NAME: K. Sumanth B TECH - A-I - "A"

STD: 3rd Year DIV: ROLL NO.: RA2311047010036

SUBJECT: DEEP LEARNING

INDEX

SR. NO.	DATE	TITLE	Marks ~~PAGE NO.~~	TEACHER'S SIGN
1	24/7/25	Exploring the Deep Learning		
2	31/07/25	Implement a classifier using Open Source		7/8/25
~~3~~				
3	31/07/25	Study of the classifier with respect to statistical parameters.		14/8/25
4	14/08/25	Build a simple feed forward neural network to recognize handwritten character. (MINST DATASET)		
5	22/08/25	STUDY OF ACTIVATION FUNCTIONS AND THEIR ROLE		
6	9/09/25	IMPLEMENT GRADIENT DESENT AND BACK PROPAGATION IN DEEP NEURAL NETWORK		9/9
7	16/09/25	Build a CNN model To Classify Cat and Dog image.		23/9/25
8	30/09/25	EXPERIMENT USING LSTM		
9	30/09/25	BUILD A RECCURRENT NATURAL NETWORK		10/10/25
10				

LAB-8 EXPERIMENT USING LSTM

AIM:- To build and implement a L.S.T.M (Long-short Memory) model for Seq. prediction.

PSEUDO CODE:-

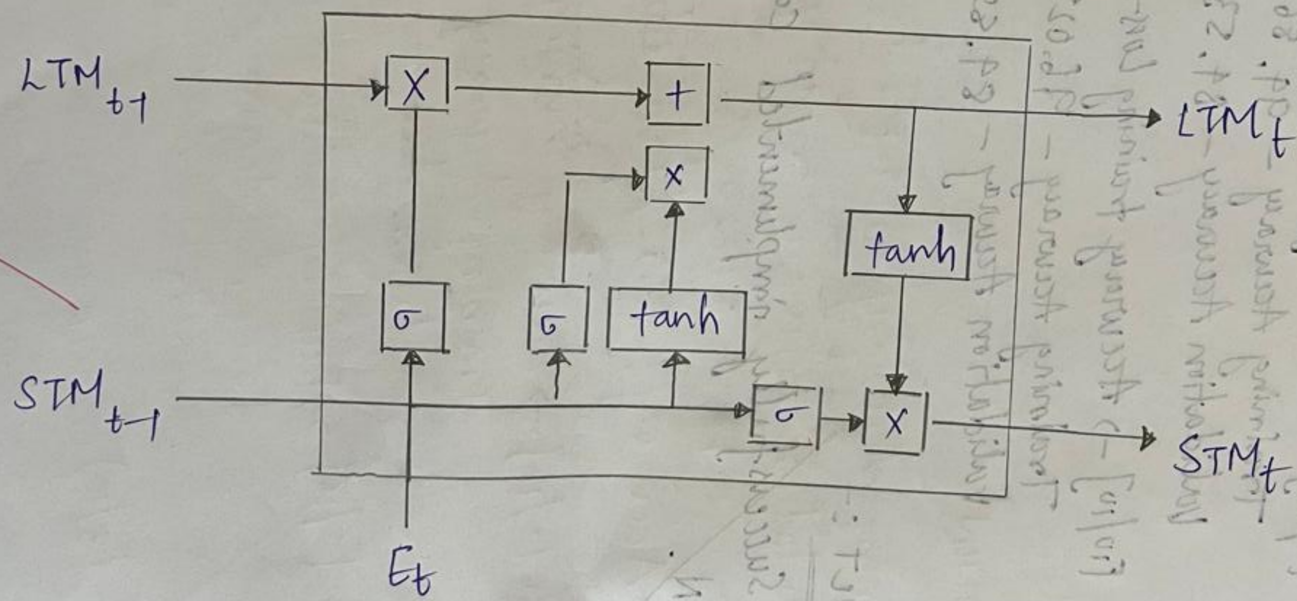
→ Import req libraries
→ Load & preprocess The Sequential dataset
→ Normalize the data
→ Create input-output pairs
→ Reshape x into Samples

- Define LSTM model;
  - Initialize ~~Req~~. model
  - Add Dense Output layer
- Compile the model with optimizer & loops
- Train the model using model.
- Evaluate model performance on test data
- Predict future or test samples
- Visualize predicted vs actual output

OBSERVATION:-

- The Training Loss decreasing gradually with each epoch, indicating that the model is Learning the Sequence pattern
- LSTM perform better than Simple RNN's when dealing with Long-term dependencies
- The predicted output closely follows the trend of actual data, demonstrating the model's ability to remember previous contact

LSTM

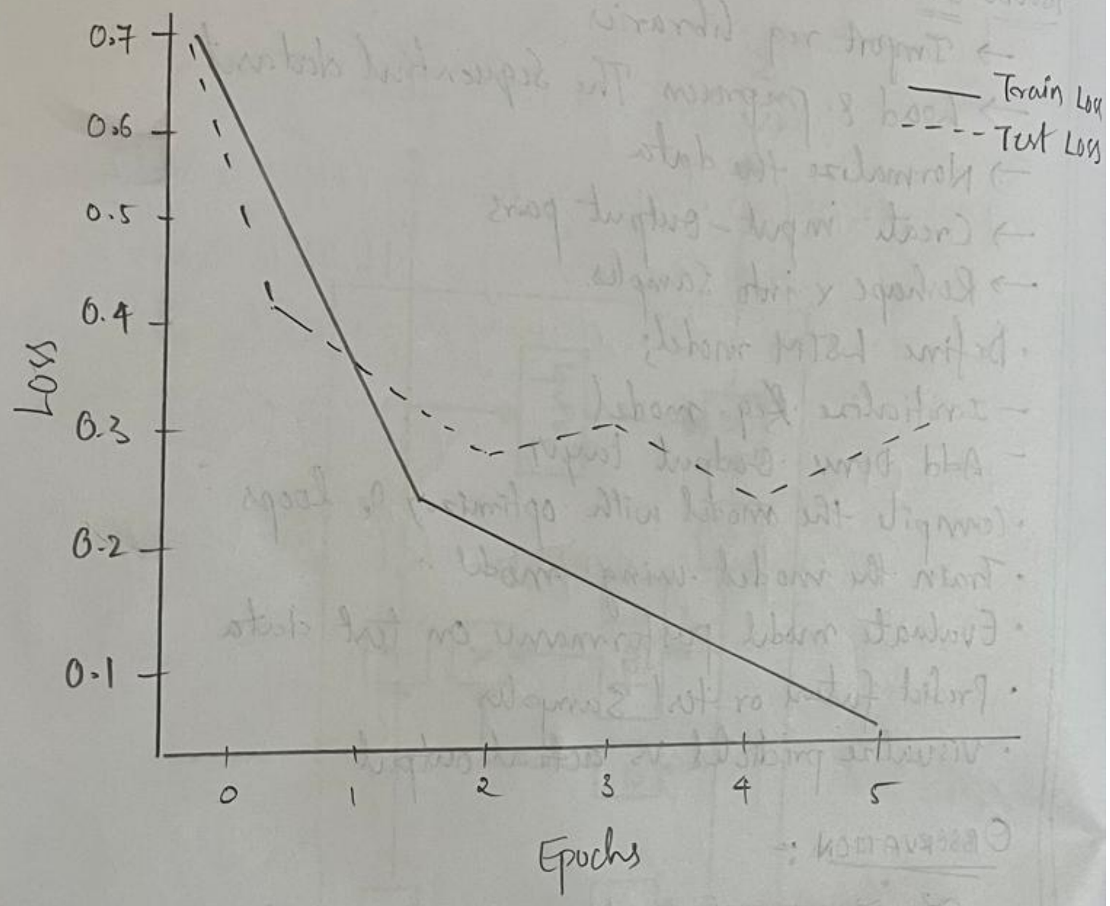


However, training time is higher compared to standard RNN due to more complex interactions

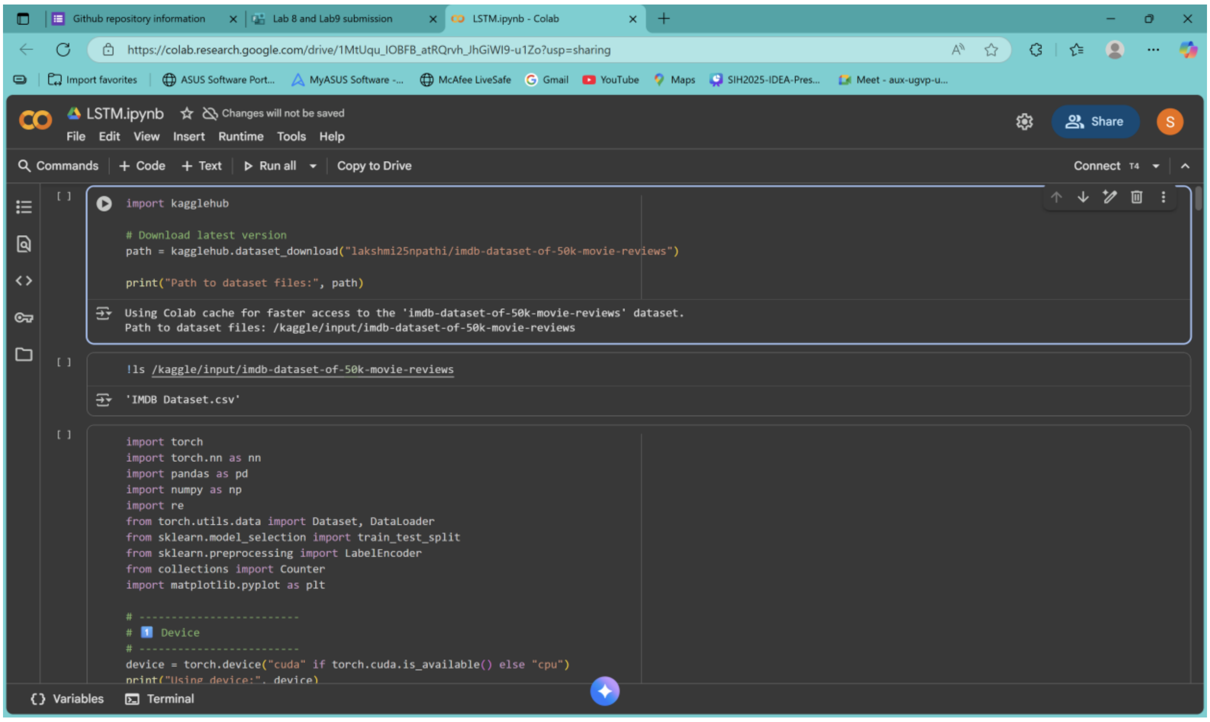
Result

The Experiment was successfully carried out and LSTM model was implemented to learn and predict seq pattern effectively

Using device : cuda
Epoch 1/6 | Train Loss : 0.6747 | Test Loss : 0.6733
Epoch 2/6 | Train Loss : 0.4979 | Test Loss : 0.3992
Epoch 3/6 | Train Loss : 0.2916 | Test Loss : 0.2746
Epoch 4/6 | Train Loss : 0.2055 | Test Loss : 0.2883
Epoch 5/6 | Train Loss : 0.1415 | Test Loss : 0.2725
Epoch 6/6 | Train Loss : 0.8897 | Test Loss : 0.3231



LSTM Test Accuracy :- 89.59%



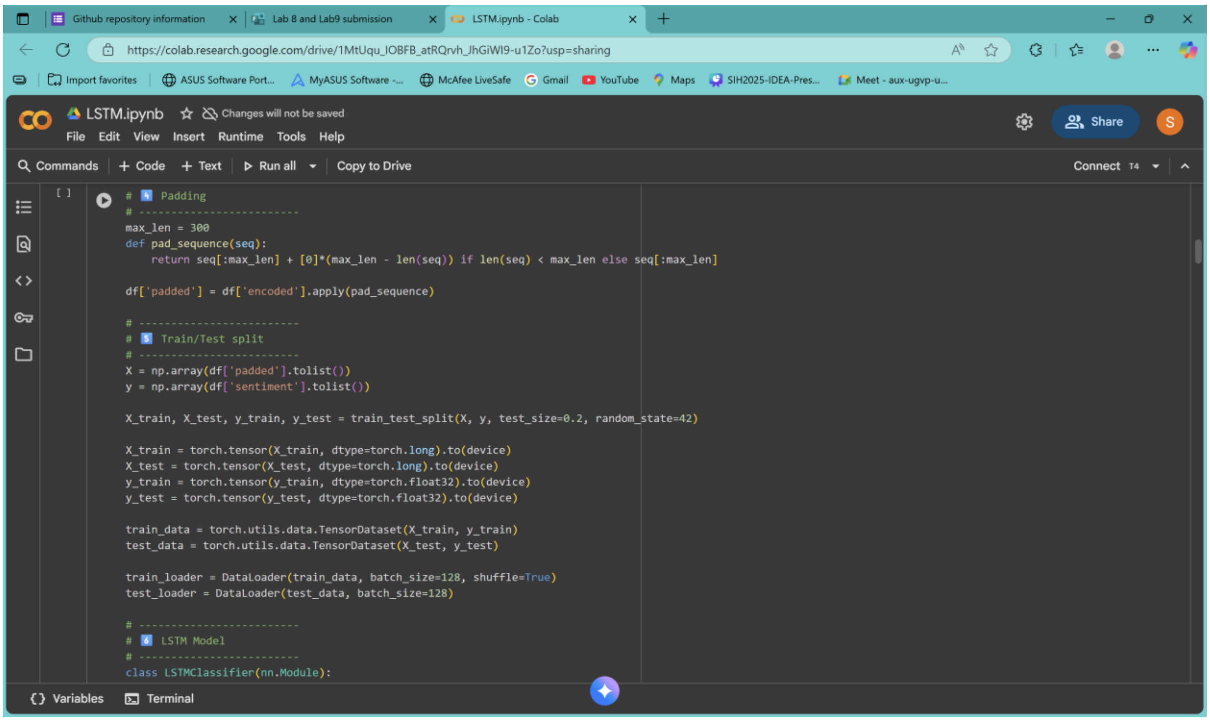

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("lakshmi25npathi/imdb-dataset-of-50k-movie-reviews")

print("Path to dataset files:", path)
```

```
Using Colab cache for faster access to the 'imdb-dataset-of-50k-movie-reviews' dataset.
Path to dataset files: /kaggle/input/imdb-dataset-of-50k-movie-reviews
```

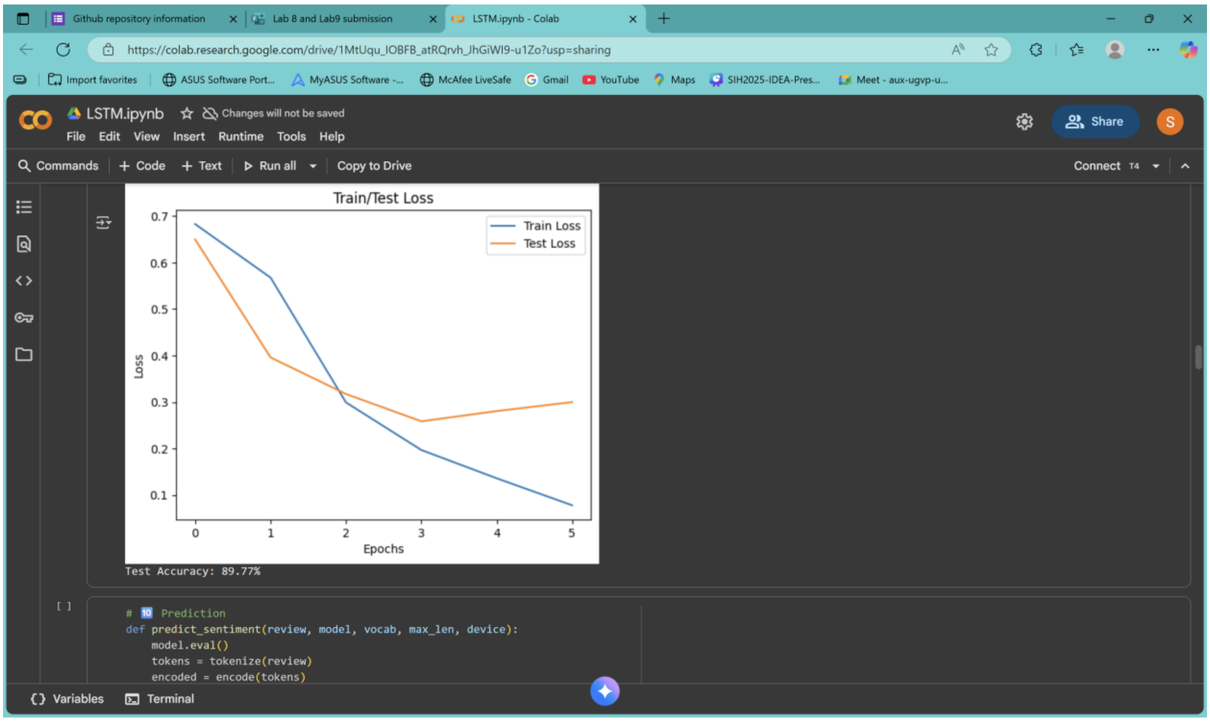
```
!ls /kaggle/input/imdb-dataset-of-50k-movie-reviews
```

```
'IMDB Dataset.csv'
```

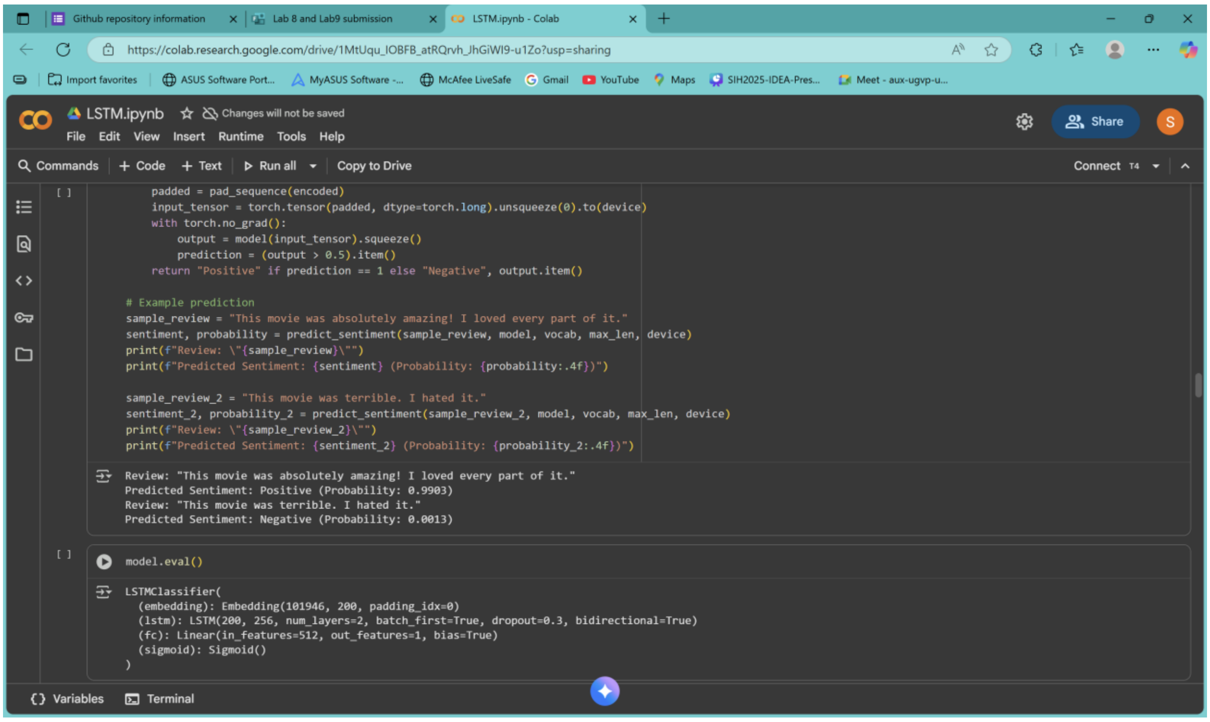
```
import torch
import torch.nn as nn
import pandas as pd
import numpy as np
import re
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from collections import Counter
import matplotlib.pyplot as plt

# ------------------------
# 1 Device
# ------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```




```
# 4 Padding
# ------------------------
max_len = 300
def pad_sequence(seq):
    return seq[:max_len] + [0]*(max_len - len(seq)) if len(seq) < max_len else seq[:max_len]

df['padded'] = df['encoded'].apply(pad_sequence)

# ------------------------
# 5 Train/Test split
# ------------------------
X = np.array(df['padded'].tolist())
y = np.array(df['sentiment'].tolist())

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

X_train = torch.tensor(X_train, dtype=torch.long).to(device)
X_test = torch.tensor(X_test, dtype=torch.long).to(device)
y_train = torch.tensor(y_train, dtype=torch.float32).to(device)
y_test = torch.tensor(y_test, dtype=torch.float32).to(device)

train_data = torch.utils.data.TensorDataset(X_train, y_train)
test_data = torch.utils.data.TensorDataset(X_test, y_test)

train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128)

# ------------------------
# 6 LSTM Model
# ------------------------
class LSTMClassifier(nn.Module):
```



LSTM.ipynb - Colab
https://colab.research.google.com/drive/1MtUqu_IOBFB_atRQrvh_JhGiWI9-u1Zo?usp=sharing
LSTM.ipynb
Changes will not be saved
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all Copy to Drive
Share
Connect T4
Train/Test Loss
Train Loss
Test Loss
Loss
Epochs
Test Accuracy: 89.77%
# 10 Prediction
def predict_sentiment(review, model, vocab, max_len, device):
    model.eval()
    tokens = tokenize(review)
    encoded = encode(tokens)
Variables
Terminal



LSTM.ipynb - Colab
https://colab.research.google.com/drive/1MtUqu_IOBFB_atRQrvh_JhGiWI9-u1Zo?usp=sharing
LSTM.ipynb
Changes will not be saved
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all Copy to Drive
Share
Connect T4
    padded = pad_sequence(encoded)
    input_tensor = torch.tensor(padded, dtype=torch.long).unsqueeze(0).to(device)
    with torch.no_grad():
        output = model(input_tensor).squeeze()
        prediction = (output > 0.5).item()
    return "Positive" if prediction == 1 else "Negative", output.item()
# Example prediction
sample_review = "This movie was absolutely amazing! I loved every part of it."
sentiment, probability = predict_sentiment(sample_review, model, vocab, max_len, device)
print(f"Review: \"{sample_review}\"")
print(f"Predicted Sentiment: {sentiment} (Probability: {probability:.4f})")
sample_review_2 = "This movie was terrible. I hated it."
sentiment_2, probability_2 = predict_sentiment(sample_review_2, model, vocab, max_len, device)
print(f"Review: \"{sample_review_2}\"")
print(f"Predicted Sentiment: {sentiment_2} (Probability: {probability_2:.4f})")
Review: "This movie was absolutely amazing! I loved every part of it."
Predicted Sentiment: Positive (Probability: 0.9903)
Review: "This movie was terrible. I hated it."
Predicted Sentiment: Negative (Probability: 0.0013)
model.eval()
LSTMClassifier(
  (embedding): Embedding(101946, 200, padding_idx=0)
  (lstm): LSTM(200, 256, num_layers=2, batch_first=True, dropout=0.3, bidirectional=True)
  (fc): Linear(in_features=512, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)
Variables
Terminal

LAB-9 BUILD A RECCURRENT NATURAL NETWORK

AIM :- To design, implement and evaluate a RNN model for Sequential data, Such as text, and analysis its performance

PSEUDO CODE : . LOAD the dataset .

- preprocess data

-> Convert sequenced into input-output pairs

- Define RNN model:

-> RNN layer + Dense Output layer with activation

- Compile model

-> Select optimizer

- Train Model.

-> Fit data into RNN for given epoch & batch size

-> Monitor ~~validation~~ layer

- Evaluate ~~model~~.

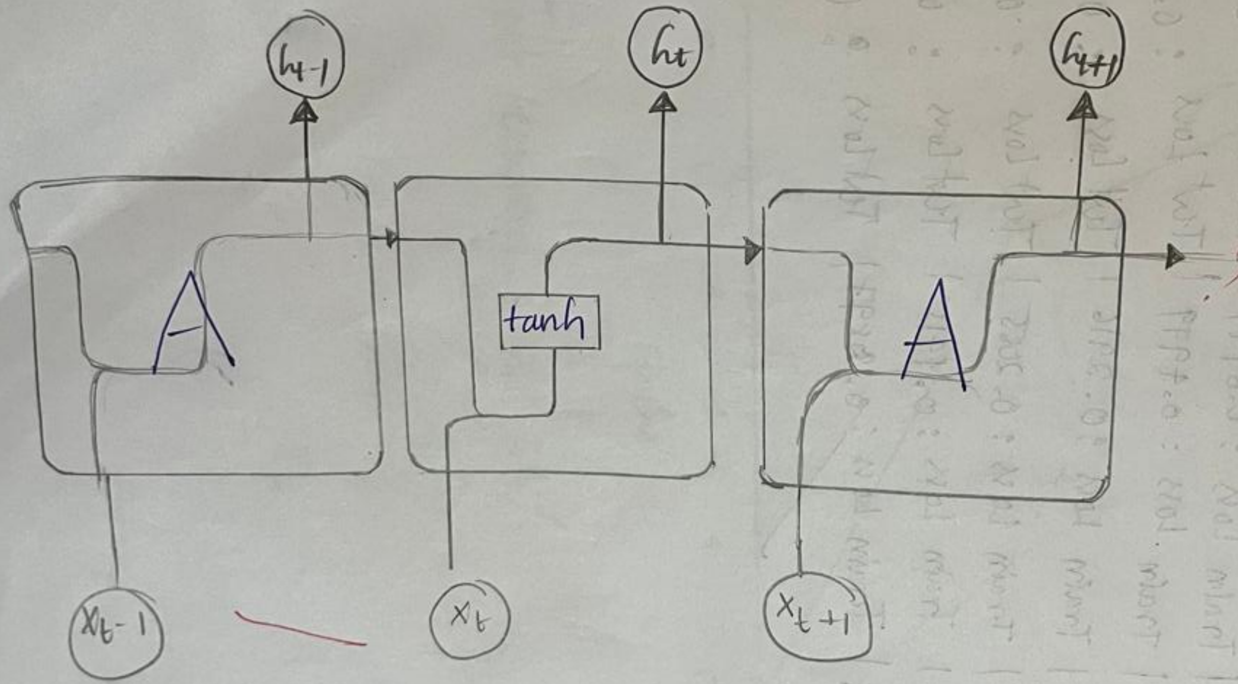
-> Test data

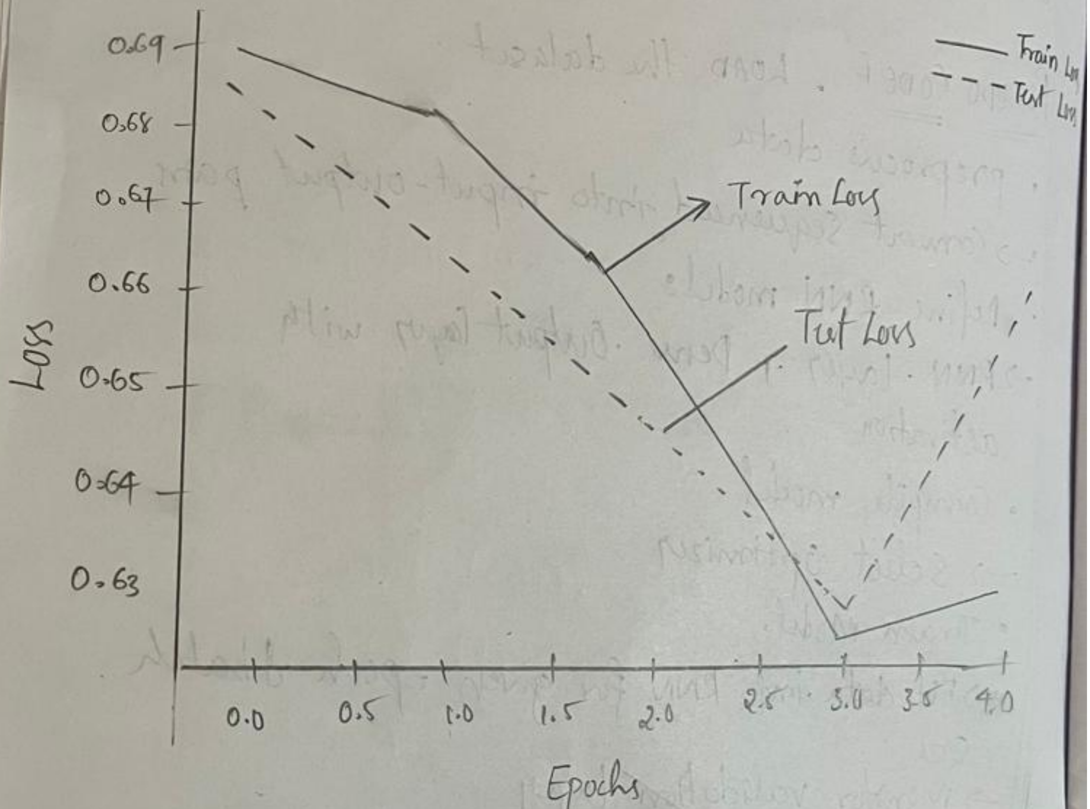
- Visualize results:

-> plot accurracy & load Curves

- Conducts observation & Results

RNN





RNN Test Accuracy : 58.64%

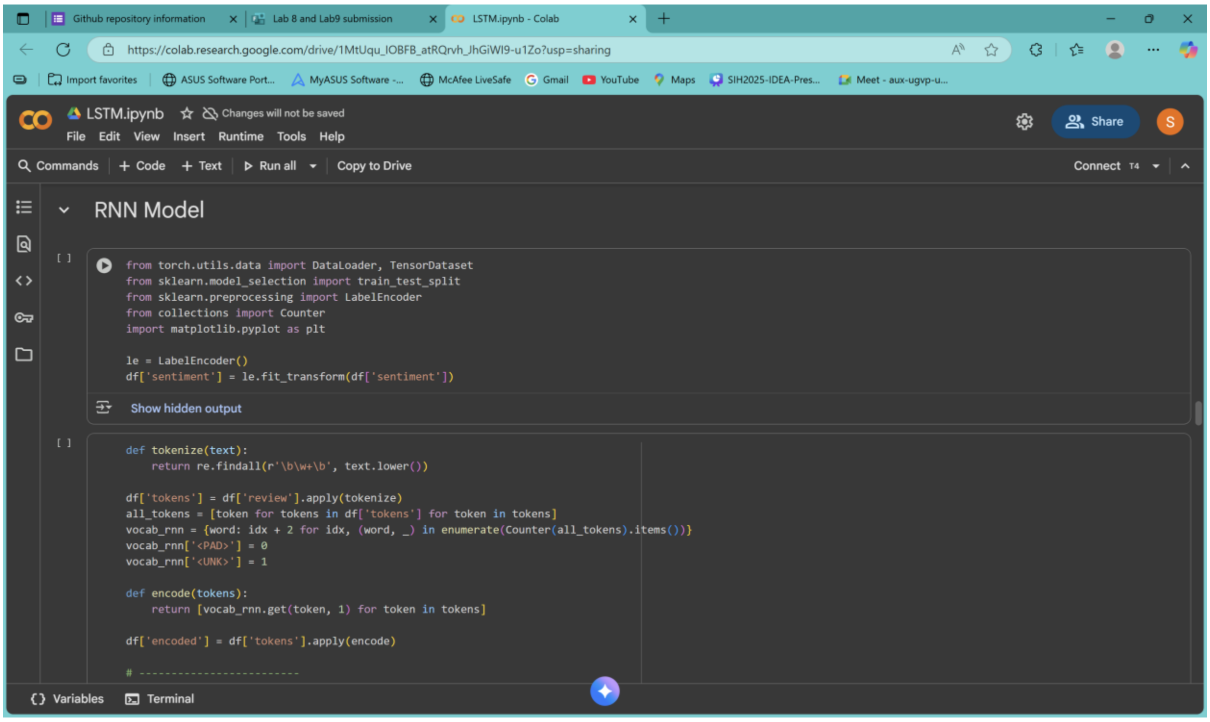
OBSERVATION :-

- The training accuracy increases with epochs, while the loss decreased
- Overfitting can occur if too many epochs are used without regularization (dropout)
- LSTM variants perform more efficiently on long sequences due to vanishing gradient mitigation
- validation performance depends on dataset complexity & preprocessing quality

Result: A R.N.N as successfully and trained a sequential data. "Successfully Implemented".

Epoch 1/5 | Train Loss: 0.6893 | Test Loss: 0.6854
Epoch 2/5 | Train Loss: 0.6832 | Test Loss: 0.6708
Epoch 3/5 | Train Loss: 0.6645 | Test Loss: 0.6527
Epoch 4/5 | Train Loss: 0.6277 | Test Loss: 0.6308
Epoch 5/5 | Train Loss: 0.6296 | Test Loss: 0.6724

1/10/20

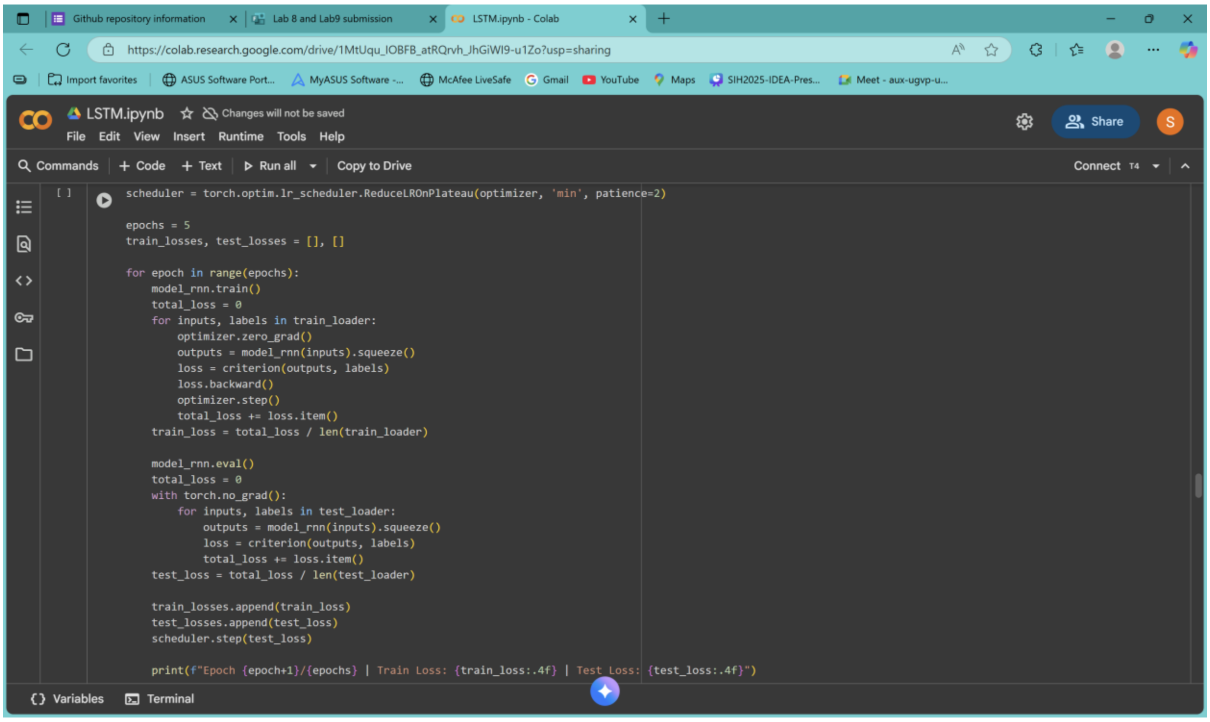


RNN Model

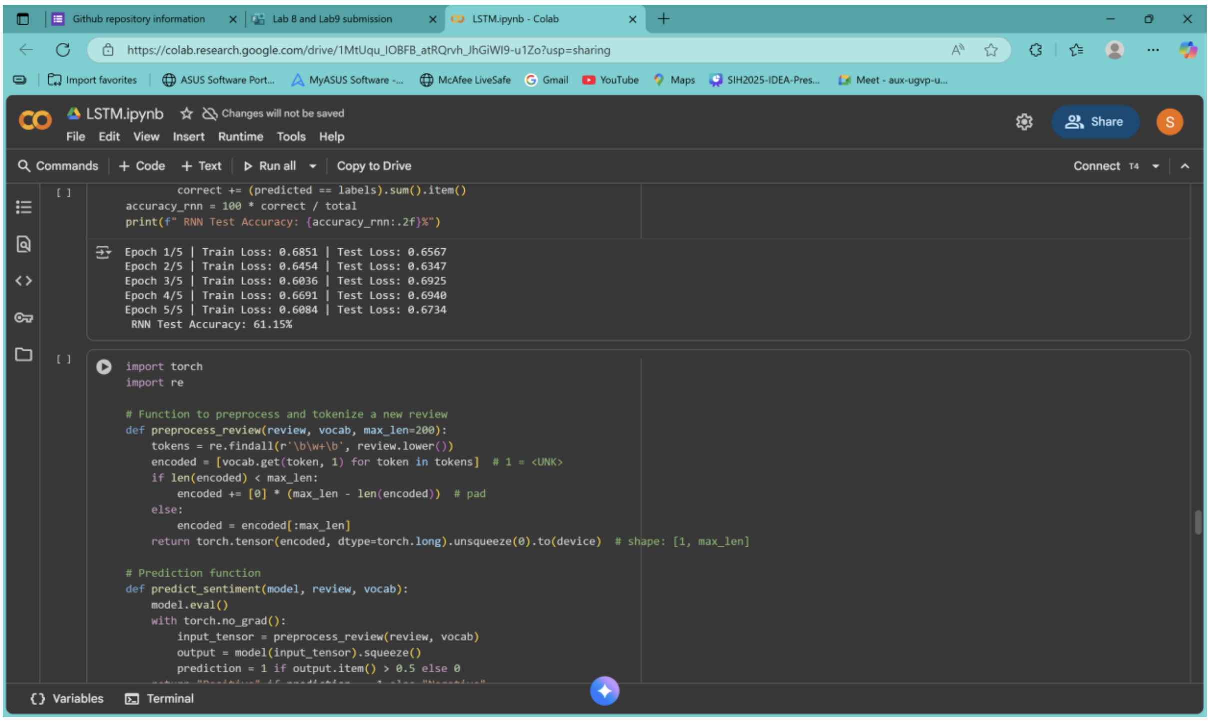
```python
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from collections import Counter
import matplotlib.pyplot as plt

le = LabelEncoder()
df['sentiment'] = le.fit_transform(df['sentiment'])
```

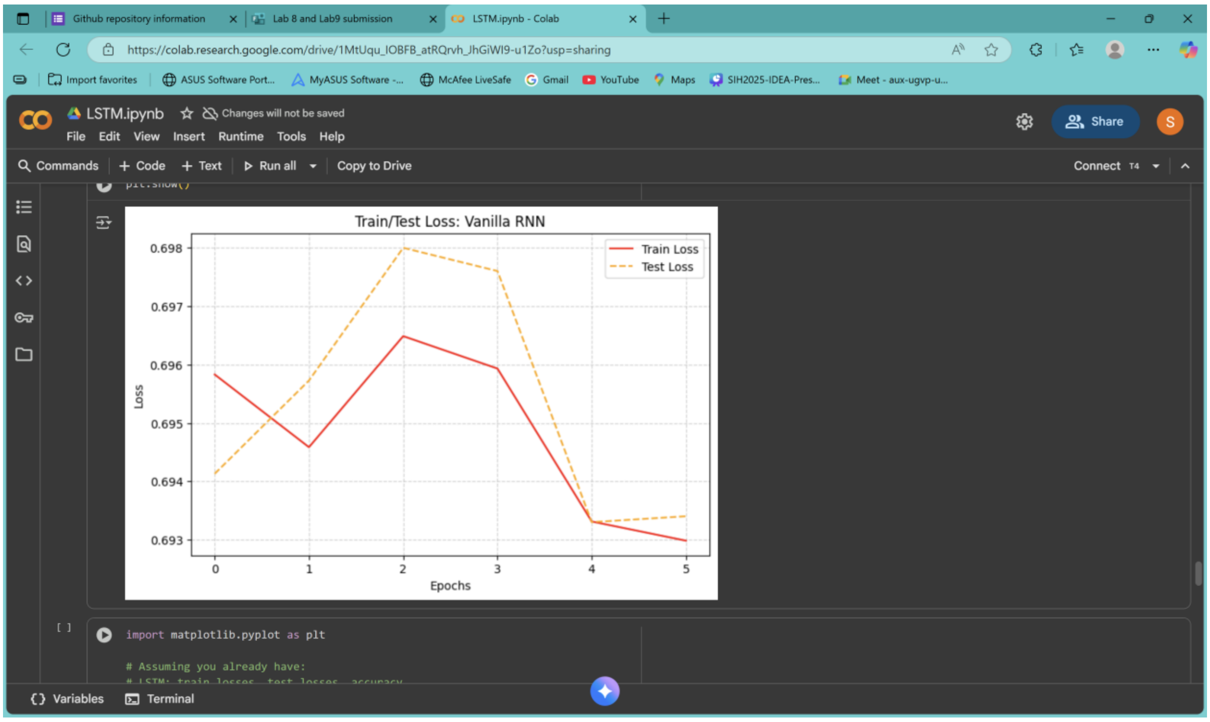
Show hidden output

```python
def tokenize(text):
    return re.findall(r'\b\w+\b', text.lower())

df['tokens'] = df['review'].apply(tokenize)
all_tokens = [token for tokens in df['tokens'] for token in tokens]
vocab_rnn = {word: idx + 2 for idx, (word, _) in enumerate(Counter(all_tokens).items())}
vocab_rnn['<PAD>'] = 0
vocab_rnn['<UNK>'] = 1

def encode(tokens):
    return [vocab_rnn.get(token, 1) for token in tokens]

df['encoded'] = df['tokens'].apply(encode)

# ------------------------
```



```python
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=2)

epochs = 5
train_losses, test_losses = [], []

for epoch in range(epochs):
    model_rnn.train()
    total_loss = 0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model_rnn(inputs).squeeze()
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    train_loss = total_loss / len(train_loader)

    model_rnn.eval()
    total_loss = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            outputs = model_rnn(inputs).squeeze()
            loss = criterion(outputs, labels)
            total_loss += loss.item()
    test_loss = total_loss / len(test_loader)

    train_losses.append(train_loss)
    test_losses.append(test_loss)
    scheduler.step(test_loss)

    print(f"Epoch {epoch+1}/{epochs} | Train Loss: {train_loss:.4f} | Test Loss: {test_loss:.4f}")
```


LSTM.ipynb
Changes will not be saved
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all Copy to Drive
Connect
correct += (predicted == labels).sum().item()
accuracy_rnn = 100 * correct / total
print(f" RNN Test Accuracy: {accuracy_rnn:.2f}%")
Epoch 1/5 | Train Loss: 0.6851 | Test Loss: 0.6567
Epoch 2/5 | Train Loss: 0.6454 | Test Loss: 0.6347
Epoch 3/5 | Train Loss: 0.6036 | Test Loss: 0.6925
Epoch 4/5 | Train Loss: 0.6691 | Test Loss: 0.6940
Epoch 5/5 | Train Loss: 0.6084 | Test Loss: 0.6734
RNN Test Accuracy: 61.15%
import torch
import re
# Function to preprocess and tokenize a new review
def preprocess_review(review, vocab, max_len=200):
tokens = re.findall(r'\b\w+\b', review.lower())
encoded = [vocab.get(token, 1) for token in tokens] # 1 = <UNK>
if len(encoded) < max_len:
encoded += [0] * (max_len - len(encoded)) # pad
else:
encoded = encoded[:max_len]
return torch.tensor(encoded, dtype=torch.long).unsqueeze(0).to(device) # shape: [1, max_len]
# Prediction function
def predict_sentiment(model, review, vocab):
model.eval()
with torch.no_grad():
input_tensor = preprocess_review(review, vocab)
output = model(input_tensor).squeeze()
prediction = 1 if output.item() > 0.5 else 0
Variables
Terminal


LSTM.ipynb
Changes will not be saved
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all Copy to Drive
Connect
Train/Test Loss: Vanilla RNN
Train Loss
Test Loss
Loss
Epochs
0.698
0.697
0.696
0.695
0.694
0.693
0 1 2 3 4 5
import matplotlib.pyplot as plt
# Assuming you already have:
Variables
Terminal