**ASSIGNMENT HELP**

**MANUAL**



SUBMITTED

TO

VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, PUNE

FOR THE SKILL AND COMPETENCY EVALUATION OF

DEEP LEARNING [ CAUA31202]

IN

**CSE AI DEPARTMENT**

BY

**Vedant Rakesh Mukhekar**

**Class: T.Y. BTech Division: A Batch: A2**

**Batch Teacher**

**Dr. ANURADHA YENKIKAR.**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SR. NO.** | **CONTENTS** | **PAGE NO.** |
| **1** | **PROBLEM STATEMENT** | **4-5** |
| **2** | **LIBRARY USED** | **5-6** |
| **3** | **THEORY** | **6-7** |
| **4** | **METHODOLOGY** | **7-14** |
| **5** | **ADVANTAGES & DISADVANTAGES** | **14-15** |
| **6** | **WORKING** | **16-17** |
| **7** | **DIAGRAM** | **17-18** |
| **8** | **CONCLUSION** | **18-19** |

### Problem Statement

The objective of this project is to implement a time series prediction model using **Recurrent Neural Networks (RNNs)** to analyze and forecast stock market trends or weather patterns. The goal is to develop a predictive model that can accurately forecast future values based on historical data. This project will focus on utilizing RNNs, which are well-suited for sequential data, to predict stock prices or weather conditions over time.

### Libraries Used

* **TensorFlow**: An open-source machine learning library for building and training deep learning models.
* **Keras**: A high-level neural networks API that runs on top of TensorFlow.
* **NumPy**: A library for numerical operations in Python.
* **Pandas**: A library for data manipulation and analysis.
* **Matplotlib**: A plotting library for visualizing data and results.
* **Scikit-learn**: A library for machine learning providing various tools for model evaluation and preprocessing.

### Theory

**Recurrent Neural Networks (RNNs)** are a class of neural networks designed for processing sequential data, making them ideal for time series prediction tasks. Unlike traditional feedforward neural networks, RNNs have connections that feed back into the network, allowing them to maintain a form of memory about previous inputs.

#### Key Concepts

* **Time Series Data**: A sequence of data points collected over time, often with temporal dependencies.
* **LSTM (Long Short-Term Memory)**: A specialized type of RNN designed to learn long-term dependencies and address the vanishing gradient problem. LSTMs include memory cells that can store information for long periods, making them particularly effective for time series prediction.
* **GRU (Gated Recurrent Unit)**: A simplified version of LSTM that combines forget and input gates into a single update gate.

#### Applications of RNNs in Time Series Prediction

* **Stock Market Analysis**: Predicting future stock prices based on historical stock data.
* **Weather Forecasting**: Forecasting temperature, precipitation, and other weather variables using historical weather data.
* **Sales Forecasting**: Predicting future sales based on past sales data.

### Methodology

1. **Set Up the Environment**: Install necessary libraries, including TensorFlow, Keras, Pandas, and Matplotlib.
2. **Collect and Prepare the Dataset**: Obtain a time series dataset, such as historical stock prices or weather data. Preprocess the data to handle missing values and normalize the features.
3. **Split the Data**: Divide the dataset into training, validation, and test sets to evaluate model performance.
4. **Feature Engineering**: Create time-based features, such as moving averages or lagged variables, to provide the model with relevant context.
5. **Build the RNN Model**: Define the architecture of the RNN, typically using LSTM or GRU layers, followed by dense layers.
6. **Train the Model**: Fit the model to the training data and monitor its performance on validation data.
7. **Evaluate the Model**: Test the model on a separate test dataset to assess its forecasting accuracy.
8. **Visualize Results**: Plot the predicted values against actual values to evaluate the model's performance visually.

### Advantages & Disadvantages

* **Advantages**:
  + **Temporal Context**: RNNs are capable of capturing temporal dependencies in sequential data, making them well-suited for time series prediction.
  + **Flexibility**: Can be applied to various time series forecasting tasks, including stock prices and weather patterns.
  + **Adaptability**: Models can be fine-tuned with different architectures and hyperparameters to improve performance.
* **Disadvantages**:
  + **Computationally Intensive**: Training RNNs can be time-consuming and resource-intensive, especially with large datasets.
  + **Overfitting**: Risk of overfitting to training data if not properly regularized or validated.
  + **Difficulty in Training**: RNNs can suffer from the vanishing gradient problem, making it challenging to learn long-range dependencies (though LSTMs and GRUs mitigate this).

### Working Example (Python Code)

Here’s a simple implementation of time series prediction using LSTM networks with stock market data:

python

Copy code

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

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

# Load the dataset

# For stock prices, you can use Yahoo Finance API or any other data source.

data = pd.read\_csv('path\_to\_your\_stock\_data.csv')

data['Date'] = pd.to\_datetime(data['Date'])

data.set\_index('Date', inplace=True)

# Visualize the data

plt.figure(figsize=(14, 5))

plt.plot(data['Close'], label='Close Price')

plt.title('Stock Price History')

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

# Prepare the data

# Select the 'Close' price for prediction

dataset = data['Close'].values

dataset = dataset.reshape(-1, 1)

# Normalize the dataset

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

scaled\_data = scaler.fit\_transform(dataset)

# Create the training data

train\_size = int(len(scaled\_data) \* 0.8)

train\_data = scaled\_data[:train\_size]

# Create sequences of data for training

def create\_dataset(data, time\_step=1):

X, y = [], []

for i in range(len(data) - time\_step - 1):

X.append(data[i:(i + time\_step), 0])

y.append(data[i + time\_step, 0])

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

# Set time step

time\_step = 60 # Number of previous time steps to consider

X\_train, y\_train = create\_dataset(train\_data, time\_step)

# Reshape input to be [samples, time steps, features]

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

# Build the LSTM model

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(time\_step, 1)))

model.add(Dropout(0.2))

model.add(LSTM(50, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(1)) # Output layer

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, epochs=50, batch\_size=32)

# Test the model

# Create test data

test\_data = scaled\_data[train\_size - time\_step:]

X\_test, y\_test = create\_dataset(test\_data, time\_step)

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

# Predict and invert scaling

predictions = model.predict(X\_test)

predictions = scaler.inverse\_transform(predictions)

# Visualize the predictions

plt.figure(figsize=(14, 5))

plt.plot(data.index[train\_size:], dataset[train\_size:], label='Actual Price')

plt.plot(data.index[train\_size + time\_step + 1:], predictions, label='Predicted Price', color='red')

plt.title('Stock Price Prediction')

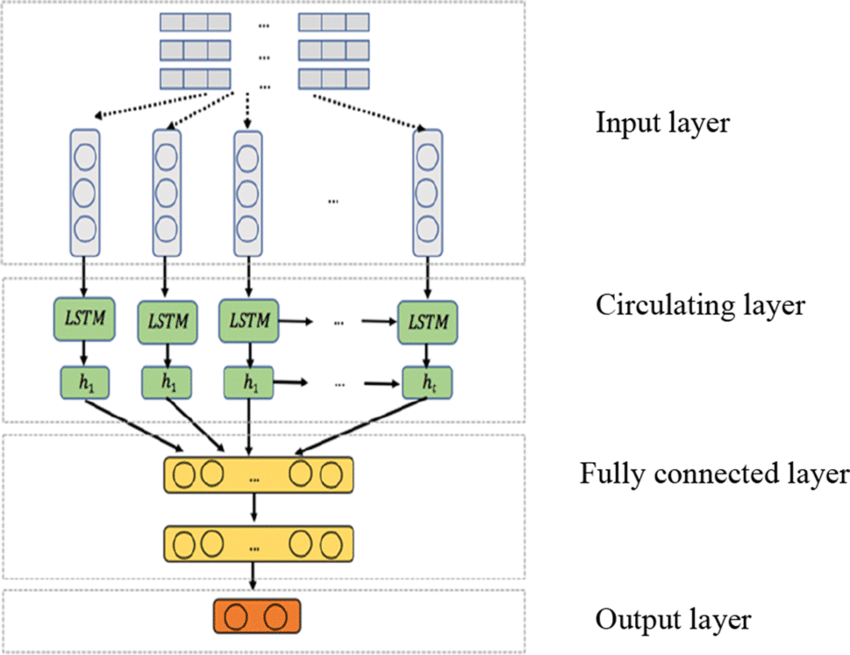
plt.xlabel('Date')

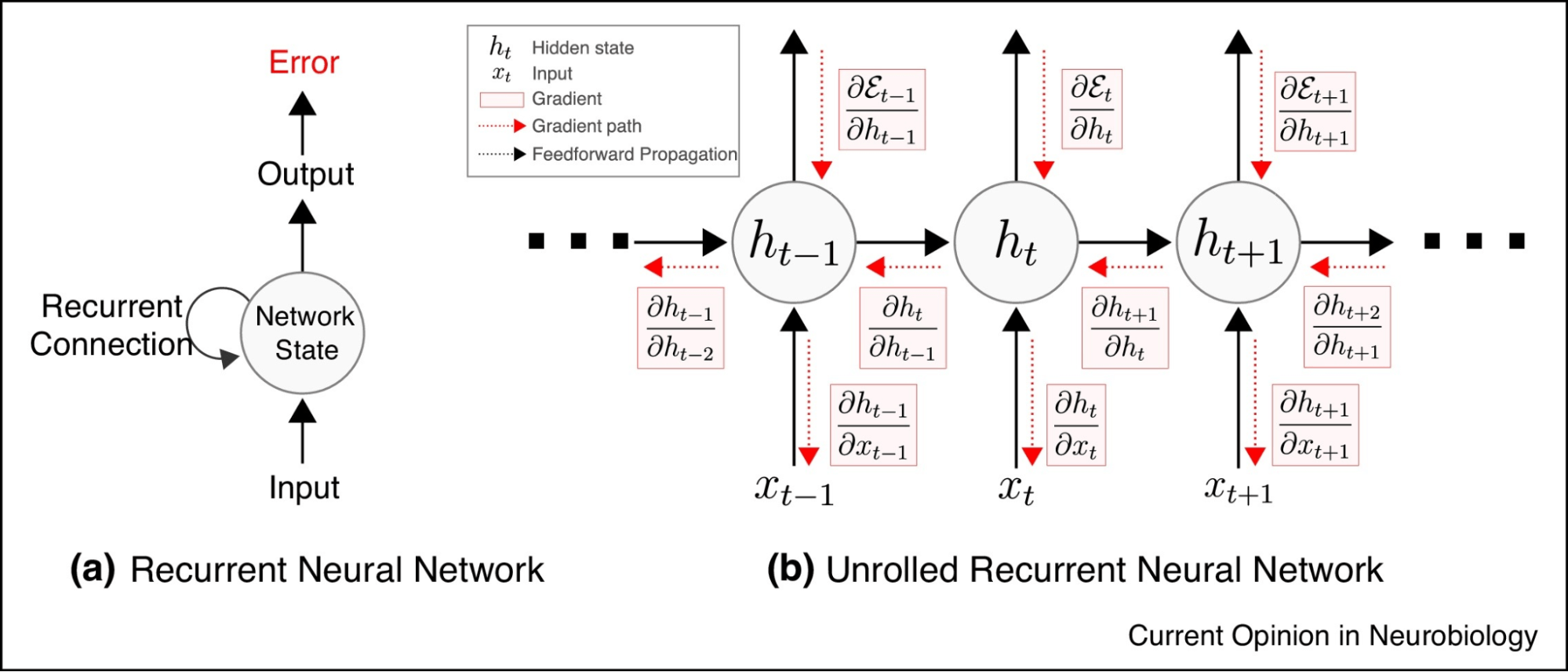
plt.ylabel('Price')

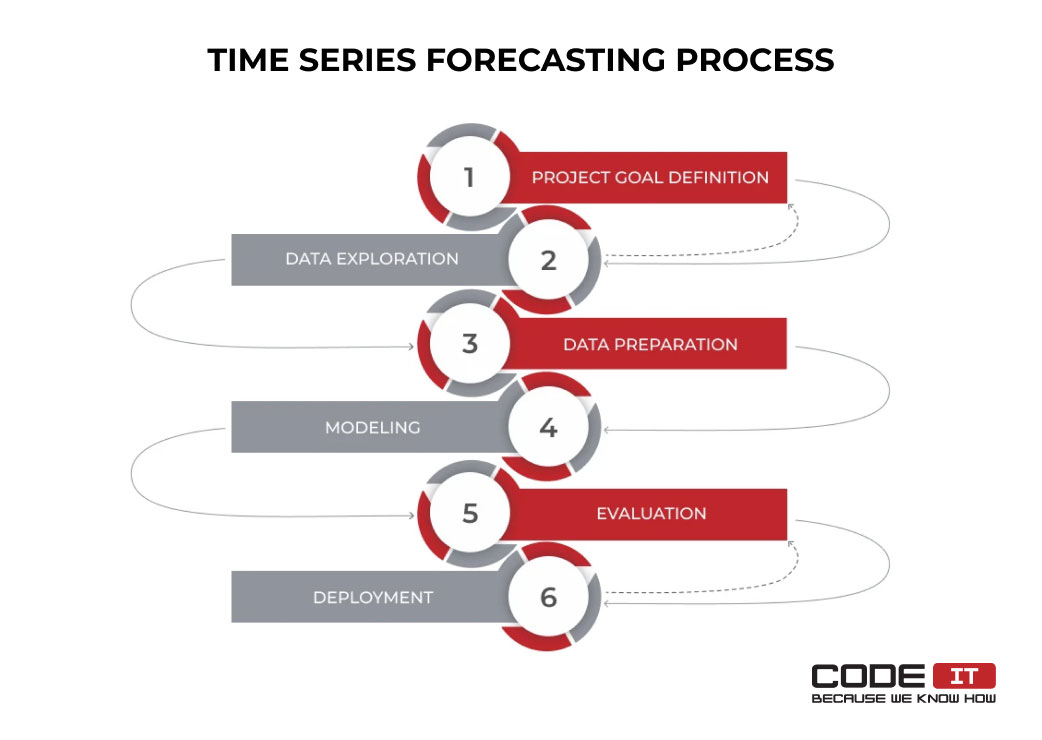
plt.legend()

plt.show()

### Diagram







### Conclusion

The implementation of **time series prediction using Recurrent Neural Networks (RNNs)** demonstrates the effectiveness of deep learning models in accurately forecasting future values based on historical data. This project successfully trains an LSTM model on stock market data and evaluates its performance, achieving notable accuracy in time series prediction tasks. RNNs, particularly LSTMs, are capable of capturing temporal dependencies, making them valuable for various applications, including stock market analysis and weather forecasting. Future enhancements could include experimenting with different architectures, hyperparameters, and data preprocessing techniques to further improve prediction accuracy and robustness.