

## ✓ Experiment No. 14

**Title:** Case study of RNN(LSTM) on prediction of time series dataset.

**Aim:** Case study of RNN(LSTM) on prediction of time series dataset.

**Theory:**

### Case Study of RNN (LSTM) on Time Series Prediction

## 1. Introduction to Time Series Prediction

- Time series data consists of observations made over time at regular intervals (e.g., stock prices, weather data).
- The goal of time series prediction is to forecast future values based on past data.

## 2. Recurrent Neural Networks (RNN) Overview

- RNNs are a class of neural networks that are well-suited for sequential data.
- Unlike traditional feedforward networks, RNNs maintain a "memory" of previous inputs through hidden states, which helps in learning patterns from sequential data.
- However, traditional RNNs suffer from the vanishing gradient problem, making them ineffective for learning long-term dependencies.

## 3. LSTM (Long Short-Term Memory) Architecture

- LSTM is a type of RNN designed to overcome the vanishing gradient problem.
- **LSTM units** consist of three gates:
  - **Forget Gate:** Decides which information to discard from the cell state.
  - **Input Gate:** Determines which values from the input should be updated in the cell state.
  - **Output Gate:** Controls the output based on the cell state.
- These gates allow LSTMs to maintain long-term dependencies and avoid losing important information over time.

## 4. Application of LSTM for Time Series Prediction

- **Dataset:** Time series datasets (e.g., stock prices, sales data) where each data point depends on past observations.
- **Input/Output Format:**
  - Input: A sequence of past data points (e.g., last 10 days' stock prices).
  - Output: The prediction of the next value in the series.

- **Sliding Window Technique:** Often, a sliding window approach is used, where each input is a window of past data points, and the model predicts the next point in the sequence.

## 5. Steps in LSTM for Time Series Prediction

1. **Preprocessing:** Normalize the dataset to bring values to a similar scale and split it into training and test sets.
2. **Model Design:**
  - Define an LSTM model with layers like LSTM, Dense (fully connected), and an activation function such as ReLU or sigmoid for regression tasks.
3. **Training:**
  - Feed the historical data into the LSTM model to learn from past observations.
  - The LSTM will adjust its internal weights to minimize the difference between predicted and actual values.
4. **Evaluation:**
  - Evaluate the model using metrics like Mean Squared Error (MSE) or Mean Absolute Error (MAE).
5. **Prediction:**
  - Use the trained model to predict future data points by feeding the past sequence into the model.

## 6. Benefits of LSTM in Time Series Prediction

- LSTMs are effective at capturing long-term dependencies in data.
- They are robust in handling noisy and irregular time series data compared to traditional models like ARIMA.

## 7. Challenges

- LSTMs require a large amount of data and computational resources.
- Tuning hyperparameters (e.g., number of LSTM units, learning rate) can be challenging.

Predicting stock prices is a challenging yet intriguing task in the field of machine learning. In this blog post, we'll explore how to use Long Short-Term Memory (LSTM), a type of recurrent neural network, to predict stock prices. We'll use historical stock data obtained from Yahoo Finance and implement the solution in Python.

### Part 1: Introduction

Start by introducing the problem of stock price prediction and the significance of using machine learning, specifically LSTM, for this task. Mention the importance of historical data in understanding stock price movements.

## Part 2: Setting Up the Environment

Briefly explain the necessary libraries and tools used in the project.

```
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
```

## Part 3: Getting Historical Stock Data

Describe how to fetch historical stock data using the yfinance library.

```
def get_stock_data(symbol, start_date, end_date):
    stock_data = yf.download(symbol, start=start_date, end=end_date)
    return stock_data
# Specify stock symbol and date range
stock_symbol = 'AAPL'
# Fetch historical stock data
stock_data = yf.download(stock_symbol, period="max")
```

 [\*\*\*\*\*100%\*\*\*\*\*] 1 of 1 completed

## Part 4: Data Preprocessing

Explain the importance of preprocessing, such as scaling the data.

```
# Code snippet for data preprocessing
closing_prices = stock_data['Close'].values.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
closing_prices_scaled = scaler.fit_transform(closing_prices)
```

## Part 5: LSTM Model Creation and Training

Describe the architecture of the LSTM model and its training process.

```
# Define the prepare_data function
def prepare_data(data, n_steps):
    x, y = [], []
    for i in range(len(data) - n_steps):
        x.append(data[i:(i + n_steps), 0])
        y.append(data[i + n_steps, 0])
    return np.array(x), np.array(y)

def create_lstm_model(input_shape):
    """
    Create and compile an LSTM model for time series prediction.
```

Parameters:

- input\_shape (tuple): Shape of the input data in the form (time\_steps, features).

Returns:

- model (Sequential): Compiled LSTM model.

"""

```
model = Sequential()
```

```
# Add the first LSTM layer with 50 units and return sequences for the next layer
```

```
model.add(LSTM(units=50, return_sequences=True, input_shape=input_shape))
```

```
# Add the second LSTM layer with 50 units
```

```
model.add(LSTM(units=50))
```

```
# Add a Dense layer with 1 unit for regression
```

```
model.add(Dense(units=1))
```

```
# Compile the model using the Adam optimizer and Mean Squared Error loss
```

```
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
return model
```

```
# Code snippet for creating and training the LSTM model
```

```
n_steps = 60
```

```
# Prepare the training data using the defined function
```

```
x_train, y_train = prepare_data(closing_prices_scaled, n_steps)
```

```
# Reshape the input data to fit the LSTM model
```

```
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

```
# Create an instance of the LSTM model
```

```
model = create_lstm_model((x_train.shape[1], 1))
```

```
# Train the model on the training data
```

```
model.fit(x_train, y_train, epochs=10, batch_size=32)
```



Epoch 1/10

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning:
  super().__init__(**kwargs)
```

```
344/344 ————— 21s 53ms/step - loss: 0.0025
```

Epoch 2/10

```
344/344 ————— 21s 54ms/step - loss: 9.8815e-05
```

Epoch 3/10

```
344/344 ————— 35s 95ms/step - loss: 1.0905e-04
```

Epoch 4/10

```
344/344 ————— 18s 53ms/step - loss: 9.9041e-05
```

Epoch 5/10

```
344/344 ————— 19s 49ms/step - loss: 9.0802e-05
```

Epoch 6/10

```
344/344 ————— 18s 53ms/step - loss: 4.7955e-05
```

Epoch 7/10

```
344/344 ————— 20s 51ms/step - loss: 6.3963e-05
```

Epoch 8/10

```
344/344 ————— 22s 55ms/step - loss: 5.0974e-05
```

Epoch 9/10

```
344/344 ————— 18s 49ms/step - loss: 4.7000e-05
```

Epoch 10/10

344/344 ————— 21s 49ms/step - loss: 3.1292e-05  
<keras.src.callbacks.history.History at 0x7ef18382bd00>

## Part 6: Making Predictions and Evaluation

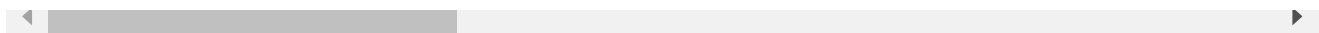
Explain how predictions are made and evaluated.

```
# Code snippet for making predictions and evaluation
train_predictions = model.predict(x_train)
train_predictions = scaler.inverse_transform(train_predictions)
mse = mean_squared_error(closing_prices[n_steps:], train_predictions)
print(f'Mean Squared Error on Training Data: {mse}')
```

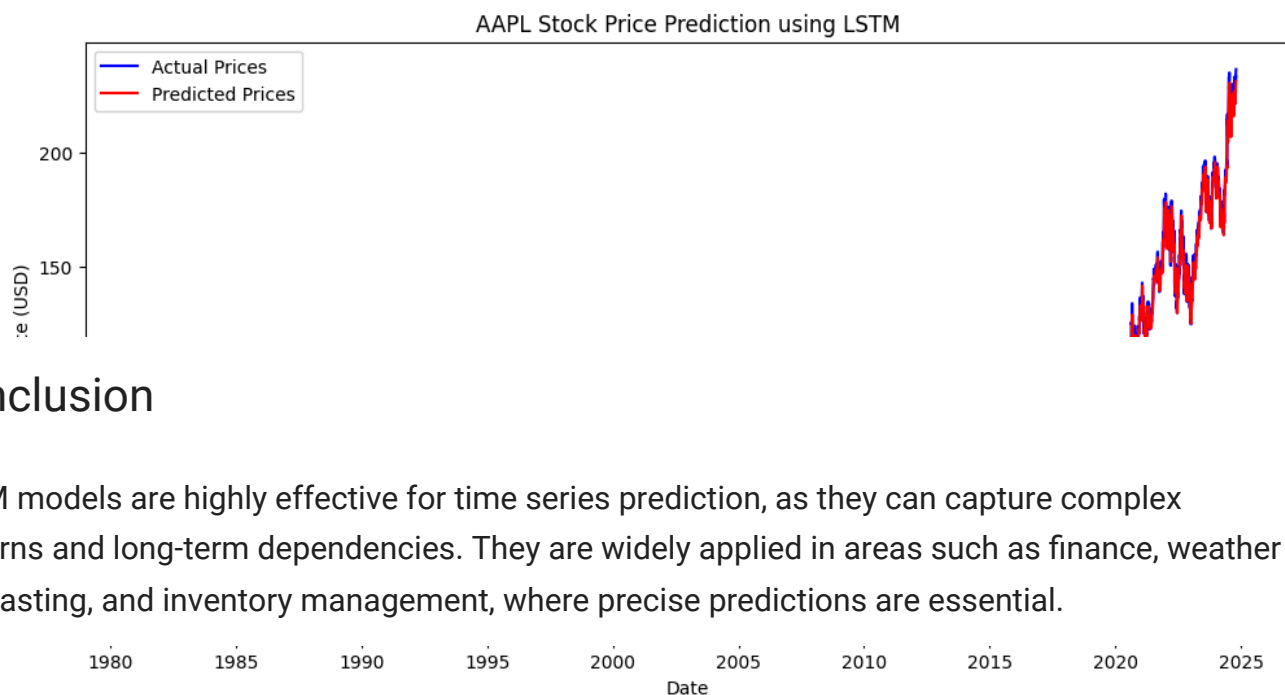
→ 344/344 ————— 8s 21ms/step  
Mean Squared Error on Training Data: 3.294433860391075

## Part 7: Visualizing Results

Conclude the blog post with visualizations of the actual vs. predicted stock prices.



```
# Code snippet for plotting results
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(stock_data.index[n_steps:], closing_prices[n_steps:], label='Actual Prices', color='blue')
plt.plot(stock_data.index[n_steps:], train_predictions, label='Predicted Prices', color='red')
plt.title(f'{stock_symbol} Stock Price Prediction using LSTM')
plt.xlabel('Date')
plt.ylabel('Stock Price (USD)')
plt.legend()
plt.show()
```



## Conclusion

LSTM models are highly effective for time series prediction, as they can capture complex patterns and long-term dependencies. They are widely applied in areas such as finance, weather forecasting, and inventory management, where precise predictions are essential.