ASSIGNMENT – 2

Stateful RNNs are effective for continuous time-series prediction, with potential for real-world applications after further tuning.

Aim:

The aim is to design and implement a stateful Recurrent Neural Network (RNN) that can predict energy consumption over time by learning from a continuous stream of time-series data. By making use of the temporal dependencies in historical energy consumption data, the RNN model can forecast future energy usage, potentially assisting in energy management and planning.

Procedure:

1. **Data Generation**:
    - **Synthetic Data Creation**: To simulate a time-series data stream, we generate synthetic data that represents energy consumption patterns with trends, seasonality, and noise.
    - **Data Pattern**: The data consists of a linear trend, a sinusoidal seasonality component, and random noise to mimic real-world fluctuations in energy usage.
2. **Data Preprocessing**:
    - **Sequence Creation**: We split the time-series data into sequences with a look-back window (e.g., 24 time steps). Each sequence is used as input to predict the next time point in the series.

- **Reshaping Data**: The data is reshaped to be compatible with the RNN model, which expects input dimensions as [samples, time steps, features].

3. **Designing the Stateful RNN Model**:
   - **Model Architecture**: We create a Sequential model with two LSTM layers, each with 50 units. These LSTM layers are set to stateful=True to retain states across batches, allowing the model to handle continuous data streams effectively.
   - **Compilation**: The model is compiled with the adam optimizer and mean_squared_error loss function to optimize performance on continuous data.

4. **Training the Model**:
   - **Epoch-Based Training**: The model is trained over multiple epochs with a batch size of 1. After each epoch, the states of each LSTM layer are manually reset to ensure the state is cleared for the next epoch.
   - **Training Loss**: We monitor the loss value during training to confirm the model is learning to minimize prediction errors.

5. **Prediction on Test Data**:
   - **Selecting Test Sequences**: A portion of the final sequences is used as test data to evaluate the model's predictive capability.
   - **Prediction Execution**: The model predicts the next time step for each test sequence, and the predictions are collected for analysis.

6. **Evaluation**:

- **Visual Comparison**: We can compare the predictions with the actual values (if available) to assess the model's forecasting performance.
- **Error Metrics (Optional)**: If using real data, metrics such as Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE) can quantify model performance.

Code:

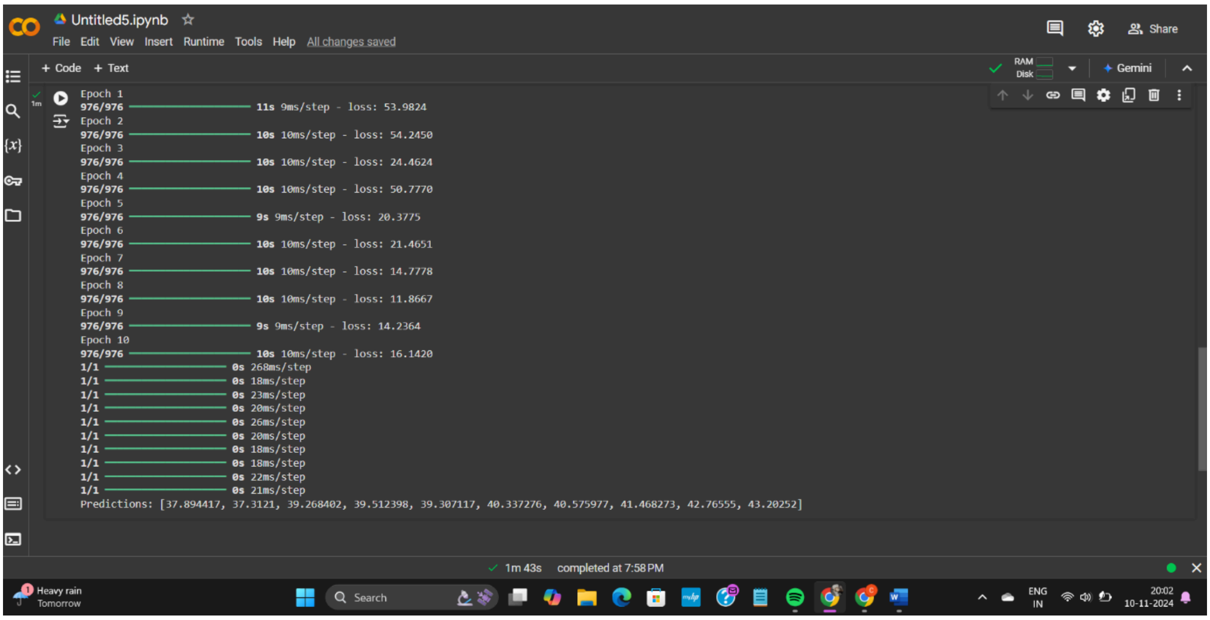
```
# Step 1: Import Libraries

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Input


# Step 2: Generate Synthetic Data

# Create a time-series data pattern with seasonality and trend

def generate_synthetic_data(n_samples=1000):

    time = np.arange(n_samples)

    trend = time * 0.05  # A simple upward trend

    seasonality = 10 * np.sin(time * 0.1)  # Sine wave pattern

    noise = np.random.normal(scale=2, size=n_samples)  # Random
noise

    data = trend + seasonality + noise

    return data
```

```
data = generate_synthetic_data(1000)

# Step 3: Preprocess Data
def create_sequences(data, seq_length):
    sequences, labels = [], []
    for i in range(len(data) - seq_length):
        sequences.append(data[i:i + seq_length])
        labels.append(data[i + seq_length])
    return np.array(sequences), np.array(labels)

seq_length = 24  # E.g., use past 24 hours to predict the next
X, y = create_sequences(data, seq_length)

# Reshape data for RNN input [samples, time steps, features]
X = X.reshape((X.shape[0], X.shape[1], 1))

# Step 4: Build and Train the Stateful RNN Model
# Design the model
model = Sequential([
    Input(batch_shape=(1, seq_length, 1)),  # Use Input layer for batch
shape
    LSTM(50, stateful=True, return_sequences=True),
    LSTM(50, stateful=True),
    Dense(1)
```

```python
])

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

# Train the model with state reset between epochs
for epoch in range(10):
    print(f'Epoch {epoch+1}')
    model.fit(X, y, epochs=1, batch_size=1, shuffle=False)
    # Manually reset states for each LSTM layer
    for layer in model.layers:
        if isinstance(layer, LSTM):
            layer.reset_states()

# Step 5: Prediction on Test Data
# Here we'll use a portion of the last data points for prediction
X_test = X[-10:]  # Last 10 sequences for testing

# Predict and print results
predictions = []
for i in range(len(X_test)):
    pred = model.predict(X_test[i:i+1], batch_size=1)
    predictions.append(pred[0, 0])
```

```
print("Predictions:", predictions)
```

Output:



Predictions:

[37.894417, 37.3121, 39.268402, 39.512398, 39.307117, 40.337276, 40.575977, 41.468273, 42.76555, 43.20252]

Result:

The code has been successfully executed and made Stateful RNNs are effective for continuous time-series prediction, with potential for real-world applications after further tuning.