**EX NO:06**

**DATE:**

**RNN ARCHITECTURE FOR TIME SERIES DATA**

**AIM:**

To demonstrate the implementation of a Recurrent Neural Network (RNN) for time series

forecasting using synthetic sine wave data.

**ALGORITHM:**

STEP 01: Import required libraries such as numpy, matplotlib and tensorflow.keras.

STEP 02: Define a function to generate synthetic sine wave sequences for time series data.

STEP 03: Split the data into input sequences (X) and corresponding next-step targets (y).

STEP 04: Reshape input data into the format [samples, timesteps, features] for RNN.

STEP 05: Initialize a Sequential model.

STEP 06: Add a Simple RNN layer with 50 hidden units and tanh activation.

STEP 07: Add a Dense output layer with 1 neuron to predict the next value.

STEP 08: Compile the model using Adam optimizer and mse loss function.

STEP 09: Train the model with training data, specifying batch size, epochs, and validation

split.

STEP 10: Evaluate the model by predicting on sample data and plot training & validation

loss curves.

STEP 11: Plot the training and validation loss against epochs and display the performance

graph.

**CODING:**

# Step 1: Import Libraries

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense

# Step 2: Generate Synthetic Time Series Data (Sine wave)

def generate\_sine\_wave(seq\_length, num\_samples):

X, y = [], []

for \_ in range(num\_samples):

start = np.random.rand() \* 2 \* np.pi

xs = np.linspace(start, start + 3 \* np.pi, seq\_length + 1)

data = np.sin(xs)

X.append(data[:-1])

y.append(data[-1])

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

seq\_length = 50

num\_samples = 1000

X, y = generate\_sine\_wave(seq\_length, num\_samples)

# Reshape for RNN [samples, timesteps, features]

X = X.reshape((num\_samples, seq\_length, 1))

# Step 3: Build RNN Model

model = Sequential([

SimpleRNN(50, activation='tanh', input\_shape=(seq\_length, 1)),

Dense(1) # Predict next value in sequence

])

# Step 4: Compile Model

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

# Step 5: Train Model

history = model.fit(X, y, epochs=20, batch\_size=32, validation\_split=0.2)

# Step 6: Evaluate & Predict

pred = model.predict(X[:10])

print("\nSample Predictions:")

for i in range(5):

print(f"True: {y[i]:.3f}, Predicted: {pred[i][0]:.3f}")

# Step 7: Plot Training Loss

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel("Epochs")

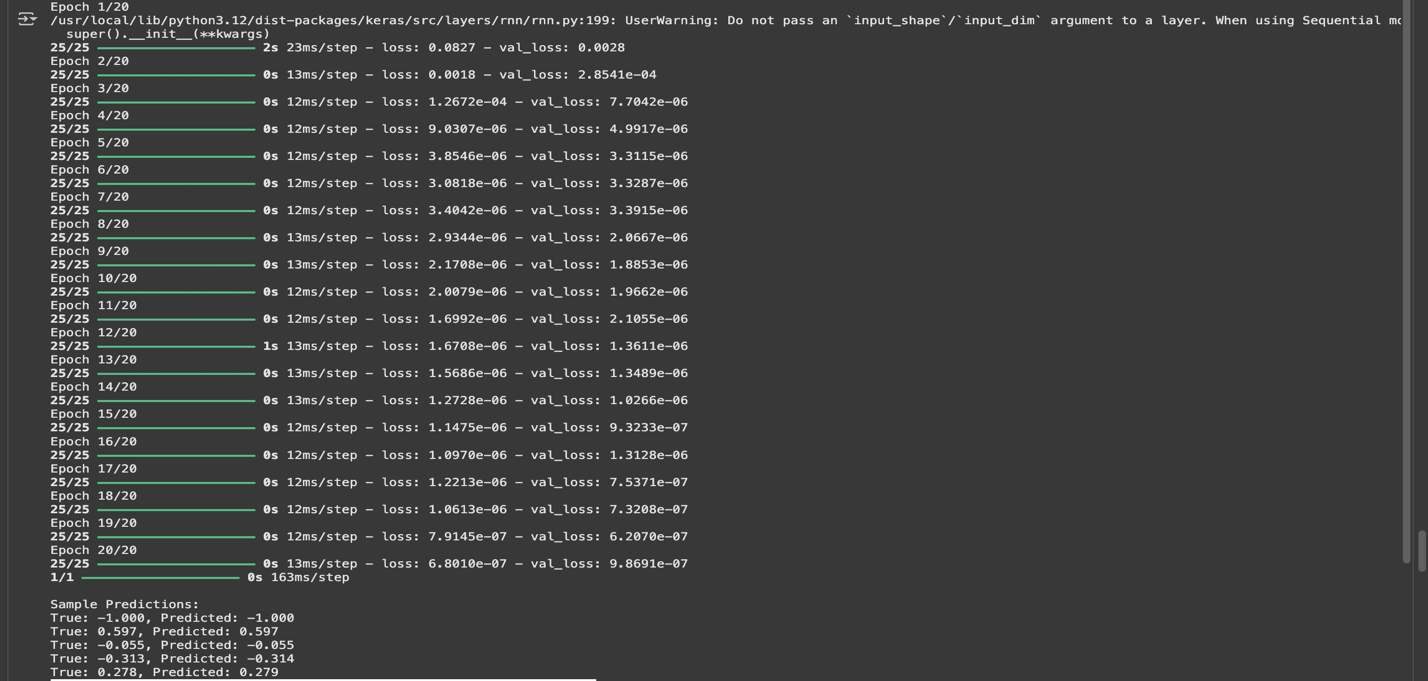
plt.ylabel("Loss (MSE)")

plt.legend()

plt.title("RNN Training Performance")

plt.show()

**Output:**

****

****

|  |  |
| --- | --- |
| COE (20) |  |
| Record (20) |  |
| VIVA (10) |  |
| Total (50) |  |

**RESULT:**

The RNN model was successfully implemented for time series forecasting.The model

learned the sine wave pattern and predicted future values with low error, as shown by

decreasing training and validation loss.