Aim: Implement an RNN using Keras using LSTM and any other layer

Problem Description: Predicting the direction of the stock market is always hard and unpredictable. This program is an attempt to read Google stock data and make a prediction of the price based on the day.

Data:

The data for this task consists of 2 files, one for training and the other for test. The data consists of opening, closing, maximum and minimum stock prices of the day and the number of shares traded (Volume). The dataset is obtained from Kaggle, which got it from https://finace.yahoo.com.

Link: https://www.kaggle.com/ptheru/googledta

General Idea:

The data preprocessing involves normalization and creating a structure with 100 time steps and 1 output. The RNN model consists of 1 SimpleRNN layer, 3 LSTM layers and a dense output layer. We use an Adam optimizer for adaptive learning rate and the loss function is 'mean_squared_error'. Compile the model and load test file. Follow the same preprocessing steps. Predict the stock price using the model.

The model gave good results with a loss of about 4×10^{-4}

Procedure:

Step 1: Import required modules

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import SimpleRNN
from keras.layers import GRU
```

Step 2: Reading the dataset

```
dataset_train =
  pd.read_csv("../input/gooogle-stock-price/Google_Stock_Price_Train.csv")
```

Step 3: Data Preparation

```
# 3.1: Normalization of dataset
training_set = dataset_train.iloc[:,1:2].values
sc = MinMaxScaler(feature_range = (0,1))
training_set_scaled = sc.fit_transform(training_set)
plt.plot(training_set_scaled)
plt.title('Google stock price')
plt.xlabel('time [days]')
plt.ylabel('price')
plt.show()
```

```
# 3.2: Divide the dataset into training and validation sets
X_train = []
y_train = []
t = 100  # timesteps
l = len(training_set_scaled)
for i in range(t,l):
        X_train.append(training_set_scaled[i-t:i, 0])
        y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
X_train,X_test,y_train,y_test=train_test_split(X_train,y_train,test_size=0.3)

# 3.3: Reshaping
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1],1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1],1))
```

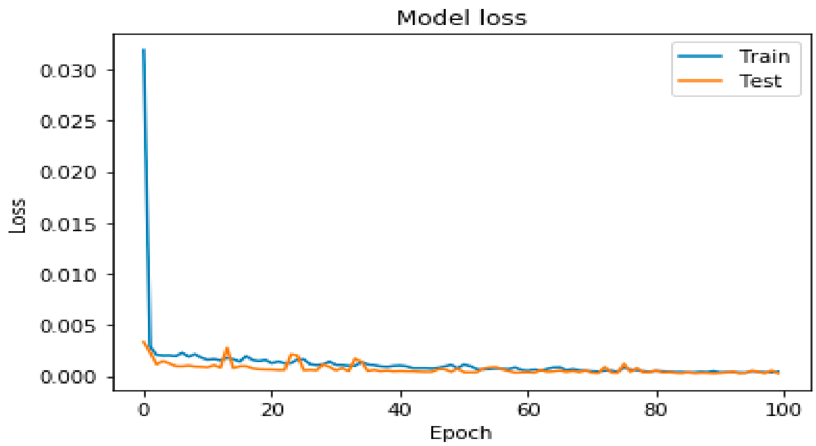
Step 4: Creating the model

```
model = Sequential()
model.add( SimpleRNN (units = 128, return_sequences = True, input_shape =
                                    (X_train.shape[1], 1)))
model.add(LSTM(units = 64, return_sequences = True))
model.add(LSTM(units = 64, return_sequences = True))
model.add(LSTM(units = 32, return_sequences = True))
model.add(GRU(10))
model.add(Dense(units = 1))
```

Step 5: Training the model

```
# 5.1: Compiling the RNN
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
# 5.2: Fitting the RNN to the Training set
history=model.fit(X_train, y_train, epochs = 100, batch_size = 32, validation_data =
(X_test,y_test))
```

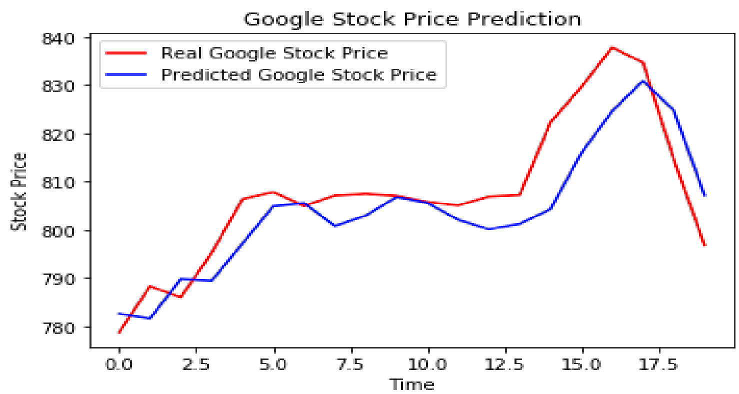
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upperleft')
plt.show()
```



Step 6: Testing the model

```
# 6.1: Read the data and then perform normalization
dataset_test =
                    pd.read_csv("../input/gooogle-stock-price/Google_Stock_Price_Test.csv")
real_stock_price = dataset_test.iloc[:,1:2].values
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), axis = 0)
inputs = dataset_total[len(dataset_train)- t: ].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
#  6.2: Creating a data structure with 100 timesteps and 1 output
X_test = []
m = len(inputs)
for i in range(t,m):
          X_test.append(inputs[i-t:i, 0])
X_test  = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1 ))
```

```
# 6.3: Prediction
predicted_stock_price = model.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
plt.plot(real_stock_price, color = 'red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock Price')
plt.title("Google Stock Price Prediction")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```



Observations:

1. Stock price is complex and unpredictable and hence, lesser accuracy on test results.
2. Time required increases with increase in number of layers.
3. A model with a more complex architecture can classify better.
4. LSTMs are prone to overfitting and it is difficult to apply the dropout.