**Assignment No: - 4**

**Time Series Prediction Using Recurrent Neural Networks (RNNs)**

**Problem Statement:**

Implementing time series prediction using Recurrent Neural Networks (RNNs) for stock market analysis or weather forecasting.

**Objective:**

* To understand the architecture and functioning of Recurrent Neural Networks.
* To learn how to preprocess time series data for RNN training.
* To implement an RNN model using Keras and TensorFlow for time series prediction.
* To evaluate model performance using test data.
* To visualize predictions and compare them with actual values.

**S/W Packages and H/W apparatus used:**

* **Operating System:** Windows/Linux/MacOS
* **Kernel:** Python 3.x
* **Tools:** Jupyter Notebook, Anaconda, or Google Colab
* **Hardware:** CPU with minimum 4GB RAM; optional GPU for faster processing

**Libraries and packages used:**

* TensorFlow
* Keras
* NumPy
* Pandas
* Matplotlib

**Theory:**

**Definition:**  
Recurrent Neural Networks (RNNs) are designed for sequential data processing. They leverage information about prior inputs through internal memory, making them suitable for tasks like time series prediction.

**Structure:**

* **Input Layer:** Accepts time series data (e.g., stock prices, weather).
* **Recurrent Layers:** Comprising RNN cells (e.g., LSTM or GRU) that maintain hidden states capturing temporal dependencies.
* **Fully Connected Layer:** Maps output from recurrent layers to the prediction output.
* **Output Layer:** Predicts future time steps.

**Activation Functions:**

* Common activations include **Tanh** and **Sigmoid** for managing the flow of values through the network.

**Memory Cells:**  
RNNs utilize memory cells like LSTMs and GRUs to handle long-term dependencies and address issues like vanishing gradients.

**Methodology:**

1. **Data Acquisition:**
   * Load historical stock market or weather data from sources like Yahoo Finance or weather APIs.
2. **Data Preparation:**
   * Select relevant features and normalize values between 0 and 1.
3. **Sequence Creation:**
   * Prepare input-output pairs using sequences (e.g., past 60-time steps predict the next one).
4. **Model Architecture:**
   * Create a Sequential Keras model.
   * Add RNN layers (e.g., LSTM or GRU), and a Dense layer for output.
5. **Model Compilation:**
   * Use the **Adam optimizer** and **Mean Squared Error (MSE)** as the loss function.
6. **Model Training:**
   * Fit the model on training data and validate on a separate test set.
7. **Model Evaluation:**
   * Evaluate model performance on the test set.
8. **Prediction Visualization:**
   * Plot predicted vs. actual values to assess model performance visually.

**Advantages:**

* **Sequential Data Handling:** Captures temporal dependencies effectively.
* **Long-Term Memory:** LSTMs and GRUs manage long sequences well.
* **Dynamic Computation:** RNNs adapt to varying input lengths.

**Limitations:**

* **Computational Complexity:** Training can be resource-intensive.
* **Vanishing Gradient Problem:** Though mitigated by LSTMs, traditional RNNs can struggle with long sequences.
* **Overfitting Risk:** Particularly with small datasets.

**Applications:**

* **Stock Market Analysis:** Predicting trends based on historical prices.
* **Weather Forecasting:** Analyzing time series for predicting future weather conditions.

**Working / Algorithm:**

**Step 1:** Import Necessary Libraries:

* Load essential libraries like NumPy, Pandas, Keras, and Matplotlib.

**Step 2:** Load Dataset:

* Load stock market data (e.g., stock prices) and set 'Date' as the index for the DataFrame.

**Step 3:** Preprocess the Data:

* Extract closing prices, convert them to a NumPy array, and normalize values using MinMaxScaler.

**Step 4:** Create Sequences:

* Define a function to generate input-output sequences for time steps.

**Step 5:** Train-Test Split:

* Split the data into training and testing sets (e.g., 80% train, 20% test).

**Step 6:** Build the RNN Model:

* Use Keras to build the Sequential RNN model. Add an LSTM or GRU layer and Dense output layer for regression.

**Step 7:** Model Training:

* Compile the model with Adam optimizer and train it on the training data.

**Step 8:** Prediction for Future Days:

* Use the trained model to predict the next 20 days' stock prices.

**Step 9:** Inverse Transform Predictions:

* Convert predicted values back to the original stock price scale.

**Step 10:** Compare Actual vs. Predicted Prices:

* Extract actual prices for the next 20 days and compare them.

**Step 11:** Plot Results:

* Use Matplotlib to plot actual vs. predicted prices over time.

**Step 12:** Display Actual and Predicted Prices:

* Print the comparison between actual and predicted stock prices for the next 20 days.

**Diagram:**

 **Conclusion:**

Recurrent Neural Networks (RNNs) excel in time series prediction tasks like stock market analysis or weather forecasting. Their ability to model sequential data and maintain long-term dependencies through LSTMs and GRUs makes them powerful tools in various domains. Despite their complexity and potential for overfitting, with proper tuning, RNNs can provide accurate and meaningful predictions based on historical data.