**Exp No:9**

**Date:**

**1. Objective**

The objective is to develop a deep learning model for time series forecasting using LSTM (Long Short-Term Memory) networks, a type of Recurrent Neural Network (RNN) that is well-suited for sequential data.

**2. Why LSTM for Time Series Forecasting?**

* LSTM can learn long-term dependencies in time series data.
* It handles non-linearity better than traditional models like ARIMA.
* Suitable for multi-step forecasting.

**3. Implementation Steps with Code**

**Step 1: Install and Import Required Libraries**

python

CopyEdit

# Install missing libraries

!pip install tensorflow keras scikit-learn pandas numpy matplotlib seaborn

# Import required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

**Step 2: Load and Preprocess Data**

python

CopyEdit

# Load dataset

df = pd.read\_csv("your\_timeseries\_data.csv") # Replace with your dataset

# Convert 'DATE' column to datetime format and set as index

df['DATE'] = pd.to\_datetime(df['DATE'], dayfirst=True, errors='coerce') # Adjust based on format

df.set\_index('DATE', inplace=True)

# Display first few rows

print(df.head())

# Plot the time series data

plt.figure(figsize=(10, 5))

plt.plot(df.index, df['VALUE'], label="Original Data", color='blue')

plt.xlabel("Date")

plt.ylabel("Value")

plt.title("Time Series Data")

plt.legend()

plt.show()

**Step 3: Normalize Data**

python

CopyEdit

# Normalize the data for better model performance

scaler = MinMaxScaler(feature\_range=(0,1))

df['VALUE\_scaled'] = scaler.fit\_transform(df[['VALUE']])

# Convert the data into supervised learning format

def create\_sequences(data, time\_steps=10):

X, y = [], []

for i in range(len(data) - time\_steps):

X.append(data[i:i+time\_steps])

y.append(data[i+time\_steps])

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

# Define time steps (window size)

time\_steps = 10

# Create sequences

X, y = create\_sequences(df['VALUE\_scaled'].values, time\_steps)

# Split into training and testing sets

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

X\_train, y\_train = X[:train\_size], y[:train\_size]

X\_test, y\_test = X[train\_size:], y[train\_size:]

# Reshape input data for LSTM (samples, time steps, features)

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)

print("Training data shape:", X\_train.shape)

print("Testing data shape:", X\_test.shape)

**Step 4: Build and Train LSTM Model**

python

CopyEdit

# Build the LSTM model

model = Sequential([

LSTM(50, activation='relu', return\_sequences=True, input\_shape=(time\_steps, 1)),

Dropout(0.2),

LSTM(50, activation='relu', return\_sequences=False),

Dropout(0.2),

Dense(25),

Dense(1)

])

# Compile the model

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

# Train the model

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=16, validation\_data=(X\_test, y\_test), verbose=1)

# 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")

plt.legend()

plt.title("Training and Validation Loss")

plt.show()

**Step 5: Make Predictions**

python

CopyEdit

# Make predictions

y\_pred = model.predict(X\_test)

# Inverse transform predictions to original scale

y\_test\_inv = scaler.inverse\_transform(y\_test.reshape(-1,1))

y\_pred\_inv = scaler.inverse\_transform(y\_pred)

# Plot actual vs predicted values

plt.figure(figsize=(10, 5))

plt.plot(df.index[-len(y\_test):], y\_test\_inv, label="Actual", color="blue")

plt.plot(df.index[-len(y\_test):], y\_pred\_inv, label="Predicted", color="red")

plt.xlabel("Date")

plt.ylabel("Value")

plt.title("LSTM Model - Time Series Forecasting")

plt.legend()

plt.show()

**Step 6: Evaluate Model Performance**

python

CopyEdit

# Calculate RMSE (Root Mean Squared Error)

rmse = np.sqrt(mean\_squared\_error(y\_test\_inv, y\_pred\_inv))

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

**4. Conclusion**

✅ The **LSTM-based Neural Network** successfully forecasts future time series data.  
✅ The model learns from past data and predicts future trends with high accuracy.  
✅ The **scaling, LSTM layers, and dropout layers** improve model performance.