Coin pair agnostic price prediction for cryptocurrency swaps on DEXes

Justyna Czestochowska **Paweł Młyniec**
EPFL, Lausanne, Switzerland
{justyna.czestochowska,pawel.mlyniec@epfl.ch}

Abstract

This work tackles the problem of price prediction for swap transactions of cryptocurrencies on Decentralized Exchanges (DEX). We study predictive abilities of two candidate machine learning models: Linear Regression and Recurrent Neural Network on the task of price prediction. We study six different cryptocurrencies pairs and examine models for ability to predict next transaction prices in one-step and autoregressive regime. Furthermore, we assess predictive abilities of Linear Regression model in coin pair agnostic setup, trying to create a versatile, one fits them all model. We conclude that price prediction is inherently difficult especially for long-term forecasting. Comparing to a Naïve Baseline, Linear Regression model managed to perform better on one-step prediction proving its short-term forecasting abilities.

1 Introduction

Since bitcoin's first release as an open-source software in 2009 the market of cryptocurrencies has flourished and expanded. Currently there exist over 8000 different cryptocurrencies (Cap, 2021). As these digital assets gathered a lot of attention a necessity emerged to create an agile environment where sellers and buyers could meet and arrange a trade. Following a classical approach to finance, centralized exchanged such as Coinbase[1] or Binance[2] were created to offer a space for cryptocurrencies' transactions. These platforms offer services such as: crypto trading services, lending, borrowing, margin trading, cross chain exchange for multiple cryptocurrencies, and conversion of fiat currency to cryptocurrency and vice-versa. In terms of exchanging cryptocurrencies the central exchanges follow a classical approach of order book to match sellers and buyers of a given coin pair to be exchanged.

However convenient and easy to design the centralized setup may be, it is still vulnerable to data breaches and security leaks violating the main idea of cryptocurrencies - privacy (Cen, 2021). Excluding a third-party creates its own challenges which are constantly being solved by newer and newer versions of platforms for decentralized exchange, *DEX*. DEXes rely on smart contracts and solutions such as AMM - *automated market maker* to conduct buyer and seller matching in a decentralized fashion. Thanks to these two elements a user can be sure that the transaction will never fail and an exchange request will be successfully executed (Cen, 2021).

As cryptocurrencies' prices experience sudden ups and downs (Haar, 2022), price prediction became a central interest to many companies and research institutions (Ji et al., 2019; Chen et al., 2020; Phaladisailoed and Numnonda, 2018; Smith, 2021). The aim of this work is to explore the possibility of price prediction on decentralized exchanges with use of machine learning methods. As cryptocurrencies' prices are characterized by a high volatility (Anonymous, 2018), an accurate, even one-step prediction, i.e a one that predicts the next price only, could provide useful for e.g. fast-frequency trading purposes. Also it can be used for dynamic coin swapping. Swapping is a similar process but with more flexibility. It allows to acquire the desired coin instantly.

In this work we:

- Train two machine learning models: a linear regression and a recurrent neural network for price prediction task
- Evaluate both models in one-step prediction and autoregressive scenario, i.e we compare how models perform when predicting only

[1]https://www.coinbase.com/
[2]https://www.binance.com/

the price of the next transaction to how they perform while predicting multiple next transactions taking its own previous prediction as an input.

- We compare both models against a Naïve Baseline which predicts the next price to always be the last one that was provided to the model as input.
- We perform this in coin pair agnostic and a coin pair specific scenario. In the first setup we train each model for all six available cryptocurrency pairs in our dataset while in the second setup we train one of each models per pair of cryptocurrencies.
- We additionally explore providing Binance centralized prices as a predictive feature of the next transaction price.

2 Data

2.1 SushiSwap

We use a dataset of transactions from popular decentralized exchange (DEX) SushiSwap[3]. The data consists of six comma separated files containing transactions records spanning different periods between September 2020 and September 2021. Each file contains swapping transactions between wrapped Ethereum (WETH) and another cryptocurrency, namely: Axie Infinity Shards (AXS), Convex Finance (CVX), Universal Market Access (UMA), Tether (USDT), 0x (ZRX), and Yearn Finance (YFI).

All files follow the same schema containing a timestamp - exact date and time of a transaction, isBid - a boolean (true or false) true when buy was initiated and false if sell was initiated, price - exchange rate for a pair, and the volume - the amount exchanged.

timestamp	isBid	price	volume
2021-05-17 10:27:19	True	0.000075	0.05
2021-05-17 10:31:27	True	0.000084	0.07
2021-05-17 10:34:51	True	0.000098	0.10
2021-05-17 10:36:55	True	0.000115	0.10
2021-05-17 13:02:58	True	0.000129	0.05

Table 1: First five rows of the CVX_WETH.csv file which contains record of transactions between wrapped Ethereum and Convex Finance. All other files follow the same data format.

[3]https://sushi.com/

Prices in SushiSwap dataset exhibit different ranges of values. Detailed description of price statistics is presented in Table 3.

2.2 Binance

To explore potential predictive power of incorporating prices from Centralized Exchanges we benefit from open access data from Binance[4].

In order to match Binance inputs with SushiSwap trades we need to align the timestamps. As SushiSwap is our main data source we use only the Binance data that is aligned with timestamps from SushiSwap. This was the case only for one pair of cryptocurrencies, namely Ethereum (WETH) and Tether (USDT).

However, as these timestamps are not exactly aligned we perform an hourly aggregation on Binance prices and merge these with the SushiSwap prices.

timestamp	price_sushi	price_binance
2020-09-09 19:56:56	356.81	357.10
2020-09-09 19:58:56	356.73	357.10
2020-09-09 20:04:29	356.70	354.88
2020-09-09 20:06:22	356.50	354.88
2020-09-09 20:06:22	356.62	354.88

Table 2: First five rows of the CVX_WETH_with_binance.csv file which contains exchange prices between wrapped Ethereum and Tether from both SushiSwap and Binance. Binance prices were aggregated per hour: note the change of Binance price as transactions start after 8pm.

3 Methods

For the task of price prediction we tested two machine learning models in two different evaluation regimes. Machine learning models are a class of methods that use a subset of available data referred to as *training set* to tune - *train*, parameters of an adaptive model (Jordan et al.).

In our case a model is trained using historical prices of a cryptocurrency pair. Once the model is trained it can then determine the next transactions' prices, which are said to comprise a *test set*. The ability to categorize correctly new examples that differ from those used for training is known as *generalization* (Jordan et al.).

One of our candidate models is a Recurrent Neural Network model which falls into a narrower category of machine learning models, namely deep

[4]https://data.binance.vision/?prefix=data/spot/daily/trades

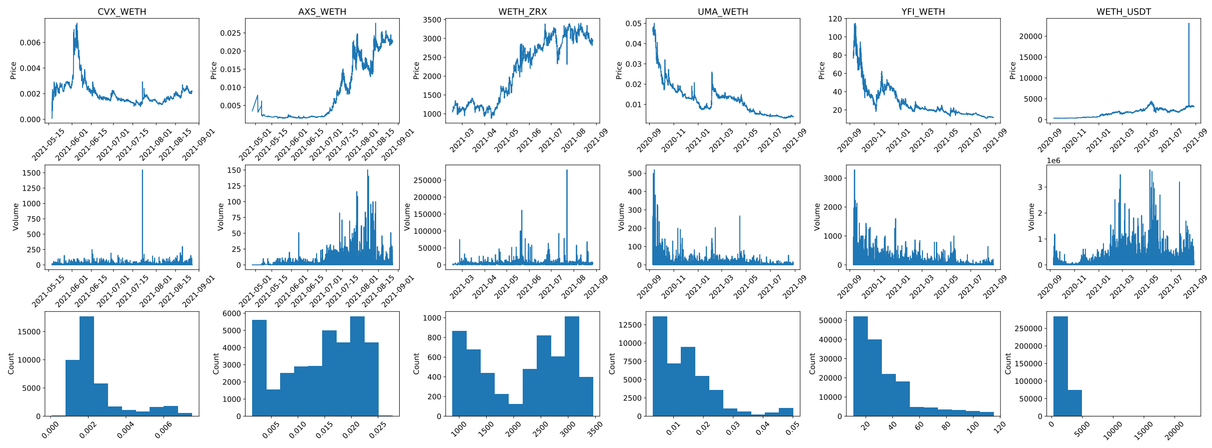

Figure 1: Visualization of buy (isBid true) time-series for price and volume. Each file is one column of the panel. The top row contains price, the middle row volume, and the bottom row is a histogram of time-series values.

pair	μ	σ	minimum	25%	50%	75%	maximum
AXS_WETH	0.0141	0.0071	0.0014	0.0087	0.0154	0.0206	0.0277
CVX_WETH	0.0025	0.0014	0.0001	0.0016	0.0019	0.0026	0.0075
UMA_WETH	0.0140	0.0097	0.0031	0.0059	0.0128	0.0184	0.0501
WETH_USDT	1845.2122	936.4094	316.7426	1118.5669	1898.3541	2489.0592	23116.9188
WETH_ZRX	2188.8198	799.5341	880.1226	1294.8318	2425.7944	2936.9823	3451.8028
YFI_WETH	34.0806	21.4919	10.7351	18.9538	26.5854	42.3862	115.0038

Table 3: Descriptive statistics of exchange prices in SushiSwap data for each pair of cryptocurrencies.

learning. Deep Learning constitutes a class of models that work as without any additional feature extraction. These methods benefit from large datasets and are especially successful in computer vision applications (Lecun et al., 1998; Redmon et al., 2016; He et al., 2015). As in total our dataset consists of 636,879 data points deep learning methods seem appropriate.

3.1 Models

The task of price prediction falls into category of time-series forecasting. Usually for such a task a window of previous time-series values is chosen as an input to the model to determine the next value in the series. We applied this approach using different window size for each model. A window size is a hyper-parameter of our model, window sizes for all models were obtained using a procedure known as *hyper-parameter tuning*.

3.1.1 Naïve Baseline

To gain a better understanding of our models' performance we compare errors against error achieved by a Naïve Baseline model. Given a window of size D of historical prices for a given coin pair, the baseline model always outputs x_{D-1}, i.e the last value from a provided window, Eq. 1. We conclude that if our models are able to outperform the Naïve Baseline then they posses certain predictive power.

$$y(\mathbf{x}) = x_{D-1} \tag{1}$$

3.1.2 Linear Regression

Linear regression is one of the most basic approaches in statistical modeling. The simplest linear model for regression is one that involves a linear combination of the input variables (Jordan et al.):

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_D x_D. \tag{2}$$

Where y is the output of the model - price of the next transaction which depends on $\mathbf{w}$ and $\mathbf{x}$. $\mathbf{w}$ are the parameters of the model and $\mathbf{x}$ are the inputs. For the task of price prediction $x_1, x_2, \ldots, x_D$ are the consecutive historical prices for a given coin pair. The full equation of our linear model is formulated in equation 3.

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{i=1}^{D} w_i x_i \tag{3}$$

Window size D is determined by hyper-parameter tuning Sec. 3.2. For the training procedure we used a method in a popular Python package scikit-learn which fits the model using Ordinary Least Squares objective, minimizing the residual sum of squares between the price targets, and the prices predicted by the regression model (Pedregosa et al., 2011).

Since coin agnostic model takes as a training set time-series for all six cryptocurrency pairs which have different price magnitudes and ranges, it is necessary to rescale the data to a common range. As the objective is distance based (mean squared error), without rescaling a small mistake on WETH_ZRX of, for example 5 units, would be a very large mistake for CVX_WETH where price magnitudes are of 10^{-3}. For the coin pair agnostic models we used min-max scaling on the training set which is presented in Eq. 4.

$$\tilde{x} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4}$$

The minimum and maximum was computed using all the values from the training time-series of all the six time-series available. It is important to find the scaling values (x_{min}, x_{max}) only on the training set to fairly mimic the inference situation. In the testing phase model has no access to the future values and therefore should not be scaled using the testing set. Such scaling would not provide a fair evaluation - assessment of its generalization abilities - of the model. During testing we scale the input to the model with scaling factors computed on the training set.

3.1.3 Recurrent Neural Network - LSTM

Recurrent neural networks (RNNs), form a family of neural networks for processing sequential data such as time-series or natural language. A recurrent neural network is a neural network that is specialized for processing a sequence of values $x_1, ..., x_D$. (Goodfellow et al., 2016). Even though such networks can process sequences of variable lengths, we also use a fixed size window D as in Linear Regression model.

Recurrent neural networks (RNNs) are extensively used to model sequential data and are recently often applied for financial time-series processing (Sezer et al., 2020). One of the most popular RNN architectures are Long Short-Term Memory (LSTM) (Hochreiter and Urgen Schmidhuber, 1997) cells. The function of each LSTM layer is described by the following equations Eq. 5

$$\begin{aligned}
i_t &= \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right) \\
f_t &= \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\right) \\
g_t &= \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}\right) \\
o_t &= \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}\right) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh\left(c_t\right)
\end{aligned} \tag{5}$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, $h_{(t-1)}$ is the hidden state of the layer at time t-1 or the initial hidden state at time 0, and i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function, and $\odot$ is the Hadamard product (PyTorch).

Main difference of LSTM over vanilla RNN is having gate-based architecture. Thanks to that it faces two main RNN problems, which are exploding gradient and vanishing gradient.

An error gradient is the direction and magnitude calculated during the training of a neural network that is used to update the network weights in the right direction and by the right amount.

In deep networks or recurrent neural networks, error gradients can accumulate during an update and result in very large gradients. These in turn result in large updates to the network weights, and in turn, an unstable network. At an extreme, the values of weights can become so large as to overflow and result in NaN values.

The explosion occurs through exponential growth by repeatedly multiplying gradients through the network layers that have values larger than 1.0. (Brownlee)

Vanishing gradient on the other hand, occurs when error in long sequences is propagated with smaller and smaller values. When it is close to zero model stops learning and is stuck in one place without ant improvements.

This problems were solved by using previously mentioned gates. They regulate how much information is passed through the network and therefore how large error gradient is passed back. Also through control in gates how much information from the previous input is passed to actual output it decides whether information from the past in sequence should be replaced by new which is entering the model or should be preserved. This feature allows LSTM network to remember information from many previous inputs, better than vanilla RNN.

pair	window size
AXS_WETH	9
CVX_WETH	1
UMA_WETH	10
WETH_USDT	1
WETH_USDT_with_binance	3
WETH_ZRX	1
YFI_WETH	1
agnostic	2

Table 4: Values of optimized window sizes for Linear Regression model for different cryptocurrencies pairs.

pair	window size
AXS_WETH	9
CVX_WETH	7
UMA_WETH	8
WETH_USDT	-
WETH_ZRX	4
YFI_WETH	3
agnostic	2

Table 5: Values of optimized window sizes for RNN model for different cryptocurrencies pairs. We failed to achieve results for WETH_USDT

In our work we apply LSTM model with 4 hidden layers, 128 neurons, *Adam* (Kingma and Ba, 2017) optimizer and Mean Squared Error as a loss. To build the model we used a popular deep learning Python library Keras (Chollet et al., 2015).

We choose *Adam* – an adaptive learning rate optimizer over a basic Stochastic Gradient Descent, as it is a standard weapon of choice in deep learning community since its appearance in 2015.

Optimizer defines how error gradient will be applied to neural network weight, in other words how learning will be applied to the model. According to Kingma and Ba, the method is ”computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters”. It's also adaptive method, which during training changes how much weights are updated as learning unfolds.

3.2 Hyper-parameter tuning

Hyper-parameter tuning is a procedure which involves evaluating multiple values of a hyperparamter - in our case window size - so that the evaluation metric is optimized. For the parameter tuning of window size for Linear Regression and Naïve Baseline we perform a simple grid search. We cross-validate both models in coin pair specific and coin pair agnostic scenarios. In each scenario we check possible values of window size $D \in \{1, 2, 3, ..., 10\}$.

Afterwards, for each model we choose a window size that minimizes the cross-validation Mean Absolute Error Eq. 11. Tuned window sizes are presented in Table 4. Interestingly, for half of the models the best window size was 1, only the last price counts for model's prediction.

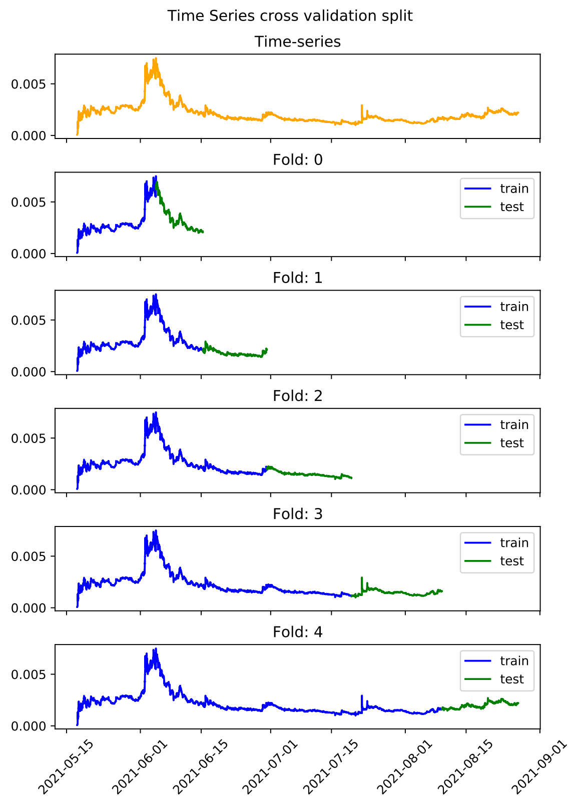

Figure 2: Visualization of forward chaining method for train-test split of time series. The top plot in the first row shows the time-series in question. Subsequent plots represent different train-test splits for each cross-validation fold. Blue part of the time-series is a training set and the green time-series is the testing set for a given fold. Note that unlike standard cross-validation methods, successive training sets are supersets of those that come before them (Pedregosa et al., 2011).

3.3 Evaluation

To evaluate both models we used a 5 fold cross-validation. As our data includes time-series we followed a forward chaining method illustrated in figure 2. We compared all models in two regimes: one-step prediction and autoregressive. Every model was trained for the task of one-step prediction only.

One-step prediction assumed that each value in the testing part of the time-series is one data point of the testing set and the D previous values are the input, i.e the dataset consisted of pairs:

$$\begin{aligned}&\{((x_0, x_1, x_2, x_3, x_4, x_5, x_6), y_0 \equiv x_7),\\&((x_1, x_2, x_3, x_4, x_5, x_6, x_7), y_1 \equiv x_8), \dots,\\&((x_{n-6}, x_{n-5}, x_{n-4}, x_{n-3},\\&x_{n-2}, x_{n-1}, x_n), y_{n-D} \equiv x_n)\}\end{aligned} \tag{6}$$

Where $x_i, ..., x_{i+D-1}$ were the price values in a given input window and y_i was the target price value for the window. This means that consecutive target values become inputs in next windows. That dataset construction allowed for one-step prediction regime where we were interested in error between the predicted price of the next transaction - $\hat{y}$ and the real target value that appeared in the time-series. The aim of this evaluation is to assess short-term prediction capabilities of the models.

For the autoregressive regime we compare the error between the whole series predicted for the testing part of the time-series and the actual target values in the testing part. Autoregressive regime means that in fact we only use the first D values of the testing part of the time-series as input to the model. The rest of the testing part of the time-series is only used to compare to the model output unlike in the one-step regime where the whole testing part of the time-series was used as input as separate data points.

In autoregressive regime model outputs serve as inputs in the next predictions. The aim of this evaluation was to verify whether this model could be used for long-term forecasting. The consecutive steps for autoregressive regime are shown in equations 7, 8, 9, 10. In this prediction the previous output of the model was treated as an input in the next step.

$$\hat{y_0} = w_0 + \sum_{i=1}^{D} w_i x_i \tag{7}$$

$$\hat{y_1} = w_0 + \sum_{i=1}^{D-1} w_i x_i + w_D \hat{y_0} \tag{8}$$

$$\hat{y_2} = w_0 + \sum_{i=1}^{D-2} w_i x_i + w_{D-1} \hat{y_0} + w_D \hat{y_1} \tag{9}$$

$$\hat{y_n} = w_0 + \sum_{i=1}^{D} w_i \hat{y_i} \tag{10}$$

To be able to compare coin pair specific and coin pair agnostic models we needed a to take into account that coin agnostic model was scaled to range between 0 and 1. This scaling was performed separately for each fold during the cross-validation. Hence for the final comparison between the coin pair specific and coin agnostic we reversed the scaling for the agnostic model to be able to compare the error between these models. Had the output of the coin agnostic model been not rescaled back we would look at errors of different magnitudes.

For our evaluation metric we used mean absolute error defined in equation 11. MAE is a convenient metric for regeression models that is more robust to outliers than mean squared error.

$$MAE = \frac{\sum_{i=0}^{n-1} |y_i - \hat{y_i}|}{n} \tag{11}$$

4 Results

4.1 Coin pair specific

Figure 3 presents comparison of models in autoregressive evaluation regime. Autoregressive task proved difficult to be well generalized. Linear regression outperformed the baseline model only for AXS_WETH and UMA_WETH exchange. The Naïve Baseline proved superior in most scenarios. LSTM network is the worst on all scenarios only in CVX_WETH being comparable to other models. Our linear regression was ineffective for long-term forecasting purposes. It did not perform better than a model that always predicts the same value.

Figure 4 presents comparison of models in one-step evaluation regime. Short-term forecasting turned out more feasible for the linear regression model. Again other models outperformed LSTM in all cases. In coin pair specific models linear regression utperformed the Naïve Baseline for AXS_WETH and UMA_WETH pairs and achieved comparable results for other pairs.

Using Binance data for WETH_USDT pair we achieved slightly worse results for one-step regime and much better results for autoregressive regime, the detailed comparison is presented in Table 6.It seems like incorporating centralized exchange prices was beneficial for the long-term forecasting abilities, however as we can only report it for one cryptocurrency pair this result may be coincidental.

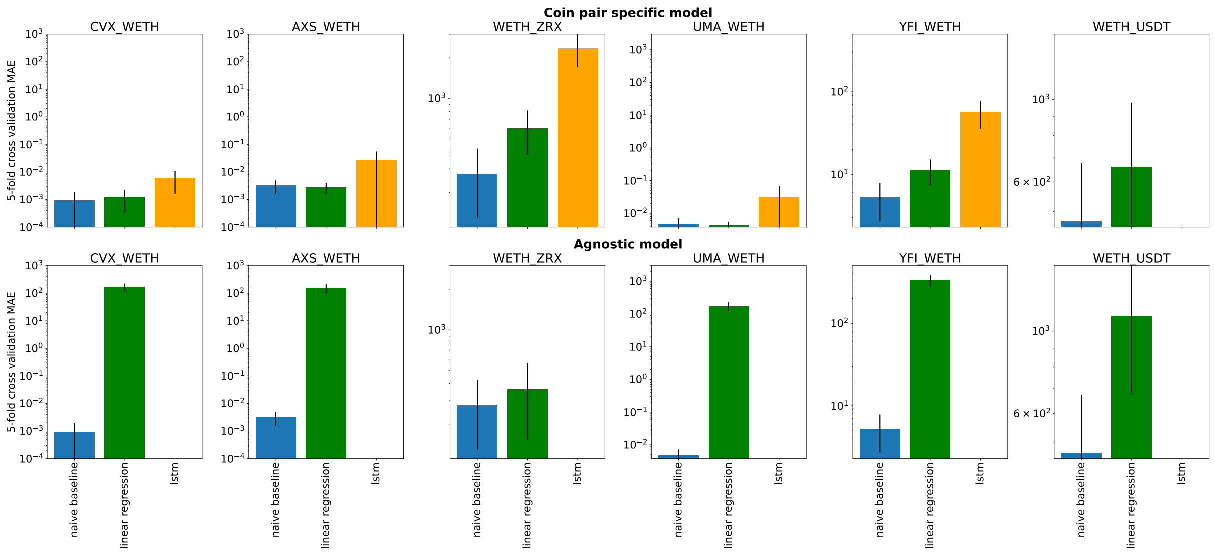

Figure 3: Cross-validation MAE in autoregressive evaluation for baseline model and linear regression model. Bars represent mean across folds and error bars are standard deviations across folds. Results should be compared columnwise per pair of cryptocurrencies. Note different scale for each column. Lower is better. Empty bars are models for which we did not manage to obtain results.

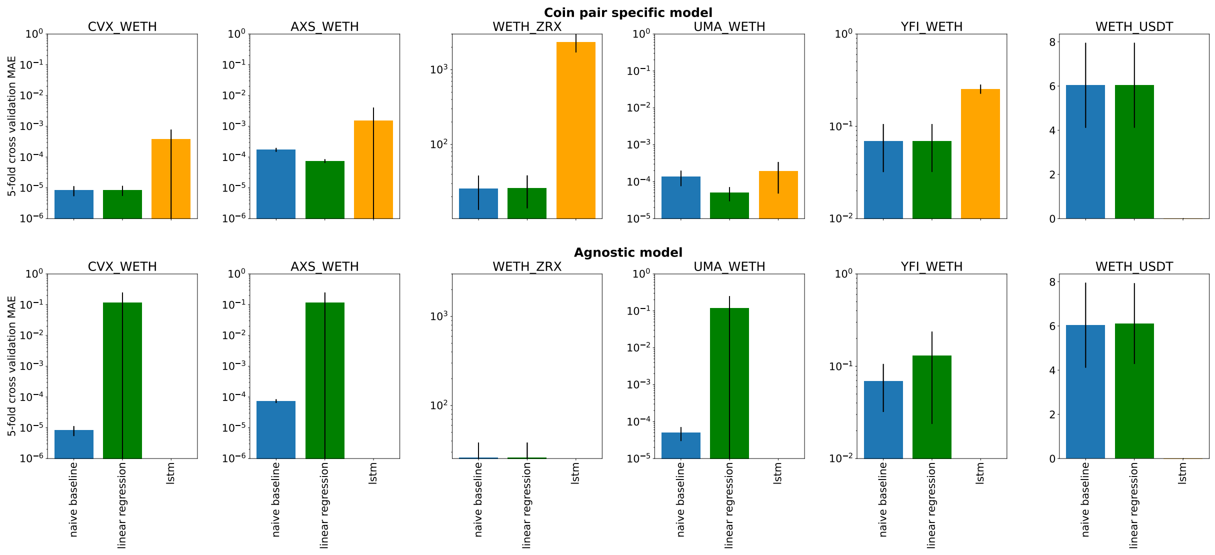

Figure 4: Cross-validation MAE in one-step evaluation for baseline model and linear regression model. Bars represent mean across folds and error bars are standard deviations across folds. Results should be compared columnwise per pair of cryptocurrencies. Note different scale for each column. Lower is better. Empty bars are models for which we did not manage to obtain results.

Overall, short-term forecasting seems already possible with a very simple model such as linear regression and is vastly time and resource consuming for recurrent neural networks.

In Figures 5, 6, 7, 8, 9 we present autoregressive results on the last fold for five out of six studied cryptocurrency pairs. Green solid line shows predicted values and blue solid line shows actual time-series values. We can visually inspect the fit of the two candidate models. For all shown pairs we

pair	regime	MAE
WETH_USDT	autoregressive	659.15
WETH_USDT_with_binance	autoregressive	11.61
WETH_USDT	one-step	6.04
WETH_USDT_with_binance	one-step	6.53

Table 6: Cross-validation MAE for both evaluation regimes for Ethereum and Tether with and without Binance prices as additional feature.

can see that Linear Regression outperforms LSTM model. Interestingly, for WETH_ZRX pair LSTM model failed even on the training part of the time-series, which is also reflected in the highest error shown in Figure 3.

4.2 Coin pair agnostic

In autoregressive evaluation regime the error for agnostic model is always at least a magnitude higher than in coin pair specific models for all studied pairs of cryptocurrencies. That shows that coin agnostic model lacks the generalization ability for long-term forecasting. In one-step regime the coin agnostic model failed as well for all studied pairs of cryptocurrencies. LSTM network failed to accomplish training due to lack of the resources on our machines and in colab service.

For now coin agnostic modeling did not prove successful. Naïve baseline is better for both regimes and for all coin pairs. The LSTM network although being the most advanced model took so many resources that it was extremely hard to obtain results with standard computational resources.

One of the possible reasons for the discrepancy of the results between the Naïve Baseline and Linear Regression is the scaling method we used to construct the coin agnostic dataset.

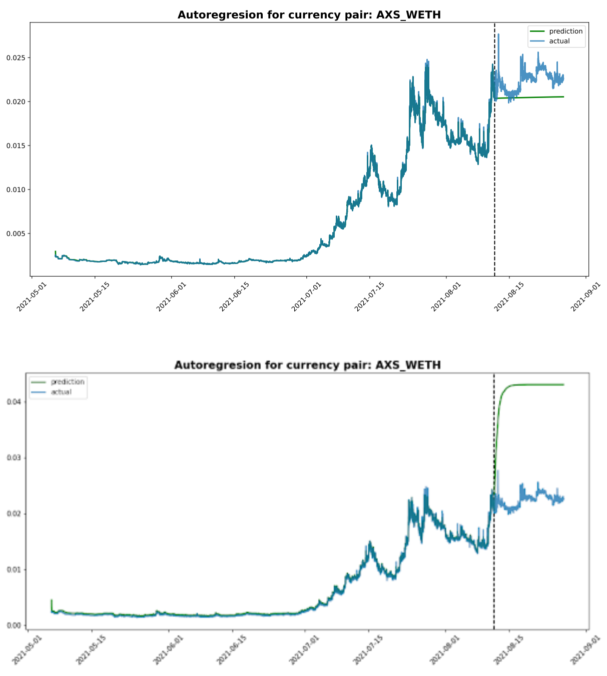

Figure 5: Prediction on the last fold of Axie Infity Shard with Ethereum from tuned models for coin pair specific setup. Top: Linear Regression, bottom: LSTM. Note different y-axis range between top and bottom.

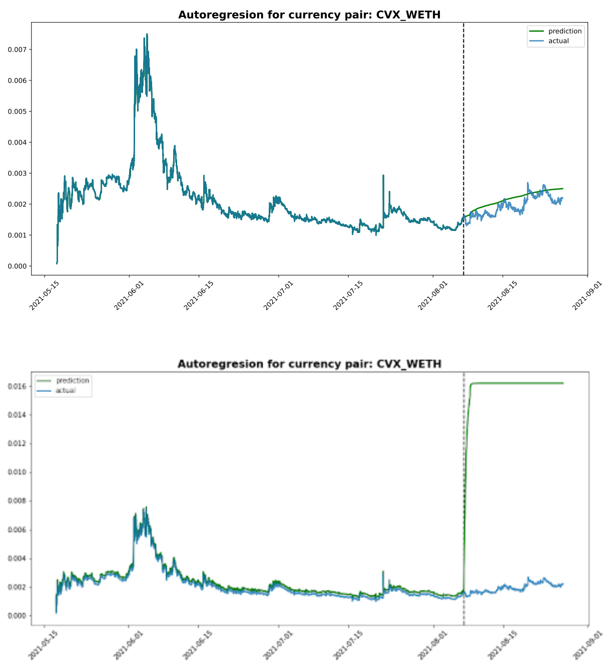

Figure 6: Prediction on the last fold of Convex Finance with Ethereum from tuned models for coin pair specific setup. Top: Linear Regression, bottom: LSTM. Note different y-axis range between top and bottom.

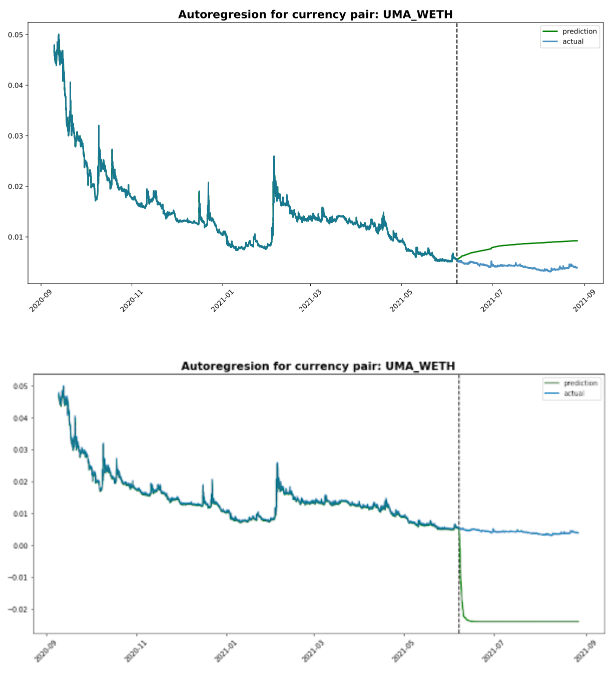

Figure 7: Prediction on the last fold of Universal Market Access with Ethereum from tuned models for coin pair specific setup. Top: Linear Regression, bottom: LSTM. Note different y-axis range between top and bottom.

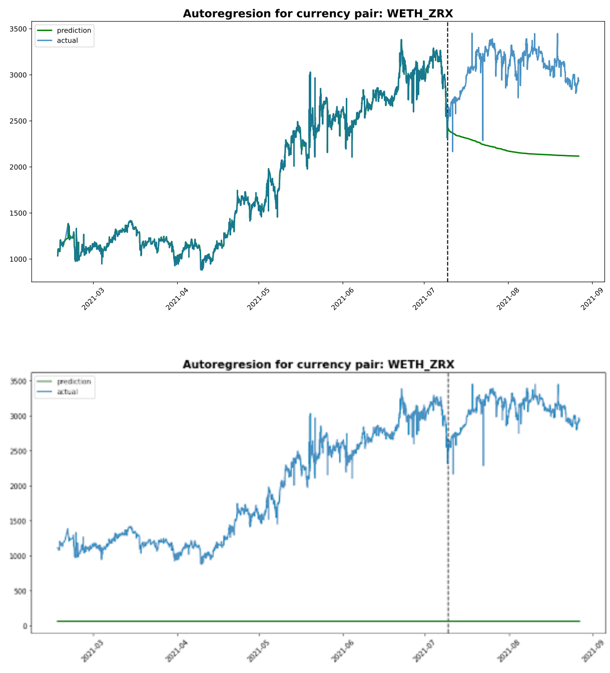

Figure 8: Prediction on the last fold of Ethereum with 0x from tuned models for coin pair specific setup. Top: Linear Regression, bottom: LSTM. Note different y-axis range between top and bottom.

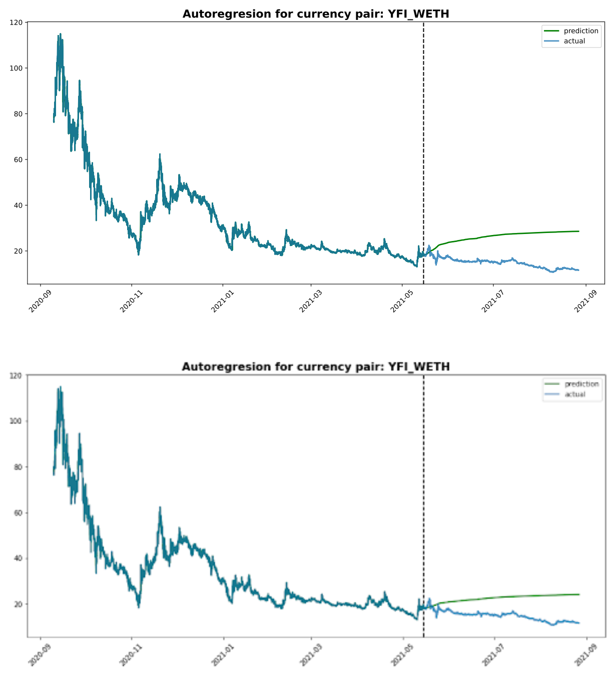

Figure 9: Prediction on the last fold of Yearn Finance with Ethereum from tuned models for coin pair specific setup. Top: Linear Regression, bottom: LSTM. Note different y-axis range between top and bottom.

5 Discussion

Even though our models did not prove effective for many scenarios, Linear Regression managed to outperform the Naive Baseline model in one-step prediction evaluation for few studied cryptocurrency pairs.

Accurate one-step prediction could be useful for fast frequency trading purposes and automatic traders. Given the simplicity of our setup this roughly suggests an untapped potential in machine learning models to create such short-term forecasting systems.

Even though long-term forecasting is a holy-grail in many applications the autoregressive evaluation proved how challenging this task is.

Additionally, creating a coir pair agnostic model turned out futile. We did not manage to build it for Reccurent Neural Network model (hyperparameter-tuning turned out to be both time and resources consuming). Furthermore, Linear Regression model failed in comparison with the Baseline model. This most likely happened because of data normalization technique that we used. To improve the modelling, more careful data normalization should be applied and deeper analysis of time-series properties. Finally, one size fits them all may not at all be possible in a given price prediction task.

With this work we confirm how challenging is the task of price prediction. Even though high expectations lie in deep learning based methods, those require a lot of resources and careful hyperparameter tuning to work properly. Our Recurrent Neural Network based on Long-Short Term Memory Cells did not outperform a very simple Linear Regression model.

We limited ourselves to price prediction based on previous prices only. One could incorporate other information such as transaction volume to enhance the predictive abilities. Moreover, no feature engineering was performed in this work. Features such as: moving average, exponential average, opening or closing prices per hour should be explored in subsequent work on the topic.

Interestingly, incorporating prices from centralized exchange (Binance) improved results of Linear Regression in autoregressive setup. This result may be incidental as we only managed to construct a dataset for one cryptocurrency pair, however it would be interesting to explore this property further in a rigorous experimental setup.

References

2021. Centralized Finance (CeFi) vs Decentralized Finance (DeFi) — The Battlefield of Cryptocurrencies — by Blockchain Simplified — Medium.

Anonymous. 2018. Institutional Investors Will Bet Big on Cryptocurrencies in 2018.

Jason Brownlee. A Gentle Introduction to Exploding Gradients in Neural Networks.

Coin Market Cap. 2021. Cryptocurrency Prices, Charts And Market Capitalizations — CoinMarketCap.

Zheshi Chen, Chunhong Li, and Wenjun Sun. 2020. Bitcoin price prediction using machine learning: An approach to sample dimension engineering. *Journal of Computational and Applied Mathematics*, 365:112395.

Francois Chollet et al. 2015. Keras.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Ryan Haar. 2022. Bitcoin Hit Another New All-Time High Last Month. Crypto Investors Should Ignore the Ups and Downs.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Sepp Hochreiter and J J Urgen Schmidhuber. 1997. LSTMOriginal. Technical Report 8.

Suhwan Ji, Jongmin Kim, and Hyeonseung Im. 2019. A Comparative Study of Bitcoin Price Prediction Using Deep Learning. *Mathematics 2019, Vol. 7, Page 898*, 7(10):898.

M Jordan, J Kleinberg, and B Schölkopf. *Pattern Recognition and Machine Learning*.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Fabian Pedregosa, Vincent Michel, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Jake Vanderplas, David Cournapeau, Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Bertrand Thirion, Olivier Grisel, Vincent Dubourg, Alexandre Passos, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830.

Thearasak Phaladisailoed and Thanisa Numnonda. 2018. Machine learning models comparison for bitcoin price prediction. *Proceedings of 2018 10th International Conference on Information Technology and Electrical Engineering: Smart Technology for Better Society, ICITEE 2018*, pages 506–511.

PyTorch. torch.nn — PyTorch master documentation.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection.

Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. 2020. Financial time series forecasting with deep learning : A systematic literature review: 2005–2019. *Applied Soft Computing*, 90:106181.

Sean Stein Smith. 2021. Crypto And Blockchain Predictions For 2022.