Name:- Parth Kulkarni
PRN:- 202201040007
Batch:- T2

Assignment 5

Experiment 5.1:

Objective:
To forecast future values of a univariate time series using LSTM-based models.

Experiment 5.2: Sequence Text Prediction using LSTM

Objective:
To generate next characters/words based on a given input sequence using LSTM.

Experiment 5.3: Sequence Text Classification using LSTM

Objective:
To classify text sequences using LSTM-based models (e.g., sentiment or spam detection).

Colab Link:- Parth Kulkarni_DL_Assignment_5ipynb

DL Lab Assignment - LSTM

Course Name: Deep Learning-MDM

Lab Title: LSTM Lab Assignment

Student Name: Rohit Jagtap

Student ID: 202201040048

Experiment 1:

Objective:To forecast future values of a univariate time series using LSTM-based models.

Dataset-link (Melbourne Daily Min Temp — Weather Dataset): https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv

Cell 1: Import Required Libraries

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

Cell 2: Load and Display Dataset Info

```
# Load Dataset (Melbourne Daily Min Temp – Weather Dataset)
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv'
df = pd.read_csv(url)
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df = df[['Temp']]  # Using only the minimum daily temperature

# Dataset Summary
print("Dataset: Daily Minimum Temperatures – Melbourne")
print("Date Range:", df.index.min(), "to", df.index.max())
print("Total Records:", len(df))
df.head()
```

```
Dataset: Daily Minimum Temperatures – Melbourne
Date Range: 1981-01-01 00:00:00 to 1990-12-31 00:00:00
Total Records: 3650
```

Date	Temp
1981-01-01	20.7
1981-01-02	17.9
1981-01-03	18.8
1981-01-04	14.6
1981-01-05	15.8

Next steps: Generate code with df | View recommended plots | New interactive sheet

Cell 3: Normalize Data

```
# Normalize the temperature values
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(df)

print("Data normalized using MinMaxScaler.")
```

```
Data normalized using MinMaxScaler.
```

Cell 4: Create Sequences

```
# Function to create time series sequences
def create_sequences(data, seq_len):
    X, y = [], []
    for i in range(len(data) - seq_len):
        X.append(data[i:i + seq_len])
        y.append(data[i + seq_len])
    return np.array(X), np.array(y)

# Set sequence length
seq_len = 30
X, y = create_sequences(data_scaled, seq_len)

print("Sequence length used for LSTM:", seq_len)
print("Input shape:", X.shape)
```

```
Sequence length used for LSTM: 30
Input shape: (3620, 30, 1)
```

Cell 5: Train/Test Split

```
# Train-test split
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

print("Train/Test split ratio: 80/20")
print("Training samples:", X_train.shape[0])
print("Testing samples:", X_test.shape[0])
```

```
Train/Test split ratio: 80/20
Training samples: 2896
Testing samples: 724
```

Cell 6: Define and Compile LSTM Model

```
# Define LSTM Model
model = Sequential()
model.add(LSTM(64, input_shape=(seq_len, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

print("LSTM Model Defined:")
print("LSTM units: 64")
print("Optimizer: Adam")
print("Loss function: MSE")

model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argu
  super().__init__(**kwargs)
LSTM Model Defined:
LSTM units: 64
Optimizer: Adam
Loss function: MSE
Model: "sequential"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	16,896
dense (Dense)	(None, 1)	65

```
Total params: 16,961 (66.25 KB)
Trainable params: 16,961 (66.25 KB)
Non-trainable params: 0 (0.00 B)
```

Cell 7: Train the Model

```
# Train LSTM model
epochs = 100
batch_size = 32
print(f"Training the model for {epochs} epochs with batch size {batch_size}...")

history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 72/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0084 - val_loss: 0.0073
Epoch 73/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0093 - val_loss: 0.0070
Epoch 74/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0086 - val_loss: 0.0070
Epoch 75/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0084 - val_loss: 0.0070
Epoch 76/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0088 - val_loss: 0.0071
Epoch 77/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0086 - val_loss: 0.0072
Epoch 78/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0088 - val_loss: 0.0070
Epoch 79/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0089 - val_loss: 0.0070
Epoch 80/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0088 - val_loss: 0.0070
Epoch 81/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0085 - val_loss: 0.0069
Epoch 82/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0086 - val_loss: 0.0071
Epoch 83/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0084 - val_loss: 0.0070
Epoch 84/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0083 - val_loss: 0.0069
Epoch 85/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0090 - val_loss: 0.0070
Epoch 86/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0087 - val_loss: 0.0070
Epoch 87/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0083 - val_loss: 0.0069
Epoch 88/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0085 - val_loss: 0.0071
Epoch 89/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0082 - val_loss: 0.0070
Epoch 90/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0085 - val_loss: 0.0070
Epoch 91/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0090 - val_loss: 0.0072
Epoch 92/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0085 - val_loss: 0.0071
Epoch 93/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0088 - val_loss: 0.0070
Epoch 94/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0087 - val_loss: 0.0072
Epoch 95/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0088 - val_loss: 0.0071
Epoch 96/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0085 - val_loss: 0.0070
Epoch 97/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0087 - val_loss: 0.0074
Epoch 98/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0083 - val_loss: 0.0070
Epoch 99/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0086 - val_loss: 0.0069
Epoch 100/100
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0080 - val_loss: 0.0071
```

Cell 8: Make Predictions

```
# Predict using the trained model
y_pred = model.predict(X_test)
print("Prediction complete.")
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
Prediction complete.
```

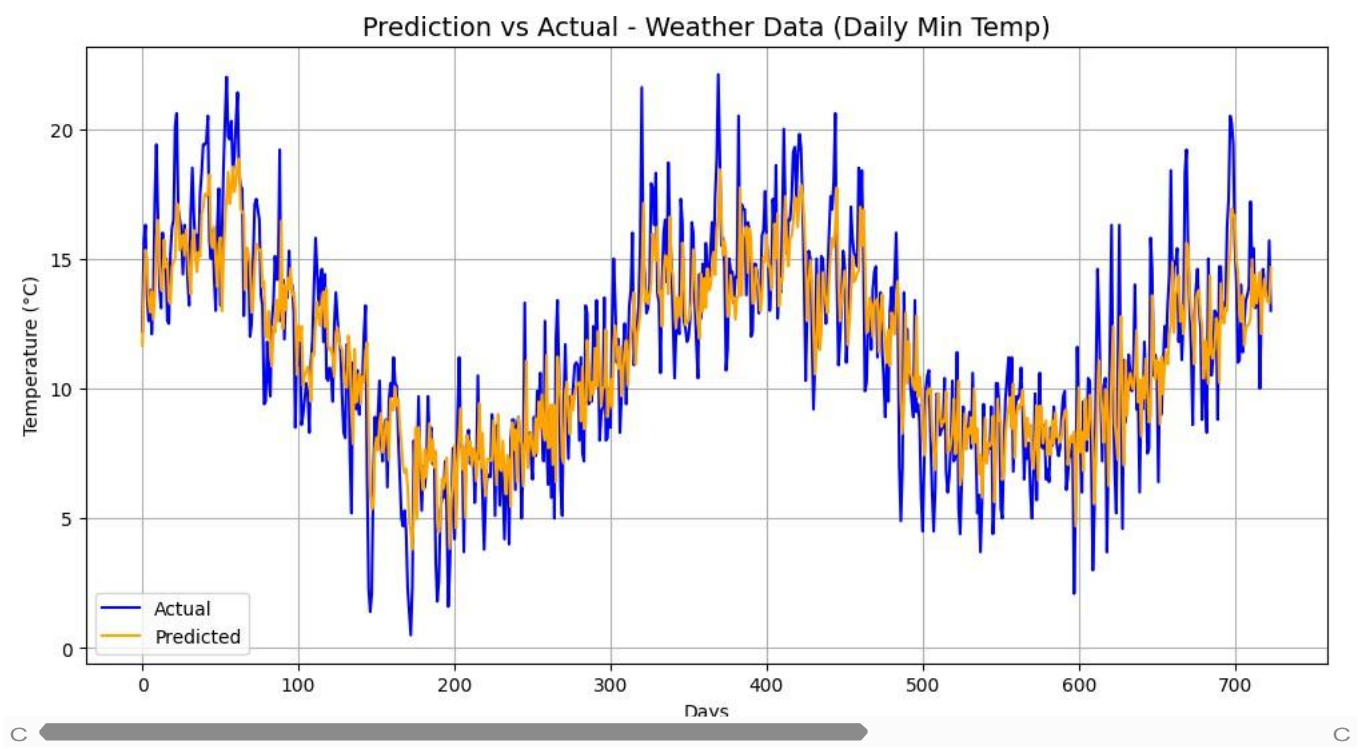
Cell 9: Evaluate Performance

```
# Inverse scale
y_test_inv = scaler.inverse_transform(y_test)
y_pred_inv = scaler.inverse_transform(y_pred)

# Calculate RMSE and MAE
rmse = math.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
mae = mean_absolute_error(y_test_inv, y_pred_inv)

print(f"Evaluation Metrics:")
print(f"RMSE (Root Mean Squared Error): {rmse:.2f}")
print(f"MAE (Mean Absolute Error): {mae:.2f}")
```

```
Evaluation Metrics:
RMSE (Root Mean Squared Error): 2.21
MAE (Mean Absolute Error): 1.74
```

Cell 10: Visualize Results

```
# Plot actual vs predicted values
plt.figure(figsize=(12,6))
plt.plot(y_test_inv, label='Actual', color='blue')
plt.plot(y_pred_inv, label='Predicted', color='orange')
plt.title('Prediction vs Actual - Weather Data (Daily Min Temp)', fontsize=14)
plt.xlabel('Days')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.grid(True)
plt.show()
```



Experiment 2: Sequence Text Prediction using LSTM

Objective:To generate next characters/words based on a given input sequence using LSTM.

Dataset: Shakespeare's Text (TensorFlow Datasets)

Step 1: Install Required Libraries

```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/p
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.13.1)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (
```

```
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.15.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tenso
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflo
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorf
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.
```

Step 2: Load Dataset

```
import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np

# Load dataset without supervised mode
data, info = tfds.load("tiny_shakespeare", with_info=True)

# Read the full text
text_data = ""
for example in data['train']:
    text_data += example["text"].numpy().decode("utf-8")

print(f"Total characters in dataset: {len(text_data)}")
print(f"Sample text:\n{text_data[:500]}")
```

```
WARNING:absl:Variant folder /root/tensorflow_datasets/tiny_shakespeare/1.0.0 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/tens
Dl Completed...: 100%  1/1 [00:00<00:00,  2.34 url/s]
Dl Size...:  1/0 [00:00<00:00,  2.44 MiB/s]
Dataset tiny_shakespeare downloaded and prepared to /root/tensorflow_datasets/tiny_shakespeare/1.0.0. Subsequent calls will reuse t
Total characters in dataset: 1003854
Sample text:
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.

All:
We know't, we know't.

First Citizen:
Let us kill him, and we'll have corn at our own price.
Is't a verdict?

All:
No more talking on't; let it be done: away, away!

Second Citizen:
One word, good citizens.

First Citizen:
We are accounted poor
```

Step 3: Text Preprocessing

```
# Create character-level vocabulary
vocab = sorted(set(text_data))
char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)

# Vectorize text
text_as_int = np.array([char2idx[c] for c in text_data])
```

```
# Set sequence length
seq_length = 100
examples_per_epoch = len(text_data)//seq_length

# Create training sequences
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
sequences = char_dataset.batch(seq_length+1, drop_remainder=True)

def split_input_target(chunk):
    input_text = chunk[:-1]
     target_text = chunk[1:]
     return input_text, target_text

dataset = sequences.map(split_input_target)
```

Step 4: Create Batches

```
# Batch size and buffer for shuffling
BATCH_SIZE = 64
BUFFER_SIZE = 10000

dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)
```

Step 5: Build the LSTM Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Input

vocab_size = len(vocab)  # vocab_size should be the length of your vocabulary
embedding_dim = 256  # you can adjust this value as needed
rnn_units = 1024

model = Sequential([
    Input(shape=(None,)),  # Input shape defined here
    Embedding(vocab_size, embedding_dim),
    LSTM(rnn_units, return_sequences=True),
    Dense(vocab_size)
])

model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, None, 256)	16,640
lstm_1 (LSTM)	(None, None, 1024)	5,246,976
dense_1 (Dense)	(None, None, 65)	66,625

```
Total params: 5,330,241 (20.33 MB)
```

Step 6: Define Loss and Compile

```
def loss(labels, logits):
    return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)

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

** Step 7: Train the Model**

```
history = model.fit(dataset, epochs=10)
model.save_weights("shakespeare_model.weights.h5")  # ✅ Save weights after training
```

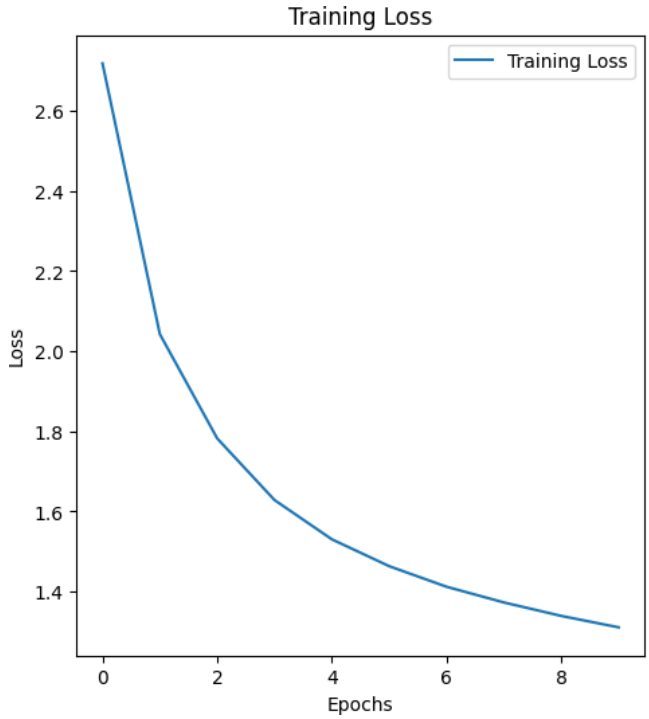
```
Epoch 1/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 17s 66ms/step - loss: 3.1914
Epoch 2/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 12s 67ms/step - loss: 2.1289
Epoch 3/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 21s 69ms/step - loss: 1.8315
```

```
Epoch 4/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 12s 70ms/step - loss: 1.6547
Epoch 5/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 71ms/step - loss: 1.5491
Epoch 6/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 71ms/step - loss: 1.4726
Epoch 7/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 72ms/step - loss: 1.4160
Epoch 8/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 72ms/step - loss: 1.3767
Epoch 9/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 72ms/step - loss: 1.3416
Epoch 10/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 71ms/step - loss: 1.3123
```

Step 8: Plot Training Loss

```
import matplotlib.pyplot as plt

# Plot training loss
plt.figure(figsize=(12, 6))

# Plot training loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7e3bb6abe990>
```



Step 9: Text Generation Function

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense
import tensorflow as tf

# Build model with Input() layer that defines batch_input_shape
model = Sequential([
    Input(batch_shape=(1, None)),  # Define input shape here!
    Embedding(vocab_size, embedding_dim),
    LSTM(rnn_units, return_sequences=True, stateful=True, recurrent_initializer='glorot_uniform'),
    Dense(vocab_size)
])

# Load weights before text generation
model.load_weights("shakespeare_model.weights.h5")  # Use saved weights

# Model is already built with Input layer
# Now you can generate text
```

```
def generate_text(model, start_string):
    num_generate = 500
    input_eval = [char2idx[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []
    temperature = 0.5

    # Reset the states of the LSTM layer
    model.layers[1].reset_states()  # Reset states of LSTM layer (index 1)

    for i in range(num_generate):
        predictions = model(input_eval)
        predictions = tf.squeeze(predictions, 0)

        predictions = predictions / temperature
        predicted_id = tf.random.categorical(predictions, num_samples=1)[-1, 0].numpy()

        input_eval = tf.expand_dims([predicted_id], 0)
        text_generated.append(idx2char[predicted_id])

    return start_string + ''.join(text_generated)

# Run the generation
print(generate_text(model, start_string="To be or not to be, "))
```

```
To be or not to be, be promsesh in his prisoner
To the seators of a way for a word,
That I shall be a woman's country's blood,
In the rest be bear to the man's life.

KING RICHARD III:
Why, how now, as the cousin Blood of a good
With him he would be dead: but be a cause,
So shall be so some at like to answer thee,
The other way with compassion to him.

SICINIUS:
Now, sir, sir, and my son, sir, and leave the dear beaten
As was a traitor of thy name of my son,
Where is the corn provost and will accused bear
The provo
```

```
print(generate_text(model, start_string="Once upon a time, "))
```

```
Once upon a time, we shall have a love to see
The princes his mother with a peace, be holy dread,
And that in the county banish'd blood;
And in the people are desperation of a traitor,
That love the true love as honour will not say.

DUCHESS OF YORK:
How now, good mad, resist they shall be so strength
And not a sweet princely gentleman,
That he may command of the world with me;
The earth hath see the second should be so.

First Musician:
Why, let me not we will make you to your pleasure to my best
All find to her
```

```
print(generate_text(model, start_string="In the heart of the jungle, "))
```

```
In the heart of the jungle, bring a sight were false
To himself and my part and love and great days.

KING RICHARD III:
This is the duke of worthy man that I may be gone,
And so he hath along in the other person,
That we did not move a man as death,
The wish'd son, heart it not with a loather's day.

LADY ANNE:
When I shall be the duke provoked by this prince,
And with the traitor of the country--

GLOUCESTER:
Now gone, be gone, the second well say the commons
And person the king and honour and dead
The sight shall have no
```

Experiment 3: Sequence Text Classification using LSTM

Objective:To classify text sequences (movie reviews) as positive or negative sentiment using an LSTM model.

Dataset-Link (IMDb dataset): https://www.tensorflow.org/datasets/catalog/imdb_reviews

Step 1: Install Required Libraries

```
!pip install seaborn
```

```
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /usr/local/lib/python3.11/dist-packages (from seaborn) (2.0.2)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.11/dist-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/local/lib/python3.11/dist-packages (from seaborn) (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.1.0
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seabo
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=
```

Step 2: Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from sklearn.metrics import classification_report, confusion_matrix
```

Step 3: Load IMDb Dataset

```
# Load top 10,000 words
num_words = 10000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_words)

print("Training samples:", len(x_train))
print("Test samples:", len(x_test))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Training samples: 25000
Test samples: 25000
```

Step 4: Preprocess - Padding Sequences

```
# Padding to ensure equal input length
maxlen = 200
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

print("x_train shape:", x_train.shape)
print("x_test shape:", x_test.shape)
```

```
x_train shape: (25000, 200)
x_test shape: (25000, 200)
```

Step 5: Build LSTM Model

```
model = Sequential([
    Embedding(num_words, 100, input_length=maxlen),
    LSTM(64, return_sequences=True),
    LSTM(32),
    Dropout(0.5),
    Dense(64, activation='relu'),
```

```
    Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Build the model before training
model.build(input_shape=(None, maxlen))
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated.
  warnings.warn(
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 200, 100)	1,000,000
lstm_3 (LSTM)	(None, 200, 64)	42,240
lstm_4 (LSTM)	(None, 32)	12,416
dropout (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 64)	2,112
dense_4 (Dense)	(None, 1)	65

Total params: 1,056,833 (4.03 MB)

Step 6: Train the Model

```
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2,
                    verbose=1)
```

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 15s 18ms/step - accuracy: 0.6919 - loss: 0.5467 - val_accuracy: 0.8666 - val_loss: 0.3169
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - accuracy: 0.9028 - loss: 0.2541 - val_accuracy: 0.8690 - val_loss: 0.3153
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.9387 - loss: 0.1747 - val_accuracy: 0.8780 - val_loss: 0.3354
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - accuracy: 0.9605 - loss: 0.1164 - val_accuracy: 0.8636 - val_loss: 0.3539
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.9725 - loss: 0.0807 - val_accuracy: 0.8622 - val_loss: 0.5038
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - accuracy: 0.9796 - loss: 0.0636 - val_accuracy: 0.8474 - val_loss: 0.4641
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.9825 - loss: 0.0560 - val_accuracy: 0.8600 - val_loss: 0.5045
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.9917 - loss: 0.0275 - val_accuracy: 0.8588 - val_loss: 0.6274
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - accuracy: 0.9881 - loss: 0.0379 - val_accuracy: 0.8500 - val_loss: 0.5811
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.9939 - loss: 0.0213 - val_accuracy: 0.8512 - val_loss: 0.7073
```

Step 7: Evaluate the Model

```
loss, accuracy = model.evaluate(x_test, y_test)
print(f"\n📊 Test Accuracy: {accuracy:.4f}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 6s 8ms/step - accuracy: 0.8513 - loss: 0.7145

📊 Test Accuracy: 0.8508
```

Step 8: Predictions and Classification Metrics

```
y_pred = (model.predict(x_test) > 0.5).astype("int32")

print("\n📄 Classification Report:\n")
print(classification_report(y_test, y_pred, digits=4))
```

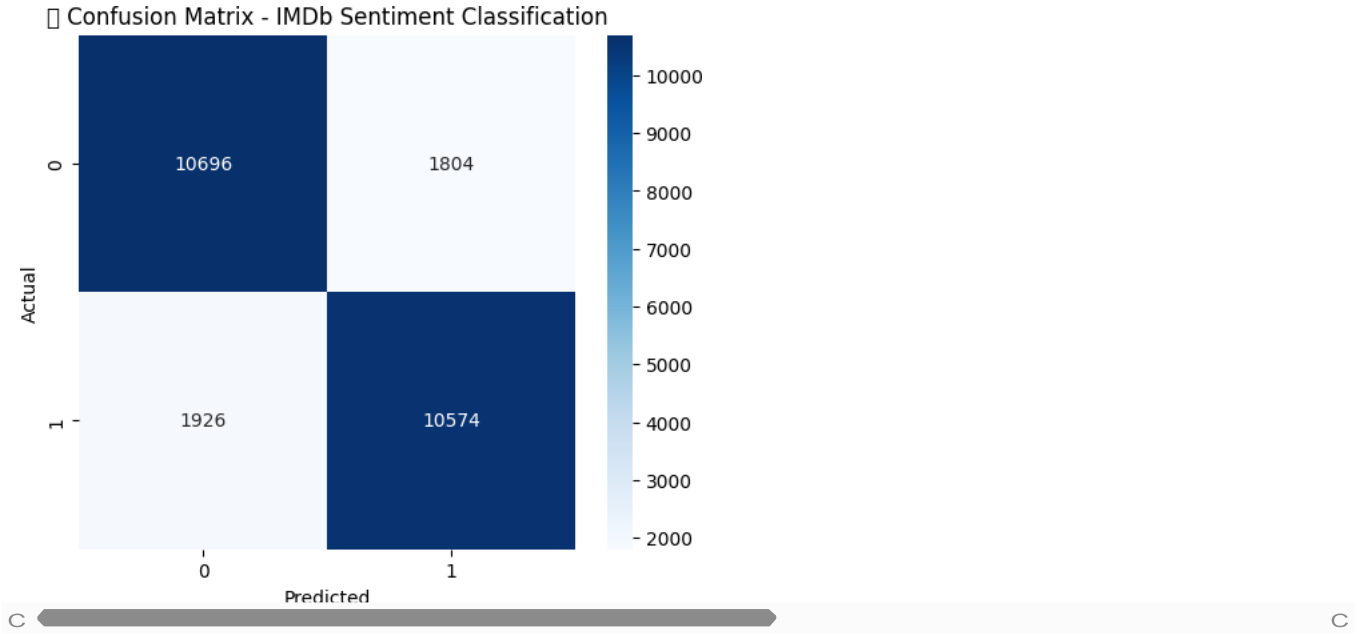
```
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step

📄 Classification Report:

              precision    recall  f1-score   support
```

```
              0     0.8474    0.8557    0.8515     12500
              1     0.8543    0.8459    0.8501     12500

       accuracy                         0.8508     25000
      macro avg     0.8508    0.8508    0.8508     25000
   weighted avg     0.8508    0.8508    0.8508     25000
```

Step 9: Confusion Matrix Visualization

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("📊 Confusion Matrix - IMDb Sentiment Classification")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font
  fig.canvas.print_figure(bytes_io, **kw)
```



📝 Declaration

I, Rohit Jagtap, confirm that the work submitted in this assignment is my own and has been completed following academic integrity guidelines. The code is uploaded on my GitHub repository account, and the repository link is provided below:

Github: https://github.com/RohitJagtap123/Deep_Learning_Assignment.git

Signature: Rohit Jagtap