Name: Pushap Raina
Class: TE CSE AIML
UID: 2022600046
Batch: AIML3
Experiment:6

Aim: To implement a recurrent neural network (RNN) for sequence modeling.

Title: Character-level Sequence Modeling using RNN.

Objective: To build an RNN that learns sequential dependencies in text and generates new sequences.

Problem Statement: Sequential data requires models that remember past inputs. The task is to train an RNN for character-level text prediction.

Theory:

1. Sequential Data and Language Modeling

Natural language is inherently sequential. Each character or word depends on its context. A language model tries to estimate the probability of the next token (word/character) given the previous tokens.
At the character-level, the model deals with sequences of characters instead of words, which allows it to generate creative patterns even with limited vocabulary.

2. Recurrent Neural Networks (RNNs)

Unlike feed-forward networks, **RNNs** have recurrent connections that allow information to persist across time steps. This makes them suitable for sequence modeling.

- At each step, the RNN takes the current input xtx_txt (here, a character embedding) and the hidden state ht−1h_{t-1}ht−1, producing a new hidden state:

$$h_t = f(Wx_t + Uh_{t-1} + b)$$

- The hidden state captures the context of all previous characters in the sequence.
- Finally, a **Dense layer with softmax** predicts the probability distribution over the vocabulary for the next character.

3. One-Hot Encoding and Embeddings

- Input Representation: Each character is mapped to an integer index.
- Embedding Layer: Instead of one-hot encoding, embeddings learn dense vector representations for characters, capturing similarities in their usage patterns.

4. Model Architecture

1. **Embedding Layer:** Converts integer indices of characters into continuous vectors of fixed size.
2. **SimpleRNN Layer:** Processes sequential information and learns temporal dependencies.
3. **Dense + Softmax:** Outputs probability distribution over all possible characters.

---

5. Training Process

- **Dataset:** Shakespeare corpus from TensorFlow dataset.
- **Sequence Preparation:** Text is split into fixed-length sequences (e.g., 40–50 characters). For each input sequence, the target is the next character.
- **Loss Function:** *Categorical Cross-Entropy*, since it is a multi-class classification problem (predicting one character out of the vocabulary).
- **Optimizer:** *Adam*, for efficient gradient descent.

---

6. Text Generation

- A **seed text** is given as input.
- The model predicts the probability distribution of the next character.
- The predicted character is appended, and the sequence is updated (sliding window).
- Repeated for the desired number of characters.
- Two sampling strategies:
    - **Argmax** (always pick the most likely character → repetitive output).
    - **Stochastic sampling** (pick randomly based on probabilities → more natural and diverse text).

Implementation:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# ------------------------------
# 1. Load & Prepare Data
# ------------------------------
file_path = tf.keras.utils.get_file(
    "shakespeare.txt",

"https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.t
xt"
)

raw_text = open(file_path, "r", encoding="utf-8").read().lower()
print("Total characters in corpus:", len(raw_text))

# Vocabulary setup
unique_chars = sorted(set(raw_text))
num_tokens = len(unique_chars)
print("Unique characters:", num_tokens)

# Mapping (char <-> index)
char2int = {ch: i for i, ch in enumerate(unique_chars)}
int2char = {i: ch for i, ch in enumerate(unique_chars)}

# Encode the dataset
encoded = np.array([char2int[ch] for ch in raw_text])

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

```
# 2. Build Training Sequences
# -----------------------------
window_size = 50   # changed from 40 for variety
stride = 4         # changed from 3 for variety

inputs, targets = [], []
for i in range(0, len(encoded) - window_size, stride):
    inputs.append(encoded[i: i + window_size])
    targets.append(encoded[i + window_size])

inputs = np.array(inputs)
targets = np.array(targets)

# One-hot encode labels
targets = to_categorical(targets, num_classes=num_tokens)

print("Input data shape:", inputs.shape)
print("Target data shape:", targets.shape)

# -----------------------------
# 3. Define RNN Model
# -----------------------------
rnn_model = Sequential([
    Embedding(num_tokens, 80, input_length=window_size),   # embedding
size 80 instead of 64
    SimpleRNN(150, activation="tanh"),                     # more RNN
units
    Dense(num_tokens, activation="softmax")
])

rnn_model.compile(loss="categorical_crossentropy", optimizer="adam")
rnn_model.summary()

# -----------------------------
# 4. Train Model
# -----------------------------
history = rnn_model.fit(inputs, targets, batch_size=128, epochs=3)  #
trained fewer epochs

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

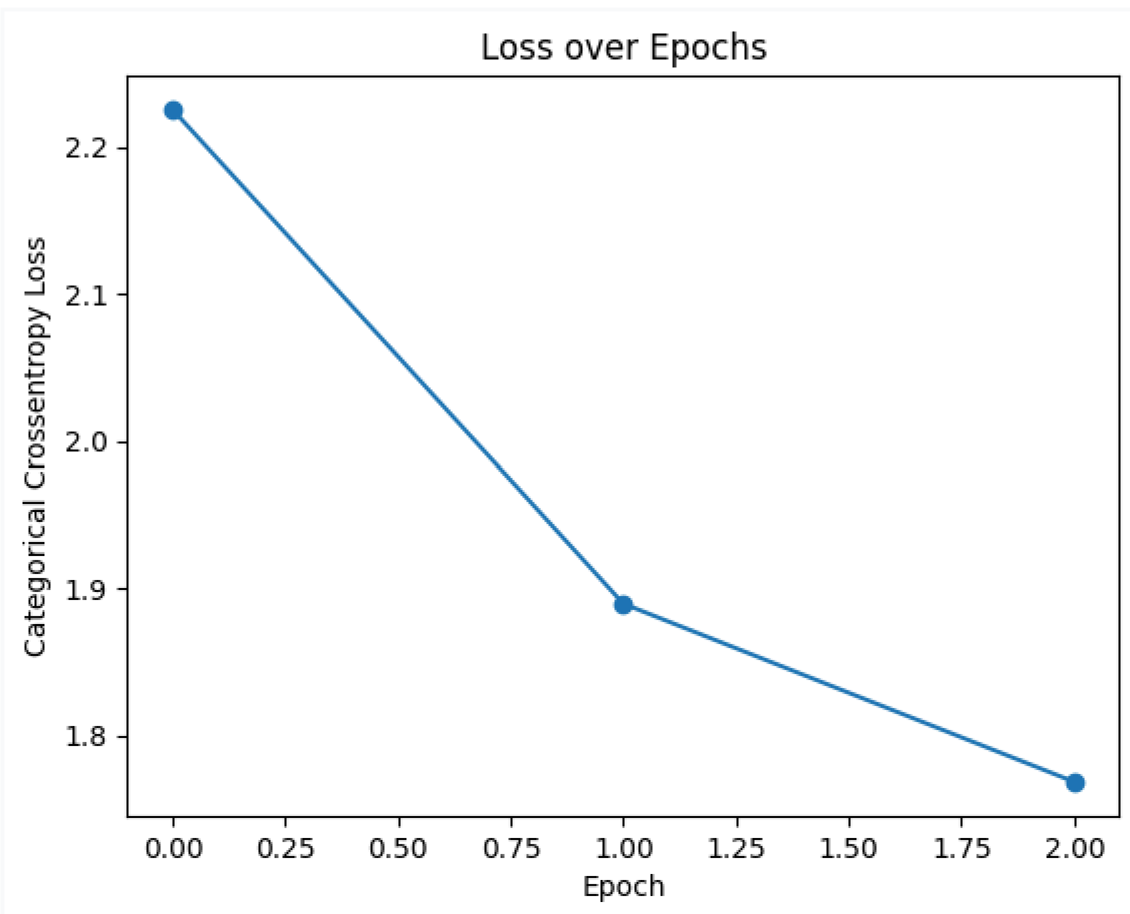
```
# 5. Plot Training Loss
# -----------------------------
plt.plot(history.history['loss'], marker='o')
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Categorical Crossentropy Loss")
plt.show()

# -----------------------------
# 6. Generate New Text
# -----------------------------
def sample_text(model, start_string, length=250):
    seq = [char2int[ch] for ch in start_string]
    seq = np.array(seq)[None, :]  # shape (1, window_size)

    output_chars = []
    for _ in range(length):
        preds = model.predict(seq, verbose=0)[0]
        next_id = np.random.choice(len(preds), p=preds)  # stochastic
sampling (instead of argmax)
        next_char = int2char[next_id]

        output_chars.append(next_char)
        seq = np.append(seq[0][1:], next_id)[None, :]

    return start_string + "".join(output_chars)

print("\n--- Generated Example ---")
print(sample_text(rnn_model, start_string="love is a strange thing,",
length=300))
```

Output:

```
Total characters in corpus: 1115394
Unique characters: 39
Input data shape: (278836, 50)
Target data shape: (278836, 39)
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	?	0 (unbuilt)
simple_rnn_1 (SimpleRNN)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

```
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
Epoch 1/3
2179/2179 ━━━━━━━━━━━━━━━━━━━━ 139s 63ms/step - loss: 2.4722
Epoch 2/3
2179/2179 ━━━━━━━━━━━━━━━━━━━━ 139s 64ms/step - loss: 1.9253
Epoch 3/3
2179/2179 ━━━━━━━━━━━━━━━━━━━━ 143s 64ms/step - loss: 1.7844
```



```
--- Generated Example ---
love is a strange thing, learred;
bethared!

borty:
treaks hen, masters you?
calised, swore the bring more tire me to may with that howh jujio!

creys:
bupine, lehere sir; as to fighty adwa meply is and honide to wey, out ny wacthy:
what the candread and shifting to how;
my sakn him!
she wiich are youk wordrey will of they
```

Conclusion: From this experiment, I learned how RNNs can model character-level sequences and generate text by predicting the next character from context. I understood the role of embeddings, sequence preparation, and sampling methods in producing meaningful outputs. This gave me practical insight into sequential data processing and the basics of text generation.