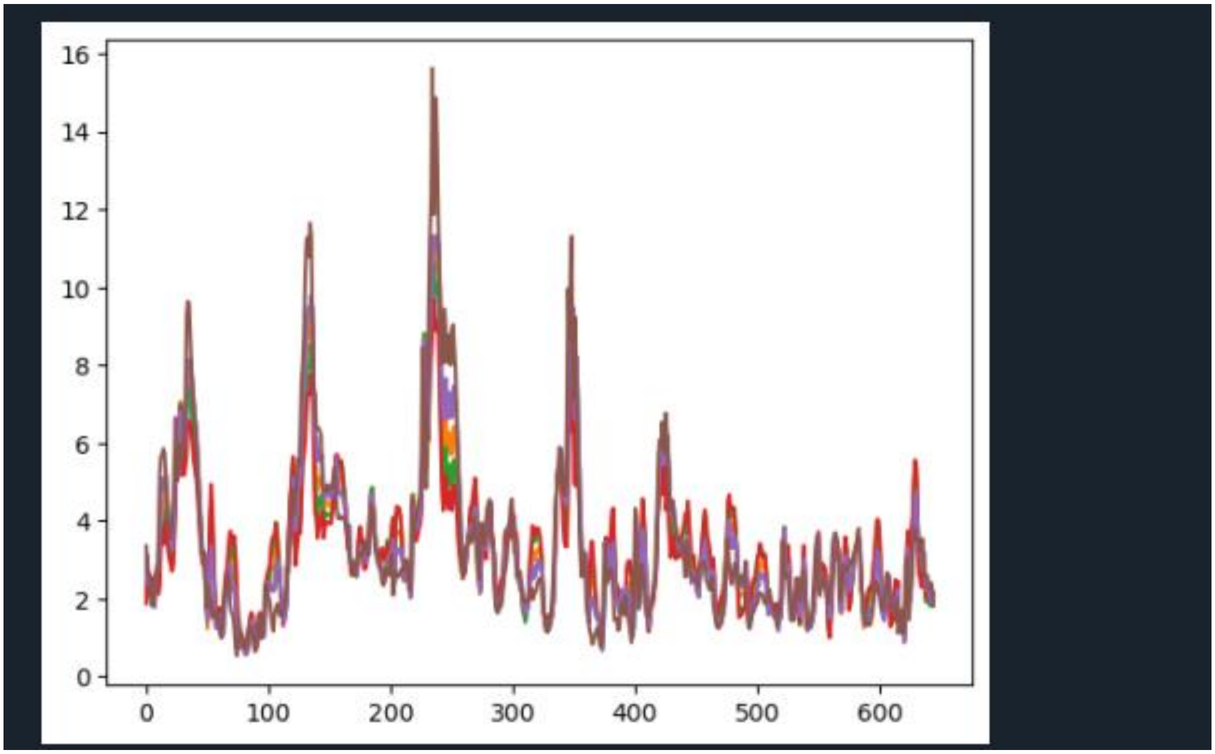
```
Epoch 142/150, Loss: 0.017895
Epoch 143/150, Loss: 0.019308
Epoch 144/150, Loss: 0.021125
Epoch 145/150, Loss: 0.021733
Epoch 146/150, Loss: 0.018767
Epoch 147/150, Loss: 0.018976
Epoch 148/150, Loss: 0.017991
Epoch 149/150, Loss: 0.018577
Epoch 150/150, Loss: 0.020029

Total Test Loss (MSE): 0.692901
Test Loss for 1 hour(s) ahead: 0.287682
Test Loss for 2 hour(s) ahead: 0.575706
Test Loss for 3 hour(s) ahead: 0.827717
Test Loss for 4 hour(s) ahead: 1.080501
```

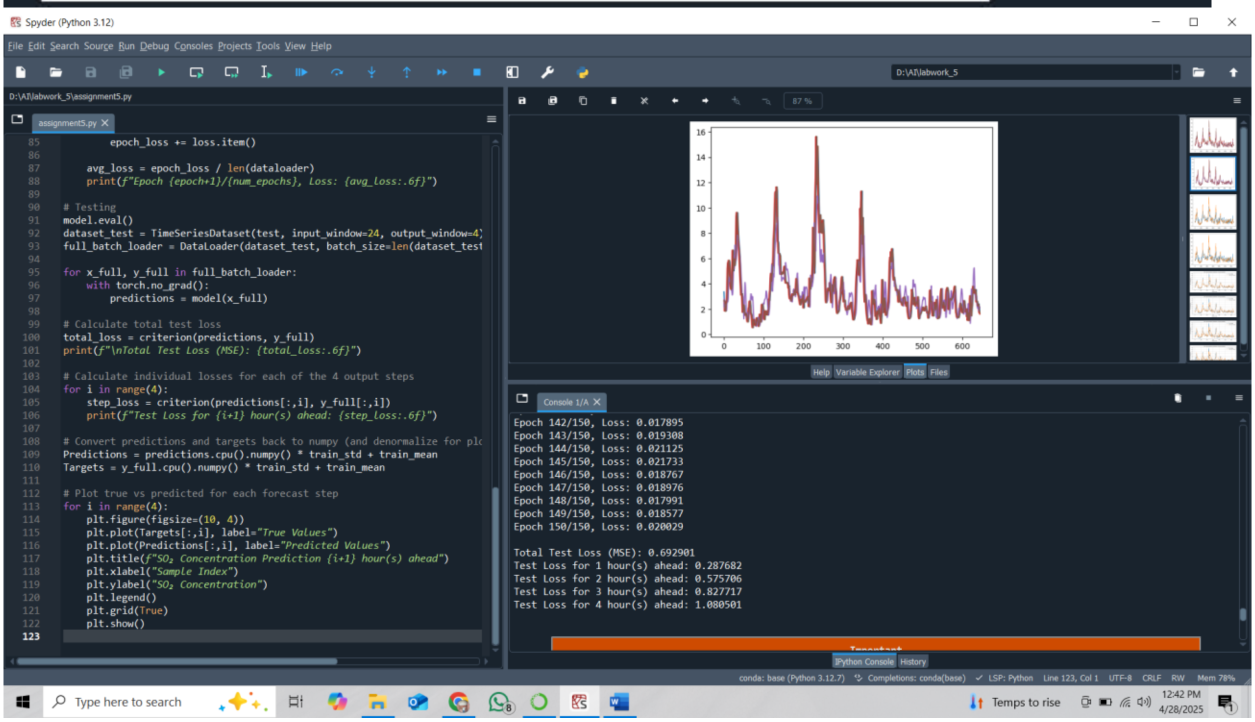
Name	Type	Size	Value
avg_loss	float	1	0.020028527670850355
dataloader	utils.data.dataloader.DataLoader	12	DataLoader object of torch.utils.data.dataloader module
dataset	TimeSeriesDataset	717	TimeSeriesDataset object of __main__ module
dataset_test	TimeSeriesDataset	645	TimeSeriesDataset object of __main__ module
epoch	int	1	149
epoch_loss	float	1	0.24034233205020428
full_batch_loader	utils.data.dataloader.DataLoader	1	DataLoader object of torch.utils.data.dataloader module
i	int	1	3
learning_rate	float	1	0.001
loss	Tensor	1	Tensor object of torch module
num_epochs	int	1	150
optimizer	optim.adam.Adam	1	Adam object of torch.optim.adam module
predictions	Tensor	(645, 4)	Tensor object of torch module
Predictions	Array of float32	(645, 4)	[[3.3887262 3.4917698 3.2151942 2.752795] [2.957333 2.4283543 2.040 ...
step_loss	Tensor	1	Tensor object of torch module
Targets	Array of float32	(645, 4)	[[3.3564901 2.653337 2.6919339 2.5543158] [2.653337 2.6919339 2.554 ...
test	Array of float32	(672,)	[-0.527278 -0.47309926 -0.70915896 ... -0.7410652 -0.7592201 -0.79 ...

Help Variable Explorer Plots Files

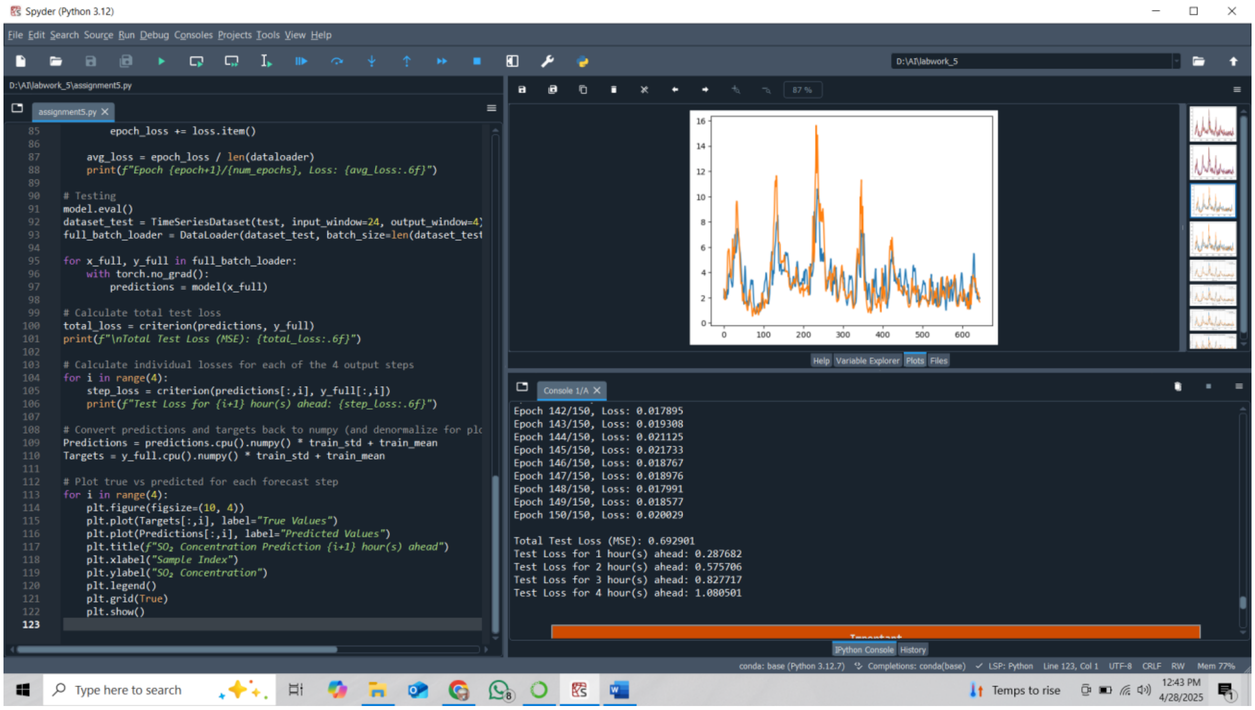
Name	Type	Size	Value
learning_rate	float	1	0.001
loss	Tensor	1	Tensor object of torch module
num_epochs	int	1	150
optimizer	optim.adam.Adam	1	Adam object of torch.optim.adam module
predictions	Tensor	(645, 4)	Tensor object of torch module
Predictions	Array of float32	(645, 4)	[[3.3887262 3.4917698 3.2151942 2.752795] [2.957333 2.4283543 2.040 ...
step_loss	Tensor	1	Tensor object of torch module
Targets	Array of float32	(645, 4)	[[3.3564901 2.653337 2.6919339 2.5543158] [2.653337 2.6919339 2.554 ...
test	Array of float32	(672,)	[-0.527278 -0.47309926 -0.70915896 ... -0.7410652 -0.7592201 -0.79 ...
total_loss	Tensor	1	Tensor object of torch module
train	Array of float32	(744,)	[-0.35445136 -0.5467527 -0.45554823 ... 0.7760028 -0.25899518 -0.3 ...
train_mean	float32	1	3.0607738
train_std	float32	1	1.8394786
x_batch	Tensor	(13, 24, 1)	Tensor object of torch module
x_full	Tensor	(645, 24, 1)	Tensor object of torch module
y_batch	Tensor	(13, 4)	Tensor object of torch module
y_full	Tensor	(645, 4)	Tensor object of torch module



0
2
4
6
8
10
12
14
16
0
100
200
300
400
500
600



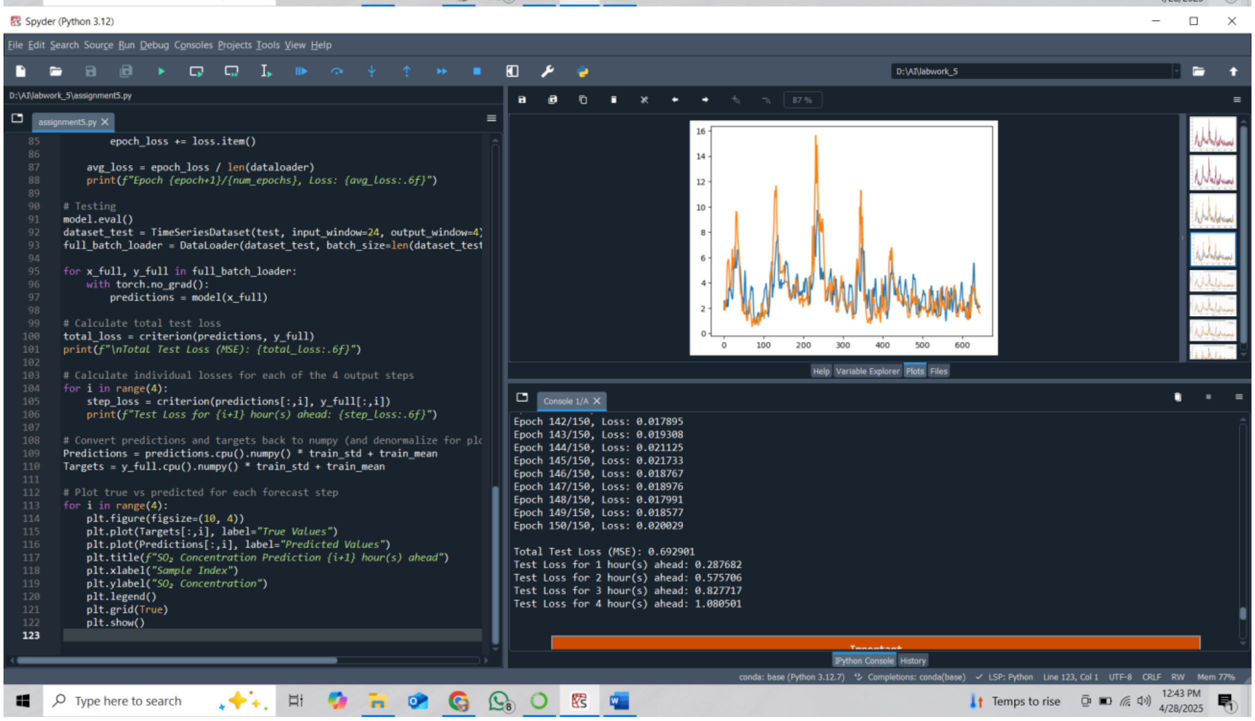
Spyder (Python 3.12)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\AI\labwork_5
D:\AI\labwork_5\assignment5.py
assignment5.py
epoch_loss += loss.item()
avg_loss = epoch_loss / len(dataloader)
print(f"Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.6f}")
# Testing
model.eval()
dataset_test = TimeSeriesDataset(test, input_window=24, output_window=4)
full_batch_loader = DataLoader(dataset_test, batch_size=len(dataset_test
for x_full, y_full in full_batch_loader:
with torch.no_grad():
predictions = model(x_full)
# Calculate total test loss
total_loss = criterion(predictions, y_full)
print(f"\nTotal Test Loss (MSE): {total_loss:.6f}")
# Calculate individual losses for each of the 4 output steps
for i in range(4):
step_loss = criterion(predictions[:,i], y_full[:,i])
print(f"Test Loss for {i+1} hour(s) ahead: {step_loss:.6f}")
# Convert predictions and targets back to numpy (and denormalize for plo
Predictions = predictions.cpu().numpy() * train_std + train_mean
Targets = y_full.cpu().numpy() * train_std + train_mean
# Plot true vs predicted for each forecast step
for i in range(4):
plt.figure(figsize=(10, 4))
plt.plot(Targets[:,i], label="True Values")
plt.plot(Predictions[:,i], label="Predicted Values")
plt.title(f"SO₂ Concentration Prediction {i+1} hour(s) ahead")
plt.xlabel("Sample Index")
plt.ylabel("SO₂ Concentration")
plt.legend()
plt.grid(True)
plt.show()
Help Variable Explorer Plots Files
Console 1/A
Epoch 142/150, Loss: 0.017895
Epoch 143/150, Loss: 0.019308
Epoch 144/150, Loss: 0.021125
Epoch 145/150, Loss: 0.021733
Epoch 146/150, Loss: 0.018767
Epoch 147/150, Loss: 0.018976
Epoch 148/150, Loss: 0.017991
Epoch 149/150, Loss: 0.018577
Epoch 150/150, Loss: 0.020029
Total Test Loss (MSE): 0.692901
Test Loss for 1 hour(s) ahead: 0.287682
Test Loss for 2 hour(s) ahead: 0.575706
Test Loss for 3 hour(s) ahead: 0.827717
Test Loss for 4 hour(s) ahead: 1.080501
IPython Console History
conda: base (Python 3.12.7)
Type here to search
Temps to rise



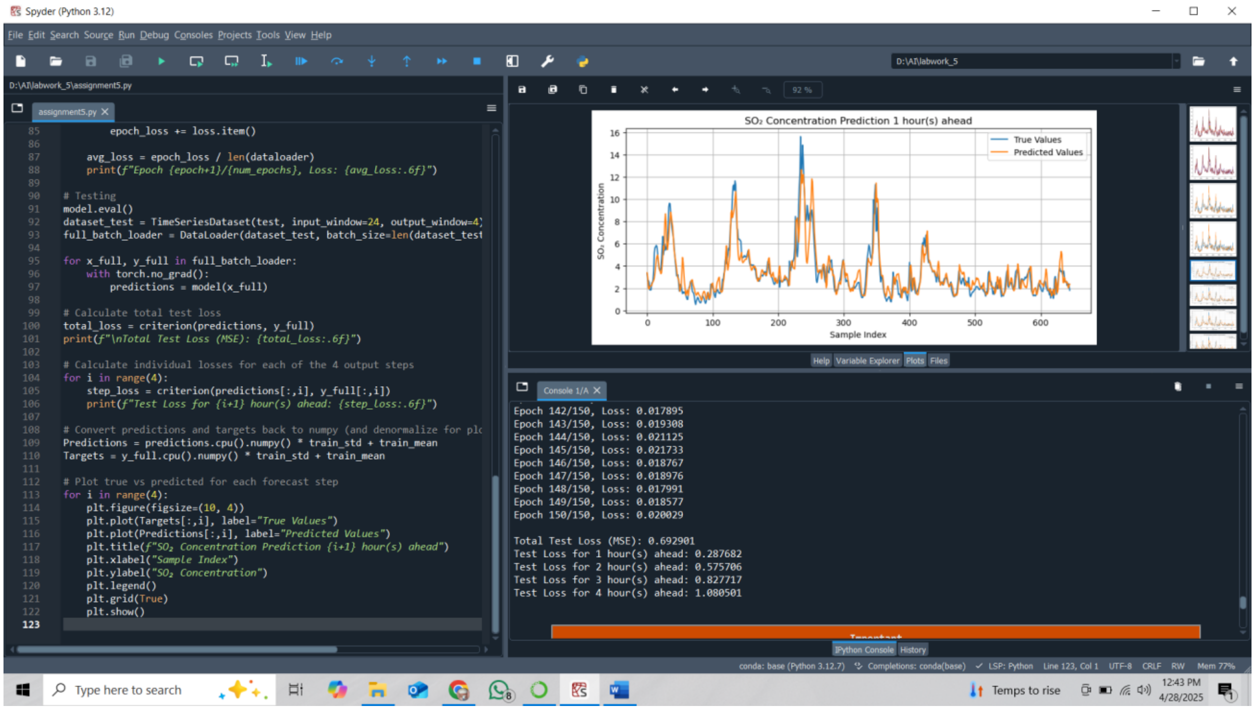
Spyder (Python 3.12)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\AI\labwork_5
D:\AI\labwork_5\assignment5.py
assignment5.py
        epoch_loss += loss.item()

    avg_loss = epoch_loss / len(dataloader)
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.6f}")

# Testing
model.eval()
dataset_test = TimeSeriesDataset(test, input_window=24, output_window=4)
full_batch_loader = DataLoader(dataset_test, batch_size=len(dataset_test
for x_full, y_full in full_batch_loader:
    with torch.no_grad():
        predictions = model(x_full)
# Calculate total test loss
total_loss = criterion(predictions, y_full)
print(f"\nTotal Test Loss (MSE): {total_loss:.6f}")
# Calculate individual losses for each of the 4 output steps
for i in range(4):
    step_loss = criterion(predictions[:,i], y_full[:,i])
    print(f"Test Loss for {i+1} hour(s) ahead: {step_loss:.6f}")
# Convert predictions and targets back to numpy (and denormalize for plo
Predictions = predictions.cpu().numpy() * train_std + train_mean
Targets = y_full.cpu().numpy() * train_std + train_mean
# Plot true vs predicted for each forecast step
for i in range(4):
    plt.figure(figsize=(10, 4))
    plt.plot(Targets[:,i], label="True Values")
    plt.plot(Predictions[:,i], label="Predicted Values")
    plt.title(f"SO₂ Concentration Prediction {i+1} hour(s) ahead")
    plt.xlabel("Sample Index")
    plt.ylabel("SO₂ Concentration")
    plt.legend()
    plt.grid(True)
    plt.show()
Help Variable Explorer Plots Files
Console 1/A
Epoch 142/150, Loss: 0.017895
Epoch 143/150, Loss: 0.019308
Epoch 144/150, Loss: 0.021125
Epoch 145/150, Loss: 0.021733
Epoch 146/150, Loss: 0.018767
Epoch 147/150, Loss: 0.018976
Epoch 148/150, Loss: 0.017991
Epoch 149/150, Loss: 0.018577
Epoch 150/150, Loss: 0.020029
Total Test Loss (MSE): 0.692901
Test Loss for 1 hour(s) ahead: 0.287682
Test Loss for 2 hour(s) ahead: 0.575706
Test Loss for 3 hour(s) ahead: 0.827717
Test Loss for 4 hour(s) ahead: 1.080501
IPython Console History
conda: base (Python 3.12.7) Completions: conda(base) LSP: Python Line 123, Col 1 UTF-8 CRLF RW Mem 77%
Type here to search
Temps to rise
12:43 PM
4/28/2025

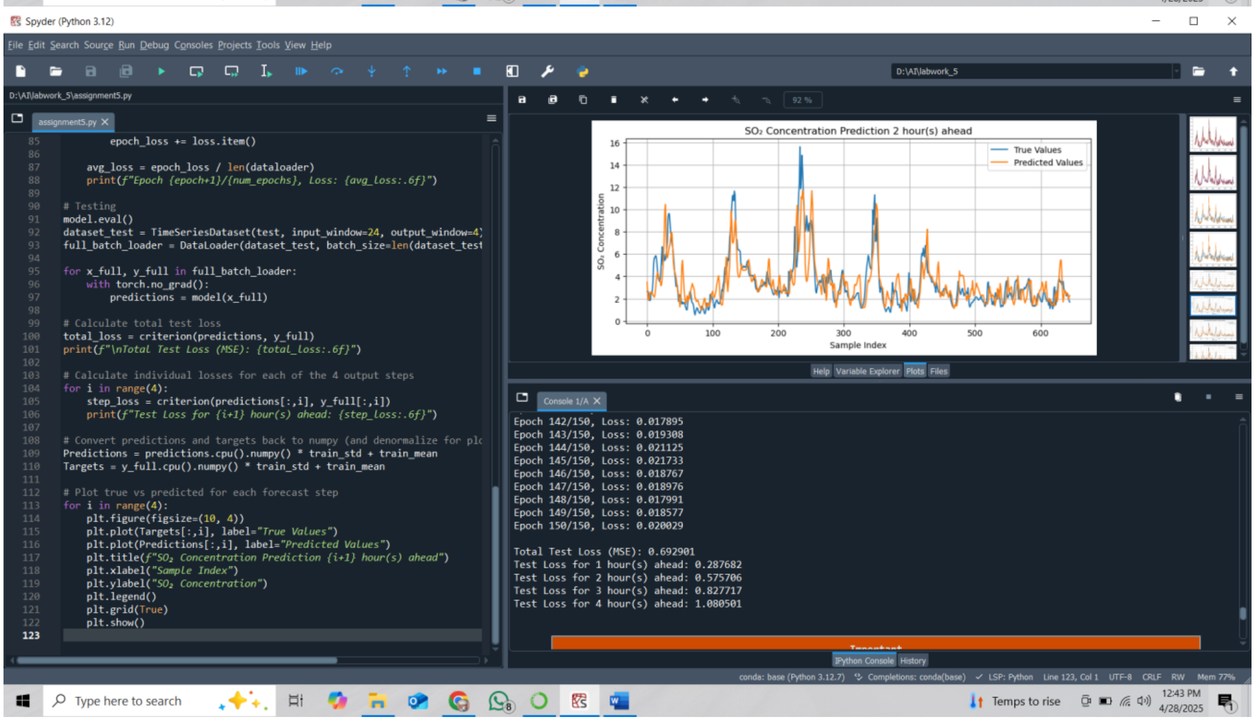


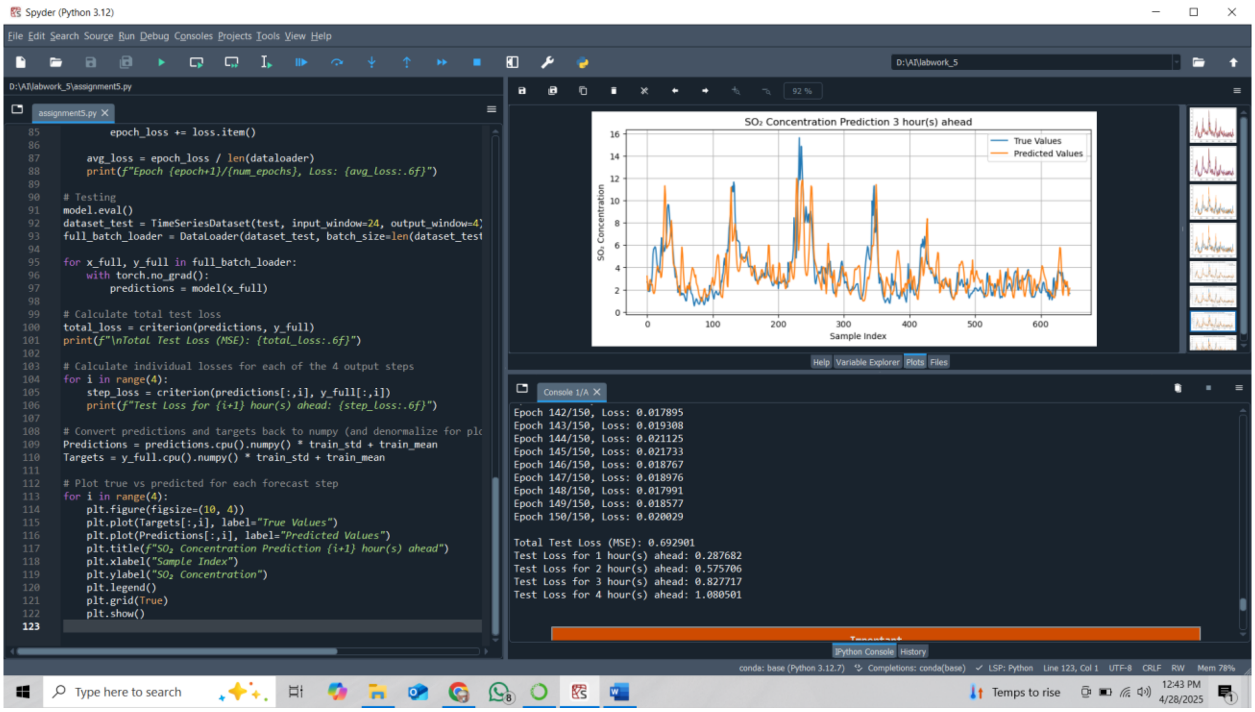
Spyder (Python 3.12)
Console 1/A
Total Test Loss (MSE): 0.692901
Test Loss for 1 hour(s) ahead: 0.287682
Test Loss for 2 hour(s) ahead: 0.575706
Test Loss for 3 hour(s) ahead: 0.827717
Test Loss for 4 hour(s) ahead: 1.080501
12:43 PM
4/28/2025

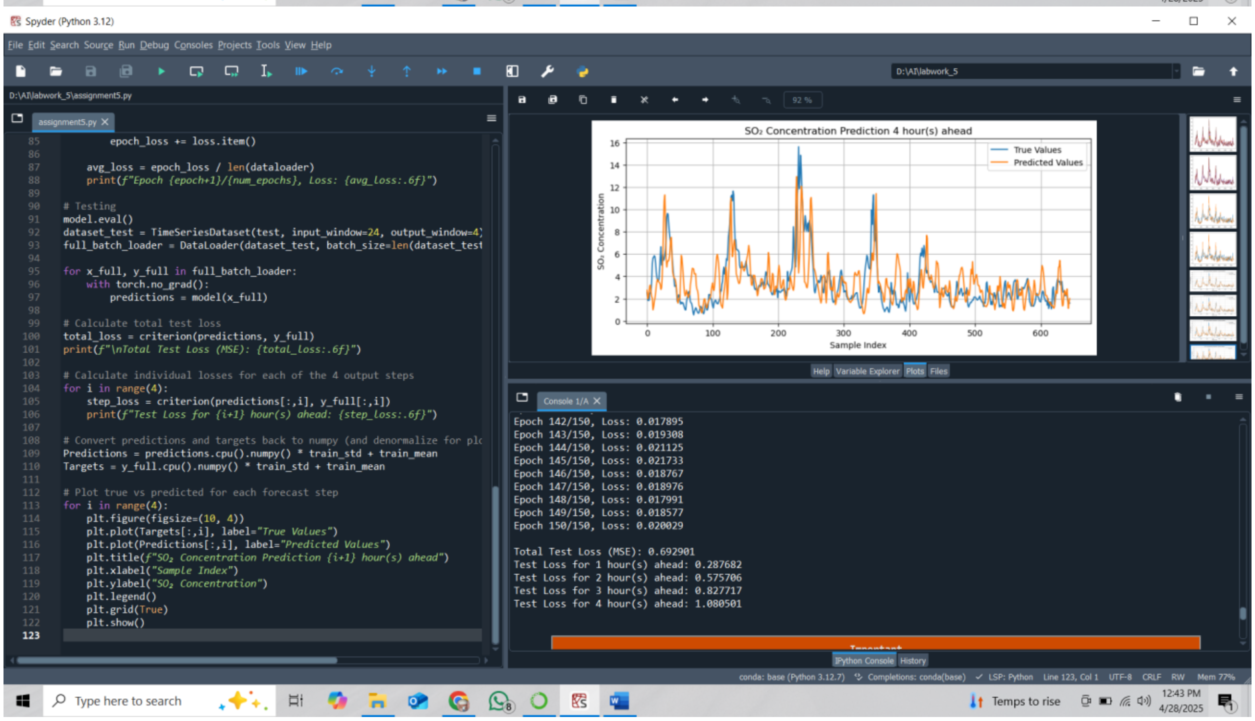

Spyder (Python 3.12)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\AI\labwork_5\assignment5.py
assignment5.py
        epoch_loss += loss.item()

    avg_loss = epoch_loss / len(dataloader)
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.6f}")

# Testing
model.eval()
dataset_test = TimeSeriesDataset(test, input_window=24, output_window=4)
full_batch_loader = DataLoader(dataset_test, batch_size=len(dataset_test
for x_full, y_full in full_batch_loader:
    with torch.no_grad():
        predictions = model(x_full)
# Calculate total test loss
total_loss = criterion(predictions, y_full)
print(f"\nTotal Test Loss (MSE): {total_loss:.6f}")
# Calculate individual losses for each of the 4 output steps
for i in range(4):
    step_loss = criterion(predictions[:,i], y_full[:,i])
    print(f"Test Loss for {i+1} hour(s) ahead: {step_loss:.6f}")
# Convert predictions and targets back to numpy (and denormalize for plo
Predictions = predictions.cpu().numpy() * train_std + train_mean
Targets = y_full.cpu().numpy() * train_std + train_mean
# Plot true vs predicted for each forecast step
for i in range(4):
    plt.figure(figsize=(10, 4))
    plt.plot(Targets[:,i], label="True Values")
    plt.plot(Predictions[:,i], label="Predicted Values")
    plt.title(f"SO₂ Concentration Prediction {i+1} hour(s) ahead")
    plt.xlabel("Sample Index")
    plt.ylabel("SO₂ Concentration")
    plt.legend()
    plt.grid(True)
    plt.show()
SO₂ Concentration Prediction 1 hour(s) ahead
True Values
Predicted Values
SO₂ Concentration
Sample Index
Console 1/A
Epoch 142/150, Loss: 0.017895
Epoch 143/150, Loss: 0.019308
Epoch 144/150, Loss: 0.021125
Epoch 145/150, Loss: 0.021733
Epoch 146/150, Loss: 0.018767
Epoch 147/150, Loss: 0.018976
Epoch 148/150, Loss: 0.017991
Epoch 149/150, Loss: 0.018577
Epoch 150/150, Loss: 0.020029
Total Test Loss (MSE): 0.692901
Test Loss for 1 hour(s) ahead: 0.287682
Test Loss for 2 hour(s) ahead: 0.575706
Test Loss for 3 hour(s) ahead: 0.827717
Test Loss for 4 hour(s) ahead: 1.080501
Type here to search
Temps to rise
12:43 PM
4/28/2025


Spyder (Python 3.12)
D:\AI\labwork_5\assignment5.py
SO₂ Concentration Prediction 2 hour(s) ahead
True Values
Predicted Values
SO₂ Concentration
Sample Index
Total Test Loss (MSE): 0.692901
Test Loss for 1 hour(s) ahead: 0.287682
Test Loss for 2 hour(s) ahead: 0.575706
Test Loss for 3 hour(s) ahead: 0.827717
Test Loss for 4 hour(s) ahead: 1.080501
12:43 PM
4/28/2025





End of Lab Report