Training Day-78 Report:

1. Long Short-Term Memory (LSTM) Architecture

LSTMs are an advanced type of Recurrent Neural Network (RNN) designed to overcome the vanishing gradient problem and handle long-term dependencies in sequences. They achieve this with a memory cell and gating mechanisms.

LSTM Components:

1. **Cell State** (CtC_t):
   - The "memory" of the network.
   - Modified by input, forget, and output gates.
2. **Gates**:
   - **Forget Gate**: Decides what information to discard. ft=σ(Wf·[ht−1,xt]+bf)f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)
   - **Input Gate**: Decides what new information to store. it=σ(Wi·[ht−1,xt]+bi)i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) C~t=tanh(WC·[ht−1,xt]+bC)\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) Update the cell state: Ct=ft·Ct−1+it·C~tC_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t
   - **Output Gate**: Determines what part of the cell state contributes to the output. ot=σ(Wo·[ht−1,xt]+bo)o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) ht=ot·tanh(Ct)h_t = o_t \cdot \tanh(C_t)
3. **Hidden State** (hth_t):
   - The output at time tt, influenced by the cell state and output gate.

2. Building a Story Writer using Character-Level RNN

A character-level RNN generates text one character at a time by predicting the next character given the current sequence.

Steps:

1. **Prepare the Data**: Convert text into a sequence of integers, each representing a character.
2. **Build the Model**: Use an LSTM layer to process the character sequences.
3. **Train the Model**: Train the RNN to predict the next character in a sequence.

4. **Generate Text**: Use the trained model to generate text by sampling predictions iteratively.

Implementation: Character-Level RNN with LSTM

```
import tensorflow as tf
import numpy as np

# Sample text data
text = "Once upon a time in a faraway land, there was a little girl named Red Riding Hood."
chars = sorted(set(text))  # Unique characters
char_to_idx = {char: idx for idx, char in enumerate(chars)}
idx_to_char = {idx: char for char, idx in char_to_idx.items()}

# Convert text to integer sequence
encoded_text = np.array([char_to_idx[char] for char in text])

# Prepare input-output pairs
seq_length = 40
input_sequences = []
output_sequences = []
for i in range(len(encoded_text) - seq_length):
    input_sequences.append(encoded_text[i:i + seq_length])
    output_sequences.append(encoded_text[i + seq_length])

input_sequences = np.array(input_sequences)
output_sequences = np.array(output_sequences)

# One-hot encode the input and output
vocab_size = len(chars)
X = tf.keras.utils.to_categorical(input_sequences, num_classes=vocab_size)
y = tf.keras.utils.to_categorical(output_sequences, num_classes=vocab_size)
```

```
# Build the LSTM model
model = tf.keras.Sequential([
   tf.keras.layers.LSTM(256, input_shape=(seq_length, vocab_size),
return_sequences=True),
   tf.keras.layers.Dropout(0.2),
   tf.keras.layers.LSTM(256),
   tf.keras.layers.Dense(vocab_size, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy')
model.summary()

# Train the model
model.fit(X, y, epochs=20, batch_size=64)

# Generate text
def generate_text(model, start_string, gen_length=100):
   input_seq = [char_to_idx[char] for char in start_string]
   input_seq = tf.keras.utils.to_categorical(input_seq, num_classes=vocab_size).reshape(1, -
1, vocab_size)

   generated_text = start_string
   for _ in range(gen_length):
      pred = model.predict(input_seq, verbose=0)
      next_char = idx_to_char[np.argmax(pred)]
      generated_text += next_char

      # Update input sequence
      next_input = tf.keras.utils.to_categorical([np.argmax(pred)], num_classes=vocab_size)
```

```
        input_seq = np.concatenate((input_seq[:, 1:, :], next_input.reshape(1, 1, vocab_size)),
axis=1)

    return generated_text

# Generate a story snippet
start = "Once upon a time"
print(generate_text(model, start))
```

Key Points:

1. **LSTM**:
   - Manages long-term dependencies effectively using gates.
2. **Character-Level RNN**:
   - Generates text by learning patterns in sequences of characters.
3. **Applications**:
   - Chatbots, story generation, code autocompletion, and more.