PROJECT REPORT

ON

Application of GRU for Stock Price prediction

BY

UMAR AYOUB HAJAM

(PROJECT LEAD)

Executive Summery

Stock market prediction is an attempt of determining the future value of a stock traded on a stock exchange. This project report attempts to provide an optimal Gated Recurrent Unit model for the prediction of stock prices From the experiments it is found that Stacked GRU with Hyperbolic Tangent and Rectified Linear Unit activation function is most successful in stock price prediction. In addition the experiments have suggested suitable values of the look-back period that could be used with GRU.

Machine and ide details

PyCharm 2021.1 (Community Edition)

Build #PC-211.6693.115, built on April 6, 2021

Runtime version: 11.0.10+9-b1341.35 amd64

VM: Dynamic Code Evolution 64-Bit Server VM by JetBrains s.r.o.

Windows 10 10.0

GC: ParNew, ConcurrentMarkSweep

Memory: 9933M

Cores: 8

Modules/Libraries

The modules/libraries used in model generation are

1. Sklearn
2. Pandas
3. Matplotlib
4. Numpy
5. Keras

Importing the Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from keras import Sequential
from sklearn.metrics import max_error, mean_absolute_error,
mean_squared_error
from keras.layers import LSTM, Dense, RNN, GRU, Dropout
from keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error
```

Dataset

```
df = pd.read_csv("DataFrame.csv")
# Droping few columns
df = df.drop("Type", axis=1)
df = df.drop("classification", axis=1)
df["Time"] = pd.to_datetime(df['Time'])

df1 = df.reset_index()['close']
df1.describe
```

Scaling and Splitting the data

```
scalar = MinMaxScaler(feature_range=(0, 1))
df1 = scalar.fit_transform(np.array(df1).reshape(-1, 1))
training_size=int(len(df1)*0.85)
test_size = len(df1)-training_size
train_data, test_data = df1[0:training_size, :],
df1[training_size:len(df1), :1]

print(training_size, test_size)


def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

MODEL

```
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1],
1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
regressorGRU = Sequential()
# First GRU layer with Dropout regularisation
regressorGRU.add(GRU(units=50, return_sequences=True,
input_shape=(X_train.shape[1],1), activation='tanh'))
regressorGRU.add(Dropout(0.3))
# Second GRU layer
regressorGRU.add(GRU(units=50, return_sequences=True,
input_shape=(X_train.shape[1],1), activation='relu'))
regressorGRU.add(Dropout(0.3))
# Third GRU layer
regressorGRU.add(GRU(units=64, return_sequences=True,
input_shape=(X_train.shape[1],1), activation='relu'))
regressorGRU.add(Dropout(0.3))
# Fourth GRU layer
regressorGRU.add(GRU(units=64, activation='tanh'))
regressorGRU.add(Dropout(0.2))
# The output layer
regressorGRU.add(Dense(units=1))

regressorGRU.compile(optimizer=SGD(lr=0.01, decay=1e-7,
momentum=0.9, nesterov=False), loss='mean_squared_error')

# fitting the model

regressorGRU.fit(X_train, y_train, epochs=80, batch_size=150,
validation_split=0.15, callbacks=[tensorboard_callback])
```

Predicting and Plotting the predictions

```
predicted_with_gru = regressorGRU.predict(X_test)
predicted_with_gru =
scalar.inverse_transform(predicted_with_gru)
regressorGRU.save("GRUCLASS.h5")


def plot_predictions(test, predicted):
    plt.plot(test, color="red", label="realstock price")
    plt.plot(predicted, color="blue", label="predicted stock
price")
    plt.title("stock price prediction")
```

```
    plt.xlabel("time")
    plt.ylabel("stock price")
    plt.legend()
    plt.show()
```

Model Evaluation

```
def return_rmse(test, predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("the root mean squared error is : {}.".format(rmse))


train_predict = regressorGRU.predict(X_train)
test_predict = regressorGRU.predict(X_test)

train_predict = scalar.inverse_transform(train_predict)
test_predict = scalar.inverse_transform(test_predict)
math.sqrt(mean_squared_error(y_train, train_predict))
math.sqrt(mean_squared_error(ytest, test_predict))

plot_predictions(ytest, predicted_with_gru)

return_rmse(ytest, predicted_with_gru)
# shift train predictions for plotting
look_back=100
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] =
train_predict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1,
:] = test_predict
# plot baseline and predictions
plt.plot(scalar.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```