School of Computer Science Engineering and Technology

Course - B.Tech. | **Semester – 4th**
Course Code - CSET301 | **Course Name - AIML**
Year – 2025 | **Semester - Even**

LAB ASSIGNMENT #9_2

Assignment 9: RNN Implementation for Text Generation

Objective:
Implement a Recurrent Neural Network (RNN) for text generation using PyTorch or Keras. The task is to train an RNN-based language model on a given text dataset and generate new text sequences.

Steps to Complete the Assignment:

Step 1: Dataset Loading

- **Use a large text dataset (e.g., Shakespeare's plays, song lyrics, news articles).**
- **Example: Download the Shakespeare dataset in TensorFlow/Keras:**

```
import tensorflow as tf

import tensorflow.keras.preprocessing.text as tf_text


path = tf.keras.utils.get_file("shakespeare.txt",
"https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt")

text = open(path, 'rb').read().decode(encoding='utf-8')
```

Step 2: Data Preprocessing

- **Tokenize the text into characters or words.**
- **Convert tokens into numerical sequences (e.g., using Tokenizer or Char-to-Index mapping).**
- **Create input-output pairs where:**
    - **Input: A sequence of words/characters**
    - **Output: The next predicted word/character**

Example:

```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(text)
sequences = tokenizer.texts_to_sequences(text)
```

Step 3: Model Construction

- **Build an RNN-based text generator .**
- **The model should include:**
  - **Embedding layer**
  - **rnn layers**
  - **Dense layer with softmax activation for predicting the next character/word**

Step 4: Model Training and Testing

- **Train the model with categorical cross-entropy loss and Adam optimizer.**
- **Train for 20 and 50 epochs to compare performance.**
- **Use a subset of the dataset for validation.**

Example:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test))
```

Step 5: Text Generation

- **Use the trained model to generate text.**
- **Start with a seed sentence and predict the next characters/words iteratively.**

Example:

```
def generate_text(seed_text, next_words=50):
```

```
    for _ in range(next_words):
        tokenized = tokenizer.texts_to_sequences([seed_text])
        padded = pad_sequences(tokenized, maxlen=100)
        predicted_index = np.argmax(model.predict(padded))
        next_word = tokenizer.index_word[predicted_index]
        seed_text += " " + next_word
    return seed_text
```

Step 6: Model Saving & Evaluation

- **Save the trained model for future use.**
- **Evaluate model performance using:**
    - **Perplexity score**
    - **Generated text quality**