|  |  |
| --- | --- |
| **Ex No: 8**  **Date: 25/09/2024** | **Character level language model - Name generation**  **RNN** |

**Objective:** The primary objective of this code is to demonstrate how to build and train a Recurrent Neural Network (RNN) using a dataset. RNNs are often used for sequential data tasks, such as time series prediction, natural language processing, and more.

**Descriptions:**

The code provides a hands-on guide to implementing an RNN, training it on a dataset, and evaluating its performance. The code uses the TensorFlow and Keras libraries to implement the model, showcasing how RNNs handle sequential data.

**Model:**

The core model used is an RNN-based architecture. RNNs are ideal for handling sequential data because they have loops in their architecture that allow information to persist.

**Building the parts of algorithm:**

The key steps in building the RNN model include:

1. **Data Preparation:** Loading and preprocessing the dataset to make it suitable for training.
2. **Model Building:** Constructing the RNN model using Keras.
3. **Model Training:** Training the RNN model on the preprocessed data.
4. **Evaluation:** Evaluating the trained model's performance.

**Detailed Code Explanation 1. Importing Libraries:** import numpy as np

import matplotlib.pyplot as plt import pandas as pd import tensorflow as tf from tensorflow import keras from keras.models import Sequential from keras.layers import Dense, SimpleRNN

* **Objective:** Import the necessary libraries and modules.
* **Explanation:** These imports bring in core libraries for numerical operations (NumPy), data visualization (Matplotlib), data handling (Pandas), and deep learning (TensorFlow and Keras).

1. **Loading and Preparing Data** data = pd.read\_csv('data.csv') dataset = data['value'].values dataset = dataset.reshape(-1, 1)
   * **Objective:** Load and preprocess the dataset.
   * **Explanation:** The dataset is read from a CSV file, and the relevant column is extracted and reshaped into a suitable format for training.
2. **Splitting Data into Training and Test Sets**

train\_size = int(len(dataset) \* 0.8)

train, test = dataset[:train\_size], dataset[train\_size:]

* + **Objective:** Split the data into training and test sets.
  + **Explanation:** The data is divided into 80% training and 20% testing.

1. **Data Normalization**

from sklearn.preprocessing import MinMaxScaler scaler = MinMaxScaler(feature\_range=(0, 1)) train\_scaled = scaler.fit\_transform(train) test\_scaled = scaler.transform(test)

* + **Objective:** Normalize the data to ensure it’s in a suitable range for training.
  + **Explanation:** The MinMaxScaler scales the data between 0 and 1, which helps improve the training process. **5. Creating Sequences for RNN Input** def create\_sequences(dataset, step): X, y = [], [] for i in range(len(dataset) - step):

X.append(dataset[i:i + step, 0])

y.append(dataset[i + step, 0])

return np.array(X), np.array(y)

step = 10

X\_train, y\_train = create\_sequences(train\_scaled, step)

X\_test, y\_test = create\_sequences(test\_scaled, step)

X\_train = X\_train.reshape((X\_train.shape[0], X\_train.shape[1], 1))

X\_test = X\_test.reshape((X\_test.shape[0], X\_test.shape[1], 1)) ● **Objective:** Create input-output sequences for training the RNN.

* + **Explanation:** A function create\_sequences generates training samples where step defines the length of each input sequence. This step is crucial for feeding sequential data into the RNN.

1. **Building the RNN Model** model = Sequential()

model.add(SimpleRNN(50, activation='relu', input\_shape=(step, 1))) model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

* + **Objective:** Define the RNN architecture.
  + **Explanation:** The RNN model is built using Sequential(). It consists of a SimpleRNN layer with 50 units and a dense layer for output. The model is compiled using the Adam optimizer and Mean Squared Error (MSE) loss function.

1. **Training the Model** history = model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test)) ● **Objective:** Train the RNN model.
   * **Explanation:** The model is trained for 50 epochs using the training data, and its performance is validated on the test data.
2. **Evaluating the Model Performance**

plt.plot(history.history['loss'], label='train') plt.plot(history.history['val\_loss'], label='test') plt.legend() plt.show()

* + **Objective:** Visualize the training and validation loss.
  + **Explanation:** The training and validation losses are plotted over epochs, allowing us to observe how the model learns over time.

1. **Making Predictions** predictions = model.predict(X\_test) predictions = scaler.inverse\_transform(predictions) y\_test = scaler.inverse\_transform(y\_test.reshape(-1, 1))
   * **Objective:** Generate predictions using the trained model.
   * **Explanation:** The model predicts values on the test set, and the predictions are then transformed back to their original scale.
2. **Visualizing the Results**

plt.plot(y\_test, label='True') plt.plot(predictions, label='Predicted') plt.legend() plt.show()

* + **Objective:** Visualize the model’s predictions against the true values.
  + **Explanation:** A comparison plot shows how well the RNN model predicted the test data.

**GitHubLink:** GitHub Link: https://github.com/princeranjan789