Stock-Price-Prediction

AIM

To develop a **Recurrent Neural Network (RNN)** model for predicting stock prices using historical data.

Problem Statement and Dataset

Stock price prediction is a challenging task due to the non-linear and volatile nature of financial markets. Traditional methods often fail to capture complex temporal dependencies. Deep learning, specifically **Recurrent Neural Networks (RNNs)**, can effectively model time-series dependencies, making them suitable for stock price forecasting.

- **Problem Statement**: Build an RNN model to predict the future stock price based on past stock price data.
- **Dataset**: A stock market dataset containing **historical daily closing prices** (e.g., Google, Apple, Tesla, or NSE/BSE data). The dataset is usually divided into **training and testing sets** after applying normalization and sequence generation.

Design steps

Step 1:

Import required libraries such as `torch` , `torch.nn` , `torch.optim` , `numpy` , `pandas` , and `matplotlib` .

Step 2:

Load the dataset (e.g., stock closing prices from CSV), preprocess it by **normalizing** values between 0 and 1, and create input sequences for training/testing.

Step 3:

Define the **RNN model architecture** with an input layer, hidden layers, and an output layer to predict stock prices.

Step 4:

Compile the model using **MSELoss** as the loss function and **Adam optimizer**.

Step 5:

Train the model on the training data, recording training losses for each epoch.

Step 6:

Test the trained model on unseen data and visualize results by plotting the **true stock prices vs. predicted stock prices**.

Program

Name: Keerthivasan K S

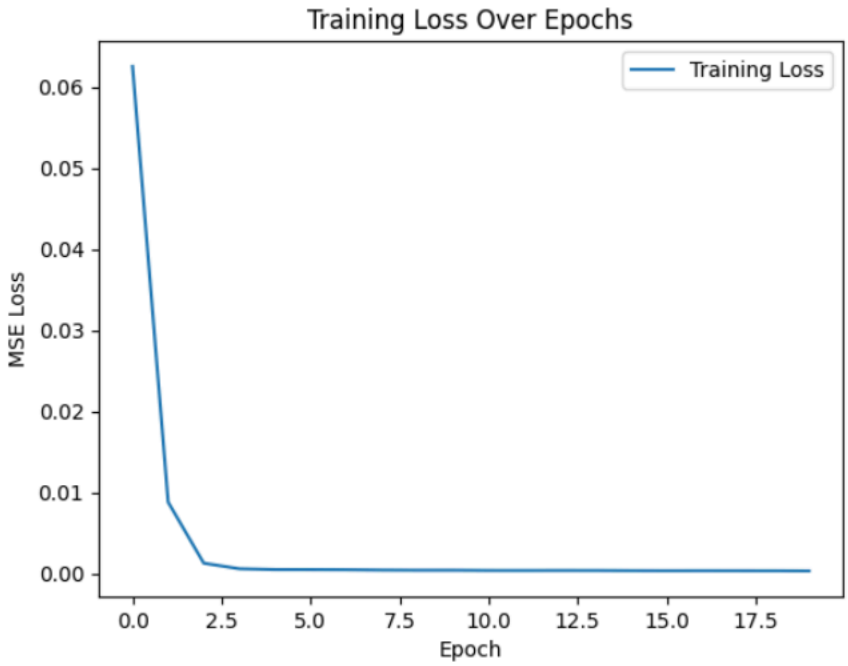
Register Number: 212224230120

```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt

# Define RNN Model
class RNNModel(nn.Module):
    def __init__(self, input_size=1, hidden_size=64, num_layers=2, output_size=1):
        super(RNNModel, self).__init__()

        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.rnn(x)
        out = self.fc(out[:, -1, :])  # last time step
        return out


# Initialize Model, Loss, Optimizer
model = RNNModel()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training Loop
epochs = 20
model.train()
train_losses = []
for epoch in range(epochs):
    epoch_loss = 0
    for x_batch, y_batch in train_loader:
        x_batch, y_batch = x_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        outputs = model(x_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
    train_losses.append(epoch_loss / len(train_loader))
    print(f"Epoch [{epoch+1}/{epochs}], Loss: {train_losses[-1]:.4f}")

# Plot Training Loss
plt.plot(train_losses, label="Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Output

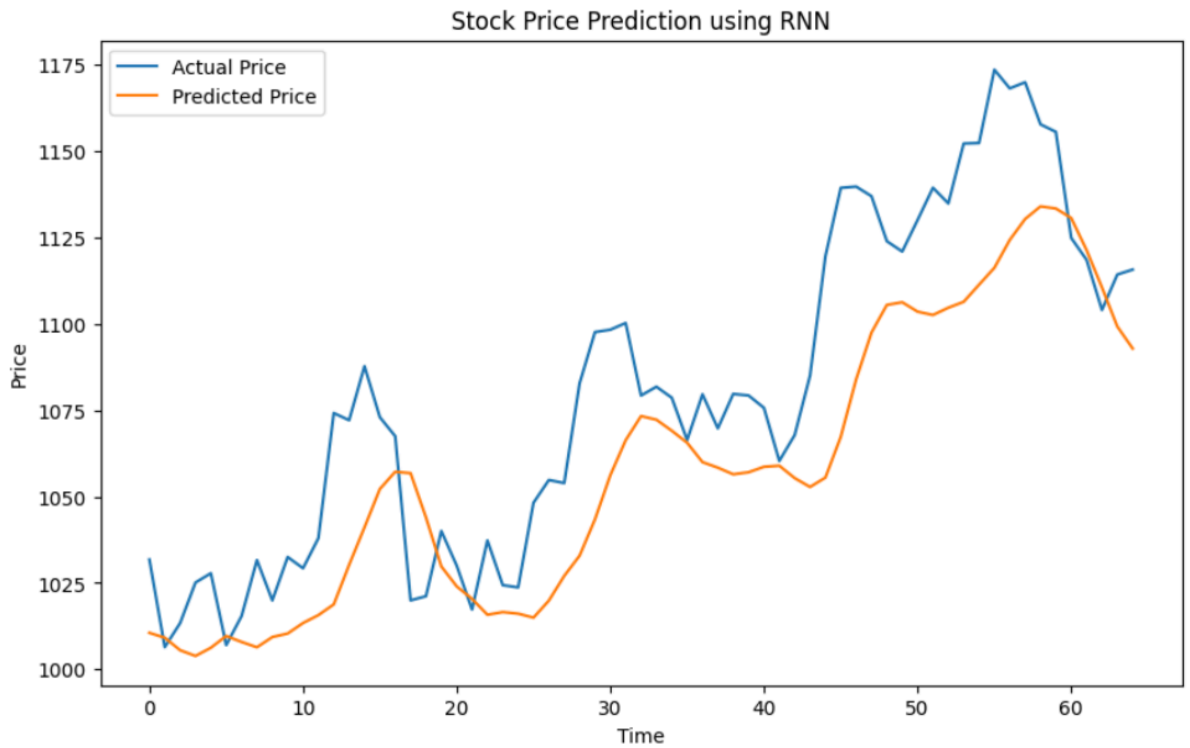
True Stock Price, Predicted Stock Price vs Time

```
Name: Keerthivasan K S
Register Number: 212224230120
```



Predictions

```
Name: Keerthivasan K S
Register Number: 212224230120
```




```
Predicted Price: [1092.824]
Actual Price: [1115.65]
```

Result

The **RNN model** was successfully implemented for **stock price prediction**.