Comparison of Different Recurrent Neural Networks for Forecasting Cryptocurrency Price Using Small Dataset and its Modifications

Name	SID
Ip Ho Yin	1155

***Can I get higher marks as I finish it myself, since I can't find teammate interested on this topic**

The content of this paper is as follows:

I. Introduction

In recent years, Cryptocurrencies have been a hot topic. It is a form of digital product with a decentralized structure. The purpose of Cryptocurrencies is to build up a market without control of governments and third parties.

This project is going to find out the predicting power of Recurrent Neural Network (RNN) in a small dataset. For the RNN part, this report will use python. The modification part will also be included as well.

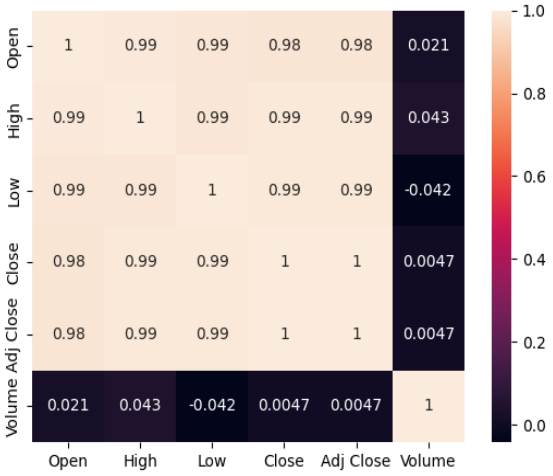
In this project, 10 different cryptocurrency stocks will be used for experiments' generalization. A small dataset will be used in this project. Since there is much research on RNN models proving that the LSTM model will always perform the best in similar tesk, the worst is SimpleRNN. However, they mostly undergo an ideal situation. It is valuable to compare the performance when a small dataset is used. Since it is not always perfect in the real world, the data may not be good enough. The cost of collecting data may be so high in special situations. Therefore, deep learning with a small dataset is a good research idea.

In this project, we want to find out:

- Which RNN model is the best ?
- How to modify the RNN models ?
- What is their performance under different settings?

II. Dataset

The data download from Yahoo Finance. Each .csv file includes 1 year + 7 days data, it is a small dataset. The first 360 days will be used for training and the last 7 days will be used for testing. Since cryptocurrency stock is not stopped by holidays. It means that we do not need to worry about the data preprocessing will be too much trouble.



It is the correlation matrix. Coding part: SEEM_correlation.py

'Adj Close' is highly correlated with other features and the open price of the next day. It matches the expectation. Since we need to predict the price of cryptocurrency and 'Adj Close' has high correlation, dimensionality reduction will not be used. Besides, the normal dimensionality reduction method may not work for time series data.

<u>Data Preposcessing</u>

Since 5 timesteps are used. For each cryptocurrency, we need to split the training data as

$$x_i^{train} = (x_i, \dots, x_{i+4}) \text{ and } y_i^{train} = x_{i+5}$$

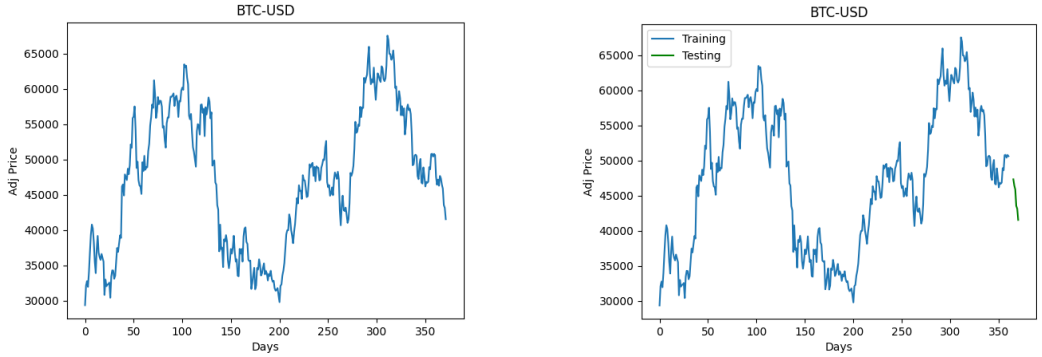
$$where\ x_i^{train}\ is\ the\ features\ of\ training\ data,\ y_i^{train}\ is\ the\ label\ of\ training\ data.$$

$$i < 361\ as\ we\ just\ use\ one\ year\ data.$$

$$when\ i = 361, the\ next\ year's\ data\ will\ be\ used\ as\ a\ label.$$

For testing data, we will predict next week's data after one year.

$$i.e.\ x_i^{test} = (x_i, \dots, x_{i+4})\ and\ y_i^{test} = x_{i+5},\ 360 < i < 368$$



Left: The BTC Adj price
Right: Predicting the green line by using the blue line as training data.
Coding part: SEEM_correlation.py

III. RNN model

Since the time series data is autocorrelated, i.e. the day i data may related to day $i-1$, the day $i-1$ data may related to day $i-2$. The RNN model can simulate this property by adding the previous data with weights and bias to the new data in the same layer. The relation between these data is not only perpendicular, but also parallel. That is why RNN works for time series time like stock or NLP. Three kinds of RNN will be used. They are Simple RNN (SimpleRNN), Long Short-term Memory (LSTM) and Gate Recurrent Unit (GRU). We set 5 as the timestep.

Experiment setup

Mean square error will be used as a loss (cost) function.

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

The Adam optimizer will be choosed. It combines the Momentum and RMSprop methods. The Momentum method can make sure the optimal problem finding a global minimum instead of local minimum. The RMSprop methods can ensure the optimal problem will not jump across the minimum by lessening the gradient.

$$\omega_d^{(t+1)} = \omega_d^{(t)} + \frac{\eta}{\sqrt{\hat{v}_d^{(t)} + \varepsilon}}\hat{m}_d^{(t)}, \; for\; d = 1, \ldots, D \;\text{ where } m_d^{(t)} = \beta_1 m_d^{(t-1)} - (1-\beta_1) g_d^{(t)}$$

$$v_d^{(t)} = \beta_2 v_d^{(t-1)} + (1-\beta_2)(g_d^{(t)})^2 \qquad \hat{m}_d^{(t)} = \frac{m_d^{(t)}}{1-\beta_1^{(t)}} \qquad \hat{v}_d^{(t)} = \frac{v_d^{(t)}}{1-\beta_2^{t}}$$

Also, since the value of the price of cryptocurrency is large, the ReLU activation function will be used. The sigmoid function will be used in the model modification (normalization part) section.

$$y = max(0, x)$$

The mini-batch gradient descent will be used and set the batch size as 32, each model is going to train 3 times on each cryptocurrency. As 3 trials are enough to get a good performance. The number of trials can increase if needed.

RNN Model Description

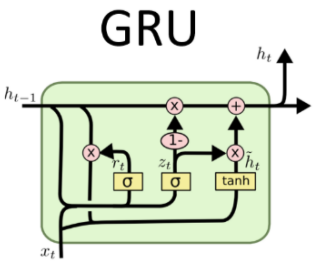
1. For Simple RNN, each neuron will add up the previous neuron with weight and bias that is in the same layer. Then, plug in the activation function and move it to the next layer. It is a very typical RNN model.

$$
\begin{aligned}
&(\textit{Cell output}) && \boldsymbol{h}_t^{(\ell)} = \boldsymbol{f}_\ell(\boldsymbol{W}_h^{(\ell)}\boldsymbol{h}_t^{(\ell-1)} + \boldsymbol{V}_h^{(\ell)}\boldsymbol{h}_{t-1}^{(\ell)} + \boldsymbol{b}_h^{(\ell)}), \quad \text{for} \quad \ell = 1, \cdots, L-1; \\
&(\textit{Final output}) && \boldsymbol{o}_t = \boldsymbol{\sigma}(\boldsymbol{W}_o\boldsymbol{h}_t^{(L-1)} + \boldsymbol{b}_o).
\end{aligned}
$$

Source: STAT4012

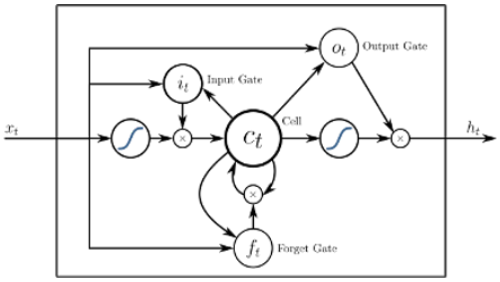
Different from the SimpleRNN model, the GRU model and LSTM model will undergoes different meaningful procedures by using several 'gates'.

2. For GRU model, it is a sophisticated version of SimpleRNN model, two gates are added to each neuron.
Reset gate: The gate that decides the previous data is going to forget or not.
Update gate: Updating the data and keeping the data.



Source:https://www.tutorialexample.com/lstm-vs-gru-network-which-has-better-performance-deep-learning-tutorial/

3. Similar to GRU, but LSTM will add more complicated procedures with three gates in each neuron.



Forget gate: The gate that decides which data is going to keep or not by sigmoid function.
Input date: To decide which data are used to update this neuron.
Output gate: Decide which data to be moving to the next neuron.

Source: https://en.wikipedia.org/wiki/Long_short-term_memory

RNN Model Evaluation and Report (Remember change the model yourself)

Coding part: SEEM_Project_BasicRNN.py
The graphs are included in the project file.

RNN (time) *Crypto*	*Basic SimpleRNN(time)*	*Basic LSTM (time)*	*Basic GRU (time)*
ADA	0.00248(7.7)	0.00241(10.8)	0.002972(11.3)
AVAX	114.1487(7.5)	179.6635(10.6)	107.3557(11.3)
BNB	463.511(7.6)	928.60(10.6)	351.9465(11.3)
BTC	1359434.7(7.6)	2552339.7(10.6)	1776321.9(11.4)
DOGE	0.000032(7.8)	0.000043(10.3)	0.000044(11.3)
ETH	29380.91(7.7)	49161.17(10.6)	22495.29(11.4)
HEX	0.000683(7.6)	0.00081(10.7)	0.000758(11.4)
LUNA1	34.11595(7.6)	27.8095(10.6)	31.4764(11.1)
SOL	105.0189(7.5)	229.2197(10.6)	101.1529(11.4)
XRP	0.000777(7.6)	0.002547(10.6)	0.000838(11.4)
Mean Time	2.54s for each	3.53s for each	3.78s for each

Remark: The graph shown in the appendix and other files.
****The value looks terrible since it is not normalized** **The best result** **Near the best (difference < 10%)**

From the table above, the SimpleRNN model is the best. At the same time, it is the fastest compared with other models. GRU models also perform quite well. The worst performance is the LSTM model.

Theoretically, the LSTM requires the most parameters compared with other two RNN models, it should perform the best. As it is very flexible because of lots of trainable parameters. The reason why LSTM is the worst is that we are using a small dataset this time. Normally, the LSTM model always needs a large amount of data for training the parameters by backpropagation. A lack of data will cause the model to not move to its optimal solution (minimum point in this report). Even adding move epochs does not really work, as it is easy to cause overfitting.

It is very interesting to see that it is totally different from the scene. For a small dataset: Less is more. It also shows the limitations of deep learning.

Besides, the table shows the time used of GRU model is longer than LSTM model. It is theoretically not reasonable. Since the trainable parameters of GRU model is less than LSTM model. One conjecture is that the LSTM model has better implementation in Tensorflow than GRU model. Now, faster models are introduced, they are CuDNNLSTM and CuDNNGRU. Further findings will be included in the next part.

RNN Model Modification

Since GRU model and LSTM model are not performing as well as expected, it is going to find out some methods to enhance the performance.

In this section, we only use numeric metrics to find out the improvement of the model, so the graph will not be included. BTC is used here.

Method 1: Using different data sizes and timestep for training

Coding part: SEEM_Project_Mod1.py

Larger data size and longer timestep

BTC	*SimpleRNN*	*LSTM*	*GRU*
Larger data size and longer timestep	4293119	7248595	3446878
Larger data size and shorter timestep	3137300	2971379	2133746
Smaller data size and longer timestep	3678256	6708832	1847545
Smaller data size and shorter timestep	1677370	2184644*	1325561*

***Perform better than basic model**

Results and Findings

It can see that the performance of both GRU and LSTM are improved. So, using Shorter timestep or smaller data size can enhance the performance.

It can be evident that the early data may not always be useful for training the data. Since cryptocurrency is very sensitive to the recent market/world situation. For example, the cryptocurrencies price is always affected by Government policy or Elon Reeve Musk's twitter. Now, the price may be affected by the war. So, the early data may mislead the RNN model. Since the RNN model always treats the newest data and the previous data as the same, we actually want the RNN to put a higher weight on the newest data.

It is very interesting. Since for complex models like LSTM or GRU, we want to put more data for the training. However, the early data will be misleading. Also, the simple model can not be as flexible as complex models. By these two phenomena, it means that we can not predict the price with high accuracy as these two limitations always exist.

So, how to find a suitable time step and time interval for the train is a good problem. Each cryptocurrency may be different.

Method 2: Deeper and wider deep neural network (may cause gradient vanishing)
There are too many choices for this method, as different selection of depth or width will cause gradient vanishing or gradient explosion or the degradation problem or overfitting or underfitting easily. Since the network is too deep it will cause the gradient to be too small. This is not stable and there are too many choices, so do not try here.

Besides, if this method is to be used, grid search or genetic algorithms can be very useful for finding good hyperparameters.

Therefore, we may change the structure of the recurrent/deep neural network instead of just making it deeper or wider. It will be discussed on **method 5**.

Method 3: Normalization and use sigmoid function
Extreme values of data will cause the value of backpropagation to become too small or too large, it will affect the whole model performance. So, normalizing the data will be a good choice to avoid this problem.

Remark: Since normalization will convert the value, 3 cryptocurrencies will be used as testing data to view the changes. They refer to large value (BTC), middle value (SOL) and small value (XRP) data.

Results and Findings
Coding part: SEEM_Project_Mod3.py

	SimpleRNN	*LSTM*	*GRU*
BTC	1062972*	1210687*	1113774*
SOL	162.9588	164.6870*	159.4772
XRP	0.0005068*	0.0005151*	0.0004833*

***Perform better than basic model (New) the best (New) near the best**

Some of them become better than the basic model. So, it is a good method for this time.
Method 4: Model selection method and hyperparameters tuning
\The old data may not relate to our predicting target. We can choose the best model by choosing the model that has the lowest MSE when predicting the newest data. i.e., increasing the total training times, number of epochs, and the metric of best loss. That is the original metric of best loss in basic model:

```
self.best_model = model
best_loss = model.evaluate(X_train[-50:], y_train[-50:])
```

To choose the best model that relates to the newest data, we change it from 50 to 10.
Also, change epochs from 100 to 200 and ntry from 2 to 5.

Results and Findings
Coding part: SEEM_Project_Mod4.py

	SimpleRNN	*LSTM*	*GRU*
BTC	1316202	1733679*	1232101*

***Perform better than basic model**

By the result, it is easy to see that the performance has become better now.

Method 5: Using different architecture of model
Remark: This part is complicated, so 3 cryptocurrencies will be used as testing data. They refer to large value (BTC), middle value (SOL) and small value (XRP) data.

Using Conv1d as data preprocessing with a dense layer.
The convolution neural network is a feature extraction procedure. It will try to find out some pattern between the time intervals. Then, each pattern is autocorrelated. For example, the pattern found from [1, 2, 3, 4, 6] is [-2, -2, -3] by filter [1, 0, -1] with bias 0. The filter finds an increasing pattern when the output is negative. However, each negative value in output will also have a relationship, they depend on their previous values, as their original data is autocorrelated. So, RNN can be applied after the Conv1D layer. Afterward, a dense layer will be used for connecting different patterns (produced by different filters).

Our padding is 'same' ; it keeps the output dimension the same as the input dimension in the Conv1D layer.

Results and Findings
Coding part: SEEM_Project_Mod5.0.py, SEEM_Project_RNNMod5.0_normalized.py

	Conv1D+simpleRNN	*Conv1D+LSTM*	*Conv1D+GRU*
BTC	2323537	#2379542*	2744718/#1880173
SOL	58.7140*	102.0406*	67.8437*
XRP	0.000593*	0.000666*	0.000531*

***Perform better than basic model #This data is predicted under normalized**
(New) the best (New) near the best Remark: Some of value have not shown as the MSE is too high

Except for the large value data, these three models improved the performance. However, the improvement in small value is not as good as method 3. **The middle value is better than method 3.**

Using Bidirectional model
BiSimpleRNN, BiLSTM and BiGRU refer to Bidirectional LSTM and Bidirectional GRU.

The ‘Bidirectional’ means the data are not only put into the RNN layer in order, they are also put into another RNN layer in reverse of the original input sequence. Then, they will be combined in the output layer.

Results and Findings

Coding part: SEEM_Project_Mod5.1.py, SEEM_Project_RNNMod5.1_normalized.py

	BiSimpleRNN	*BiLSTM*	*BiGRU*
BTC	2205273/#2267217	#1782120*	#1551594.45*
SOL	80.83*/#108.35	131*/#149*	86.4787*/#126
XRP	0.00051*/#0.00057*	0.00075*/#0.00066*	0.00137/#0.00052*

***Perform better than basic model #This data is predicted under normalized**

(New) the best (New) near the best Remark: Some of value have not shown as the MSE is too high

Using this method is not as good as method 3 when predicting small and large value data. At the same time, the middle value prediction is not better than the previous Conv1D+RNN model.

Method 6: Mix different methods

Since a more complex model is difficult to mix together as there is no way to know what is happening between the neural network. The performance may become worse than before when these complicated models are mixed. Method 1 and method 4 can easily combine together. Therefore, just method 1 and 4 are chosen to mix.

By trying to shorten the train set and shorten timestep separately, it found that shortening the train set does not help in improving the performance. From previous methods, the GRU model always performs better than the LSTM model, so the GRU model will be choosed this time.

Results and Findings

Coding part: SEEM_Project_RNNFinalGRU.py

Crypto	*Basic SimpleRNN*	*Modified GRU*
ADA	0.00248	0.00427
AVAX	114.1487	24.2847*
BNB	463.511	249.741*
BTC	1359434.7	1244307*
DOGE	0.000032	0.000631

ETH	29380.91	25695.83
HEX	0.000683	0.000738*
LUNA1	34.11595	25.2177*
SOL	105.0189	109.7524
XRP	0.000777	0.00049*

***Perform better than basic model**

The best result in this table **near the best**

Therefore, when handling the data with small values, we can use SimpleRNN. On the other hand, we can use GRU for data with large values.

IV. Conclusion and Discussion

Model Performance: By above table, both SimpleRNN and GRU can give a good performance on predicting the price and trends (refer to graphs) in different aspects.
Problem with LSTM: The problem with LSTM is that it will be useless when the data size is small. There is not enough data to train the parameter as it has three gates. In the SimpleRNN layer, there are just 35 parameters needed to train, but GRU is 120, in the LSTM that will be 140. So, LSTM needs relatively more data to train the parameters in order to get the optimal result. That is why the LSTM may not perform better than others in the basic model. Besides, increasing the epochs is not an ideal method, as it can make LSTM perform better, but it will cause overfitting easily. Therefore, we mainly use SimpleRNN and GRU for comparison this time.
MSE and Time Trade-off: Although RNN performs better than ANN, there is a trade off between the relatively low error and running time. Even though SimpleRNN is simpler than GRU and LSTM, it still needs more time on calculation than ANN (higher time cost). Of course, the fastest way must be the traditional time series model such as AR, MA, ARIMA.
Difference between GRU and SimpleRNN: In the last part table, the GRU model can predict the large value data with the lowest error compared with other two RNN. Correspondingly, the SimpleRNN model can get a good performance in small value data compared with GRU. One conjecture is that GRU is more complex than SimpleRNN. When we increase the ecoph, the error of GRU decreases (except the small value data). Therefore, combining these two observations, the gradient vanishing may occured in GRU.
Suggestion for further improvement: The best way to predict each cryptocurrency is to customize the model for each cryptocurrency. The easiest way is finding a suitable (too early data will lead to misleading as we discuss in RNN part) large data size to train the model with regularization terms. Besides, grid search to find out the best number of neurons in a dense layer is one way to optimize. **I did not try it as this project has limited the number of pages.**

All of the coding part is stored in different .py files.

Appendix

References

Fernando, J. (2022, April 6). *Autoregressive*. Investopedia. Retrieved May 2, 2022, from https://www.investopedia.com/terms/a/autoregressive.asp

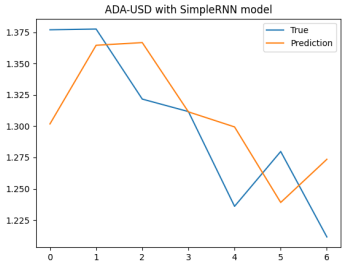
Ognjanovski, G. (2020, June 7). *Everything you need to know about neural networks and backpropagation - machine learning made easy...* Medium. Retrieved May 2, 2022, from https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a

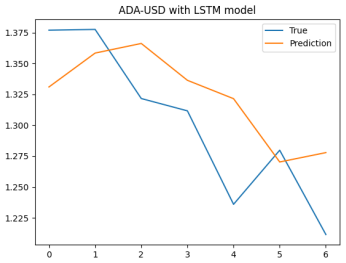
Sagar, A. (2019, December 3). *Cryptocurrency Price Prediction Using Deep Learning*. Medium. Retrieved May 2, 2022, from https://towardsdatascience.com/cryptocurrency-price-prediction-using-deep-learning-70cfca50dd3a

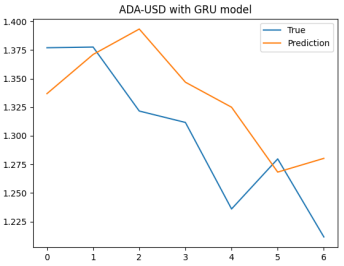
Graph of Basic RNN model

Simple RNN LSTM GRU

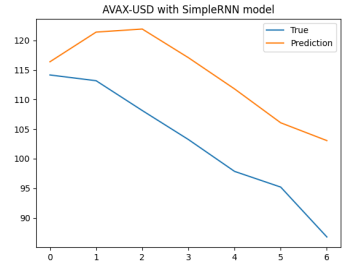
ADA

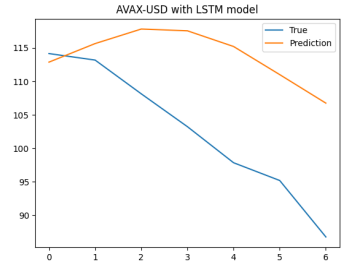


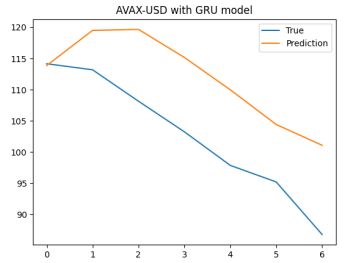




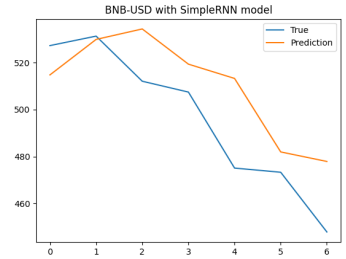
AVAX

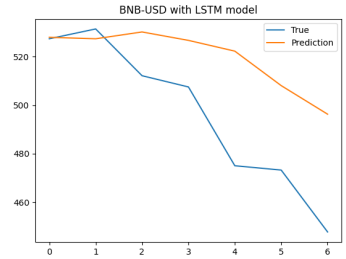


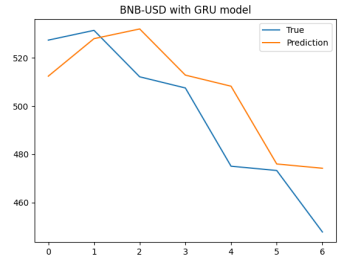




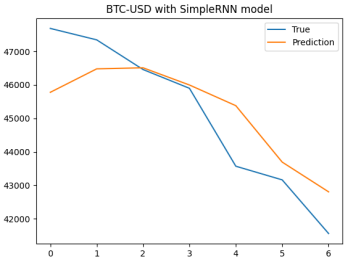
BNB

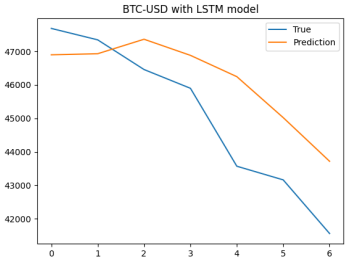


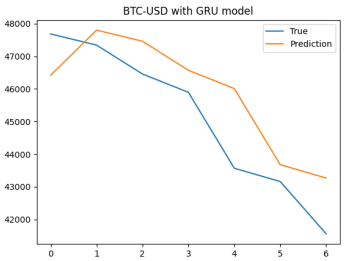




BTC

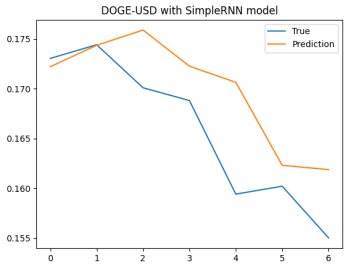


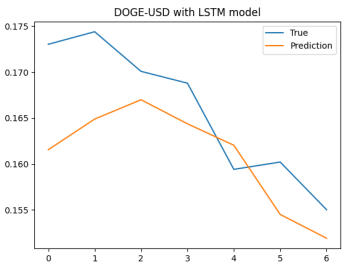


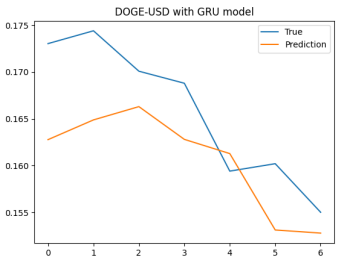


Simple RNN LSTM GRU

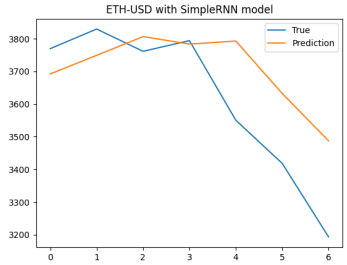
DOGE

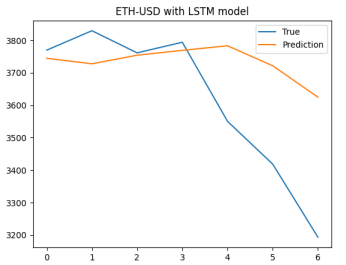


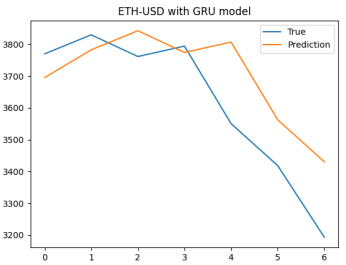




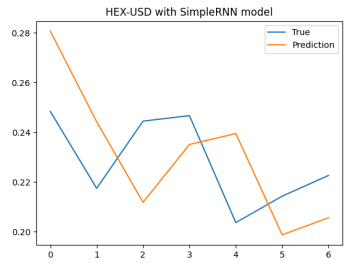
ETH

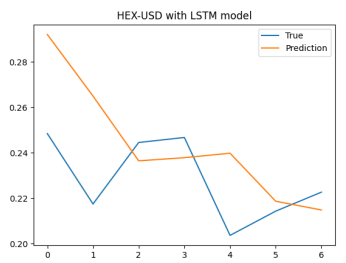


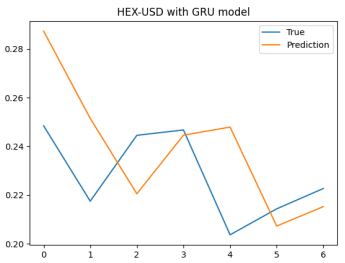




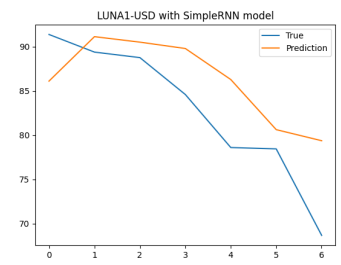
HEX

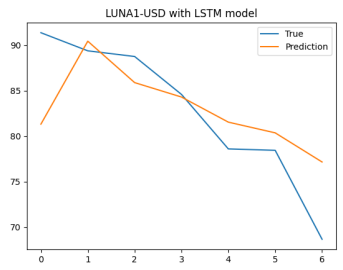


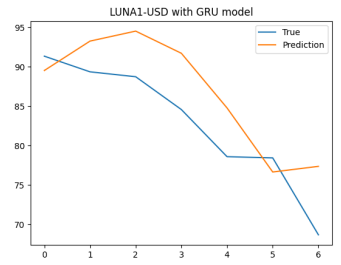




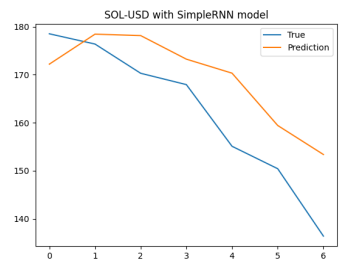
LUNA

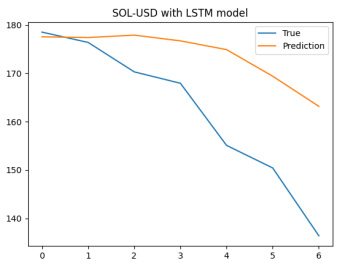


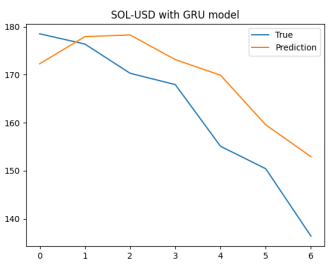




SOL







Simple RNN LSTM GRU

XRP