Expert Systems With Applications 164 (2021) 114002

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Deep reinforcement learning for portfolio management of markets with a dynamic number of assets

Carlos Betancourt *, Wen-Hui Chen

Graduate Institute of Automation Technology, National Taipei University of Technology, Taipei, Taiwan

ARTICLE INFO

Keywords:
Reinforcement learning
Deep learning
Portfolio management
Transaction costs
Multiple assets

ABSTRACT

This work proposes a novel portfolio management method using deep reinforcement learning on markets with a dynamic number of assets. This problem is especially important in cryptocurrency markets, which already support the trading of hundreds of assets with new ones being added every month. A novel neural network architecture is proposed, which is trained using deep reinforcement learning. Our architecture considers all assets in the market, and automatically adapts when new ones are suddenly introduced, making our method more general and sample-efficient than previous methods. Further, transaction cost minimization is considered when formulating the problem. For this purpose, a novel algorithm to compute optimal transactions given a desired portfolio is integrated into the architecture. The proposed method was tested on a dataset of one of the largest cryptocurrency markets in the world, outperforming state-of-the-art methods, achieving average daily returns of over 24%.

1. Introduction

Generating financial profits by trading cryptocurrencies is challenging due to their price erratic changes. Cryptocurrencies are decentralized electronic financial assets that appeared as an alternative to fiat currencies (Nakamoto, 2008). However, according to Corbet et al. (2014) the prices of cryptocurrencies are affected by government announcements, policies and actions, in spite of the fact they are decentralized assets. Additionally, cryptocurrency prices show higher volatility than those of traditional assets. For instance, in early 2017, the price of Bitcoin, the well-known cryptocurrency, reached its maximum historical peak of approximately 19,000 USD per unit, but during the subsequent months it plunged to 3000 USD, followed by a strong bounce to its current price of approximately 8,000 USD per unit. For this price behavior, formulating cryptocurrency trading strategies is a non-trivial task.

Reinforcement learning (RL) is a suitable framework to process complex data and handle difficult decision-making processes such as asset trading. A trading process can be naturally formulated as an RL process. In this type of process, an agent takes actions over an environment based on observations of the states of that environment; rewards are received by that agent as a consequence of both the states visited and the actions taken. In the specific case of asset trading, a state of the environment is equivalent to the recent history of the assets; actions are the transactions made to get rid of some of the assets held by the agent and acquire new ones, and the rewards are scalar functions of the earnings or losses seen by the agent for taking those actions. The vector containing the information of the assets held by an agent at any moment is called the *portfolio*, hence this type of process is also known as portfolio management.

Typically RL algorithms have fixed state and action spaces. However, new assets are often added to cryptocurrency markets (Narayanan et al., 2016). Hence, to rapidly incorporate those assets into the process, adaptable state and action spaces are necessary. Most works on automatic asset trading assume the number of assets is static (Bu & Cho, 2018; Jiang & Liang, 2017; Liang et al., 2018; Pendharkar & Cusatis, 2018). Thereby, convolutional layers of neural networks can extract useful information about the prices of that specific set of assets. But, by doing so, a large portion of data is wasted because only a small number of assets is used for training algorithms while datasets contain information collected from dozens and even hundreds of assets. This is an important issue, not only from a sample-efficiency point of view, but also because critical earnings may be accomplished by trading assets that are suddenly incorporated into a market. For instance, in Fig. 1, Dock (coin) reached 2.2 times its original price during the first 20 days in the market, then it fell and settled at about 1.2 times its original price on the subsequent days. This behavior has been observed often in assets recently added to markets; however, we are confident it can be predicted and exploited.

This work proposes an RL method using a recurrent neural network (RNN) to perform portfolio management on markets in which the

* Corresponding author.
E-mail addresses: t104669004@ntut.edu.tw (C. Betancourt), whchen@ntut.edu.tw (W.-H. Chen).

https://doi.org/10.1016/j.eswa.2020.114002
Received 1 April 2020; Received in revised form 10 September 2020; Accepted 11 September 2020
Available online 16 September 2020

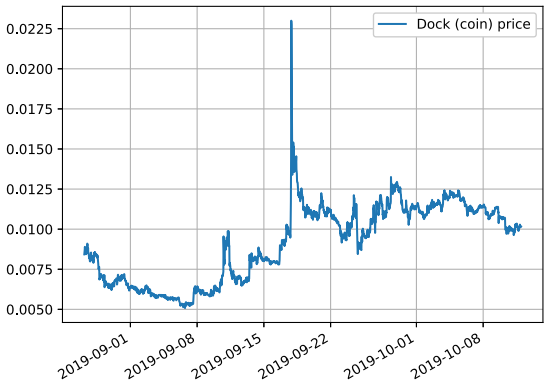

Fig. 1. Price evolution of *Dock* (cryptocurrency) during the first month after being released in the Binance website.
Source: binance.com.

number of assets may change over time. Proximal Policy Optimization (PPO) (Schulman et al., 2017) was adapted for this purpose. PPO is a popular deep RL algorithm, with an actor–critic architecture, that has been shown to perform well on difficult tasks such as video-game playing and dexterous robotics control (OpenAI et al., 2020; Schulman et al., 2017). PPO has been recently applied to portfolio management in markets with a fixed number of assets (Liang et al., 2018). However, to adapt to a dynamic number of assets, we propose a particular architecture that processes assets individually and uses the current portfolio entries for weighting. This results in a network able to effectively process assets that were never even seen during training, without requiring extra training or memory. The proposed method was backtested using data of a cryptocurrency market along state-of-the art baselines in three different setups, which correspond to episodes with lengths of one day, 30 days and 16 weeks with holding periods of 30 min, one day and one week, respectively. The performances of the methods were evaluated using two standard measures for investing and trading: total return and Sharpe ratio. Our method outperformed the baselines in all the tested setups.

Keeping the number of transactions as small as possible is an important issue to consider while doing asset trading. Markets obtain revenues from their services in the form of transaction costs. Any agent that buys or sells assets gives a small percentage of those transactions to the service provider. Cryptocurrency market transaction fees are typically lower than 1%, which are among the lowest compared to fees found in other types of financial asset markets. However, these seemingly negligible fees become important when transactions are made frequently, for instance in periods of minutes or hours. This is because the changes in the assets acquired by the agent may not compensate for the losses suffered by transaction costs. Hence, the algorithm should aim to keep the number of transactions low. To cope with this issue, in our design, the current portfolio vector is given to the network in the output layers, penalizing assets not held by the agent.

Additionally, a novel algorithm to compute the optimal transactions is given in this work. In a market where transaction costs exist, if an agent wants to obtain a portfolio vector satisfying certain specific proportions, the agent needs to perform transactions, thus giving up some amount for doing that. Ormos and Urbán (2013) proposed an iterative method to compute the values of the transactions needed to convert some portfolio into another with minimal cost. However, this method assumes transaction costs are the same for all assets, and this is not always the case. We propose a different approach to this problem in which this assumption is not needed. To do this, the problem of finding the optimal transactions given some desired portfolio proportions is converted into a linear program (LP). The main contributions of this work are:

- Formulation of a trading system without the limitation of having a market with a fixed number of assets. Our method is sample-efficient, its implementation is straightforward, and during deployment, it is able to integrate assets into the process that appear suddenly in the market without the need of extra training.
- Transaction costs are considered and managed in our work. A novel algorithm to compute the optimal transactions is proposed and integrated into the system.
- Implementation of the proposed method using the dataset of a cryptocurrency market. The reliability of our method is tested under three different trading setups to show its adaptability.

The rest of this paper is organized as follows. Section 2 presents related works in the field. Section 3 describes the mathematical definition of the portfolio management problem. Section 4 describes the proposed method. Section 5 describes the transaction optimization process. Section 6 explains the experiment setups and metrics used to evaluate them. In Section 7, the results of the experiments are discussed. Finally, conclusions and directions for future work are presented in Section 8.

2. Related works

Deep learning approaches for portfolio management can be divided into two groups: model-based and model-free methods. Model-based methods, as their name suggests, assume models of the asset behavior exist, and deep neural networks (DNNs) are used to approximate these models using supervised learning on price datasets. Model-based methods do not cope with asset trading directly; instead, they require secondary methods to process the predicted prices, which typically use conventional heuristics based on human knowledge. Works that follow this approach include (Heaton et al., 2017; Niaki & Hoseinzade, 2013). Model-free solutions, on the other hand, compute trading actions without explicitly predicting prices. This is done by neural networks that directly map asset features into portfolio vectors. The training of these networks is typically formulated using RL, where financial performance measures such as daily returns, Sharpe ratios, maximum drawdowns, etc., are optimized to obtain agents that combine both profit-seeking and risk-aversion behaviors. Works that follow this strategy include (Dempster & Leemans, 2006; Moody & Saffell, 2001; Zhang & Maringer, 2013), which used RL with recurrent networks for portfolio management in stock and foreign exchange markets.

In recent years, algorithms such as deep Q-learning (Mnih et al., 2013) allowed researchers to train deeper neural network architectures using RL. Since then, deep RL approaches became dominant in portfolio management research. Jiang and Liang (2017) used a Monte Carlo policy gradient method to train a convolutional neural network (CNN) for cryptocurrency trading. They reported high returns, however they also stated to test their method in a real market, it had to be modified to include real life constraints. Bu and Cho (2018) trained a deep long–short-term-memory network using double Q-learning (Van Hasselt et al., 2016), obtaining positive rewards in a cryptocurrency market with a decreasing tendency. Pendharkar and Cusatis (2018) compared Q-learning and other value-based RL methods for asset trading in stock markets. Liang et al. (2018) proposed an adversarial training method that improves the performance of deep RL methods including actor–critic and policy-gradient based methods. They used a deep residual network (He et al., 2016) in their designs, and tested them on a Chinese stock market reporting positive returns. Jeong and Kim (2019) proposed a transfer learning method to pre-train neural networks when data amounts are not sufficiently large. They applied this technique along with Q-learning to trade financial indexes, such as the S&P500 and KOSPI, by augmenting an index dataset with data of stocks that have statistical similarities to the index. Aboussalah and Lee (2020) proposed a Gaussian process that searches for good neural network topologies for stock market trading. Wu et al. (2020) compared Q-learning and policy-gradient methods for stock market trading, finding

policy gradient methods give better results. Park et al. (2020) proposed a two-part network trained with Q-learning. The first part extracts features using a set of recurrent layers to process each asset separately. The second part, named the DNN regressor, is a small dense neural network that processes the extracted features and returns the portfolio. This work uses a similar strategy because this type of architecture saves memory resources. However, the second part of our design is different. The DNN regressor is fixed; hence, if new assets are suddenly added to the process, the second part has to be modified and retrained. Our architecture does not have this limitation. Lei et al. (2020) proposed a design similar to that of Park et al. (2020), but they used attention layers (Vaswani et al., 2017) along with recurrent layers for feature extraction.

The approach presented in this work falls into the model-free category, but is distinct from previous works in some key respects. The aforementioned approaches assume a fixed number of assets during training and testing. Therefore, adding new assets to those methods requires a change in design, since the input and output layers of those networks are a function of that fixed number. Additionally, if the number of assets in the market is large, the size of those networks may become inconveniently large as well. To make our design independent of the number of assets, the input layers of the network separately process each of them and the results are combined using weighting layers in the network output. These changes make our approach sample-efficient because the entire dataset is used instead of a segment of some specific number of assets. Additionally, the network does not need to be adjusted or retrained if new assets are suddenly introduced into the market.

3. Problem definition

This section introduces the concepts of market, investor and trading session, as well as the mathematical definition of portfolio management.

3.1. Market

A financial market is an environment where investors trade assets to obtain profit. The prices of assets change over time. Hence, by predicting these changes, investors can purchase assets which may increase in value and generate profit from those assets.

In all markets, there is at least one asset which is considered to keep its value constant over time. Prices of all assets are measured with respect to this special asset, and it can be used to purchase any asset in the market. Hence, it is referred to as cash. In stock and fiat currency markets, the most common cash asset is the U.S. Dollar (USD). In cryptocurrency markets, on the other hand, the most popular is the Tether[1] (USDT). This is because the USDT was designed to keep its exchange rate constant with respect to the USD (Berentsen & Schär, 2019); hence, cryptocurrency investors who want to close their positions or halt their trading sessions can rapidly exchange their assets for this risk-free asset.

3.2. Portfolio

In stock markets, the specific amounts of assets held by an investor are named *shares*. Thus, we termed the vector containing the specific amounts of assets held by an investor the *share vector*. The share vector is denoted by $\boldsymbol{q}$ and its entries are denoted as q_i, where $i \in \{0, 1, \ldots, n-1\}$ in a market with n assets. The entries are measured in cash units, and the index 0 is used for the cash asset. The sum of the entries of $\boldsymbol{q}$ is the 'total portfolio value'; it represents the money an investor would make by selling all the assets at that moment (without counting transaction costs), and it is denoted by Q. A vector containing the proportional contributions of each asset to the total portfolio value is named the portfolio vector, denoted by $\boldsymbol{p}$, and it is computed by taking the share vector divided by the total portfolio value, i.e. $\boldsymbol{q}/Q$. The entries of $\boldsymbol{p}$ are computed using Eq. (1).

$$p_i = q_i/Q \tag{1}$$

Only long trading is considered in this formulation. Investors begin a trading session with some capital, and all profits are obtained by purchasing assets that gain value during that session. Further, borrowing is not allowed, and neither is short selling nor margin trading. Hence, the entries of both $\boldsymbol{p}$ and $\boldsymbol{q}$ at every moment are non-negative, and the sum of the entries of $\boldsymbol{p}$ adds up to 1. Therefore, there must be at least one p_k and q_k strictly positive at every moment.

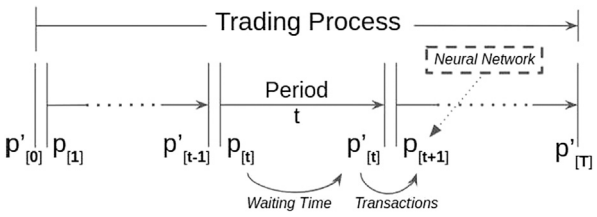

Fig. 2. Trading process diagram.

3.3. Trading session

In stock markets, trading sessions are specific hours during working days in which investors are allowed to exchange stocks. However, cryptocurrency markets do not have this limitation, since they are open 24 h. Nonetheless, we use the term *trading session* for the time in which the implemented algorithms are allowed to perform transactions in a simulated environment. In our experiments, trading sessions are subdivided in periods of equal length, which are named *holding periods*, since transactions are only allowed at the end of each of them.

The asset prices changing at every moment due to the interactions between investors and market is called *market dynamics*. Hence, portfolio vectors at the beginning and end of each period generally differ; let us denote these vectors by $\boldsymbol{p}_{[t]}$ and $\boldsymbol{p}'_{[t]}$, respectively, for some period t during a trading session. The entries of these vectors are related by Eq. (2), where $c_{i[t]}$ and $c_{i[t]}'$ are the prices of each asset at the beginning and end of period t, respectively. Eq. (2) states the change in the value of some asset i during period t is proportional to the amount held by the investor and the ratio between the prices at the beginning and the end of that period.

$$q_{i[t]}' = \frac{q_{i[t]} c_{i[t]}'}{c_{i[t]}} \tag{2}$$

At the end of each period, the recent history of the market is analyzed to propose a portfolio vector that is likely to increase its value in the subsequent period. The market history consists of prices, volumes, market capitalizations and other features of the assets recorded in the latest periods of the trading session. Fig. 2 depicts the steps of a trading session. In this study, the analysis of the market history and proposition of portfolio vectors is carried out by a deep neural network, which is trained using RL. The design of our network and the training process are described in Section 4.

In general, the portfolio vector chosen by the network for period $t+1$ and the portfolio held by the agent at the end of period t differ, because the purpose of the network is to propose assets that have the potential to increase their value in the near future, which are not necessarily those assets held by the agent at that moment. Therefore, some assets need to be exchanged to obtain the portfolio proposed by the network. The exchange is made in two steps. First, shares of some assets are sold,

[1] https://tether.to.

and the earnings from those transactions are added to the cash. Then, new assets are purchased using the accumulated cash.

However, all transactions have costs proportional to the traded amounts, and those costs decrease the total portfolio value. Thus, it is important to trade efficiently. The problem of finding a set of transactions that yield a desired portfolio, in this case the one proposed by the network, has multiple solutions in general, and different solutions give different decrements in portfolio value. Hence, the use of optimization techniques is necessary to obtain a set of optimal transactions. The ratio between the portfolio values, before and after the transactions, gives a measure of the loss due to those transactions; this ratio is shown in Eq. (3). The best set of transactions is the one leading to the maximum value of μ. The steps to obtain those optimal transactions are described in Section 5.

$$\mu_{[t+1]} = \frac{Q_{[t+1]}}{Q'_{[t]}} \tag{3}$$

Finally, the performances of investors are evaluated using the earnings and losses obtained during trading sessions. The main objective of investors is to obtain net increments in total portfolio value, i.e. profits from the invested capital. However, profit is not the only way to analyze the performance of a set of investments. The oscillations of the earning and losses seen by an investor during a trading session are used to measure the risk of those investments. Thus, our trading methods are evaluated using both profit-seeking and risk-aversion metrics, which are described in Section 6.2.

In short, due to the market dynamics, it is possible to obtain profits by trading assets. To do this, appropriate portfolios have to be periodically selected, and optimal transactions have to be computed and implemented to satisfy the selected portfolios. The details of our solutions to these problems are described below.

4. Proposed method

A trading session can be naturally formulated in the RL framework. In an RL process, an agent visits the states of an environment and takes actions in each visited state. In return, the environment gives rewards to the agent for taking those actions. After the agent executes an action, the environment evolves into a new state due to both the environmental dynamics and the action itself. An *episode* is the set of interactions between the agent and environment from its initialization until a final state is visited. The environmental dynamics are the set of rules by which the environment changes over time, which are not necessarily known by the agent, but can be indirectly measured using the data generated in the process, i.e. states, actions and rewards. These data are also used to improve the performances of agents in future episodes. This is typically done by optimizing a policy function which depends on the rewards received in previous episodes. This framework is suitable for asset trading because the concepts of the environment, agents and rewards are analogous to those of markets, investors and profits. A state consists of the market history at some specific point in time in which the investor is allowed to perform transactions. The action is the set of transactions made by the investor, guided by market history. The transition between states is a waiting period of time in which prices change due to the market dynamics. The rewards are the actual profits or losses obtained by the investor at the end of the period due to the transactions made. States, actions and rewards are represented by s, a and r, respectively. The trading process in the framework or RL is depicted in Fig. 3.

4.1. Portfolio management as an RL process

The iteration t of the trading process begins when the agent arrives at state $s_{[t]}$; this is depicted in the upper part of Fig. 3. State $s_{[t]}$ is the market history at that specific time, which is represented by a tensor with dimensions $n \times f \times k$, where n is the number of available assets

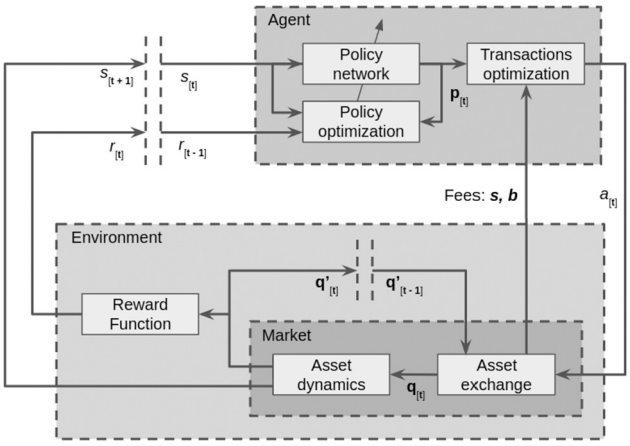

Fig. 3. Block diagram of the trading environment.

in the market, f is the number of features per asset, and k represents the latest steps of the process that the agent is allowed to observe. The chosen features f are: open, close, maximum and minimum prices in the period, volume, quote-asset volume and the entry of the portfolio corresponding to that asset, a total of 7 features.

The state $s_{[t]}$ is analyzed by the agent to decide on the best possible action $a_{[t]}$ to take. The action $a_{[t]}$ is a vector containing the shares of the assets to be sold and acquired for the next period. These values are computed by the *policy*. A policy is any map taking states as inputs and assigning actions as outputs. This map is typically implemented using neural networks. In this work, the policy has two parts: *policy network* and *transaction optimizer*. Our policy network has two gated-recurrent-unit layers (GRU) (Cho et al., 2014). The policy network is denoted by π_θ, where the subindex θ represents the weights of the network that need to be optimized. The output of the policy network is the proposed portfolio vector $\boldsymbol{p}_{[t]}$. The action $a_{[t]}$ is computed using the desired portfolio $\boldsymbol{p}_{[t]}$, the portfolio in that moment $\boldsymbol{p}'_{[t-1]}$, and the fees for trading the assets. This process is described in Section 5.

The action $a_{[t]}$, which carries the desired amounts to be sold and acquired, is executed in the *asset exchange* of the environment, resulting in a new set of assets satisfying the portfolio vector proposed by the network. The following step in the process is a waiting time, in which the state $s_{[t]}$ evolves into $s_{[t+1]}$, and consequently $\boldsymbol{q}_{[t]}$ evolves into $\boldsymbol{q}'_{[t]}$ due to the market dynamics. The entries of $\boldsymbol{q}'_{[t]}$ are passed through the reward function which gives the reward for step t. Finally, once the new state $s_{[t+1]}$ is reached, the reward $r_{[t]}$ is given to the agent and a new cycle begins.

The reward $r_{[t]}$ is a scalar value representing the performance of the agent in the current period. The reward is computed using a *reward function*, which in a trading environment depends on the earnings and losses received by the agent in recent periods. In this work, two financial measures are used for this purpose: the period return and the differential Sharpe ratio (Moody et al., 1998). Reward functions are analogous to overall performance measures, but they are computed for individual periods instead of the entire process. Hence, both reward functions and overall performance measures are described together in Section 6.

The process finishes when a maximum allowed number of states have been visited. When this happens, the rewards are used to compute the total discounted reward (R), shown in Eq. (4). This is a performance measure used to train the policy. The variable γ in the equation is named the discount factor, and it lies in the open interval $(0, 1]$, t is the index of each step in the process, and T is the index of the terminal state. The interpretation of Eq. (4) is agents value earlier rewards more than later ones. From a financial point of view, this means if large earnings are obtained early, the potential to obtain large final profits

increases, because profit depends on both the initial capital and enough time for the assets to gain value.

$$R = \sum_{t=1}^{T} \gamma^{t-1} r_{[t]} \tag{4}$$

4.2. Policy optimization

A policy is considered optimal if it maximizes the total expected discounted reward of the process. The problem of finding optimal policies is generally hard, but reasonable approximations are obtained using RL algorithms. Since our architecture uses an RNN, we chose Proximal Policy Optimization (PPO) (Schulman et al., 2017) to train our policies, which is an RL algorithm compatible with this type of architecture. PPO is an actor–critic algorithm, meaning it not only retrieves a policy π (actor), but it also retrieves a value-function $v_{\pi}(s)$ (critic), which estimates the value of the discounted reward the agent will receive at the end of the process following policy π starting from any state s.

On each iteration cycle of PPO, a specific number of copies of the policy (named *workers*) are created and assigned to copies of the environment. Workers interact with their environments during a specific number of steps, and data of those interactions are stored in the memory. These data are named *rollouts*, and are used to improve the policy at each iteration. Even though the workers are identical, the data collected by them differs because PPO uses stochastic policies during training. This means noise is added to the outputs of workers; consequently, when they land in the same state, they will take slightly different actions. Thus, adding variations to the generated data, which leads to training robust policies. Once the rollouts are complete, the data generated are mixed and divided into mini-batches, which are used to improve the policy by applying the Adam-optimizer (Kingma & Ba, 2014) to the PPO objective function, which is shown in Eq. (5). This objective function is equal to the *clip* objective function minus a constant coefficient multiplied by the value-function loss.

$$L_{PPO}(\theta) = L_{clip}(\theta) - w_v L_v(\theta) \tag{5}$$

The key feature of PPO is its clip objective function, shown in Eq. (6). In this formula, $u(\theta)$ represents the probability ratio between new and all policies. This ratio is computed dividing the likelihood of the action taken by the current policy at a state s by the likelihood of the previous policy choosing the same action at the same state; this is shown in Eq. (7). $\hat{A}_{[t]}$ in Eq. (6) is the generalized advantage (GA), introduced by Schulman et al. (2015), computed using Eqs. (8) and (9). The idea behind the PPO objective function is *small* differences between consecutive policies result in stable training processes. The *clip* function and hyper-parameter ϵ in Eq. (6) ensure that the difference between the actions of policies before and after an iteration cycle are enclosed in a small range.

$$L_{\text{clip}}(\theta) = \mathbb{E}\left[\min\left(u(\theta)\hat{A}_{[t]}, \text{clip}\left(u(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_{[t]}\right)\right] \tag{6}$$

$$u(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta'}(a|s)} \tag{7}$$

$$\hat{A}_{[t]} = \sum_{i=0}^{T-t-1} (\gamma\lambda)^i \delta_{[t+i]} \tag{8}$$

$$\delta_{[t]} = r_{[t]} - v_{\pi}\left(s_{[t]}\right) + \gamma v_{\pi}\left(s_{[t+1]}\right) \tag{9}$$

The value-function loss, shown in Eq. (10), is the squared error between the predictions of the value function and the real discounted returns obtained by the agents in the rollouts. The variable $R_{[t]}$ in the equation is the discounted reward computed from state t, and w_v is weight of the value-function loss. The expectations in these formulas are averages over the samples stored in the rollouts.

$$L_v(\theta) = \mathbb{E}\left[\left(v_{\pi}\left(s_{[t]}\right) - R_{[t]}\right)^2\right] \tag{10}$$

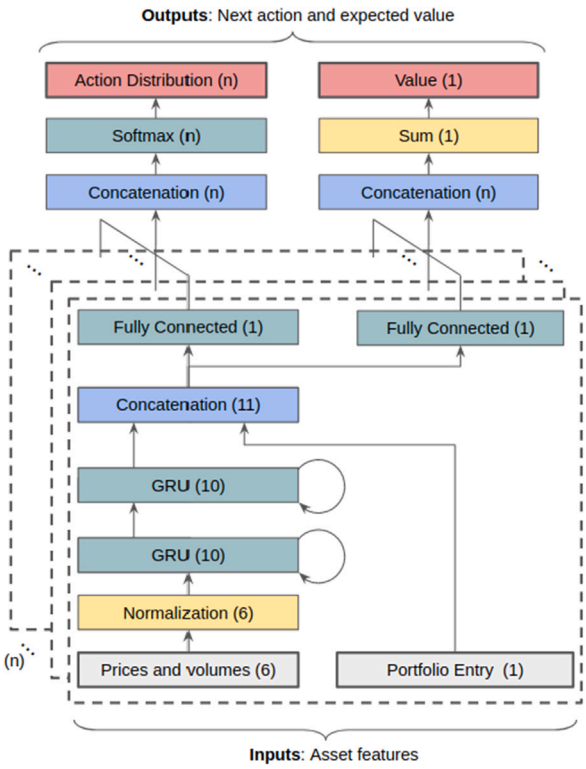

Fig. 4. Actor–Critic neural network architecture.

4.3. Market with a dynamic number of assets

Several changes were made to the original PPO architecture to make the process independent of the number of assets in the market. The schematic of our actor–critic architecture is shown in Fig. 4. The architecture allows both the observation space and action space to change dynamically depending on the number of assets. Note, the values of the inputs are normalized before being passed to the recurrent layers. The normalization is done for each asset individually, and it is done separating the features into two groups: price features (open, close, maximum and minimum) and volume features (volume and quote-asset volume). In both groups, features are mapped to values in the interval $[0, 1]$. This is done by a linear mapping which transforms the minimum and maximum values of each group to zero and one, respectively. The normalization map is shown in Eq. (11), where F represents a group of features and f an individual feature.

$$f_{normed} = \frac{f - min(F)}{max(F) - min(F)} \tag{11}$$

The feature tensor of each asset is passed through the network separately and independently. This is equivalent to cloning the network for each asset. In this way an independent evaluation of the potential to increase the value of each asset is obtained. Additionally, an artificial feature tensor is used for the cash-asset as well. In this case, the price features were set to 1 and the volume features to 0. This is done to give the network a reference point to evaluate the rest of the assets. Hence, a measure of potential growth is given to each asset as an output of the recurrent layers. These outputs are concatenated to the portfolio vector entries of each asset. These new features are passed through a linear layer and a softmax layer that retrieves the desired output vector. The reason for having these final layers is to penalize assets that are not held by the agent. This is done to avoid unnecessary transactions, since they reduce total portfolio value.

Note, the information is passed separately through the network for each asset, and it is only combined before the output layers; this makes

the network independent of the number of assets in the market. This architecture differs in several key respects to those designed for a fixed number of assets. Our design uses all the available data of the market, instead of only using the data of a small subset of assets. This makes the resulting network more robust at processing assets that were never seen by the network. Additionally, our network requires less memory because the input does not depend on the number of assets but only on the number of features. A possible disadvantage of our design is correlations between assets may not be found. However, our architecture is able to process higher amounts of data during training; therefore, a more general space of solutions is explored obtaining higher robustness than other approaches. The method used to train our agents is summarized in Algorithm 1. The values of all hyper-parameters are listed in Table A.1.

Algorithm 1 Trading with dynamic number of assets (DNA)

Inputs: $\mathcal{S}$ (# of environment steps) , $\mathcal{W}$ (# of parallel environments), $\mathcal{L}$ (# of rollouts per epoch), $\mathcal{B}$ (# of mini-batches, $\boldsymbol{b}$ and $\boldsymbol{s}$ (fee vectors).
Outputs: π_θ ▷ Policy network

```
Initialize action-state-reward buffer.
Reset all environments at random states.
for epoch i = 1, 2, ..., E do                          ▷ E = S/(W × L)
    for worker j = 1, 2, ..., W do
        for rollout k = 1, 2, ..., L do
            p_[t+1] ← π_θ(s_[t])
            a_[t] ← Trans-Opt(p'_[t], p_[t+1], b, s)   ▷ LP (1)
            Save to buffer: s_[t], a_[t], r_[t] and s_[t+1]
            if t + 1 = T then                          ▷ Terminal state
                Reset environment at a random state.
    Shuffle samples in buffer.
    for Mini-batch j = 1, 2, ..., B do
        s_[j], a_[j], r_[j], s_[j+1] ← draw samples from buffer.
        Â_[j] ← compute GA from samples.
        L_PPO ← compute using Eq. (10).
        π_θ ← Adam(L_PPO, α, β1, β2)                   ▷ Optimize π_θ
```

5. Transaction optimization problem

The outputs of the policy network are the entries of the portfolio vector $p_{i[t]}$. In general $p_{i[t]} \neq p'_{i[t-1]}$, consequently some assets have to be sold and others purchased to satisfy the desired portfolio vector. For simplicity, let us drop the time dependency of the expressions, since this is understood by context, for instance $p_{i[t]}$ and $p'_{i[t-1]}$ are written as p_i and p'_i, respectively. Let us also represent the shares acquired and sold per asset at some period by the non-negative variables $\hat{x}_i$ and $\hat{y}_i$, respectively. Then, the resulting amounts for non-cash assets due to the transactions at that period are given by Eq. (12). This formula tells the shares of any asset after the transactions are the shares before those transactions minus the sold amount plus the acquired amount. Intuitively, either $\hat{x}_i$ or $\hat{y}_i$ should be exactly zero because it does not make sense to sell an asset and then buy it again in the same period; however, the optimization algorithm described at the end of this section takes care of this issue.

$$q_i = q'_i - \hat{y}_i + \hat{x}_i, \ i \in \{1, 2, \dots, n-1\} \tag{12}$$

The shares sold for each asset are added to the cash, and that amount is used to buy new assets. However, both buying and selling assets reduce total portfolio value. These conditions are represented in Eq. (13), where b_i and s_i are the buying and selling fees for the ith asset, and lie in the open interval $[0, 1)$. Similar to the previous formula, Eq. (13) states the amount of cash after transactions is the amount before transactions plus the cash obtained by selling the undesired assets minus the cash used to purchase new assets. In addition, it contains the amounts subtracted from the cash due to transaction costs. Note, the losses due to transaction costs are proportional to both the traded shares and the fees for each asset.

$$q_0 = q'_0 + \sum_{j=1}^{n-1} \hat{y}_j - \sum_{j=1}^{n-1} \hat{x}_j - \sum_{j=1}^{n-1} s_j \hat{y}_j - \sum_{j=1}^{n-1} b_j \hat{x}_j \tag{13}$$

Eqs. (12) and (13) need to be modified to incorporate the variable μ defined in Eq. (3), which represents the decrement in total portfolio value that needs to be maximized to obtain a set of optimal transactions. Additionally, the entries of the vector $\boldsymbol{p}$ also have to be incorporated because this is the vector computed by the neural network. To obtain explicit expressions for μ and the entries of $\boldsymbol{p}$, Eqs. (1) and (3) were substituted into Eqs. (12) and (13), and the resulting expressions were divided by Q'. The results of these operations are shown in Eqs. (14) and (15), where x_i and y_i are a new set of variables defined by $x_i = \hat{x}_i/Q'$ and $y_i = \hat{y}_i/Q'$.

$$\mu p_i = p'_i - y_i + x_i, \ i \in \{1, 2, \dots, n-1\} \tag{14}$$

$$\mu p_0 = p'_0 + \sum_{j=1}^{n-1} (1 - s_j) y_j - \sum_{j=1}^{n-1} (1 + b_j) x_j \tag{15}$$

To optimize μ, an explicit function for this variable was obtained by adding the $n-1$ expressions represented by Eqs. (14) and (15), the resulting expression is shown in Eq. (16).

$$\mu = 1 - \sum_{j=1}^{n-1} s_j y_j - \sum_{j=1}^{n-1} b_j x_j \tag{16}$$

Note, Eqs. (14)–(16) are linear functions of μ, x_i and y_i; hence, using these equations the transaction optimization problem can be formulated as an LP. If a feasible solution exists for some LP, then the optimal solution to that LP can always be written as a closed-form expression using Dantzig's Simplex Method (Dantzig, 1998), or approximated up to any desired accuracy using other convex optimization techniques (Boyd & Vandenberghe, 2004).[2]

To reduce one decision variable in the problem, Eq. (16) is substituted into Eq. (14), resulting in Eq. (17), and in this way μ only appears explicitly in the objective function. Note, Eq. (15) is implicit in Eqs. (16) and (17), so it is omitted in the formulation.

$$\left(1 - \sum_{j=1}^{n-1} s_j y_j - \sum_{j=1}^{n-1} b_j x_j\right) p_i = p'_i - y_i + x_i \tag{17}$$

Additionally, no more than the available shares held by the investor can be sold. This condition is explicitly shown in Eq. (18). Similarly, the amount expended purchasing assets and paying transaction fees has to be at most the cash before the transactions plus the amount obtained by selling shares, therefore, Eq. (19) has to be satisfied as well.

$$q'_i \geq \hat{y}_i \tag{18}$$

$$q'_0 + \sum_{j=1}^{n-1} \hat{y}_j \geq \sum_{j=1}^{n-1} \hat{x}_j + \sum_{j=1}^{n-1} b_j \hat{x}_j + \sum_{j=1}^{n-1} s_j \hat{y}_j \tag{19}$$

Dividing Eqs. (18) and (19) by Q' and rearranging terms result in Eqs. (20) and (21) that are equivalent to Eqs. (18) and (19), but contain x_i and y_i, which are the decision variables of the LP described below.

$$p'_i \geq y_i \tag{20}$$

$$p'_0 \ \geq -\sum_{j=1}^{n-1} (1 - s_j) y_j + \sum_{j=1}^{n-1} (1 + b_j) x_j \tag{21}$$

The LP shown in Eq. (22) was stated using Eq. (16) as the objective function, Eq. (17) as the set of equality constraints and Eqs. (20) and (21) as the set of inequality constraints. This LP is always feasible; thus,

[2] There are plenty of LP solvers written in C++, Python and other languages available on internet at no cost.

an optimal solution for it can always be found efficiently using convex optimization techniques. A proof of this fact is given in Appendix B.

LinearProgram :

$$\textbf{Maximize}: \mu = 1 - \sum_{j=1}^{n-1} s_j y_j - \sum_{j=1}^{n-1} b_j x_j$$

Subjectto :

$$\begin{aligned}
&\textbf{1}. y_i - x_i + \left(\sum_{j=1}^{n-1} s_j y_j + \sum_{j=1}^{n-1} b_j x_j \right) p_i = p'_i - p_i \\
&\textbf{2}. y_i \leq p'_i \\
&\textbf{3}. -\sum_{j=1}^{n-1} (1 - s_j) y_j + \sum_{j=1}^{n-1} (1 + b_j) x_j \leq p'_0 \\
&\textbf{4}. y_i, x_i \geq 0, \text{for } i \in \{1, 2, \dots, n-1\}
\end{aligned} \tag{22}$$

6. Experiments

This section describes the setups of our experiments. These include dataset features, metrics for evaluating the algorithms and implementation details. We assume the amounts traded by our agents are sufficiently small that the prices of assets are not affected by these transactions, and the available shares in the market are large enough that transactions are executed immediately.

6.1. Dataset

The dataset of *Binance*[3] was used in all our experiments. Binance is one of the largest cryptocurrency markets in the world,[4] and its dataset can be accessed at no cost through the website's API. The data used in this work corresponds to the market history from 2017-08-17 to 2019-11-01. Only assets that can be directly exchanged for USDT were considered, these include Bitcoin, Ethereum, Litecoin and others. Binance market has grown rapidly since its creation in 2017. Note, during the selected period, the number of active assets that can be exchanged for USDT incremented from three to 85, as shown in Fig. 5. The dataset consists of asset features recorded in equal-length sampling periods. The dataset has multiple available sampling periods; three of them were chosen for our experiments: 30 min, six hours and one day. There are nine features available for each asset at each sampled period, which include four price features: open, high, low, and close prices; four volume features: standard, quote-asset, taker-buy asset and taker-buy-quote-asset volumes; and the number of trades in the sampled period. The first six features were used as inputs for our network. The data was divided chronologically into two parts. The first 60% of the data was used for training and the rest for testing. This type of division was chosen because investors use past price behavior of assets to predict future tendencies or repetition of patterns. The test dataset corresponds to 11 months of asset history. Hence, the results can give a general idea of the expected average performances of our methods at any point of the year. Missing data were completed using linear interpolation. Fees were set as in the Binance exchange website, 0.1% for all assets except BNB (coin), which has a tax of 0.05%.[5] Table 1 summarizes the main properties of the dataset.

6.2. Metrics

Two metrics were used in this study for evaluating the profit-seeking and risk-aversion behaviors of the implemented algorithms: the total return (TR) and the Sharpe ratio (SR) (Sharpe, 1994). The total return is the total profit or loss obtained by the investor in a trading session, and it is computed using Eq. (23). This measure only considers the portfolio values at the beginning and end of a session. Thus, it gives high scores to trading strategies that seek high profits.

$$TR = \frac{Q_{[T]} - Q'_{[0]}}{Q'_{[0]}} \tag{23}$$

On the other hand, the SR was used to evaluate the risks taken by the algorithms during the trading sessions. The SP is computed using Eq. (24), where the mean and standard deviation are taken for the period returns (PR) of all steps in the trading session. PRs are computed using Eq. (25). The risk of an investment is the uncertainty the investor has about the future price of an asset that he or she intends to buy or sell, which is hard to evaluate. Instead, the SR measures the risk of a set of investments after observing the outcome of them. This is done by computing the statistical measures of the earnings and losses at each period, which are represented by PR. Note, PR and TR are similar expressions; the difference between them is PR is computed for each individual step and TR is computed for the entire process. The SP and the TR favor algorithms which seek high returns; however, the difference between them is SR penalizes portfolios that experiment large oscillations in gain in the trading sessions.

$$SR = \frac{\underset{t}{\text{mean}}(PR_{[t]})}{\underset{t}{\text{std}}(PR_{[t]})} \tag{24}$$

$$PR_{[t]} = \frac{Q_{[t]} - Q'_{[t-1]}}{Q'_{[t-1]}} \tag{25}$$

6.3. Reward functions

The value of $r_{[t]}$ is a measure of performance of an agent at period t. Any real scalar function that maps good performances to high values can be used to compute rewards. In trading environments, the most

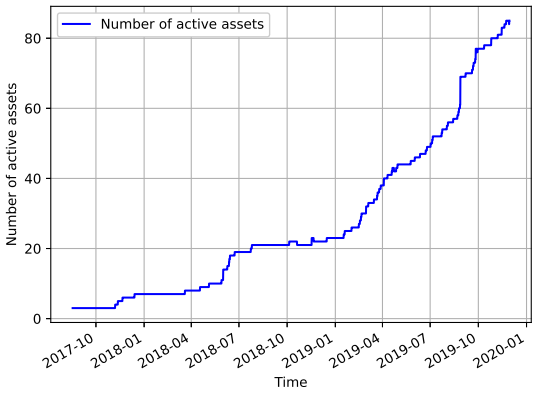

Fig. 5. Number of active assets in Binance market in the period: 2017-08-17 to 2019-11-01 (only assets directly exchangeable for USDT were counted).

Table 1
Dataset properties.

Feature	Value
Total days	806
Training days	484 (60%)
Test days	322 (40%)
# of assets	3-85
# of features	9
Fees (except BNB)	0.1%
Fees BNB	0.05%
# of entries per sampling period	
30 minutes	38688
six hours	3224
one day	806

[3] www.binance.com.
[4] According to Coin Market Cap (www.coinmarketcap.com).
[5] This discount changes after the first 12 months of use of the platform.

Table 2
Trading session lengths and holding periods.

Session length	Holding period	# of periods
1 day	30 min	48
14 days	6 h	56
30 days	1 day	30

Table 3
Summary of the implemented algorithms.

Algorithm	# of assets	# of features	Description
TD(λ) (Pendharkar & Cusatis, 2018)	2	2	Value function
CNN (Jiang & Liang, 2017)	12	14	Policy gradient
DQN (Bu & Cho, 2018)	8	8	Double-Q learning
DNA-R (ours)	3 - 85	6	PPO, period return
DNA-S (ours)	3 - 85	6	PPO, Diff. Sharpe ratio

natural choice for this purpose is the PR (Eq. (25)), which is the profit or loss in each period. However, this function does not consider risks. The issue of finding a risk function that can be computed at each step, and can be used as reward function in RL processes, was first studied by Moody et al. (1998). They derived an approximation for the contributions to the SP at each step, termed the differential Sharpe ratio (DSR), which is computed using Eq. (26). They reported agents trained with the DSR performed better than those trained with measures that do not consider risks. Both PR and DSR were used to train our agents. We named our agents DNA for the *dynamic number of assets*. The agent trained with the PR is named DNA-R, and the one trained with the DSR is named DNA-S.

$$
\begin{aligned}
DSR_{[t]} &= \frac{B_{[t-1]}\Delta A_{[t]} - 0.5A_{[t-1]}\Delta B_{[t]}}{\left(B_{[t-1]} - A^2_{[t-1]}\right)^{3/2}} \\
A_{[t]} &= A_{[t-1]} + \eta\left(r_{[t]} - A_{[t-1]}\right) \\
B_{[t]} &= B_{[t-1]} + \eta\left(r^2_{[t]} - B_{[t-1]}\right)
\end{aligned}
\tag{26}
$$

6.4. Experiment setups

Three different trading session lengths were chosen for our backtest experiments: 24 h, 14 days and 30 days, each of them with holding periods: 30 min, six hours and one day, respectively. These setups were chosen to test the robustness of our methods to changes in holding periods and trading session lengths. The setups are summarized in Table 2.

During training, episodes were drawn randomly across the train dataset. For testing, the algorithms were run through all the episodes of the test dataset; the performances shown in this work are the average TR and SR obtained by each algorithm during testing.

6.5. Baselines

Our agents were compared to methods presented in (Bu & Cho, 2018; Jiang & Liang, 2017; Pendharkar & Cusatis, 2018); these baselines were described in the Section 2. The main features of our agents and baselines are summarized in Table 3. All baselines trade a fixed number of assets, which are the cash asset and those assets with the highest capitalizations in the market.

7. Results

In the first experiment, which corresponds to trading sessions of 1 day and holding periods of 30 min, DNA-S obtained the highest results; these are shown in Table 4. DNA-S obtained an average TR of 0.224, which is more than double the score of the closest competitor CNN, which obtained 0.088. DNA-R obtained the third best results with an average TR of 0.041. The other two approaches: DQN and TD(λ) obtained the lowest scores, with average TRs close to zero. The average SRs obtained by all the algorithms correlate to their TRs, for instance DNA-S has the best average SR among the competing algorithms, which is 0.468. Hence, in this setup, DNA-S not only sought high profits, but also managed investment risks better than its competing approaches.

Table 4
Score summary for the trading sessions with length: one day and holding periods: 30 min.

Algorithm	Total Return	Sharpe Ratio
TD(λ) (Pendharkar & Cusatis, 2018)	0.014 ± 0.021	0.064 ± 0.107
CNN (Jiang & Liang, 2017)	0.088 ± 0.086	0.274 ± 0.127
DQN (Bu & Cho, 2018)	−0.019 ± 0.024	−0.180 ± 0.176
DNA-R (ours)	0.041 ± 0.042	0.166 ± 0.153
DNA-S (ours)	**0.244 ± 0.154**	**0.468 ± 0.200**

Table 5
Score summary for the trading sessions with length: 14 days and holding periods: 6 h.

Algorithm	Total Return	Sharpe Ratio
TD(λ) (Pendharkar & Cusatis, 2018)	0.224 ± 0.183	0.317 ± 0.125
CNN (Jiang & Liang, 2017)	0.680 ± 0.257	0.411 ± 0.088
DQN (Bu & Cho, 2018)	1.108 ± 0.503	**0.618 ± 0.088**
DNA-R (ours)	**2.283 ± 1.371**	0.533 ± 0.095
DNA-S (ours)	0.468 ± 0.314	0.336 ± 0.150

Table 6
Score summary for the trading sessions with length: 30 days and holding periods: one day.

Algorithm	Total Return	Sharpe Ratio
TD(λ) (Pendharkar & Cusatis, 2018)	0.329 ± 0.265	0.404 ± 0.126
CNN (Jiang & Liang, 2017)	1.012 ± 0.495	0.527 ± 0.143
DQN (Bu & Cho, 2018)	1.151 ± 0.521	0.661 ± 0.158
DNA-R (ours)	**7.116 ± 4.566**	**0.758 ± 0.186**
DNA-S (ours)	2.798 ± 2.065	0.609 ± 0.201

The results of the second experiment (trading sessions of 14 h and holding periods of 6 h) show important contrasts with respect to the previous experiment. DNA-R obtained the best average TR in this setup, as shown in Table 5, but it did not obtain the best average SR. The average TR of DNA-R in this setup was 2.283, but the standard deviation was 1.371. This indicates the performances of this algorithm over the course of a year could have large variations. The SR obtained by this algorithm supports this observation. The best average SR in this setup belong to DQN (0.618). Hence, DQN had higher stability in the increments in portfolio value in the trading sessions. However, this approach only obtained half as much average profit (1.08) as that of DNA-R. CNN and DNA-S have the next best scores, which is surprising, since these two approaches obtained the best results in the previous experiment. Again, TD(λ) had the lowest performance. Nonetheless, its results are better in this setup than in the previous one.

In the third setup (trading sessions of 30 days and holding periods of one day), our agents obtained significantly higher scores than those obtained by the baselines. These results are shown in Table 6. DNA-R and DNA-S obtained the highest average TRs: 7.116 and 2.798, respectively. On the other hand, the results of the baselines are actually very similar to those of the previous experiment, even though the trading session was twice as long. The SR scores correlate to those of the TR, except for DQN, which obtained a higher score than DNA-S.

In general, our methods adapted better than the baselines to changes in holding periods and trading session lengths. Our experiments show DNA-R is the most robust approach for both profit-seeking and risk-aversion. The TRs and SRs obtained by this agent are good throughout all setups, even though DNA-S had the best performance in the setup with the shortest holding period. We also found being able to analyze a large pool of assets is beneficial for the agent. Tables 4–6 show the

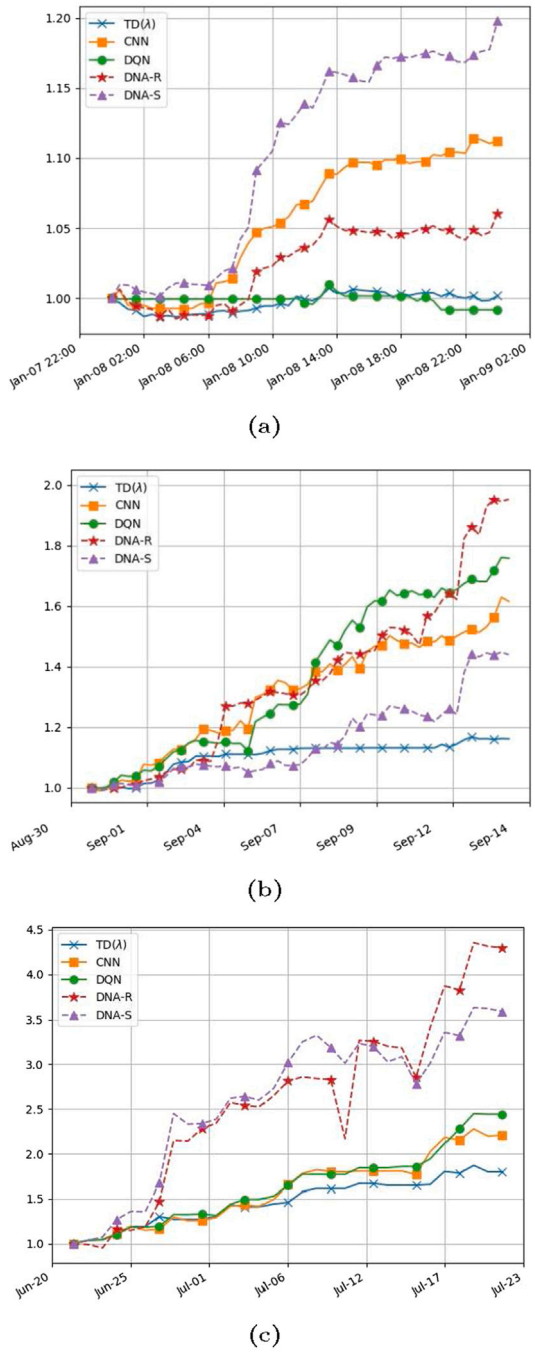

Fig. 6. Examples of tests on the three setups: (a) Trading session with length: one day and holding periods: 30 min (January, 2019). (b) Trading session with length: 14 days and holding periods: 6 h (September, 2019). (c) Trading session with length: 30 days and holding periods: one day (June and July, 2019).

correlation between the number of assets processed by the competing approaches and their overall performances, for instance, the agent that performed worst in all setups was TD(λ), which processed only two assets. Fig. 6 depicts the evolution of the portfolio values of all competing approaches in the three setups.

Based on these results, we believe our method is a step forward in the automatic trading of cryptocurrencies. In this work, we included transaction costs to make our simulations as close to real markets as possible. Trading assets is risky, especially cryptocurrencies which are extremely volatile. Our recommendation for institutional investors is to use these methods along with loss limiting functions, such as stop-loss, to mitigate the volatility of assets. Additionally, since it was assumed the traded amounts have to be sufficiently small that prices of assets are not affected, we recommend trading only small amounts. Further research is necessary to assess the impact of the size of the investments in automatic trading with RL.

Table A.1
List of hyperparameters.

Hyperparameter	Value
# of environments ($\mathcal{W}$)	10
# of mini-batches ($\mathcal{B}$)	10
# of environment Steps ($\mathcal{E}$)	100000
# of Rollouts per epoch ($\mathcal{L}$)	10
# of optimization steps	10000
PPO decay rate (γ)	0.99
PPO value-loss coefficient (w_v)	0.99
PPO clip parameter (ϵ)	0.2
PPO noise distribution $\mathcal{N}(\mu, \sigma^2)$	0, 0.01
GA parameter (λ)	0.95
# of recurrent layers	2
Recurrent layers length	10
Adam-Optimizer (α, β_1, β_2)	7e−4, 0.9, 0.999
DSR parameter (η)	0.01

8. Conclusions and future work

We introduced a method that performs portfolio management on markets with transaction costs in which the number of assets is dynamic. Even though our method is able to integrate new assets into the process during deployment, it does not require extra training or memory and its implementation is straightforward. Our method was tested on a cryptocurrency market outperforming state-of-the-art methods under three distinct setups. Additionally, a novel algorithm to compute transactions with minimal costs, formulated as an LP, was given in this work.

Future works include the implementation of the proposed method into traditional markets such as stocks and fiat currency markets. Additionally, due to its straightforward implementation, our method is compatible with advanced trading strategies such as limit, trailing and stop-loss; hence, future directions can also include the integration of the mentioned mechanisms to the method.

CRediT authorship contribution statement

Carlos Betancourt: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Wen-Hui Chen:** Resources, Writing - review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Hyperparameters

See Table A.1.

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.eswa.2020.114002.

References

Aboussalah, A. M., & Lee, C.-G. (2020). Continuous control with stacked deep dynamic recurrent reinforcement learning for portfolio optimization. *Expert Systems with Applications*, *140*, Article 112891.

Berentsen, A., & Schär, F. (2019). Stablecoins: The quest for a low-volatility cryptocurrency. In *The economics of fintech and digital currencies* (pp. 65–71). CEPR Press.

Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization*. Cambridge university press.

Bu, S.-J., & Cho, S.-B. (2018). Learning optimal q-function using deep boltzmann machine for reliable trading of cryptocurrency. In *International conference on intelligent data engineering and automated learning* (pp. 468–480). Springer.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In Conference on empirical methods in natural language processing (EMNLP 2014).

Corbet, S., McHugh, G., & Meegan, A. (2014). The influence of central bank monetary policy announcements on cryptocurrency return volatility. In *Investment management and financial innovations 14, Iss. 4* (pp. 60–72). Business Perspectives, Publishing Company.

Dantzig, G. B. (1998). *Linear programming and extensions*. Princeton university press.

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

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778). IEEE.

Heaton, J., Polson, N., & Witte, J. H. (2017). Deep learning for finance: Deep portfolios. *Applied Stochastic Models in Business and Industry*, *33*, 3–12.

Jeong, G., & Kim, H. Y. (2019). Improving financial trading decisions using deep q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications*, *117*, 125–138.

Jiang, Z., & Liang, J. (2017). Cryptocurrency portfolio management with deep reinforcement learning. In *2017 Intelligent systems conference (IntelliSys)* (pp. 905–913). IEEE.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. CoRR, abs/1412.6980.

Lei, K., Zhang, B., Li, Y., Yang, M., & Shen, Y. (2020). Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading. *Expert Systems with Applications*, *140*, Article 112872.

Liang, Z., Chen, H., Zhu, J., Jiang, K., & Li, Y. (2018). Adversarial deep reinforcement learning in portfolio management. arXiv preprint arXiv:1808.09940.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

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

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

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies: A comprehensive introduction*. Princeton University Press.

Niaki, S. T. A., & Hoseinzade, S. (2013). Forecasting s&p 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International*, *9*(1).

OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., & Zaremba, W. (2020). Learning dexterous in-hand manipulation. *International Journal of Robotics Research*, *39*, 3–20.

Ormos, M., & Urbán, A. (2013). Performance analysis of log-optimal portfolio strategies with transaction costs. *Quantitative Finance*, *13*, 1587–1597.

Park, H., Sim, M. K., & Choi, D. G. (2020). An intelligent financial portfolio trading strategy using deep q-learning. *Expert Systems with Applications*, Article 113573.

Pendharkar, P. C., & Cusatis, P. (2018). Trading financial indices with reinforcement learning agents. *Expert Systems with Applications*, *103*, 1–13.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Sharpe, W. F. (1994). The sharpe ratio. *Journal of portfolio management*, *21*, 49–58.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In Thirtieth AAAI conference on artificial intelligence.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Wu, X., Chen, H., Wang, J., Troiano, L., Loia, V., & Fujita, H. (2020). Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*.

Zhang, J., & Maringer, D. (2013) Indicator selection for daily equity trading with recurrent reinforcement learning. In Proceedings of the 15th annual conference companion on Genetic and evolutionary computation (pp. 1757–1758).