Master's Thesis

Optimal Order Placement when trading bitcoins at orderbook level, using Reinforcement Learning

Axel Perschmann

June 20, 2017

Albert-Ludwigs-University Freiburg im Breisgau
Faculty of Engineering
Department of Computer Science
Machine Learning Lab

Writing period

12. 01. 2016 – 07. 31. 2017

Examiners

Dr. Joschka Bödecker

Dr. Frank Hutter

External Adviser:

Manuel Blum, *Psiori GmbH*

Declaration

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Freiburg, June 20, 2017

Place, date

Axel Perschmann

Acknowledgements

I like to thank the following people for their help and support:

- Manuel Blum, who directed me and provided me with great feedback throughout the whole project.
- All employees of the data science company *PSIORI GmbH*, for the pleasant working atmosphere and their occasional feedback on my topic.

Abstract

English abstract

Zusammenfassung

Deutsche Zusammenfassung

Contents

1 Introduction

1.1 Motivation

In the domain of computational finance, much research is performed to find and improve algorithms that help maximize revenue. One possibility to maximize revenue is to minimize inevitably occurring costs.

In the first place, investors participating in exchange markets, must expect fees, charged by the respective market place organizer in return for granting access to their infrastructure. Additionally, there are hidden costs to be considered as well. Most markets function after the microeconomic supply and demand [**todo**] model, where a universe of opposing trading interests determines the current price of a commodity.

While trades with little capital (relative to the whole market liquidity) usually cause minor impact on the current market situation, large-scale investors must be cautious when it comes to order placement. Large orders can have a major impact on supply and demand, which leads to diminishing availability, worsening prices and as such this so called slippage must be seen as hidden costs.

Well considered trading strategies help large-scale investors to reduce their impact and avoid costly market turbulences by unwinding large orders of shares over time. Nevmyvaka et al. [1] applied reinforcement learning to optimally distribute the trading activity over a fixed time horizon.

1.2 Objectives

This thesis tackles the important problem of optimized trade execution, which frequently occurs in the domain of financial computing. In its simplest form, the problem is defined by a particular financial instrument (here: bitcoins), which must be bought or sold within a fixed time horizon, while minimizing the expenditure (share price) for doing so.

The scope of this thesis is to transfer Nevmyvakas [1] reinforcement learning approach from traditional stock markets with expensive, proprietary data sources, to the relatively young market of bitcoin trading and to improve it's general ability to solve the important problem of optimized trade execution. In contrast to their experiments this thesis builds on inexpensive, publicly retrievable bitcoin exchange data. Snapshots of the current market situation are retrieved on a low-resolution, minute-scale basis from the open bitcoin exchange platform Poloniex. As such the usability of the retrieved dataset remains to be shown.

Additional market features, describing the current market situation as well as historic market performance, are evaluated in terms of cost impact. An Orderbook Trading Simulator (OTS), which simulates the individual traders influence on the current market situation, is implemented and used in order to learn and evaluate various trading strategies.

1.3 Related Work

x

1.4 Contributions

- An orderbook trading simulator framework is presented which takes into account the individual traders influence on the current market situation.

1.5 Outline of Contents

The remainder of this thesis is structured as follows:
Section 2 gives a general introduction into the vocabulary of financial computing and the machine learning techniques employed, Section 3 describes the Orderbook Trading Simulator, developed within the scope of this thesis, and Section 4 covers the machine learning part. Section 5 closes with a conclusion and discussion.

2 Background

bla

2.1 Exchange Markets, Bitcoins and Trading Basics

An exchange is a market, where financial instruments are sold and bought. It is typically organized by a broker, which can be both, an individual or a firm, executing buy and sell orders on behalf of dealers for a certain fee or commission. The respective prices are determined by the current market situations, in particular by supply and demand.

Specialized exchanges concentrate on certain sub-types of financial instruments and offer a trading venue for those willing to buy and sell these instruments. Some of them are listed below:

Stock Exchange Market A stock exchange or bourse provides companies access to investment capital in exchange to a share of ownership. Especially in times with notoriously low interest rates, investors tend to accept the greater risk of business development over a risk free, but faint investment, to grow their assets.
E. g. NASDAQ, Deutsche Börse, ...

Commodity exchange market Commodity exchange markets allow for speculations with goods like oil, gold, corn, ...
E. g. Eurex, ...

Foreign exchange market Foreign exchange (short: forex) is considered the largest financial market in the world. The forex market is responsible for determining currency exchange rates.

Bitcoin exchange market x
E. g. Poloniex, ...

Most modern markets are usually fully electronic.......

2.1.1 Ask and Bid

Most exchange markets function after the so called auction market model, where the exchange acts as a mediator between buyers and sellers to ensure fair trading. Here buyers can *bid* a price they are willing to pay for a certain number of shares and sellers can *ask* a price they are aiming to make with a number of shares. The highest of all bids is called the *bid price*, the lowest of all offers ist called the *ask price*. Together they represent the current price at which an instrument is traded.

2.1.2 Limit Order Book and Market Depth

A limit orderbook reflects supply (asks) and demand (bids) for a particular financial instrument. It is usually maintained by the trading venue and lists the number of shares being bid or offered, organized by price levels in two opposing books. Incoming orders are constantly appended to this highly dynamic list, while a matching engine cautiously resolves any inconsistencies (i. e. overlaps) between asks and bids by mediating between the involved parties.

It is usually not before the matching engine has arranged an actual trade, that a trading venue claims a certain percentage of the turnover as a service fee. To encourage active market participation, the pure submission, revision and cancelation of orders is typically free of charge.

	Amount	Type	Volume	VolumeAcc	norm_Price
31.00	200.0	ask	6200.0	8425.0	1.074533
30.00	50.0	ask	1500.0	2225.0	1.039871
29.00	25.0	ask	725.0	725.0	1.005208
28.85	NaN	center	NaN	NaN	NaN
28.70	200.0	bid	5740.0	5740.0	0.994810
28.50	100.0	bid	2850.0	8590.0	0.987877
28.00	300.0	bid	8400.0	16990.0	0.970546

Table 1: Exemplary snapshot of a limit orderbook for stocks of AIWC[1]

Table 1 shows a limit orderbook snapshot up to a market depth of 3, as seen by market participants. Here Alice offers 25 shares per 29$, Bob and Cedar offer 20 and 30 shares respectively per 30$ and David offers 200 shares per 31$.

Based on their trading needs, traders can typically choose between multiple levels of real-time market data.

Level 1 Market Data Basic informations only:
Bid price + size, Ask price + size, Last price + size

Level 2 Market Data Additional access to the orderbook.
Usually data providers display the orderbook only up to a certain market depth m, i. e. the lowest m asks and the highest m bids.

Level 3 Market Data Full data access.
Typically only accessible for the market maker.

1 Acme Internet Widget Company

2.1.3 Slippage

Slippage is defined as the difference between expected and achieved price at which a trade is executed. Slippage may occur due to delayed trade execution. Especially during periods of high volatility, markets might change faster than the order takes to be executed. Slippage is also liked to the order size, as larger orders tend to *eat* into the opposing book and are fulfilled at successively worse price levels.
Slippage can be both positive or negative, depending on the current market movements and must be taken into account by serious investors.

2.1.4 Trading

Investors can execute orders of different types, of which the most common ones are described below:

Market Orders are the most simple form of orders. Here, the investor only specifies the number of shares he want's to buy/sell and the full order is executed immediately, at any price. Especially for large-scale traders or traders with level 1 data access only, these simple market orders are rather hazardous, since the achieved price can significantly differ from the expected price due to sparse supply and demand.

Limit Orders additionally feature a worst price, i. e. the highest price a buyer is willing to pay per share or, respectively the lowest price a seller is willing to make per share. Limit orders are immediately placed into the orderbook and (partially) executed, once the matching engine finds a corresponding trade in the opposing book.
Limit orders reduce the risk of slippage, but do not guarantee execution.

Hidden Orders are placed into the market makers internal orderbook, but not displayed to other market participants with level 2 market data access. They represent a simple solution to large-scale investors seeking anonymity in the market, aiming to obfuscate their trading intention from other market participants.

2.2 Bitcoin

2.2.1 Marktreaktionen

"Bitcoin ist eine Währung, die äußerst sensibel auf Nachrichten reagiert. Begründet wird dies vor allem durch die Möglichkeit, ständig am Markt teilnehmen zu können: Es gibt keine zentrale Ausgabestelle mit geregelten Handelszeiten, an die man gebunden ist."

"Auch die Tatsache, dass viele Anfänger in Bitcoins investieren, führte bereits in der Vergangenheit zu den ein oder anderen Panikverkäufen. Wer sein Geld in Bitcoins investieren möchte, kann die meist lukrative Möglichkeit nutzen, sollte sich jedoch regelmäßig über Marktveränderungen informieren.

Da viele Investoren schnell auf Meldungen reagieren, kann es innerhalb von Stunden zu großen Kursverlusten oder Gewinnen kommen." https://www.btc-echo.de/bitcoin-trading-tipps-prinzipien-des-bicoin-handels_2015022502/

2.3 Supervised Learning

Supervised learning is a subdomain of machine learning, where a function is learned from labeled training data $\{(x_1, y_1), ..., (x_N, y_N)\}$. Each training sample maps a feature vectors $x_i \in X$ to a desired target value or label $y_i \in Y$. Target values may either be categorial, making the learning task a *classification* problem, or continuous, making the learning task a *regression* problem.

A supervised learning algorithms seeks to find a general function $g_w()$ (or it's parameters w), such that $g_w(x_i) \approx y_i | i \leq N$. The learned function should ideally avoid overfitting by finding a generalization to previously unseen data.

Markow Decision Process

Value Function and Bellmann Equation

Value Iteration

Q-Learning

2.3.1 Logistic Regression

bla

2.3.2 Random Forest

bla

2.4 Reinforcement Learning

Reinforcement learning is a subdomain of machine learning, where strategies are learned by an *agent* interacting with its environment. Rather than learning from labeled training data, the agent applies a *trial and error* pattern and exploits external rewards to find actions, maximizing his expected future reward.

2.4.1 Dynamic backward programming

bla

2.4.2 Tree-Based Batch Mode Reinforcement Learning

bla

2.4.3 Neural Fitted Q-learning

bla

3 Orderbook Trading Simulator

This chapter describes the Orderbook Trading Simulator (OTS) and it's underlying OrderbookContainers, implemented within the scope of this thesis. Fed with historic orderbook data it serves as a backtesting framework for testing out various trading strategies. The OTS provides detailed feedback in terms of trading progress, achieved prices and accrued costs.

3.1 Data Origin

Since typical financial data providers must make an earning from their treasures, they typically only deliver delayed market data on a complimentary basis. Investors dependent on real time or level 2 market data (see Section 2.1.2) are usually charged horrendous monthly subscription fees.

A costless alternative exists in open cryptocurrencies, like bitcoins (see Section 2.2). The digital asset exchange platform Poloniex [3] provides an open API for querying detailed market data in real time. As their push API, to receive live order book updates and trades, was rather error-prone and buggy when this project started, the decision was made, to query full orderbooks on a minutely basis.

On Nov, 10th 2016, 10:00 am, a daemon was started, to fetch orderbook snapshots up to a market depth of 5000 from Poloniex via HTTP GET requests. The volume of recorded orderbook snapshots for nine distinct currency pairs[2] has since grown to roughly 100GB (as per 2017-06-20). This thesis is based on a condensed version of the currency pair USDT/Bitcoin.

Listing 1: Data fetched from Poloniex via HTTP GET request

```
# https://poloniex.com/public?command=returnOrderBook&currencyPair=USDT_BTC&depth=5000
{"asks" :[[ "705.450000" ,2.772181], [ "705.450196", 0.139212] ,["706.170000" ↩
    ,0.052838] , ... ], "bids":[["705.000000",0.158232],["703.700000" ,0.001250], ↩
    ... ], "isFrozen": 0, "seq": 63413296}
```

2 Recorded currency pairs include USDT/BTC, BTC/ETH, BT/XMR, BTC/XRP, BTC/FCT, BTC/NAV, BTC/-DASH, BTC/MAID, BTC/ZEC

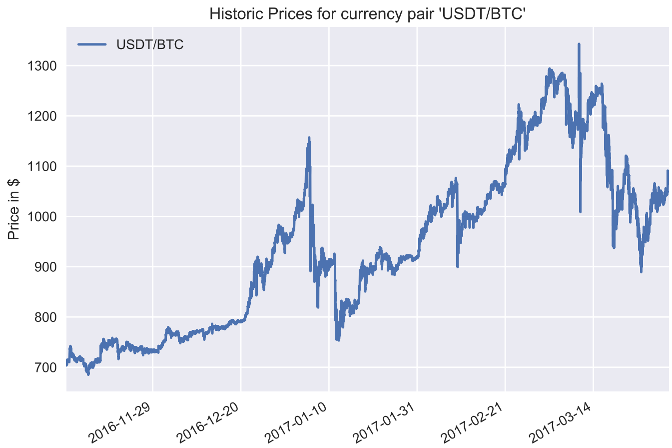

Figure 1: Historic center prices between Nov, 10th 2016 and Mar, 31 2017, as fetched from Poloniex.

3.2 Data preprocessing

The python `class OrderbookContainer` aggregates all informations contained in an individual orderbook snapshot. It enforces correct price ordering in the two opposing bid and ask books and provides additional methods for market visualization and feature extraction. To restrict wasteful memory usage, orderbook snapshots are condensed in two ways:

- Almost identically price levels are round to the second decimal and their respective order volumes merged.

$$\left.\begin{array}{r} 0.139212 * 705.450000 \\ 2.632969 * 705.450196 \end{array}\right\} = 2.772181 * 705.45$$

- Market depth is capped just above the threshold of 100 bitcoins, roughly corresponding to a market depth of 100-140 prices levels in both books. This threshold allows to simulate trades up to a market order price of 70.000 $ at any time throughout the whole recording period.
- Erroneous orderbook snapshots have been discarded.

Listing 2 shows the most important functions, provided by the OrderbookContainer class. OrderbookContainer instances are vigorously used by the OTS.

Listing 2: OrderbookContainer

```
ob = OrderbookContainer(timestamp="2016-11-08T10:00",
                        bids=pd.DataFrame([200., 100., 300.],
                        columns=['Amount'], index=[28.7, 28.5, 28]),
                        asks=pd.DataFrame([25., 50., 200.],
                        columns=['Amount'], index=[29., 30., 31.]))
```

```
# Available methods
ob.plot(outfile='sample.pdf')  # plt.show or plt.savefig
ob.asks  # pd.DataFrame
ob.bids  # pd.DataFrame
ob.features  # returns a dict of precomputed features
ob.get_bid(), ob.get_ask(), ob.get_center()  # float
ob.get_current_price(volume=100)  # achievable cashflow by market order
ob.get_current_sharecount(cash=70000) # number of shares aquirable by market order
ob.compare_with(other_ob) # returns orderbook deltas used by the OTS
ob.enrich()  # computes Volume, VolumeAcc and norm_Price
ob.head(depth=3)  # returns the orderbook, capped at a market depth of 3
ob.plot()
```

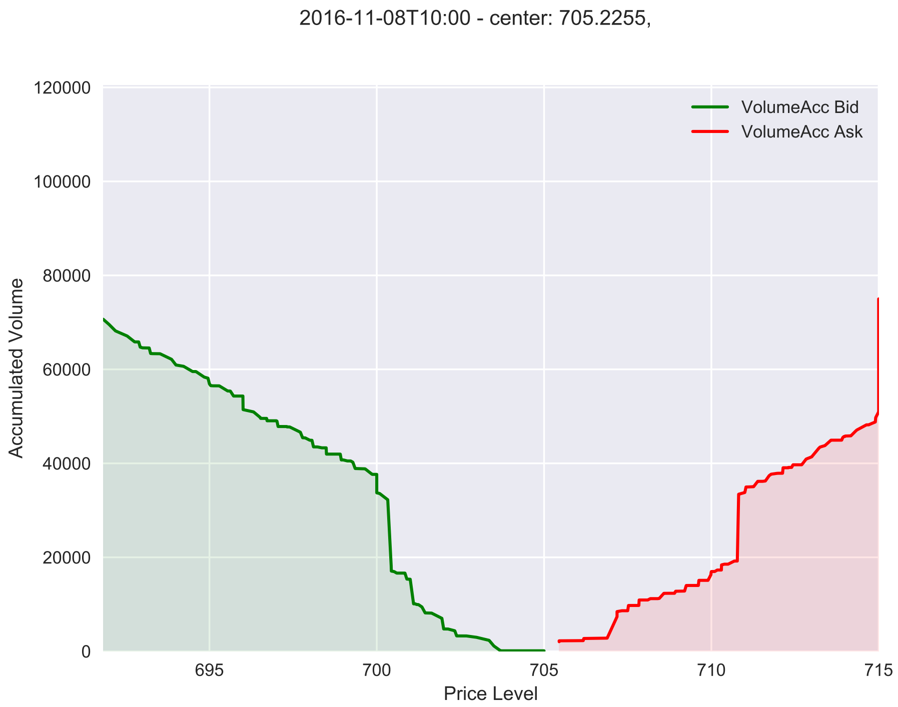

Figure 2: A simple visualization of an limit orderbook.

3.3 Simulator

The OTS framework serves as basis for all preceding experiments and evaluations. Each simulator instance is fed with an array of subsequent OrderbookContainers (*orderbook windows*) and a targeted trading volume V, which it pretends to trade into cash or visa versa within a fixed time horizon H, according to an external strategy. In the rare case of missing orderbook snapshots (see Section 3.2), the *real* time horizon may be larger than usual, since always H subsequent orderbooks are selected.

Limit orders may be placed at predefined, discrete time steps within the trading horizon H. The simulator is done, once the remaining trading volume is zero, which is enforced at the very last time point. Any remaining volume at *H-1* is transformed into a simple market order and executed immediately, at any price. Additional parameters control the simulators behavior, when its main function `trade(limit=...)` is called:

volume : The targeted trading volume V.
Positive values indicate buy orders, negative values indicate sell orders.

consume : `'cash'` or `'volume'`
Defines whether `volume` should be interpreted as *cash* (goal: buy/sell shares for V dollars), or as *sharecount* (goal: buy/sell V shares).

period_length : `default=15`
Defines the duration at which a limit order is executed. After a trade has been placed, the simulator iterates over the next `period_length` orderbooks, before the results are reported and a reviewed order may be placed.

tradingperiods : `default=4`
Defines the number of trade reviews, that can be made within the time horizon $H =$ `period_length * tradingperiods`.

costtype : `default='slippage'`
Defines which of multiple cost functions to use in the returned reports.

3.3.1 Orderbook and strategy visualization

Figure 3 visualizes a 60 minutes long orderbook windows, where the solid red lines mark the *average* (a) and *worst* (b) price, that has to be paid at a given time point, in case of an immediate market order of 100, 75, 50 and 25 bitcoins respectively. Analogously, the solid green lines represent achieved prices for sale orders of -25, -50, -75 and -100 bitcoins.

As can be seen in this graph, ask prices deviate more from the center price than bid prices. An plausible inference might be, that imbalances between demand and supply might serve as a valuable indicator for future price trends.

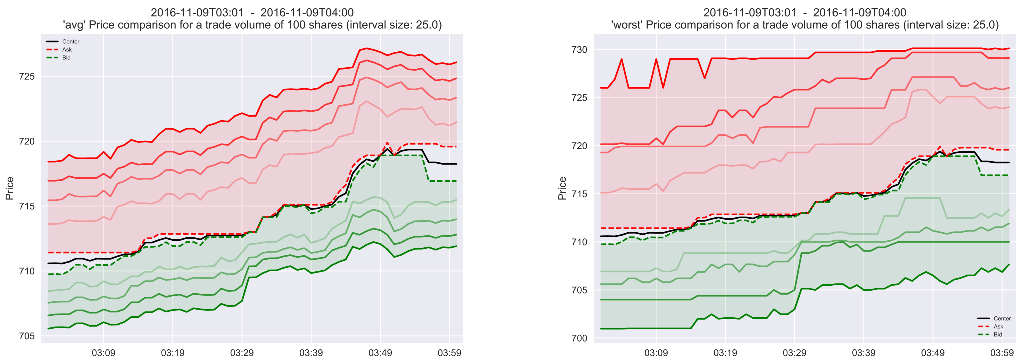

Figure 3: An orderbook window over a period of 60 minutes.

3.3.2 Masterbook

During instantiation, the OTS creates a copy of the first orderbook, called the *masterbook*. Hereinafter executed trades do only affect this internal *masterbook*, which can be reset to it's initial state at any time. This avoids overhead, when testing multiple strategies, since the same instance can be used again and again.

The remaining orderbooks are then converted into *deltabooks*, containing only changes between subsequent orderbooks. In case of an OTS reset, these *deltabooks* need not be recomputed, avoiding another overhead, while the original orderbooks are kept in memory as well.

3.3.3 Trade execution

bla

3.4 Evaluation / Comparison of strategies

bla

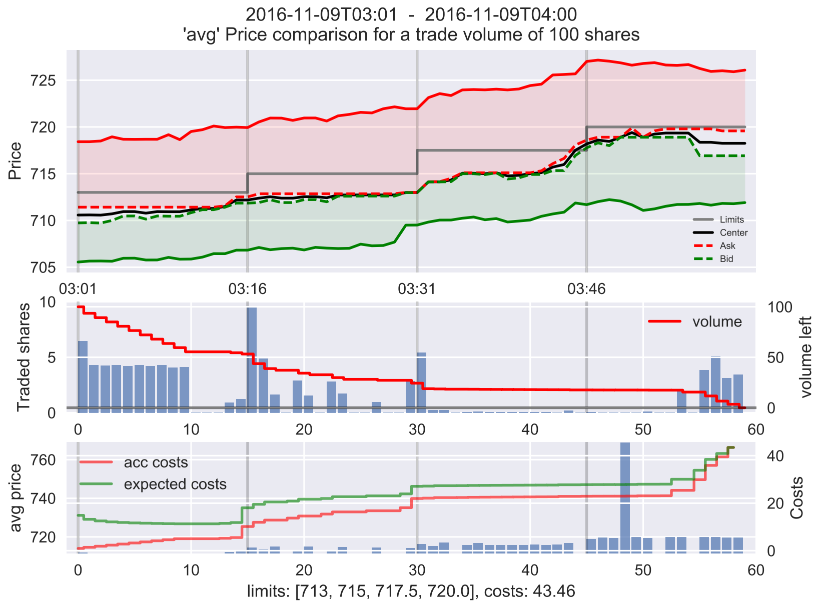

Figure 4: An orderbook window over a period of 60 minutes.

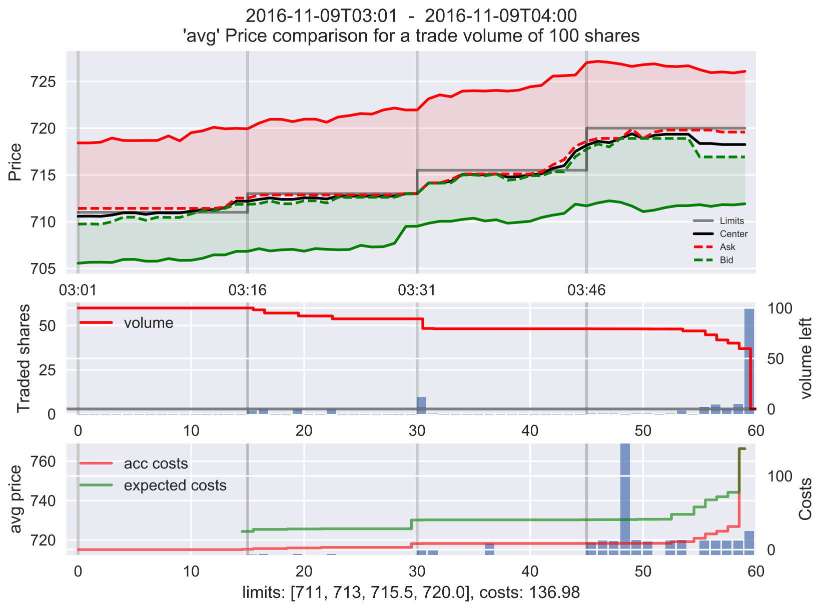

Figure 5: This strategy placed bad limits, resulting in a high slippage of 136.98
The largest portion of the order is executed as a costly market order and after the prices have risen to the worse.

4 Reinforcement Learning on Orderbook Data

This chapter is about a concrete, real world example. The seminars secondary task was to analyze stock exchange data to find out how fast automated traders respond to published ad hoc messages. The complete code for this example can be found online at [2] and is split into several R scripts:

4.1 RL Agents

bla

4.1.1 QTable Agent

bla

4.1.2 BatchTree Agent

bla

5 Conclusion

Due to the volume of data, the real world example shown in **??** would have been a tough job on any single local workstation or students notebook. When analyzing big data, it is a great relief or even an inevitable thing to use a cluster of computer for distributed storage and data processing.

Hadoop is a great and powerful cluster framework and R is a highly popular and well-advanced programming language for statisticians. In a world of ever growing data, Hadoop and R make a perfect fit. Both combined, the mightful analytic capabilities of R can be applied to big data.

A Glossary

OTS Orderbook Trading Simulator

B References

[1] Y. Nevmyvaka, Y. Feng, and M. Kearns. *Reinforcement Learning for Optimized Trade Execution*. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: ACM, 2006, pp. 673–680. ISBN: 1-59593-383-2.

[2] A. Perschmann. *Orderbook Agents*. https://github.com/axelperschmann/orderbook_agent. 2017.

[3] Poloniex. *Poloniex Digital Asset Exchange*. http://www.poloniex.com. [Online; accessed 20-June-2017]. 2010.

C List of Figures

D List of Tables

E Listings