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) (0.6.0)
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/python3.11/dist-packages (from tensorflow) (5.29.4)
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) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0-
>tensorflow) (0.45.1)
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-
>tensorflow) (3.4.1)
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) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0-
>tensorflow) (2025.1.31)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18-
>tensorflow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /usr/local/lib/python3.11/dist-packages (from
tensorboard<2.19,>=2.18->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18-
>tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1-
>tensorboard<2.19,>=2.18->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0-
>tensorflow) (3.0.0)
```

```
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0-
>tensorflow) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0-
>rich->keras>=3.5.0->tensorflow) (0.1.2)
```

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/tensorflow_datasets/tiny_shakespeare/1.0.0...

{"model_id":"f764ede09f21428d947e8ecdacb3b5e9","version_major":2,"vers
ion_minor":0}

{"model_id":"2e1a3a8bddfd42739265be50c496e368","version_major":2,"vers
ion_minor":0}

{"model_id":"030f7a68019c4924b578e40432ed8d5c","version_major":2,"vers
ion_minor":0}

{"model_id":"515e6332ddce4d75bbcea6d4a9d2dfd8","version_major":2,"vers
ion_minor":0}

{"model_id":"292812cce70541f4802a6f9ae5f53583","version_major":2,"vers
ion_minor":0}

{"model_id":"30b96f4058114fc3b98adb08fef0d748","version_major":2,"vers
ion_minor":0}
```

{"model_id":"918ee2b695944ddb8b2b1b28adcb04a3","version_major":2,"version_minor":0}

{"model_id":"1b5f39d1d25643ce973f146289763ecb","version_major":2,"version_minor":0}

{"model_id":"5103eafcfebb4571a5ba800e744b46af","version_major":2,"version_minor":0}

```
Dataset tiny_shakespeare downloaded and prepared to
/root/tensorflow_datasets/tiny_shakespeare/1.0.0. Subsequent calls
will reuse this data.
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_2 (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)

 Trainable params: 5,330,241 (20.33 MB)

 Non-trainable params: 0 (0.00 B)
```

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**

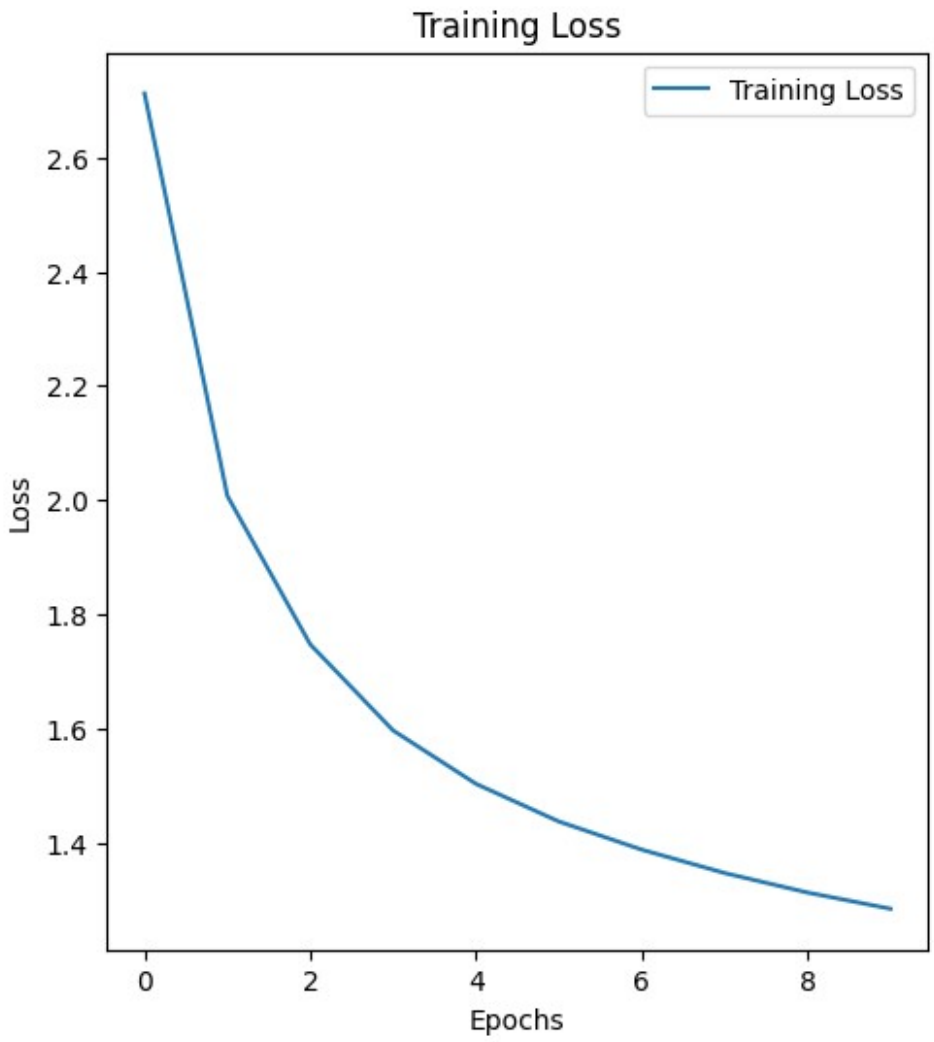
```
history = model.fit(dataset, epochs=10)
model.save_weights("shakespeare_model.weights.h5")  # □ Save weights
after training

Epoch 1/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 14s 68ms/step - loss: 3.2002
Epoch 2/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 12s 69ms/step - loss: 2.0928
Epoch 3/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 21s 71ms/step - loss: 1.7927
Epoch 4/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 72ms/step - loss: 1.6213
Epoch 5/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 73ms/step - loss: 1.5151
Epoch 6/10
```

```
155/155 ━━━━━━━━━━━━━━━━━━━━ 21s 74ms/step - loss: 1.4447
Epoch 7/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 76ms/step - loss: 1.3908
Epoch 8/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 14s 76ms/step - loss: 1.3502
Epoch 9/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 75ms/step - loss: 1.3131
Epoch 10/10
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 74ms/step - loss: 1.2799
```

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 0x7bbb40b36990>
```



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, shall not the talk,
And we shall do before the prince of see,
To make the state that hath show'd the thing
That lie hath but forth rebused in the work,
I am desperation, when thou art not heard,
Who is a shame, that which I have been cause
The prince like the sentence of the duke.

LEONTES:
They speak the duke of thy death?

Page:
My lord, I see the charge must be seen.

LADY ANNE:
What is the daughter as we great would be not so.

MENENIUS:
```

```
What, sir, we shall be revenged this cold,
Did have an

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

Once upon a time, the crown the second shame to death.

KING EDWARD IV:
Now, my lord.

KING RICHARD II:
What should I do tell the time that we do not see.

KING RICHARD III:
Why, sir, let's be wrong.

CLARENCE:
What is the better who let him like a way,
And please your father was revenged him.

CLIFFORD:
Here is the court? What is the commons? make the king,
Is dead is dead to the father of the new down.

KING RICHARD II:
Why, he, my lord, then lessed blood is done,
Common me to the poor boys, that cannot be?

CL

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

In the heart of the jungle, strike the like
To him and say of her father and with the earth.

PAULINA:
What is the destery? Is the man of the which
From man that came the earth as state should
In child, who shall I be arms me contents
The shame of the city for that the death
To say the crown and love a sister, that
Which is not that our hearts I should slay
What is not think that fair suffer'd arms.

DUKE VINCENTIO:
What may be married, the duke of the mother?

KING EDWARD IV:
Why shall be should the store that I do be con
```

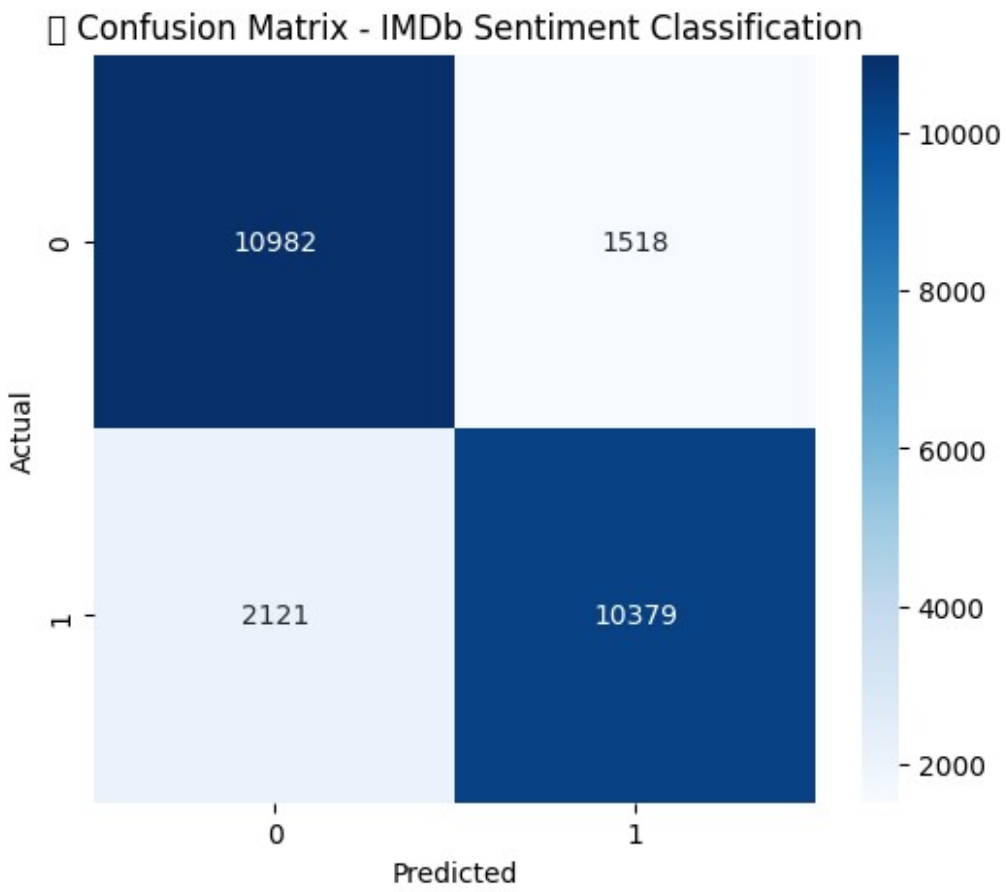
```
y_pred = (model.predict(x_test) > 0.5).astype("int32")

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

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step

 Classification Report:

              precision    recall  f1-score   support

           0     0.8381    0.8786    0.8579     12500
           1     0.8724    0.8303    0.8508     12500

    accuracy                         0.8544     25000
   macro avg     0.8553    0.8544    0.8544     25000
weighted avg     0.8553    0.8544    0.8544     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(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Confusion Matrix - IMDb Sentiment Classification
10982
1518
2121
10379
Actual
Predicted
0
1
0
1
10000
8000
6000
4000
2000