Lab 8: Experiment Using LSTM

Aim: To implement a Long Short-Term Memory (LSTM) neural network using PyTorch on an existing dataset and analyze its performance through traing metrics.

Description:

LSTM is a type of Recurrent Neural Network designed to model sequential data. while overcoming the vanishing and exploding gradient problems commonly encountered in standard RNNs.

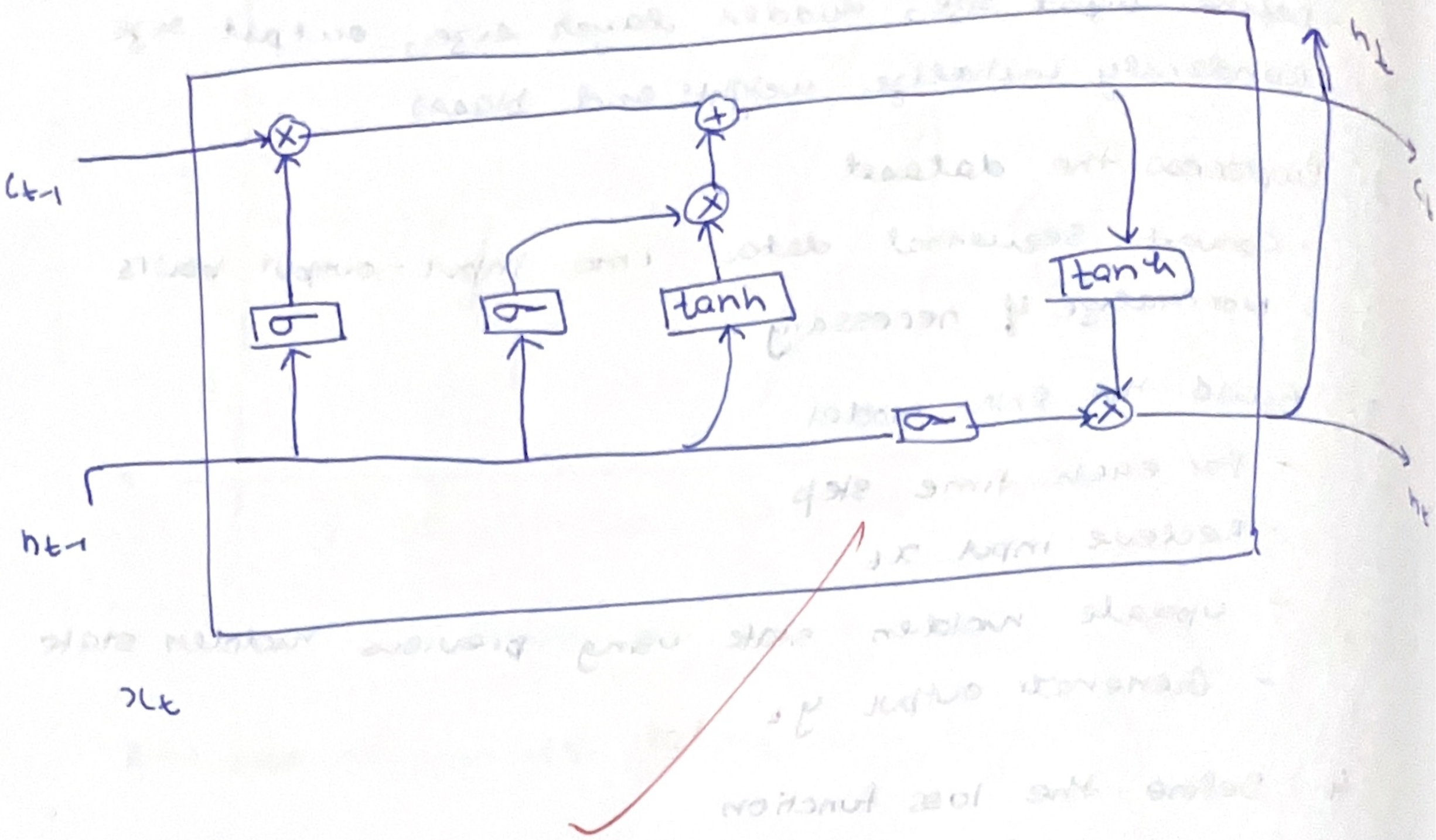
LSTMs are capable of learning long-term dependecies using three main gates:

Input gate - controls the extent to which new information

Forget gate - determines how much past information should be forgotten

Output gate - decides what information to output from the current cell state.

At each time step t, the LSTM processes an input vector x_t and a hidden state h_{t-1}, producing a new hidden state h_t and cell state C_t. This allows it to capture temporal dependencies in sequential data such as text, time series or speech.


h_t
C_{t-1}
C_t
σ
σ
tanh
tanh
σ
h_{t-1}
x_t

Pseudocode:

1. Import required libraries
2. Load dataset and split into train and test data
3. Tokenize text data and convert to numerical sequences.
4. Define an LSTM-based model:
   - Embedding layer
   - LSTM layer
   - Fully connected output layer
5. Define loss function and optimizer
6. Train the model:
   - Forward pass through LSTM
   - Compute loss
   - Back propogate gradients
   - Update parameters
   - Record training loss for each epoch
7. Evaluate on test data

~~8. Plot training loss~~

Result:

The LSTM network was successfully implemented using PyTorch. The models training loss reduced consistenty, indicating that it learned to predict future sine values accurately.

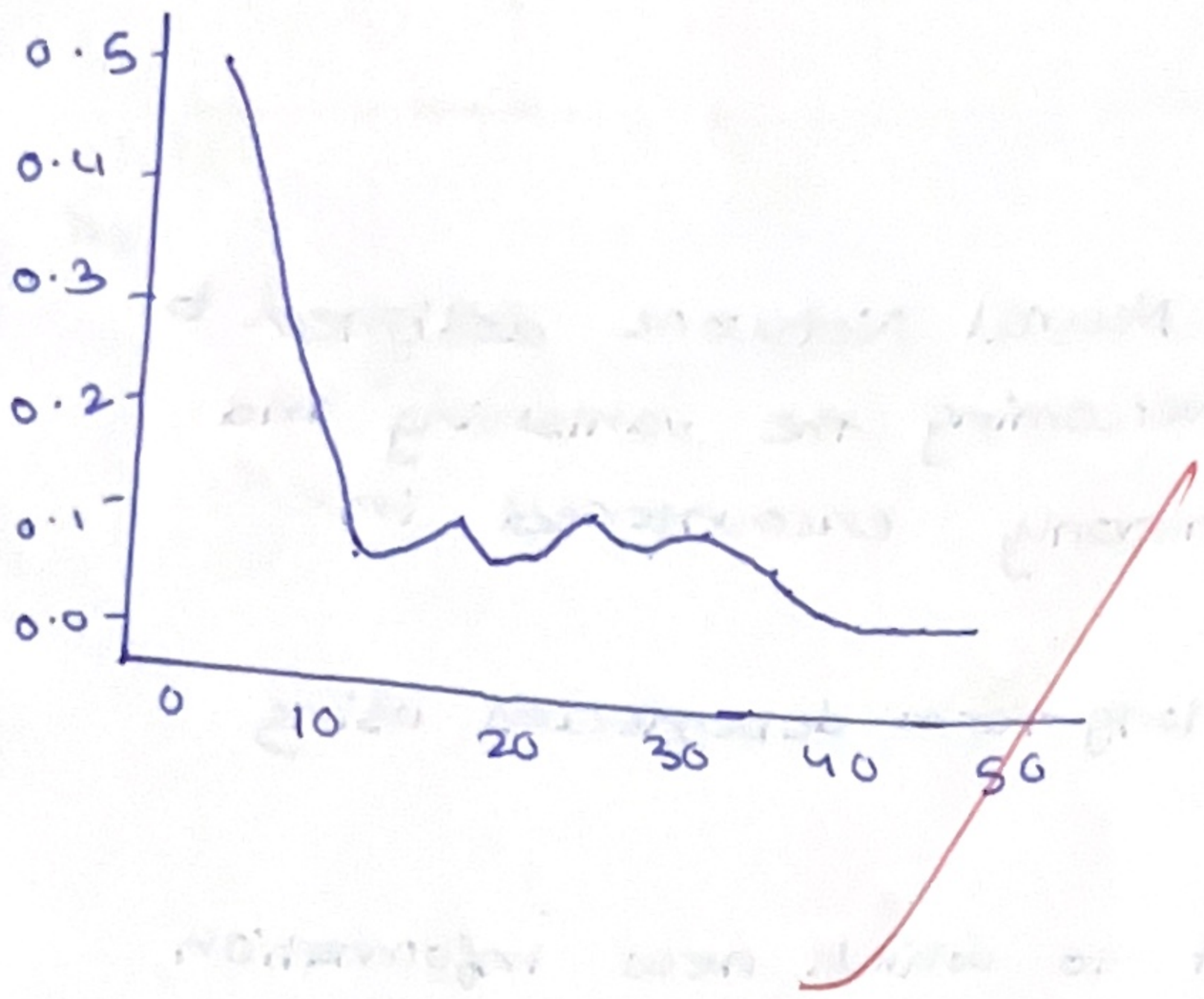
Epoch [10/50], Loss: 0.0886

Epoch [20/50], Loss: 0.0055

Epoch [30/50], Loss: 0.0008

Epoch [40/50], Loss: 0.0003

Epoch [50/50], Loss: 0.0003



File Edit Selection View Go Run ··· CODE

lab8.ipynb | lab11.ipynb | lab12.ipynb | lab14.ipynb | lab15.ipynb

.vscode > PPS > lab8.ipynb > import torch

Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ··· Python 3.13.1

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

# Generate sine wave data
x = np.linspace(0, 100, 1000)
y = np.sin(x)

# Prepare sequences
def create_seq(data, seq_len):
    xs, ys = [], []
    for i in range(len(data)-seq_len):
        xs.append(data[i:i+seq_len])
        ys.append(data[i+seq_len])
    return np.array(xs), np.array(ys)

seq_len = 50
X, Y = create_seq(y, seq_len)
X = torch.tensor(X).unsqueeze(-1).float()
Y = torch.tensor(Y).unsqueeze(-1).float()

# LSTM Model
class LSTM(nn.Module):
    def __init__(self, input_size=1, hidden_size=50, num_layers=1, output_size=1):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = self.fc(out[:, -1, :])
        return out

model = LSTM()
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

# Training loop
epochs = 50
losses = []
for epoch in range(epochs):
    optimizer.zero_grad()
    output = model(X)
    loss = criterion(output, Y)
    loss.backward()
    optimizer.step()
    losses.append(loss.item())
    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# Plot training loss
plt.plot(losses)
plt.title('Training Loss Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.show()
```

[3]

Spaces: 4 LF {} Cell 1 of 2 Go Live

ENG IN 23:59 03-11-2025

```
Epoch [10/50], Loss: 0.0367
Epoch [20/50], Loss: 0.0053
Epoch [30/50], Loss: 0.0017
Epoch [40/50], Loss: 0.0006
Epoch [50/50], Loss: 0.0003
```