ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Master Thesis Econometrics and Management Science

Active Trading In Cryptocurrency Markets: Momentum Vs. Machine Learning

Hans C. Becker

433128

Supervisor: Prof. Dr. Michel van der Wel
Second assessor: Dr. Anastasija Tetereva

Company supervisors VB Risk Advisory:
Koen van Wissen
Diederick Venekamp

Date final version: January 31, 2022

Abstract

In recent years, automated cryptocurrency trading has become very popular and there is a large universe of possible trading strategies available. Direct reinforcement learning seems to be well suited to trading cryptocurrencies and has been shown to outperform a naive buy-and-hold benchmark (Koker & Koutmos, 2020). This research investigates whether direct reinforcement learning can also outperform a less naive benchmark by comparing it against technical momentum strategies (Rohrbach, Suremann, & Osterrieder, 2017). We find that the machine learning strategies outperform technical strategies in most cases for most common performance measures, such as Sharpe ratio, Sortino ratio, cumulative returns and maximum drawdown. The number of trades made by machine learning strategies is higher than for the technical strategies. We introduce a validation strategy for learning the direct reinforcement model that mitigates overfitting and we perform an analysis of the influence of different hyperparameters such as learning rate, the number of lags and the performance function used for training on the performance of the machine learning strategies. We find favorable parameters, although a grid search to determine the optimal parameters is needed. We consider an online direct reinforcement strategy, but this is not able to outperform technical strategies.

Contents

1 Introduction

Due to high availability of data and ease of access to markets, automated cryptocurrency trading strategies have become quite popular. Some are quite old and well known, like technical momentum trading strategies. Some are more sophisticated and require more computation to arrive at a buy or a sell signal, like quantitative strategies that rely on modelling. Cryptocurrency markets are different to equity or foreign exchange (FX) markets in a number of ways. Examples are 24/7 trading, large arbitrage opportunities across exchanges (Makarov & Shoar, 2020) and a lack of oversight or regulation. Trading strategies that work for equity markets might not perform well in cryptocurrency markets because of these differences.

In this research we look at direct reinforcement learning (DRL) for trading cryptocurrencies and comparing it's performance to technical trading strategies (Rohrbach et al., 2017). We do this comparison for four of the largest cryptocurrencies in circulation, namely Bitcoin, Litecoin, Ethereum and Neo. DRL was first developed by Moody, Wu, Liao, and Saffell (1998) and first applied to cryptocurrency trading by Koker and Koutmos (2020). Koker and Koutmos (2020) show that it's possible to outperform a naive buy-and-hold strategy with a DRL approach. Cryptocurrencies experience large drawdowns, so improving upon the benchmark buy-and-hold strategy by short selling cryptocurrencies during drawdowns could possibly be a simple way to improve performance. We therefore compare DRL to a less naive benchmark to determine whether all of the extra computation is justified. The question we try to answer in this research is: is direct reinforcement learning able to outperform simple technical strategies in cryptocurrency markets? The benchmark we use is based on momentum, which relies on the notion that when a trend forms, it is likely to persist for some time.

The contributions of this paper are as follows. We first investigate whether DRL is able to outperform a more sophisticated benchmark when trading cryptocurrencies. Many research into DRL focused on making improvements to the algorithm, leaving out a broader

view of how performance is compared to other approaches. Next, for training the algorithm we introduce a validation strategy that aims to mitigate overfitting. Next to that we also analyse the influence of different hyperparameters on the performance of the DRL strategies, to determine whether outperformance can be consistently achieved. Koker and Koutmos (2020) and earlier research into DRL provided little insight into the influence of hyperparameters on the performance of the model, especially since new machine learning techniques for optimization were introduced. Lastly we investigate an online learning version of the DRL model, which is able to continually adapt as new data comes in. This online version was first introduced in Moody and Saffell (2001) and required analytical derivation of the gradient. With new machine learning techniques analytical derivation is not needed and more goal functions for optimizing the model can be implemented.

The technical strategies use a short and a long exponential moving-average. When the long moving-average is lower than the short moving-average a downward trend is identified, and an upward trend if the long moving-average is above the short moving-average. The strategy sells (short) into a downward trend and buys (long) into an upward trend. The exponential moving-average uses a smoothing ratio that determines the weight of incoming price data. This smoothing ratio can be adjusted for the moving-average to follow the price more or less closely. Multiple smoothing ratios are investigated in this paper. Rohrbach et al. (2017) apply the strategies to foreign currency (FX) trading and Bitcoin trading. They conclude that cryptocurrencies exhibit a lot of momentum and it is possible to form profitable trading strategies. Maximum drawdowns are large but since returns are high the risk adjusted returns are reasonable. The strategies they implement are described in Baz, Granger, Harvey, Le Roux, and Rattray (2015).

The direct reinforcement learning strategies aim to optimize a performance function on a training dataset over a number of iterations (epochs). For each epoch, the gradient of the model parameters are taken with respect to the performance function and the model parameters are updated with the gradient to perform gradient ascent in the direction of a

higher performance function value. This model has multiple hyperparameters of interest, such as the number of return lags used to determine a signal, the learning rate used for updating the parameters and the performance function used. Hyperparameters do not directly influence the trading decisions, but determine the functional form of the model, which does influence the trading decisions. The hyperparameter analysis is an extension of the work of Koker and Koutmos (2020) who were the first to apply DRL to cryptocurrency trading and used fixed hyperparameters for their analysis. I also investigate the performance of so called online learning strategies. Whereas a standard DRL algorithm is trained, after which it is left to trade for a certain period, an online strategy updates the parameters every time new price data comes in. This give the model the possibility to react to changing markets, possibly improving its performance.

This paper finds that in most cases, the direct reinforcement learning strategies are able to outperform the technical strategies, however quantitative strategies perform more trades. The extra trading done by quantitative strategies is detrimental to performance when taking transaction costs into account. The best performing technical strategies use a smoothing ratio that favors closely following the price, which seems to work better in volatile cryptocurrency markets. Of the three performance functions investigated for training the quantitative strategies, the performance varies across coins. When looking at the hyperparameters of the quantitative strategies, this research finds several things. The first is that a lower learning rate is usually better for model performance. The second is that models with a higher number of lags perform better than models with less lags. We find that doing a grid search over the lags to determine the best optimal number of lags can improve model performance significantly. The online strategies are not able to outperform the standard DRL strategies. We find that optimizing the in sample performance function of the online strategies results in poor performance as using only the in sample performance function value tends to overfit the model, leading to poor trading performance.

The remainder of this paper is structured as follows. Section 2 reviews some of the lit-

erature underlying the DRL algorithm. Section 3 describes the data used for this research. Section 4 describes the methods used for technical and quantitative signal generation. Section 5 discusses the results and provides a comparison between the different methods. Section 6 draws conclusions from the results and provides paths for future extensions of the direct reinforcement learning framework for trading.

2 Literature review

The following section outlines the development of the DRL algorithm implemented in this research. The methods of Koker and Koutmos (2020) are based on the work of Moody and Saffell (2001). They compare the two most important forms of reinforcement learning, namely direct reinforcement learning and value function learning. Moody and Saffell (2001) show that direct reinforcement learning is better suited to trading, as this approach does not require any forecasting which is notoriously difficult for financial returns. The idea of direct reinforcement learning is developed in Moody and Wu (1997) and applied to trading in Moody et al. (1998). Their approach proposes to train the model based on a performance function which is optimized by gradient ascent. The gradient of the performance function with respect to the system parameters is derived explicitly for the Sharpe ratio. They also derive a gradient for online learning, that can be used to update the system parameters when a single return materializes. With online learning, the system is able to adapt itself constantly to new market data. A comparison of the direct reinforcement approach of Moody et al. (1998) by Moody and Saffell (2001) to a value learning approach finds that the direct reinforcement approach outperforms value function learning in devising trading strategies.

The work of Moody et al. (1998) was extended in a number of ways in the following years. Gold (2003) analysed the influence of updating of parameters on the performance of the model proposed in Moody et al. (1998) for FX trading. They found that periodically updating the parameters by using a moving window for training improves the models per-

formance. Dempster and Leemans (2006) looked to further extend the work of Moody et al. (1998) by adding a risk management layer and a dynamic optimization layer to the DRL algorithm. The dynamic optimization layer handles optimization of the hyperparameters. These hyperparameters determine the form of the model, and in the case of machine learning models parameters such as the learning rate and transaction cost. The risk management layer prevents large losses when markets suddenly turn and validates trading signals generated by the DRL algorithm. The model is able to outperform the basic version when trading FX. Maringer and Ramtohol (2012) noticed that the DRL approach of Moody et al. (1998) is not capable of handling non-linearities that are present in financial data. They proposed to extend the DRL model with a regime-switching scheme. Two variants are proposed, a threshold variant and a smooth transition variant. The regime-switching extension is shown to outperform the basic DRL model in most cases, but it also tends to trade more leading to higher transaction costs.

Until recently, interest in direct reinforcement learning faded. With deep learning approaches becoming commonplace and the development of machine learning packages interest picked up again. Deng et al. (2016) combine the approach of Moody et al. (1998) with deep learning to improve feature learning of the data. They show that the model of Moody et al. (1998) is equivalent to a single layer neural network. The single layer is extended to have multiple layers for improved feature learning. They apply their approach to stock index and commodity futures contract data and show that the deep learning approach is able to outperform the basic DRL variant. Koker and Koutmos (2020) are the first to apply the work of Moody et al. (1998) to cryptocurrency trading. They use the same decision function as in Moody and Saffell (2001) but use the Sortino ratio as performance function to optimize the model. Instead of explicitly deriving the gradient of the system parameters with respect to the performance function as in Moody et al. (1998) they use the automatic differentiation functionality that is part of the PyTorch package for Python (Paszke et al., 2017). PyTorch uses a dynamic computational graph to derive

the gradient with respect to a certain variable. Using such a technique eliminates the need to derive complicated derivatives and allows for easier implementation of different performance functions. It is also possible to implement an online learning strategy using PyTorch, by calculating the gradient after a single return materializes as is done in this paper.

3 Data

Cryptocurrency exchange application programming interfaces (API's) provide a standard format in which price data is provided. This format is open, high, low, close and volume data. This research uses data from the Binance exchange[1], which is one of the largest exchanges by US Dollar volume and was founded in 2017. Binance is also one of the few exchanges to provide consistent price records for multiple coins. Many exchanges list and delist coins due to various reasons and prices across exchanges can vary due to poor arbitraging. The following 4 cryptocurrencies are considered: Bitcoin (BTC), Ethereum (ETH), Litecoin (LTC) and Neo (NEO). These currencies have high liquidity and volume, and have relatively long price records. The first price record is from September 22nd 2017 00:00, and the last price record is from May 19th 2021 00:00. This gives a total of 30059 hourly time series observations for each cryptocurrency. In case the exchange is down and trading is not possible on the exchange, the last known price is used. This way, none of the time series have missing values.

Figure 1 gives an idea of the price development over the observed time period. Both the 2017/2018 and 2021 cryptocurrency bubbles with exponential growth phases (Kyriazis, Papadamou, & Corbet, 2020) are clearly visible in these graphs. Trends can be identified over different time periods, making it interesting to investigate whether momentum strategies are able to profit off of these trends. The model of Koker and Koutmos (2020) uses normal returns for training and in the trading decision function. In general, for each time

[1] `https://www.cryptodatadownload.com/data/binance/`

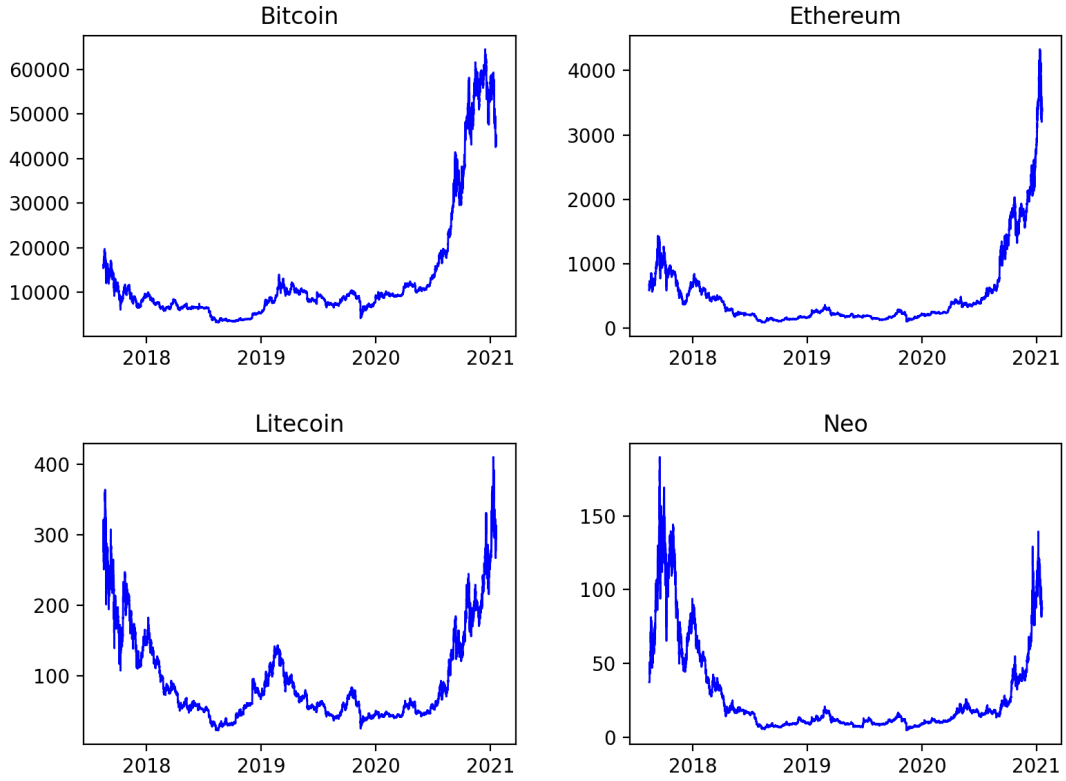


Figure 1: Development of observed close prices of various cryptocurrencies on the Binance exchange. Price is quoted in Dollars per unit of currency.

period t a price z_t is observed. This series of prices is converted to normal returns r_t with

$$r_t = \frac{z_t - z_{t-1}}{z_{t-1}}. \tag{1}$$

Table 1 gives an overview of summary statistics of the different currencies. By looking at the volatility, minimum return and maximum returns we can see that returns are very volatile. Both maximum and minimum hourly returns of this size are large when compared to equities or FX. Further, when looking at kurtosis, we can find that all observed cryptocurrency returns series are heavy tailed. Autocorrelations over the entire sample period

	Bitcoin	Ethereum	Litecoin	Neo
Max price	64577.3	4332.81	190.00	409.84
Min price	3172.05	82.17	4.58	22.75
Volatility	0.00919	0.01121	0.01475	0.01246
Max return	0.174	0.161	0.288	0.201
Min return	-0.182	-0.208	-0.184	-0.191
Skewness	-0.129	-0.142	0.774	0.184
Kurtosis	32.98	21.83	23.63	18.05

Table 1: Summary statistics for cryptocurrencies hourly returns. Price is in U.S. dollars per unit of cryptocurrency. Mean, volatility, min, max, skewness and kurtosis are calculated from normal returns.

are very small except for the first-order autocorrelation.

4 Methodology

The following sections will detail the quantitative and technical approaches to generating trading signals. We use $F_t \in \{-1, 1\}$ to denote a signal, where -1 denotes a sell signal and 1 a buy. A strategy changes position when $F_t \neq F_{t-1}$. Section 4.1 discusses the momentum trading strategies and section 4.2 discusses the DRL strategies and online learning strategies.

To compare performance of the different strategies we use a number of different measures. The most popular measure for quantifying trading performance is the Sharpe ratio, which is given by

$$\text{Sharpe} = \frac{\mu_R}{\sigma_R}, \tag{2}$$

where μ_R stands for the mean of trading returns and σ_R is the standard deviation of trading returns. Another popular measure for comparing trading returns is the Sortino ratio, which is similar to the Sharpe ratio, but only penalizes the mean returns by the

standard deviation of negative returns. The idea here is that volatility of positive returns is preferred over the volatility of negative returns. The Sortino ratio is defined as

$$\text{Sortino} = \frac{\mu_R}{\sigma_{R_{R_t \leq 0}}}. \tag{3}$$

Next to that, every trader is eventually interested in the cumulative returns that can be earned with a certain strategy. The cumulative return is the return, or increase in wealth, a trader achieves over a certain period. Cumulative returns are defined as

$$\text{Cumulative return} = \prod_{t=1}^{T}(1 + R_t) - 1. \tag{4}$$

The last measure we look at is maximum drawdown. Maximum drawdown is defined as

$$\text{Maximum Drawdown} = \max_{0 \leq t_1 \leq t_2 \leq T}(Y_{t_1} - Y_{t_2}). \tag{5}$$

Traders often don't like large drawdowns since these significantly hurt future returns and having smaller drawdowns generally indicates better trading performance.

4.1 Technical signal generation

The methods of Rohrbach et al. (2017) are derived from Baz et al. (2015) and are detailed in the following section. Technical trading rules have the advantage of being computationally less intensive compared to quantitative approaches. The strategy of Rohrbach et al. (2017) depends on the combination of exponential moving-averages (EMA) to generate a trading signal. EMA is a weighted moving-average of prices which puts more or less emphasis on recent price data. I will first investigate whether individual EMA's provide consistent trading returns, after which I will investigate whether a combination of EMA's improves

returns. The EMA is defined as follows

$$EMA_t(z_t, \alpha) = \begin{cases} z_t & \text{if } t = 0 \\ \alpha \cdot z_t + (1 - \alpha) \cdot EMA_{t-1}(z_{t-1}, \alpha), & t > 0, \end{cases}$$

where $\alpha = \frac{1}{n}$ is the exponential smoothing ratio that determines the importance of the new price data, with n being the number of days or hours, depending on the data used. A smaller n leads to a higher weight for incoming data and thus to an EMA that tracks price movement more closely. Taking a smaller n is similar to taking a smaller window for a standard moving-average. The price movement is more closely followed, but volatility is also increased. Since at $t = 0$ we have $EMA_t = z_t$, the strategy requires a 'warmup' period, where a series of z_t is used to initialize EMA_t. This research takes a warmup period of 3000 hours, which is the same length is the training period for the quantitative strategies.

The EMA is converted into a trend by

$$x_{k,t} = EMA\left(z_t, \frac{1}{n_{k,s}}\right) - EMA\left(z_t, \frac{1}{n_{k,l}}\right), \tag{6}$$

where $k = 1, 2, 3$ which identifies which EMA it is and l indicates that it is a long EMA, s a short EMA. Rohrbach et al. (2017) choose $n_{k,s} = (8, 23, 66)$ and $n_{k,l} = (24, 69, 198)$ with $n_{k,s}$ and $n_{k,l}$ in days. This research uses $n_{k,s} = (12, 48, 192)$ and $n_{k,l} = (48, 192, 768)$ with $n_{k,s}$ and $n_{k,l}$ in hours which roughly corresponds to daily, weekly and monthly trends. The choice for $n_{k,l}$ and $n_{k,s}$ is relatively arbitrary as long as there is a sufficient difference between $n_{k,l}$ and $n_{k,s}$ to identify a reliable trend. A common ratio in the literature is to have $n_{k,l}/n_{k,s} = 3$ or 4 (Rohrbach et al., 2017).

A downward trend is identified when the long EMA is larger than the short EMA and vice versa for an upward trend. A graphical example of how this works is provided in figure 2. This research first looks at k different individual trends. The signal when using

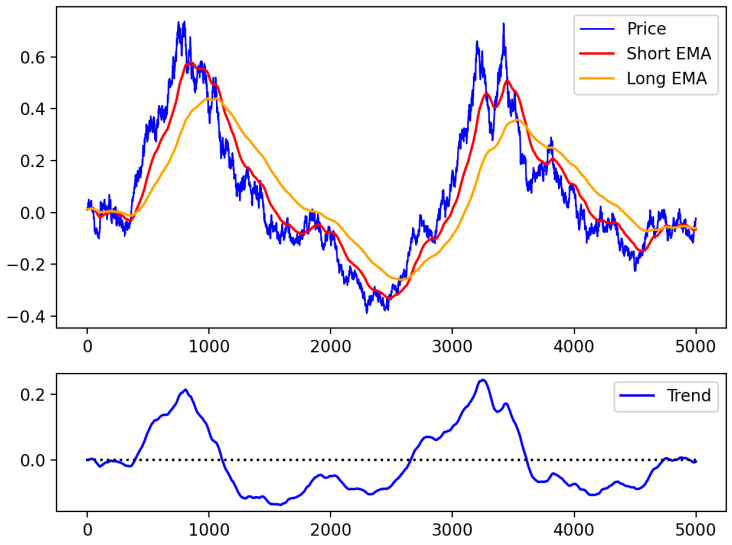


Figure 2: Example of exponential moving-averages used to determine a trend for a random walk. When the long EMA is above the short EMA the trend is positive, and when the short EMA is above the long EMA the trend is negative.

an individual trend is obtained by

$$F_t = \text{sign}(x_{k,t}). \tag{7}$$

Next to that, a linear combination of trends is taken to generate a signal as in Baz et al. (2015). To take a combination of trends, the trends first have to be normalized. This is such that the relative magnitude of a trend is taken into account, as opposed to the absolute magnitude. In case this is not done, one trend can dominate the linear combination, which practically results in the trading signal being dependent on one trend. In Rohrbach et al. (2017) $x_{k,t}$ is normalized to $y_{k,t}$ by

$$y_{k,t} = \frac{x_{k,t} - \mu_{x,k}}{\sigma_{x,k}}, \tag{8}$$

where $\mu_{x,k}$ is the mean of the trend for $k = 1, 2, 3$ and $\sigma_{x,k}$ is the standard deviation of the

trend. This normalization is needed to compare different trends.

Rohrbach et al. (2017) pass $y_{k,t}$ through an impulse response function to obtain the position size. Since we are not interested in position sizing, we can use the normalized trends as is. When combining normalized trends to obtain a signal we use

$$F_t = \text{sign}\left(\sum_{k=1}^{3} w_k \cdot y_{k,t}\right). \tag{9}$$

Rohrbach et al. (2017) use equal weights for w_k which is also adopted in this research.

4.2 Quantitative signal generation

The following section details the direct reinforcement learning model. The trader is always in the market with either a long or a short position. The general representation of the trading signal function is

$$F_t(\theta; I_t). \tag{10}$$

Where θ are the system parameters and weights and I_t is the information set at time t. The decision function used for this research is given by

$$F_t(\theta, I_t) = \text{sign}(\theta_0 r_{t-1} + ... + \theta_m r_{t-m}). \tag{11}$$

Here m stands for the number of autoregressive returns to be used. This decision function differs from Koker and Koutmos (2020) in a number of ways. Koker and Koutmos (2020) contains both an intercept and a lagged decision variable in the decision function. In practice the addition of the lagged decision variable results in poor performance. This is due to the fact that the variable F_t has either value 1 or -1. Compared to a return of for example $1\% = 0.01$ this is rather large. As a consequence F_{t-1} influences F_t a lot. This has led us to drop F_{t-1} from the decision function. The same goes for the intercept θ_0, which tends to disturb learning and results in a model with a very large intercept. Adding

other variables to (11) will also require some form of normalization to ensure none of the variables dominates the trading decision.

The *sign* function is not differentiable, which poses a problem when differentiating the performance function with respect to θ for doing gradient ascent. Therefore the *sign* function is replaced with the differentiable *tanh* function in the training phase. See figure 3 for a comparison of the *tanh* and *sign* functions.

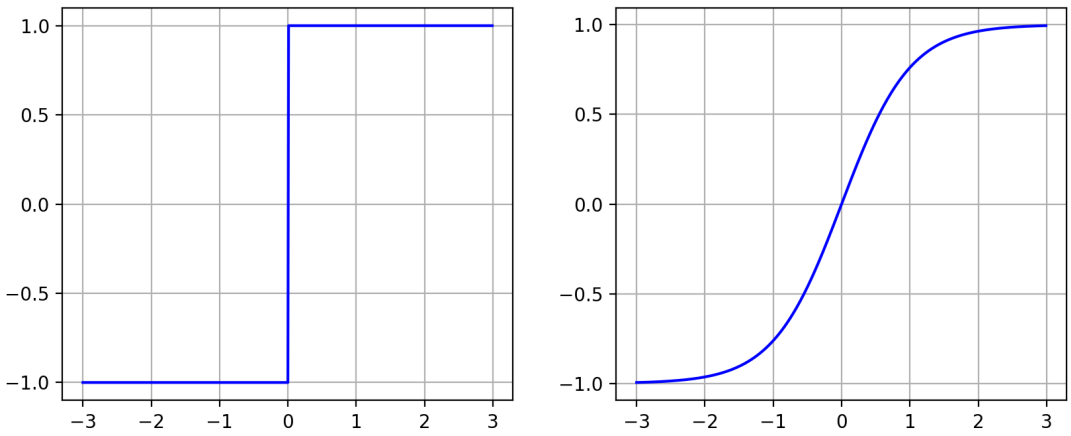


Figure 3: Comparison of the discontinuous *sign* function (left) and the continuous *tanh* function. Both functions map inputs to a value between -1 and 1.

The trading return R_t, the return that is actually earned by the trader, is defined as

$$R_t = (1 + F_{t-1} r_t)(1 - \delta |F_t - F_{t-1}|), \tag{12}$$

where δ represents the transaction cost per unit of currency. Transaction costs are only incurred when $F_t \neq F_{t-1}$, so when the system changes from a long position to a short position or vice versa. Transaction costs are only accounted for during training, Koker and Koutmos (2020) include this to prevent the model from trading excessively. The utility function to be optimized is defined as

$$U(R_1, R_2, ..., R_T; W_0, \theta). \tag{13}$$

Where R_t are the trading returns from time $t = 1, ..., T$ and W_0 stands for the initial wealth. Utility functions can be simple utility functions or can be path dependent, such as the Sharpe ratio (2) which depends on the realized trading returns. Note that the Sharpe ratio depends on F_t and θ through (12). Other goal functions that are analysed in this research are the Sortino ratio (3) and cumulative returns (4).

The following section details the training procedure that is used to set up the model. The trading parameters θ are optimized over N epochs via gradient ascent. An epoch is a single iteration for which the gradient is calculated. The system parameters are updated after each epoch with

$$\Delta\theta = \eta \frac{\mathrm{d}U(\theta)}{\mathrm{d}\theta}. \tag{14}$$

Here η is the learning rate that determines the rate at which the parameters are updated.

A higher learning rate means faster updating, but may mean that the model may not converge to an optimal set op parameters. A lower learning rate may result in the model not converging within N epochs. θ is initialized as a draw from a uniform random variable. Each epoch performs a forward pass of the data through the system to compute trading signals from $t = 0, ..., T$. The gradient $\frac{\mathrm{dU}(\theta)}{\mathrm{d}\theta}$ is computed automatically via PyTorch (Paszke et al., 2017). The full training procedure is specified in algorithm 1. We use a validation scheme, where we split the training data set into a training set and a validation set. The training dataset is used to actually train the model by optimizing the performance function. The value of the performance function is also calculated on the validation set and the parameters θ with the highest validation performance value is used as the final model. This is to prevent overfitting and improve out of sample trading performance.

After optimization of the parameters, the system can start trading. The learning rate η, transaction costs δ, number of lags m, number of epochs N and goal function $U(\cdot)$ are all hyperparameters of the system that influence the performance but are set for a single model. Multiple configurations of these hyperparameters are tried to determine the influence of these parameters to aid in looking for future improvements to the DRL

Algorithm 1: Direct reinforcement learning algorithm

input : A series of returns r_t from $t = 1, ..., T$ with the test set running from $t = 1, ..., S$ and the validation set running from $t = S + 1, ..., T$, learning rate η, number of epochs N, goal function $U(\theta)$ and transaction cost δ

output: System parameters θ

$\theta \leftarrow$ random vector drawn from uniform;

$\theta_{opt} \leftarrow 0$;

$\text{goal}_{opt} \leftarrow$ - ∞ ;

$F_0 \leftarrow 0$;

for $epoch \leftarrow 1$ **to** N **do**

`// Calculate the trading returns for the training sample`

for $t \leftarrow 1$ **to** S **do**

$F_t \leftarrow \tanh(\theta_0 r_{t-1} + ... + \theta_m r_{t-m})$;

$R_t \leftarrow (1 + F_{t-1} r_t)(1 - \delta |F_t - F_{t-1}|)$;

end

$U_{epoch} \leftarrow U(R_1, ..., R_T)$;

Compute gradient $\frac{\mathrm{d}U_{epoch}(\theta)}{\mathrm{d}\theta}$;

$\theta \leftarrow \theta + \eta \frac{\mathrm{d}U_{epoch}(\theta)}{\mathrm{d}\theta}$;

`// Calculate the trading returns for the validation sample`

for $t \leftarrow S + 1$ **to** T **do**

$F_t \leftarrow \text{sign}(\theta_0 r_{t-1} + ... + \theta_m r_{t-m})$;

$R_t \leftarrow (1 + F_{t-1} r_t)(1 - |F_t - F_{t-1}|)$

end

`// Save the best parameters if the validation goal value has improved`

$\text{goal} \leftarrow U(R_{S+1}, ..., R_T)$;

if $goal > goal_{opt}$ **then**

$\text{goal}_{opt} \leftarrow \text{goal}$;

$\theta_{opt} \leftarrow \theta$;

end

end

algorithm.

As mentioned it is also possible to update the system parameters while the system is trading, which is called online learning. This procedure first trains the model in the same manner as algorithm 1 to initialize the model. This provides the initial system parameters θ_0. Then for each time a return materializes, we are interested in finding the gradient of θ_t with respect to the goal function, and subsequently updating θ_t to obtain θ_{t+1}. This gradient updating is done by running algorithm 1 by setting θ to θ_t and F_0 to F_t and the number of epochs N to 1. This performs a single forward pass on the data to obtain a value for the goal function. The gradient is then computed, with which θ_t is updated to θ_{t+1}. The online learning strategy does not use the validation strategy when it performs a single forward pass. It calculates the value of the performance function on the training data set, calculates the gradient and updates θ after which it continues trading. It uses the last 1000 observations as the training dataset.

5 Results

Results for all strategies for the different cryptocurrencies are provided in tables 2 and 3. The first 3000 observations are used for training and validation. The first 2000 observations, from 03:00 13 December 2017 to 11:00 6 March 2018, are used to train the DRL and DRL-Online (DRLO) strategies, and the next 1000 observations, from 12:00 6 March 2018 to 03:00 7 April, are used as a validation sample. The EMA strategies use the first 3000 observations as the warmup period. The DRL(O) strategies use a moving window and were retrained after 1000 hours of trading has passed. This way, all of the obtained trading returns are completely out of sample. When retraining the last 3000 observations where used, with the same train/validation split for the training of the model. For training, the number of epochs is set to $N = 150$, the learning rate is set to $\eta = 0.05$ and transaction costs is $\delta = 0.10\%$. The number of lags to be used in the decision function is set to 70.

5.1 Comparison of technical and quantitative strategies

We first look at the results for the backtest which is done over the full sample. Most DRL strategies are able to outperform the EMA strategies and in almost all cases are able to outperform buy-and-hold as in Koker and Koutmos (2020). For Bitcoin, DRL-Totalreturns is not able to outperform the best performing EMA strategy (EMA-Short) and for Ethereum this is the case for DRL-Sortino (outperformed by EMA-Short). In the case of Litecoin all DRL strategies outperform the best performing EMA strategy. For Neo only DRL-Sortino outperforms the best performing EMA strategy, with DRL-Sharpe having similar performance. A big difference between the EMA and DRL(O) strategies is that the EMA strategies make far less trades. On most cryptocurrency exchanges trading fees are small, but traders might choose to use an EMA strategy over a DRL(O) strategy when taking this into consideration. Both EMA and DRL strategies are able to reduce maximum drawdowns compared to the buy-and-hold strategy.

When looking at EMA strategies we see that EMA-Short is generally the best performing EMA strategy, being the best performing strategy except in the case of Litecoin, where this is EMA-Medium. It seems that a smaller n is beneficial for trading cryptocurrencies. This may be caused by cryptocurrencies having high volatility and sharp drops, which is a problem for strategies with a high n that are not able to adequately adapt. The EMA-Combination strategy performs worst of all EMA strategies. The equal weights are a naive approach to combining trends, and testing different weights might improve its performance. Weighting can depend on the trends volatilty for example.

BTC	BH	EMA-Short	EMA-Medium	EMA-Long	EMA-Combination	DRL-Totalreturns
Cumulative returns	289.41%	1262.76%	219.30%	338.27%	105.87%	646.00%
Standard deviation	0.00811469	0.00811412	0.00792128	0.00795693	0.00811489	0.00777851
Sharpe ratio	0.9394	1.4533	0.8787	1.0397	0.6772	1.2274
Sortino ratio	1.1327	1.7937	1.0423	1.2510	0.8198	1.4823
Max drawdown	71.12%	46.70%	62.36%	68.21%	62.92%	63.87%
Number of trades	1	311	79	20	100	11672
BTC	DRL-Sharpe	DRL-Sortino	DRLO-Totalreturns	DRLO-Sharpe	DRLO-Sortino	
Cumulative returns	2594.73%	2123.58%	171.06%	600.56%	-34.91%	
Standard deviation	0.00777778	0.00777791	0.00777890	0.00777855	0.00777917	
Sharpe ratio	1.7762	1.6919	0.7909	1.1962	0.1797	
Sortino ratio	2.2003	2.1498	0.9898	1.5464	0.2154	
Max drawdown	53.15%	54.63%	65.16%	62.39%	70.27%	
Number of trades	10598	10608	12262	13914	14070	
ETH	BH	EMA-Short	EMA-Medium	EMA-Long	EMA-Combination	DRL-Totalreturns
Cumulative returns	303.58%	2054.59%	889.43%	1521.73%	32.43%	18783.48%
Standard deviation	0.0101976	0.0101968	0.0100654	0.0100384	0.0101979	0.0098825
Sharpe ratio	0.9352	1.4819	1.2600	1.4704	0.5696	2.2313
Sortino ratio	1.1719	1.9044	1.5618	1.7608	0.7196	2.8560
Max drawdown	90.21%	63.16%	65.41%	77.41%	92.93%	59.86%
Number of trades	1	312	74	7	85	7927
ETH	DRL-Sharpe	DRL-Sortino	DRLO-Totalreturns	DRLO-Sharpe	DRLO-Sortino	
Cumulative returns	34883.07%	593.13%	24392.21%	367.58%	96.29%	
Standard deviation	0.0098820	0.0098846	0.0098823	0.0098848	0.0098850	
Sharpe ratio	2.4390	1.1165	2.3183	0.9842	0.6916	
Sortino ratio	3.1485	1.3939	2.9927	1.2306	0.8593	
Max drawdown	42.35%	76.61%	49.77%	64.53%	86.78%	
Number of trades	9249	10473	11441	13667	13979	

Table 2: Results for different strategies for Bitcoin and Ethereum.

LTC	BH	EMA-Short	EMA-Medium	EMA-Long	EMA-Combination	DRL-Totalreturns
Cumulative returns	43.55%	130.34%	571.00%	510.89%	-31.54%	2669.83%
Standard deviation	0.0112562	0.0112560	0.0111596	0.0111997	0.0112563	0.0109583
Sharpe ratio	0.6349	0.7744	1.1133	1.1055	0.4157	1.5252
Sortino ratio	0.8263	1.0043	1.4166	1.4140	0.5264	1.9448
Max drawdown	88.78%	65.45%	70.89%	77.19%	81.91%	0.6762490772
Number of trades	1	364	69	9	137	13686
LTC	DRL-Sharpe	DRL-Sortino	DRLO-Totalreturns	DRLO-Sharpe	DRLO-Sortino	
Cumulative returns	1173.51%	651.43%	1750.32%	4.66%	1129.71%	
Standard deviation	0.0109587	0.0109589	0.0109585	0.0109596	0.0109587	
Sharpe ratio	1.2899	1.1290	1.4027	0.5280	1.2772	
Sortino ratio	1.5817	1.3931	1.7759	0.6578	1.6258	
Max drawdown	71.76%	64.79%	83.59%	88.72%	87.62%	
Number of trades	10542	12094	12336	13808	13902	
NEO	BH	EMA-Short	EMA-Medium	EMA-Long	EMA-Combination	DRL-Totalreturns
Cumulative returns	-19.05%	2157.81%	1479.15%	36.28%	-36.40%	-47.25%
Standard deviation	0.0129671	0.0129658	0.0127362	0.0127238	0.0129671	0.0125084
Sharpe ratio	0.5513	1.4078	1.3455	0.6831	0.4912	0.4151
Sortino ratio	0.7444	1.8796	1.7664	0.8886	0.6377	0.5389
Max drawdown	95.93%	66.18%	66.11%	93.08%	92.01%	94.60%
Number of trades	1	328	63	17	140	14424
NEO	DRL-Sharpe	DRL-Sortino	DRLO-Totalreturns	DRLO-Sharpe	DRLO-Sortino	
Cumulative returns	2090.11%	63408.24%	43.53%	-66.17%	-17.34%	
Standard deviation	0.0125071	0.0125047	0.0125082	0.0125084	0.0125083	
Sharpe ratio	1.4097	2.3078	0.6833	0.2978	0.5349	
Sortino ratio	1.8102	2.9808	0.8732	0.3773	0.6900	
Max drawdown	69.06%	55.12%	95.23%	98.19%	90.05%	
Number of trades	9083	12231	14547	13837	13713	

Table 3: Results for different strategies for Litecoin and Neo.

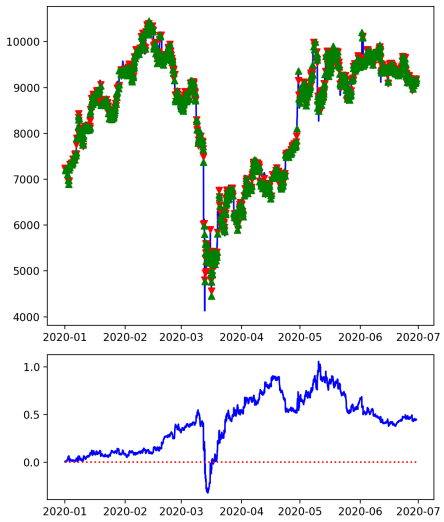
What stands out is that performance for the DRLO algorithms is worse than that of the DRL algorithms. The DRLO algorithms perform a single forward pass on the data each time a new price materializes and update the parameters θ with respect to the gradient of the goal value in the training set. This research shows in the hyperparameters analysis that optimizing with respect to the goal value in the training set will not always result in optimal parameters and easily leads to overfitting. This is likely the case for the DRLO algorithms as these update the parameters for 1000 steps, after which the model is retrained with a validation set. The 1000 steps between retraining will move θ towards a higher goal function value in the test set, which is resulting in overfitting and therefore poorer trading performance.

5.2 Performance functions DRL

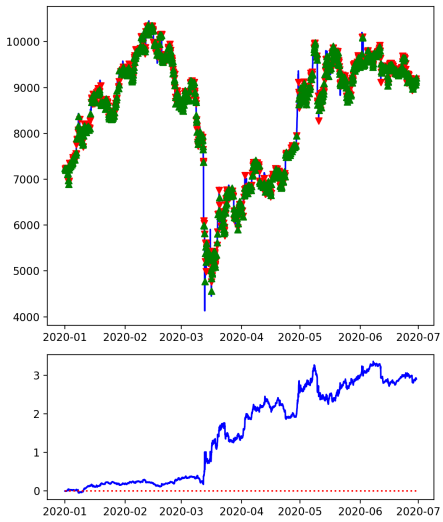
Next we look at the performance of different performance functions across coins. We see that the strategy training on a certain goal function does not always have the highest corresponding performance measure. DRL-Sharpe which is trained using the Sharpe ratio does perform a little better compared to DRL-Sortino in terms of cumulative returns across the different coins, where DRL-Totalreturns usually performs worst in terms of cumulative returns. Which goal function to choose while training the algorithm differs across coins and is therefore best to choose using a backtest to evaluate the performance.

Now we look at the performance while the strategies are trading, to get a better insight into the behaviour of the different strategies. Figure 4 shows the performance of the different DRL strategies for a subset of the data for Bitcoin, running from January 1st 2020 to June 30th 2020. This period is chosen as this period contains both up and down trends, which make for a fair comparison with the EMA strategies. What immediately stands out is the number of trades made by all of the strategies. On average, the strategies change position every 2 to 3 hours. Both the DRL-Sharpe and DRL-Sortino algorithms perform very well during this period, choosing to be short during a sharp drop in price.

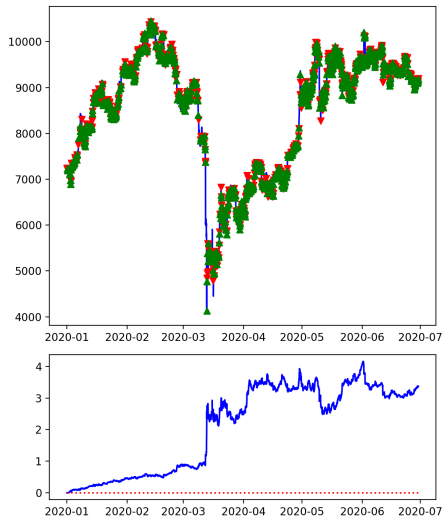
The DRL-Totalreturns algorithm does not perform as well during this period, but is able to recover after some time. It does lag behind in



(a) DRL-Totalreturns

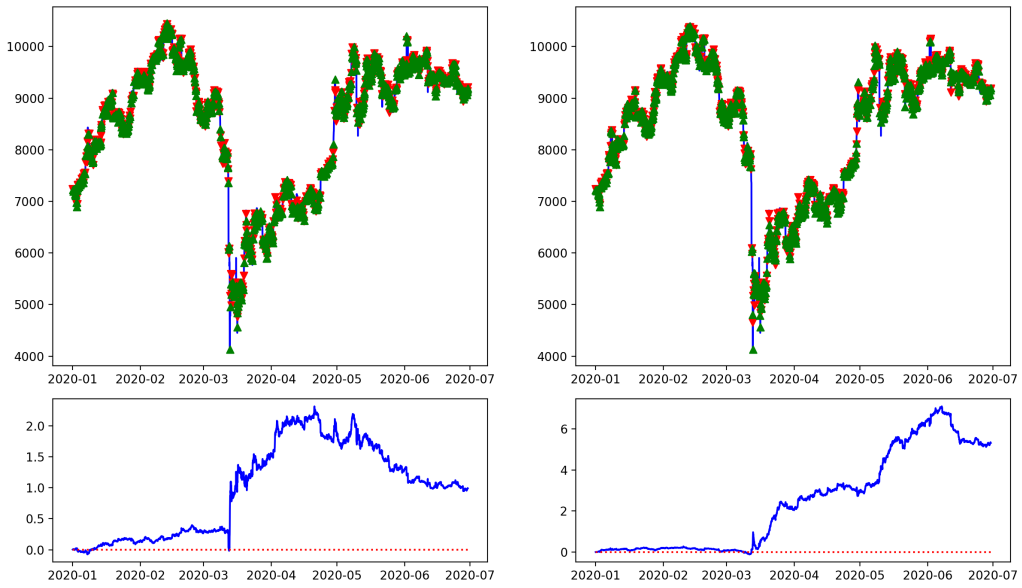


(b) DRL-Sharpe



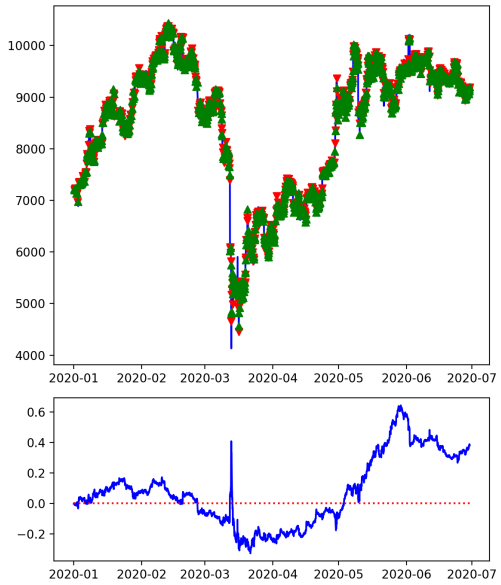
(c) DRL-Sortino

Figure 4: Trades for the DRL strategies during January 2020 until June 2020 and the cumulative return for each strategy



(a) DRLO-Totalreturns

(b) DRLO-Sharpe



(c) DRLO-Sortino

Figure 5: Trades for the DRLO strategies during January 2020 until June 2020 and the cumulative return for each strategy

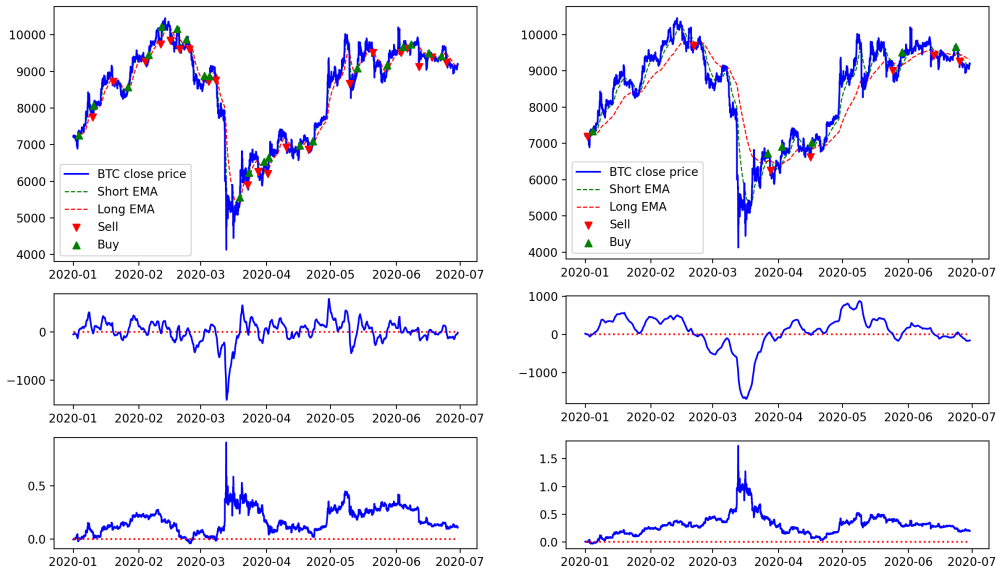
terms of cumulative returns as it sits at 50% after six months of trading, whereas the DRL-Sharpe and DRL-Sortino strategies are able to obtain a cumulative return of around 300% during this same period.

Figure 5 shows the performance for the different DRLO strategies over the period January 1st 2020 to June 30th 2020. DRLO-Totalretrurns and DRLO-Sharpe perform better over this specific period compared to the DRL strategies, whereas DRLO-Sortino performs a lot worse. What stands out from figure 5b is that the strategy is not able to generate any significant returns, but after a sharp drop in price, the algorithm is suddenly able to generate returns. Similar but less pronounced patterns are visible in figure 5a and 5c.

5.3 EMA strategy performance

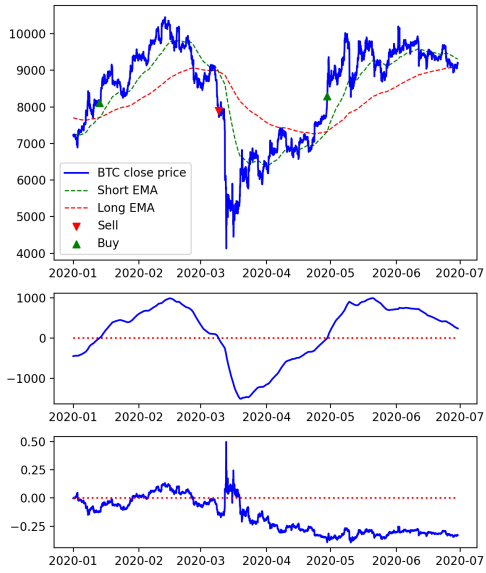
Next we take a look at the behaviour of the EMA strategies over the same time period. Figure 6 shows the trades made by the EMA algorithms, together with the short and long EMA that determine the trend, the trend and the cumulative returns. These graphs give a clear insight into the working of the EMA algorithms, and the influence of the choice for n. A smaller n, as in the EMA-Short strategy, puts more emphasis on the incoming price data, resulting in the EMA tracking the price more closely. The influence is clearly visible when comparing the different graphs. In figure 6a the EMA tracks the price very closely, compared to the EMA in figure 6c where a large n is used for the short and long EMA's.

In general, the EMA strategies are able to identify trends, but are not able to react quickly when the market turns. This can be seen in the period between March 2020 and April 2020, where a sharp downward move occurs. All of the EMA strategies are short during this downfall, but it takes a while before any of the strategies switches to a long position when the market reverses. The EMA-Short strategy is best suited for dealing with this, which suggests using a smaller n when choosing for an EMA strategy.

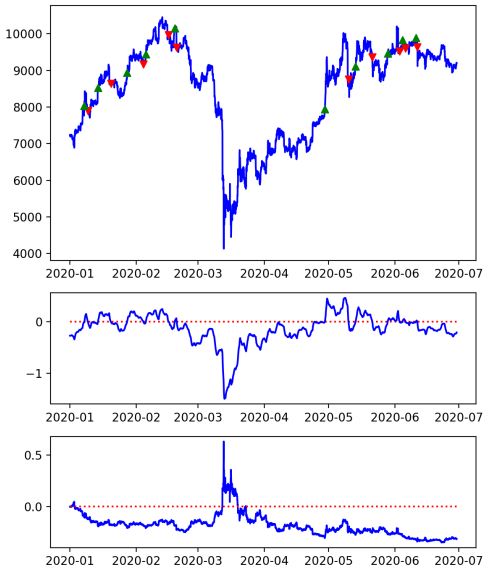


(a) EMA1

(b) EMA2



(c) EMA3



(d) EMA4

Figure 6: Trades for the EMA algorithms. The second graph for each panel shows the trend of each algorithm. The third graph in each panel is the cumulative return for each strategy.

When comparing the trends across the different EMA strategies, we can see that the trend becomes less noisy when using a larger n. This however prevents the strategy from quickly switching positions when a market turns. For the period shown, only the EMA-Short and EMA-Medium strategies are able to end with a positive cumulative return. The EMA-Combination strategy performs similarly to the EMA-Long strategy, eventually losing money. The performance in terms of cumulative returns is lower during this period compared to the DRL algorithms.

5.4 DRL Hyperparameters

Next we look at the influence of the different hyperparameters on the performance of the DRL algorithm. The hyperparameters of interest are the number of lags m, the learning rate η. For this section we use a subset of the Bitcoin time series, which is shown in graph 7 and runs from April 1st 2020 to August 1st 2020. This subset does not experience exponential growth, such as at the end of 2021, thus it is a fair representation of the general market. This timeseries contains 3000 observations, of which the first 2000 are used for training and the last 1000 are used as a validation set. Trading returns are calculated with the training model for the last 1000 observations, to assess the out of sample performance of the model.

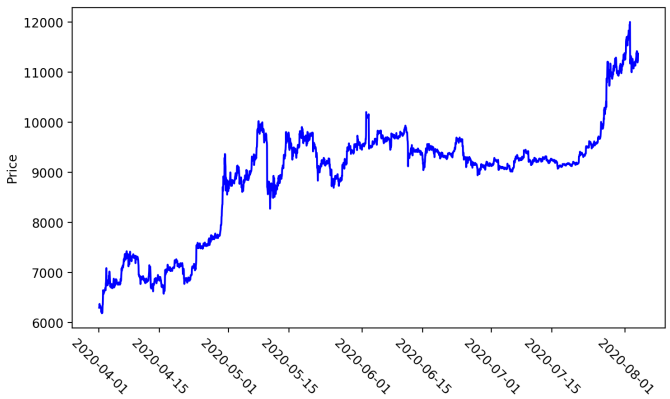
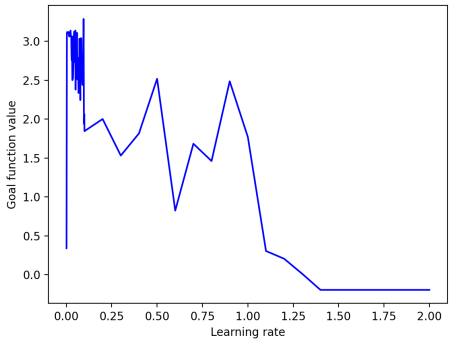


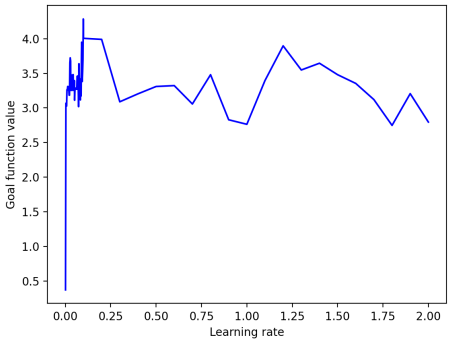
Figure 7: Graph of Bitcoin close price for the selected period

5.4.1 Learning rate

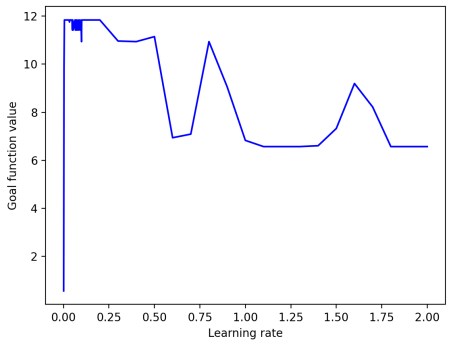
For analysing the influence of the learning rate we estimated and ran the model for 49, 50 and 51 lags. We set the learning rates to 0.001 to 0.01 with steps of 0.001, 0.01 to 0.1 with steps of 0.01 and 0.1 to 2 with steps of 0.1. In most machine learning applications a lower learning rate is preferred over a higher one, to prevent the model from not converging to an optimal value, this is why we choose to analyze more smaller learning rates. We are interested in the performance function value for the validation set, which is



(a) Sharpe ratio



(b) Sortino ratio



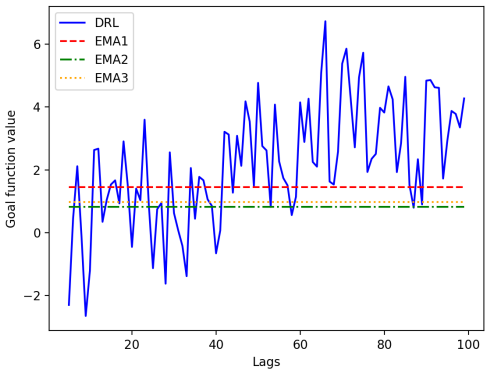
(c) Total returns

Figure 8: Performance function value for different learning rates for the different performance functions. The goal value is computed for the validation sample for learning rates 0.001 to 0.009 with steps of 0.001, 0.01 to 0.09 with steps of 0.01 and from 0.1 to 2 with steps of 2.

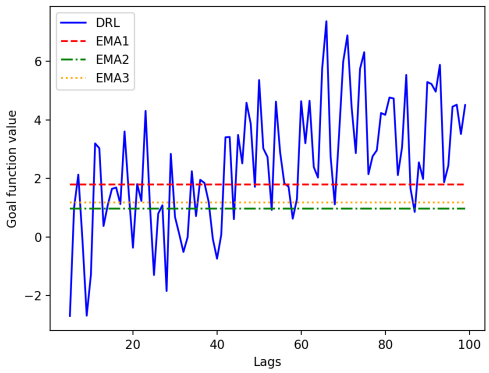
plotted for the three different performance functions in figure 8. There are a few things that stand out in these graphs. The first is that a learning rate between 0.005 and 0.25 is usually a safe bet. Even though when using the Sortino ratio as performance function a lower learning rate is not optimal, the benefit of choosing a higher learning rate is small. In the case of the Sharpe ratio the validation performance value decreases as the learning rate increases. For the model optimized with total returns the performance function value declines slightly after 0.5, but not as sharp as for the Sharpe ratio. What also stands out is the sharp drop in performance function value at the lower end of the learning rate range. This indicates that the learning rate is too small for the model to converge, or that it is so small that it ends up in a local

5.4.2 Number of lags

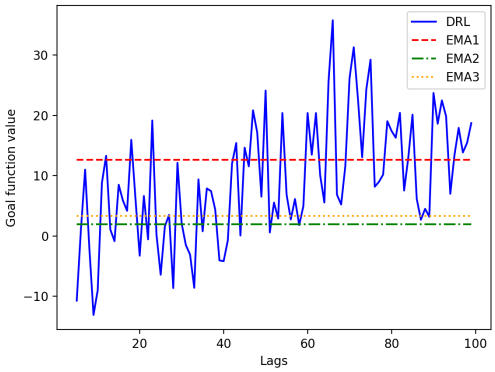
For analysing the effect of the number of lags on the models performance I ran the model with 5 to 100 lags and computed the goal function value in the validation set, with the learning rate set to 0.005, the number of epochs to 150 and the transaction cost set to 0.10%. A graph is presented in figure 9. What stands out is the similarity of the graphs, which indicates that the performance function does not make a large difference for performance. Next to that, the performance function value increases as the number of lags increases, with the optimal number of lags being between 60 and 80 for the selected dataset. The corresponding performance measure for the EMA strategies is also included in the graphs, and show for which lags DRL performs better or worse. From this set of graphs we can conclude that a grid search for the optimal number of lags can be beneficial when training the model. Given the erratic influence of the number of lags on the validation performance function, misspecifying the number of lags by 1 can have a large effect on the performance of the model. Also searching for the optimal number of lags each time the model retrains may improve performance of the DRL algorithms



(a) Sharpe ratio as goal function for the DRL algorithm



(b) Sortino ratio as goal function for the DRL algorithm



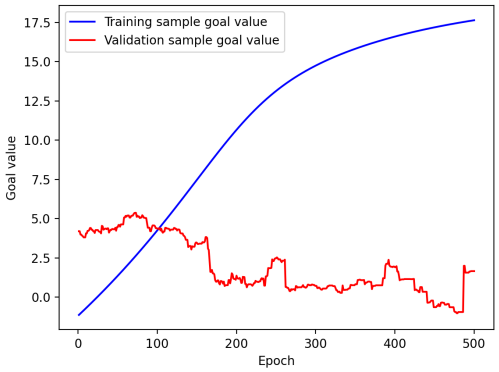
(c) Total returns as goal function for the DRL algorithm

Figure 9: Performance function value for different lags for the different performance functions. The performance function is computed for the validation sample for lags from 5 to 100. Included are the corresponding values of the performance functions in the validation sample for the EMA1, EMA2 and EMA3 strategies.

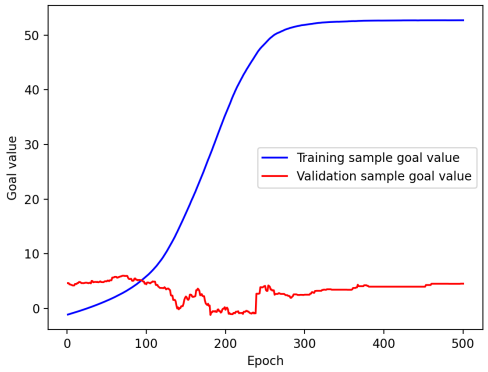
5.5 Convergence of DRL

Lastly we look at the converge of the DRL algorithm. Figure 10 shows the convergence of the model for different performance functions, using a learning rate of 0.005, except in panel 10d with a learning rate of 0.5, and the number of lags set to 70. The blue lines

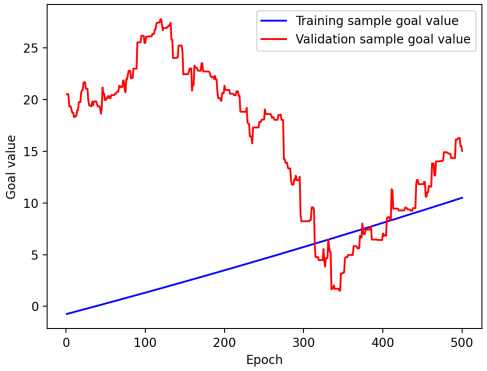
represent the performance function value on the training set and the red lines represent the performance function value on the validation set. The first different between the blue and red lines is the smoothness of the lines. The training sample goal value is computed using a smooth *tanh* function, compared to a discontinuous *sign* function for the validation goal value (graph 3). The discontinuous *sign* function causes jumps in the performance function when the model makes different decisions.



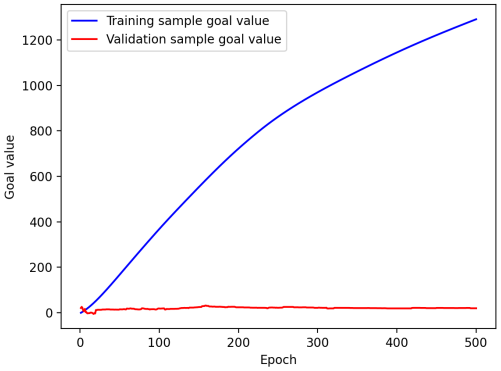
(a) Sharpe ratio as performance function



(b) Sortino ratio as performance function



(c) Total returns as performance function with learning rate of 0.005

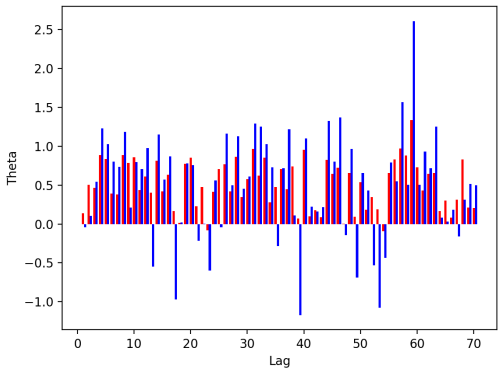


(d) Total returns as performance function with learning rate of 0.5

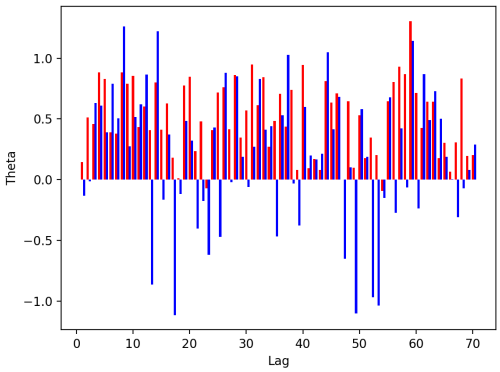
Figure 10: This graph shows the converge of the DRL algorithm for the training sample and the validation sample. The number of lags used is 70 and the learning rate is set to 0.005, unless states otherwise. The number of epochs is set to 500

The *tanh* function maps the decision function to a continuous interval between -1 and 1, and a small change in θ will also result in a small change in the goal function.

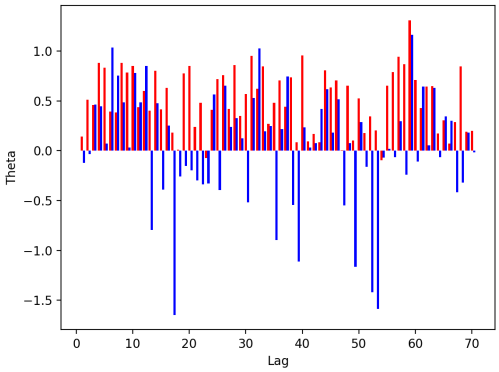
Both the models trained on the Sharpe and Sortino functions, graphs 10a and 10b, converge after a while. The highest value in the validation set is reached within 100 epochs which tells us that training the model any longer than 100 epochs will only lead to overfitting. This encourages the use of the validation strategy, which saves the parameters with the highest performance function value for the validation test.



(a) DRL-Totalreturns



(b) DRL-Sharpe



(c) DRL-Sortino

Figure 11: A comparison of the value of θ for the optimal value in the validation sample (red) and the optimal value in the training sample (blue). Each bar represents the coefficient for that lag. The number of lags is set to 70, and the learning rate to 0.005.

The model trained with the total returns as performance function does not converge as quickly. To investigate when and if it converges we set the learning rate to 0.5 and trained the model again, as can be seen in figure 10d. In this case the model does not converge yet, whereas the performance goal function already reached its maximum value around the 100th epoch. In theory the goal function value could reach very high values, when it makes the correct decision for each trade and only makes profitable trades. In practice this of course never happens, which is also shown by the validation goal function peaking earlier.

Figure 11 shows the value of each parameter in θ. In red is the value of the parameters in the validation sample with the highest goal value and in blue the value of the parameters after 500 epochs of training. As we could see in figure 10 the goal value in the training set keeps increasing. Figure 11 shows that when the model keeps training the parameter values become more extreme. What also stands out is that a lot of parameter values become negative after a long number of epochs. This is the model trying to improve the in sample goal value by continually adjusting parameters. What stands out is that the DRL-Totalreturns algorithm takes more positive values for the parameters when trained for 500 epochs.

6 Conclusion

This research set out to determine whether the increased computational intensity of direct reinforcement learning is justified when trading cryptocurrencies. These machine learning strategies are offset against computationally simple technical trading strategies and compared on several performance measures. We show that direct reinforcement learning strategies are able to outperform momentum trading strategies in most cases. A validation strategy is introduced that increases the performance of the quantitative strategies, and the influence of the different hyperparameters is analyzed to determine the best model specification. For achieving the highest performance in live trading situations, a backtesting and grid search strategy needs to be employed to determine the best hyperparameters

for the model. The validation strategy ensures the models do not overfit and encourage using a smaller number of epochs for training the models.This means direct reinforcement learning is a viable trading strategy for trading cryptocurrencies. Online direct reinforcement learning strategies will require more research to improve performance. Updating the online strategies using the training performance function value results in a model that is overfitted and makes poor trading decisions.

Since quantitative strategies are able to outperform technical strategies it is of interest to further investigate improving of direct reinforcement learning for trading. This research already introduced a validation strategy, investigated online learning and analysed the hyperparameters needed for training the model. One could try to improve the direct reinforcement learning model by doing a grid search over the hyperparameters every time the model has to be retrained. This would result in a trading strategy that is able to adjust its specification while it is trading. For online learning a validation strategy could be considered to ensure that the model does not overfit after updating the parameters a certain number of times. Next to that, determining how to include different parameters into the trading decision is of interest to trades. A trader usually doesn't make a decisions solely based on a price series, they may consider volume, number of trades or other measures to arrive at a decision. This research can be taken as a framework for improving direct reinforcement learning for trading, leading the way to automated trading systems in cryptocurrency markets.

References

Baz, J., Granger, N., Harvey, C., Le Roux, N., & Rattray, S. (2015). Dissecting investment strategies in the cross section and time series. *Available at SSRN 2695101*.

Dempster, M. A., & Leemans, V. (2006). An automated FX trading system using adaptive reinforcement learning. *Expert Systems with Applications*, *30*(3), 543–552.

Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, *28*(3), 653-664.

Gold, C. (2003). FX trading via recurrent reinforcement learning. *2003 IEEE International Conference on Computational Intelligence for Financial Engineering*, 363–370.

Koker, T. E., & Koutmos, D. (2020). Cryptocurrency trading using machine learning. *Journal of Risk and Financial Management*, *13*(8).

Kyriazis, N., Papadamou, S., & Corbet, S. (2020). A systematic review of the bubble dynamics of cryptocurrency prices. *Research in International Business and Finance*, *54*, 101254.

Makarov, I., & Shoar, A. (2020). Trading and arbitrage in cryptocurrency markets. *Journal of Financial Economics*, *135*(2), 293–319.

Maringer, D., & Ramtohol, T. (2012). Regime-switching recurrent reinforcment learning for investment decision making. *Computational Management Science*, *9*(1), 89–107.

Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE transactions on neural networks*, *12*(4), 875–889.

Moody, J., & Wu, L. (1997). Optimization of trading systems and portfolios. In *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)* (pp. 300–307). doi: 10.1109/CIFER.1997.618952

Moody, J., Wu, L., Liao, Y., & Saffell, M. (1998). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, *17*(5–6), 441–470.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2017). Automatic differentiation in PyTorch.

Rohrbach, J., Suremann, S., & Osterrieder, J. (2017). Momentum and trend following strategies for currencies revisited-combining academia and industry. *Available at SSRN 2949379*.