**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 SO₂ (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 SO₂ 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 SO₂ 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()**

A screenshot of a computer

AI-generated content may be incorrect.**A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.**

A graph of a graph

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**End of Lab Report**