Discussion Papers in Economics

No. 05/2020

Deep reinforcement learning for the optimal placement of cryptocurrency limit orders

Matthias Schnaubelt
FAU Erlangen-Nürnberg

ISSN 1867-6707

Deep reinforcement learning for the optimal placement of cryptocurrency limit orders

Matthias Schnaubelt[a,1]

[a] *University of Erlangen-Nürnberg, Department of Statistics and Econometrics, Lange Gasse 20, 90403 Nürnberg, Germany*

Abstract

This paper presents the first large-scale application of deep reinforcement learning to optimize the placement of limit orders at cryptocurrency exchanges. For training and out-of-sample evaluation, we use a virtual limit order exchange to reward agents according to the realized shortfall over a series of time steps. Based on the literature, we generate features that inform the agent about the current market state. Leveraging 18 months of high-frequency data with 300 million historic trades and more than 3.5 million order book states from major exchanges and currency pairs, we empirically compare state-of-the-art deep reinforcement learning algorithms to several benchmarks. We find proximal policy optimization to reliably learn superior order placement strategies when compared to deep double Q-networks and other benchmarks. Further analyses shed light into the black box of the learned execution strategy. Important features are current liquidity costs and queue imbalances, where the latter can be interpreted as predictors of short-term mid-price returns. To preferably execute volume in limit orders to avoid additional market order exchange fees, order placement tends to be more aggressive in expectation of unfavorable price movements.

Keywords: Finance; Optimal Execution; Limit Order Markets; Machine learning; Deep Reinforcement Learning

Email address: `matthias.schnaubelt@fau.de` (Matthias Schnaubelt)

[1]The author has benefited from many helpful discussions with Ingo Klein, Christopher Krauss, Thomas Fischer, Oleg Seifert and Markus Schmitz.

1. Introduction

According to Cont and Kukanov (2017), the trading process of professional market participants can be split into several stages of decision making: First, the *portfolio allocation* stage involves multiple decisions to select asset classes and individual assets for buying or selling and to assign portions of managed capital. Second, during the *order scheduling* stage, the volume allocated to a specific asset is split into smaller blocks that are to be executed over a time frame of several minutes to days. Third, in the *order placement* stage, volume slices are translated into a series of orders submitted to a trading venue, and further order details such as order type and limit price need to be specified. With optimized execution strategies for the last two stages, investors seek to minimize the implementation shortfall (Perold, 1988), i.e., the difference between observed market prices at the time of the portfolio allocation decision and the actually executed price. With this paper, we present the first large-scale empirical evaluation of deep reinforcement learning to optimize order placement in cryptocurrency limit order markets.

Beginning with the seminal works of Bertsimas and Lo (1998) and Almgren and Chriss (2001), optimal execution problems have been theoretically studied by academics.[2] Bertsimas and Lo (1998) analyze the stage of optimal order scheduling as a stochastic control problem with explicit models of price dynamics and market impact, which is solved by dynamic programming to minimize the implementation shortfall. Almgren and Chriss (2001) extend the analysis by also including volatility risk and the trader's risk aversion. Subsequently, extensions to multiple assets or dynamic liquidity were considered (see, for example, Obizhaeva and Wang, 2013; Tsoukalas et al., 2017, and references therein). By contrast, other works focus on the stage of optimal order placement, for example, by proposing models to study the optimal limit order price, the optimal ratio between market and limit orders or the optimal allocation across exchanges (see, for example, Bayraktar and Ludkovski, 2014; Cartea and Jaimungal, 2015; Cont and Kukanov, 2017).

To optimize execution, these approaches typically use explicit models of limit order dynamics, whose formulation requires substantial approximations, and extensions to practical applications are in many instances either difficult or impossible. By contrast, model-free reinforcement learning (RL) algorithms promise a data-driven approach to optimal execution problems by "learning what to do—how to map situations to actions—so as to maximize a numerical reward signal" (Sutton and Barto, 2018, p. 1). Inspired by recent successes of deep RL in solving complex tasks such as playing Go or controlling a robot's hand to solve the Rubik's Cube (see, for example, Silver et al., 2018; Akkaya et al., 2019), there is rising interest in both academia and practice to apply

[2]See also Cartea et al. (2015) for a survey of further works.

RL to optimal execution problems. In theory, investors could train RL algorithms to directly learn an execution strategy that minimizes the implementation shortfall under specific constraints, such as exchange commissions, by choosing the best action from a given set of alternatives. These alternatives could, for example, include decisions on the order type, limit price, traded volume, or trading venue. Hence, it is less surprising that RL is already used by market professionals, such as JPMorgan Chase & Co. (Terekhova, 2017; Noonan, 2017) or the broker firm Instinet Inc. (Cheng, 2017), to optimize execution.

Compared to the relatively large number of theoretical works modeling optimal execution as a stochastic control problem, empirical academic research on the use of RL in this domain is limited, which may partially be attributed to the substantial data requirements of such studies. In their seminal work, Nevmyvaka et al. (2006)[3] propose a problem-specific RL algorithm for order placement: Over several discrete time steps, the agent has to decide on the price of a limit order with the goal of selling all volume with minimal implementation shortfall. The study is based on a limit order exchange simulation backed by 1.5 years of high-frequency data for three stocks and shows that their algorithm reduces realized shortfalls compared to a submit-and-leave strategy.

Most subsequent applications of RL for execution optimization focus on the stage of order scheduling: Hendricks and Wilcox (2014) study how to split volume between market orders by extending the model of Almgren and Chriss (2001) to Q-learning, a table-based form of RL (Sutton and Barto, 2018). Their empirical analysis is based on one year of market data aggregated to 5 minute-bins with five levels of the limit order book. Similarly, Ning et al. (2018) apply deep double Q-networks (DDQN) in optimizing the volume trajectory of market orders. To evaluate the approach, they employ time series of mid prices from nine stocks and model the price impact as a quadratic function of executed volume. Other works study RL for order scheduling in simulated market environments without using market data (Bao and Liu, 2019; Daberius et al., 2019).

The academic evaluation of deep RL for order placement in the spirit of Nevmyvaka et al. (2006) seems to be still in its infancy: Rantil and Dahlen (2018) apply Sarsa(λ) and proximal policy optimization (PPO) to 1.5 years of data from four future contracts. Their results show that PPO may reduce the implementation shortfall in comparison to their benchmarks. Patel (2018) trains separate agents for portfolio allocation and order placement with DDQN and evaluates their performance on a few days of cryptocurrency data.

To our knowledge, no work has systematically tested the promise of deep RL for limit order placement in a large-scale empirical study. With this paper, we seek to close this gap by applying

[3]Parts of this initial work have been put in a broader context in Kearns and Nevmyvaka (2013).

state-of-the-art deep RL algorithms to optimize the placement of limit orders in cryptocurrency markets. Specifically, we make the following contributions to the literature: First, we provide a detailed description of an environment suited to train and evaluate reinforcement learning agents on the task of optimal limit order placement, in which an agent is asked to decide on the price of submitted limit orders over the course of several time steps. Based on cryptocurrency market data, order execution is simulated in a virtual limit order exchange. Our empirical evaluation is based on 264 GB of high-frequency limit order data covering 18 months, which comprises 300 million historic trades and more than 3.5 million order book states from major exchanges and cryptocurrency pairs. Based on the literature, we generate several features that inform the agent about current market liquidity and short-term return-predictive signals. A reward function rewards the agent based on the realized implementation shortfall and allows the direct inclusion of exchange-specific constraints such as order commissions.

Second, we compare the performance of two state-of-the-art deep RL algorithms in optimizing limit order placement. Specifically, we employ deep double Q-learning (van Hasselt et al., 2015) and proximal policy optimization (Schulman et al., 2017b), and benchmark their performance against several static and dynamic execution policies. Training the models multiple times in a rolling-window validation scheme, we find that PPO realizes the lowest mean total implementation shortfall. Compared to immediate execution in a single market order, PPO is found to reduce total transaction costs by up to 37.71 percent. The superior performance is found to be fairly robust, as PPO achieves the lowest shortfall among all strategies in 89.06 percent of all studied train-test splits and exchange-market combinations.

Third, we shed light into the black box of the PPO algorithm and assess whether the agent's actions align with economic rationale. In a first analysis, we find that the agent preferably uses limit orders to avoid additional exchange commissions of market orders. A second analysis on the contribution of features reveals that liquidity costs and queue imbalances provide the highest value – well in line with an interpretation of the latter in terms of short-term predictors of mid-price returns. In a third analysis, we find the agent's order placement to become more aggressive, i.e., to use more competitive limit prices, in anticipation of disadvantageous price movements.

The remainder of this paper is organized as follows: In Section 2, we describe the data sets and software used for our work. Section 3 details all building blocks of our methodology, i.e., the order execution environment and employed RL algorithms. Our results are presented in Section 4, and we conclude in Section 5.

2. Data and software

2.1. Data

Our empirical study is based on a large sample of cryptocurrency limit order data which comprises the 18 months from January 1st, 2018 to June 30th, 2019. In total, we have obtained over 264 GB of raw data. For our empirical evaluation, we select the largest exchanges and currency pairs in terms of traded volume. For the currency pair Bitcoin against US Dollar (BTC/USD), we hence obtain data from the exchanges BitFinex, Kraken and Coinbase. In total, we estimate that these three exchanges have covered 65 percent of BTC/USD limit order trading volume for the time period of our study.[4] To also evaluate algorithms for other currency pairs, we obtain data for the currency pair Ethereum to Bitcoin (ETH/BTC) and Ethereum to US Dollar (ETH/USD). Despite being relatively new and largely unregulated, cryptocurrency limit order exchanges exhibit most stylized facts from established exchanges (Schnaubelt et al., 2019). One of their key idiosyncrasies, relatively shallow limit order books and consequently a relatively high level of liquidity costs, increases their potential as test bed for optimal execution algorithms. Table 1 provides further details on the properties of the exchanges used in this study.

Following Schnaubelt et al. (2019), we obtain data from the selected exchanges using their application programming interfaces (API). Specifically, the data consist of transaction data (timestamp with millisecond resolution, executed price, volume and buy/sell flag) and data from the limit order book. The latter records the limit order volume at all price steps of the order book that were available during the time of retrieval via the exchange's API. To prepare limit order data for our models, we reconstruct the limit order book at a minutely sampling frequency. We also match sequences of trades to the respective time interval between order book states.

Exchange	**Market**	**Tick size**	**Maker fee [bp]**	**Taker fee [bp]**
BitFinex	BTC/USD	10^{-1}	10	20
	ETH/USD	10^{-2}	10	20
	ETH/BTC	10^{-5}	10	20
Kraken	BTC/USD	10^{-1}	16	26
Coinbase	BTC/USD	10^{-2}	0	30

Table 1: **Overview of exchange properties.** This table reports tick sizes, i.e., the smallest possible increments of the limit price at the exchange, and relative exchange commissions (in basis points) for all exchanges and markets used in this study. Commissions are split into a maker fee for providing liquidity to the order book and a taker fee for removing liquidity from the order book, e.g., in a market order.

	Count	**Mean**	**S.D.**	**Q1%**	**Q25%**	**Q50%**	**Q75%**	**Q99%**
Panel A: BitFinex BTC/USD								
Trade volume [BTC]	87972104	0.3483	1.8636	0.0009	0.0125	0.0550	0.2455	4.3243
Trade price [USD/BTC]	87972104	7061	2483	3440	5260	6722	8434	14369
Trade value [USD]	87972104	2393	12296	6.2272	83.20	368.09	1664	30082
Minutely trade volume [BTC]	786240	38.97	77.78	0.0000	4.5751	16.78	44.59	318.20
Minutely trade count	786240	111.89	91.65	0.0000	50.00	92.00	149.00	443.00
Daily trade volume [BTC]	546	56111	20947	15893	43198	53584	67396	117699
Panel B: Kraken BTC/USD								
Trade volume [BTC]	64828482	0.3782	1.1407	0.0000	0.0109	0.0663	0.3000	4.6166
Trade price [USD/BTC]	64828482	6870	2463	3346	4146	6705	8373	13770
Trade value [USD]	64828482	2322	6883	0.0203	79.61	428.78	1896	26686
Minutely trade volume [BTC]	776740	31.57	98.32	0.0000	0.1516	5.5000	26.11	390.25
Minutely trade count	776740	83.46	136.83	0.0000	12.00	48.00	108.00	571.00
Daily trade volume [BTC]	540	45410	22895	0.0000	35512	45702	56880	102264
Panel C: Coinbase BTC/USD								
Trade volume [BTC]	74310311	0.1723	0.7369	0.0002	0.0036	0.0178	0.0948	2.5000
Trade price [USD/BTC]	74310311	6879	2450	3359	5058	6596	8283	13977
Trade value [USD]	74310311	1202	5139	1.3742	21.25	101.51	589.81	17286
Minutely trade volume [BTC]	776634	16.48	26.00	0.0000	3.6937	8.1705	18.56	123.88
Minutely trade count	776634	95.68	78.04	0.0000	60.00	86.00	117.00	310.00
Daily trade volume [BTC]	540	23706	10693	7516	15751	22137	29596	58229
Panel D: BitFinex ETH/USD								
Trade volume [ETH]	32772982	4.8417	19.37	0.0020	0.3000	1.1127	4.7082	50.33
Trade price [USD/ETH]	32772982	408.11	300.06	90.34	167.00	276.54	589.22	1278
Trade value [USD]	32772982	1528	5733	0.7214	93.42	391.04	1338	15910
Minutely trade volume [ETH]	776634	204.32	404.44	0.0000	18.29	77.40	225.42	1807
Minutely trade count	776634	42.20	42.30	0.0000	12.00	30.00	58.00	190.00
Daily trade volume [ETH]	540	293848	147843	94392	197124	250747	340583	753885
Panel E: BitFinex ETH/BTC								
Trade volume [ETH]	43952499	2.5096	17.43	0.0045	0.1745	0.5075	1.9775	28.00
Trade price [BTC/ETH]	43952499	0.0510	0.0233	0.0260	0.0318	0.0360	0.0726	0.1049
Trade value [BTC]	43952499	0.1058	0.9591	0.0002	0.0076	0.0280	0.0808	1.1934
Minutely trade volume [ETH]	786240	140.29	908.35	0.0000	0.0000	19.26	87.62	1912
Minutely trade count	786240	55.90	88.68	0.0000	0.0000	32.00	72.00	397.00
Daily trade volume [ETH]	546	202018	108480	61362	114206	184444	260677	487366

Table 2: **Descriptive statistics of trade data.** This table reports descriptive trade statistics for all studied exchanges and markets. Data is shown for the sample period from January 1st, 2018 to June 30th, 2019. Column *S.D.* contains the respective standard deviation, and columns *Q1%* to *Q99%* indicate quantiles at the given levels.

2.2. Descriptive data statistics

Table 2 reports summary statistics of our trade data. In terms of traded volume, BitFinex is the largest exchange serving the BTC/USD market in our sample (average daily volume of 56111 BTC). The Kraken exchange ranks second (45410 BTC), and the lowest daily volume was traded at the Coinbase exchange (23706 BTC). With 0.3483 BTC and 0.3782 BTC, average trade

[4] To estimate exchange trading volume shares, we obtain data from data.bitcoinity.org for our sample period.

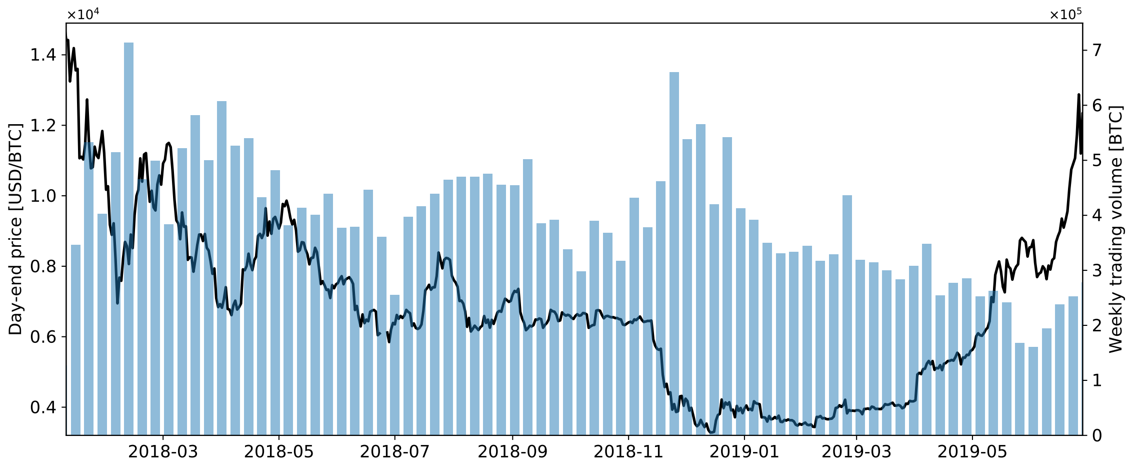

Figure 1: **Evolution of BTC/USD price and traded volume.** This plot depicts the day-end price (in UTC time, left axis, black line) and weekly cumulated trading volume (right axis, blue bar plot) for the BitFinex BTC/USD market for our sample period.

sizes at the BitFinex and Kraken exchanges are comparable. Again, the Coinbase exchange presents an exception with an average trade size of 0.1723 BTC. In an average minute, about 100 single trades are recorded at the exchanges. Figure 1 depicts the evolution of the day-end price and weekly traded volume for the BTC/USD market at the BitFinex exchange. Our sample period covers both relatively volatile and stable periods, with phases of price declines and increases. Turning to descriptive statistics for the BitFinex ETH/USD and ETH/BTC markets, we note a lower number of trades. On average, 42.20 and 55.90 trades occur per minute for the ETH/USD and ETH/BTC pairs, respectively. Also, the average value of the trades is lower. For the BitFinex ETH/USD exchange, we find a mean trade value of 1528 USD, which is lower than the average trade value for the BTC/USD pair at the same exchange (2393 USD). Similarly, the average trade value for the ETH/BTC pair is only 0.1058 BTC, which corresponds to less than one third of the mean trade volume at the BitFinex BTC/USD market (0.3483 BTC).

Table 3 presents summary statistics of the reconstructed limit order book data. Mean relative bid-ask spreads for the BTC/USD currency pair at the BitFinex and Coinbase exchanges are at 0.679 bp and 0.385 bp, which is roughly one order of magnitude smaller than at the Kraken exchange. Compared to the bid-ask spread, mean liquidity costs, i.e., the relative change of the volume-weighted-average price of market order execution compared to the mid price, for a volume of 10 BTC (for example, 3.222 bp for the BitFinex exchange), are considerably above the mean bid-ask spread. The largest average volume at the best-bid price for the BTC/USD pair is found at the BitFinex exchange (7.742 BTC), and the lowest at the Coinbase exchange (1.217 BTC). Looking at the results for the BitFinex ETH/USD and ETH/BTC pairs (Panels D and E of Table 3), we find bid-ask spreads (1.283 bp and 2.822 bp, respectively) to be considerably higher than those of the

	Count	Mean	S.D.	Q1%	Q25%	Q50%	Q75%	Q99%
Panel A: BitFinex BTC/USD								
Relative bid-ask spread [bp]	726722	0.679	1.575	0.104	0.143	0.170	0.393	7.044
Best-bid volume [BTC]	726722	7.742	22.067	0.004	0.657	2.586	7.867	72.070
Best-ask volume [BTC]	726722	7.425	20.397	0.004	0.595	2.610	8.060	68.744
Rel. bid VWAP for 10 BTC [bp]	726722	3.222	3.552	0.057	0.313	2.183	4.933	15.067
Rel. ask VWAP for 10 BTC [bp]	726722	3.384	3.775	0.057	0.287	2.250	5.208	15.995
Panel B: Kraken BTC/USD								
Relative bid-ask spread [bp]	629457	3.465	4.721	0.094	0.272	1.926	4.953	20.670
Best-bid volume [BTC]	629457	2.589	9.894	0.003	0.186	0.950	2.008	29.547
Rel. bid VWAP for 10 BTC [bp]	629457	8.032	8.400	0.079	3.203	6.044	10.102	43.604
Panel C: Coinbase BTC/USD								
Relative bid-ask spread [bp]	759005	0.385	1.505	0.007	0.013	0.016	0.026	6.020
Best-bid volume [BTC]	759005	1.217	4.161	0.001	0.017	0.200	1.000	17.878
Rel. bid VWAP for 10 BTC [bp]	759005	3.262	4.403	0.004	0.013	1.732	4.954	18.932
Panel D: BitFinex ETH/USD								
Relative bid-ask spread [bp]	719326	1.283	2.609	0.107	0.224	0.486	0.851	12.453
Best-bid volume [ETH]	719326	74.511	640.457	0.040	10.354	25.102	55.656	578.518
Rel. bid VWAP for 100 ETH [bp]	719326	5.616	5.818	0.083	1.295	4.349	7.898	28.007
Panel E: BitFinex ETH/BTC								
Relative bid-ask spread [bp]	723055	2.822	3.327	0.119	0.314	1.466	4.543	13.708
Best-bid volume [ETH]	723055	26.472	216.503	0.022	1.000	4.089	14.745	247.687
Rel. bid VWAP for 100 ETH [bp]	723055	10.195	6.709	0.145	5.443	9.517	13.966	30.015

Table 3: **Summary statistics of order book data.** This table reports descriptive statistics for the order book data. Data is shown for the sample period from January 1st, 2018 to June 30th, 2019. Column *S.D.* contains the respective standard deviation, and columns *Q1%* to *Q99%* indicate quantiles at the given levels.

BTC/USD currency pair. Judging by best-bid volumes and relative VWAP spreads, the BitFinex ETH/BTC market is the least liquid among the studied pairs at the BitFinex exchange.

2.3. Software

We implement both the execution environment and the reinforcement learning algorithms in Python (Python Software Foundation, 2016). We store all trade and limit order data in a PostgreSQL database (PostgreSQL Global Development Group, 2018), which we access using psycopg2 (Varazzo, 2011). Data processing is performed using pandas (McKinney, 2010) and numpy (Van Der Walt et al., 2011). We implement the order execution environment in line with the interface specification of the OpenAI Gym library (Brockman et al., 2016). To speed up the implementation of the virtual limit order exchange, we employ the Numba just-in-time compiler for Python (Lam et al., 2015). Finally, we use the implementation of the stable-baselines library (Hill et al., 2018) for the deep double Q-network and proximal policy optimization algorithms. Computations are performed on an AMD Ryzen Threadripper 1950X CPU, and neural networks are trained with TensorFlow (Abadi et al., 2015) on a Nvidia GeForce GTX 1080 GPU.

3. Methodology

In this section, we describe our methodology. First, we outline how we simulate limit order matching based on the available market data in a virtual limit order exchange (Section 3.1). Second, we formulate the optimal limit order placement problem in the framework of reinforcement learning (Section 3.2). Third, we train different reinforcement learning algorithms (Section 3.3) and benchmark the learned strategies against three static execution policies (Section 3.4). Finally, the performance evaluation of the learned strategies is outlined in Section 3.5.

3.1. Simulating order matching in a virtual limit order exchange

To train and evaluate algorithms for the task of optimal order placement, we set up a virtual limit order exchange that is based on high-frequency market data. In the following, we describe how the simulation models the execution of a limit order that is submitted at time step t, and which we denote as $x_t = (p_t, \omega_t)$ with a limit price $p_t \in \mathbb{R}^+$ and volume $\omega_t \in \mathbb{R}$. For $\omega_t > 0$, the order is a sell order, and else a buy order. We assume that the order is valid until the next discrete time step $t+1$. The simulation of the execution of order x_t proceeds in two consecutive steps similar to the price-time prioritized matching found at most real limit order exchanges (compare, for example, Gould et al., 2013), and is inspired by the simulation used in Rantil and Dahlen (2018). In the first step, the simulation checks whether the order can be matched immediately with current orders in the limit order book. In case of an ask order, the algorithm compares the order's limit price p_x with the prices of limit orders on the bid side of the limit order book, starting with the best-bid order. By contrast, if the submitted order is a bid order, the ask side of the limit order book is considered. If prices of order pairs are compatible, i.e., ask prices are below bid prices, a virtual trade is executed. After completion of the immediate matching, ω_t^{im} and p_t^{im} store the resulting matched volume and volume-weighted execution price, respectively. The sign of ω_t^{im} corresponds to the sign of ω_t. In the second step, the remaining limit order volume $\omega_t - \omega_t^{\text{im}}$ enters the virtual limit order book and execution is simulated based on the historic stream of trades between time steps t and $t+1$. Same-side orders in the order book at time t with equal or better limit prices are given priority, since these would have been matched before our newly submitted order. Effectively, these orders increase the volume of our own limit order. The matching of limit order volume then proceeds as follows: Any trade from the time-ordered stream of trades following the limit order book in time step t is checked in terms of execution price. If the trade was executed at a price higher (lower) than the ask (bid) limit price p_t, the traded volume reduces the remaining volume of our own limit order. After all trades between time steps t and $t+1$ have been checked, the limit order matching results in the execution of some volume ω_t^{lim} with $|\omega_t^{\text{lim}}| \in [0, \omega_t - \omega_t^{\text{im}}]$ at the volume-weighted execution price p_t^{lim}.

Several approximations are made by simulating limit order matching this way. First, we assume that the submission of our own orders is instantaneous in the sense that there is no time delay between the observation of the limit order book and the subsequent generation and submission of our own limit order. In practice, this time delay could be minimized by technological means such as direct short-distance links to the exchange and optimized hard- and software. Second, we assume that there is no hidden liquidity in the limit order book, e.g., from iceberg orders. As only one exchange has supported hidden liquidity during our sample period (the *BitFinex* exchange, compare Schnaubelt et al., 2019), we conjecture that this assumption does not severely affect empirical results. Third, we assume that the submission of our limit orders does not have a permanent impact on market price or liquidity and does not impact the behavior of other market participants.[5] A rule of thumb is that permanent price impact is insignificant as long as the daily participation rate is below 10 percent (Almgren and Chriss, 2001, p. 24). Limit order markets are typically found to possess a relatively high resilience of liquidity, i.e., liquidity levels revert to their average within a short period of time (Degryse et al., 2005; Cummings and Frino, 2010; Gomber et al., 2015). For cryptocurrency limit order markets, Schnaubelt et al. (2019) similarly find that liquidity close to the bid-ask spread recovers within a few seconds after trades in the top percentile of trading volume. As our empirical study focuses on the stage of optimal order placement and assumes that large volumes have been split into smaller blocks distributed over the day during order scheduling, and to keep our empirical analysis within the limits of these assumptions, we select initial volumes v_0 that are in the order of magnitude of typical average minutely trading volume. Also, our sampling frequency of one minute is considerably larger than typical recovery time scales.

3.2. A reinforcement learning formulation of optimal order placement

Next, we formulate the optimal order placement problem as given in Nevmyvaka et al. (2006) in the framework of reinforcement learning.[6] The agent, which we can think of as an investor, faces the following problem: Over the course of an episode, i.e., a finite number T of discrete time steps, the agent has to liquidate an initial position of $v_0 \in \mathbb{R}$ units of an asset (e.g., stocks, future contracts, foreign exchange) at a limit order exchange. The initial volume v_0 is positive in the case of selling the asset, and negative when buying. The overall goal of the agent is to minimize

[5]This assumption is not uncommon in the study of reinforcement learning for optimal execution, see, for example, Nevmyvaka et al. (2006) or Hendricks and Wilcox (2014).

[6]In the following, we use the common notation and terminology for reinforcement learning from Sutton and Barto (2018).

the total execution costs of the full volume, which consist of an implementation shortfall (Perold, 1988; Almgren and Chriss, 2001) and an order commission charged by the exchange. In every time step $t \in \{0, \ldots, T-1\}$, the agent observes the current environment through a state variable s_t and chooses an action a_t from a discrete set of possible actions to execute the position. This action encodes the price of a limit order that is submitted to the virtual exchange. The agent follows a policy π that specifies which action to choose given the observed state. Our virtual exchange then simulates the execution of the order, which may result in (partial) order execution. Based on the volume and price of the executed trade, the agent receives an immediate reward r_{t+1}, and the remaining volume is retained for execution in the next time step as v_{t+1}.

In the following, we formalize this description and extend it to a complete description of the reinforcement learning environment for the subsequent empirical analysis. First, Section 3.2.1 describes the action space. Second, Section 3.2.2 provides details on the reward function. Third, we discuss the state space in Section 3.2.3.

3.2.1. Action space

The agent chooses an action a_t from a finite set of actions $\mathcal{A} = \{0, 1, \ldots, 2N_{\text{action}}\}$ at every time step t, where N_{action} specifies the size of the action space. The first action ($a = 0$) encodes the trivial limit order of zero size. In line with Nevmyvaka et al. (2006), the remaining actions encode the number of ticks that the order is placed away from the current best-bid or best-ask price. The size of the limit order is always the remaining volume in the current time step, v_t. Specifically, if $v_t > 0$, the submitted order $x_t = (p_t, \omega_t)$ with limit price p_t and size ω_t for a given action a_t is

$$x_t^{\text{sell}} = \begin{cases} (\text{ask}_t + \Delta p(a_t - N_{\text{action}}),\ v_t)\,, & \text{if } a_t > 0, \\ (0,\ 0), & \text{if } a_t = 0. \end{cases} \tag{1}$$

Herein, Δp denotes the tick size, i.e., the minimal increment of the limit price of the exchange, and ask_t is the best-ask price at time step t. Similarly, if $v_t < 0$, the limit order x_t is given by

$$x_t^{\text{buy}} = \begin{cases} (\text{bid}_t - \Delta p(a_t - N_{\text{action}}),\ v_t)\,, & \text{if } a_t > 0, \\ (0,\ 0), & \text{if } a_t = 0, \end{cases} \tag{2}$$

where bid_t denotes the best-bid price at time step t. The limit order persists until complete execution or until the limit order is updated in time step $t+1$. In the last time step, any remaining volume to sell (buy) is executed with a market order, i.e., with a limit order price $p_{T-1} = -\infty$ ($p_{T-1} = \infty$) to ensure that the position is closed.

3.2.2. Reward function

After its action a_t has been executed in the matching simulation, we reward the agent with a scalar reward signal $r_{t+1} \in \mathbb{R}$ given by

$$r_{t+1} = \frac{\omega_t^{\text{ex}}}{v_0} \left(\frac{\omega_t^{\text{ex}} p_t^{\text{ex}} - c_t^{\text{ex}}}{\omega_t^{\text{ex}} \, \text{mid}_0} - 1 \right). \tag{3}$$

Here, $\omega_t^{\text{ex}} = \omega_t^{\text{im}} + \omega_t^{\text{lim}}$ denotes the total executed volume, and $p_t^{\text{ex}} = (p_t^{\text{im}} \omega_t^{\text{im}} + p_t^{\text{lim}} \omega_t^{\text{lim}}) / \omega_t^{\text{ex}}$ is the corresponding volume-weighted average execution price. The total exchange commission is given in terms of a fraction of trade value, i.e., $c_t^{\text{ex}} = C^{\text{im}} |\omega_t^{\text{im}}| p_t^{\text{im}} + C^{\text{lim}} |\omega_t^{\text{lim}}| p_t^{\text{lim}} \geq 0$, where $C^{\text{im}}, C^{\text{lim}} \in \mathbb{R}^+$ are relative exchange commissions for taking and providing liquidity, respectively. We measure the total realized implementation shortfall relative to the mid price at the first time step, i.e., $\text{mid}_0 = (\text{ask}_0 + \text{bid}_0)/2$. Instead of considering absolute proceeds from the trade as in Nevmyvaka et al. (2006), we use a scaled reward function to account for potential changes in market prices over the course of the training and testing periods. This form of the reward function has the favorable property that the sum of all rewards of an episode is the relative total implementation shortfall for executing the volume v_0, i.e.,

$$\sum_{t=1}^{T} r_t = \frac{\sum_{t=0}^{T-1} \left[\omega_t^{\text{ex}} p_t^{\text{ex}} - c_t^{\text{ex}} \right]}{v_0 \, \text{mid}_0} - 1. \tag{4}$$

After each episode, we store the executed volume profile $(\omega_0^{\text{ex}}, \ldots, \omega_{T-1}^{\text{ex}})$, a per-step reward profile $(r_1, \ldots, r_T)$ and a commission profile $(c_0^{\text{ex}}, \ldots, c_{T-1}^{\text{ex}})$ for subsequent performance evaluation.

3.2.3. State space and feature generation

To support its choice of action, the agent observes the market state represented as a vector

$$s_t = (\vartheta_t, \rho_t, \tilde{x}_{t,1}, \ldots, \tilde{x}_{t,N_{\text{feature}}}) \in \mathcal{S} = (0, 1] \times [-1, 1] \times \mathbb{R}^{N_{\text{feature}}}. \tag{5}$$

The first entry encodes the time remaining to liquidate the position, scaled to the unit interval, i.e., $\vartheta_t = 1 - t/T$. For the second entry, we calculate the remaining volume as a fraction of the initial volume as $\rho_t = v_t / |v_0|$. For the remaining entries, we generate N_{feature} features $x_{t,1}, \ldots, x_{t,N_{\text{feature}}}$ that capture the current state of the limit order book and previous trading activity. Based on the literature[7], we determine several measures of market liquidity and determinants of temporary and permanent price impact. All features employed in this study are listed in Table 4. Following

[7]A survey on the topic can be found in Gould et al. (2013).

Variable	**Description**	**Reference(s)**
Transaction imbalances		
TC-IMBAL	*Trade count imbalance:* Difference of the number of sell trades to the number of buy trades in the past minute, normalized by the sum of these counts	Gopikrishnan et al. (2000); Plerou et al. (2002)
TV-IMBAL	*Trade volume imbalance:* Difference of the sell to buy trade volume in the past minute, normalized by the total traded volume in the past minute	Cao et al. (2009); Cont et al. (2014)
Best-order volumes and imbalances		
BO-IMBAL	*Best-order imbalance:* Difference between the current best-bid volume and the current best-ask volume, scaled by the sum of best-bid and best-ask volumes	Gould and Bonart (2016)
VOL-BID *VOL-ASK*	*Best-order volumes:* Current volume in the order book at the best-bid and best-ask prices	Ranaldo (2004)
Q-IMBAL(N_p)	*Queue imbalance:* Difference between the bid and ask volumes at the $N_p \in \{5, 10, 15, 20\}$ best price steps of the current order book, scaled by the sum of the volumes	Cao et al. (2009)
CVOL-BID(N_p) *CVOL-ASK*(N_p)	*Cumulated order volumes:* Current cumulated volume in the order book at the best $N_p \in \{10, 15, 20\}$ bid or ask price steps	Cao et al. (2009)
Volatility and current price drift		
VOLA	*Mid-price volatility:* Volatility of the mid price, calculated as the square root of the mean of squared logarithmic mid-price returns over the past 30 minutes	Danielsson and Payne (2001); Ranaldo (2004)
DRIFT	*Mid-price return drift:* Simple return of the mid price relative to the mid price of the first time step	
Current cost of liquidity		
LC-BID(V) *LC-ASK*(V)	*Relative buy/sell liquidity cost:* Current cost of the immediate execution of a buy or sell market order, relative to current mid price. We use volumes of $V \in \{10, 20, 30, 50\}$ BTC and $V \in \{10, 20, 30, 50, 100\}$ ETH for the BTC/USD pair and ETH denominated markets, respectively.	Gomber et al. (2015)
BA-SPREAD	*Bid-ask spread:* Current bid-ask spread, calculated as the difference between ask and bid price scaled by the mid price	Ranaldo (2004); Cao et al. (2009)

Table 4: **Description of features.** This table describes all features used within the reinforcement learning models along with relevant references.

common procedure for data preprocessing when applying neural networks, we scale every feature with a rolling-window standardization,

$$\tilde{x}_{t,i} = \frac{x_{t,i} - \bar{x}_{t,i}}{\sigma_{t,i}}, \quad \forall\, i \in \{1, \ldots, N_{\text{features}}\}, \tag{6}$$

where $\bar{x}_{t,i}$ and $\sigma_{t,i}$ denote the rolling mean and standard deviation over the past 1440 values (i.e., one trading day) of the i-th feature.

3.3. Reinforcement learning algorithms

The goal of the agent is to maximize cumulated future rewards, which are expressed as the sum of all discounted future returns following time step t, i.e.,

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{7}$$

where $\gamma \in [0, 1]$ denotes the discount factor.[8] For small values of γ, the agent's goal is to maximize immediate rewards; if instead γ is close to one, the agent behaves long-sighted (Sutton and Barto, 2018). As we assess the agent's performance in terms of the total implementation shortfall after T steps, we set $\gamma = 1$ to obtain an equal weight for all time steps. The behavior of the agent is described by a policy $\pi(a|s)$, which is the probability of choosing action $A_t = a$ conditional on the observed state $S_t = s$. The expected return of a state s is described by the state-value function

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right], \ \forall s \in \mathcal{S}, \tag{8}$$

where $\mathbb{E}_\pi[\bullet]$ denotes the expected value of a random variable given that the agent behaves according to policy π (Sutton and Barto, 2018). The action-value function under policy π describes the expected return of an action a for an agent in some state s, and is defined as

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right], \ \forall s \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{9}$$

An optimal policy π_* has the optimal state-value function $v_*(s) = \max_\pi v_\pi(s)$ and the optimal action-value function $q_*(s, a) = \max_\pi q_\pi(s, a)$. The optimal action-value function fulfills a Bellman optimality equation (Sutton and Barto, 2018) which states that the optimal action-value function is given by the expected reward of action a plus the discounted expected value of the best action in the subsequent state S_{t+1}, i.e.,

$$q_*(s, a) = \mathbb{E}_{S_{t+1}, R_{t+1}} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,|\, S_t = s, A_t = a \right]. \tag{10}$$

In the following, we describe the reinforcement learning algorithms studied in this paper, specifically, a problem-specific backward-induction Q-learning algorithm (Section 3.3.1), deep double Q-networks (Section 3.3.2) and proximal policy optimization (Section 3.3.3). We choose the latter two algorithms as they are state-of-the-art representatives of two common (deep) reinforcement

[8]Therein, we use uppercase letters to emphasize random variables.

learning approaches, i.e., value-based and policy-based algorithms, and are suitable for discrete action spaces.

3.3.1. Backward-induction Q-learning

As a first benchmark, we employ the problem-specific reinforcement learning algorithm of Nevmyvaka et al. (2006). Their reinforcement learning algorithm is similar to the tabular Q-learning method, however exploits the specific structure of the optimal placement problem and the fact that all market data used in training are known a priori. Tabular Q-learning (Watkins, 1989) approximates the optimal action-value function q_* by iteratively updating the so-called Q table according to the update rule

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right], \tag{11}$$

where $\alpha \in]0, 1]$ denotes a constant step-size parameter.

By contrast, the approach of Nevmyvaka et al. (2006) assumes that the optimal placement problem can be approximated by a Markov decision process. Consequently, the optimal action does not depend on any previous action, and the optimal action-value function can be learned backwards in time, starting at the last time step, $t = T - 1$. As market data are finite, we can introduce the following sampling scheme to collect all possible state-action-reward experience tuples that can occur in time step t: First, quantile-based discretization functions for the remaining volume, $c_\rho : [-1, 1] \mapsto \{0, \ldots, k_\rho\}$, and for the market variables, $c_x : \mathbb{R}^{N_{\text{feature}}} \mapsto \{0, \ldots, k_x\}^{N_{\text{feature}}}$, are introduced, which return the indices of one of $k_\rho + 1$ (for c_x, $k_x + 1$) classes. In a second step, we iterate over all possible discretized state variables

$$\tilde{s}_t = (T - t, c_\rho(\rho_t), c_x(\tilde{x}_t)), \ \forall \rho_t \in \mathcal{R}, \tilde{x}_t \in \mathcal{T}, \tag{12}$$

where $\mathcal{T}$ denotes the available standardized training data, and $\mathcal{R}$ is the set of discretized remaining volumes given by $\mathcal{R} = \{0, 1/k_\rho, 2/k_\rho, \ldots, 1\}$ in the case of selling the asset. Third, for every state, we simulate the execution of all actions $a \in \mathcal{A}$ in the environment, and add the resulting new states

$$\tilde{s}_{t+1,a} = (T - t - 1, c_\rho(\rho_{t+1}), c_x(\tilde{x}_{t+1})), \ \forall a \in \mathcal{A}, \tag{13}$$

and rewards $r_{t+1,a}$ as experience tuples $e_a = (\tilde{s}_t, a_t, r_{t+1,a}, \tilde{s}_{t+1,a})$, $\forall a \in \mathcal{A}$, to the experience set $\mathcal{D}_t$ for this time step. Finally, the action-value table can be updated for all states of time step t by averaging over expected returns, i.e.,

$$Q(\tilde{s}, a) = \frac{1}{|\mathcal{D}_t(\tilde{s})|} \sum_{(\tilde{s}_t, a_t, r_{t+1}, \tilde{s}_{t+1}) \in \mathcal{D}_t(\tilde{s})} r_{t+1} + \max_{a'} Q(\tilde{s}_{t+1}, a'), \tag{14}$$

where $\mathcal{D}_t(\tilde{s}) = \{(\tilde{s}_t, a_t, r_{t+1}, \tilde{s}_{t+1}) \in \mathcal{D}_t : \tilde{s}_t = \tilde{s}\}$ denotes the subset of experience for the updated state $\tilde{s}$. The elements of the Q-table for the next state, $Q(\tilde{s}_{t+1}, a')$, are already fully populated as time step $t + 1$ has been visited in the last iteration. The algorithm proceeds backwards until the first time step is reached. In line with Nevmyvaka et al. (2006), we use a volume discretization with $k_\rho = 8$ and feature discretization with $k_x = 10$, where bin edges are based on the feature's deciles.

3.3.2. Deep double Q-networks

As second reinforcement learning algorithm, we employ deep double Q-networks (DDQN), which have been successfully applied to complex tasks such as playing Atari video games (van Hasselt et al., 2015). The following summary of the algorithm is based on the initial papers (Mnih et al., 2013; van Hasselt et al., 2015; Mnih et al., 2015) as well as the implementation in Hill et al. (2018). For high-dimensional state spaces, tabular Q-learning does not generalize well. Therefore, the action-value function is often estimated by a function approximator. Specifically, deep Q-learning (Mnih et al., 2013, 2015) uses a neural network (Q-network), parametrized by network weights θ, to estimate the optimal action-value function, $Q_\theta(s, a) \approx q_*(s, a)$. To train the Q-network, one iteratively minimizes a sequence of loss functions,

$$L_i(\theta_i) = \mathbb{E}_{S,A}\left[\left(y_i(S, A) - Q_{\theta_i}(S, A)\right)^2\right], \tag{15}$$

where θ_i denotes the currently optimized network weights, and the target $y_i(s, a)$ is the expected action value obtained from a target Q-network with parameters θ_i^-, i.e.,

$$y_i^{\text{DQN}}(s, a) = \mathbb{E}_{S'}\left[R_{t+1} + \gamma \max_{a'} Q_{\theta_i^-}(S', a') \,|\, S_t = s, A_t = a\right]. \tag{16}$$

The target network parameters θ_i^- are set to the current parameter values θ_i after every C steps, and are kept constant in between. The expectation in Equation (15) is taken with respect to the behavior distribution, i.e., the probability distribution over states S and actions A. To decorrelate the sequence of training observations, increase data efficiency and avoid unstable parameters during training, deep Q-learning introduces a replay memory. The agent's last N experience tuples $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ are stored in a data set $\mathcal{D} = \{e_1, \ldots, e_N\}$. In every time step, the agent first executes an action according to an ϵ-greedy policy: With probability $1 - \epsilon$, the algorithm selects the optimal (greedy) action according to the current policy, i.e., $\text{argmax}_{a'} Q_{\theta_i^-}(s_t, a')$, and with probability ϵ, it selects a random action to explore the environment. Then, previous experience is replayed: A minibatch consisting of random experience observations from the replay memory and the experience from the currently executed action is used to minimize the loss function with stochastic gradient descent. In Equation (16), the use of the maximum operator implies that the same network weights θ_i^- are used to select and evaluate action a', which bears the risk of overoptimistic value estimates. To prevent this, deep double Q-networks (van Hasselt et al., 2015) decouple the selection and evaluation of the action by replacing the target with

$$y_i^{\text{DDQN}}(s,a) = \mathbb{E}_{S'}\left[R_{t+1} + \gamma Q_{\theta_i^-}(S', \text{argmax}_{a'} Q_{\theta_i}(S', a')) \,|\, S = s, A = a\right]. \tag{17}$$

In our study, we use the implementation of deep double Q-learning from Hill et al. (2018), and keep most parameters at their default values. Specifically, we use a feed-forward network architecture with two hidden layers, each consisting of 64 neurons. The size of the replay memory is $N = 5 \times 10^4$ experience tuples, and we update the target network every $C = 500$ steps. We increase the minibatch size to 256 to reduce noise by averaging over larger experience samples. The algorithm is trained over a total of 10^7 time steps. The probability of selecting a random action, ϵ, is 1 at the beginning of the training, and reduced linearly in time over the first quarter of training time steps to a value of 0.02, which is then kept constant until the end of training.

3.3.3. Proximal policy optimization

As third reinforcement learning algorithm, we employ the proximal policy optimization (PPO) algorithm. The following summary is based on Schulman et al. (2017a) and Schulman et al. (2017b), as well as the implementation in Hill et al. (2018). PPO is a state-of-the-art actor-critic method that targets data efficiency and robustness, rendering it an attractive choice in light of noisy financial data. As a policy gradient method, PPO seeks to directly learn a parameterized stochastic policy $\pi_\theta(a|s) : \mathcal{S} \times \mathcal{A} \mapsto [0,1]$ (also called policy surrogate), by maximizing an objective function $L(\theta)$ with respect to θ. In our implementation, the policy function is approximated by a neural network. PPO introduces the objective function

$$L_t^{\text{PPO}}(\theta) = \hat{\mathbb{E}}_N \left[L_t^{\text{CLIP}}(\theta) - c_{\text{VF}} L_t^{\text{VF}}(\theta) + c_{\text{H}} H[\pi_\theta](s_t)\right], \tag{18}$$

where $\hat{\mathbb{E}}_N$ denotes an average over a finite sample of experience. Typically, the current policy π is run for N time steps, and then the sampled experience trajectory is used to update the policy π with minibatch stochastic gradient descent or Adam (Kingma and Ba, 2017). With the first term, PPO introduces a clipped objective function,

$$L_t^{\mathrm{CLIP}}(\theta) = \min\left(r_t(\theta)\hat{A}_t,\ \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right), \tag{19}$$

which suppresses very large policy updates by clipping the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ such that it falls within the interval $[1-\epsilon, 1+\epsilon]$. The advantage function $A^\pi(s,a)$ measures how much better the action is compared to the policy's default behavior, i.e., $A^\pi(s,a) = Q_\pi(s,a) - v_\pi(s)$. In practice, the advantage function is estimated from the sampled trajectory of length N with the generalized advantage estimator (Schulman et al., 2018) given by

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{N-t+1}\delta_{N-1}, \tag{20}$$

which computes an exponentially-weighted average of the temporal-difference residual of v_π, $\delta_t = r_t + \gamma v_\pi(s_{t+1}) - v_\pi(s_t)$ over time steps t to $N-1$. The advantage estimator depends on the state-value function $v_\pi(s_t)$, which is estimated with a second network, the critic network. As parameters are shared between the state-value function estimator and the policy surrogate $\pi(a|s;\theta)$, a value-function error term

$$L_t^{\mathrm{VF}} = (v_\theta(s_t) - v_t^{\mathrm{targ}})^2 \tag{21}$$

is added to the objective function (scaled by a positive constant parameter c_{VF}), where v_t^{targ} denotes the state-value function's training target. Following Williams (1992) and Mnih et al. (2016), PPO also adds the entropy of the policy π_θ for state s_t, $H[\pi_\theta](s_t)$, scaled by a constant positive parameter c_{H}, to the objective in order to discourage early convergence to purely deterministic policies, and hence to improve exploration.

Our parameter settings are based on the default values of Hill et al. (2018). Specifically, the policy network is a feed-forward network with two hidden layers, each consisting of 64 neurons. We sample from a single environment and use the default trajectory length of $N = 128$. The weighting factor of the generalized advantage estimator is set to $\lambda = 0.95$. The clipping parameter ϵ is set to 0.2, the value-function error and entropy terms are scaled by $c_{\mathrm{VF}} = 0.5$ and $c_{\mathrm{H}} = 0.01$, respectively. The objective is maximized with an Adam minibatch size of 4 and a learning rate of 2.5×10^{-4}. The algorithm is trained over a total of 10^7 time steps.

3.4. Benchmark execution strategies

3.4.1. Submit-and-leave execution

As a first benchmark, we employ a submit-and-leave strategy (Nevmyvaka et al., 2005, 2006). In the first time step, the strategy submits a limit order with the full volume v_0 at the best limit price observed in this time step. Specifically, for a sell volume, $v_0 > 0$, (buy volume, $v_0 < 0$), the limit order is placed at the best-ask price, i.e., at $x_0^{\text{sell}} = \text{ask}_0$ ($x_0^{\text{buy}} = \text{bid}_0$). This order stays unchanged and uncancelled until the end of the episode. If the limit order is not executed, all volume remaining at the final time step is executed with a market order.

3.4.2. Immediate market-order execution

As a second benchmark, we consider a strategy that immediately executes all volume v_0 at the initial time step using an appropriate market order. The market order matches with orders at the opposite side of the limit order book, thereby accepting potentially high liquidity costs.

3.4.3. Time-weighted market-order execution

As last benchmark strategy, we employ time-weighted market-order execution. This strategy executes a volume fraction of v_0/T in every time step as a market order. Under the assumption that price impact is linear in executed volume and in absence of short-term price predictability, this strategy is optimal (Bertsimas and Lo, 1998).

3.5. Study design and performance evaluation

We validate reinforcement learning models in a rolling-window forward-validation scheme (Tashman, 2000; Schnaubelt, 2019). Specifically, we train models on six months of training data, and apply models on the subsequent out-of-sample validation period of three months. We then roll forward the validation period by three months. Hence, the first validation period comprises the months July, August and September 2018. We use a total of four non-overlapping validation periods, i.e., train models in four runs for every setting of exchange, market and initial volume.

Each simulated episode in the execution environment corresponds to a block of consecutive observations from our data set. For the training of deep reinforcement learning algorithms, we randomly select a root observation from the data set to initialize the state of the environment at the beginning of an episode. As the agent steps forward in time, observations from the block of data following the root observation are used in the environment. For out-of-sample testing, we iterate over all observations in the validation period and select every observation exactly once as root observation to initialize the execution environment.[9] To evaluate the performance of execution strategies, we consider the average relative implementation shortfall, i.e., the total realized shortfall of an episode relative to the mid price at the start of the episode.

[9]To be precise, we do not select an observation as root observation during training or validation if observations are missing in the subsequent block of $T-1$ contiguous time steps.

4. Results

We present our results in four steps: First, we compare the out-of-sample performance of reinforcement learning algorithms and benchmark strategies (Section 4.1). Next, we perform in-depth analyses to shed light into the black box of deep reinforcement learning. To this end, we examine the executed volume profile of the strategies in Section 4.2. Third, we analyze the contribution of features in Section 4.3. To understand their decision making in greater detail, we finally examine the chosen actions conditional on individual feature values (Section 4.4).

4.1. Overview of empirical findings

First, we compare the out-of-sample performance of reinforcement learning algorithms with our benchmark execution strategies. To this end, we independently train the agents on our four training sets for different combinations of exchange, currency pair and initial trading volume v_0. Following Nevmyvaka et al. (2006), we select an episode length of $T = 4$. For the BTC/USD market, we calculate results for initial trading volumes v_0 of 10 BTC, 20 BTC and 50 BTC. On the one hand, these volumes correspond to approximately 25 to 300 percent of average minutely trading volume (compare Table 2), hence seem still in line with the approximations outlined in Section 3.1. On the other hand, these volumes exceed typical best-bid/best-ask order book depths (compare Table 3), hence their execution would typically incur high liquidity costs. Similarly, we select initial volumes of 100 ETH and 200 ETH for the ETH/USD and ETH/BTC markets. Then, we evaluate the trained agents in validation environments that are based on the respective test data sets.

4.1.1. Comparing algorithms by total out-of-sample shortfall

Panel A of Table 5 depicts mean realized implementation shortfalls including exchange commissions relative to the mid price at the beginning of the episode. For example, if the agent is asked to sell starting from an initial mid price $mid_0 = 5000$ USD, a realized implementation shortfall of -15 bp corresponds to an average execution price including exchange commissions of 4992.5 USD. As expected, executing all volume in the first time step with a market order (strategy IM) yields the worst realized shortfall in all settings, as the market order 'eats into the order book' to fill the requested volume. In comparison, splitting the market order equally over all T time steps (strategy TW) significantly reduces the realized implementation shortfall. For the example of selling $v_0 = 20$ BTC at the BitFinex BTC/USD market, the IM strategy results in a shortfall of -24.98 bp, which is reduced by 13.29 percent to -21.66 bp with TW execution. Averaging over all exchanges, the TW strategy reduces the total shortfall for 10 BTC by 7.24 percent and for 50 BTC by 19.57 percent when compared to the IM strategy. A further improvement over the TW strategy

Exchange	Market	Volume v_0	IM	TW	S&L	BQL	DDQN	PPO
Panel A: Mean relative shortfall, including all exchange commissions								
BitFinex	BTC/USD	-10 BTC	-22.8704	-21.0937	-16.8267	-16.9715	-16.8998	-15.7637
	(485984 obs.)	-20 BTC	-25.0609	-21.7078	-18.5535	-18.1985	-18.1207	-17.1031
		-50 BTC	-30.1776	-23.4662	-23.0463	-21.4878	-21.4081	-20.7630
		10 BTC	-22.8517	-21.0325	-16.8143	-16.9482	-16.9093	-15.7672
		20 BTC	-24.9760	-21.6577	-18.5294	-18.2759	-18.1973	-17.1093
		50 BTC	-30.0132	-23.3858	-22.9998	-21.5530	-21.3757	-21.1172
	ETH/BTC	100 ETH	-27.8538	-23.7508	-22.8057	-21.4498	-22.9759	-21.2470
	(482924 obs.)	200 ETH	-31.9487	-25.3424	-26.7186	-24.3196	-25.7515	-24.6528
	ETH/USD	100 ETH	-24.2229	-21.5278	-18.8991	-18.4362	-18.8688	-17.3880
	(478633 obs.)	200 ETH	-26.9632	-22.5119	-21.1788	-20.1197	-20.2406	-19.2230
Kraken	BTC/USD	10 BTC	-31.6315	-28.8502	-23.7581	-23.2141	-23.5195	-22.4810
	(435983 obs.)	20 BTC	-34.3564	-29.9004	-26.1868	-25.1410	-25.3096	-24.5956
		50 BTC	-40.9416	-32.3358	-32.1192	-30.0486	-30.1216	-30.1082
Coinbase	BTC/USD	10 BTC	-32.7858	-31.1540	-19.7012	-19.7012	-17.8253	-16.1073
	(518911 obs.)	20 BTC	-34.6456	-31.7403	-24.2416	-24.0410	-22.5267	-20.9857
		50 BTC	-39.4232	-33.2700	-32.3775	-30.5668	-30.5148	-29.4034
Panel B: Mean relative shortfall, excluding exchange commissions								
BitFinex	BTC/USD	-10 BTC	-2.8648	-1.0916	-2.3127	-2.2816	-1.3899	-1.3475
	(485984 obs.)	-20 BTC	-5.0510	-1.7044	-3.2922	-2.7487	-2.2311	-1.9357
		-50 BTC	-10.1577	-3.4594	-6.5148	-4.5511	-3.9597	-4.1483
		10 BTC	-2.8576	-1.0346	-2.2899	-2.2513	-1.3583	-1.3373
		20 BTC	-4.9863	-1.6610	-3.2626	-2.8805	-2.2582	-1.9664
		50 BTC	-10.0338	-3.3928	-6.4723	-4.6790	-4.3049	-4.2280
	ETH/BTC	100 ETH	-7.8695	-3.7583	-5.8203	-3.7579	-4.4381	-3.7656
	(482924 obs.)	200 ETH	-11.9727	-5.3531	-9.1389	-5.7677	-6.9593	-6.2346
	ETH/USD	100 ETH	-4.2314	-1.5308	-3.6133	-2.9864	-2.2410	-1.7288
	(478633 obs.)	200 ETH	-6.9772	-2.5169	-5.0705	-3.5393	-3.2318	-2.6883
Kraken	BTC/USD	10 BTC	-5.6462	-2.8576	-2.9150	-2.1588	-1.8293	-1.7620
	(435983 obs.)	20 BTC	-8.3782	-3.9106	-4.6408	-3.2633	-3.2253	-3.2507
		50 BTC	-14.9806	-6.3523	-9.4454	-7.3105	-7.4165	-7.4708
Coinbase	BTC/USD	10 BTC	-2.8001	-1.1598	-2.5853	-2.5853	-1.1749	-1.4980
	(518911 obs.)	20 BTC	-4.6670	-1.7493	-3.7709	-3.3530	-2.1598	-2.3942
		50 BTC	-9.4656	-3.2852	-7.8789	-5.2631	-4.8892	-5.5259

Table 5: **Comparison of execution strategies by realized shortfall.** This table compares the performance of execution strategies for different exchanges, markets and initial execution volumes v_0 for out-of-sample test data. All results are averaged over four model runs with each with individual training and test data sets. We depict results for immediate market-order execution (*IM*), time-weighted market-order execution (*TW*), submit-and-leave execution (*S&L*), backwards-induction Q-learning (*BQL*), deep double Q-networks (*DDQN*) and proximal policy optimization (*PPO*). Panel A shows the mean realized implementation shortfall including exchange commissions relative to the mid price at the beginning of the episode, in basis points. Panel B reports mean relative implementation shortfalls excluding exchange commissions, in basis points.

is achieved by the submit-and-leave strategy (S&L), which submits a single limit order at the start of the episode and executes all volume remaining in the last time step as a market order. This way, part of the volume is executed in limit orders at lower commissions compared to market orders, which reduces the total implementation shortfall on average by 11.45 percent over the TW execution strategy. Switching to the more flexible backwards-induction Q-learning approach (BQL) results in a marginal improvement of 3.51 percent over the static S&L strategy.

Turning to the performance of deep reinforcement learning agents, we find that deep double Q-networks (DDQN) achieve an average performance improvement of 3.61 percent over the S&L strategy. In comparison, proximal policy optimization (PPO) reduces the overall implementation shortfall by an average of 8.53 percent over the S&L strategy. For executing a volume of 20 BTC at the Coinbase exchange for the currency pair BTC/USD, the realized shortfall with the PPO agent is at -20.9857 bp, a significant improvement when compared to the value of -34.6456 bp for IM execution and -22.5267 bp for the second-best result realized with DDQN. We find that implementation shortfalls for selling and buying are very similar. For example, buying a volume of 10 BTC at the BitFinex BTC/USD market with the PPO agent results in a total shortfall of -15.7637 bp, while selling results in nearly the same value of -15.7672 bp. As bid- and ask-side statistics of limit order books exhibit a large degree of symmetry (compare, for example, Biais et al., 1995; Potters and Bouchaud, 2003; Schnaubelt et al., 2019), similar behaviors for buying and selling are unsurprising. We also observe that the relative performance improvement of strategies involving limit orders declines with increasing execution volumes. While using the PPO agent to execute 10 BTC results in an average shortfall reduction of 37.71 percent over IM execution, trading 20 BTC (50 BTC) results in a relative improvement of 26.95 (23.72) percent.

The complexity of deep reinforcement learning algorithms makes it difficult to trace the observed performance differences. We hypothesize that the performance improvement of PPO over DDQN might be due to several factors: First, in cases where the policy function is easier to approximate than the action-value function, PPO as a policy gradient method has an advantage over value-based algorithms such as DDQN (Sutton and Barto, 2018, p. 266). We conjecture that the action-value function might be difficult to approximate in our case because of the high degree of noise in limit order data. Second, policy gradient methods are able to learn stochastic policies, which could, compared to the ϵ-greedy policy of DDQN, result in a better exploration of the available action space. Third, PPO explicitly targets robustness, fast convergence and high data efficiency (Schulman et al., 2017b), which might represent a key advantage for applications with noisy financial data, such as the limit order placement problem studied here. As "the design of an ANN [artificial neural network] is more of an art than a science" (Zhang et al., 1998, p. 42) and we have not been able to tune hyperparameters of neither the DDQN nor the PPO algorithm because of their computationally very expensive training, we conjecture that there might be parameter settings with even better performance.

The economic significance of shortfall differences between limit order placement strategies becomes apparent when comparing shortfall reductions to average excess returns of statistical arbitrage strategies. In comparison to the S&L strategy, which does not require training and is easily implemented, PPO reduces average execution costs by roughly 2 bp. Taking the random forest-based cryptocurrency trading strategy of Fischer et al. (2019) as an example, typical mean returns per trade are at 3.8 bp after transaction costs. A reduction of transaction costs by 2 bp would increase mean excess returns by the same amount to 5.8 bp, corresponding to a performance improvement of the strategy of over 50 percent. Hence, even relatively small reductions of execution costs may greatly impact the financial performance of high-turnover trading strategies.

4.1.2. The role of exchange commissions

To study the role of exchange commissions for total shortfall results, Panel B of Table 5 reports raw shortfalls, i.e., realized shortfalls before including exchange commissions. Comparing these results to the results of Panel A, i.e., including exchange commissions, we find that the larger part of total realized implementation shortfalls is caused by exchange commissions. For smaller execution volumes of 10 BTC, roughly ten percent of execution costs are due to the raw implementation shortfall, i.e., are related to realized price differences. The majority of costs can be attributed to exchange commissions. For larger volumes, the fraction of shortfall that is due to exchange commissions is still well above 50 percent. Hence, the avoidance of exchange commissions is a major factor in the reduction of total implementation shortfalls at all studied cryptocurrency exchanges. As exchange commissions are part of the agent's reward function, reinforcement learning models are trained to reduce the total implementation shortfall, rather than the raw shortfall. Nevertheless, we find similar rankings of algorithms before and after the inclusion of exchange commissions. In many cases, PPO realizes the lowest magnitude in implementation shortfall before exchange commissions among all reinforcement learning algorithms. There is one exception in the similarity of strategy rankings: Time-weighted execution with market orders (TW) yields the best results in terms of raw shortfall. However, as this strategy executes all volume in market orders incurring higher exchange commissions, this advantage vanishes after exchange commissions. In summary, we note that – due to the elevated role of exchange commissions in overall shortfalls – differences in total shortfalls between strategies are driven by differences in realized exchange commissions rather than differences in raw implementation shortfalls. Hence, a large part of the reduction of shortfalls is due to a better use of limit orders, for example, by achieving better execution prices or a higher probability of limit order execution.

4.1.3. Fraction of limit order execution

To further analyze the latter reason, we calculate the fraction of volume that is executed with limit orders, and present results in Table 6. We make the following observations: First, with increasing initial volume v_0, the fraction of volume executed in limit orders decreases. We conjecture

Exchange	**Market**	**Volume v_0**	**S&L**	**BQL**	**DDQN**	**PPO**
BitFinex	BTC/USD	10 BTC	0.5476	0.5303	0.4449	0.5570
		20 BTC	0.4733	0.4605	0.4061	0.4857
		50 BTC	0.3473	0.3126	0.2929	0.3111
	ETH/BTC	100 ETH	0.3015	0.2308	0.1462	0.2519
		200 ETH	0.2420	0.1448	0.1208	0.1582
	ETH/USD	100 ETH	0.4714	0.4550	0.3372	0.4341
		200 ETH	0.3892	0.3420	0.2991	0.3465
Kraken	BTC/USD	10 BTC	0.5157	0.4945	0.4310	0.5281
		20 BTC	0.4454	0.4122	0.3916	0.4655
		50 BTC	0.3326	0.3262	0.3295	0.3363
Coinbase	BTC/USD	10 BTC	0.4295	0.4295	0.4450	0.5130
		20 BTC	0.3176	0.3104	0.3211	0.3803
		50 BTC	0.1834	0.1565	0.1458	0.2041

Table 6: **Fraction of volume executed with limit orders.** This table depicts the average fraction of volume executed with limit orders instead of immediate market order execution. We depict results for submit-and-leave execution (*S&L*), backwards-induction Q-learning (*BQL*), deep double Q-networks (*DDQN*) and proximal policy optimization (*PPO*), for different exchanges, markets and initial volumes v_0. Results for immediate and time-weighted execution are not shown as they execute all volume in market orders, thus exhibit a zero limit order fraction.

that liquidity is insufficient to execute larger volumes in limit orders over a relatively short time frame. Second, for the BTC/USD currency pair, the BitFinex exchange exhibits the highest fraction of limit orders. Looking at the descriptive statistics in Tables 2 and 3, BitFinex also has the tightest bid-ask spread and the highest trade volume and frequency, which might well explain a higher probability of limit order execution. Also, the reduction in the fraction of limit order execution with increasing volume is less pronounced for BitFinex than for Coinbase. Third, the S&L strategy, which submits orders at the best-ask price, executes a larger share of volume in limit orders than the BQL or DDQN strategies, and in some cases also more than the PPO agent. Yet, PPO execution still yields superior overall results in most of these cases, for example in executing 20 BTC at the Coinbase BTC/USD market (Panel A of Table 5). We conjecture that PPO compensates the lower limit order volume by posting limit orders at less aggressive prices, which are adapted in every time step and potentially better reflect the expected probability of execution given the current market state.[10]

4.1.4. Robustness of empirical results

For the majority of settings in Table 5, PPO outperforms the other order placement strategies. In only two out of 16 cases, the realized shortfall after commissions for PPO is slightly below the one of BQL. The ranking of methods and the dominance of PPO is also evident when considering all validation subperiods separately. Table 7 depicts average rank of execution strategies by exchange and market. In three out of five exchange-market configurations, PPO achieves an average rank of

[10]We will further investigate the limit order prices chosen by the agent in Section 4.4.

Exchange	Market	Count	IM	TW	S&L	BQL	DDQN	PPO
BitFinex	BTC/USD	24	6.00	4.92	3.58	2.88	2.62	1.00
	ETH/BTC	8	6.00	3.88	3.88	1.62	3.62	2.00
	ETH/USD	8	6.00	4.88	3.88	2.12	3.12	1.00
GDAX	BTC/USD	12	6.00	4.92	3.88	3.04	2.17	1.00
Kraken	BTC/USD	12	6.00	4.83	3.92	1.92	2.92	1.42

Table 7: **Mean ranking of strategies in subperiods by exchange-market configuration.** This table depicts the average ranks of all execution strategies according to their realized shortfall including commissions by exchange and market. For every row, we average ranks over all initial volumes and over all validation subperiods. A rank of 1.0 indicates that the strategy yields the best result in all subperiods and for all initial volumes.

	Mean relative shortfall, excluding commissions	Mean relative shortfall, including commissions
Panel A: Seed robustness		
Mean	-1.9415	-17.1310
Std. Dev.	0.0207	0.0274
Minimum	-1.9757	-17.1807
Maximum	-1.9192	-17.0900
Panel B: Alternative hyperparameter settings		
Baseline configuration	-1.9664	-17.1093
Two hidden layers of 128 neurons	-1.9791	-17.1516
Two hidden layers of 32 neurons	-1.8745	-17.1850
Trajectory length $N = 256$	-1.9340	-17.2434
Trajectory length $N = 64$	-2.0319	-17.2301

Table 8: **PPO robustness.** This table presents robustness checks for results obtained with proximal policy optimization and the BitFinex BTC/USD market and $v_0 = 20\,\text{BTC}$. Panel A depicts descriptive statistics from 10 runs of the PPO agent (i.e., training and out-of-sample testing for all subperiods) with different consecutive seed values of the random number generator. Panel B shows results from PPO runs with different hyperparameter settings.

1.0, indicating that PPO always performs best, i.e., yields the lowest shortfall magnitude including commissions for all volume settings and in all four train-test splits of the data. In the two remaining exchange-market configurations, the average rank of PPO is still fairly low. Over all exchange-market-subperiod combinations, PPO achieves the highest rank in terms of total out-of-sample shortfall in 89.06 percent of cases.

Next, we check whether the superior performance of PPO is due to a favorable choice of seed value or hyperparameter configuration. First, we train the PPO agent 10 times with consecutive seed values of the random number generator for all subperiods, and evaluate the distribution of out-of-sample shortfalls. From the results in Panel A of Table 8, we observe that a variation of seed value leads to small variations in shortfalls only. Second, we investigate the sensitivity of results to changes of the hyperparameter configuration. To this end, we retrain the PPO agent with different numbers of neurons in the hidden layers and different trajectory lengths (Panel B of Table 8). We observe that, despite larger variations than obtained from the seed study, results are still fairly similar to the baseline results that use the algorithm's default hyperparameters.

4.2. Analyzing the profile of traded volume

Next, we analyze the volume traded in every time step to further explore performance differences between strategies. Figure 2 depicts the distribution of the executed volume fraction, i.e., $\omega_t^{\mathrm{ex}}/v_0$, for every time step, averaged over all exchanges and for the BTC/USD currency pair. Results are shown for different initial volumes v_0 (rows) and for all models involving limit orders (columns). We observe several common patterns in the profiles of mean executed volume fractions: First, a large fraction of volume is executed in the final time step, which corresponds to the execution of remaining volume in a market order. Second, comparing the fraction of volume in the first three time steps, typically most volume is executed in the first step, where all initial volume is still unexecuted. Third, the volume executed in the final time step increases with initial volume v_0, and correspondingly, the volume fraction in the first time steps tends to decrease with v_0. Both observations can be explained by the limited liquidity in the first three time steps. Turning to differences between execution strategies, we first note that the S&L strategy executes a lower fraction of the volume in the first three steps than the deep reinforcement learning agents PPO and DDQN. While the mean volume fraction of the S&L strategy is positive, the median volume fraction is zero in all of the first three time steps. The DDQN and PPO agents exhibit fairly similar volume profiles, where the largest fraction of volume is traded in the first time step.

4.3. Importance of feature sets

We proceed with an analysis of the contribution of the features available to the PPO agent. To this end, we retain the PPO algorithm in environments with different features and calculate respective mean total realized shortfalls for the BTC/USD currency pair for selling an initial volume of $v_0 = 20\,\mathrm{BTC}$. Also, we calculate a feature importance score from the inverse distance to the total shortfall using the full feature set. Table 9 presents the results for different exchanges.

We make the following observations: First, we find that the most important features are related to the current shape of the limit order book. For the BitFinex exchange, by far the most important features are related to queue imbalances, i.e., volume differences between the bid and ask sides of the limit order book ($BO\text{-}IMBAL$ and $Q\text{-}IMBAL(N_p)$). Second and third important are features on cumulated order volumes ($VOL\text{-}BID$ and $CVOL\text{-}BID(N_p)$) and liquidity costs related to the bid side ($LC\text{-}BID(V)$). For the exchanges Kraken and Coinbase, the same feature sets appear among the most important ones. However, liquidity costs related to the bid side ($LC\text{-}BID(V)$) now lead to the largest reduction in shortfalls. The least important features are related to past trade imbalances ($TC\text{-}IMBAL$ and $TV\text{-}IMBAL$), the bid-ask spread ($BA\text{-}SPREAD$) or past realized volatility ($VOLA$). Second, we find bid-side features to be more important than ask-side features.

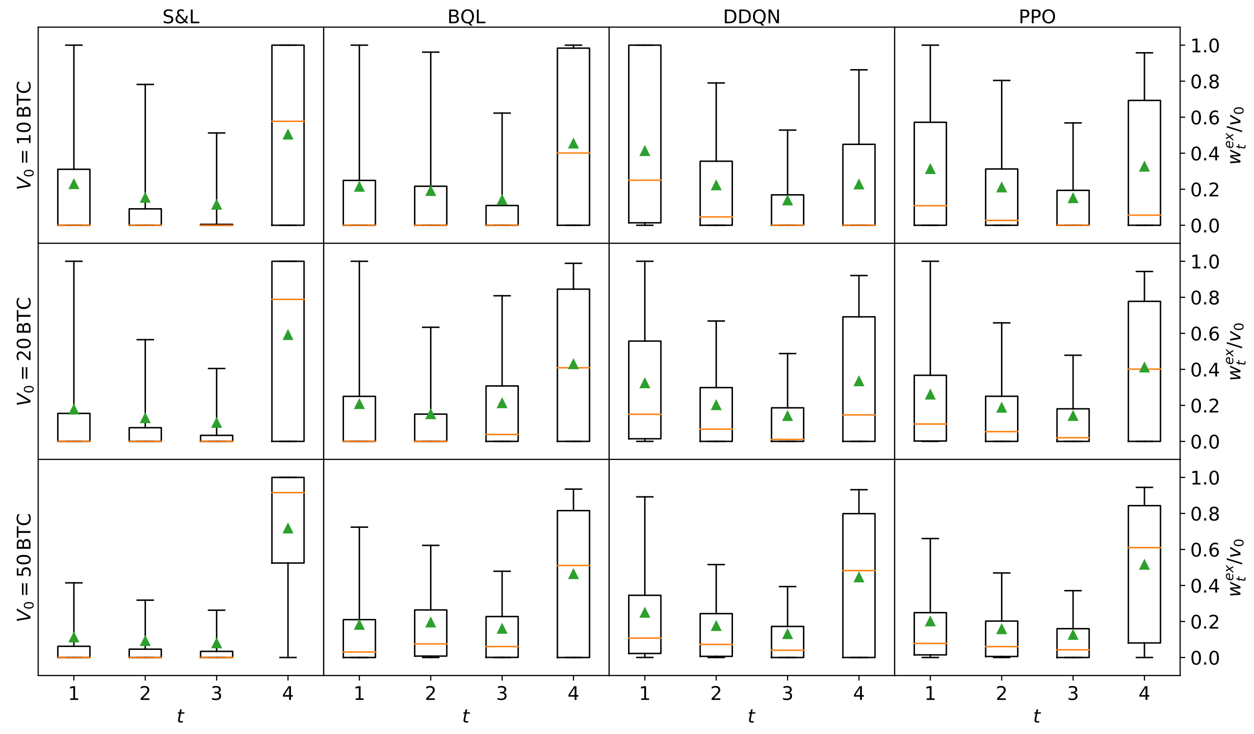

Figure 2: **Fraction of traded volume by time step.** This plot depicts the distribution of executed volume by time step (horizontal axis), initial volume v_0 (rows) and execution strategy (columns). Results for submit-and-leave execution (*S&L*), backwards-induction Q-learning (*BQL*), deep double Q-networks (*DDQN*) and proximal policy optimization (*PPO*) are shown. We calculate the fraction of traded volume $\omega_t^{\mathrm{ex}}/v_0$ for the BTC/USD market averaged over all exchanges. Box plots display interquartile ranges, medians (orange lines), means (green triangles), and 10/90 percent quantiles (whiskers).

For the BitFinex exchange, cumulated bid volume features ($VOL\text{-}BID$ and $CVOL\text{-}BID(N_p)$) result in a total shortfall of -17.53 bp, compared to a value of -17.96 bp for the ask-side features ($VOL\text{-}ASK$ and $CVOL\text{-}ASK(N_p)$). We make similar observations for the other two exchanges. Third, we find a further reduction of total shortfall when providing all features to the agent, as the the PPO agent seems to be able to successfully exploit a larger feature set in learning suitable policies. For example, with the best single feature set for the Coinbase exchange, bid-side liquidity ($LC\text{-}BID(V)$), total shortfall is at 21.1852 bp, which is further improved to -20.9857 bp (see Table 5) when providing all features to the agent.

4.4. *Exploring the agent's decision making*

Finally, we analyze the impact of market state variables on the decision making of the PPO agent. To this end, we evaluate the agent's action conditional on the value of individual features. Specifically, we compute average action values for all observations with feature values in the respective quantile intervals. Figure 3 plots the mean difference of the chosen limit price to the current best-ask price, in units of the tick size, as a function of a feature's value for selling different volumes

	Total shortfall			**Importance score**		
Feature(s)	BitFinex	Kraken	Coinbase	BitFinex	Kraken	Coinbase
TC-IMBAL	-18.1673	-25.2428	-24.0553	0.0605	0.0757	0.0331
TV-IMBAL	-18.4534	-25.2474	-24.0613	0.0476	0.0752	0.0330
BO-IMBAL, *Q-IMBAL*(N_p)	-17.3317	-25.0991	-22.0463	0.2877	0.0973	0.0958
VOL-BID, *CVOL-BID*(N_p)	-17.5296	-25.1236	-22.4134	0.1523	0.0928	0.0712
VOL-ASK, *CVOL-ASK*(N_p)	-17.9558	-25.2673	-22.8120	0.0756	0.0729	0.0556
VOLA	-18.2896	-25.3367	-24.0051	0.0542	0.0661	0.0336
DRIFT	-18.1472	-25.2615	-23.8873	0.0617	0.0736	0.0350
LC-BID(V)	-17.6075	-24.7595	-21.1852	0.1285	0.2988	0.5093
LC-ASK(V)	-18.0681	-25.2360	-22.0444	0.0668	0.0765	0.0960
BA-SPREAD	-18.0901	-25.2836	-23.7010	0.0653	0.0712	0.0374

Table 9: **Importance of features.** This table depicts results from an analysis on the importance of features on proximal policy optimization performance. We retrain the PPO agent in environments with different features and calculate respective mean total realized shortfalls (columns *Total shortfall*) for the BTC/USD currency pair traded at the indicated exchanges for selling an initial volume of $v_0 = 20\,\text{BTC}$. Columns *Importance score* calculate a feature importance score from the inverse distance to the total shortfall using the full feature set.

at the BTC/USD market. Negative values indicate a more aggressive posting of orders, i.e., setting the limit price below the current best-ask price.

Looking first at features related to the current volume present in the first ten price steps of the order book (top row of Figure 3), we find that sell pressure, i.e., an overweight of order volume in the ask side of the order book, gives rise to a more aggressive placement of the agent's own sell order. By contrast, positive values of the queue imbalance ($Q\text{-}IMBAL(10) > 0$), indicative of a larger bid than ask volume, is followed by less aggressive orders. A similar behavior is evident when looking at the volumes at both sides of the order book separately (*CVOL-BID*(10) and *CVOL-ASK*(10)), with a larger influence of the opposite (bid) side of the order book. We can understand these results by interpreting the queue imbalance as a short-term predictor of mid-price returns (compare, for example, Cao et al., 2009; Gould and Bonart, 2016, for an empirical investigation of the influence of imbalances on price): The smaller the value of the queue imbalance, the higher is the supply overweight in the order book, which might indicate a downshift of the mid price over the subsequent time period. Consequently, we find the agent to place orders more aggressively to ensure execution of its own limit order.

We find a similar influence of the cost of immediate execution on the agent's decision making (middle row of Figure 3), which could likely be due to a high degree of correlation between features that are based on the current order book. For example, low immediate execution costs of the ask side of the order book (*LC-ASK*) tend to indicate high cumulated volumes in the first ask-side price steps, which again result in a more aggressive order placement.

The chosen action is fairly independent of the bid-ask spread (*BA-SPREAD*). This could be a consequence of a high level of noise in this variable, generated for example by frequent jumps of best-

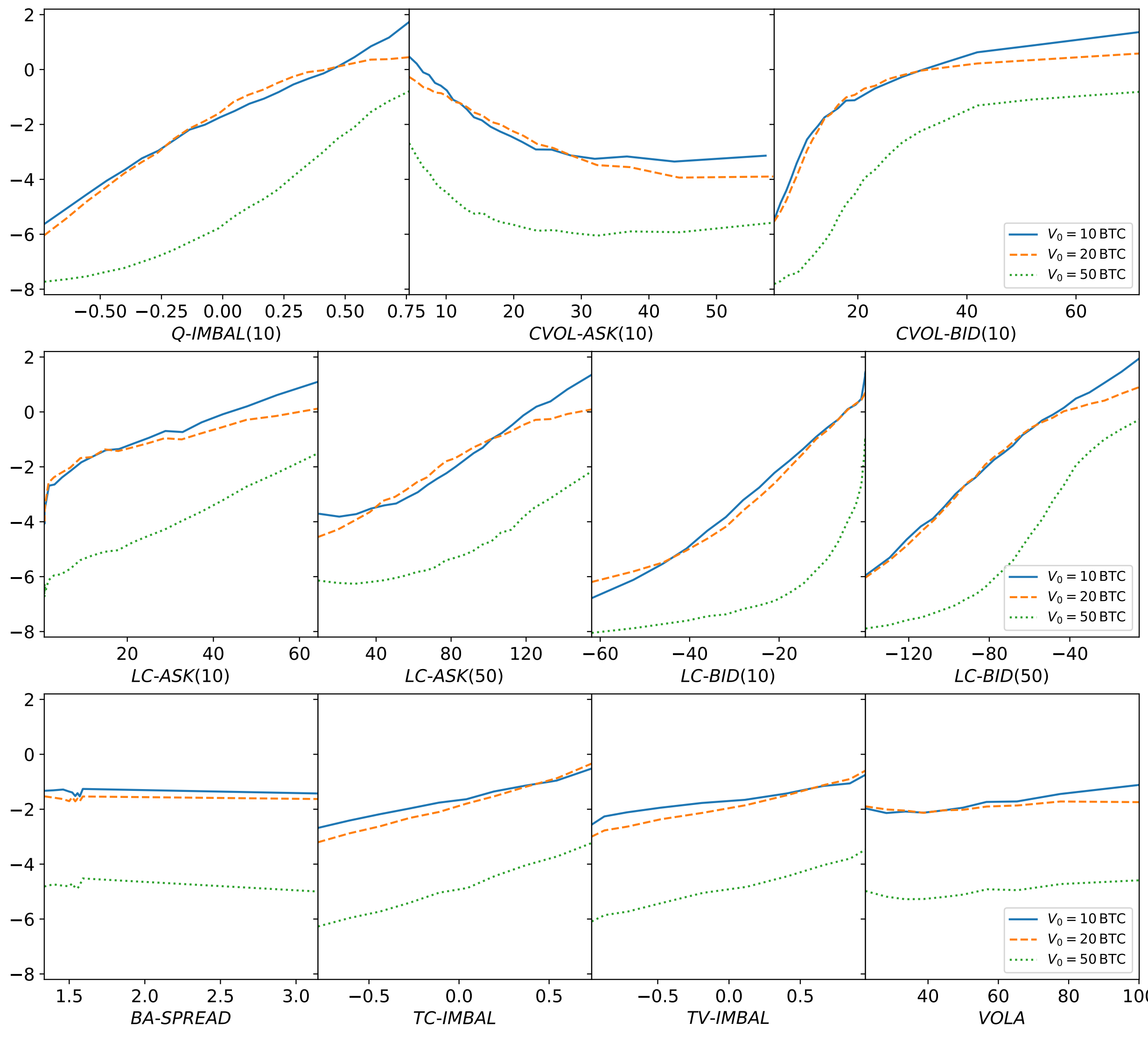

Figure 3: **Action limit price conditional on feature value.** This figure depicts the limit price selected by proximal policy optimization conditional on feature values for selling an initial volume of $v_0 = 10, 20, 50$ BTC, for the BitFinex BTC/USD market. The limit price (vertical axis) is shown relative to the current best-ask price in units of the tick size. Feature values are shown prior to rolling-window standardization. The top row depicts features related to the limit order volume in the first ten steps of the limit order book ($CVOL\text{-}BID(10)$, $CVOL\text{-}ASK(10)$: cumulated order volume in the ten best-bid/best-ask steps, $Q\text{-}IMBAL(10)$: queue imbalance), the middle row shows features related to current levels of liquidity costs ($LC\text{-}BID(V)$, $LC\text{-}ASK(V)$: cost of immediate execution for a volume of V, in basis points). The bottom row displays the bid-ask spread relative to the mid price ($BA\text{-}SPREAD$, in basis points), past trade count and volume imbalances ($TC\text{-}IMBAL$, $TV\text{-}IMBAL$) as well as past realized mid-price volatility ($VOLA$).

ask and best-bid prices, which renders it an unattractive feature for the model. The effect of the trade count imbalance ($TC\text{-}IMBAL$) and the trade volume imbalance ($TV\text{-}IMBAL$) on the chosen limit price is similar to the effect of the queue imbalance, however with a lower magnitude. Higher levels of past realized mid-price volatility ($VOLA$) slightly decrease the agent's order aggressiveness, potentially because less aggressive orders have a higher probability of execution in times of high volatility.

Generally, the shape of the feature-action dependence is – up to a shift towards more aggressive orders for larger volumes – fairly similar across different levels of executed volume v_0. While conditional actions for 10 BTC and 20 BTC are similar and the larger volume of 20 BTC exhibits only a slight tendency towards more aggressive orders, PPO places significantly more aggressive orders for a volume of 50 BTC. Comparing the relative price tick of the market at the average trade price (0.14 bp at 7000 BTC, compare Table 1) to average bid-ask spreads (0.679 bp, Table 3), we find that typical order placement occurs a few price ticks inside the bid-ask spread.

5. Conclusion

With this paper, we have successfully demonstrated that deep reinforcement learning, especially proximal policy optimization, may be used to optimize the placement of limit orders in cryptocurrency limit order markets. Leveraging a large high-frequency data set of 300 million historic trades and more than 3.5 million order book states from major exchanges and currency pairs, we empirically compare state-of-the-art deep reinforcement learning algorithms to several simpler benchmarks. We contribute to the existing literature in the following ways: First, we describe in-depth how deep reinforcement learning can be applied to the problem of optimal limit order placement. In the simulated environment, the agent is asked to decide on the limit price of limit orders over the course of several time steps, and is rewarded based on the realized shortfall. To inform the agent about its environment, we handcraft several features based on the literature. Second, we compare the out-of-sample performance of two state-of-the-art deep reinforcement learning algorithms to several benchmarks. In comparison to deep double Q-networks as well as all static and dynamic benchmark strategies, we find proximal policy optimization to reliably learn a superior order placement policy. Applied to out-of-sample data, this policy reduces average total implementation shortfalls by 37.71 percent over an execution in a single market order. Third, we perform in-depth analyses to ask whether the PPO's actions are in line with economic rationale. We find the policy to preferably execute volume in limit orders to avoid additional exchange fees for market orders. Also, the most important features are related to volumes at the first price steps of the limit order book and queue imbalances – well in line with an interpretation of queue imbalances as short-term predictors of mid-price returns. Finally, PPO's order placement tends to be more aggressive in expectation of unfavorable price movements.

Our study reveals several interesting directions for future research: First, our research design may be used in future empirical studies applying deep reinforcement learning for optimal order placement in other limit order exchanges, for example stock exchanges. Second, another interesting extension could be to move beyond handcrafted features. For example, convolutional neural networks or the neural network architecture of Sirignano (2019) could be used to directly learn suitable feature representations from the data. Third, the approach could be extended to adjacent problems with practical relevance, such as routing orders to multiple exchanges or deciding between different order types such as iceberg orders. These extensions could be integrated easily into the action space and reward function of our environment.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL: https://www.tensorflow.org/.

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., Zhang, L., 2019. Solving Rubik's Cube with a robot hand. arXiv 1910.07113.

Almgren, R., Chriss, N., 2001. Optimal execution of portfolio transactions. The Journal of Risk 3, 5–39.

Bao, W., Liu, X.y., 2019. Multi-agent deep reinforcement learning for liquidation strategy analysis. arXiv 1906.11046.

Bayraktar, E., Ludkovski, M., 2014. Liquidation in limit order books with controlled intensity. Mathematical Finance 24, 627–650.

Bertsimas, D., Lo, A.W., 1998. Optimal control of execution costs. Journal of Financial Markets 1, 1–50.

Biais, B., Hillion, P., Spatt, C., 1995. An empirical analysis of the limit order book and the order flow in the Paris Bourse. The Journal of Finance 50, 1655–1689.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. OpenAI Gym. arXiv 1606.01540.

Cao, C., Hansch, O., Wang, X., 2009. The information content of an open limit-order book. Journal of Futures Markets 29, 16–41.

Cartea, A., Jaimungal, S., 2015. Optimal execution with limit and market orders. Quantitative Finance 15, 1279–1291.

Cartea, A., Jaimungal, S., Penalva, J., 2015. Algorithmic and high-frequency trading. Cambridge University Press, Cambridge, UK.

Cheng, A.T., 2017. AI jumps into dark pools. URL: https://www.institutionalinvestor.com/article/b15yx290rz5pcz/ai-jumps-into-dark-pools (visited 14/03/2020).

Cont, R., Kukanov, A., 2017. Optimal order placement in limit order markets. Quantitative Finance 17, 21–39.

Cont, R., Kukanov, A., Stoikov, S., 2014. The price impact of order book events. Journal of Financial Econometrics 12, 47–88.

Cummings, J.R., Frino, A., 2010. Further analysis of the speed of response to large trades in interest rate futures. Journal of Futures Markets 30, 705–724.

Daberius, K., Granat, E., Karlsson, P., 2019. Deep execution – value and policy based reinforcement learning for trading and beating market benchmarks. SSRN Scholarly Paper ID 3374766. Social Science Research Network. Rochester, NY. URL: https://papers.ssrn.com/abstract=3374766.

Danielsson, J., Payne, R., 2001. Measuring and explaining liquidity on an electronic limit order book: Evidence from Reuters D2000-2. SSRN Scholarly Paper ID 276541. Social Science Research Network. Rochester, NY. URL: https://papers.ssrn.com/abstract=276541.

Degryse, H., Jong, F.D., Ravenswaaij, M.V., Wuyts, G., 2005. Aggressive orders and the resiliency of a limit order market. Review of Finance 9, 201–242.

Fischer, T.G., Krauss, C., Deinert, A., 2019. Statistical arbitrage in cryptocurrency markets. Journal of Risk and Financial Management 12, 31.

Gomber, P., Schweickert, U., Theissen, E., 2015. Liquidity dynamics in an electronic open limit order book: An event study approach. European Financial Management 21, 52–78.

Gopikrishnan, P., Plerou, V., Gabaix, X., Stanley, H.E., 2000. Statistical properties of share volume traded in financial markets. Physical Review E 62.

Gould, M.D., Bonart, J., 2016. Queue imbalance as a one-tick-ahead price predictor in a limit order book. Market Microstructure and Liquidity 2, 1650006.

Gould, M.D., Porter, M.A., Williams, S., McDonald, M., Fenn, D.J., Howison, S.D., 2013. Limit order books. Quantitative Finance 13, 1709–1742.

van Hasselt, H., Guez, A., Silver, D., 2015. Deep reinforcement learning with double Q-learning. arXiv 1509.06461.

Hendricks, D., Wilcox, D., 2014. A reinforcement learning extension to the Almgren-Chriss model for optimal trade execution, in: 2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr), pp. 457–464.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., 2018. Stable baselines. URL: https://github.com/hill-a/stable-baselines.

Kearns, M., Nevmyvaka, Y., 2013. Machine learning for market microstructure and high frequency trading, in: Easley, D., de Prado, M.L., O'Hara, M. (Eds.), High frequency trading: New realities for traders, markets, and regulators. Risk Books, London, UK.

Kingma, D.P., Ba, J., 2017. Adam: A method for stochastic optimization. arXiv 1412.6980.

Lam, S.K., Pitrou, A., Seibert, S., 2015. Numba: A LLVM-based Python JIT compiler, in: Proceedings of the 2nd Workshop on the LLVM Compiler Infrastructure in HPC, pp. 1–6.

McKinney, W., 2010. Data structures for statistical computing in Python, in: Proceedings of the 9th Python in Science Conference, pp. 51–56.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. arXiv 1602.01783.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing Atari with deep reinforcement learning. arXiv 1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.

Nevmyvaka, Y., Feng, Y., Kearns, M., 2006. Reinforcement learning for optimized trade execution, in: Proceedings of the 23rd International Conference on Machine Learning, ACM, New York, NY, USA. pp. 673–680.

Nevmyvaka, Y., Kearns, M., Papandreou, A., Sycara, K., 2005. Electronic trading in order-driven markets: Efficient execution, in: Proceedings of the seventh IEEE International Conference on

E-Commerce Technology (CEC'05), pp. 190–197.

Ning, B., Ling, F.H.T., Jaimungal, S., 2018. Double deep Q-learning for optimal execution. arXiv 1812.06600.

Noonan, L., 2017. JPMorgan develops robot to execute trades. URL: https://www.ft.com/content/16b8ffb6-7161-11e7-aca6-c6bd07df1a3c (visited 19/09/2018).

Obizhaeva, A.A., Wang, J., 2013. Optimal trading strategy and supply/demand dynamics. Journal of Financial Markets 16, 1–32.

Patel, Y., 2018. Optimizing market making using multi-agent reinforcement learning. arXiv 1812.10252.

Perold, A.F., 1988. The implementation shortfall: Paper versus reality. Journal of Portfolio Management 14, 4.

Plerou, V., Gopikrishnan, P., Gabaix, X., Stanley, H.E., 2002. Quantifying stock-price response to demand fluctuations. Physical Review E 66, 027104.

PostgreSQL Global Development Group, 2018. PostgreSQL database management system. URL: www.postgresql.org.

Potters, M., Bouchaud, J.P., 2003. More statistical properties of order books and price impact. Physica A: Statistical Mechanics and its Applications 324, 133–140.

Python Software Foundation, 2016. Python 3.5.2. URL: https://docs.python.org/3.5/.

Ranaldo, A., 2004. Order aggressiveness in limit order book markets. Journal of Financial Markets 7, 53–74.

Rantil, A., Dahlen, O., 2018. Optimized trade execution with reinforcement learning. Master's thesis. Linkoeping University. Sweden. URL: https://www.semanticscholar.org/paper/Optimized-Trade-Execution-with-Reinforcement-med-Rantil-Dahl%C3%A9n/fff05d2f0f414eead861a251aeff77f706804f6f.

Schnaubelt, M., 2019. A comparison of machine learning model validation schemes for non-stationary time series data. Discussion Papers in Economics 11/2019. Friedrich-Alexander-Universität. Erlangen/Nürnberg, Germany. URL: https://www.iwf.rw.fau.de/files/2019/11/11_2019.pdf.

Schnaubelt, M., Rende, J., Krauss, C., 2019. Testing stylized facts of Bitcoin limit order books. Journal of Risk and Financial Management 12, 25.

Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P., 2017a. Trust region policy optimization. arXiv 1502.05477.

Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P., 2018. High-dimensional continuous control using generalized advantage estimation. arXiv 1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017b. Proximal policy optimization algorithms. arXiv 1707.06347.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D., 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362, 1140–1144.

Sirignano, J.A., 2019. Deep learning for limit order books. Quantitative Finance 19, 549–570.

Sutton, R.S., Barto, A.G., 2018. Reinforcement learning: An introduction. 2nd ed., MIT Press, Cambridge, MA.

Tashman, L.J., 2000. Out-of-sample tests of forecasting accuracy: An analysis and review. International Journal of Forecasting 16, 437–450.

Terekhova, M., 2017. JPMorgan takes AI use to the next level. URL: https://www.businessinsider.de/jpmorgan-takes-ai-use-to-the-next-level-2017-8?r=US&IR=T (visited 19/09/2018).

Tsoukalas, G., Wang, J., Giesecke, K., 2017. Dynamic portfolio execution. Management Science 65, 2015–2040.

Van Der Walt, S., Colbert, S.C., Varoquaux, G., 2011. The NumPy array: A structure for efficient numerical computation. Computing in Science & Engineering 13, 22–30.

Varazzo, D., 2011. psycopg2. URL: http://initd.org/psycopg/.

Watkins, C.J.C.H., 1989. Learning from delayed rewards. PhD Thesis. King's College, University of Cambridge.

Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8, 229–256.

Zhang, G., Eddy Patuwo, B., Y. Hu, M., 1998. Forecasting with artificial neural networks: The state of the art. International Journal of Forecasting 14, 35–62.