Predicting digital asset value based on blockchain activity data

Author: Zvezdin Besarabov
National school of mathematics and natural sciences
me@zvezd.in

Mentor: Todor Kolev
Founder of the Obecto Cooperative
todor@obecto.com

November 18, 2017

Abstract

The crypto-asset markets are public, decentralized and transparent by definition. In our research, we explore the new kind of data available on the Ethereum blockchain in efforts to create an accurate prediction strategy using deep neural networks. We gather our data from multiple sources and extract the most important data points (properties) from it. Other than basic properties, we explore different methods to create complete network state snapshots by generating multiple values that are specifically ordered in time and space. We have proposed multiple methods to model a dataset by aligning the property values differently in multidimensional matrices. We have built neural network architectures that are based on basic deep, Long Short-Term Memory (LSTM) and Convolutional networks. We ran different experiments with each one of them and with different datasets. Because the convolutional network is able to extract meaning from spatially related data, it performed far superior than any other model we have tried. Due to its high volatility, Ethereum has proven difficult to predict. Despite that, our results have shown a strong correlation between blockchain and price data. We should focus most of our efforts in perfecting the quality of our dataset, before tweaking the network architecture. We have found that increasing the spatial value in our dataset also increases the prediction accuracy.

For our experiment, we have created and documented a flexible framework that allows the prediction of any cryptocurrency on the base of any data.

Contents

1 Introduction

Crypto-currencies, the decentralized new ways of exchanging value, are gaining a lot of interest recently with the emergence of Bitcoin, Ethereum and hundreds other crypto-assets totalling more than $100 billion in value. Unlike the traditional markets, where all the data about the market and the trading itself are centralized and controlled by gatekeeper organizations, the crypto-asset markets are public, decentralized and transparent by definition. This gives a unique opportunity for trading algorithms to provide predictions based on this new kind of data that is available on the blockchain.

Most of the predictions on certain assets' values are done solely based on previous price points. This data, while it can indicate how the value might change, does not represent the reason for one. And as the base principal of trading goes - simply because it has been, does not mean that it will be. With the rise of the blockchain, numerous new opportunities come to develop more efficient prediction algorithms. We can analyse the public blockchain data in an attempt to find a correlation between it and the value change, possibly predicting the future value. We can take the data, process it and use it to train Deep Neural Networks to predict the market.

The blockchain we chose for this experiment is Ethereum, due to its unique feature of being not just a currency, but also a global decentralized supercomputer for the creation and execution of decentralized applications. Despite that, we have put efforts to make our research as flexible as possible, and thus to be reproducible in different currencies.

Projects with similar end goals to ours have emerged in the past. A Stanford paper [ISA14] on the topic of Bitcoin predictions has been published in 2014. Despite being very limited, their dataset extracts some features from the Bitcoin blockchain. They only got 55% hourly prediction sign accuracy and did not publish the needed details to reproduce their results. Two more recent projects on Github also explore Bitcoin predictions [Byn15] [Ré17], However, both of them focus primarily on historical price data and did not reveal their data processing algorithms and neural architectures.

1.1 Introduction to Blockchain and Ethereum

Satoshi Nakamoto's introduction of Bitcoin in November 2008 has often been hailed as a radical development in money and currency as it's the first example of a digital asset which simultaneously has no backing or "intrinsic value" and no centralized issuer or controller. However, another arguably more important part of the Bitcoin experiment is the underlying blockchain technology as a tool of distributed consensus. The most important aspect of such technology is the absence of an intermediary (centralized server, bank, company, etc.) between the originator and the recipient, as any changes to the data in this chain are made by consensus among all members of a decentralized network. Thus, avoiding the need to trust third parties.

The blockchain platforms are already revolutionizing the technological development advancements, having a wide range of applications besides finance. International laboratories for research in this direction are currently being established. Entrepreneurship worldwide focuses on the blockchain technology, developing systems for supply chain tracking, payments, privacy, consensus between members, evaluation of assets and others.

Ethereum (ETH) is a newer cryptocurrency [Eth17a], based on the blockchain technology. The blockchain can be thought of a distributed public database with records for each transaction in history. The data structure of the blockchain is a chain of blocks, each block being a package of transactions (Please refer to figure 2). Newer blocks on the chain are added as newer transactions are executed.

1.2 Autonomous decentralized applications

The intent of Ethereum is to create an alternative protocol for building autonomous decentralized applications. The applications are well suited for situations where rapid development time, security

Figure 1: The value of one Ether based on the US dollar since the creation of Ethereum

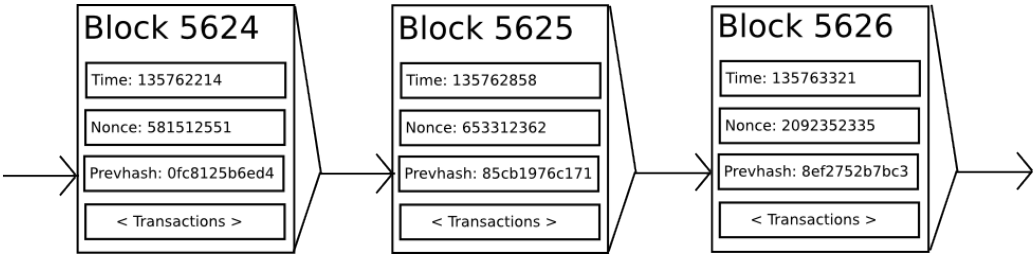

Figure 2: The basic structure of the blockchain

for small and rarely used applications, and the ability of different applications to very efficiently interact, are important.

Due to the blockchain consensus, no one has control over the code execution or storage. A decentralized application can not be stopped, deleted or attacked in any way, unless its logic allows it. Because such an immutable application resembles contractual relations, autonomous decentralized applications are referred to as "cryptocontracts" or simply contracts.

1.3 Factors for digital asset value change

In basic economics, the correlation of supply and demand [Hay17] determines the price of an asset. Higher demand or lower supply relates to higher price. By buying a digital asset from an exchange, you are increasing the demand for this asset, and with that, its value. In terms of the blockchain, this buy/sell asset activity is contained within the blockchain transaction data.

Because the digital currency's market is still small in terms of market capitalization, it is very unstable and volatile. Furthermore, many outside factors have been found to cause significant chances to the currency's value. Currently, Ethereum is suffering from artificial demand increases that are well beyond the demand for the asset's usage. Numerous of factors are to blame for that. Based on our observations of Ethereum, we have found the following factors to have a significant influence on the supply and demand of the currency.

1.3.1 Speculative investment

Speculative investment is done by buying and holding a certain asset, speculating that its value will raise in the future. This is an issue in the case of Ethereum's low overall supply, due to investors increasing the demand for the asset and decreasing its supply by holding their tokens.

In our research, we investigate methods that track the influence of this method to the end value.

1.3.2 Initial Coin Offerings

Intial Coin Offerings (ICOs) [Inv17b] are a heavily discussed topic when it comes to digital currencies. They are a form of fundraising for starting projects. The project's team creates and a new digital coin, and during a certain initial period, they sell their coins for another digital currency. The expectation of the buyers is that if the project becomes successful, their owned coins will raise in value. The idea of an ICO has existed for a couple of years, but recently the total cumulative ICO funding has increased by more than 80 times [Coi17a]. Unfortunately, investing in an ICO does not bear any of the legal protections that other investors have, which makes them a risky investment.

Without a doubt, the excitements and disappointments around ICOs have left a major mark in digital currencies as a whole. Due to Ethereum's smart contract framework, it has become the most targeted platform for ICO fundings. This is believed to have been a major reason for Ethereum's value spike during 2017.

We will track ICO activities on the blockchain in attempts to prove that they are largely related to value changes.

1.3.3 Altcoin interference

After Bitcoin, hundreds of other cryptocurrencies have emerged. Despite that they are completely unrelated and independent from each other, there have been cases where an event in one cryptocurrency can cause unexpected fluctuations in other cryptocurrencies. The ICO "explosion" in Ethereum has caused value increases in many other currencies. For example, the Litecoin value has surged at around the same time of Ethereum's raise [Coi17b]. Our explanation is that the cryptocurrency users still do not fully understand the theoretical concept of such a cryptocurrency and that the currencies are not technologically related or dependent on each other.

The interference of other alternative coins to Ethereum leads to a whole research topic on its own. We have plans to include historical data and events from other currencies to improve our predictions.

1.3.4 Media influence

Much like the altcoin interference, due to the high volatility of cryptocurrencies, their value can be easily influenced by media publications. A prime example of this is a case of widespread fake news about the *apparent* death of Ethereum's founder Vitalik Burterin [Rob17]. This lead to temporary plummet and loss of 4$ billion from Ethereum's market capitalization.

We will experiment with cases where we track the news stores about Ethereum and similar cryptocurrencies, for example via sentiment analysis.

1.4 Use of Deep Learning in asset value predictions

Deep Learning [YYG15] is the act of designing a multi-layered neural network architecture and training it with large amounts of data. This technique is especially useful when there is a need to find a correlation, dependence or patterns in a large sequence of data. That makes it ideal for our needs, as we are trying to find if, given data based on the blockchain transactions, we can find a pattern with value change.

2 Data sources

We gather two kinds of data: Historical financial tick data and raw Ethereum blockchain data. The financial data is aggregated from multiple exchanges to achieve a global view of the financial state, without being heavily biased towards the activity of a single exchange, as this often does not reflect the true state. We also process all data in the Ethereum blockchain, which includes every value transaction, every crypto contract usage and every blockchain event throughout the existence of the cryptocurrency. The blockchain is hundreds of gigabytes in size, growing each day [Eth17b]. Due to the large size, we filter out most of the irrelevant to our research raw data.

Our blockchain data spans from 8-08-2015 and covers any event in the existence of Ethereum. For each block in the chain, we hold the following data points:

- **Date**: The timestamp when the block was created;
- **Miner**: The account of the miner who created the block;
- **Transactions**: List of transactions that are contained in it;
- **Gas limit**: The limit on computational resources for the block;
- **Gas used**: Sum of computational resource usage from all transactions;
- **Number**: The block index;
- **Size**: Size of the block body in bytes;
- **Difficulty**: Measure of how difficult it is for a miner to mine a new block. Measured in Hashes;
- **Total difficulty**: Sum of all difficulties of previous blocks down the chain.

For each transaction in each block, we save the following data points:

- **From/To**: Initiator and receiver of the transaction;
- **Value**: Amount transferred. Measured in Wei;
- **Gas**: Use of computational resources for the transaction execution;
- **Gas price**: The price of one gas unit in Wei that the initiator has paid.

Our financial data also covers the interval from 8-08-2015. The size of each tick is one hour. This is the most frequent and highest quality data that we were able to find public. For each tick we have information about the *open*, *close*, *low* and *high* prices, as well as the trade volume to and from the currency.

The total size of our filtered raw blockchain data is around 60GB, containing $4,300,000$ blocks with a total of $60,331,058$ transactions. It took 14 consecutive days to download and another three to parse, process and save to a database.

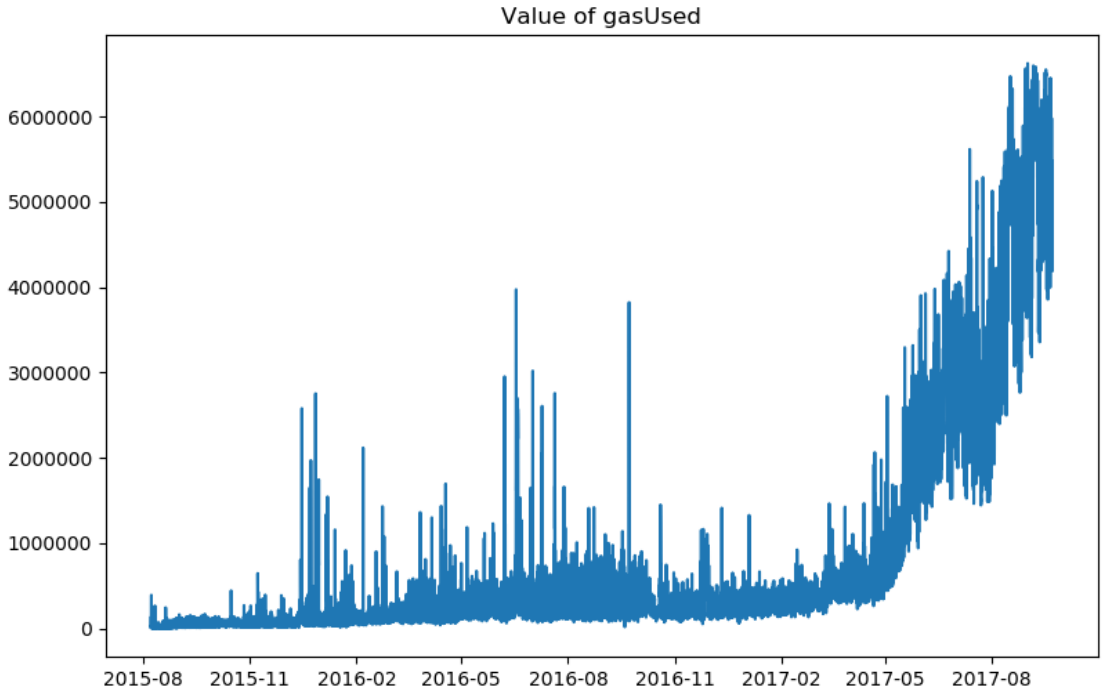

Figure 3: The values of property $gasUsed$ over time.

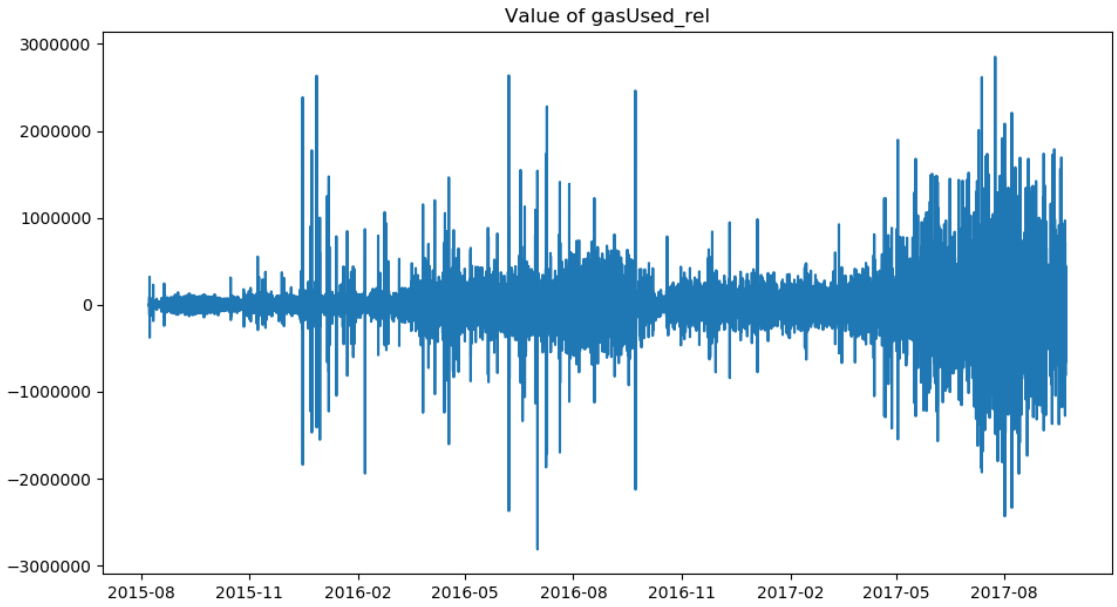

Figure 4: The relative values of property *gasUsed* over time.

3 Generation of data properties

Our data is raw and basic. In order to optimize the learning process and minimize overfit on our neural models, we need to extract the most important properties (or features) from our data. We are going to apply post-processing of the raw data and calculate certain properties that we have picked based on their importance and significance to the end goal of our research. For each financial tick in our historical data, we calculate the value of each property in our property set. Generated properties are absolute values. We turn them into relative ones and save both absolute and relative variants. Please see examples at Figure 3 and Figure 4.

The basic properties that we generate are denoted in Table 1. The following subsections explain more advanced concepts to expand our set of properties.

Open value	The value of ETH at the start of the tick
Close value	The value of ETH at the end of the tick
Stick price	The difference between the close and open prices
Volume to	Exchange volume to that cryptocurrency
Volume from	Exchange volume from that cryptocurrency
Transaction count	Amount of transactions
Operations with Dapps	Amount of transactions to crypto contracts
Block size	The average size of a block in bytes
Difficulty	The average difficulty for block mining
Network account growth	How many new accounts have been established in the network
Gas limit	Average limit on computational resource usage
Gas price	Average price per gas unit
Gas used	Average amount of gas used
Total network hashrate	Combined hashrate of every miner in the network
Total Ether supply	Total amount of Ether in circulation
Pending transaction pool	How many transactions per minute are pending for inclusion in the next block
Blockchain size growth	Growth of the blockchain size in gigabytes

Table 1: List and description of the properties compiled from the raw data. The properties relate to a time period of one financial tick.

3.1 Financial indicators

Other than the previously listed properties, we also calculate a set of financial indicators that are often used with currency predictions. The indicators include Simple Moving Average (SMA), Exponential Moving Average (EMA), Relative Strength Index (RSI) and Moving Average Convergence Divergence (MACD). Usually, this set of indicators is the only data used by traditional traders on the stock market. We experiment whether adding these indicators to the rest of our data, results in higher prediction accuracy.

3.2 Account balance distribution

Another method to extract the valuable information from our raw data is by creating an account balance distribution. We take all active Ethereum accounts within a period of one price tick and distribute them on a graph based on their balance. We assign a certain group to each account. This is denoted by the following equation:

$$group = \min(\lfloor \frac{\log_{10} bal}{\log_{10} max} * groupN \rfloor, groupN - 1)$$

Where bal is the balance of that account, max is the maximum account balance at that point and $groupN$ is the number of groups (usually 10). The use of a logarithmic scale is due to the fact that the correlation between balance and number of accounts is non-linear.

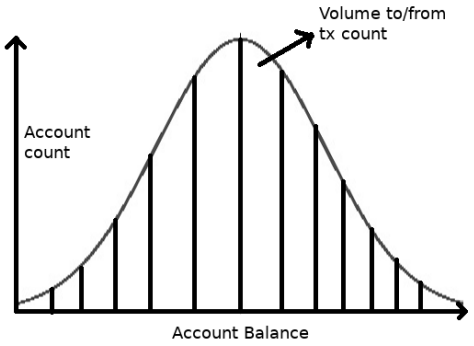

Figure 5: An example account balance distribution.

Then, for each group, we calculate the total volume of ETH transferred to, transferred from and total number of transactions initiated by any account from the group for the certain financial tick. These 3 values calculated for each group are saved as 3 different properties. Therefore, this set of $3 * groupN$ properties represents a snapshot of the summarized activity in the network for that period of time. Unlike the aforementioned properties, this set is spatially-related. The properties in the set are not independent from one another and their ordering must be kept in order to preserve their spatial value. Please see Figure 5 for a visual example of the distribution.

3.3 Account number distributions

While the account balance distribution has spatial relations, the end result is a list of property values, being shaped as a (1 x $3 * groupN$) matrix. If we want to increase the spatial value, we need to increase the dimensions of the resulting set of properties. We would also like to process data outside of just one financial tick, making a whole network state snapshot. For every financial tick, we take all ethereum accounts in existence at that point and assign two group numbers to each one of them. Please refer to the following equation:

$$group_{1,2} = \min(\lfloor \frac{scale_{1,2}(feature_{1,2})}{scale_{1,2}(max_{1,2})} * groupN_{1,2} \rfloor, groupN_{1,2} - 1)$$

Where $feature$ can be any arbitrary feature which we want to track for the accounts. Example features include days since last transaction, total ETH volume sent in the last 30 days, average transaction value, average time between transactions and others. The value of max is the maximum value for that feature. Any accounts with feature values larger than that are assigned to the last group. The *scale* function can be any scaling method and depends on the nature of the feature which we are tracking. Examples include no scaling ($y = x$), logarithmic scaling ($y = \log_{10} x$) and the alike. Because we are assigning two group numbers for each account, we have two sets of independent groups (denoted by indices 1 and 2). Each set of groups can track a different $feature$, *scale* a different way and have a different $groupN$. We arrange the accounts in the following matrix (please refer to table 2).

group	0	1	2	... $groupN_1 - 1$
0	4	567	485	898
1	3497	34	2358	497
2	0	34986	5798	2
... $groupN_2 - 1$	56	2973	45798	6

Table 2: A spatial representation of an account distribution based on two groups.

In a sense, we are creating two account distributions and are crossing them together. The X-axis is $group_1$ and the Y-axis is $group_2$. The value at (x, y) is the amount of accounts that are assigned to the two groups. This way we are able to create a list of properties with the ($groupN_1$ x $groupN_2$) shape. This matrix contains more spatial value, as the ordering of both coordinates is required to be preserved. This distribution of accounts gives us a clear snapshot of the current network state, not just the activity within one financial tick. The concept adds $groupN_1 * groupN_2$ unique properties to our property set.

4 Generation of a dataset

4.1 Prediction target

Our prediction target is the value of one or multiple properties in the financial tick that follows the certain dataset sample. Because we calculate the values of each of our properties for each financial tick, the prediction target can be the value of any one property or group of properties from our property set. We have chosen to focus on predicting the value of the cryptocurrency tokens, but our prediction strategies are perfectly suitable for a wide range of predictions reguarding cryptocurrencies. Example targets can be predicting how many new users will use the currency, how much will the trade volume be and how many transactions will be executed. We can even go as far as to predict cloggings of the cryptocurrency network (a serious issue in the light of recent events [Mad17]) and possibly attempts of 51% attacks [Inv17a].

4.2 Normalization

Operations with large floating point values in computer arithmetics often lead to a significant difference when compared to the expected and intuitive answer. This is known as numerical instabilitiy and is an issue in neural networks, where a single input value can undergo thousands of floating point operations. To combat this, we need to normalize every dataset value to fit within small interval, in our case $\in [0, 1]$. We have proposed the following solutions to the problem:

$$basic_i = arr_i - min(arr)/(max(arr) - min(arr))$$

Basic scaling to normalize any sequence. It requires the whole sequence at once, and there is no guarantee that future values of the same sequence can fit the initial normalization.

$$around\_zero_i = \frac{arr_i + max(|max(arr)|, |min(arr)|)}{2 * max(|max(arr)|, |min(arr)|)}$$

A variant of basic normalization, where positive inputs are mapped to $\in [0.5, 1.0]$, while negative ones fit within $\in [0.0, 0.5]$. We recommend this approach when normalizing signed values, as the zero always stays fixed.

$$sigmoid_i = \frac{1}{1 + \exp(-arr_i)}$$

Non-linear normalization function that provides consistent normalization no matter the sequence. We recommend this to be used with relatively small values, as normalizing large enough values results in a practically indistinguishable difference in the output value.

Normalization is done before generating the dataset with a sliding window over the property values. Each independent property can be normalized via a different algorithm, depending on the function of that property.

To combat sequences with large varience, we also investigate cases where the inputs to the normalization algorithm are first scaled by a logarithm or another scaling function.

However, problems emerge when we have large sequences that grow over time (like in Figure 3), as future values after we generate the dataset may not fit the initial normalization.

A method to deal with this problem, we employ methods of local normalization. That is, normalize each window of n values separately, instead of all property values at once. This gives the advantage that resulting values will be mostly consistent, regardless of how the absolute property value increases over time.

Another, arguably more successful, method to solve the ever increasing value problem is to turn the absolute property values into relative ones (like Figure 4) and build the dataset using them. This method has the advantage that later values usually easily fit within the same normalization, because only the difference is used. We recommend the *around_zero* algorithm in this case. As later experiments show, predicting based on a relative value also largely lowers the overfit and forces the neural network to find an accurate solution to the prediction problem.

To increase the probability that future values will fit within the initial normalization, we have created a technique called normalization space overprovisioning. It consists of leaving out a percentage of the normalization space unused. For 20% overprovisioning, that would mean fitting all values in the sequence within $\in [0.0, 0.8]$ for the case of basic normalization and $\in [0.1, 0.9]$ when using *around_zero* normalization.

Complex properties that depend on the placement and existence of others (like the discussed in sections 3.2 and 3.3) should and are normalized with the same normalization strategy and at once, not in separate runs. This is important, as the proportion of normalized values in the different properties must remain the same, or have a consistent coorelation. We call this strategy joint normalization.

Based on our observations and experiments, we recommend using the relative values method, coupled with *around_zero* normalization and 20% overovisioning for most properties.

4.3 Dataset models

Up until now we have discussed methods to extract the most valuable data points from our enormous raw data. In order to train a neural network for value prediction, we need models to compile normalized property values to form a dataset. All of our models have the following in common. We are working with a configurable set of properties. To generate the dataset samples, we use a sliding window with size of n and step of 1 over the property values. Please refer to the following equations:

$$x \in [0, n-1]$$
$$y \in [0, property\_count - 1]$$
$$window_{x,y} = propertyY_{x+step}$$
$$target = propertyT_{n+step}$$

Where $property0, 1, 2, .., y$ is a list of the chosen properties. $step$ denotes the step of the moving window, which is incremented by one for each dataset sample. $propertyT$ is the prediction target, which may or may not be within the chosen properties. $target$ denotes the value of the prediction target for that dataset sample.

We transform each window to a matrix. The structure of the matrix depends on the chosen dataset model. The value of n is configurable and dependent on the chosen set of properties, the size of the financial tick, the chosen neural model and the alike.

We have created the following dataset models that take as an input the chosen properties and the chosen prediction target, run the sliding window property value iteration, execute a chosen normalization strategy and arrange the values a certain way in a matrix.

4.3.1 Matrix model

The matrix model is an LSTM-friendly representation of property values. A single dataset sample is created as follows:

$$sample_{x,y} = normalization_y(window_{x,y})$$

Where *normalization* is the chosen normalization strategy for the certain property.

Please refer to table 3 for an example of a generated dataset sample.

The format has a lot in common with LSTM network's input shape, which is (*time_steps*, *features*). For our case, each *feature* is a property and each property value is a *time_step*

Property	Value 0	Value 1	... Value $n + step - 1$
Open value	345$	356$	
Close value	349$	358$	
Number of TXs	2876	1583	
Operations with Dapps	459	508	
Total volume exchanged	5349478$	173808$	
...			

Table 3: An example of a non-normalized dataset sample generated by the matrix model.

4.3.2 Stacked layers model

The stacked layers model is based around Convolutional Neural Network's perception of space. The idea is to take property values and stack them to form a multi-layered image. We still use the sliding window to select property values for our sample, however this time we *unpack* the values for one financial tick from a one dimensional column to two dimensional space, forming a layer. The length of the window n denotes the number of layers that we form, which we stack together to form a single image with n color channels. The shape of a dataset sample is (*height*, *width*, n), where $width * height = property\_count$. The size of the image is configurable, as is the ordering of the unpacked properties. Because this model allows 2D ordering of properties, it can successfully compile account number distributions (Section 3.3), which require 2D spatial alignment.

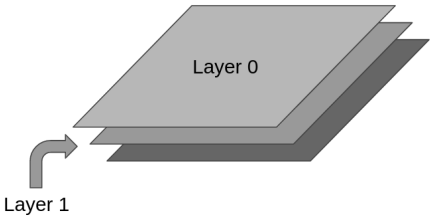

Figure 6: A visualization of the stacked layers model.

5 Generated datasets

We have generated multiple variants of a dataset, on the base of which we are running our experiments. Each dataset is split on three sets: *warmup*, *train* and *test* datasets respectively. As we have analyzed the history of Ethereum (1), we have chosen the most appropriate time interval for our datasets to be from 2017-03-01 to 2017-11-01. This range contains the most important events in Ethereum, including Ethereum's all-time high price (2017-06-12), the hoax over the death of Ethereum's founder [Rob17], the rise of Initial Coin Offerings (ICOs) [Inv17b] and the largest fluctuations in price of Ethereum's history.

The first month of the range is reserved for the *warmup* dataset. This part is only utilized in our stateful LSTM experiments and is used to warm the network up to a certain state before training or evaluation. The following 6 months build our *train* dataset, on the basis of which neural network training is executed. Last month in our interval is taken by our *test* dataset. This set is used to evaluate the performance of the already trained network.

The datasets in use in our experiments are denoted in Table 4:

6 Measures of error

In order to evaluate our neural network's performance, we need good and adequate error measurements. We have chosen and use the following measurement factors during training and evaluation:

$$MSE = \frac{1}{n}\sum_{t=1}^{n} err_t^2$$

N	Properties in dataset	Model	Window	Normalization	Target
1	relative highPrice	Matrix	104	basic	relative highPrice
2	(all relative) highPrice, volume to, volume from, transaction count, gas used	Matrix	104	around_zero	relative highPrice
3	(all absolute) highPrice, volume to, volume from, transaction count, gas used	Matrix	104	basic	absolute highPrice
4	relative account balance distribution	Matrix	104	joint around_zero	relative highPrice
5	absolute account balance distribution	Matrix	104	joint basic	relative highPrice
6	absolute account balance distribution	Matrix	104	joint basic	absolute highPrice
7	absolute account balance distribution	Matrix	104	joint basic	network account growth

Table 4: List of the datasets that we use in our experiments.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n} err_t^2}$$

$$R^2 = 1 - \frac{\sum_{t=1}^{n} err_t^2}{\sum_{t=1}^{n} null_t^2}$$

$$custom = \frac{1}{n}\sum_{t=1}^{n} \frac{|err_t|}{min(prediction_t, actual_t)}$$

$$err_t = prediction_t - actual_t$$

$$null_t = 0.5 - actual_t$$

Where n denotes the number of samples in our dataset, *prediction* and *actual* are lists of our predicted and expected values respectively.

R^2 is an interesting measure of accuracy, where the performance of our model is compared to the performance of the *null* model - a model, that returns the same value (0.5 in this case) no matter the input. The aim is to reach the perfect R^2 score of 1.0. MSE and $RMSE$ are measures of error, where the target is to reach 0.0. With efforts to observe our network's performance from different angles, we have created our own error function, called *custom*.

We use MSE as the training loss function for all of our networks, with the rest being used for evaluation of trained network performance.

7 Hyperparameter optimization

Some deep models can have tens, if not hundreds of hyperparameters that have to be tuned in order to achieve low error rates. Other than our supervised experiments and manual fine-tuning of hyperparameters, we also have written certain short algorithms that try to further optimize some of the parameters of our networks by continuous training and parameter changes. They receive initial hyperparameters as an input and experiment by changing one or multiple hyperparameter values at once based on predefined rules. This is a very time-consuming process and as such, a good mix of supervised and automatic tuning is required.

8 Experiments

8.1 Neural Network architectures

For our experiments with the generated datasets, we have created 3 neural network architectures - a regular deep model, a Long Short-Term Memory (LSTM) model [Hoc97] and a Convolutional model. Please see their network architectures at Figure 7. The purpose of the deep model is to use it as a baseline when comparing the performance on different networks on the same dataset. Because of its simple structure, we do not expect the best results to be from it. Due to the fact that our items in our dataset are a sequence of time intervals with property values, we experiment with LSTM networks [Hoc97], which are well known for their accuracy in working with sequential or time series data and it would be interesting to see how well they interpret our sequences. LSTMs can be stateful or stateless. Stateless networks get their memory (or state) reset after each batch of data, while stateful ones keep it. Because we experiment with distributions (Sections 3.2 and 3.3) that have strict value in space, rather time, we also investigate Convolutional networks, known to perform well in finding dependencies or patterns in spatially related data.

We train each of our networks on each dataset listed in Table 4, evaluate their performance and comment on the results.

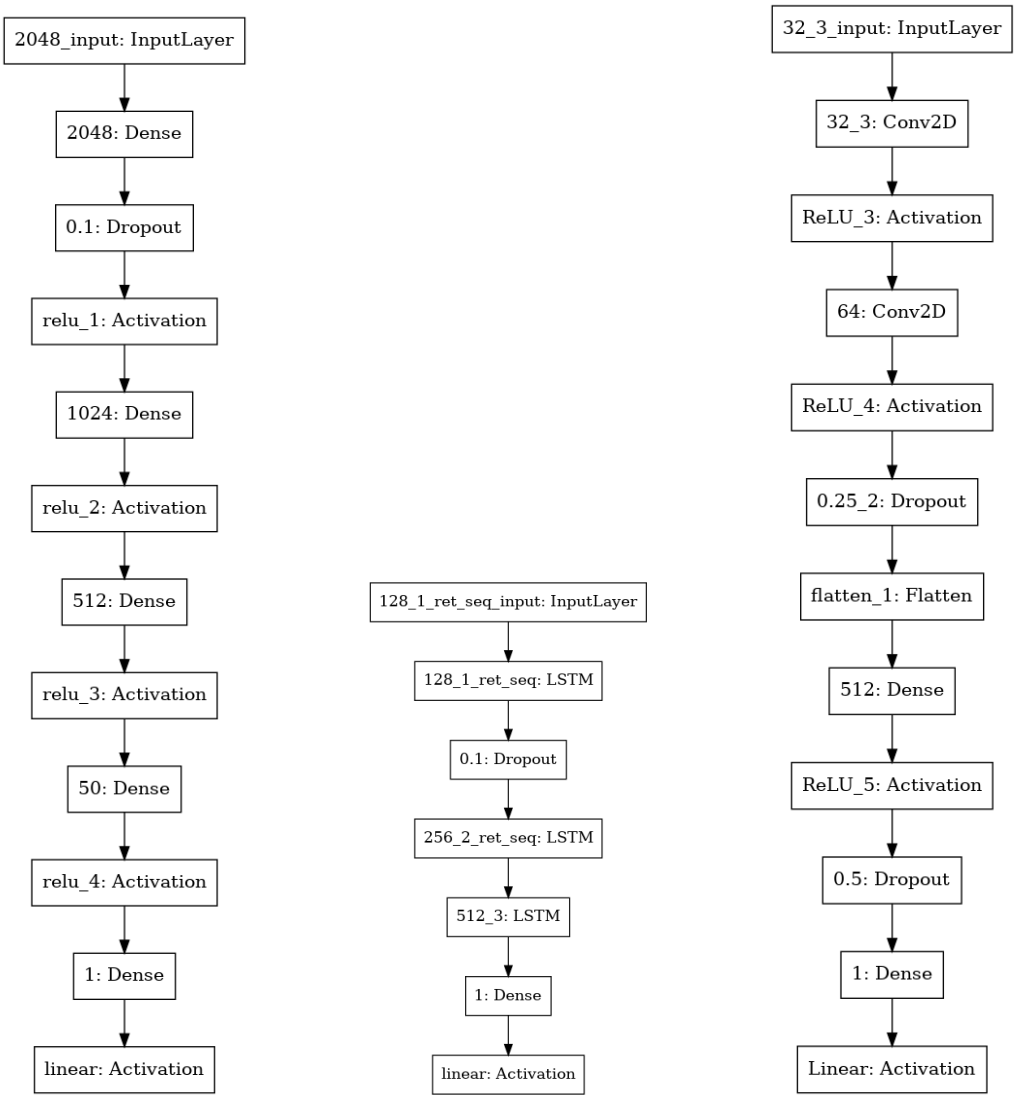

Figure 7: The architecture of standart Deep Network, LSTM and Convoluitonal models respectively

8.2 Experiments with Deep networks

The first model we created is a standart deep model. We have trained it on every listed dataset via Stochastic Gradient Descent [Bot10] and using the Adam optimizer [DPK14]. with *batch_size* of 16 and 0.0001 *learning_rate* on 10 *epochs*. We have chosen this *batch_size* for all our networks, because it provides a balance between training efficiency and network loss. The results are in Table 5.

N	Test dataset scores			Train dataset scores		
	RMSE	*custom*	R^2	*RMSE*	*custom*	R^2
1	0.074765	0.116757	-0.063221	0.055050	0.074970	-0.019824
2	0.073415	0.113763	-0.025179	0.054776	0.072237	-0.009714
3	0.032130	0.036633	0.985606	0.016738	0.031025	0.996094
4	0.072570	0.111599	-0.001708	0.054830	0.070051	-0.011676
5	0.080377	0.132644	-0.228850	0.040486	0.060048	0.448396
6	0.152966	0.203045	0.673763	0.039350	0.092518	0.978410
7	0.081765	0.322353	0.912021	0.042026	0.433335	0.988188

Table 5: The results of the experiments on deep networks.

Each experiment number is corresponds to the dataset number in Table 4.

Please keep in mind that results from experiments with different prediction targets are incomparable directly. In this case, experiments 3 and 6 predict absolute price, experiment 7 predicts new accounts, while the rest - relative price movements.

Experiment 1 shows the network's inability to find a pattern in the data, consisting of a series of past prices. Please refer to Figure 8. A big part of stock market predictions are done solely based on past prices, and the low results here are surprising. Hundreds of experiments on the same dataset with different hyperparameters and network architectures gave similar results. We conclude, that cryptocurrency markets are simply too small, unpredictable, stochastic and volatile and therefore accurate predictions solely based on historical price data seem unlikely. We need more data for better results. Experiment 2 gives similar results, despite having more properties in the dataset. This is likely because of the deep network's inability to find relations between the different time steps. More training is required to give a clear answer.

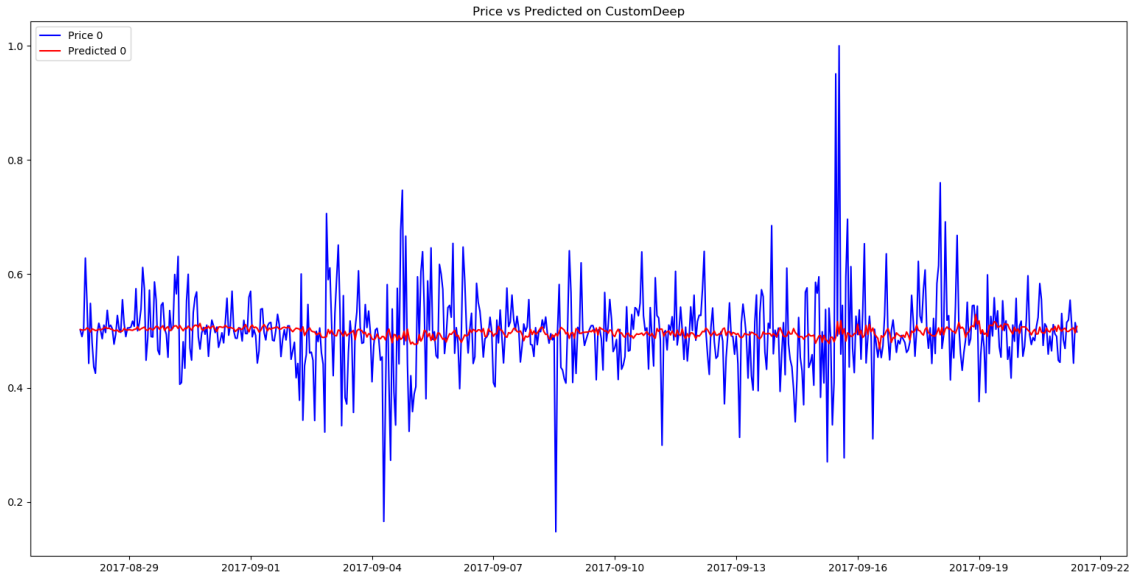

Figure 8: The test results of experiment 1, visualized on a graph

Experiment 3 is a classic case of overfitting. It is giving impressive accuracy results both on train and test datasets. However, once we examine the graph (Figure 9), we observe that no predictions are taking place at all. Instead of predicting, the network gets the current price from

the input dataset and returns it. We suggest, that the reason is the lack of patterned data. The network finds this behavior as the best way to lower its loss. We have observed similar results in all cases, where we predict the absolute price and the previous price values are included in the dataset. Our 2 suggestions to force the network into finding a better pattern in the data are to always predict relative price value, or do not include previous price values in the dataset.

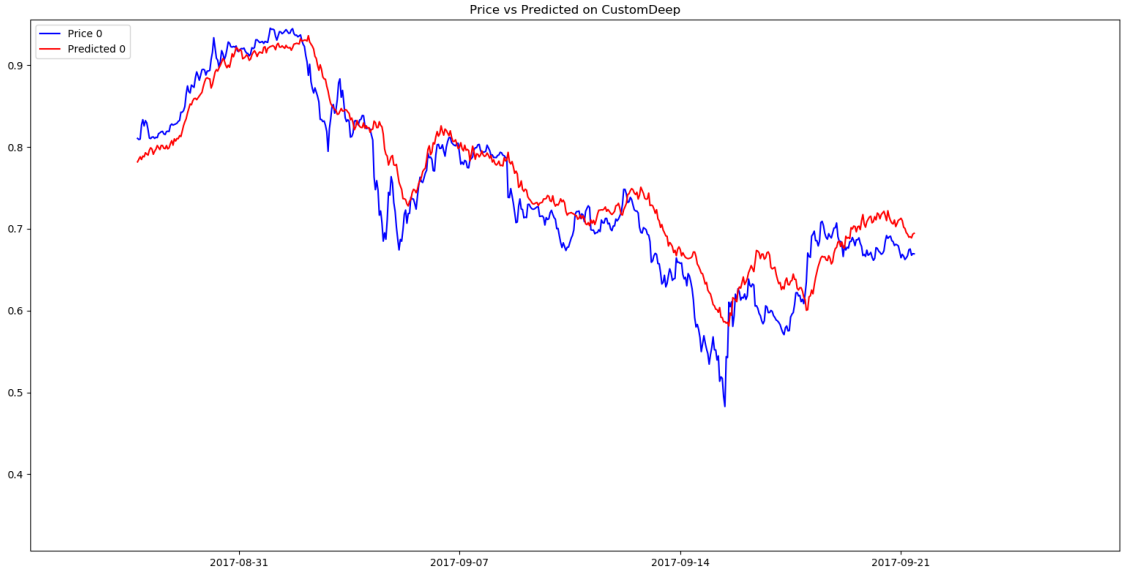

Figure 9: The test results of experiment 3 visualized on a graph

The dataset in experiments 4 and 5 consists only of relative and absolute account balance distributions respectively, without any price data. The goal is to find a correlation between the distribution and the future price. With the relative distribution, the network performance is almost identical to that of experiment 1. However, what surprised us are the results from experiment 5. While the test results in numbers are similar to our previous ones, the train results area skyrocketing compared to any previous values. Evidently, this signifies serious overfitting on the train dataset. Please refer to Figures 10 and 11 for a detailed visualization of the train and test performance. While we can observe strong overfitting in the train dataset, we can clearly see that the network is much more confident in its predictions on the test set. Instead of the usual flat line of predictions, we see more aggressive attempts to predict, many of which perfectly match the real values. This is a prime example why pure accuracy values should not be blindly trusted, without visualizing the performance on a graph. The better performance is a result of the larger and richer input data. We conclude that further research with different regularization strategies to lower the overfit should be conducted.

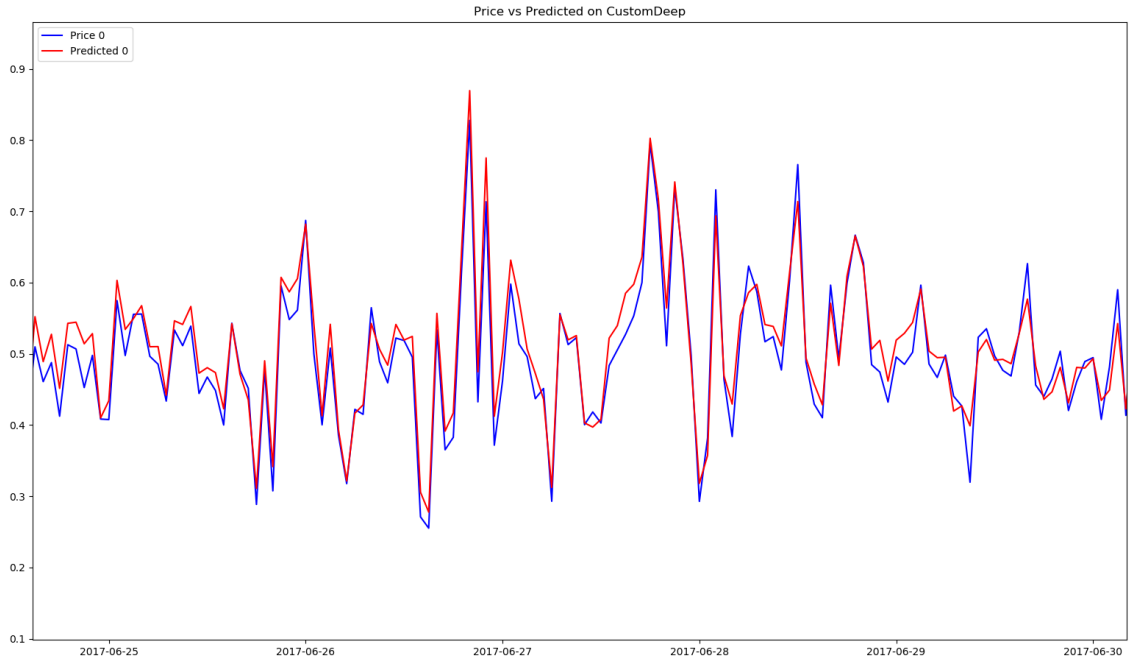

Figure 10: A close view of experiment 5 performance on the train set.

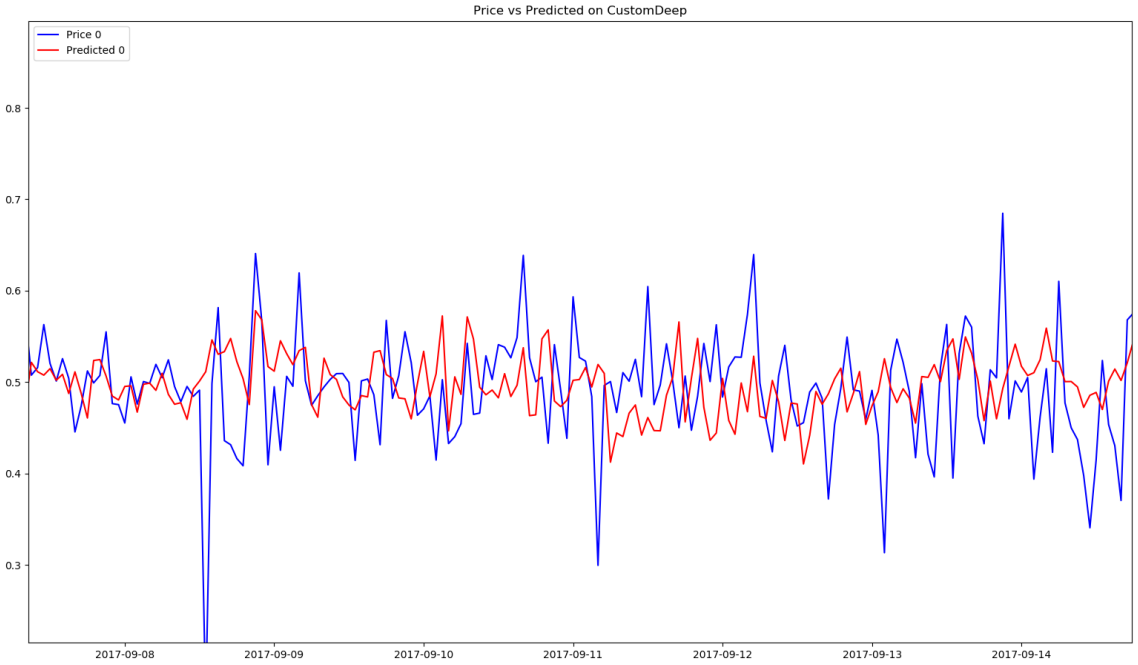

Figure 11: A close view of experiment 5 performance on the test set.

Experiment 6 results are simular to 5, except that the predidction target is absolute price. Confirming our previous findings and claims, there is a serious overfit on the train dataset once again. On the test set, the predictions are more aggressive and in many cases right.

Other than the aforementioned experiments, we have ran hundreds of others, all with different network architectures and datasets. We have not seen major improvements when tuning the network hyperparameters or layers, which again points to the fact that our current dataset is not rich in patterns. Among our other experiments, we have tried with different amount and sizes of

the hidden layers, different location and percentage of dropout [NSS14], L2 regularization [Ng], small and large *batch_size*, different *learning_rate* and various number of training *epochs*.

8.3 Experiments with LSTM

8.3.1 Stateless LSTM

We have trained our LSTM model in stateless mode, 16 *batch_size*, 0.0001 *learning_rate* on 10 *epochs*. The results are in Table 6

N	Test dataset scores			Train dataset scores		
	RMSE	*custom*	R^2	*RMSE*	*custom*	R^2
1	0.073808	0.116297	-0.036197	0.055624	0.075822	-0.041224
2	0.073112	0.113036	-0.016748	0.055475	0.072803	-0.035643
3	0.020411	0.021455	0.994191	0.014843	0.022967	0.996928
4	0.073263	0.115364	-0.020951	0.055035	0.074760	-0.019274
5	0.072656	0.112024	-0.004087	0.054583	0.069051	-0.002588
6	0.140329	0.167734	0.725439	0.130508	0.326100	0.762512
7	0.102360	0.497800	0.862118	0.058692	0.997271	0.976961

Table 6: The results of stateless LSTM.

The stateless LSTM model performs slightly better than our deep model in most cases. The results and prediction graphs of experiments 1 and 2 are more evidence to prove our claims that the data is insufficient. Experiment 3 is suffering from less overfit (Figure 12).

Figure 12: View of experiment 3 performance on the test set.

The low score on experiment 4 again shows that the account balance distribution (Section 3.2) should never be used as relative values. While the score on experiment 5 is higher, its graph more closely resembles a flat line. Experiment 6 is similar to that of our deep network, but because the graph is flatter, the network is less confident in making decisions. Experiment 7 does not show progress compared to our deep network.

We have also made experiments with different architectures and datasets. We have tested 1, 2 and 3 stacked LSTM layers with different unit count. We have tried using normal RNN cells rather than LSTM ones, which resulted in a difference on the model's behavior but not on the model's accuracy. We have tried using Bi-Directional RNN wrappers [MKA97] with both LSTM and RNN cells with no significant difference.

8.3.2 Stateful LSTM

We have trained our LSTM model in stateful mode with the same hyperparameters as our stateless experiments. We have modified all datasets to have window size of 1, because we should rely on the network's memory for the previous values. The results are in Table 7

N	Test dataset scores			Train dataset scores		
	$RMSE$	$custom$	R^2	$RMSE$	$custom$	R^2
1	0.087518	0.155499	-0.456880	0.074534	0.134930	-0.869470
2	0.079127	0.134962	-0.190925	0.061922	0.100856	-0.290330
3	0.015270	0.015688	0.996749	0.014557	0.051777	0.997045
4	0.075905	0.121674	-0.095889	0.060270	0.090136	-0.222396
5	0.076813	0.122695	-0.122283	0.064126	0.101971	-0.383832
6	0.288459	0.578946	-0.160145	0.260995	1.380791	0.050198
7	0.127947	0.782952	0.784569	0.077963	1.319233	0.959350

Table 7: The results of stateful LSTM.

Experiments 1, 2 and 4 are similar to the previous. Experiment 3 achieves significantly lower error scores, compared to our deep model. This is due to a larger overfit. Experiment 5 achieves lower error, compared to both our stateless LSTM and deep model. The model is also more certain in its predictions. Experiment 6 is more interesting, because we observe an unexpected pattern in the predictions (Figure 13). This is likely due to the large amount of unique input values, which lead to diffculties in collecting the needed state in time for predictions.

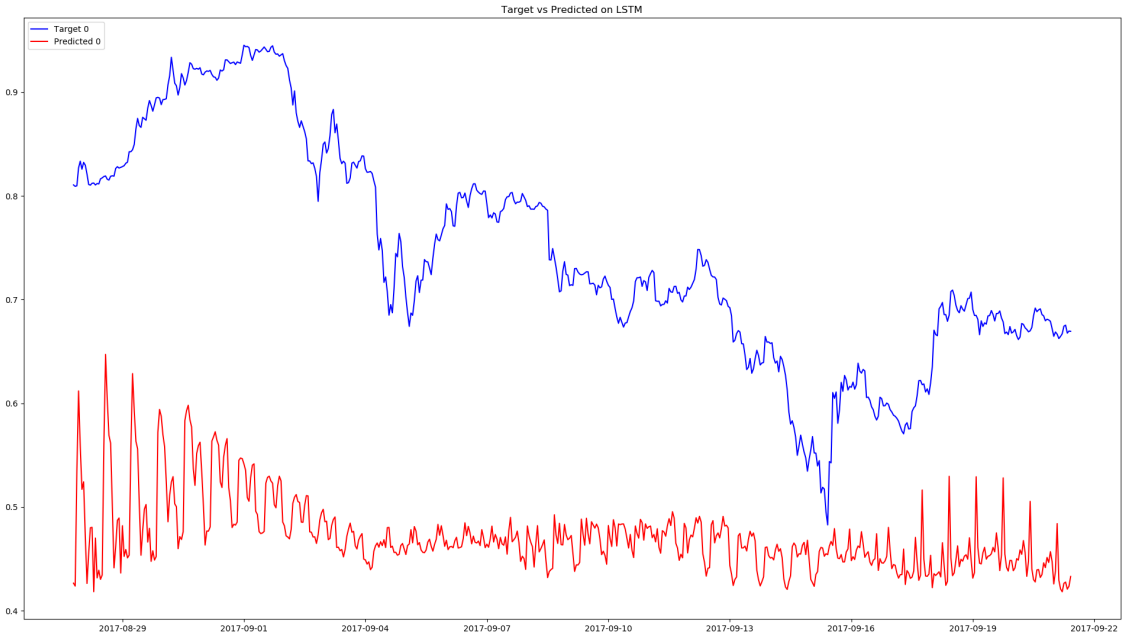

Figure 13: View of experiment 6 performance on the test set.

In experiment 7, the stateful network is more confident in some and less confident in other predictions. The error scores are similar to our previous findings.

We have also experimented with different number of stacked stateful LSTM layers, their unit count, as well as Bi-directional wrappers [MKA97]. The results on our other experiments are similar in most ways to our previous findings.

LSTM networks perform best with one or multiple input sequences (in our case, property sequences). However, they tend to slow down with a large number of sequences and have shown inability to find cross-sequence patterns. Because Ethereum's volatility requiers as much data as possible, we conclude, that pure LSTM networks are inefficient for our problem. However, we have hopes in complex architectures that also include LSTM layers.

8.4 Experiments with CNN

We have also conducted experiments with Convolutional Neural Networks (CNNs). Our model consisted of a few convolutional layers, ReLU activations inbetween and two dense layers in the end (see Figure 7 for more details). All of our kernel sizes are $(3, 3)$ with strides $(1, 1)$ and *same* padding on every second convolutional layer.

N	Test dataset scores			Train dataset scores		
	$RMSE$	*custom*	R^2	$RMSE$	*custom*	R^2
1	-	-	-	-	-	-
2	0.074589	0.124495	-0.058241	0.051280	0.076216	0.115087
3	0.072088	0.426569	0.927545	0.052330	0.137169	0.961816
4	0.072659	0.119591	-0.004180	0.053763	0.076784	0.027314
5	0.077936	0.136401	-0.155344	0.050324	0.082316	0.147759
6	0.153790	0.938401	0.670241	0.086252	0.234185	0.896268
7	0.080395	0.078045	0.914944	0.047036	0.041982	0.985204

Table 8: The results of CNN.

Experiment 1 could not run due to insufficient data.

Experiment 2, compared to our previous results on this experiment, is performing significantly better. The network is able to predict the value more accurately, it is more confident and its predictions are more aggressive. Please refer to figure 14. The overall performance is still not the best we have got up until this point, but it shows how the convolutional network can extract meaning even from pattern scarce datasets.

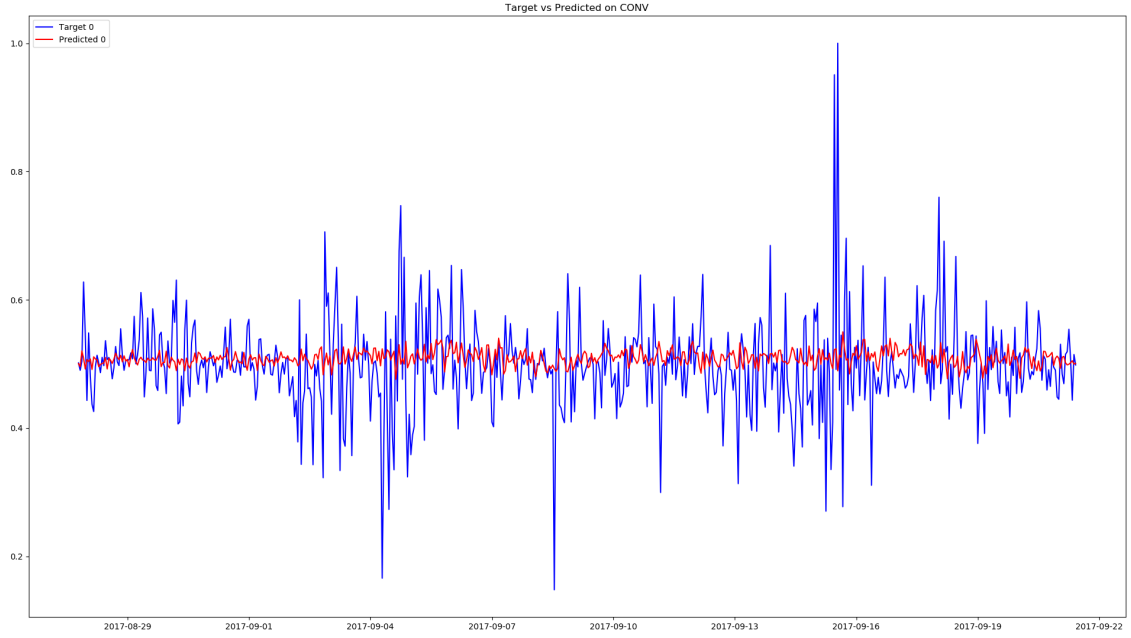

Figure 14: View of experiment 2 performance on the test set.

Experiment 3, as expected, suffers from significant overfit. Not much has changed, compared to our previous results on the experiment.

The results on experiment 4 further confirm our speculation, that the account balance distribution should not be used as relative values.

Experiment 5 is performing better than any of our previous findings. The convolutional network is able to successfully determine the sign and in many cases the value of the future change. It is more confident than any of the other networks and the overfit is lower. Please refer to Figure 15.

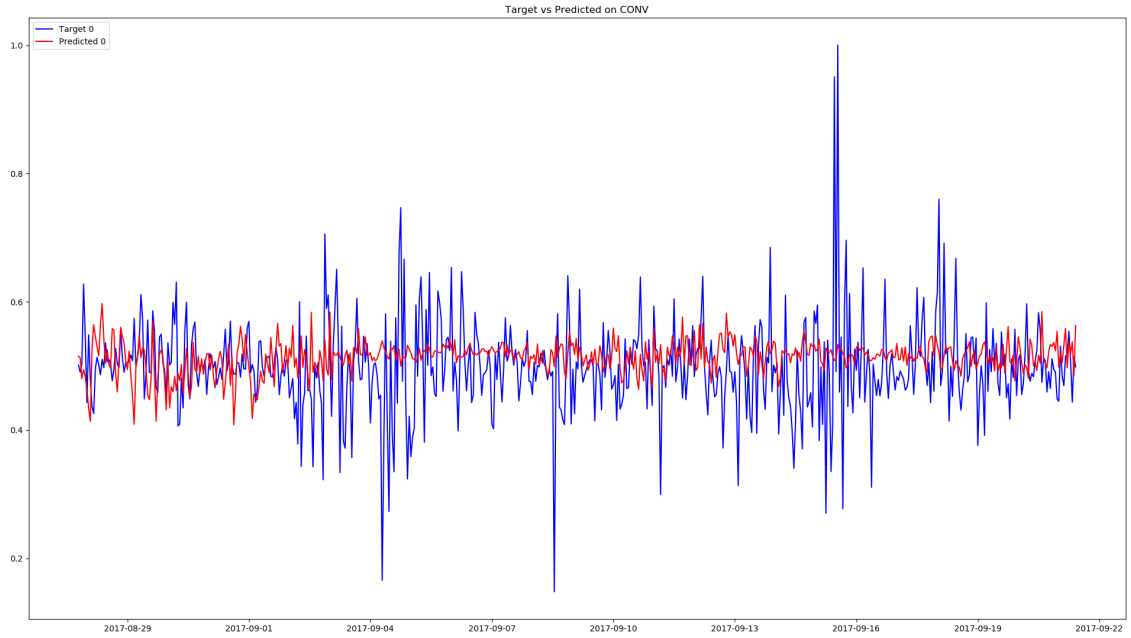

Figure 15: View of experiment 5 performance on the test set.

Our findings on experiment 6 are rather unexpected. They seem completely inverted from their expected values (Please refer to Figure 16). If we invert the predictions ($y = 1 - x$), the results are the best we have seen on this experiment. Their sign and value is mostly correct and overall the network is more confident in its predictions. Unlike previously, there is no major overfit on the train set.

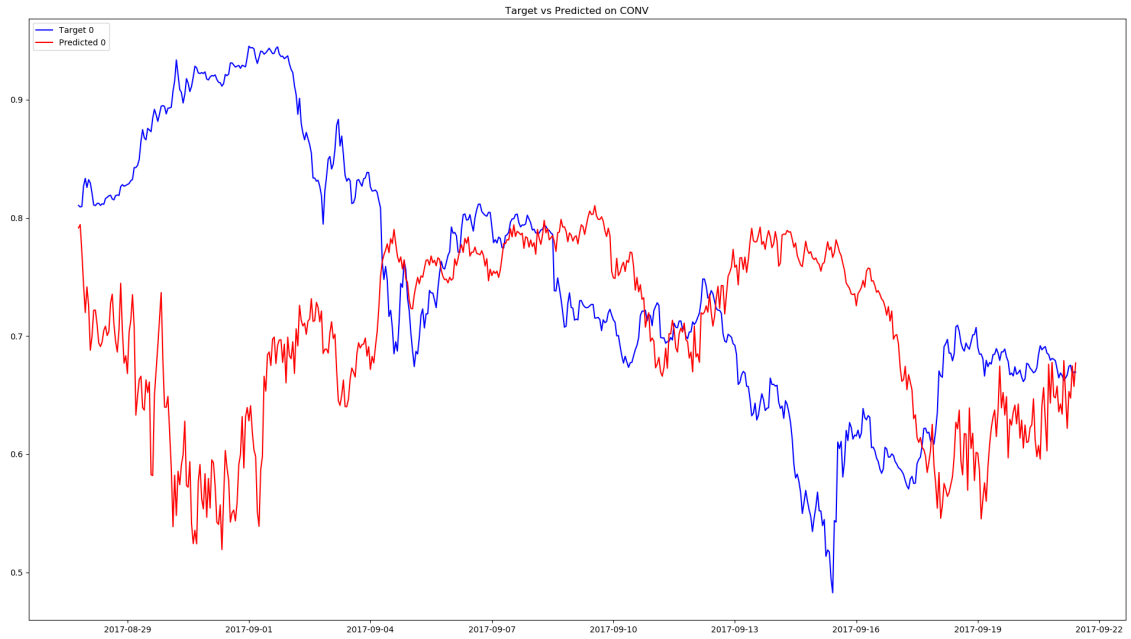

Figure 16: View of experiment 6 performance on the test set.

In experiment 7, we observe similar prediction behavior to our previous findings. The results are better in terms of error scores, but this is close to the margin of error.

The Convolutional network has given is very optimistic results so far. In efforts to test its behavior under pressure, we ran experiment 6 with 2 prediction targets. This time, the network

had to predict both the *highPrice* and *lowPrice* at once. We set *epochs* to 3 in order to test network's learning performance on insufficient training and also to reduce overfit. To improve the network's generalization, we changed the *batch_size* to 1. Please refer to Figure 17. The network is able to accurately predict both the value and the sign of the change on both of our tagets at once, despite the short training. In terms of score, please refer to Table 9

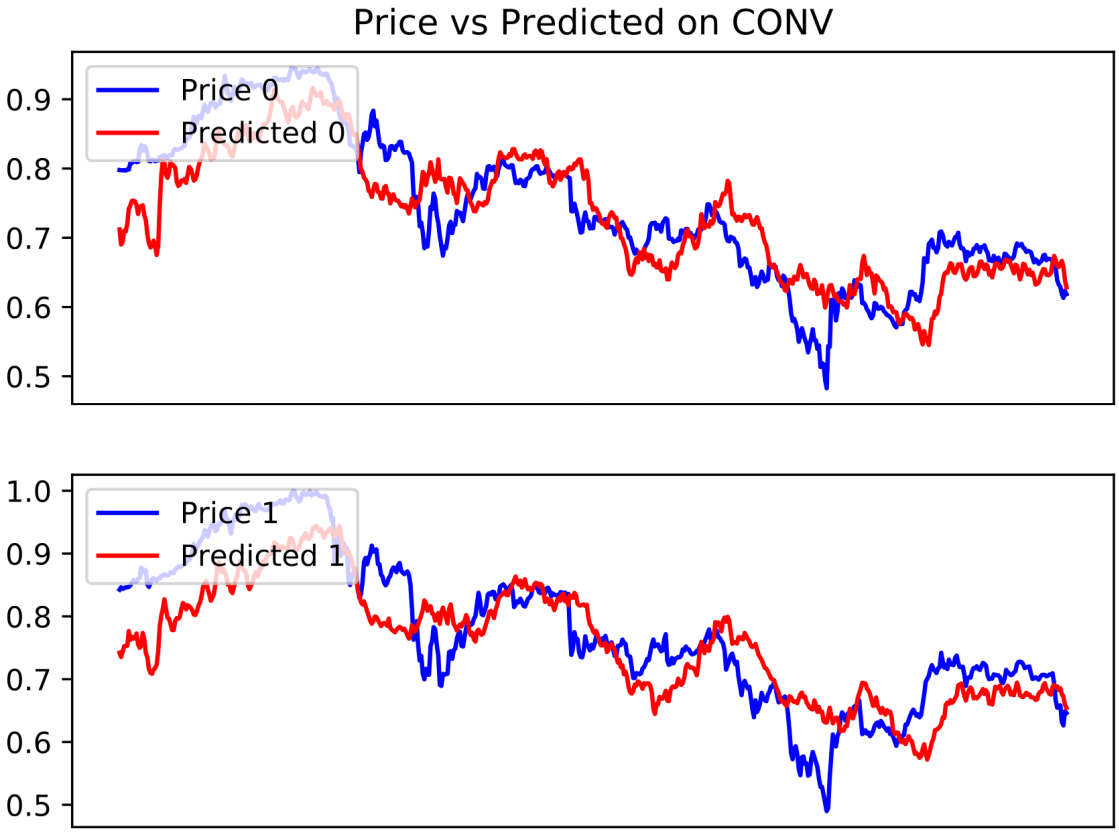

Figure 17: View of our double target experiment performance. The upper graph is for *highPrice*, while the other is for *lowPrice* predictions.

Target	Test dataset scores			Train dataset scores		
	RMSE	*custom*	R^2	*RMSE*	*custom*	R^2
highPrice	0.127368	0.776627	0.772452	0.136682	0.369659	0.744840
lowPrice	0.134406	4.569225	0.801341	0.138791	0.590322	0.751945

Table 9: The CNN results on our extensive test. Each target is scored separately.

These are the lowerst error scores and highest accuracy values that we have achieved among all of our experiments on the account balance distribution data (dataset 6). This clearly shows the dataset as the most pattern-rich and the Convolutional architecture as the best performer in fitting it.

We have tried adding different types of pooling, but this seriously decreased the network's ability to generalize. This behavior was expected, as pooling is often used when extracting non-positional features from the data. We have also conducted experiments with different ordering, size and count of the convolutional layers. Unlike our previous findings in the other neural models, every change of this type lead to largely different results. This further suggests that our convolutional network is best suited for our data, because it is able to extract the most spatial meaning out of it.

9 Results

Through our experiments, we have observed the problem from multiple different angles, by training multiple networks on multiple datasets and tracking multiple error and accuracy measurements.

Our results show, that the chosen properties in the dataset and their normalization strategy have the biggest impact on the prediction accuracy across all models. We have shown that, unlike with the stock market, Ethereum is too volatile to be predicted solely based on a few properties. This conclusion also explains the low hourly prediction scores on the Stanford Bitcoin predictions research [ISA14].

Our best results were on the datasets that used the account balance distribution (Section 3.2), which are also the largest datasets in terms of property count. We have shown that this distribution must always be used as absolute values and that the normalization strategy should be joint basic.

Open and close prices differ significantly based on the certain financial ticker. The best target we found is relative highPrice value, when we are predicting the price. We strongly recommend to use a relative target, rather than an absolute one, because this seriously lowers the ability of the network to overfit and forces it to find the right solution.

The results also revealed the most efficient neural network type for our research - the convolutional network. It was able to find dependencies in our data where all other models failed, and it provided the best accuracy scores in every single experiment we held.

About the training process, we have found the Adam optimizer [DPK14] and 0.0001 $learning\_rate$ to be the most suitable for our problem. Our experiments show, that lower $batch\_size$ always leads to higher accuracies, but also to higher train times. And while there are methods to slightly increase accuracy with large batches [PG17], the end result will always be better with a smaller batch.

10 Technical implementation

For our experiment, we have created and documented a flexible framework that allows the prediction of any cryptocurrency on the base of any data. Our properties, dataset models, normalization strategies and the aforementioned techniques are all implemented into the framework. However, it still gives complete freedom for the implementation of another technique, if it is needed for the use case.

The framework is split into multiple tools, managing data download and storage, property generation, dataset generation and neural network training and evaluation.

10.1 Data extraction module

The data extraction module holds connections to the data providers explained in Section 2. It is responsible for contacting the data providers, downloading the data, validating the integrity of it and saving it temporarily for the next module to access.

10.2 Data storage module

The data storage module takes as an input the data returned by the data extraction module (Section 10.1) and applies post-processing. It filters unimportant to our research data, and saves the rest as a time series using a data chunking algorithm that allows fast retrieval by date interval. The chunks are then saved to a local database.

10.3 Property generation module

The property generation module reads the post-processed and optimized raw data from the data processing and storage module (Section 10.2) and generates a list of properties and values based on that. The module reads the raw input data and generates the properties by calculating and composing different data properties for each financial tick. The process is described in more detail in Section 3. The generated properties are saved in the database via the data storage module.

10.4 Dataset generation module

The dataset generation module reads the generated properties from the property generation module (Section 10.3), normalizes them and applies a chosen dataset model to generate the dataset items. The dataset model and used properties depend on the neural architecture. The different methods for representation proposed by us have been described in Section 4.

10.5 Training and evaluation module

The final module is the training and evaluation module. It takes as input the created data frames by the data representation module (Section 10.4). The module uses the train data frames to train a chosen neural network model. The proposed models by us are outlined in Section 8.1. After the training process, the network runs predictions on the test dataset and is evaluated for accuracy with our proposed measurements (Section 6).

10.6 Automated tests

We have developed unit and integration tests for all of our tools, where applicable. This is done to ensure that the dataset generated by those tools is always with the expected and true values.

11 Technologies

1. **JavaScript** is used to write the data extraction module;
2. **Web3 API** is a JS library which allows the extraction of blockchain data from a local running Ethereum node;
3. **Node.js** is used to connect to other APIs that download the rest of our raw data;
4. **Python** is used for the rest of the modules;
5. **Pandas** provides high-performance data structures and data analysis tools for Python;
6. **MongoDB** is the database we use to store our data throughout all stages of data processing;
7. **Arctic** is a high performance datastore for numeric and time-series data. It is used in hedge funds;
8. **Tensorflow** is the deep learning library most commonly used in deep learning research and development;
9. **Keras** is a frontend framework for a backend (like Tensorflow) that allows fast prototyping of neural network architectures.

12 Future development

The results and experiments conducted up until now were initial. Their objective was not to solve the prediction problem, but to analyze the situation and to give us a clear idea of where to experiment next.

Our next steps will be in mastering our dataset generation. We still have not experimented with the account number distributions (Section 3.3) and the stacked dataset model (Section 4.3.2). Based on our good initial results with the account balance distribution (Section 3.2), we believe that the added spatial value of our planned techniques will result in large accuracy increases. Furthermore, using multiple account number distributions should also affect our results. Combining the distributions with our other properties is another experiment idea. We should also experiment with normalization space overprovisioning (Section 4.2), because without it, real-world and real-time usage would be impractical.

Other than perfecting our blockchain-based dataset, we also plan to track outside factors, like altcoin interference (Section 1.3.3) and media influence (Section 1.3.4).

The neural network that we will be mainly using from now on will be our convolutional model. It has provided us excellent initial results, compared to our other models. The added spatial value

in our planned dataset development should only help the model learn better and overfit less. We are interested in exploring Sequence to Sequence networks [IOQ14], in order to receive predictions for multiple steps in the future. We also plan to experiment with the Highway Convolutional Network [RKJ15], the Network in Network architecture [MQY14] and Deep Residual networks [KXSJ15]. It would be interesting to also take a look at state of the art convolutional architectures like Alexnet [AIG12], VGGNet [KA14], ResNeXt [SRP+17], DenseNet [GZKL16] and the Google Inception architecture [CVS+15].

We think that this experiment, if it is successful and gives a good result, has a direct application in the trading markets. The predictions can be transformed into "buy" and "sell" signals and tied to any trading or exchange API. We are planning to create a free web service that provides real-time value predictions on multiple digital currencies publicly in the form of a web site and an HTTP API.

13 Github repository

Our technical implementation is open source. The GitHub repository of the project can be found at the following address: https://github.com/Zvezdin/blockchain-predictor

The respository contains technical documentation reguarding the project, including setup, usage instructions and steps to reproduce our findings.

14 Conclusion

With the exponential rising interest in digital currencies and digital assets, we believe that now are the golden years of blockchain technology. With blockchain come numerous possibilities to access and process data that has never been public before. The also rising interest in Deep Learning opens other possibilities to process and find patterns and relations in huge datasets. The combination of the two has the potential to develop numerous newer and improved trading and prediction strategies with never seen before accuracies.

During our initial experiments, we have proven that there is a strong correlation between blockchain data and coin value. Further experiments must be conducted before we can give any certain conclusion on the topic.

References

[AIG12] Krizhevsky Alex, Sutskever Ilya, and E. Hinton Geoffrey. Imagenet classification with deep convolutional neural networks. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks, 2012.

[Bot10] Leon Bottou. Large-scale machine learning with stochastic gradient descent. http://leon.bottou.org/publications/pdf/compstat-2010.pdf, 2010.

[Byn15] Christopher Bynum. Bitpredict. https://github.com/cbyn/bitpredict, 2015.

[Coi17a] Coindesk. Ico tracker. https://www.coindesk.com/ico-tracker/, 2017.

[Coi17b] CoinGecko. Litecoin price chart (ltc/usd. https://www.coingecko.com/en/price_charts/litecoin/usd#panel, 2017.

[CVS+15] Szegedy Christian, Vanhoucke Vincent, Ioffe Sergey, Shlens Jonathon, and Wojna Zbigniew. Rethinking the inception architecture for computer vision. https://arxiv.org/abs/1512.00567, 2015.

[DPK14] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, 2014.

[Eth17a] Ethereum. A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper, March 2017.

[Eth17b] Etherscan. Ethereum blockchain size. https://etherscan.io/chart/chaindatasizefull, 2017.

[GZKL16] Huang Gao, Liu Zhuang, Q. Weinberger Kilian, and van der Maaten Laurens. Densely connected convolutional networks. https://arxiv.org/abs/1608.06993, 2016.

[Hay17] Adam Hayes. Suppply and demand. http://www.investopedia.com/university/economics/economics3.asp, 2017.

[Hoc97] Sepp Hochreiter. Long short-term meory. http://www.bioinf.jku.at/publications/older/2604.pdf, 1997.

[Inv17a] Investiopedia. 51% attack. https://www.investopedia.com/terms/1/51-attack.asp, 2017.

[Inv17b] Investiopedia. Initial coin offering. https://www.investopedia.com/terms/t/trillion-dollar-coin.asp, 2017.

[IOQ14] Sutskever Ilya, Vinyals Oriol, and V. Le Quoc. Sequence to sequence learning with neural networks. https://arxiv.org/abs/1409.3215, 2014.

[ISA14] Madan Isaac, Saluja Shaurya, and Zhao Aojia. Automated bitcoin trading via machine learning algorithms. http://cs229.stanford.edu/proj2014/Isaac%20Madan,%20Shaurya%20Saluja,%20Aojia%20Zhao,Automated%20Bitcoin%20Trading%20via%20Machine%20Learning%20Algorithms.pdf, 2014.

[KA14] Simonyan Karen and Zisserman Andrew. Very deep convolutional networks for alrge-scale image recognition. https://arxiv.org/pdf/1409.1556v6.pdf, 2014.

[KXSJ15] He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. Deep residual learning for image recognition. https://arxiv.org/abs/1512.03385, 2015.

[Mad17] Paul Madore. Status ico clogs up ethereum blockchain. https://www.cryptocoinsnews.com/status-ico-clogs-up-ethereum-blockchain/, 2017.

[MKA97] Schuster Mike, Paliwal Kuldip, and General A. Bidirectional recurrent neural networks. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.331.9441, 1997.

[MQY14] Lin Min, Chen Qiang, and Shuicheng Yan. Network in network. https://arxiv.org/abs/1312.4400, 2014.

[Ng] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. http://www.andrewng.org/portfolio/feature-selection-l1-vs-l2-regularization-and-rotational-invariance/.

[NSS14] Alex Krizhevsky Ilya Sutskever Nitish Srivastava, Geoffrey Hinton and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf, 2014.

[PG17] Ross Girshick Pieter Noordhuis Lukasz Wesolowski Aapo Kyrola Andrew Tulloch Yangqing Jia Kaiming He Priya Goyal, Piotr Dollar. Accurate, large minibatch sgd: Training imagenet in 1 hour. https://arxiv.org/pdf/1706.02677.pdf, 2017.

[RKJ15] Kumar Srivastava Rupesh, Greff Klaus, and Schmidhuber Jürgen. Highway networks. https://arxiv.org/abs/1505.00387, 2015.

[Rob17] Jeff John Roberts. Hoax over 'dead' ethereum founder. http://fortune.com/2017/06/26/vitalik-death/, 2017.

[Ré17] Philippe Rémy. Deep learning bitcoin. https://github.com/philipperemy/deep-learning-bitcoin, 2017.

[SRP+17] Xie Saining, Girshick Ross, Dollár Piotr, Tu Zhuowen, and He Kaiming. Aggregated residual transformations for deep neural networks. https://arxiv.org/abs/1611.05431, 2017.

[YYG15] LeCun Yann, Bengio Yoshua, and Hinton Geoffrey. Deep learning. http://pages.cs.wisc.edu/~dyer/cs540/handouts/deep-learning-nature2015.pdf, 2015.