**Prediction of Gas Price in Ethereum Blockchain using Deep Learning**

*Report submitted to the SASTRA Deemed to be University as the requirement for the course*

## CSE300: MINI PROJECT

*Submitted by*

## Trivikraman S (123018111, B.Tech CSBS)

June-2022

# SCHOOL OF COMPUTING THANJAVUR, TAMIL NADU, INDIA-613401

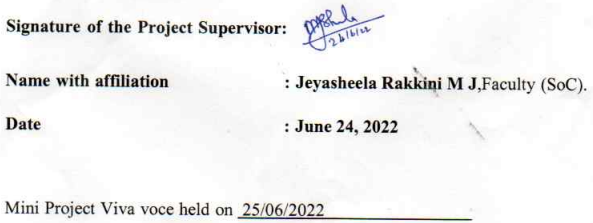




**SCHOOL OF COMPUTING THANJAVUR, TAMIL NADU, INDIA-613401**

# Bonafide Certificate

This is to certify that the report titled “Prediction of Gas Price in Ethereum Blockchain using Deep Learning” submitted as a requirement for the course, CSE300: MINI PROJECT for B.Tech is a bonafide record of the work done by, **Mr.S.Trivikraman(**Reg.No.**123018111,B.Tech-CSBS)** during the academic year 2021-22, in the School of Computing, under my supervision.





**Acknowledgement**

I have successfully completed the project. However, it would have not been possible without the kind of support and help of many individuals and organization. I would like to express my sincere thanks to all of them.

I am grateful to our Vice Chancellor **Dr. S. Vaidhyasubramaniam** and our Registrar **Dr. R. Chandramouli** for giving me the opportunity to work on this project and for providing their continuous support.

I am also thankful to our Dean **Dr. A. Umamakeswari** for her guidance and support in completing the project.

I am highly indebted to **Jeyasheela Rakkini M J** for her guidance and constant supervision as well as providing necessary information regarding the project continuously and also for her support in completing the project on time. Working on this project was an immense source of knowledge for me. I also express my sincere thanks to my guide for extending valuable guidance, support and critical review of the project and for the moral support provided at all points of time.

I would also like to express my gratitude towards my parents and members of SASTRA University for their kind co-operation and encouragement which helped me in completion of this project.

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| **FIGURE. NO.** | **TITLE** | **PAGE. NO.** |
| 1 | Sigmoid activation function | 4 |
| 2 | Tanh activation function | 4 |
| 3 | GRU vs LSTM | 4 |
| 3 | Workflow diagram | 5 |
| 4 | Prophet vs real data plot | 19 |
| 5 | Loss function plot for GRU | 19 |
| 6 | Loss function plot for LSTM | 20 |
| 7 | GRU vs real data plot | 20 |
| 8 | LSTM vs real data plot | 21 |
| 9 | Comparison of models with geth gas price oracle plot | 21 |

# LIST OF TABLES

|  |  |  |
| --- | --- | --- |
| **S.no** | **TITLE** | **PAGE.NO** |
| 1 | Error Metrics of Prophet, Lstm, Gru models and Geth Oracle | 18 |

# Abbreviations

**RNN** Recurrent Neural Network

**LSTM** Long Short Term Memory

**GRU** Gated Recurrent Unit

**MSE** Mean Squared Error

**MAE**  Mean Absolute Error

**RMSE** Root Mean Squared Error

# Notations

No specific notation is used

# Abstract

Ethereum, the open source, decentralized, distributed blockchain with inherent traits of data provenance, immutability, has ether as the native currency. The miners in Ethereum collect the transactions and form the block to get the rewards for appending the block in the Ethereum blockchain. Ethereum follows proof of stake for consensus of appending block to Ethereum blockchain. Ethereum smart contracts are used to read, write and trigger functionalities in the Ethereum blockchain.

Ethereum blockchain transactions have gas fee, which is the fees/reward to be paid to miners for the selection of this transaction from the memory pool of transactions in the full node and including them in the block. Ethereum transaction sender is exposed to the task of having to choose an optimal gas price, underpaying likely results in a transaction not being picked by miners, whereas overpaying leads to unnecessary high costs.

We are addressing this problem by formulating an approach for prediction and recommendation of gas price of all transactions with ether crypto currency. Prediction of gas price for the transactions that can be included in the next block to be mined is done by us as a comparative study of deep learning models.We compared models such as Facebook Prophet API, Long short term memory and Gate recurrent unit for the metrics such as accuracy score, mean absolute error, mean squared error and many more. We also aim to compare with the most used gas price oracles.

**Keywords**: Gas Price, Ethereum, Ether, Gwei, Blockchain, Deep Learning.

# Table of Contents

**Title Page No.**

[Bonafide Certificate ii](#_TOC_250005)

Acknowledgements iii

[List of Figures iv](#_TOC_250004)

[List of Tables v](#_TOC_250003)

[Abbreviations vi](#_TOC_250002)

[Notations vi](#_TOC_250001)

[Abstract vii](#_TOC_250000)

1. Summary of the base paper 1
2. Merits and Demerits of the base paper 2
3. Project Phases & Source Code 3
4. Snapshots 19
5. Conclusion and Future Plans 22
6. References 23
7. Appendix -Literature Survey 24

# CHAPTER 1 SUMMARY OF THE BASE PAPER

**Title :** A Machine Learning Approach for Gas Price Prediction in Ethereum Blockchain

**Authors :** Rawya Mars, Amal Abid, Saoussen Cheikhrouhou, Slim Kallel

**Published in :** 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)

**Publisher :**IEEE

**Date Added to IEEE *Xplore* :**09 September 2021

The blockchain-based platform Ethereum provides a global computational infrastructure to run smart contracts. Ethereum Blockchain adopts a gas-based metering approach to charge for smart contract and transaction execution in the blockchain which is designed to motivate miners to operate the network and protect it against threats. To be specific ,miners receive fees from all transactions included in the mined block in addition to the mining reward.

Higher the gas price in the transactions faster will be the selection and execution of transaction as higher fees will be paid to the miner. Therefore, Ethereum transaction sender is exposed to the task of having to choose an optimal gas price as underpaying likely results in a transaction not being picked by miners, whereas overpaying leads to unnecessary high costs. This paper addresses this problem and provides recommendation approach that proposes an appropriate gas price to users.

It looks into several forecasting algorithms applied for the upcoming Ethereum Blockchain block's gas price forecasts. Facebook Prophet model and the deep learning models such as Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) are used to forecast the price of gas price. Additionally, it seeks to contrast these methods with the popular gas price oracles.

The LSTM and GRU models outperform the Prophet model and the Geth gas price oracle, according to an analysis of the results. LSTM and GRU offer a low mean squared error (MSE) of 0,008, whereas Geth and Prophet offer MSEs of 0.016 and 0.014, respectively.

# CHAPTER 2

**Merits and Demerits of the base paper**

## Merits:

The task of predicting gas price of a transaction in the Ethereum network is a necessity considering the amount of capital involved. So far Ethereum users used gas price oracles like EthGasStation , Geth client to get recommended with gas price. But the existing gas price oracles are proven to be less efficient and not suggesting an optimal price. This paper addresses this problem and suggests an approach for predicting and recommending optimal gas price for the users.

.

## Demerits:

Even though the proposed model is performing well, the decision cannot be taken completely based on this method. This is because the world of cryptocurrency is driven by sentiments of the network users. So, for taking decision a mathematical/statistical method is alone is not enough. An additional sentimental analysis component has to be added to the algorithm for predicting gas price. The method relies only on past data for prediction which makes it inefficient at times as the price get affects by other factors such as news related to politics and disaster

# CHAPTER 3

**Project Phases & Code**

## Project Phases

* + 1. **Phase -1: Data collection, preprocessing and visualization**
* We are using a real-world dataset which consists of historical minimum gas price per block from the period of 14 days which sums around 90000 blocks.
* This data was indexed for the block timestamp or block date to improve accessing vastly
* We also have geth gas price oracle suggested data for the same period for sake of comparison
* Necessary data preprocessing is done to reduce the impact of imbalanced data.Outliers are removed, data is resampled by averaging consecutive blocks at 5 minute intervals and normalization is done inorder to make it suitable for input to the deep learning algorithms used in future phases

## Phase-2: Implementation of prophet model

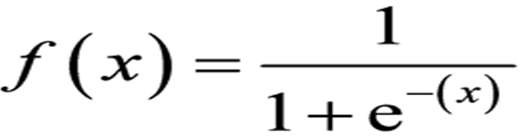
* Prophet, developed by Facebook, is an open-source tool designed for time series forecasting.
* Prophet essentially is an additive regression model that decomposes a time series into (i) a linear/logistic (piecewise) trend, (ii) an annual seasonal component, (iii) a weekly seasonal component, and (iv) an optional list of important days (such as vacations, special events, ...).
* The model states that it is "robust to missing data, trend changes, and significant outliers", which would make it well suited to this particular task.

## Phase-3: Implementation of RNN algorithms-LSTM and GRU

* Inorder to achieve the goal of predicting the minimum gas price over a pre-defined period algorithms capable of detecting patterns in data are very useful.
* So much research and investigation has proven RNNs to be best suit for forecasting timeseries data. RNN models LSTM and GRU have been well investigated and proved to be good market forecasters.
* Considering the fact that RNNs are good at forecasting timeseries data, RNN models are implemented inorder to forecast etherum gas price.

## Activation Functions used:

* Activation functions define the resulting output of the node for the given input (or set) on inputs. It will map the output of the Network with a certain range. Here, mainly 2 activation functions are used.
  + - 1. **Sigmoid:** This activation function will map the output to (0,1), but the curve will look like S-shaped. So, this function is used in the places where we have to predict the probability.



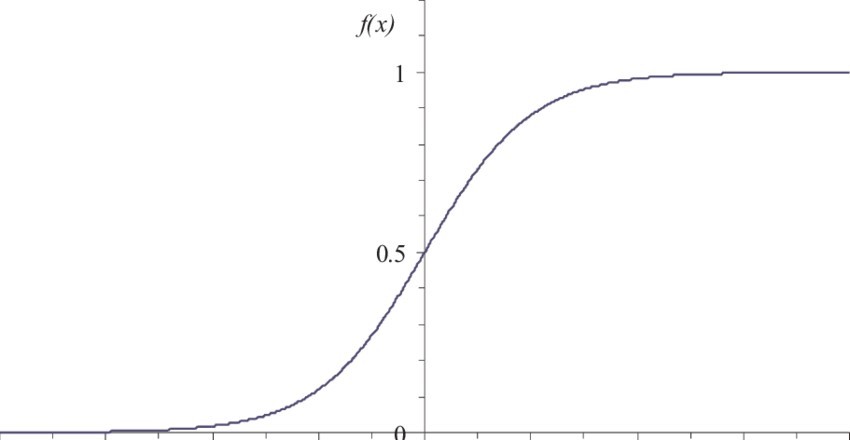


Fig. 3.1 Graph of sigmoid Activation function.

**2.Tan h activation function:** tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).



Fig. 3.2 Graph of tanh Activation function.

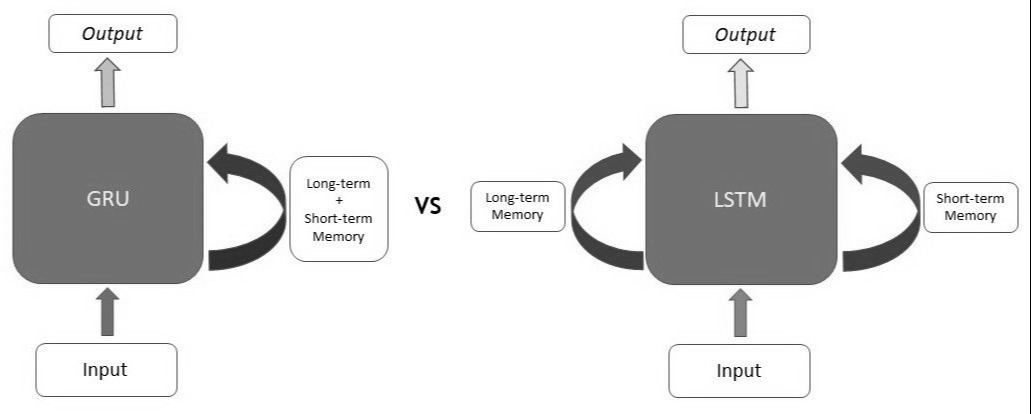


Fig. 3.3 GRU VS LSTM.

## Phase-4: Comparison of approaches :

## All the forecast data of the previous 2 phases are compared with the geth forecasts and real price and it is visualized proving visually that LSTM and GRU is outperforming Prophet and geth forecasts.

## In addition to visualization error metrics are calculated for each approach.Performance meaurements used are Mean Absolute error, Mean Square Error, Root Mean Square Error and R-squared value proving the same inform of numbers

## 

Project Work Flow

## 

Fig 3.4 Project Workflow

## Source Code:

## 

## LIBRARIES

import numpy as np  
from numpy import mean  
from numpy import std  
import matplotlib.pyplot as plt  
import matplotlib  
import pandas as pd  
from datetime import datetime  
from fbprophet import Prophet  
from sklearn.metrics import mean\_squared\_error,mean\_absolute\_error,r2\_score  
from sklearn.preprocessing import MinMaxScaler  
import math  
from math import sqrt  
from keras.models import Sequential  
from keras.layers import Dense,LSTM,GRU  
from tensorflow.keras.optimizers import SGD  
from numpy import mean  
from numpy import std

# 

# DATA PREPROCESSING AND VISUALIZATION

## Historical gas price dataset

## 

#Loading data  
df=pd.read\_csv('data10\_25\_oct.csv',sep=';', names=['Date', 'ming'])

#converting the data to time-series type by taking index to be the time  
mydateparser = lambda x: datetime.strptime(x,"%Y-%m-%d %H:%M:%S")  
dataset = pd.read\_csv('data10\_25\_oct.csv',sep=';', names=['Date', 'ming'], parse\_dates=['Date'], date\_parser=mydateparser)

dataset=dataset.sort\_values(by=['Date'],ascending=True)

dataset.isnull().sum()

dataset.isna().any()

#Data visualization  
dataset.set\_index('Date').plot(figsize=(16,6),legend=True,grid=True  
plt.legend(['Real Gas Price'])  
plt.show()

#Step 1 : data re-sampling   
dataset=dataset.set\_index('Date').resample('5 Min').mean()  
dataset.plot(figsize=(16,6),legend=True,grid=True)  
plt.legend(['Real Gas Price'])  
plt.show()

#Step 2 : removing outliers   
## calculate summary statistics  
data\_mean, data\_std = mean(dataset["ming"]), std(dataset["ming"])  
# identify outliers  
cut\_off = data\_std \* 2  
lower, upper = data\_mean - cut\_off, data\_mean + cut\_off  
for index,row in dataset.iterrows():  
 if row["ming"] < lower:  
 row["ming"]=lower  
 elif row["ming"] > upper:  
 row["ming"]=upper  
#remove outliers   
outliers\_removed = [x for x in dataset["ming"] if x >= lower and x <= upper]  
print('Non-outlier observations: %d' % len(outliers\_removed))  
#Data visualization legend  
dataset["ming"].plot(figsize=(16,6), =True,grid=True)  
plt.legend(['Real Gas Price'])  
plt.show()

## Geth gas price oracle dataset

#loading data   
mydateparser = lambda x: datetime.strptime(x, "%Y-%m-%d %H:%M:%S")  
gethdata = pd.read\_csv('geth.csv',sep=';', names=['Date', 'gethprice'], parse\_dates=['Date'], date\_parser=mydateparser, low\_memory=False)  
#convert the data to time-series type, by taking index to be the time

dataset\_geth=gethdata.sort\_values(by=['Date'],ascending=True)

dataset\_geth.isnull().sum()

dataset\_geth.isna().any()

#Data visualization  
gethdata.set\_index('Date')['2020-10-21 08:40:00':'2020-10-24 17:05:00']["gethprice"].plot(figsize=(16,6),legend=True,grid=True)  
plt.legend(['Geth gas price'])  
plt.show()

#Data preprocesssing steps:   
#Step 1 : data re-sampling   
gethdata=gethdata.set\_index('Date')['2020-10-21 08:40:00':'2020-10-24 17:05:00'].resample('5 Min').mean()

#Step 2 : removing outliers   
# calculate summary statistics  
data\_mean, data\_std = mean(gethdata["gethprice"]), std(gethdata["gethprice"])  
# identify outliers  
for index,row in gethdata.iterrows():  
 if row["gethprice"] < lower:  
 row["gethprice"]=lower  
 elif row["gethprice"] > upper:  
 row["gethprice"]=upper  
#remove outliers   
outliers\_removed = [x for x in gethdata["gethprice"] if x >= lower and x <= upper]  
print('Non-outlier observations: %d' % len(outliers\_removed))

#Data visualization  
gethdata['2020-10-21 08:40:00':'2020-10-24 17:05:00']["gethprice"].plot(figsize=(16,6),legend=True,grid=True)  
plt.legend(['Geth gas price'])  
plt.show()

# 

# PROPHET MODEL

dataset1 =dataset[['ming']]  
#splitting data into Train/Test data  
training\_set = dataset1['2020-10-10 00:00:00':'2020-10-21 08:35:00']  
training\_set = training\_set.reset\_index()[['Date', 'ming']]  
testsetp = dataset1['2020-10-21 08:40:00': '2020-10-24 18:05:00']  
testsetp = testsetp.reset\_index()[['Date', 'ming']]  
training\_set.columns = ['ds', 'y']

#Create Prophet Model  
m = Prophet(changepoint\_prior\_scale=0.0005,yearly\_seasonality=False, weekly\_seasonality=True, daily\_seasonality=True).fit(training\_set)

#Forecast on Test Set  
future = m.make\_future\_dataframe(periods=testsetp["ming"].count(), freq='5 Min', include\_history= True)  
fcst = m.predict(future)

#forecasts visualisation  
fig = m.plot(fcst)  
  
  
#dots is real data  
#blue line is the projection done   
#light blue space is trend space

fig = m.plot\_components(fcst)

real=np.array(dataset1.ming)  
real=real.reshape(-1, 1)  
pred=np.array(fcst.yhat)  
pred=pred.reshape(-1, 1)  
sc = MinMaxScaler(feature\_range=(0,1))  
real = sc.fit\_transform(real)  
pred = sc.transform(pred)  
#Error Metrics On Test Set  
print('prophet MAE : %.3f'%mean\_absolute\_error(real, pred))  
print('prophet MSE: %.3f'%mean\_squared\_error(real, pred))  
print('prophet R2-score : %.3f'%r2\_score(real, pred))  
print('prophet RMSE : %.3f'%sqrt(mean\_squared\_error(real, pred)))  
fcst=fcst.set\_index('ds')  
dataset1=dataset1.reset\_index()[['Date', 'ming']]  
dataset1.columns = ['ds', 'y']  
fcst = fcst.reset\_index()[['ds', 'yhat']]

#visualize prophet prediction  
metric\_df = fcst.set\_index('ds')[['yhat']].join(dataset1.set\_index('ds').y).reset\_index()  
metric\_df.y=metric\_df.y.fillna(0)  
print(metric\_df.head())  
plt.legend(['true' , 'prdicted'])  
metric\_df.set\_index('ds').y.plot(figsize=(16,6),legend=True,grid=True)  
metric\_df.set\_index('ds').yhat.plot(figsize=(16,6),legend=True,grid=True)  
plt.legend(['real gasprice' , 'predicted gas price'])  
plt.title('Prophet Gas Price Predicton ')  
plt.show()

#visualize prophet prediction over training and testing period   
metric\_df.set\_index('ds')['2020-10-10 00:00:00':'2020-10-21 08:35:00'].y.plot(figsize=(16,6),legend=True,grid=True)  
metric\_df.set\_index('ds')['2020-10-10 00:00:00':'2020-10-21 08:35:00'].yhat.plot(figsize=(16,6),legend=True,grid=True)  
plt.legend(['real gasprice' , 'predicted gas price'])  
plt.title('Prophet Gas Price Predicton over training period')  
plt.show()

metric\_df.set\_index('ds')['2020-10-21 08:40:00': '2020-10-24 18:05:00'].y.plot(figsize=(16,6),legend=True,grid=True)  
metric\_df.set\_index('ds')['2020-10-21 08:40:00': '2020-10-24 18:05:00'].yhat.plot(figsize=(16,6),legend=True,grid=True)  
plt.legend(['real gasprice' , 'predicted gas price'])  
plt.title('Prophet Gas Price Predicton over testing period')  
plt.show()

# 

# LONG-SHORT TERM MEMORY

dataset2 = dataset[['ming']]  
# removing missing value rows  
dataset2 = dataset2.dropna()  
dataset2

#splitting data into Train/Test data  
training\_set = dataset2['2020-10-10 00:00:00':'2020-10-21 08:35:00']  
test\_set = dataset2['2020-10-21 07:20:00':'2020-10-24 17:05:00']

#applying min-max normalization   
sc = MinMaxScaler(feature\_range=(0,1))  
training\_set = sc.fit\_transform(training\_set)  
test\_set = sc.transform(test\_set)

#Create x, y test and train data windows and into input and outputs  
training\_set = np.array(training\_set)  
test\_set = np.array(test\_set)  
x\_train = []  
y\_train = []  
previous =15  
for i in range(len(training\_set)-previous-1):  
 x\_train.append(training\_set[i:i+previous])  
 y\_train.append(training\_set[i+previous])  
x\_train, y\_train = np.array(x\_train), np.array(y\_train)  
print(x\_train)

# reshape input to be 3D [samples, timesteps]  
y\_train = np.reshape(y\_train, (y\_train.shape[0],-1))  
x\_train = np.reshape(x\_train, (x\_train.shape[0],x\_train.shape[1],1))  
x\_train.shape, y\_train.shape  
x\_test = []  
y\_test = []  
for i in range(len(test\_set)-previous-1):  
 x\_test.append(test\_set[i:i+previous])  
y\_test.append(test\_set[i+previous])  
x\_test, y\_test = np.array(x\_test), np.array(y\_test)

# reshape input to be 2D [samples, timesteps]  
y\_test = np.reshape(y\_test, (y\_test.shape[0],-1))  
x\_test = np.reshape(x\_test, (x\_test.shape[0],x\_test.shape[1],1))

#Create LSTM Model  
modelL = Sequential()  
modelL.add(LSTM(units =15,activation='tanh',input\_shape=(x\_train.shape[1], x\_train.shape[2]),dropout=0.01))  
modelL.add(Dense(1))  
opt = SGD(lr=0.01, decay=0.0001)  
modelL.compile(loss='mean\_absolute\_error', optimizer=opt)

# fit network  
history2 = modelL.fit(x\_train, y\_train, epochs=500,batch\_size=112,validation\_split=0.10,shuffle=False,verbose=0)

#visualize results   
plt.plot(history2.history['loss'])  
plt.plot(history2.history['val\_loss'])  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train', 'validation'], loc='upper right')  
plt.show()

# make a prediction for training data  
trainLSTM = modelL.predict(x\_train)

#calculate error rates   
maeLSTMTR=mean\_absolute\_error(y\_train, trainLSTM)  
mseLSTMTR=mean\_squared\_error(y\_train, trainLSTM)  
r2LSTMTR=r2\_score(y\_train, trainLSTM)  
rmseLSTMTR=sqrt(mean\_squared\_error(y\_train, trainLSTM))  
print('LSTM : %.3f'%maeLSTMTR)  
print('LSTM : %.3f'%mseLSTMTR)  
print('LSTM : %.3f'%r2LSTMTR)  
print('LSTM : %.3f'%rmseLSTMTR)

# invert scaling for forecast and real   
trainLSTM = sc.inverse\_transform(trainLSTM)  
y\_train = sc.inverse\_transform(y\_train)

#visualaize forecastings   
plt.figure(figsize=(18,5))  
plt.plot(y\_train, color='blue',label='Real '+' Gas Price')  
plt.plot(trainLSTM, color='red',label='Predicted '+' Gas Price')  
plt.title(' Gas Price Prediction(LSTM)')  
plt.xlabel('Time')  
plt.grid()  
plt.ylabel(' Gas Price')  
plt.legend()  
plt.show()

# make a prediction for training data  
testLSTM = modelL.predict(x\_test)

#calculate error rates   
maeLSTMTS=mean\_absolute\_error(y\_test, testLSTM)  
mseLSTMTS=mean\_squared\_error(y\_test, testLSTM)  
r2LSTMTS=r2\_score(y\_test, testLSTM)  
rmseLSTMTS=sqrt(mean\_squared\_error(y\_test, testLSTM))  
print('LSTM : %.3f'%maeLSTMTS)  
print('LSTM : %.3f'%mseLSTMTS)  
print('LSTM : %.3f'%r2LSTMTS)  
print('LSTM : %.3f'%rmseLSTMTS)

# invert scaling for forecast and real   
testLSTM = sc.inverse\_transform(testLSTM)  
y\_test = sc.inverse\_transform(y\_test)

#visualaize forecastings   
plt.figure(figsize=(18,5))  
plt.plot(y\_test, color='blue',label='real '+' Gas Price')  
plt.plot(testLSTM, color='red',label='Predicted '+' Gas Price')  
plt.title(' Gas Price Prediction(LSTM)')  
plt.xlabel('Time')  
plt.grid()  
plt.ylabel(' Gas Price')  
plt.legend()  
plt.show()

# 

# GATED RECURRENT UNIT-GRU

dataset2 = dataset[['ming']]  
# removing missing value rows  
dataset2 = dataset2.dropna()  
dataset2

#splitting data into Train/Test data  
training\_set = dataset2['2020-10-10 00:00:00':'2020-10-21 08:35:00']  
test\_set = dataset2['2020-10-21 07:20:00':'2020-10-24 17:05:00']

#applying min-max normalization   
sc = MinMaxScaler(feature\_range=(0,1))  
training\_set = sc.fit\_transform(training\_set)  
test\_set = sc.transform(test\_set)

#Create x, y test and train data windows and into input and outputs  
training\_set = np.array(training\_set)  
test\_set = np.array(test\_set)  
x\_train = []  
y\_train = []  
previous =15  
for i in range(len(training\_set)-previous-1):  
 x\_train.append(training\_set[i:i+previous])  
 y\_train.append(training\_set[i+previous])  
x\_train, y\_train = np.array(x\_train), np.array(y\_train)

# reshape input to be 3D [samples, timesteps]  
y\_train = np.reshape(y\_train, (y\_train.shape[0],-1))  
x\_train = np.reshape(x\_train, (x\_train.shape[0],x\_train.shape[1],1))  
x\_train.shape, y\_train.shape  
x\_test = []  
y\_test = []  
for i in range(len(test\_set)-previous-1):  
 x\_test.append(test\_set[i:i+previous])  
 y\_test.append(test\_set[i+previous])  
x\_test, y\_test = np.array(x\_test), np.array(y\_test)

y\_test = np.reshape(y\_test, (y\_test.shape[0],-1))  
x\_test = np.reshape(x\_test, (x\_test.shape[0],x\_test.shape[1],1))

#Create GRU Model  
modelG = Sequential()  
modelG.add(GRU(units=15,activation='tanh',input\_shape=(x\_train.shape[1], x\_train.shape[2]),dropout=0.01))  
modelG.add(Dense(1))  
opt = SGD(lr=0.01, decay=0.0001)  
modelG.compile(loss='mean\_absolute\_error', optimizer=opt)

# fit network  
history = modelG.fit(x\_train, y\_train, epochs=500,batch\_size=128,shuffle=False,validation\_split=0.10,verbose=0)

#visualize results   
plt.plot(history.epoch, history.history['loss'])  
plt.plot(history.epoch , history.history['val\_loss'])  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train', 'validation'], loc='upper right')  
plt.show()

# make a prediction for training data  
trainGRU = modelG.predict(x\_train)

#calculate error rates   
maeGRUTR=mean\_absolute\_error(y\_train, trainGRU)  
mseGRUTR=mean\_squared\_error(y\_train, trainGRU)  
r2GRUTR=r2\_score(y\_train, trainGRU)  
rmseGRUTR=sqrt(mean\_squared\_error(y\_train, trainGRU))  
print('gru : %.3f'%maeGRUTR)  
print('gru : %.3f'%mseGRUTR)  
print('gru : %.3f'%r2GRUTR)  
print('gru : %.3f'%rmseGRUTR)

# invert scaling for forecast and real   
trainGRU = sc.inverse\_transform(trainGRU)  
y\_train = sc.inverse\_transform(y\_train)

#visualaize forecastings   
plt.figure(figsize=(18,5))  
plt.plot(y\_train,color='blue',label='Real '+' Gas Price')  
plt.plot(trainGRU, color='yellow',label='Predicted '+' Gas Price')  
plt.title(' Gas Price Prediction(GRU)')  
plt.xlabel('Time')  
plt.grid()  
plt.ylabel(' Gas Price')  
plt.legend()  
plt.show()

testGRU = modelG.predict(x\_test)

#calculate error rates   
maeGRUTS=mean\_absolute\_error(y\_test, testGRU)  
mseGRUTS=mean\_squared\_error(y\_test, testGRU)  
r2GRUTS=r2\_score(y\_test, testGRU)  
rmseGRUTS=sqrt(mean\_squared\_error(y\_test, testGRU))  
print('gru : %.3f'%maeGRUTS)  
print('gru : %.3f'%mseGRUTS)  
print('gru : %.3f'%r2GRUTS)  
print('gru : %.3f'%rmseGRUTS)

# invert scaling for forecast and real  
testGRU = sc.inverse\_transform(testGRU)  
y\_test = sc.inverse\_transform(y\_test)

#visualaize forecastings  
plt.figure(figsize=(18,5))  
plt.plot(y\_test, color='blue',label='real '+' Gas Price')  
plt.plot(testGRU, color='yellow',label='Predicted '+' Gas Price')  
plt.title(' Gas Price Prediction(GRU)')  
plt.xlabel('Time')  
plt.grid()  
plt.ylabel(' Gas Price')  
plt.legend()  
plt.show()

# 

# COMPARISON WITH GETH AND VISUALIZATION

gethdata = gethdata.reset\_index()[['Date', 'gethprice']]  
geth=np.array(gethdata['gethprice'])  
geth = sc.transform(geth.reshape(-1,1))  
y\_test=sc.transform(y\_test)

#calculate Geth error rates   
print('Geth : %.3f'%mean\_absolute\_error(y\_test, geth))  
print('Geth : %.3f'%mean\_squared\_error(y\_test, geth))  
print('Geth : %.3f'%r2\_score(y\_test, geth))  
print('Geth : %.3f'%sqrt(mean\_squared\_error(y\_test, geth)))

y\_test=sc.inverse\_transform(y\_test)

#visualize of different prediction models  
gethdata['predP']=fcst['yhat'][0:966]  
gethdata['predL']=testLSTM[0:966]  
gethdata['predG']=testGRU[0:966]  
gethdata['real']=y\_test[0:966]

gethdata.set\_index('Date').predL.plot(figsize=(18,6),color='r',legend=True,grid=True)  
gethdata.set\_index('Date').gethprice.plot(figsize=(18,6),color='g',legend=True,grid=True)  
gethdata.set\_index('Date').real.plot(figsize=(18,6),color='b',legend=True,grid=True)  
gethdata.set\_index('Date').predG.plot(figsize=(18,6),color='y',legend=True,grid=True)  
gethdata.set\_index('Date').predP.plot(figsize=(18,6),color='orange',legend=True,grid=True)  
plt.legend(['Geth' , 'real','predLSTM','predGRU','predProphet'])  
plt.title(' Gas Price methods comparison')  
plt.show()

**Results:**

Table 3.1. Error Metrics Of Prophet, Lstm, Gru Models And Geth Oracle

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **APPROACH** | **MAE** | **MSE** | **R2\_SCORE** | **RMSE** |
| **PROPHET** | 0.094 | 0.014 | 0.667 | 0.119 |
| **LSTM** | *0.063* | *0.008* | *0.896* | *0.088* |
| **GRU** | *0.063* | *0.008* | *0.896* | *0.088* |
| **GETH** | 0.097 | 0.016 | 0.778 | 0.128 |

# CHAPTER 4

**Snapshots**

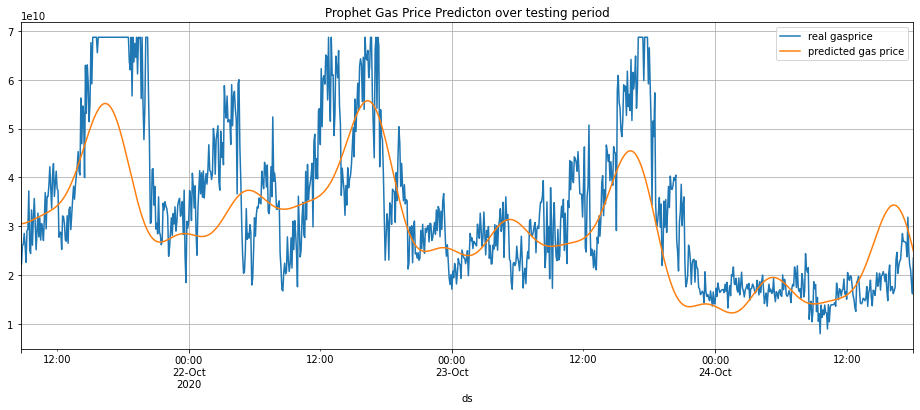


Fig. 4.1 Prophet vs real data plot

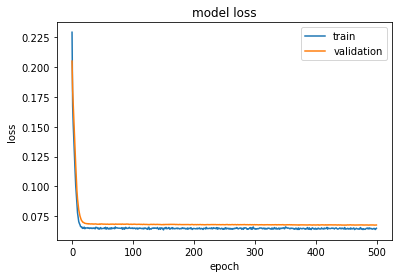


Fig. 4.2 Loss function plot for GRU

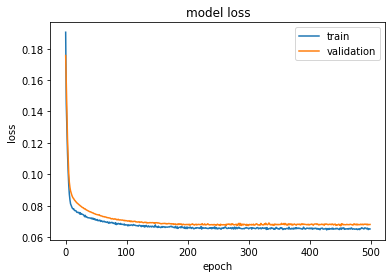


Fig. 4.3 Loss function plot for LSTM

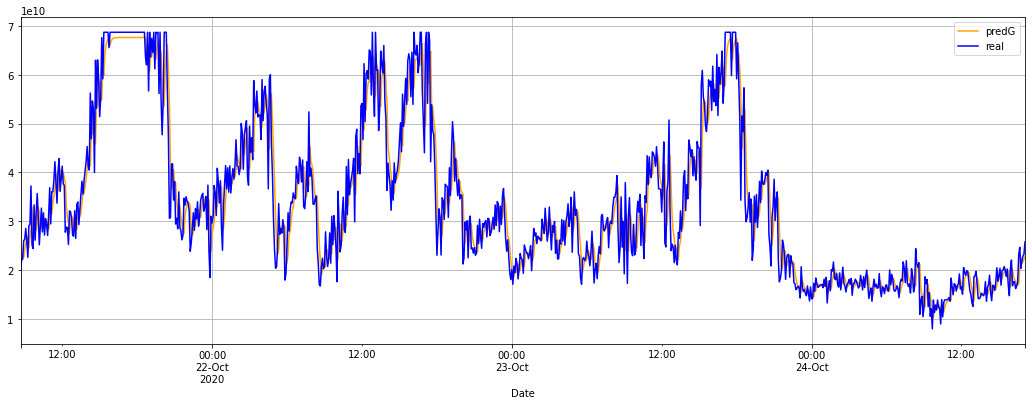
0

Fig. 4.4 GRU vs real data plot

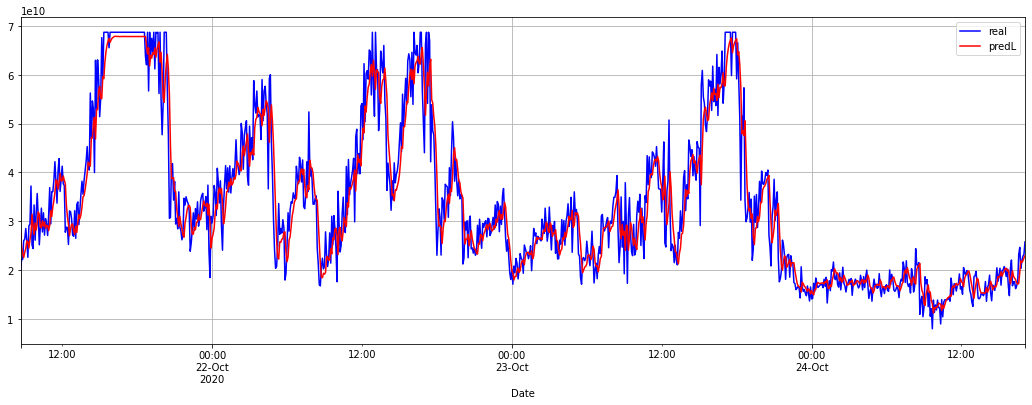


Fig. 4.5 LSTM vs real data plot

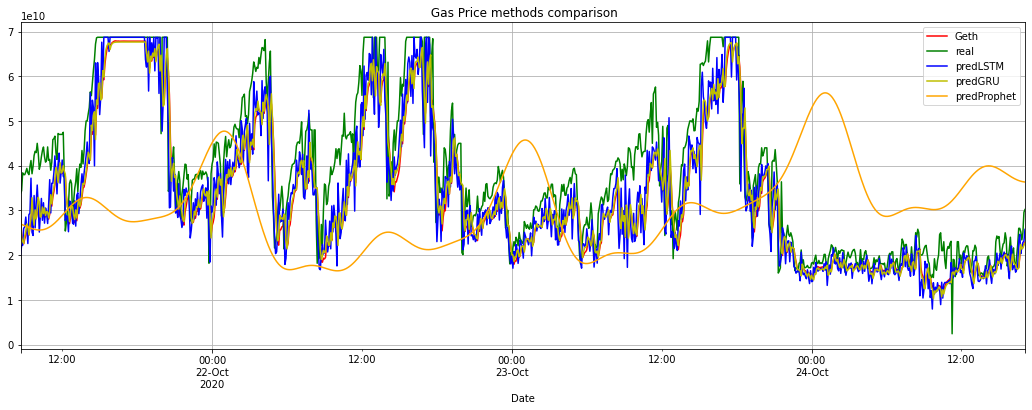


Fig. 4.6 Comparison of models with geth gas price oracle plot

# CHAPTER 5

**Conclusion and Future Plans**

## Conclusion:

In the proposed approach, sub-daily Ethereum gas price data was simulated using Prophet, LSTM and GRU models. The historical gas prices data are compared with the three prediction models and Geth predicted data in the same period. In addition, error metrics such as MSE, RMSE, MAE, R-squared value were used to calculate the error rate of different approaches. The results prove that LSTM and GRU models outperformed Geth oracle and Prophet model. Thus a novel approach to predict Ethereum gas price based on deep learning models is presented

successfully.

## Future Works:

* + - Future work could explore more in-depth data analytics of ethereum transactions and their attributes, other deep learning techniques and extend more data to compare them with other gas price oracles, such us EthGasStation.
    - Future work also aim to release a Web application based on the model proposed that would be available to any Ethereum user to make gas price predictions before making a transaction on the network.

# CHAPTER 6

**References**

**[1]** **Nakamoto, Satoshi, and A.** Bitcoin. "A peer-to-peer electronic cash system." Bitcoin.–URL: https://bitcoin.org/bitcoin.pdf 4 (2008).

**[2]** **Wood, Gavin.** "Ethereum: A secure decentralised generalized transaction ledger." Ethereum project yellow paper151.2014 (2014): 1-32.

**[3]** **Buterin, Vitalik.** "A next-generation smart contract and decentralized application platform." white paper 3.37 (2014).

**[4] Werner, Sam M., Paul J. Pritz, and Daniel Perez.** "Step on the gas? A better approach for recommending the ethereum gas price." Mathematical Research for Blockchain Economy. Springer, Cham, 2020. 161-177.

**[5] Ethgasstation:** Eth gas price recommendations. https://ethgasstation. info/. Accessed: 2020-11-24

**[6] Pierro, Giuseppe Antonio, and Henrique Rocha.** "The influence factors on ethereum transaction fees." 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). IEEE, 2019.

**[7] Selvin, Sreelekshmy, et al.** "Stock price prediction using LSTM, RNN and CNN-sliding window model." 2017 International conference on advances in computing, communications and informatics (icacci). IEEE, 2017.

**[8] Ducasse, Stéphane, et al.** "Smartanvil: Open-source tool suite for smart contract analysis." (2019).

# CHAPTER 7

**Appendix**

## Literature Survey

|  |  |  |
| --- | --- | --- |
| **Literature** | **Proposed Method** | **Limitations** |
| **Werner et al[4]** | Presented a gas price recommendation approach that combines a deep learning-based price forecasting model as well as an algorithm parameterized by an urgency value that can be set by the user to recommend gas prices. The proposed recommendation strategy accepts a single parameter representing urgency. The urgency parameter is used to balance the gas price with the waiting time: the lower the urgency is, the lower the gas price is and thus the longer the waiting time be | Though the paper presents a recommendation approach which economizes the gas price to pay, the recommended gas price would likely need to be increased to ensure timely inclusion in a block |
| **Ducasse et al[8]** | Suggested an open-source platform for the analysis of Blockchains called SmartAnvil, their work focuses on Ethereum Blockchains and contracts written in Solidity language. | Our gas price prediction will trigger their future work of adding gas estimation functionality |
| **Selvin, Sreelekshmy, et al[7]** | Presented the Stock price prediction model using LSTM, RNN and CNN | Most of the time-series forecasting papers have not explored the ether gas price as a problem statement. |
| **Ethgasstation[5] Eth gas price recommendations. https://ethgasstation. info/. Accessed: 2020-11-24.** | It is a third-party tool. It generates adaptive gas price estimates allowing the users to know the appropriate price to use according to their confirmation time requirements using a Poisson regression model based on data from the previous 10,000 blocks. | Pierro and Rocha [6] investigated the impact of causality by external factors on transaction fees in Ethereum and the accuracy of the gas price recommendation given by the EthGasStation oracle and  concluded that the existing gas price recommendations provided to users are not accurate, although they focus on data from a specific oracle service, where the margin of error was up to 28%. |