Hyunjae Cho
Machine Learning II
Final Report
May, 2nd, 2021

Recurrent Neural Network for Bitcoin Prediction

INTRODUCTION

In January 2020, COVID-19 has started and spread all over the world, and the stock market and local economy have crushed. The Dow Jones and Standard & Poor's showed that U.S. share values dropped by 20% since mid-March 2020 (Machmuddah, 2020). It took more than half a year to recover the economy and stock shares fall dragged young 20-30s people's interests. Robinhood, a stock and cryptocurrency trading app, gained 3.1 million users, half of them first-time investors in the first quarter of 2020 (Gao, 2020). Likewise, young people became interested in stocks and bitcoins while taking the risks of losing money as well. In this project, recurrent neural network (RNN), Long Short Time Memory (LSTM), and Gated Recurrent Unit (GRU) models will be used to predict Bitcoins and these models would provide insights to the new investors. In this project, deep learning-driven models are built and compared to find the best models for the prediction. The code is available at GitHub.

DATA

The data obtained from Polenix, a cryptocurrency exchange website and the Poloniex package has been uploaded to PyPI. I requested a private API from Polenix to import the current Bitcoin prices. Private API allows customers to read and write operations on the account, but I just imported Bitcoin prices from the websites. The starting date of Bitcoin is 2014 and the ending date is the most current date, which makes the dataset increase daily. Therefore, I exported the downloaded data to the local file and used it for the data analysis. The data consisted of open, close, low, and high prices of each day, and close prices were used for the modeling.

Experimental Setup

For the data preprocessing, MinMax scaler, a skit-learn library, is used to normalize the data, and the data converted from 0 to 1. Most importantly, matrix formatted data should be reshaped into the array before the transformation. Train and test dataset should be tuple and input size should be the (length, features). If the input size and output size does not match, it will return the errors in the model. Also, data preprocessing is required for the forecasting. I tested both normalized and raw datasets for the prediction, and preprocessed dataset had significant improvements on the predicted values as figure 1 shown below. In the model, the number of iterations was 10, the learning rate was set to 0.001. The hidden layers of each network used 3 layers and adding more hidden layers turned out to be an overfitting result. The tanh used as the activation function and MSE (Mean Squared Error) used as the loss function.

$$X - std = (X - X_{min})/(X_{max} - X_{min})$$

$$X_{scaled} = X_{std}(max - min) + min \cdots \text{ normalization equation}$$

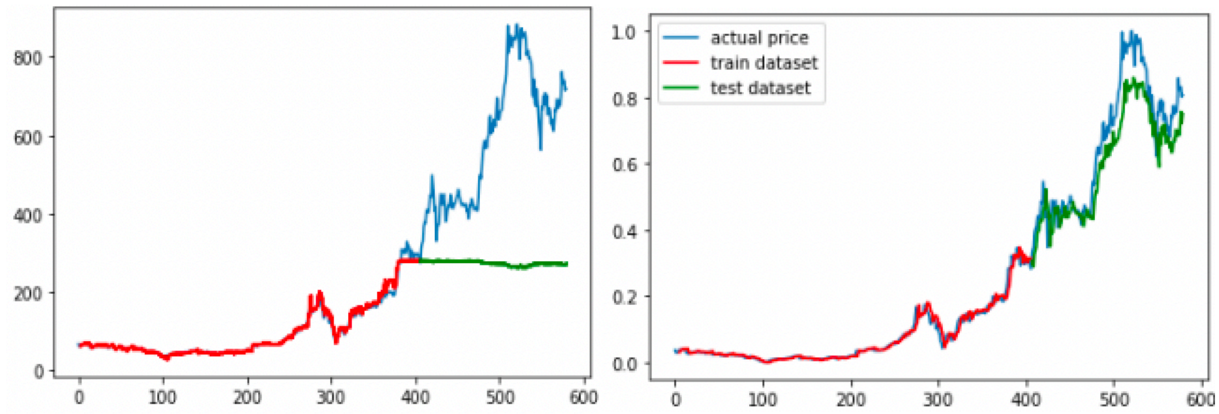

Figure1. The plot on the left side is raw data prediction and the plot on the right is preprocessed dataset prediction.

METHODS

The first method was Vanilla RNN, which passes information from past steps to the next steps, it is one of the well-known methods that suitable for stock prediction. Using the SimpleRNN and made multiple layers, and tanh activation function was used, which has a similar property of logistic regression, and it is useful in the forecasting model as it transforms the output from 0 to 1. As the sigmoid has the similar property of tanh, both sigmoid and tanh activation functions were used. The tanh returned smaller RMSE value, and the model built with tanh activation function. If other activation functions are used, outputs could be infinity or negative value. Additionally, as RNN hidden nodes are connected by directional edges to form the sequences, it can accept inputs and outputs regardless of the length of the sequence.

The next method was the Long Short memory network (LSTM), which was derived from the RNN model. LSTM addresses the vanishing gradient problem, when the gradient gets too small or large (Yamak, 2019). By adding cell state to the existing vanilla RNN and controls how much past information will be retained. Although most of RNNs architecture have hidden layers, hidden layers are replaced with LSTM cells in the LSTM architecture (Selvin, 2017). LSTM cells consist of an input gate that consists of input, a state gate that runs through the entire network and has the ability to add or remove information with the help of gate and forget gate, which decides the fraction of the information to be allowed (Selvin, 2017). The gate vector equations are retrieved below from Zhang, 2019 paper. Lastly, the output gate will return the output generated by LSTM, and my model will return the prediction using the result of the sigmoid activation function. Additionally, Adam parameter optimization was used to continuously update the weight of the neural network (Zhang, 2019), and the network later and batch size are tuned to select the optimal one. Shortly, LSTM model for the Bitcoin prediction was set with Adam parameter with 0.001 learning rate.

$$f_t = \delta\left(W_f * [h_{t-1}, x_t] + b_f\right) \cdots \text{forget gate vector}$$
$$i_t = \delta(W_t * [h_{t-1}, x_t] + b_i) \cdots \text{input gate vector}$$
$$c_t = tanh(W_c * [h_{t-1}, x_t] + b_c) \cdots \text{cell state vector}$$
$$o_t = \delta(W_o * [h_{t-1}, x_t] + b_o) \cdots \text{output gate vector}$$
$$h_t = o_t * tanh(c_t) \cdots \text{output vector}$$

Lastly, GRU is a modification of LSTM and it consists of two gates: reset gate and update gate. Update gate is responsible for deciding the degree of historic memory which should be maintained by the node and the reset gate determines how to combine the current input in the state with the historic memory (Sethia, 2018). As the GRU consists of two gates, it is less complex and LSTM and the GRU model created fewer parameters than LSTM. To compare three models, the tanh activation function was used, and the model complied with mean square error. When the model was compiled with categorical_crossentropy, the loss function returned negative values.

$$r_t = \sigma(x_t U^r + h_{t-1} W^r) \cdots \text{reset gate}$$
$$z_t = \delta(x_t U^z + h_{t-1} W^r) \cdots \text{update gate}$$
$$h_t = (1 - z) * k + z * h_{t-1} \cdots \text{ hidden state}$$
$$k = tanh(x_t U^k + (h_{t-1} r) W^k) \cdots \text{cell state}$$

RESULTS

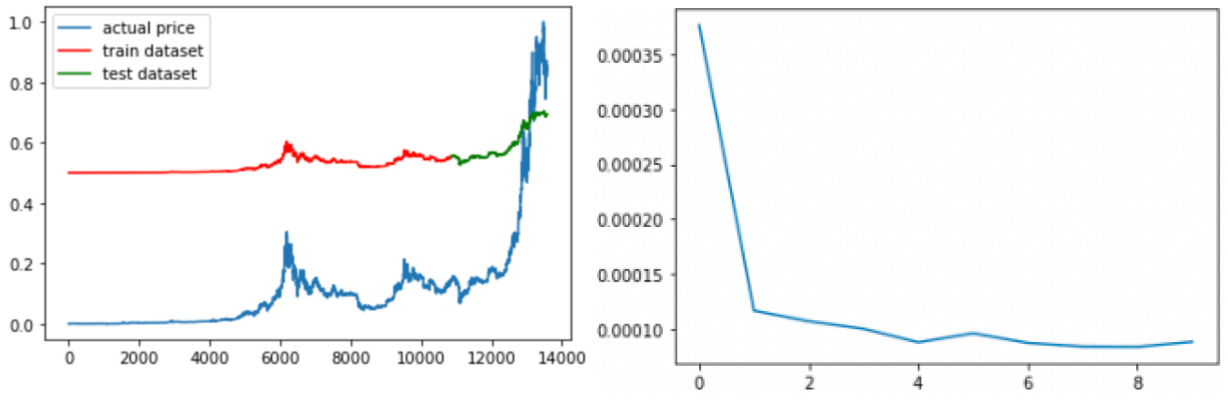

Figure2 – RNN train/test dataset and the loss function plot

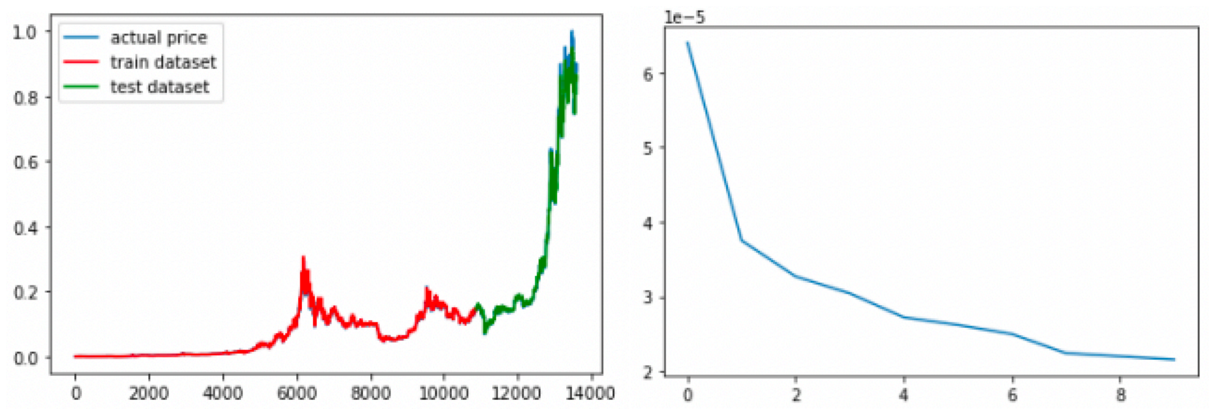

Figure3 – LSTM train/test dataset and loss function plot

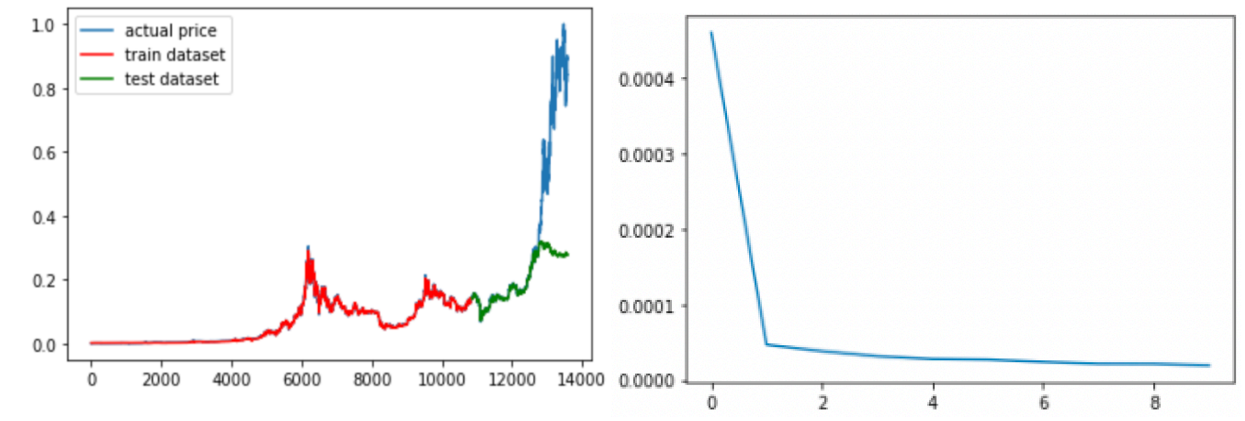

Figure4- GRU train/test dataset and loss function plot

	RNN	LSTM	GRU
Time	181.1197 seconds	137.345 seconds	310.1849 seconds
RMSE	0.3239	0.0158	0.2679

I created the model using both Torch and Keras. Throughout the semester, I learned Torch is very suitable in the deep learning field. Compared to Keras, Torch returns much faster results. However, Torch's modeling was not good as Keras's. Toch forward and backward loss function made the model overfitting and resulted in poor results. Therefore, I decided to use Keras modeling for the report, but the Torch model has been uploaded in GitHub as well.

The window size was 3 days. The window size is a parameter in the time series modeling that represents next-day prediction values based on the window size. The train dataset was 10864 and the test dataset was 2716. As we can compare the training and testing dataset results above, the RNN train and test dataset are apparently different from the actual values. Based on the research, it might be caused by long-term dependency, which means long sequence addressees all the information from the past state and resulted in poor prediction.

As the plot Figre2 – 4 shows, loss values are decreasing as the dataset is trained. Also, the plot shows that the LSTM training dataset predicts the dataset close to the actual dataset. GRU is a less complex algorithm that consists of a reset gate and update gate. GRU expected to have a shorter time to return the results but LSTM had the shortest time. For the evaluation, RMSE measured the average squared difference between the predicted values and the actual values. RMSE is an absolute measure of fit, which means the lower value of RMSE would be the best fitted model. Therefore, based on the figure and the table, we can conclude that LSTM is the best fitted model for the Bitcoin prediction.

SUMMARY

In this paper, three deep learning models; Vanilla RNN, LSTM, and GRU were used to predict Bitcoin. The data transformed into a 0 to 1 value using Minmax Scale, and it proved significant improvement in the prediction. RNN values are disparate from the actual values, but LSTM train dataset are close to the actual value. The reason for the significant improvement was that LSTM was used to address the vanishing gradient issue and to retain long-term dependency. Among the three models, LSTM had the fastest and lowest loss values, and it seems LSTM is the most efficient model for the Bitcoin prediction.

In the future, I would like to test more variety model in the GPU environment and test the time series model with higher window size. While using the Google Cloud Platform for the final project, I used all credits and I needed to run the model with CPU, which took longer time to evaluate the check the differences between the models. For example, when I set the 100 epochs, it took 50 minutes to run the one model. Google Collab offers one GPU system, but it loses the history when the network disconnected. However, throughout the project, I could learn how to build the model based on the logics and which parameters and activation functions would be useful for the prediction model.

REFERENCE

Z. Machmuddah, St.D Utomo, E. Suhartono, S. Ali, and W. Ghulam, Stock Market Reaction to COVID-19: Evidence in Customer Goods Sector with the Imlication for Open Innovation (2020), Journal of Open Innovation, 6, 99, doi. 10.3390

M. Gao, College Students are Buying Stocks – but Do They Know What They're Doing? (2020), CNBC, https://www.cnbc.com/2020/08/04/college-students-are-buying-stocks-but-do-they-know-what-theyre-doing.html

T. Zhang, S. Song, L Ma, S. Pan, L. Pan, Research on Gas Concentration Prediction Models Based on LSTM Multidimensional Time Series (2019), Energies, 12(1): 161, doi.10.3390

P T. Yamak, L. Yuigian, and P.K Gadosey, A Comparison Between ARIMA, LSTM, and GRU for Time Series Forecasting

S. Selvin, Rr. Vinayakumar, E. A. Gopalakrrishnan, and P. Soman, Stock Price Preidctions Using LSTM, RNN, and CNN Sliding Window Model (2017). International Conference on Advances in Computing, Comminucations and Informatics (ICACCI), pp 1643-1647, doi:10.1109

A. Sethia, P. Raut, and D. Sanghiv, Application of LSTM, GRU and ICA for Stock Price Prediction (2018), Smart Innovation, Systems and Technologies, pp 479 – 487, doi:10.1007

Code Reference:
https://github.com/emadRad/lstm-gru-pytorch/blob/master/lstm_gru.ipynb
https://blog.floydhub.com/gru-with-pytorch/
https://stackabuse.com/time-series-prediction-using-lstm-with-pytorch-in-python/
https://coding-yoon.tistory.com/131

Appendix

```
# ==================================================================
# Import libraries
# ==================================================================
import matplotlib.pyplot as plt
import yfinance as yf
import numpy as np
import json
import os
import pandas as pd
from urllib.request import urlopen
import urllib
import requests
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

# keras model
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Activation, Flatten,Reshape
from keras.layers import Conv1D, MaxPooling1D
from keras.utils import np_utils
from keras.layers import LSTM, LeakyReLU
from keras.callbacks import CSVLogger, ModelCheckpoint
import tensorflow as tf

import time

# ==========================================================================
# Open Poloniex with the API Key
# ==========================================================================
#pip install Poloniex

# from poloniex import Poloniex
# import os
# polo = Poloniex()
# API_KEY = "H1LNX332-FDI51G65-O5QGGEKB-Y0KPIHQV"
# API_SECRET =
"0900c0326db91ed0c504eaa12323210005d64f5846e9331a3eac876a2af902ae2f80f65dda626c39
36ac9158dad6926fbd58d74931a158a7b6c41e0c07b52388"

# api_key = os.environ.get(API_KEY)
# api_secret = os.environ.get(API_SECRET)
# polo = Poloniex(api_key, api_secret)
# ticker = polo.returnTicker()['BTC_ETH']
# #https://pypi.org/project/poloniex/

# print(ticker)

# #Import Poleniex data
# df =
requests.get('https://poloniex.com/public?command=returnChartData&currencyPair=USDT_BT
C&start=1405699200&end=9999999999&period=14400')
# df_json = df.json()
# df2 = pd.DataFrame(df_json)
# df2.head()
# df2.to_csv('project_df.csv', encoding = 'utf-8')

# ==========================================================================
# Dataset
# ==========================================================================
# import the data from the exported dataset
df2 = pd.read_csv('project_df.csv', index_col = 0)

# ==========================================================================
# Plot the dataset
# ==========================================================================
# Plot the data and define it with a function
def plot_data(dataset):
  plt.figure(figsize=(6,6))
  plt.plot(dataset['open'].values, 'g--',color = 'green', label = 'open')
  plt.plot(dataset['close'].values, 'y--', color = 'yellow', label = 'close')
  plt.plot(dataset['low'].values, color = 'red', label = 'low')
  plt.plot(dataset['high'].values, label = 'high')
  plt.legend()
  plt.xlabel('time')
  plt.ylabel('$ price')
  plt.show()

plot_data(df2)

# ==========================================================================
# Data preprocessing and split into train and test dataset
# ==========================================================================

# Data preprocessing
def data_processing(days, df):
    # min max scaler
    scale = MinMaxScaler()
    columns = ['open', 'close', 'low', 'high']
    for i in columns:
        df[i] = scale.fit_transform(df[i].values.reshape(-1, 1))
    days = days
    # current data
    current = []
    # future data
    future = []
    price = df['close'].values.tolist()
    for i in range(len(price) - days):
        current.append([price[i+j] for j in range(days)])
        future.append(price[days+i])
    current, future = np.asarray(current), np.asarray(future)
    return current, future

current, future = data_processing(3, df2)
print(current.shape, future.shape) #(13580, 3) (13580,)

# split the train and test dataset
def split_dataset(days, df):
    current, future = data_processing(days, df)
    train_test_split = int(len(current) * 0.8)
    x_train = current[:train_test_split, :]
    y_train = future[:train_test_split]
    x_test = current[train_test_split:, :]
    y_test = future[train_test_split:]
    return x_train, y_train, x_test, y_test

x_train, y_train, x_test, y_test = split_dataset(3, df2)
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape) #(10864, 3) (10864,) (2716, 3)
(2716,)

# ==============================================================
# LSTM Modeling
# ==============================================================
os.environ['CUDA_DEVICE_ORDER'] = 'PCI_BUS_ID'
os.environ['CUDA_VISIBLE_DEVICES'] = '1'
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'

EPOCHS = 10
BATCH_SIZE = 1
def LSTM_model(df, days):
    x_train, y_train, x_test, y_test = split_dataset(days, df)
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
    x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))
    model = Sequential()
    # input_shape = (length, features)
    model.add(LSTM(units=128, input_shape=(x_train.shape[1], 1),return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation = 'linear'))
    model.compile(loss = 'mse', optimizer = 'adam')
    #lstm.summary()
    model.fit(x_train, y_train, epochs = EPOCHS, batch_size = BATCH_SIZE)
    return model


# ===================================================================
# Evaluation
# ====================================================================
def evaluation(df, days):
  x_train, y_train, x_test, y_test = split_dataset(days, df)
  x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
  x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))
  model = LSTM_model(df, days)
  train_predict = model.predict(x_train)
  test_predict = model.predict(x_test)
  return train_predict, test_predict

def evaluation_plot(df, days):
  days = 3
  price = df['close']
  train_test_split = int(len(df) * 0.8)
  train_predict, test_predict = evaluation(df, days)
  plt.plot(price, label = 'actual price') # blue is actual price
  plt.plot(np.arange(days, train_test_split + 1 , 1), train_predict, color = 'red', label = 'train
dataset')
  plt.plot(np.arange(train_test_split+ days, train_test_split+ days + len(test_predict), 1),
test_predict,
         color='green', label = 'test dataset')
  plt.legend()
  plt.show()

start = time.time()
evaluation_plot(df2, 3)
print(time.time() - start)

# ==================================================================
# Loss plot
# ==================================================================

def LSTM_loss(df, days):
    x_train, y_train, x_test, y_test = split_dataset(days, df)
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
    x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))
    model = Sequential()
    # input_shape = (length, features)
    model.add(LSTM(units=128, input_shape=(x_train.shape[1], 1),return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation = 'linear'))
    model.compile(loss = 'mse', optimizer = 'adam')
    #lstm.summary()
    results = model.fit(x_train, y_train, epochs = EPOCHS, batch_size = BATCH_SIZE)
    plt.plot(results.history['loss'])
    plt.show()

LSTM_loss(df2, 3)

# ===============================================================
# Evaluation
# ================================================================
import math
from sklearn.metrics import mean_squared_error
EPOCHS = 10
BATCH_SIZE = 1
# evaluation
x_train, y_train, x_test, y_test = split_dataset(3, df2)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))
model = LSTM_model(df2, 3)
train_predict = model.predict(x_train)
test_predict = model.predict(x_test)

def evaluation_rmse(days, df):
    x_train, y_train, x_test, y_test = split_dataset(days, df)
    test = x_test[0:, 1]
    res = math.sqrt(mean_squared_error(test, test_predict))
    return res

print(evaluation_rmse(3, df2))

# ==========================================================================
# RNN Model
# ==========================================================================
EPOCHS = 10
BATCH_SIZE = 1
def RNN_model(df, days):
    x_train, y_train, x_test, y_test = split_dataset(days, df)
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
    x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))

    # input_shape = (length, features)
    model = Sequential()
    model.add(SimpleRNN(128, input_shape=(x_train.shape[1], 1), return_sequences = True,
activation = 'tanh'))
    model.add(Dropout(0.2))
    model.add(SimpleRNN(64, return_sequences = True))
    model.add(Dropout(0.2))
    model.add(SimpleRNN(32))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    adam = optimizers.Adam(lr = 0.001)
    model.compile(loss = 'mean_squared_error', optimizer = adam, metrics = ['accuracy'])
    #lstm.summary()
    model.fit(x_train, y_train, epochs = EPOCHS, batch_size = BATCH_SIZE)
    return model

# ==========================================================================
# Evaluation
# ==========================================================================
def evaluation(df, days):
  x_train, y_train, x_test, y_test = split_dataset(days, df)
  x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
  x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))
  model = RNN_model(df, days)
  train_predict = model.predict(x_train)
  test_predict = model.predict(x_test)
  return train_predict, test_predict

def evaluation_plot(df, days):
  days = 3
  price = df['close']
  train_test_split = int(len(df) * 0.8)
  train_predict, test_predict = evaluation(df, days)
  plt.plot(price, label = 'actual price') # blue is actual price
  plt.plot(np.arange(days, train_test_split + 1 , 1), train_predict, color = 'red', label = 'train
dataset')
  plt.plot(np.arange(train_test_split+ days, train_test_split+ days + len(test_predict), 1),
test_predict,
        color='green', label = 'test dataset')
  plt.legend()
  plt.show()

start = time.time()
evaluation_plot(df2, 3)
print(time.time() - start)


# ============================================================
# Loss plot
# ============================================================

def RNN_loss(df, days):
    x_train, y_train, x_test, y_test = split_dataset(days, df)
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
    x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))

    # input_shape = (length, features)
    model = Sequential()
    model.add(SimpleRNN(128, input_shape=(x_train.shape[1], 1), return_sequences = True,
activation = 'tanh'))
    model.add(Dropout(0.2))
    model.add(SimpleRNN(64, return_sequences = True))
    model.add(Dropout(0.2))
    model.add(SimpleRNN(32))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    adam = optimizers.Adam(lr = 0.001)
    model.compile(loss = 'mean_squared_error', optimizer = adam, metrics = ['accuracy'])
    #lstm.summary()
    results = model.fit(x_train, y_train, epochs = EPOCHS, batch_size = BATCH_SIZE)
    plt.plot(results.history['loss'])
    plt.show()

RNN_loss(df2, 3)


# =====================================================================
# RMSE evaluation
# =====================================================================

import math
from sklearn.metrics import mean_squared_error
EPOCHS = 10
BATCH_SIZE = 1
# evaluation
x_train, y_train, x_test, y_test = split_dataset(3, df2)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))
model = RNN_model(df2, 3)
train_predict = model.predict(x_train)
test_predict = model.predict(x_test)

def evaluation_rmse(days, df):
    x_train, y_train, x_test, y_test = split_dataset(days, df)
    test = x_test[0:, 1]
    res = math.sqrt(mean_squared_error(test, test_predict))
    return res

print(evaluation_rmse(3, df2))
```

```
# ============================================================
# GRU
# ============================================================

EPOCHS = 10
BATCH_SIZE = 1
from keras.layers import GRU, Dense, Dropout
def model_GRU(df, days):
    x_train, y_train, x_test, y_test = split_dataset(days, df)
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
    x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))

    model = Sequential()
    model.add(GRU(units = 128, input_shape=(x_train.shape[1], 1), return_sequences = True,
activation = 'tanh'))
    model.add(Dropout(0.2))
    model.add(GRU(64, return_sequences = True))
    model.add(Dropout(0.2))
    model.add(GRU(32))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    adam = optimizers.Adam(lr = 0.001)
    model.compile(loss = 'mse', optimizer = adam, metrics = ['accuracy'])
    model.fit(x_train, y_train, epochs = EPOCHS, batch_size = BATCH_SIZE)
    #lstm.summary()
    return model
# ============================================================
# Evaluation
# ============================================================
def evaluation(df, days):
  x_train, y_train, x_test, y_test = split_dataset(days, df)
  x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
  x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))
  model = model_GRU(df, days)
  train_predict = model.predict(x_train)
  test_predict = model.predict(x_test)
  return train_predict, test_predict
```

```
def evaluation_plot(df, days):
  days = 3
  price = df['close']
  train_test_split = int(len(df) * 0.8)
  train_predict, test_predict = evaluation(df, days)
  plt.plot(price, label = 'actual price') # blue is actual price
  plt.plot(np.arange(days, train_test_split + 1 , 1), train_predict, color = 'red', label = 'train
dataset')
  plt.plot(np.arange(train_test_split+ days, train_test_split+ days + len(test_predict), 1),
test_predict,
         color='green', label = 'test dataset')
  plt.legend()
  plt.show()

start = time.time()
evaluation_plot(df2, 3)
print('GRU takes:', time.time() - start)


# ===================================================================
# Evalulate RMSE
# ===================================================================
import math
from sklearn.metrics import mean_squared_error

start = time.time()
x_train, y_train, x_test, y_test = split_dataset(3, df2)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))
model = model_GRU(df2, 3)
train_predict = model.predict(x_train)
test_predict = model.predict(x_test)

print('GRU takes:', time.time() - start)

def evaluation_rmse(days, df):
    x_train, y_train, x_test, y_test = split_dataset(days, df)
    test = y_test[0:, 1]
    res = math.sqrt(mean_squared_error(test, test_predict))
    return res

print(evaluation_rmse(3, df2))


x_test2 = np.asarray([item[0] for item in x_test])
from sklearn.metrics import r2_score
r2_y_predict = r2_score(x_test2,test_predict)
print('R2 : ', r2_y_predict)


# ==============================================================
# Plot loss
# ==============================================================

def GRU_loss(df, days):
    x_train, y_train, x_test, y_test = split_dataset(days, df)
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1)) #### Importatnce to
reshape!!!!
    x_test= np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))

    model = Sequential()
    model.add(GRU(units = 128, input_shape=(x_train.shape[1], 1), return_sequences = True,
activation = 'tanh'))
    model.add(Dropout(0.2))
    model.add(GRU(64, return_sequences = True))
    model.add(Dropout(0.2))
    model.add(GRU(32))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    adam = optimizers.Adam(lr = 0.001)
    model.compile(loss = 'mse', optimizer = adam, metrics = ['accuracy'])
    results = model.fit(x_train, y_train, epochs = EPOCHS, batch_size = BATCH_SIZE)
    plt.plot(results.history['loss'])
    plt.show()

GRU_loss(df2, 3)
```