# **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.

### **Objectives**

* 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 a class of artificial neural networks designed for processing sequential data. They have the capability to maintain information about previous inputs in their internal memory, making them particularly suitable for time series prediction tasks.

#### **Structure**

* Input Layer: Accepts sequences of data points (e.g., stock prices or weather measurements).
* Recurrent Layers: Contain RNN cells (e.g., LSTM or GRU) that process sequences and maintain hidden states to capture temporal dependencies.
* Fully Connected Layer: Connects outputs from recurrent layers to the final prediction.
* Output Layer: Produces the predicted values for the next time step.

#### **Activation Functions**

* Commonly used: Tanh and Sigmoid.
* They regulate values during training and ensure stability.

#### **Memory Cells**

* LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) address vanishing gradient problems and help retain long-term dependencies in sequential data.

### **Methodology**

1. Data Acquisition
   * Collect historical stock market data (e.g., stock prices) or weather data (e.g., temperature, humidity) from sources like Yahoo Finance or weather APIs.
2. Data Preparation
   * Select relevant features.
   * Normalize values (range 0 to 1).
3. Sequence Creation
   * Create sequences from time series data.
   * Example: Use past 60 time steps to predict the next value.
4. Model Architecture
   * Sequential model using Keras.
   * Add recurrent layers (LSTM/GRU).
   * Add fully connected and output layers for regression.
5. Model Compilation
   * Optimizer: Adam.
   * Loss Function: Mean Squared Error (MSE).
6. Model Training
   * Train on the dataset.
   * Validate on a separate validation set.
   * Track loss and accuracy.
7. Model Evaluation
   * Evaluate performance on the test dataset.
8. Prediction Visualization
   * Plot actual vs predicted values to assess performance.

### **Advantages**

* Sequential Data Handling: Captures temporal dependencies effectively.
* Long-Term Memory: LSTM and GRU retain information over long sequences.
* Flexibility: Can handle variable input lengths.
* Dynamic Computation: Works with sequences of varying sizes.

### **Limitations**

* Computational Complexity: Training can be time-consuming.
* Vanishing Gradient Problem: A challenge in standard RNNs.
* Overfitting Risk: More likely with smaller datasets.
* Data Requirements: Requires large amounts of historical data.

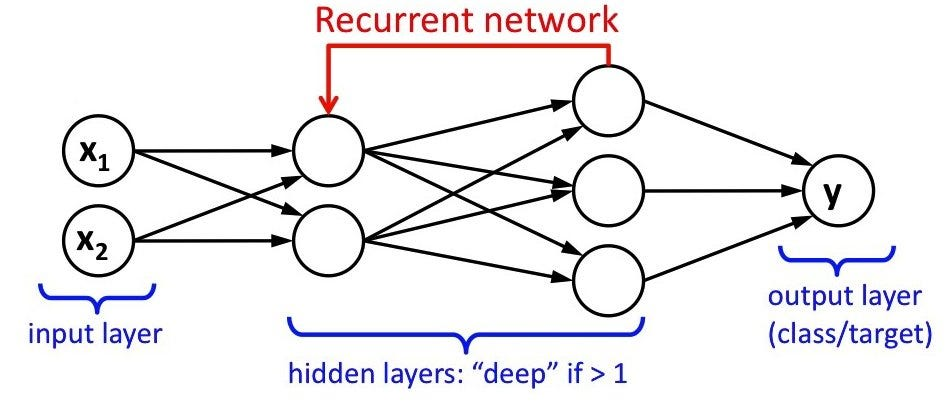
### **Applications**

* Stock Market Analysis – Predicting prices and trends.
* Weather Forecasting – Forecasting conditions like temperature and rainfall.
* Natural Language Processing – Language modeling and text generation.

### **Working / Algorithm**

1. Import Libraries – NumPy, Pandas, Matplotlib, Keras.
2. Load Dataset – Stock market dataset with Date and Close price.
3. Preprocess Data – Extract closing prices, normalize using MinMaxScaler.
4. Create Sequences – Use past 60 time steps to predict the next one.
5. Split Dataset – 80% training, 20% testing.
6. Build Model – Add RNN (e.g., SimpleRNN/LSTM) and Dense output layer.
7. Train Model – Compile with Adam + MSE, fit with epochs and batch size.
8. Predict Next 20 Days – Generate sequential predictions.
9. Inverse Transform – Convert scaled predictions back to original price scale.
10. Compare with Actual Data – Compare predicted vs real prices.
11. Plot Results – Graphical visualization of predictions vs actual values.
12. Print Results – Display actual vs predicted values.

### **Diagram**



### **Conclusion**

Recurrent Neural Networks (RNNs) provide a robust approach to time series prediction by capturing temporal patterns from historical data. Their memory capabilities through LSTM and GRU make them effective for applications such as stock market prediction and weather forecasting. Although computationally demanding and prone to overfitting, RNNs remain a powerful tool for predictive modeling when sufficient data and regularization are applied. With proper tuning, they deliver accurate and meaningful insights into sequential datasets.