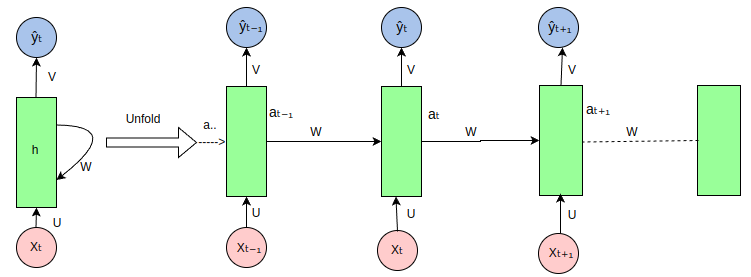
|  |  |
| --- | --- |
| **Ex No: 7**  **Date: 17 Oct** | **Recurrent Neural Networks** |

**Objective:**The objective of this lab is to gain a thorough understanding of Recurrent Neural Networks (RNNs) and implement an RNN model using Python with a deep learning library like TensorFlow or PyTorch. The lab will focus on sequence modeling tasks, including applications in time series forecasting and text processing, where RNNs excel in capturing temporal dependencies within data. This hands-on exercise will enable exploration of RNN architectures, hyperparameter tuning, and evaluation techniques to strengthen skills in sequential data modeling and prediction.

**Description:**

A Recurrent Neural Network (RNN) is a type of neural network designed for sequence-based data, where order and context play a significant role. Unlike traditional neural networks that assume inputs are independent of each other, RNNs are designed to recognize patterns in sequential data by maintaining a “memory” of previous inputs. This makes them well-suited for tasks such as time series prediction, text processing, and speech recognition, where the order of data points affects the output.



This lab provides a step-by-step guide to implementing an RNN, focusing on essential processes such as data preprocessing, model construction, training, and evaluation. Detailed code explanations will accompany each step to ensure a clear understanding of the entire process.

**Building the Model**

**1. Importing Libraries**

Essential libraries are imported to enable data handling, plotting, and model building. numpy and pandas are used for data manipulation, matplotlib.pyplot for visualizations, and tensorflow.keras for constructing and training the RNN model.

**2. Loading and Preprocessing Data**

The time-series data is loaded from a CSV file, and a specific column is selected as the target variable. Min-Max scaling is applied to this data, transforming it to a range between 0 and 1, which is essential for efficient and stable training of neural networks.

**3. Creating Training and Testing Datasets**

The scaled data is divided into training and testing datasets with an 80-20 split. This separation allows the model to be trained on one subset of data while being evaluated on unseen data, providing an accurate assessment of its performance.

**4. Converting Data to RNN-Compatible Format**

The data is structured into sequences of a specified time step, preparing it for input into the RNN. This sequence format is necessary for capturing the temporal dependencies inherent in time-series data, enabling the RNN to learn patterns across time steps.

**5. Reshaping Input for RNN**

The input data is reshaped into a three-dimensional array with the format [samples, time steps, features], as required by RNNs. This step ensures that each data point is organized correctly for the RNN model to process sequential information effectively.

**6. Building the RNN Model**

An RNN model is defined using Keras’ Sequential API, consisting of a Simple RNN layer with 50 units and a dense output layer. The model is compiled with mean squared error as the loss function and the Adam optimizer, preparing it for the training phase.

**7. Training the Model**

The model is trained on the training data for 50 epochs with a batch size of 32. During training, a portion of the data is set aside for validation, allowing us to monitor the model’s performance and ensure it is learning without overfitting.

**8. Model Evaluation and Predictions**

After training, the model generates predictions on both training and testing datasets. These predictions are then inverse-transformed to their original scale, enabling a clear evaluation of the model’s accuracy and comparison with the actual data.

**Conclusion**

This lab demonstrates the complete process of implementing an RNN for time-series prediction, covering essential steps from data preprocessing and scaling to model building, training, and evaluation. By structuring the data into sequences, reshaping it for RNN input, and utilizing Min-Max scaling, we ensure the model can efficiently learn temporal patterns. The training process, with validation monitoring, enables us to track model performance over time, while the inverse transformation of predictions allows for an intuitive comparison to actual values. Overall, this RNN model showcases the effectiveness of recurrent neural networks in sequence modeling tasks and lays the foundation for further enhancements, such as exploring more complex architectures (e.g., LSTM, GRU) or tuning hyperparameters for improved accuracy in real-world applications.

**Github Link:** https://github.com/Kashishvarmaa/DL-CS3232/tree/main/Lab\_7