ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics

Master Thesis [programme Econometrics and Management Science]

Dynamic portfolio management in a cryptocurrency market using deep reinforcement learning

Name student: Sten van Brummelen
Student ID number: 488532

Supervisor: dr. (Andrea) A.A. Naghi
Second assessor: dr. (Paul) P.C. Bouman

Date final version: 2022-08-01

Abstract

A model-based reinforcement learning framework is proposed for solving the dynamic portfolio management problem in the cryptocurrency market. By formulating the problem as a Markov decision process and working out the future dynamics of the environment, we learn an intelligent deep learning agent to assess the trade-off between transaction costs and future utility. Moreover, due to a reformulation of the action space, the temporal dependency of the agent's decision-making has been eliminated, improving the efficiency of the training process. Back-tests on historical data have shown that the proposed methods outperform baseline methods that do not account for transaction costs correctly. Multiple experiments were performed using time-series cross-validation where it was found the proposed method was able to increase profit, as well reduced risk. It was found that the proposed method is able to deal with considerably high transaction fees such that in future research, the decision interval might be decreased as an attempt to increase profit and to increase the amount of available data.

Contents

1 Introduction

In an age where data plays a more prominent role each day, new models that can learn from these large amounts of data are developed at a fast pace to solve a broad range of problems. Artificial Intelligence (AI) is expected to have a great impact on our future daily lives as it enables outsourcing simple and repetitive tasks like driving, data entry, bookkeeping and accounting. Even though human peak performance might still outperform some AI algorithms, the algorithm can perform highly repetitive tasks at great speed over great lengths of time without suffering from decreasing performance. Moreover, robots do not require pay.

Technological advancements in the information technology (IT) sector have also impacted the financial landscape. In 2008, in the white paper *Bitcoin: A Peer-to-Peer Electronic Cash System*, a person, or group of persons under the pseudonym Satoshi Nakamoto presented to the world a protocol designed to execute electronic transactions in a way that eliminates the need for a mutually trusted third party [Nakamoto, 2008]. The system implemented a novel technology, known as a blockchain, that proves the legitimacy of a transaction to both parties. In 2009, Nakamoto released Bitcoin, the world's first cryptocurrency that makes use of cryptography to eliminate the need for a trusted third party such as a bank to perform peer-to-peer transactions.

Cryptocurrencies can be exchanged on cryptocurrency exchanges with Binance as today's most prominent example with a daily traded volume that composes 30.55% of the total market volume of roughly two trillion dollars, according to www.coincap.com. Most exchanges allow users to place orders and retrieve historical market data with sampling intervals as short as minute through application programming interfaces (API's). This infrastructure is ideal for algorithmic trading as it provides a large amount of historical data that can be used to accurately evaluate performance, and it provides the infrastructure that is necessary for implementation.

Even though trading rules may be implemented successfully, doing so requires expert-knowledge about the financial exchange. Machine learning methods on the other hand may be implemented that do not rely on financial theory. Given the high sampling intensity, we might try to increase the frequency of the decision making process in order to react to sudden market changes more rapidly. This however raises the importance of managing transaction costs as (i) total traded volume increases and (ii) transaction costs are expected to be proportionally larger relative to short term price changes. In classical financial investment theory that predates such small sampling intervals, holding periods are assumed to be much longer such that transaction may even be ignored.

To successfully solve the dynamic portfolio management problem (DPMP), one needs to accommodate for both price changes as well as transaction costs. Deep reinforcement learning (DRL) provides a suitable setting that may be used to solve this this problem. In DRL methods, an agent, represented by artificial neural network (ANN), is trained to maximize cumulative future rewards. This is done by letting the agent explore an environment through a set of actions followed by its reinforcement using observations about the rewards with respect to its actions.

The goal of this thesis is to propose and evaluate a DRL framework for the DPMP that accommodates for transaction costs in a general financial exchange. Various variations of the framework utilizing different aspects of existing DRL methods will be evaluated with the ultimate goal of improving the efficiency of the method without loss of performance. The findings in the research can be extended to other financial markets as well as the proposed framework does not make restrictive assumptions about the structure of the market. It is relevant for both practical applications in algorithmic trading and for researchers in the field of computational finance.

2 Literature study

Lucarelli and Borrotti have proposed a multi-agent deep Q-learning framework for portfolio management in the cryptocurrency market [Lucarelli and Borrotti, 2020]. The agents were able to manage risk by regulating financial exposure of an asset through a discretized action space. Apart from a regular deep Q-network (DQN), a double DQN and a dueling double DQN were tested but it was found that the regular DQN performed best. Performance of the framework was shown to outperform buy-and-hold (BH) baselines and the framework achieved average positive returns in nine out of the ten test periods of 15 days. Park et al. proposed a deep Q-learning method that was tested in a multitude of financial stock markets and was found to outperform baseline strategies including momentum and mean reversion strategies [Park et al., 2020]. In contrast to the method proposed by Lucarelli and Borrotti, the method used a single DQN agent that was able to trade in a multi-asset environment. The agent had access to the combinatorial action space consisting of every possible combination of asset positions $\{\text{sell}, \text{hold}, \text{buy}\}$ for each asset. The number of output nodes was thus equal to 3^n and hence exponential in the number of assets n, leading to scalability limits.

No method was found in literature that was able to efficiently define a discrete action space in a multi-asset environment. The main bottleneck being the impracticability of risk management: this requires the even further discretization of an already exponential action space. Methods that were able to incorporate risk management were thereby forced to take a multi-agent approach, likely leading to sub-optimal performance due to the decomposition of the problem.

Betancourt and Chen proposed an actor-critic method that allows for a continuous action space. Proximal policy optimization (PPO) was used to train a network consisting of gated recurrent unit (GRU) structures to output a target portfolio distribution [Betancourt and Chen, 2021]. To reinforce the agent for the transaction costs, a linear program was solved to determine the minimal transaction costs of realizing the target distribution given the initial distribution. The method was shown to outperform state-of-the-art methods, but due to the lack of experimental control it remains unclear exactly what differences between the methods may have contributed. Jiang and Liang proposed an actor-only method in which a convolutional neural network (CNN) agent outputs a portfolio vector without being reinforced by a critic network [Jiang and Liang, 2016]. Instead, a reward function is used that directly maps the output of the network to reward values. Thereby eliminating the critic, improving the efficiency of the method and expectedly lowering variance. Even though commission fees were not accounted for in the reward function, when back-testing the agent, the agent outperformed all benchmarks except for passive aggressive mean reversion (PAMR) in terms of average total profit. Another method that directly reinforces an agent in a continuous action space was introduced by Moody and Saffell in 2001 before any of the model-free DRL methods had been invented [Moody and Saffell, 2001]. The method works by recursively calculating the portfolio value as a function of the agent's actions over a multitude of time steps. The agent is then be reinforced by recurrent backpropagation over what is essentially a one-layer gated recurrent unit (GRU). For larger networks however, backpropagation over multiple time steps may slow down the learning process.

No methods were found in literature that were able to correctly evaluate transaction costs without the need for a critic network. The main goal of this thesis is to improve the efficiency of the learning process by reformulating the continuous action space such that it can be used to evaluate transaction costs directly. We can then correctly reinforce our agent with respect to its actions much more simply and directly. Model-free reinforcement learning methods have been formulated most generally such that they can be used to solve an immensely broad range of problems that require close to no prerequisite knowledge. After all, it is clear that efficiency can be gained by making full use of that which we do know about the dynamics of the problem.

3 Problem definition

In reinforcement learning, an intelligent agent learns to maximize some form of utility that it gains by means of interacting with a system, or environment. The environment can be characterized by a state that is observed by the agent and that forms the basis for its decision-making. The agent responds to the state of the system by performing that action which it deems most profitable in terms of expected total future utility. As a result of the action, the system transitions to a next state through a stochastic process that depends partially on the action of the agent given the state of the system. Depending on the state transition and the action of the agent, the agent obtains a reward, which is used to reinforce the agent in order to maximize long-term utility. Where long-term utility is some function in the rewards that are obtained over the state transitions during the process.

Such a decision-making process can be mathematically expressed in terms of a Markov decision process (MDP). Let $s \in S$ be the state of the system, $a \in A$ the agent's action, and r the immediate reward that is observed as the system transitions from the current state s to the next state s'. The actions are determined depending on the state through a policy $\pi : S \to A$ that is held by the agent. It is the goal of the agent to find the optimal policy π^* that maximizes long-term utility, which is usually defined as R_t, the sum of all future rewards that follow some time period t. As the time horizon may very well be infinite, future rewards are often discounted against a discount rate $\gamma \in [0, 1]$ such that R_t converges according to a geometric series. Table 1 shows some important notation that is often used in reinforcement learning and that shall be used later to derive a reinforcement learning framework for solving the dynamic portfolio management problem.

Notation	Expression	Name
R_t	$\sum_{k=0}^{\infty} \gamma^k r_{t+k}$	Discounted reward
$Q_t^\pi(s,a)$	$\mathbb{E}_\pi[R_t \vert s_t = s, a_t = a]$	State-action value
$V_t^\pi(s)$	$\mathbb{E}_\pi[R_t \vert s_t = s]$	State value
$A_t^\pi(s,a)$	$Q_t^\pi(s,a) - V_t^\pi(s)$	Advantage

Table 1: General reinforcement learning notation.

3.1 Formulating the Markov decision process

Let $s_t = (p_t, y_t)$ be the state of the system at time t, with price vector $p_t \in \mathbb{R}_+^n$ and portfolio vector $y_t \in \mathbb{R}_+^n$, where n is the number of financial assets that are included in the portfolio. The price vector p_t holds for each asset in the portfolio the last period's closing price, that is measured in terms of a risk-less asset: the asset that sits in the first position in our portfolio such that $p_{t0} = 1$ by definition. The portfolio vector y_t holds for each asset in the portfolio the total allocated value in terms of the risk-less asset. Right at the beginning of a time period, the agent performs an action $a_t \in A$ that moves the current portfolio y_t to an intermediate portfolio $\hat{y}_t$ that is held over the remainder of that time period. Closing prices are observed at the end of each time period after which the system transitions to the next state $s_{t+1} = (p_{t+1}, y_{t+1})$. Here, the price changes determine the transition of the portfolio via $y_{t+1} = c_t \circ \hat{y}_t$ with price change multipliers $c_t = p_{t+1} \oslash p_t$. The reward that is awarded to the agent is defined as the logarithmic growth rate of the total portfolio value over the transition as in (1).

$$r_t = \ln\left(|y_{t+1}| \,/\, |y_t|\right) = \ln\left({c_t}'\widehat{y_t}\right) - \ln|y_t| \tag{1}$$

3.2 Formulating the decision variables

For an agent to be able to manage a portfolio, we need to define decision variables that can be controlled by an agent in order to change the composition of a portfolio. First, we need to define some ground rules according to which we may change the state of a portfolio. To do so, we define the directed graph $G = (V, A)$ as a mathematical representation of a financial market with nodes V representing the assets that are available for trading, and arcs A representing the exchange environments over which the assets can be exchanged. Considering the arbitrary arc $(u, v) \in A$, we can exchange one unit of asset u for $p_{(u,v)}$ units of asset v against a fixed commission rate $\zeta \in [0, 1]$, such that the effective exchange rate equals $p_{(u,v)}(1-\zeta)$ units of asset v per unit of asset u.

The portfolio can be dynamically redistributed by performing a series of ordered exchanges over the arcs in the graph. A path $h \in H$ is defined as an ordered sequence of arcs that joins a sequence of vertices, connecting the path's origin node h_0 to its destination node h_1. Where H denotes the set of paths that are available for trading. A sequence of exchanges over a path $h \in H$ can be summarized as an exchange between the assets h_0 and h_1 at a path-specific exchange rate γ_h. Where the path-specific exchange rate γ_h is the product of all exchange rates in the path as in (2).

$$\gamma_h = \prod_{a \in h} p_a (1-\zeta) = (1-\zeta)^{|h|} \prod_{a \in h} p_a, \quad h \in H \tag{2}$$

The agent is then able to manage the portfolio using the decision variables $q_h \geq 0$ that indicate the quantity of asset h_0 that is exchanged for asset h_1 over path h for each $h \in H$. A solution to the dynamic portfolio management problem is thus defined as a value assignment to the decision variables q_h for each $h \in H$ as defined in Definition (1), where we have used the set notation in (3).

Definition 1 *A solution is defined as $\hat{y}_u = \sum_{h \in H_{\bullet u}} q_h \gamma_h$ for each asset $u \in V$. With decision variables $q_h \geq 0$ for each path $h \in H$ such that $\sum_{h \in H_{u\bullet}} q_h = y_u$ for each asset $u \in V$.*

$$\begin{aligned} H_{uv} &= \{h \in H : u = h_0, v = h_1\} \\ H_{u\bullet} &= \{h \in H : u = h_0\} = \cup_{v \in V} H_{uv} \\ H_{\bullet u} &= \{h \in H : u = h_1\} = \cup_{v \in V} H_{vu} \end{aligned} \tag{3}$$

As the number of paths may be exponential in the size of the graph, we can not hope to include all paths in our solution method. To reduce the number of paths, we select for each asset pair $u, v \in V$ only the path with optimal exchange rate among the set of paths H_{uv} that have origin node u and destination node v. The problem of finding the path with optimal exchange rate is a shortest path problem, can be solved using a general linear programming (LP) formulation as shown below in (4).

$$\begin{array}{rrcll} \min & z_{uv} & = & \sum_{a \in A} b_a x_a & \\ \text{s.t.} & f_w & = & \sum_{v \in V_{w\bullet}} x_{(w,v)} - \sum_{v \in V_{\bullet w}} x_{(v,w)}, & w \in V \\ & x_a & \geq & 0, & a \in A \end{array} \qquad f_w = \begin{cases} -1 & \text{if } w = u \\ 1 & \text{if } w = v \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

By solving (4) using weights $b_a = -\ln(p_a(1-\zeta))$ for each $a \in A$, the optimal exchange rates can be calculated as $\gamma_{uv} = \max_{w \in W_{uv}} \{\gamma_w\} = e^{-z_{uv}}$ for each asset pair $u, v \in V$. To ensure boundedness of (4), we ensure that each weight b_a is positive by setting $p_a = 1$ for each $a \in A$ (this is equivalent to the assumption of efficient market prices). Then, $b_a = -\ln(1-\zeta)$ for each $a \in A$ such that the optimality of (4) is independent of market prices. The optimal exchange rates γ_{uv} are constant over tiem and we only need to evaluate (4) once.

We can rewrite $\widehat{y}_u$ and y_u using Definition 1 and the set notation in (3) as:

$$\begin{array}{lllllll} \hat{y}_u & = & \sum_{h\in H_{\bullet u}} q_h\gamma_h & = & \sum_{v\in V}\sum_{h\in H_{uv}} q_h\gamma_h & = & \sum_{v\in V}\gamma_{vu}\sum_{h\in H^*_{uv}} q_h & u\in V \\ y_u & = & \sum_{h\in H_{u\bullet}} q_h & = & \sum_{v\in V}\sum_{h\in H_{uv}} q_h & = & \sum_{v\in V}\sum_{h\in H^*_{uv}} q_h & u\in V \end{array}$$

Defining $H^*_{uv} = \{h\in H_{uv} : \gamma_h = \max_{h\in H_{uv}}\{\gamma_h\}\}$ as the set of paths with optimal exchange rate connecting the assets u and v, we replace W_{uv} by W^*_{uv}. By redefining the decision variables as $q_{uv} := \sum_{h\in H_{uv}} q_h$ for each asset pair $u, v\in V$ we have reformulated Definition 1 using a quadratic number of decision variables as in Definition 2.

Now, let Q and γ be the matrices with respective elements q_{uv} and γ_{uv} for each asset pair $u, v\in V$. Using this reformulation, we can effectively express the intermediate portfolio $\widehat{y}$ using the decision matrix Q using simple matrix algebra as in Definition 2.

Definition 2 *A solution is defined as* $\widehat{y} = y + (Q\circ\gamma)'\mathbf{1_n}$*, with* $Q\in\mathbb{R}_+^{n\times n} : Q\mathbf{1_n} = y$.

In Definition 2, the decision matrix Q depends on the portfolio y through the constraint $Q\mathbf{1_n} = y$. However, as the portfolio varies over time, we wish to normalize the decision variables such that they are independent of the portfolio. We rewrite the constraint $Q\mathbf{1_n} = y$ from Definition 1 as

$$\sum_{v\in V} q_{uv} = y_u \implies \sum_{v\in V} q_{uv}/y_u = \sum_{v\in V} a_{uv} = 1$$

, for each asset $u\in V$. Where we have defined the normalized decision variable $a_{uv} = q_{uv}/y_u$ as the fraction of y_u, the total available fund in asset u, that is exchanged for asset v. We then substitute the normalized decision variables in $\widehat{y} = y + (Q\circ\gamma)'\mathbf{1_n}$ as

$$\begin{aligned} \widehat{y}_u &= y_u + \sum_{v\in V}(\gamma_{vu}q_{vu} - q_{uv}) \\ &= y_u + \sum_{v\in V}(\gamma_{vu}a_{vu}y_v - a_{uv}y_u) \\ &= y_u + \sum_{v\in V}\gamma_{vu}a_{vu}y_v - y_u\sum_{v\in V} a_{uv} \\ &= \sum_{v\in V}\gamma_{vu}a_{vu}y_v \end{aligned}$$

, for each asset $u\in V$. Where we have used that $\sum_{v\in V} a_{uv} = 1$ for each $u\in V$. We can then express Definition 2 using the normalized decision variables as in Definition 3, where we can satisfy $A\mathbf{1_n} = \mathbf{1_n}$ by applying the softmax activation function over the rows of a real-valued decision matrix $Z\in\mathbb{R}^{n\times n}$ as $A = \text{softmax}(Z)$. As we have that $\text{softmax}(Z)\mathbf{1_n} = \mathbf{1_n}$ by definition, we can apply unconstrained optimization techniques to optimize a policy $\pi : S\to\mathbb{R}^{n\times n}$.

Definition 3 *A solution is defined as* $\widehat{y} = (A\circ\gamma)'y$*, with* $A\in\mathbb{R}_+^{n\times n} : A\mathbf{1_n} = \mathbf{1_n}$

3.3 Defining the state-transitions

The definition of the action space in Definition 2 enables us to express the state transition to y_{t+1} using the price changes c_t that is given by the data and the action A_t that is given by our policy as in (5). Where we have used the matrix presentation $\hat{y}_t = (A_t \circ \gamma)' \, y_t$ of Definition 2 and we have defined the matrix $C_t = \text{diag}(c_t)$ for the sake of notation. Consequently, the immediate reward r_t of the transition can be calculated as in (6) by substituting (5) in the original definition of the immediate rewards in (1).

$$\begin{aligned} y_{t+1} &= c_t \circ \hat{y}_t \\ &= c_t \circ \big((A_t \circ \gamma)' y_t\big) \\ &= \text{Diag}\,(c_t)(A_t \circ \gamma)' y_t \\ &= C_t (A_t \circ \gamma)' y_t \end{aligned} \qquad (5) \qquad \begin{aligned} r_t &= \ln |y_{t+1}| - \ln |y_t| \\ &= \ln \big|C_t (A_t \circ \gamma)' y_t\big| - \ln |y_t| \end{aligned} \qquad (6)$$

To make notation more compact, we define the transition matrix $P_t = C_t (A_t \circ \gamma)'$ such that the transitions are denoted by $y_{t+1} = P_t y_t$. We can then efficiently express any portfolio as a sequence of matrix products of the previous transition matrices up to the initial portfolio as in (7). Such a sequence of transition matrices themselves form a multi-step transition matrix P_t^{k-1} such that $y_{t+k} = P_t^{k-1} y_t$.

$$\begin{array}{rclcl} y_{t+1} & = & P_t^1 y_t & = & P_t y_t \\ y_{t+2} & = & P_t^2 y_t & = & P_{t+1} P_t y_t \\ & & \vdots & & \\ y_{t+k} & = & P_t^{k-1} y_t & = & P_{t+k-1} P_{t+k-2} \cdots P_t y_t \end{array} \qquad (7)$$

3.4 Deriving the value functions

The state transitions that have been defined in the previous subsection will now be used to express the future rewards as a function of the action of the agent and the portfolio state that results because of it. It are these two variables that the agent has influence over, as the price changes and commission fees are given by the data. The immediate reward $r_t(y, A)$, the state-action value, and the state $v_t(y)$ in (8) follow from the definition of the immediate reward in (6). Where we have used the definition of temporal differences to define the state-action value $Q_t(y, A)$ and the definition of the value function $V_t(y) = Q_t\,(y, \pi_t(y))$. Introducing our policy $\pi_t(y)$ that takes an action at time t depending on the portfolio state at that time.

$$\begin{array}{rcl} r_t(y, A) & = & \ln \big|C_t (A \circ \gamma)' y\big| - \ln |y| \\ Q_t(y, A) & = & r_t(y, A) + V_{t+1}\big(C_t (A \circ \gamma)' y\big) \\ & = & \ln \big|C_t (A \circ \gamma)' y\big| - \ln |y| + V_{t+1}\big(C_t (A \circ \gamma)' y\big) \\ V_t(y) & = & Q_t(y, \pi_t(y)) \\ & = & \ln \big|C_t (\pi_t(y) \circ \gamma)' y\big| - \ln |y| + V_{t+1}\big(C_t (\pi_t(y) \circ \gamma)' y\big) \end{array} \qquad (8)$$

Again, to make notation more compact, transition matrices are introduced as was done in (7). In this case however as we have introduced the policy π, we have that the actions depend on the portfolio state at each time period and that the previous actions thus influence the future actions as a result. The transition is now denoted by $P_t(y) = C_t (\pi_t(y) \circ \gamma)'$ such that the transition to the next portfolio is denoted by $P_t(y) y$. This notation is adapted in the definition of the state value function $V_t(y)$ in (9) and through forward recursion a general recursive expression is derived for an arbitrary value of k.

$$\begin{aligned}V_t(y) &= \ln|P_t(y)y| - \ln|y| + V_{t+1}(P_t(y)y)\\ &= \ln|P_t(y)y| - \ln|y| + V_{t+2}(P_{t+1}(P_t(y)y)P_t(y)y)\\ &\quad + \ln|P_{t+1}(P_t(y)y)P_t(y)y| - \ln|P_t(y)y|\\ &= \ln\left|P_t^2(y)y\right| - \ln|y| + V_{t+2}\big(P_t^2(y)y\big)\\ &= \ln\left|P_t^k(y)y\right| - \ln|y| + V_{t+k}\big(P_t^k(y)y\big)\end{aligned} \tag{9}$$

As we assume a potentially infinite time horizon, the expression in (9) is an infinite series such that it has no closed form solution; as long as the time horizon continues, the expression contains an unknown term about the future rewards. We now take the limit of the right-hand side in (9) as k approaches infinity and we rewrite $V_t(y)$ as in (10). Here, we have defined $v_t(y)$ as the state value function that takes a portfolio y and returns a vector containing for each of the assets the local state value that is obtained for any value allocated to this asset. It follows that the value function can be rewritten as the logarithm of a simple linear combination of the local state values with respect to the portfolio entries. From the definition of the local state value function $v_t(y)$, we then find a recursive relationship for it in (11) that we can use for estimation methods later on.

$$\begin{aligned}V_t(y) &= \lim_{k\to\infty}\left[\ln\left|P_t^k(y)y\right| - \ln|y|\right]\\ &= \lim_{k\to\infty}\left[\ln\left(\mathbf{1_n}'P_t^k y\right) - \ln|y|\right]\\ &= \ln\left(\mathbf{1_n}'\lim_{k\to\infty}\left[P_t^k\right]y\right) - \ln|y|\\ &= \ln\left(\exp\{v_t(y)\}'y\right) - \ln|y|\end{aligned} \tag{10}$$

$$\begin{aligned}\exp\{v_t(y)\} &= \lim_{k\to\infty}\left[P_t^k(y)'\right]\mathbf{1_n}\\ &= \lim_{k\to\infty}\left[P_t(y)'P_{t+1}^{k+1}(P_t(y)y)'\right]\mathbf{1_n}\\ &= P_t(y)'\lim_{k\to\infty}\left[P_{t+1}^{k+1}(P_t(y)y)'\right]\mathbf{1_n}\\ &= P_t(y)'\exp\{v_{t+1}(P_t(y)y)\}\end{aligned} \tag{11}$$

By filling in $P_t(y) = (\pi_t(y)\circ\gamma)\,C_t$ in (11) we find that we can further decompose $v_t(y)$ into a deterministic part that is directly determined by the action of the agent, and a random part that includes unobserved price changes of future time periods. As the deterministic part is known, it would be futile to include it in the target variable in our estimation methods. To prevent this, the passive local state value function $f_t(y)$ is defined that excludes this deterministic part about the action of the agent in the current time period. In contrast, $v_t(y)$ will from now on be referred to as the active local state value function as it includes this part about the action of the agent. Using the definition of the passive local state value function in (12) and (11), we can derive a similar recursive relationship for $f_t(y)$ as we did for $v_t(y)$.

$$\begin{aligned}\exp\{v_t(y)\} &= P_t(y)'\exp\{v_{t+1}(P_t(y)y)\}\\ &= (\pi_t(y)\circ\gamma)\,C_t\exp\left\{v_{t+1}\big(C_t(\pi_t(y)\circ\gamma)'y\big)\right\}\\ &= (\pi_t(y)\circ\gamma)\exp\left\{f_t\big((\pi_t(y)\circ\gamma)'y\big)\right\}\\ \exp\{f_t(y)\} &= C_t\exp\{v_{t+1}(C_ty)\}\\ f_t(y) &= \ln\left(C_t\left(\pi_{t+1}(C_ty)\circ\gamma\right)\exp\left\{f_{t+1}\big((\pi_{t+1}(C_ty)\circ\gamma)'C_ty\big)\right\}\right)\end{aligned} \tag{12}$$

In (**??**) we substitute $f_t(y)$ for $v_t(y)$ in the definition of the value function in (10) using the relationship in (12). Before using this value function for reinforcement, we make the observation that due to the possibly infinite time horizon, the cumulative rewards may increase indefinitely such that we need to stabilize the value function. Usually in reinforcement learning, future rewards are discounted at a discount rate such that the resulting cumulative rewards are guaranteed to be finite. This method however has several drawbacks. First and foremost, discounting future rewards implies that the agent becomes more focused on maximizing short-term rewards rather than maximizing the long term rewards in the infinite time horizon. Thereby we alter the objective of the problem. Another disadvantage of the discount rate is that it imposes another hyperparameter that requires tuning, which is a complex and time consuming optimization problem. What we will do instead is to subtract a baseline from the value function such that the relative difference between the value function and the baseline is finite. A suitable baseline is chosen as $Q_t(y, I_n) = F_t(y)$, which corresponds to the future reward that is obtained when holding the current portfolio using the action I_n. We refer to $F_t(y)$ as the passive state value, as it is the future reward that is associated with passively holding a portfolio. Parallel to the relationship between the global and local active state value functions, we have that the passive state value function $F_t(y)$ is a weighted average of the passive local state values with respect to the portfolio entries as is shown in (13). By subtracting the active global state value function from the passive global state value function, we get the advantage value function in (13), which expresses the relative advantage of performing the agent's action over passively holding the portfolio. This is the function that is responsible for reinforcing the agents in this thesis and it is the main theoretical result that will be presented.

$$\begin{aligned}
V_t(y) &= \ln\left(\exp\{v_t(y)\}'y\right) - \ln|y| \\
&= \ln\Big(\exp\big\{f_t\big((\pi_t(y)\circ\gamma)'y\big)\big\}'(\pi_t(y)\circ\gamma)'y\Big) - \ln|y| \\
F_t(y) &= Q_t\left(y, I_n\right) \\
&= \ln\Big(\exp\big\{f_t\big((I_n\circ\gamma)'y\big)\big\}'(I_n\circ\gamma)'y\Big) - \ln|y| \\
&= \ln\left(\exp\{f_t(y)\}'y\right) - \ln|y| \\
V_t(y) - F_t(y) &= \ln\Big(\exp\big\{f_t\big((\pi_t(y)\circ\gamma)'y\big)\big\}'(\pi_t(y)\circ\gamma)'y\Big) - \ln|y| \\
&\quad - \left(\ln\left(\exp\{f_t(y)\}'y\right) - \ln|y|\right) \\
&= \ln\Big(\exp\big\{f_t\big((\pi_t(y)\circ\gamma)'y\big)\big\}'(\pi_t(y)\circ\gamma)'y\Big) - \ln\left(\exp\{f_t(y)\}'y\right)
\end{aligned} \tag{13}$$

3.5 Introducing state-independency

We can see from the expression of the advantage function in (12) that the advantage function depends on the state of the portfolio. This raises the concern that we might over fit to portfolio states that have been visited in the training procedure. To prevent this, we might add random noise to the policy's output to promote visiting a more diverse set of states. A more efficient approach however that does not require exploration of the environment in such a way would be to pre-define the states. Here, the most simple solution is suggested in which we fix the initial state as an equally weighted portfolio $y = \mathbf{1_n}$ for each time period. Thereby the policy will for each time period give a generalized action in which each asset has equal amount of attention. Now we can simplify the results from the previous subsection as we have fixed the initial portfolio y. As a result we get that both the actions as well as the value functions are fully determined by the data such that we may ignore the arguments in the value function as well as in the policy as in (14).

$$\begin{aligned}
f_t(\mathbf{1_n}) &= \ln\left(C_t\left(\pi_{t+1}(C_t\mathbf{1_n})\circ\gamma\right)\exp\left\{f_{t+1}\big((\pi_{t+1}(C_t\mathbf{1_n})\circ\gamma)'C_t\mathbf{1_n}\big)\right\}\right) \\
f_t &= \ln\left(C_t\left(\pi_{t+1}\circ\gamma\right)\exp\{f_{t+1}\}\right) \\
V_t - F_t &= \ln\left(\exp\{f_t\}'(\pi_t\circ\gamma)'\mathbf{1_n}\right) - \ln\left(\exp\{f_t\}'\mathbf{1_n}\right) \\
&= \ln|(\pi_t\circ\gamma)\exp\{f_t\}| - \ln|\exp\{f_t\}| \\
&= \ln|(\pi_t\circ\gamma)\,\mathrm{softmax}(f_t)|
\end{aligned} \tag{14}$$

3.6 Stabilizing the cumulative rewards

One topic that has remained untouched thus far is that of stabilizing the cumulative reward values. As we assume a possibly infinite time horizon, the cumulative rewards might increase infinitely over time. Typically a discount rate is applied to future rewards such that the cumulative reward converges according to a geometric series. In the case we have knowledge about the future dynamics of the problem, we might be able to develop an alternative method to stabilize reward values that dos not require the use of an additional hyperparameter. Also, we may replace the approximate method of discounting future rewards by an exact method such that we do not degrade the optimality of the method.

Here, we suggest a method for stabilizing the cumulative rewards that works by calculating the relative difference of the cumulative rewards for each asset with respect to a baseline, similar to the method with which the advantage values are stabilized. By choosing as baseline the cumulative reward of an asset that is included in the portfolio, we may reduce the number of values that need to be estimated as well. By convention, we define the baseline that is subtracted from the cumulative rewards as the cumulative reward of the risk-less asset. We denote the stabilized values using a hat-notation and we will show that we can use these values to replace the unstabilized values without loss of generality.

$$\begin{aligned}\widehat{f}_t &= f_t - b \\ &= \ln\left(C_t(\pi_t \circ \gamma)\exp\left\{f_{t+1}\right\}\right) - b \\ &= \ln\left(C_t(\pi_t \circ \gamma)\exp\left\{f_{t+1} - b\right\}\right) \\ &= \ln\left(C_t(\pi_t \circ \gamma)\exp\left\{\widehat{f}_{t+1}\right\}\right)\end{aligned} \tag{15}$$

$$\begin{aligned}\text{softmax}(\widehat{f}_t) &= \text{softmax}(f_t - b) \\ &= \frac{\exp\{f_t - b\}}{|\exp\{f_t - b\}|} \\ &= \frac{\exp\{f_t\}\exp\{-b\}}{|\exp\{f_t\}|\exp\{-b\}} \\ &= \text{softmax}(f_t)\end{aligned} \tag{16}$$

4 Deep reinforcement learning methods

Amongst the class of model-free reinforcement learning methods, methods can be subdivided based on the method that is used to estimate the values to which the agent is reinforced. The reason for this division lies in the definition of the action space, which can be either discrete or continuous. In the case of a discrete action space, one might estimate the value function for each action individually. In a continuous action space there exist an infinite number of actions such that the former method becomes impossible. Even though a continuous action space might be discretized, this quickly becomes infeasible for high dimensional action spaces. To overcome this problem, the problem can be decomposed by training an actor network to output the actions and training a critic to estimate the value function given an action. This gives rise to the existence of critic-only, actor-critic, and actor-only methods. Where in the last method we use the knowledge that is available about the future dynamics of the environment to directly reinforce the agent.

4.1 Critic-only methods

In the first class of deep reinforcement learning methods, an agent learns to interact with its environment by learning the state-action value function that maps states to estimates of the expected cumulative rewards for each individual action in a finite set of actions. It was first proposed by Mnih et al. as the Deep Q-Network (DQN), in which the network has an output node for every action in the finite set of actions depending on the problem at-hand. In the original DQN paper, the authors learned an AI agent to play Atari games using a convolutional neural network to map pixilated on-screen data of the video game to actions that were performed by the character of the video game.

The state-action values, or Q-values, are denoted by $Q_t(s,a)$ and are defined as the expected cumulative reward of performing action a at state s as $Q_t(s,a) = \mathbb{E}[R_t|s_t = s, a_t = a]$. The agent then chooses the action based on a greedy policy that performs the action with highest expected cumulative reward by $a_t^* = \arg\max_{a\in A}\{Q_t(s,a)\}$. Data is collected and stored in a data buffer by observing the immediate rewards and state transitions that occur after performing actions. The data buffer is a collection of tuples (s,a,r,s') containing the state, action, reward and next state that have previously been observed. This data is used for learning using the method of temporal differences, in which the temporal differences in (17) are calculated as target values for the Q-value estimates of the agent. Usually by minimizing the mean squared error between the Q-values and the temporal differences using mini-batch gradient descent.

$$Q_t(s,a) \leftarrow r_t + \max_{a\in A}\{Q_{t+1}(s',a)\} \tag{17}$$

In a financial market, the action space could consist of actions that tell the agent which assets to sell, hold or buy. Optionally, the discrete action space could be expanded such that it also includes for each action a quantity or proportion of the asset to sell, hold or buy in order to manage risk. This approach however suffers from serious drawbacks as discretization of a high-dimensional action space means an exponential number of actions as well. As we do not wish to suffer from an exponential number of actions, we choose to not discretize the action space and deal only with actions that indicate selling or buying the full amount of assets. But even then some adjustments need to be made in order to keep the number of output nodes small in our neural network. A combinatorial action space as was proposed by Park et al. will inevitably result in an exponentially large set of actions. The method that is proposed in this thesis overcomes this problem by exploiting the theoretical properties of the Markov decision process that has been defined in the precious section. As a result, instead of estimating the action values for each possible combination, we will estimate the value function for each asset and derive the greedy policy using the knowledge about the structure of the market and transaction costs that have been derived in the previous section.

In Section 3 we have defined a general problem definition for a continuous action space. We will now discretize the continuous action space into a discrete action space by adding additional constraints to Definition 3. To do so, we will define a set of m equally-spaced values between zero and one that that each entry is able to take in the action matrix A. For a given value of $m \geq 2$, we define the action space of a single entry as $a_{uv} \in \{i/m, i = 0, 1, \ldots, m-1\}$ for each pair of assets $u, v \in V$. The size of the complete action space is then equal to $m^{n \times n}$ and is thereby exponential in the number of assets in the portfolio for any value of $m \geq 2$. To ensure that the critic method is scalable in the number of assets that are included in the portfolio, we will derive a policy from our theoretical framework in 3 that requires a number of output nodes in our network that is only linear in the number of assets that are included in the portfolio.

We wish to find the policy that maximizes the advantage values in (14). Note that in the case that f_t is known, we can find this policy by solving the simple optimization problem in (18). The solution is the greedy policy in (20). This policy makes use of the fact that f_t is given. In reality, the exact value of f_t is however unknown as it is defined as a limit as the time horizon approaches infinity. Instead, we may estimate it using for instance temporal difference learning on the recursive expression of f_t in (14). We can express a single element of f_t in (19) and use our solution from (18) to derive a simple expression of the temporal difference for each of the assets.

$$\begin{array}{rl} \max & \ln\left|(\pi \circ \gamma)' f\right| \\ \text{s.t.} & \pi \mathbf{1_n} = \mathbf{1_n} \end{array} \Leftrightarrow \begin{array}{rl} \max & \left|(\pi \circ \gamma)' f\right| \\ \text{s.t.} & \pi \mathbf{1_n} = \mathbf{1_n} \end{array} = \begin{array}{rl} \max & \sum_{u,v \in V} \pi_{uv} \gamma_{uv} f_v \\ \text{s.t.} & \sum_{v \in V} \pi_{uv} = 1, v \in V \end{array}$$
$$= \sum_{u \in V} \left(\begin{array}{rl} \max & \sum_{v \in V} \pi_{uv} \gamma_{uv} f_v \\ \text{s.t.} & \sum_{v \in V} \pi_{uv} = 1 \end{array} \right) = \sum_{u \in V} \max_{v \in V} \{\gamma_{uv} f_v\} \tag{18}$$

$$\begin{aligned} f_{tu} &= \ln\left(c_{tu} \sum_{v \in V} \pi_{uv}^{t+1} \gamma_{uv} \exp\{f_{t+1,v}\} \right) \\ &= \ln c_{tu} + \ln\left(\max_{v \in V} \{\gamma_{uv} \exp\{f_{t+1,v}\}\} \right) \\ &= \ln c_{tu} + \max_{v \in V} \{\ln(\gamma_{uv} \exp\{f_{t+1,v}\})\} \\ &= \ln c_{tu} + \max_{v \in V} \{f_{t+1,v} - \beta_{uv}\} \end{aligned} \tag{19}$$

$$\pi_{uv} = \begin{cases} 1 & \text{if } v = \arg\max_{w \in V} \{\gamma_{uw} f_w\} \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

Let $f^{\phi}(X_t) \in \mathbb{R}^n$ denote the output of our critic network that are the estimates of the temporal difference targets in (19). Where ϕ are the tunable weights of the critic network and X_t is the data matrix containing the history of the market. The temporal difference targets in (19) can then be learned to the critic network by minimizing the mean squared error between the estimates and the target values with respect to the weights of the network ϕ as in (21).

$$\ell_t(\phi) = \frac{1}{n} \sum_{u \in V} \left(f_u^{\phi}(X_t) - \ln c_{tu} - \max_{v \in V} \{f_{t+1,v} - \beta_{uv}\} \right)^2 \tag{21}$$

4.2 Actor-critic methods

As the critic method uses a discrete action space, it is unable to diversify the portfolio. In order to manage risk, we require a policy that can perform continuous actions such that it does not completely allocate all value in a single asset. Then we can learn our agent to manage risk by optimizing over a risk-adjusted objective function such as the Sharpe ratio. As we have seen, discretization of the action space using a critic-only method results in an exponential number of actions. By defining an actor agent that outputs continuous action matrix directly we can overcome this problem. We still however need to reinforce our agent somehow. This can be done using an actor-critic method in which we learn a critic network to estimate the value function to which the actor is reinforced.

Let π^θ be the actor network with weights θ and f^ϕ denote the critic network that estimates the passive local value function. We use the result in (13) to derive the temporal difference targets that are learned to the critic network. Then, the actor network tries to maximize the advantage function using the estimates of the critic network.

$$\begin{aligned} f_t &= \ln\Big(C_t\big(\pi^\theta(X_{t+1})\circ\gamma\big)'\exp\left\{f^\phi(X_{t+1})\right\}\Big) \\ \ell_t(\phi,\theta) &= \frac{1}{n}\big(f^\phi(X_t)-f_t\big)'\,\big(f^\phi(X_t)-f_t\big)-\ln\big|\big(\pi^\theta(X_t)\circ\gamma\big)\,\mathrm{softmax}(f_t)\big| \end{aligned} \tag{22}$$

4.3 Actor-only methods

The last method that is proposed in this thesis is the actor-only method, in the agent is reinforced directly with respect to historic data data using direct reinforcement learning. In direct reinforcement learning, we have available a direct relationship between the actions of the agent and the reward function such that we do not need to estimate the rewards using a critic network. We effectively replace the estimates of the critic network by approximations of the value function that are directly calculated from the data itself. To explain as to why we can do this, we need to consider the assumptions that have been made on the time horizon of the Markov decision process. Previously we have assumed that the time horizon is potentially infinite. As there is no clear endpoint at which we would like to stop the process, we simply assume that the time horizon is infinite. The implication of an infinite time horizon however is that the exact future reward is unknown. Previously we solved this problem by estimating the future rewards, but now we would like to derive the future rewards directly from the data as an approximation method. We relax the assumption of the infinite time horizon and we will optimize over a finite time horizon such that we can derive a closed form solution to the recursive equations that we have derived in the problem definition in Section 3.

The loss function of the actor-only method can be derived almost directly from the expressions in (14) by substituting for the policy action π_t the output of the actor network $\pi^\theta(X_t)$. We then define the time horizon of the Markov decision process as $t = 1, 2, \ldots, T$, where T is the last available time period in our data set. We assume that we stop trading at time T such that $f_{T+1} = 0$, which ensures that (23) has a closed form solution. We can now complete the recursion and find $f_1, f_2, \ldots, f_T$, which we use to directly reinforce our agent using the loss function $\ell_t(\theta)$.

$$\begin{aligned} f_{t-1} &= \ln\left(C_{t-1}\left(\pi^\theta(X_t) \circ \gamma\right) \exp\{f_t\}\right), \quad t = T, T-1, \ldots, 2 \\ \ell_t(\theta) &= -\ln\left|(\pi^\theta(X_t) \circ \gamma)\,\text{softmax}(f_t)\right|, \quad t = 1, 2, \ldots, T \end{aligned} \tag{23}$$

5 Data

Some cryptocurrency exchanges offer publicly available data though application programming interfaces (API's), which can be accessed through representational state transfer (REST) HTTP requests. This means that data can be accessed fast and easily without complex coding and data processing. The proposed method can be applied to any financial exchange that offers historical price data as the price changes are used to reinforce our agent. Binance is used as data source for the following reasons: a large number of cryptocurrencies can be exchanged, a large amount of data is available as sampling intervals start at one minute, it offers a sufficient number of features and it contains historical data that traces back to 2017 for the oldest cryptocurrencies.

Instead of on-line learning, historical data can be used to speed up the learning process. Two data types that are typically used are order book data and candlestick charts. In our case, at least the candlestick data is required as it contains the historical prices used for reinforcement. For reasons of simplicity, order book data is not considered. It also makes comparison of the performance to performance in other literature more relevant as in many cases only candlestick charts are used.

Binance offers historical candlestick data through a public API endpoint [Binance, 2022]. A candlestick summarizes the evolution of the price of a financial instrument in the time period between the open and closing time. It is specified by an open, high, low and closing price. Where the open and closing prices are the instrument's prices upon open and closing times and the high and low prices are the maximum and minimum value of the price in the period between the open and closing times. This information can be graphically presented in a candlestick chart, which is a series of sequential candlesticks over time. Nowadays, candlesticks are typically accompanied by other data types such as information about the trading activity within the period. Table 2 shows the features that are available through Binance's candlesticks API endpoint.

Column	Feature
1	Open time
2	Open
3	High
4	Low
5	Close
6	Volume
7	Close time
8	Quote asset volume
9	Number of trades
10	Taker buy base asset volume
11	Taker buy quote asset volume

Table 2: Binance's historical data features.

For reasons of simplicity, only the fifth column containing the closing prices were used. They were used as input data for the agents as well as a means to reinforce the agents with respect to its output. The immediate reward values to which the agent is reinforced can be expressed using simple mathematical functions using the growth-factors $c_t^u = p_u^{t+1}/p_u^t$ for each asset $u \in V$ and time period $t = 1, ..., T-1$. By using the log-changes, which are calculated by taking the logarithm of the growth-rates, as input data, the prices data is both linearized and stabilized. The input data was calculated from the growth rates by Z-Score normalizing the log-changes for each asset individually as $x_u^t = (\ln c_u^t - \mu_u)/\sigma_u$, where μ_u and σ_u are the mean and standard deviation of the log-changes of asset u respectively for each $u \in V$.

6 Experimental procedure

To determine the effectiveness of the methods, the three methods proposed in section 4 will be evaluated and their performance will be compared against two baseline methods. Assets that are included in the portfolio are selected by highest traded volume during the validation period, under the condition that the asset's first data entry dates before the start of the training period. Only time periods for which data was available for every asset was used, others were removed. To ensure fair comparison, the same neural network architecture was used for each of the methods. Only minor changes were applied depending on the needs of the method at-hand. The first baseline method is the reinforcement learning method proposed by Jiang and Liang, in which commission fees are ignored. This provides a suitable comparison for our models as we hope to improve the profitability of the model by correctly modeling the transaction costs. Second, a equally weighted buy and hold method (EWBH) is used as a baseline. In which we initially divide all our funds equally over the assets in the portfolio and do not perform any transactions over the entire test period.

To accurately evaluate performance, time series cross-validation can be applied as visualized in Figure 1. This method ensures that the chronologically ordered structure of the data is not violated while using all historical data that is available for training at each test period. Part of the training data will be kept separate as a validation set for hyperparameter tuning and model selection. Early stopping is applied by monitoring the validation score and terminating the training process as soon as the validation score has failed to improve for over a certain number of epochs. By choosing to validate the models in this way, we assure that the data that is used in the training phase is relevant for the back-test period.

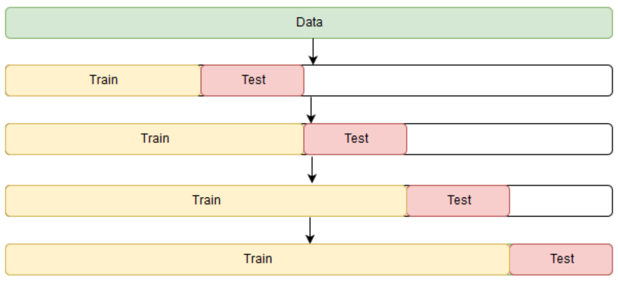

Figure 1: Time series cross-validation

6.1 Back-testing

To compare the performance of the methods, back-tests are performed on the test set after early stopping is executed based on the validation score. The weights of model with the highest validation score are recovered and are used for back-testing. To ensure fair comparison, the rules of the back-test procedure is the same for each method. In Section 3, we have defined a general formulation of the action space in Definition 2 that will be used for the back-test simulation in Algorithm (1). Here, each agent takes the market data matrix X_t and the current portfolio state y_t as input and returns as output the exchange matrix Q_t. That is in turn used to transition the portfolio to the next state as defined in the problem definition in Section 3.

In the case of the actor-critic and actor-only methods, we have defined a redistribution matrix A_t that can be used to calculate Q_t by multiplying the rows of the redistribution matrix A_t by the portfolio entries in y_t via $Q_t = \text{Diag}\,(y_t)A_t$. For the critic-only method, the transition matrix can be found by selecting for each asset the target asset with maximum state-value estimate. In the case the portfolio distribution is outputted directly however, an optimization problem must be solved to find the action matrix given the current distribution and the desired target distribution. A linear programming (LP) formulation was proposed by Betancourt and Chen to solve this optimization problem. We take as an input the current portfolio y and a target distribution a, and return the action matrix A by minimizing the total traded volume. Let $\text{LP}\,(y_t, a_t)$ be the result of this optimization problem, denoting the penalty multiplier that results from the transactions that are required to realize the target distribution a_t when starting at distribution y_t. It is defined as $|\widehat{y_t}| \,/\, |y_t|$, where $\widehat{y_t}$ in this case is the portfolio that results from executing the optimal exchange quantity matrix Q that is found. Using the general problem definition 2 we can then find $\widehat{y_t}$ as $\widehat{y_t} = y_t + (Q \circ \gamma - Q')\,\mathbf{1_n}$.

Algorithm 1 Back-testing procedure

```
**Input:** $(\mathcal{D} = \{(X_t, C_t), t = 1, 2, \ldots, T\}, \text{agent})$
**Output:** $(\mu_r, \sigma_r)$
Initialize $y_1 \in \mathbb{R}^n_+$
**for** $t = 1, 2, \ldots, T$ **do**
    $Q_t = \text{agent}\,(X_t, y_t)$
    $y_{t+1} = C_t\left(y_t + \left(Q_t \circ \gamma - Q_t{}'\right)\mathbf{1_n}\right)$
    $r_t = \ln|y_{t+1}| - \ln|y_t|$
**end for**
**return** $r_1, r_2, \ldots, r_T$
```

6.2 Network architecture

Several neural network architectures have been proposed in the literature that have been shown to be able to beat the market. As the evaluation of the performance of the architecture is outside of the scope of this thesis, the main requirements for the architecture is that its performance has been demonstrated literature, it is easy to implement and convergence is fast. For instance, [Jiang and Liang, 2016] proposed a convolutional neural network architecture for solving the DPMP with a portfolio consisting of 12 assets including the risk-less asset. They found that increasing the complexity of their network did not improve test results due to overfitting and they have demonstrated in their paper that this model was able to generate steady profits. As the method that has been implemented by the authors is similar to the methods that are proposed in this thesis, and the architecture is relatively small, easy to implement and has been shown to converge relatively fast, this architecture was used as a basis for each of the reinforcement learning methods. Only small changes were applied to accommodate for the differences between the reinforcement learning methods.

A schematic overview of the differences between the networks architectures that were used in each of the methods is shown in Figure 2. The network architecture was adapted to the individual methods depending on their state-dependency, the formulation of the action space and the use of value estimates for methods using temporal difference learning. In Figure 2, it can be seen that the architecture is divided into individual components that were included in the architecture depending on the method. The components can be grouped depth-wise into three layers: as shown in Figure 2 we have, from top to bottom, the input layers (shown in red), a hidden layer (shown in green), and the output layers (shown in blue). All methods share the same prices input layer in the top right of the figure, as well as the same hidden layer that is used for processing and combining the features generated in the convolutional layer. The other input layer contains the portfolio entries and may be used in methods in which the output of the network depends on the state of the portfolio. Such a method will not be implemented however here and this is shown only for the purpose of illustration. The output layers in blue at the bottom of the figure correspond, from left to right, to the output of the portfolio distribution, value estimates, and redistribution matrix.

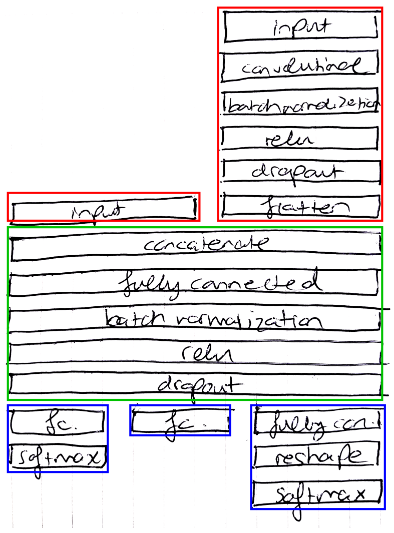

Figure 2: Architecture and its decomposition into individual components of the artificial neural network. The architecture is based mostly on the one proposed by Jiang and Liang, with the only adaptation being the variable action space and input layers, as well as the introduction of batch normalization layers prior to the ReLU activation function.

Two types of input layers are used to feed the data about the state of the system to the model. First, the prices input layer processes the prices data using a one-dimensional convolution with kernel with kernel size four. The number of filters is set equal to n, the number of assets that are included in the portfolio. The remaining part of the layer consists of a feed-forward through a batch normalization, ReLU and dropout layer, where dropout was applied with dropout rate 0.7, as suggested by Jiang and Liang. Finally, the output is flattened such that it can be concatenated with the portfolio entries and fed to the hidden layer. The second type of input layer is the input layer of the portfolio entries and is used in methods in which the output of the network depends on the state of the current portfolio.

The hidden layer is used in all methods. Its purpose is to process and combine the features that are generated by the convolutional layer and (optionally) the portfolio entries. It is a fully connected layer consisting of 500 neurons to which the same structure of batch normalization, ReLU activation and dropout is applied. Again with dropout rate 0.7. The hidden layer allows correlation structures between the features to be modelled and can be seen as the intelligent part of the network that connects the information that is available in the input features to the implications that they have on the expected outcome of the market.

Finally, the output layers are chosen depending on the formulation of the output space of the method. Three types of output layers are used. Firstly, the portfolio distribution layer is defined as a layer consisting of n nodes to which the softmax activation function is applied. This output layer is used for methods that directly output a target distribution of the portfolio, which is done in all of the reinforcement learning baseline methods of Jiang and Liang, Moody and Saffell and Betancourt and Chen. The second output layer is used in the actor-only and actor-critic methods proposed in this thesis. It is a layer consisting of n^2 nodes, which is reshaped into the $n \times n$ redistribution matrix, over which the softmax activation function is applied over the individual rows. The last output layer is used in the actor-critic and critic-only methods and consists of $n - 1$ nodes, which are used to estimate the value function to which the agent is reinforced.

6.3 Hyperparameter settings

As in any machine learning algorithm, hyperparameters have to be tuned as they determine the way the agent learns. The methods that have been developed in this thesis make use of less tunable parameters due to the fact that they exploit the properties of the Markov decision process according to which the DPMP is modeled. Due to the smaller number of hyperparameters, tuning of these models was found to be considerably more efficient than was the case in other models. Table 3 shows the tuned hyperparameter values that have been used for each of the reinforcement learning methods.

Name	Value
Risk-less asset	Bitcoin
# Assets	12
Interval	30 minutes
Commission rate	0.0025
# Lags	50
Learning rate	1e-4
# Batches per epoch	8
Patience	200
Warm-up	50

Table 3: Hyperparameters and their respective values as implemented in the experimental procedure.

7 Results

Prices data was gathered from Poloniex for 11 assets such that including the risk-less asset, the portfolio consisted of twelve assets. Assets were included based on highest total traded volume of during the validation period. The experiments were performed using data from 2015/06/27 to 2016/08/27 and the risk-less asset was chosen as Bitcoin. Both the validation and test periods consisted of 7 weeks of data and the experiment was repeated for three test periods. The test periods were thus defined as the time periods between 2016/04/02 to 2016/05/21, 2016/05/21 to 2016/07/09 and 2016/07/09 to 2016/08/27. Results were gathered according to the principle of time-series cross-validation in Figure 1 with a constant number of training observations. By fixing the number of training observations, we prevent that (i) the network overfits on early test periods that don't have as much data available and (ii) it underfits for later test periods when more data becomes available. Moreover, by fixing the number of training examples, we keep the batch size constant and prevent the need for adjustable tuning over the different periods.

Early stopping was applied with a patience of 200 epochs after which the weights of the best scoring model on the validation set were recovered. The methods were allowed a warm-up time of 50 epochs, meaning that during the first 50 epochs, the validation score was left unmonitored to allow the validation score to stabilize in order to prevent premature termination of the training process. Data was shuffled and split in eight equal-sized batches that were fed to the optimization algorithm one by one. In the case of the Actor-Only method, the target values were updated after each training epoch and were directly used to compute the validation score as well. Finally, after reaching failing to improve the validation score for 150 epochs, back-tests were performed on the test set using the back-test algorithm in Algorithm 1.

Figure 3 shows the back-test results for each method over each of the three test periods. The vertical axis shows the total logarithmic return over time for each of the methods. The horizontal axis shows the number of time periods (30 minutes) that have passed since the beginning of the test period. In the case of the reinforcement learning methods, the portfolio was initialized as $y_1 = (1, 0, 0, \ldots, 0)$, such that during each back-test, the agent started with a single unit of the risk-less asset. In the case of the baseline method, which is the equal weighted buy and hold (EWBH) method, the portfolio was initialized as $\mathbf{1_n}/n$, after which no transactions were applied during the remainder of the test period.

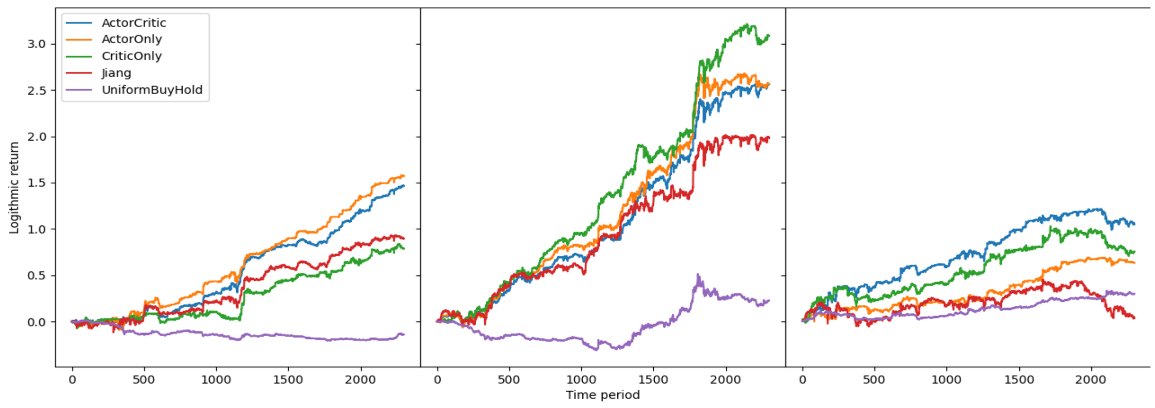

Figure 3: Back-test results for each of the methods for each of the three test periods. With, from left to right the tests period 2016/04/02 to 2016/05/21, 2016/05/21 to 2016/07/09 and 2016/07/09 to 2016/08/27. The vertical axis shows the total logarithmic return, which is equal to the natural logarithm of the total portfolio value. The horizontal axis shows the total number of time periods of 30 minutes that have passed since the start of the test period.

Some performance measures were calculated from the back-test results and they are reported below in Table 4. These are simple functions of the rewards r_t for each time period as defined in (8). Apart from the average return μ_r, we also include a variety of performance measures that tell us something about the risk that results from the trading methods. The values of σ_r and σ_r^- measure the risk that is involved in each of the methods. Where σ_r is simply the standard deviation of the reward values, and σ_r^- is defined as the down-side risk, which is the standard deviation of the downside losses $r_t^- = \min\{r_t, 0\}$. Finally, in the far right two columns we have the Sharpe ratio S_r and the Sortino ratio S_r^- that are calculated as μ_r/σ_r and μ_r/σ_r^-. They are measures of risk-adjusted profit as the ratio of profit to risk, where the Sortino ratio adjusts only for downside risk.

	μ_r ($\times 10^{-4}$)	σ_r ($\times 10^{-3}$)	σ_r^- ($\times 10^{-3}$)	S_r ($\times 10^{-3}$)	S_r^- ($\times 10^{-3}$)
Actor-critic	6.39	7.62	3.93	**8.39**	**16.25**
Actor-only	**6.84**	8.50	5.24	8.05	13.05
Critic-only	3.43	8.55	5.17	4.01	6.63
Jiang	3.90	9.02	5.16	4.32	7.56
EWBH	-0.60	**2.74**	**1.71**	-2.17	-3.49

	μ_r ($\times 10^{-4}$)	σ_r ($\times 10^{-3}$)	σ_r^- ($\times 10^{-3}$)	S_r ($\times 10^{-3}$)	S_r^- ($\times 10^{-3}$)
Actor-Critic	11.13	14.36	8.03	7.75	13.85
Actor-Only	11.15	15.30	8.66	7.29	12.88
Critic-Only	**13.40**	15.75	8.88	**8.51**	**15.10**
Jiang	8.62	16.04	9.06	5.37	9.51
EWBH	0.99	**7.92**	**4.82**	1.26	2.06

	μ_r ($\times 10^{-4}$)	σ_r ($\times 10^{-3}$)	σ_r^- ($\times 10^{-3}$)	S_r ($\times 10^{-3}$)	S_r^- ($\times 10^{-3}$)
Actor-Critc	**4.54**	9.50	5.28	**4.78**	**8.60**
Actor-Only	2.74	7.64	4.44	3.58	6.16
Critic-Only	3.18	10.69	6.42	2.97	4.95
Jiang	0.14	11.76	7.08	0.12	0.20
EWBH	1.29	**3.68**	**2.18**	3.49	5.90

Table 4: Back-test results for the reinforcement learning and baseline methods on the three test periods. With, from top to bottom, test period 1 from 2016/04/02 to 2016/05/21, test period 2 from 2016/05/21 to 2016/07/09 and test period 3 from 2016/07/09 to 2016/08/27.

Considering the results in Table 4, we see that the baseline method EWBH has lowest average returns during the first two test periods, and second lowest in the third period. As this method completely relies on the performance of the market, the reinforcement learning methods have beaten the market in almost all cases. With the exception of the method proposed by Jiang and Liang, and only for the third test period. On the other hand, we can clearly see that the EWBH-strategy has lowest risk as measured by both σ_r and σ_r^- of all methods. This (perhaps inevitable) result is the well-known trade-off of profit and risk that one is confronted with upon investing in volatile assets such as cryptocurrency. Still however, after adjusting the profit for risk, the reinforcement learning methods score considerably higher on the risk-adjusted performance measures of the Sharpe (S_r) and Sortino (S_r^-) ratios. With the only exception of the method proposed by Jiang and Liang, and the Critic-Only method during the last test period.

Comparing the Critic-Only method with discrete action space to the Actor-Critic and Actor-Only methods with continuous action space, we see that in each of the three test periods the Critic-Only method has highest risk in both measures, with the exception of the downside risk during the first test period. In terms of absolute profit, the Critic-Only method scores considerably lower than the Actor-Critic and Actor-Only methods during the first test period and scores relatively low as well during the third test period. With the great exception however for the second test period, where it outperforms all other models in terms of absolute profit and surprisingly also in terms of the risk-adjusted performance measures of the Sharpe and Sortino ratios. In general however, the Critic-Only method shows unstable results over the test periods and it has been shown that even in terms of absolute profit, the Critic-Only method does not necessarily perform better than the Actor-Critic and Actor-Only methods. It is thus safe to say that by using a continuous action space instead of a discrete one, we have successfully reduced risk. This was expected considering the ability of the agent to diversify the portfolio.

Comparing the methods with continuous action space to each other, we see that the Actor-Critic method outperforms the Actor-Only method in all three test periods based on the risk-adjusted performance measures that are the Sharpe and Sortino ratios. In terms of absolute returns, the Actor-Only method scores higher on the first two test periods, but shows a sudden drop in performance over the third test period. As to why this is the case we can only speculate, but a probable reason for this is instability of the training process. As the number of models that are included in the experimental procedure is quite large, and models were re-trained for each test periods, the learning rate was set relatively high to promote fast convergence. Perhaps this has resulted in unstable results. Where some methods more than others have suffered due to the fact that the same hyperparameter settings were used for all methods.

8 Limitations and recommendations

It is not unthinkable that the outcomes presented in this thesis depend on the case-specific problem instance, i.e. the graph $G = (V, E)$ that represents the structure of the market, the number of assets that are included in the portfolio, the commission fee rate ζ etc. In order to obtain more robust results, the experimental procedure presented in Section 6 should be repeated while varying the problem instance. For example by performing the experiment for a variety of markets. Or to repeat the experiment for a set of different commission fees to assess the extend to which the methods are able to adjust the policy for transaction costs. Expanding on instance-specific biases that are imposed on the learning process, we consider the fact that commission fees might have changed over the course of time. Or that the structure of the market may have changed over time such that the penalty matrix γ in reality changes over time. These are all factors that influence the way in which consumers interact with the exchange and thus how prices change over time. Gathering and implementing this information may be difficult. In general, to avoid such problems, it is recommended to consider mostly recent data for learning such that these changes in conditions become less influential.

Even though the network architecture is outside the scope of this thesis, it would be interesting to explore more developed and complex architectures than the one used here. Several other options for network architectures have been proposed in literature with promising results on recent data. For example the results obtained by Betancourt and Chen, where a significantly larger network structure was used that includes multi-head attention layers to extract features from the data. In general, it is very reasonable to assume that the performance depends on the network architecture such that exploration of this area is highly recommended. Moreover, for reasons of simplicity, only the prices data was used as an input to the agents. As the closing prices capture only a small part of the state of the market, it is possible that by including more features, the performance may be improved.

Several limitations exist that could negatively impact the performance of a real-time implementation of the proposed methods in financial market. First and foremost, depending on the volume of the orders that are placed by the agent, interaction of the agent with the financial market may influence the state of the market and thus the profitability of the actions. As we have only made use of historical price data, the true influence that the actions would have on the state of the market can not be learned. For small volumes however, it can safely be assumed that the influence of the agent's actions on the state of the market is negligible.

Another complication that might occur in a real-time implementation is the occurrence of slippage, which is defined as the discrepancy between the expected and the actual price of an executed order. Depending on the size of the slippage and the profit margins of the trades, this could hurt performance and should thus be considered in order to manage risk. Especially in the case a high commission fee is applied or in the case of high sampling intensities, where the price changes are expected to be small. As the sampling intensity decreases however, the profit margin increases in size such that the negative effect that slippage has on profitability becomes smaller. Moreover, slippage depends on the liquidity of the market and may even be negligible when dealing with markets with high liquidity. As a result, in some cases, the effect that slippage has on total profitability may be sufficiently small such that they may be ignored in the model.

To deal with the aforementioned complications that could arise due to slippage or capital impact, one could apply on-line learning strategies. In on-line learning algorithms, the agent explores an unknown environment as it simultaneously stores data about the state transitions in a data buffer that is sampled for learning. By observing the state transitions that follow its actions, the agent could learn the influence of its own actions on future rewards. In theory the agent could then exploit its influence over the market prices in order to generate profit, especially when trading in low-volume markets where its influence is larger. Furthermore, prices of executed orders could be stored and used to learn to predict the occurrence of slippage based on the state of the market. As the introduction of slippage and capital impact severely complicate the dynamics of the environment, if one wishes to, it is highly recommended to first try and implement model-free reinforcement learning algorithms; they do not require any previous knowledge about the dynamics of the environment. On the other hand, some of the most successful reinforcement learning implementations make use of model-based reinforcement learning algorithms. As we have seen in the results section, by exploiting the problem structure we have improved the learning efficiency as well as improved the performance. Even though the model does not incorporate for slippage or capital impact, it could serve as a base for a model that does consider these factors in the future.

It has been shown that the proposed method is able to generate profit by accounting for the incurred transaction costs in the decision mechanism of the model. This means that the model is able to maintain a profitable strategy for smaller profit margins than when compared to methods that do not account for transaction costs. Thereby it is able to trade with shorter holding periods. It would be interesting to see how far we can take this by decreasing the holding period. Modern cryptocurrency exchanges offer historical candlesticks data with sampling frequencies as high as one observation per minute. Alternatively, the sampling frequency could be increased even further by making use of data streams, which sometimes even have sampling frequencies of one observation per millisecond. The enormous amount of data that comes available for such high sampling frequencies could lead to better generalization of the models, as well as it could give more reliable performance estimates. No to mention the benefit of being able to respond quickly to market changes. Moreover, by the law of large numbers we get the result that as the number of trading periods increases, the long-term profit converges to the true performance of the agent. Such that the probability of unexpected long-term losses decreases as the number of periods increases.

9 Conclusion

In this thesis, we have proposed a general reinforcement learning framework for solving the dynamic portfolio management problem in a general financial market. Where we have developed a model-based reinforcement learning algorithm that enables an artificial neural network agent to learn to assess the trade-off between expected benefit and transaction costs, while managing the risk to which the portfolio is exposed. We have shown very clearly that by including the future dynamics of the market with respect to the price changes and transaction costs, we have both increased profit and reduced risk. By working out the future dynamics of the environment as a Markov decision process, we have improved the learning efficiency of the reinforcement learning method. By redefining the continuous action space using a continuous redistribution matrix that contains asset pair-specific actions, we have derived a differentiable reward function such that no optimization methods are needed to determine transaction costs. Rather, the reward function is fully differentiable, improving the efficiency of the learning method.

In the future, the proposed learning method can be extended to deal with more complex scenario's. Most notably, by implementing an on-line learning method we could learn an agent to take advantage of the influence that it has over the market prices in order to generate profit. As we have improved the learning efficiency as well as the ability of the agent to make short-term decisions by including transaction costs in the model, we may try to increase the trading frequency. Greater quantities of more relevant data may be collected by shortening the decision interval. It is highly recommended to explore more complex network structures, as it is highly doubted that the network structure used in these experiments may realize these predictions.

Model-free reinforcement learning is often seen as a brute-force technique, where compensate for our lack of knowledge about the dynamics of the environment using incredible computational power. Even though such equipment might be incredibly powerful and lead to better results, the limitations that arise due to our lack of knowledge persists. The absence of a simple, but well-defined model does impose a bottleneck to the training process. By developing model-based reinforcement learning methods we might improve the efficiency of the learning process and eliminate time-consuming processes such as exploration of an unknown environment, or learning a highly complex and unknown reward function. I hope that this thesis opens the door to a real-time implementation of deep learning agents in financial markets, and I do believe that this method can be used to develop on-line learning strategies that may act on arbitrarily small decision intervals in the near future.

References

Carlos Betancourt and Wen Hui Chen. Deep reinforcement learning for portfolio management of markets with a dynamic number of assets. *Expert Systems with Applications*, 164, 2 2021. ISSN 09574174. doi: 10.1016/j.eswa.2020.114002.

Binance. Binance api documentation, 2022. URL `https://binance-docs.github.io/apidocs/spot/en/market-data`

Zhengyao Jiang and Jinjun Liang. Cryptocurrency portfolio management with deep reinforcement learning. 12 2016. URL `http://arxiv.org/abs/1612.01277`.

Giorgio Lucarelli and Matteo Borrotti. A deep q-learning portfolio management framework for the cryptocurrency market. *Neural Computing and Applications*, 32:17229–17244, 12 2020. ISSN 14333058. doi: 10.1007/s00521-020-05359-8.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 12 2013. URL `http://arxiv.org/abs/1312.5602`.

John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, 12:875–889, 7 2001. ISSN 10459227. doi: 10.1109/72.935097.

Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL `www.bitcoin.org`.

Hyungjun Park, Min Kyu Sim, and Dong Gu Choi. An intelligent financial portfolio trading strategy using deep q-learning. *Expert Systems with Applications*, 158, 11 2020. ISSN 09574174. doi: 10.1016/j.eswa.2020.113573.

10 Appendix: Python implementation

This project was implemented using the Python programming language, specifically using the TensorFlow library for deep learning. Where the auto-differentiation tool was heavily used for the custom loss functions used in reinforcement learning. The code was run on Windows 10 OS 64-bit, with processor Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz, 8.00 GB of DDR3 RAM memory, with NVIDIA GeForce RTX 2060 GPU 6GB memory.

Data was gathered from Poloniex' public API with end-point https://api.poloniex.com/ using REST requests and processed using Python's NumpPy package. The full project is included in a ZIP file. Hyperparameters can be tuned in the JSON files in the settings folder for the individual methods. Code for each of the methods and baselines can be found in the methods folder. The complete experimental procedure can be run by executing the main file. Results are then gathered automatically in a new folder that is named as the time stamp of the time of execution of the code in the results folder. The results folder includes for each of the methods the backtest results for training, validation and test periods. As well as a text file showing loss of the current training session that to monitor the progress.