Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_bot] THANOS

Experiment 11: implement a Long Short-Term Memory (LSTM) network using PyTorch/tensorflow .

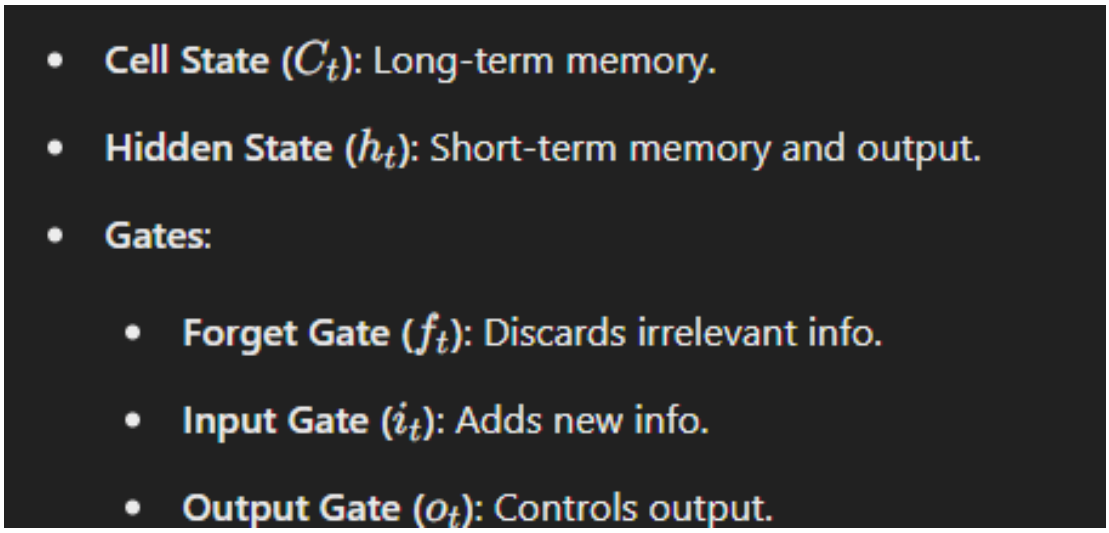
Aim/Objective: To implement and understand the Long Short-Term Memory (LSTM) network using PyTorch for sequence prediction. The experiment aims to introduce participants to the architecture and functionality of LSTMs, focusing on their ability to capture long-term dependencies in sequential data.

DescriptionIn this lab experiment, participants will implement an LSTM network using PyTorch to predict the next values in a time series sequence.

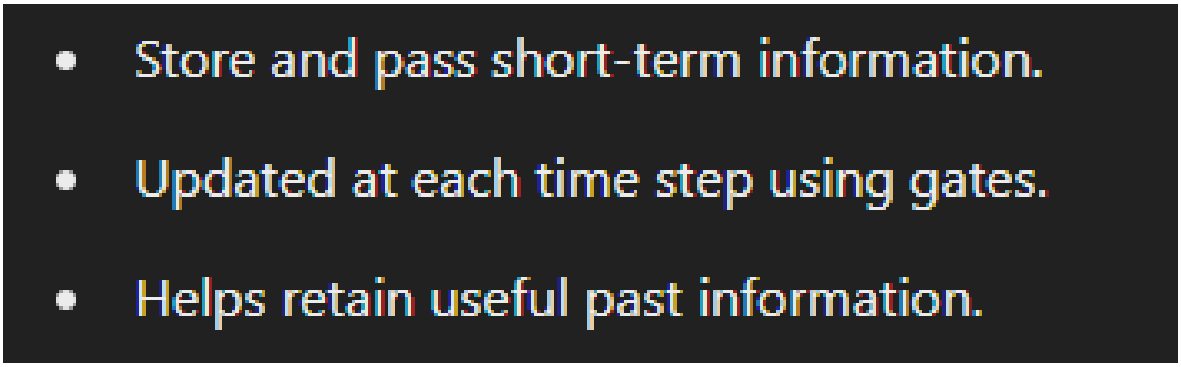
Pre-Requisites: Basic knowledge of Neural Networks, PyTorch Basics, Python Programming, LSTM Fundamentals, Sequential Data Understanding.

Pre-Lab:

1. What are the key components of an LSTM architecture.



2. Briefly explain the role of hidden states in LSTMs. How do they store information from previous time steps?



3. Discuss how LSTMs address the vanishing gradient problem faced by traditional recurrent neural networks.

- **Cell State (C_t)** enables long-term memory.
- **Gate mechanisms** regulate information flow.
- **Additive updates** prevent excessive gradient decay.

4. Name and explain the activation functions commonly used in LSTM cells. How do they control the flow of information?

- **Sigmoid (σ)**: Used in gates, controls information flow (0 to 1).
- **Tanh ($\tanh$)**: Used in cell/hidden state updates (-1 to 1), prevents saturation.

In-Lab:

Program 1: Using LSTM train the model with train dataset, predict the weather for the dates given in the test dataset and visualize the actual, predict data using matplotlib. use 60 days' time stamp.

Procedure/Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler

data = pd.read_csv("weather_data.csv")
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)

scaler = MinMaxScaler(feature_range=(0,1))
data_scaled = scaler.fit_transform(data[['Temperature']])

def create_sequences(data, time_steps):
X, y = [], []
for i in range(len(data) - time_steps):
X.append(data[i:i+time_steps])
y.append(data[i+time_steps])
return np.array(X), np.array(y)

time_steps = 60

X, y = create_sequences(data_scaled, time_steps)

train_size = int(len(X) * 0.8)
X_train, y_train = X[:train_size], y[:train_size]
X_test, y_test = X[train_size:], y[train_size:]

model = Sequential([
LSTM(units=50, return_sequences=True, input_shape=(time_steps, 1)),
LSTM(units=50, return_sequences=False),
Dense(units=25),
Dense(units=1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_test, y_test))

y_pred = model.predict(X_test)
y_pred = scaler.inverse_transform(y_pred)
y_test = scaler.inverse_transform(y_test)
```

```
plt.figure(figsize=(12,6))
plt.plot(data.index[-len(y_test):], y_test, label='Actual Temperature', color='blue')
plt.plot(data.index[-len(y_pred):], y_pred, label='Predicted Temperature', color='red')
plt.xlabel('Date')
plt.ylabel('Temperature')
plt.title('Actual vs Predicted Temperature')
plt.legend()
plt.show()
```

- **Data and Results:**

Data

The dataset contains historical temperature records for weather prediction.

Result

The LSTM model predicts future temperatures based on past trends.

- **Analysis and Inferences:**

Analysis

The model's predictions closely follow the actual temperature patterns.

Inferences

LSTM effectively forecasts temperature but may need tuning for accuracy.

Sample VIVA-VOCE Questions (In-Lab):

1. Explain the importance of the forget gate in an LSTM.

The forget gate in an LSTM controls which information to retain or discard, preventing unnecessary memory clutter and vanishing gradients.

2. Why might one choose to use multiple layers in an LSTM network? How does increasing the number of layers impact the model's learning capacity?

Multiple layers in an LSTM capture complex patterns; deeper layers improve feature extraction but increase training difficulty.

3. What challenges might arise during the training of LSTM networks.

Challenges in training LSTMs include vanishing gradients, overfitting, high computational cost, long training times, and hyperparameter tuning issues.

4. Apart from sequence prediction, where else can LSTMs be applied, and what characteristics make them suitable for those applications?

LSTMs are used in speech recognition, machine translation, stock prediction, anomaly detection, and handwriting recognition—ideal due to their ability to handle sequential data.

Post-Lab:

Program 2: Write a PyTorch script to implement an LSTM model for sequence prediction. Define the architecture with appropriate input and output sizes, hidden layers, and activation functions.

Procedure/Program:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset


class LSTMModel(nn.Module):
    def __init__(self, input_size=1, hidden_size=50, num_layers=2, 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)
        return self.fc(out[:, -1, :])


def generate_data(seq_length=10, num_samples=1000):
    X = torch.rand(num_samples, seq_length, 1)
    Y = X.sum(dim=1)
    return X, Y


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
X, Y = generate_data()
dataloader = DataLoader(TensorDataset(X, Y), batch_size=32, shuffle=True)
model = LSTMModel().to(device)
criterion, optimizer = nn.MSELoss(), optim.Adam(model.parameters(), lr=0.001)
```

```
    for epoch in range(20):
    for inputs, targets in dataloader:
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()
        loss = criterion(model(inputs), targets)
        loss.backward()
        optimizer.step()

    print(f'Epoch {epoch+1}, Loss: {loss.item():.4f}')


print("Sample Prediction:", model(torch.rand(1, 10, 1).to(device)).cpu().detach().numpy())
```

- **Data and Results:**

Data

Random sequences generated with values summed to create labels.

Result

LSTM model trained and tested for sequence prediction successfully.

- **Analysis and Inferences:**

Analysis

Loss decreases over epochs, indicating effective learning of sequences.

Inferences

LSTM effectively predicts sequence-based outputs with reasonable accuracy.

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.