Jacob Coles

LT2326 Project: R.O.C.N.E.T. Regression of Crypto Network

Introduction

This project is an investigation into time-series forcasting using pytorch; specifically predicting the future price of cryptocurrencies using RNNs. In this project, a number of experiments were performed to evaluate the performance improvement by changing parts of the model or using different features. This would include:

- Comparing the performance of LSTM vs GRU for series prediction
- Scaling the input and output of the model (with the idea in mind to make the system more robust to data at different scales)
- Adding bias in the data
- Introducing the additional feature of ‘sentiment’ from text-based analysis as a feature to predict (was not accomplished in this report)

Motivation

Complex analytical models and algorithms exist for predicting the future price of stocks and cryptocurrencies, however there is a limit to how well the future (price) can be predicted based on historic price (and market-related features). If it were possible to get good predictions with a machine learning model, then it might be possible to add more features which aren’t typically used in statistical models.

Firstly, we can see if it possible to create a machine-learning model which can use the same features as statistical models; and secondly, is it possible to add more features (i.e. more information) to improve on these predictions.

Due to time limitations, only the first of these questions was investigated for this report. A web-scraper was built to get text features from reddit however this was not included, as the language-based model wasn’t implemented.

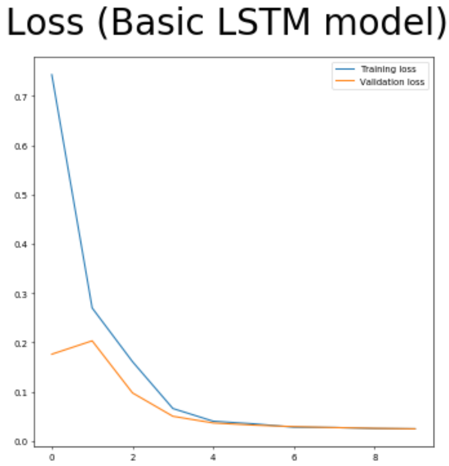
Design/Research

Model 1: LSTM

Our first model will be an LSTM with a couple of linear layers for the output prediction. We will begin with a relatively 'basic' model, and we will perform experiments to see if we can improve on it. This model takes a sequence of the price (where price is the only feature in the sequence), and for each sequence the next price is output. We scale the input to the model between 0 and 1, since our LSTM will have difficulty determining patterns when cryptocurrencies exist on such different scales (for example, Bitcoin is valued over 10000 times the value of Dogecoin). On the output of the lstm, we take the hidden dimensions for the final value (that is, the vector corresponding to the next price to be predicted), and pass it through a couple of small linear layers to get the final price on the output.

We limit the length of sequences to 60 timesteps (days) due to the fact that it's the largest we can do given the smallest set (cryptocurrency) in our dataset. This also ensures that the model generalises over a relatively recent past. Besides this, an LSTM would suffer from the vanishing gradient problem, so there would be little benefit to having a large amount of timesteps.

Loss of the model is as follows:



Training loss: 0.027502658995217644
Validation loss: 0.023446537768904818

Model 2: LSTM with non-linear scaling

A problem with LSTMs and time series data is that it is difficult to generalise over a wide (in this case, price) scale. This is why we scaled the data to fit between 0 and 1 in preprocessing. An experiment we can perform here is to see if scaling the data again in a non-linear way can help.

The function we will use to the input of the LSTM is:

$$log(input\ +\ 1)$$

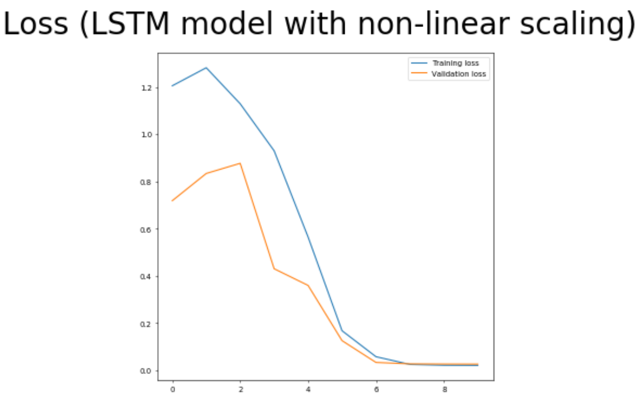
We will do the reverse scaling on the output with:

$$e^{output}-1$$

This will scale down large values more than it scales down small values. The idea behind this is to allow price changes on a small scale to have a larger effect than equivalent changes on a large scale.

For example, if a cryptocurrency with market value $1 goes up in value by $1, then its value is doubled, which is a significant change. If a currency of value $100 goes up by $1, then it is a relatively small change.

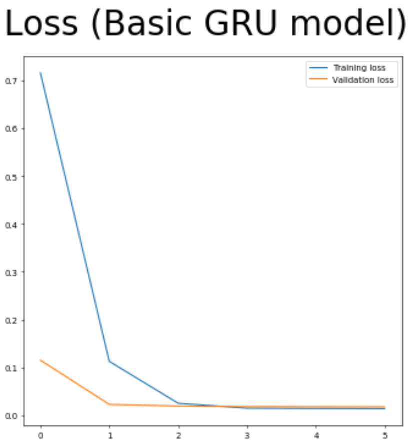
The loss of the model is as follows:



Training loss: 0.030352314159244997
Validation loss: 0.023219682298076805

Model 3: GRU

We can also test using another variant of RNN, a GRU (Gated Recurrent Unit) and compare it to the LSTM.



Training loss: 0.014004082099972948
Validation loss: 0.017623749396079802

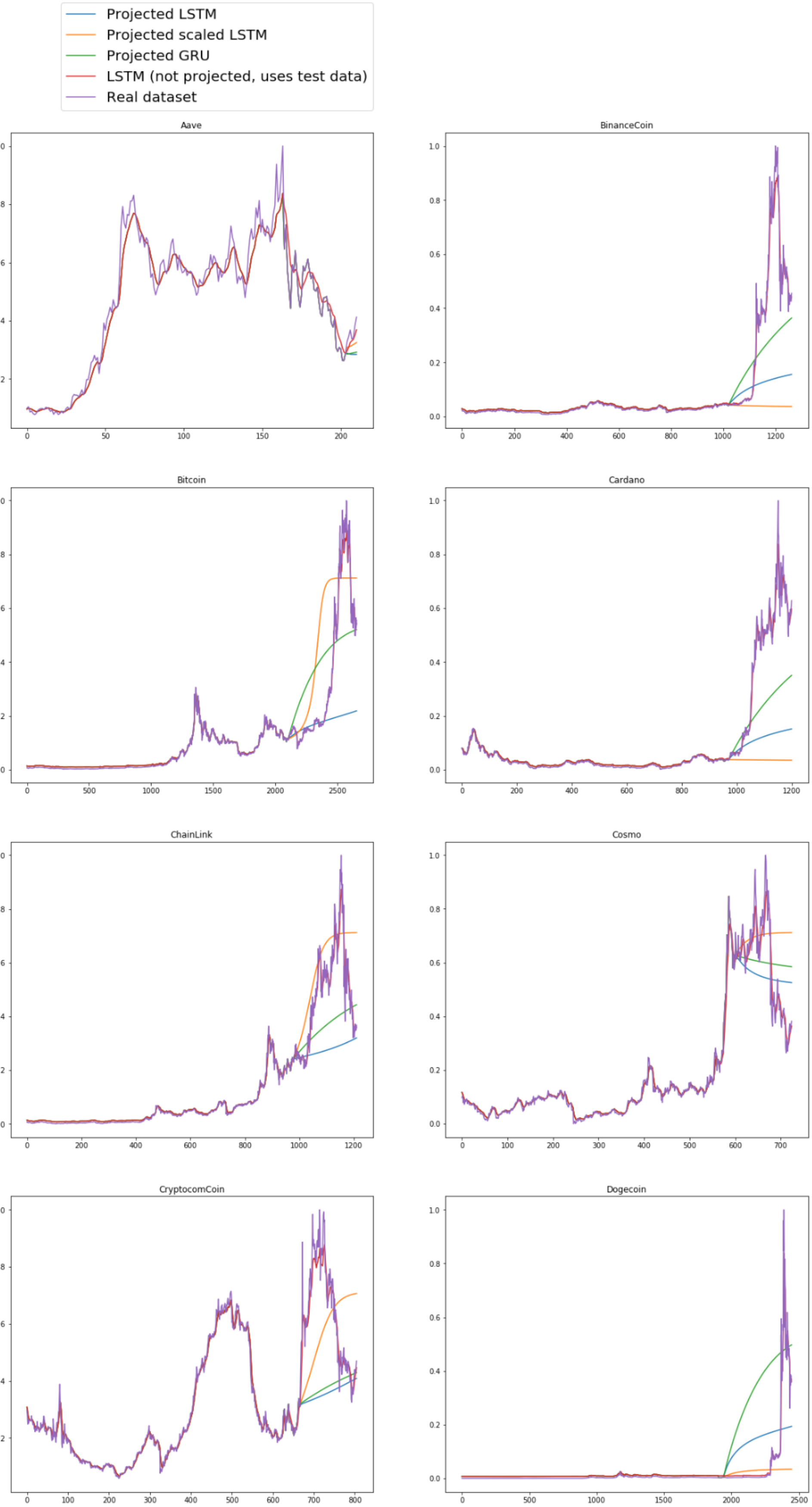
Evaluation/Results

We can now inspect the solutions with graphs. We will look at:

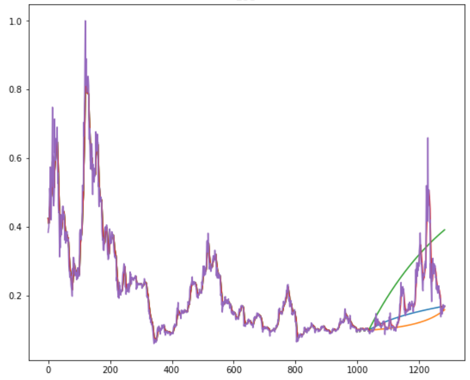
- The gold standard
- The output from the LSTM (predicting each time step from the 'real' data)
- The projected outputs for each of the models (corresponds to the size of the test set)

We can also inspect the MSE of the projected outputs of the models (not just the MSE from the training/validation/test sets).

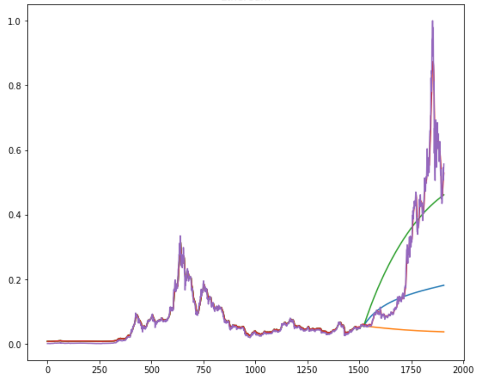
Model comparison



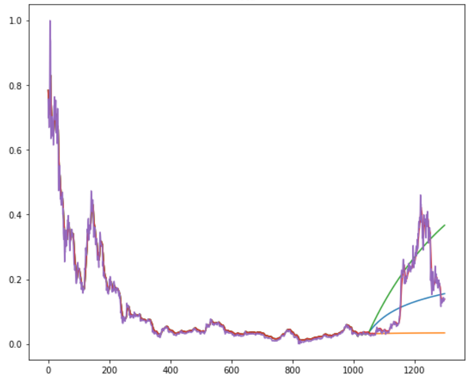
EOS



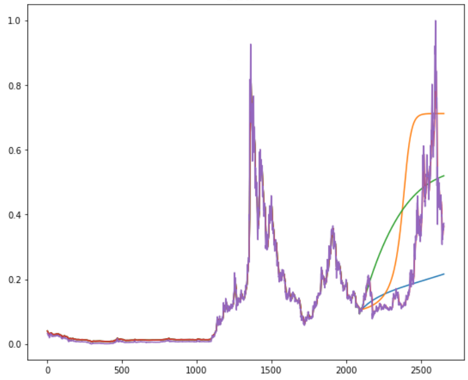
Ethereum



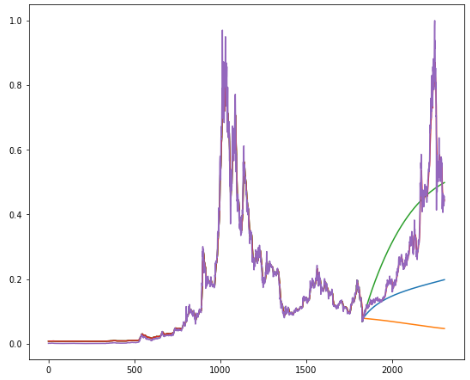
Iota



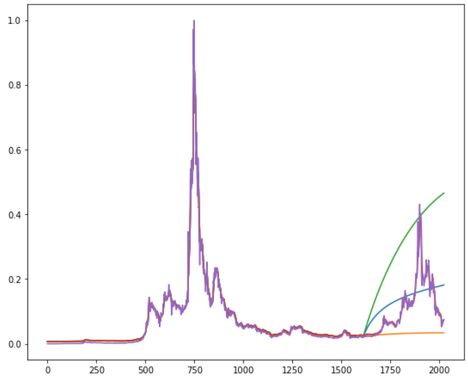
Litecoin



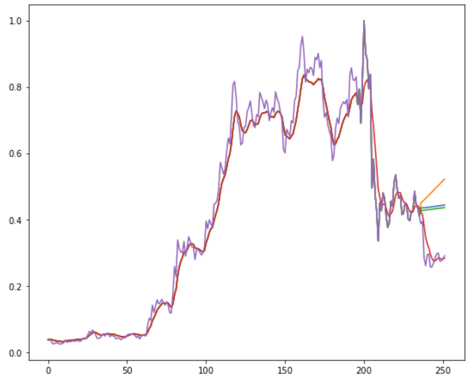
Monero



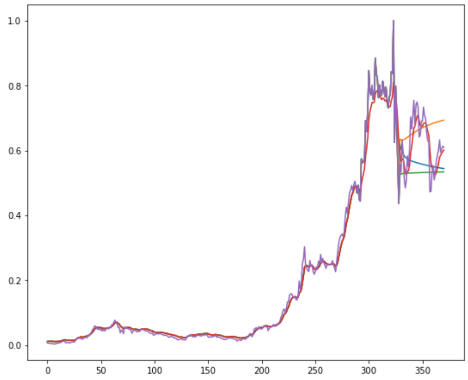
NEM

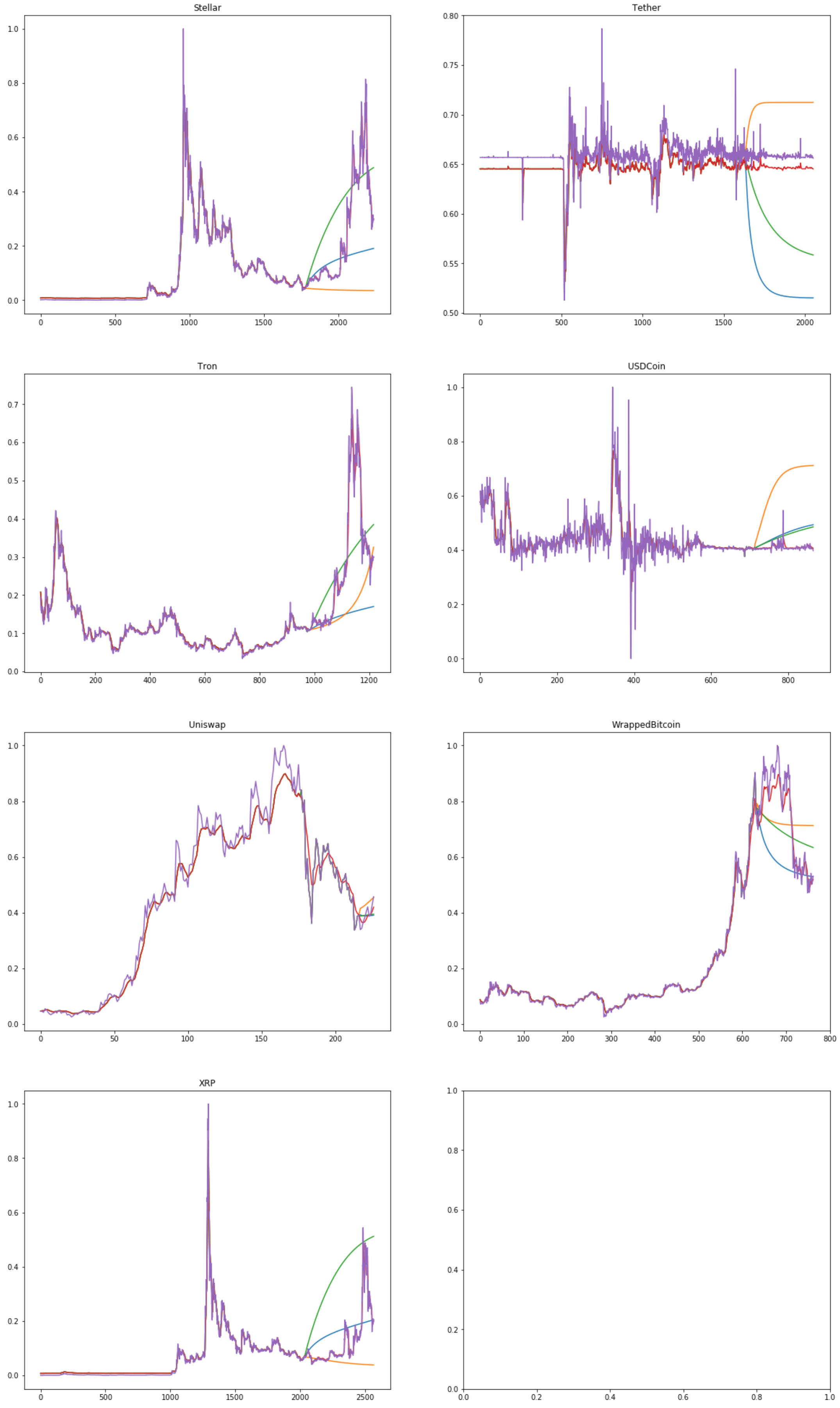


Polkadot



Solana




Stellar
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
Tether
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
500
1000
1500
2000
Tron
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
200
400
600
800
1000
1200
USDCoin
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
Uniswap
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
WrappedBitcoin
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
600
700
800
XRP
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
2500
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

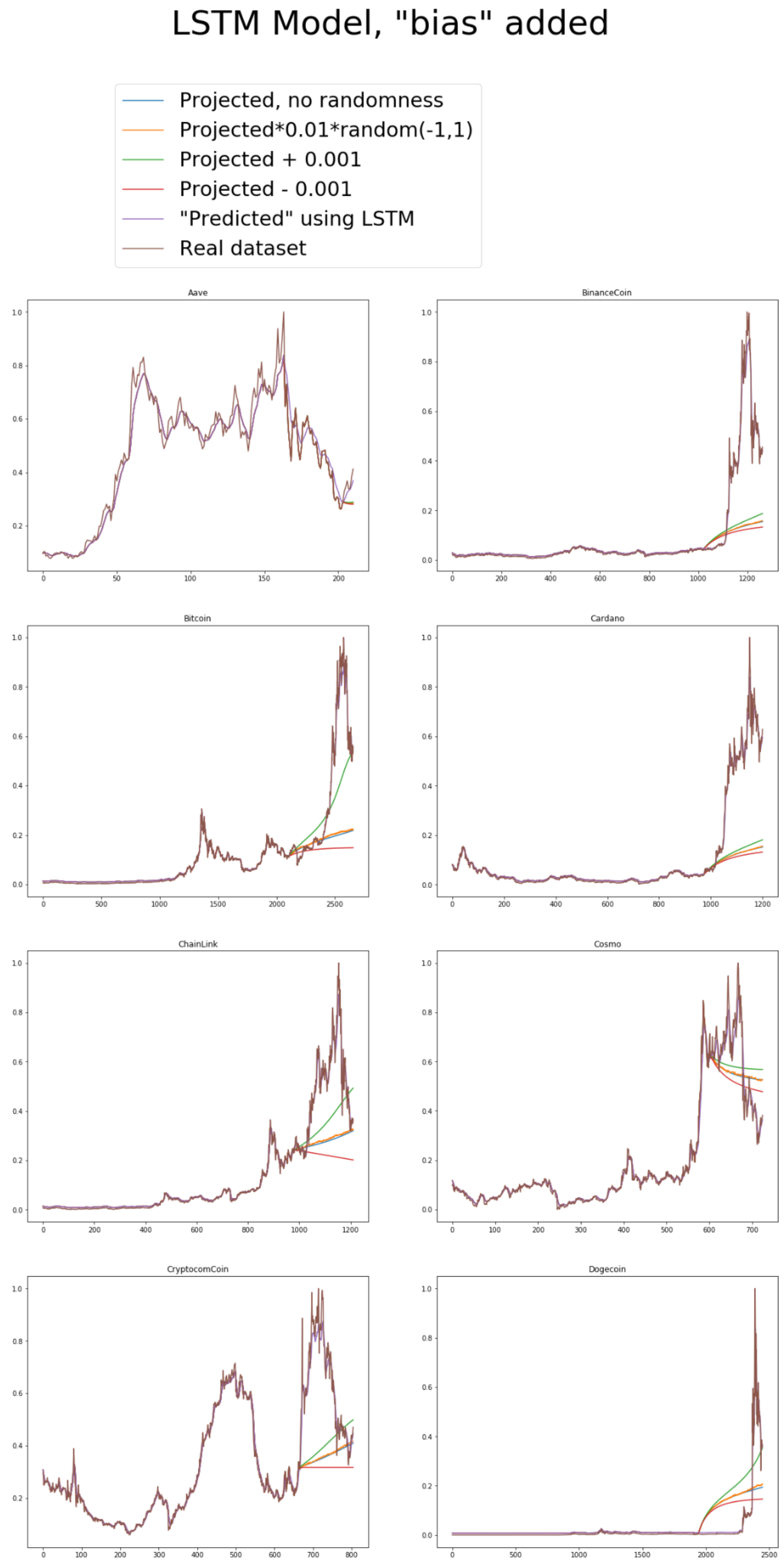
MSE sum for:

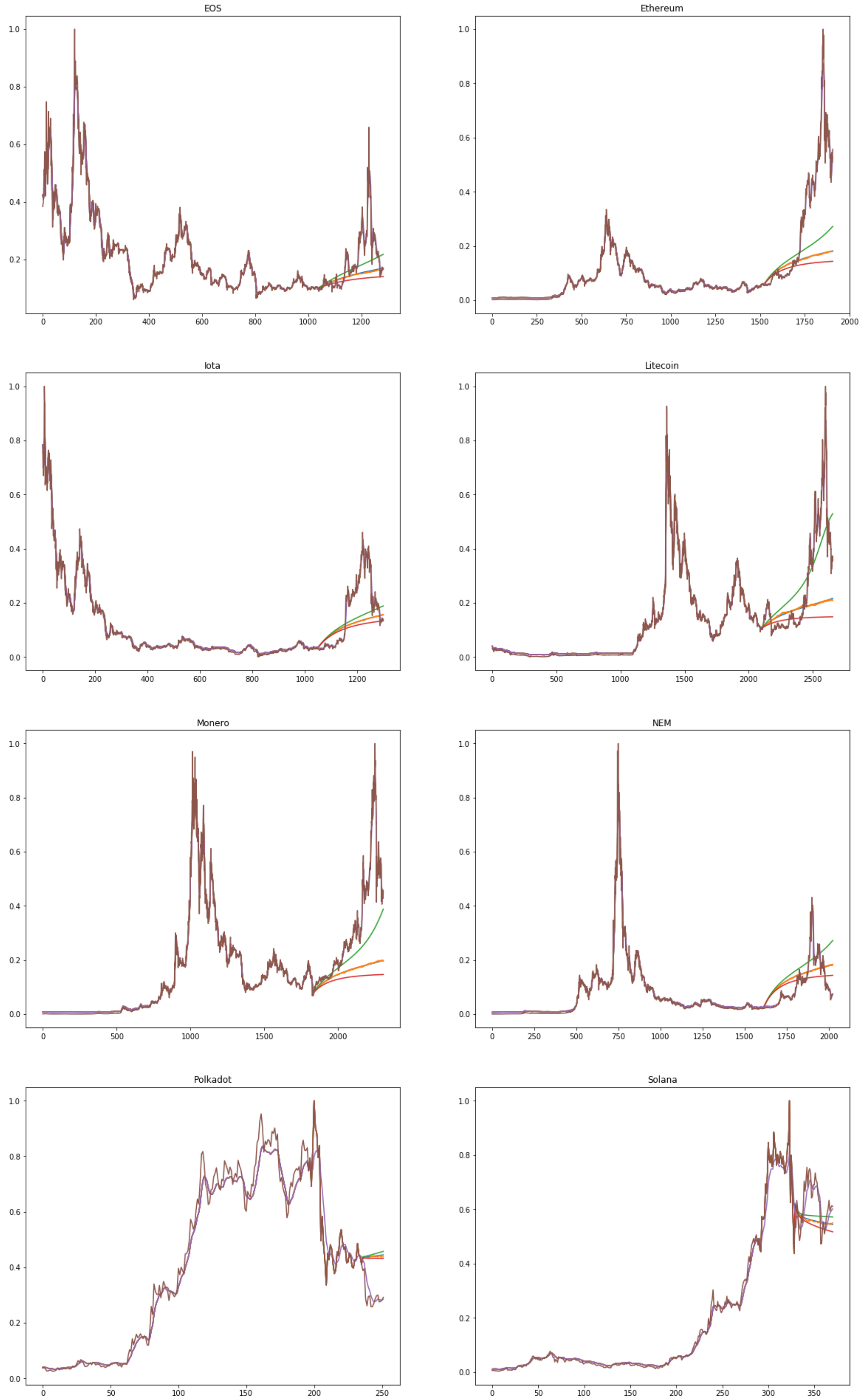
LSTM	0.833790123462677
Scaled LSTM	1.1338814496994019
GRU	0.961683988571167

Another experiment

Another question we can ask is; is it possible to introduce bias or randomness into the model to stop it from converging as easily, and does this improve performance of the model?

We will simply add a small value using random.random() to each timestep to create ‘chaos’. We can also try a couple of constant bias terms. We will test a couple few different values:


LSTM Model, "bias" added
Projected, no randomness
Projected*0.01*random(-1,1)
Projected + 0.001
Projected - 0.001
"Predicted" using LSTM
Real dataset
Aave
BinanceCoin
Bitcoin
Cardano
ChainLink
Cosmo
CryptocomCoin
Dogecoin
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
400
500
600
700
800
1000
1200
1500
2000
2500


EOS

Ethereum

Iota

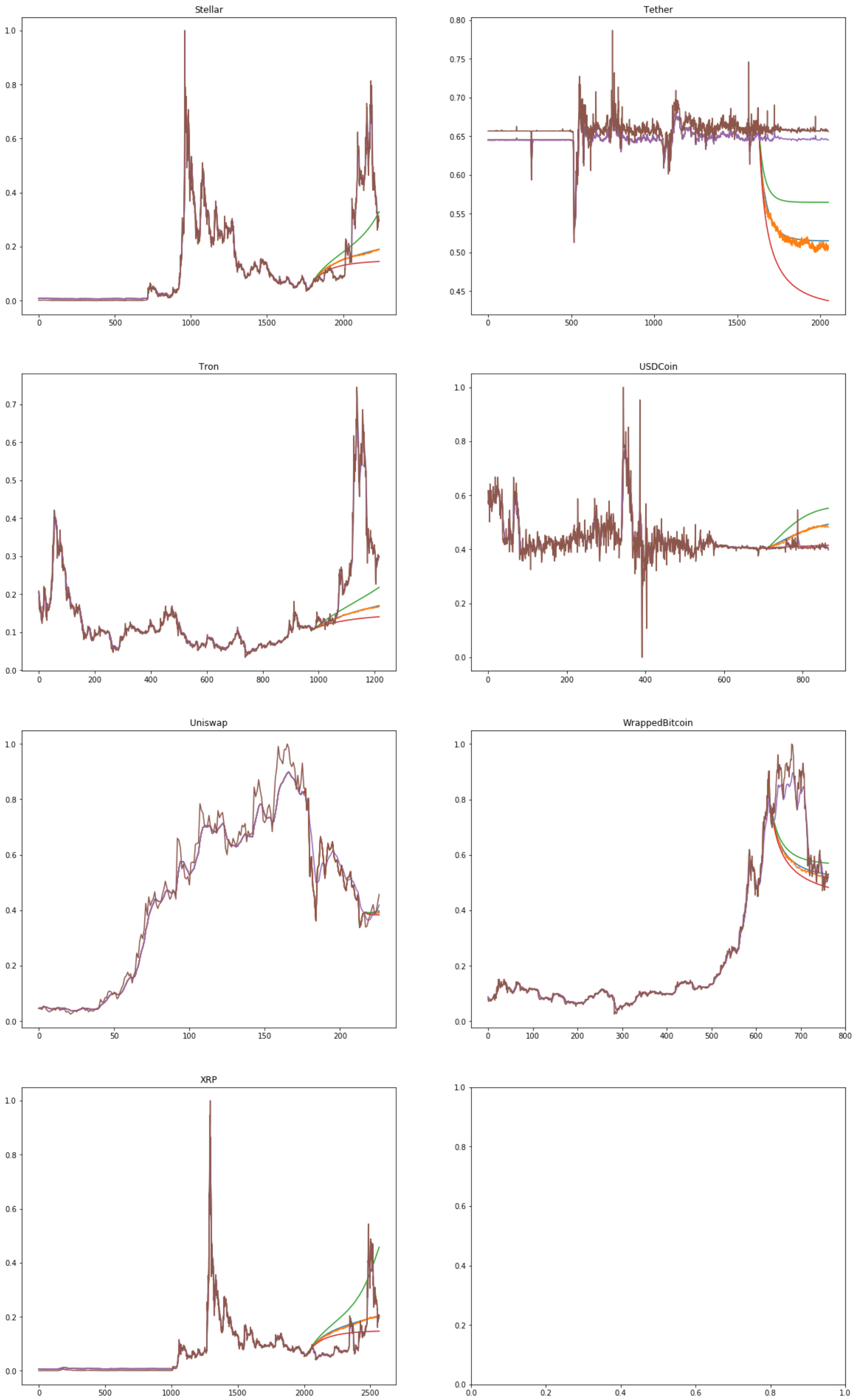
Litecoin

Monero

NEM

Polkadot

Solana


Stellar
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000
Tether
0.45
0.50
0.55
0.60
0.65
0.70
0.75
0.80
0
500
1000
1500
2000
Tron
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
200
400
600
800
1000
1200
USDCoin
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
Uniswap
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
WrappedBitcoin
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
400
500
600
700
800
XRP
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000
2500
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

MSE sum for base LSTM model with:

No randomness	0.833790123462677
Projected*0.01*random(-1,1)	0.8376890420913696
Projected + 0.005	0.6410854458808899
Projected - 0.005	1.0053616762161255

Discussion

If we look at the mean-squared error under training and validation we see that the lowest loss was achieved by the GRU model, however contrastingly, we see that the regular LSTM is better at predicting the future, evaluating by mean-squared-error (MSE) on the future projection. So even though the GRU appeared to perform better under our standard training conditions, upon long-term/distance predictions, the model doesn’t perform as well as the LSTM.

Perhaps the mismatch is due to the fact that we are trying to predict the future price many timesteps in advance, which is not what we do under training. This is an indication that perhaps the way in which our system/experiments are designed do not reflect the way in which we want to use our data, so this could be rectified with another design that takes all this into consideration.

Curiously, in our later experiments, adding a constant 'bias' term to the output of the model at every time step decreases the MSE of the projection and increases the accuracy of prediction, on average.

This could be due to the fact that cryptocurrencies generally tend to trend upwards, and the model doesn't capture this so well. Whatever the cause, it would be interesting to know if this is a statistical anomaly or if this represents some lack in the model itself.

Future Work

As per the original idea of this project, using text-based features could yield interesting results. This could first be tested by using ‘sentiment’ as a feature. This could also be extended by using some kind of embedding (with more features). Using BERT embeddings, perhaps with fine-tuning (on the type of texts/forums we intend to analyse) could provide a more detailed look at information about the cryptocurrencies prices which we are trying to predict.

To begin with we could scrape reddit (a scraper was built for this project but not yet used), to get daily text relating to the cryptocurrencies in question. We could incorporate the subreddits of the specific cryptocurrencies we wish to look at, but it would also be possible to eventually scrape other news sites, twitter and more.

Utilising a more comprehensive model for time-series prediction, perhaps something based on attention (and not an RNN) could yield interesting results.

Conclusions

The models we created performed to varying degrees of accuracy. A GRU achieved the lowest loss (mean-squared-error/MSE) under training and validation, however the LSTM achieved lower MSE when running a future prediction.

Introducing a bias term improved prediction performance, which could help suggest that new experiments need to be done.