MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Marcell Németh

Deep reinforcement learning based algorithmic trading with ensemble models

Scientific Students' Association Report

SUPERVISOR

Dr. Gábor Szűcs

BUDAPEST, 2021

Contents

Abstract

In the financial sector, any type of trading is considered, algorithmic trading where the determination of the order parameters (timing, price, quantity) as well as the order management is performed by a computer algorithm, with or without limited human intervention. The research resulting in the dissertation aims to find ways to develop and refine deep reinforcement learning models to execute trading that maximize revenue and minimize investment risk.

Developments using deep reinforcement learning in the literature have already demonstrated the viability of machine trading algorithms in portfolio management. The aim of the present research is to create a more robust reinforcement learning model that can simulate trading with varying risk appetite in line with market dynamics while maximizing portfolio value and minimizing risk and other negative consequences.

The dissertation begins with a theoretical overview of state-of-the-art methods of reinforcement learning, paying special attention to the practical advantages and disadvantages of each algorithm and model. This is followed by the introduction of an ensemble trading model that provides the baseline for new developments, along with a description of the simulated stock market environment, factors and data used to model market dynamics.

The main focus of the research is on a new ensemble technique based on the distribution of features and the optimization of decision-making by agents learning in parallel, which is able to outperform the results of the baseline model. Initially, a single agent, trained in the space formed by all input features, is replaced by several models, the input factors of which cover a group of technical indicators compiled on the basis of targeted statistical methods. Parallel learning models can use each group to make an unbiased decision by observing some distinct aspects of the market dynamics, which are aggregated at the very end of the Markov process, resulting in a robust decision. Further developments are presented by discussing the reward functions of each model. The distributed operation of the agents also allowed for different reward maximization options: different target functions allow for more flexible management of trading parameters, thus providing a greater degree of transparency regarding the operation of the model.

Összefoglaló

A pénzügyi szektorban algoritmikus kereskedésnek számít minden olyan kereskedési mód, ahol a megbízás paramétereinek (időzítés, ár, mennyiség) meghatározását, valamint a megbízás kezelését egy számítógépes algoritmus végzi, korlátozott emberi beavatkozással vagy emberi beavatkozás nélkül. A TDK dolgozatot eredményező kutatás célja olyan módszereket találni, mély megerősítéses tanulási modelleket kidolgozni és fejleszteni a kereskedés végrehajtására, amelyekkel maximalizálható a bevétel és minimalizálható a befektetés kockázata.

A szakirodalomban mély megerősítéses tanulást alkalmazó fejlesztésekkel már bizonyították a gépi kereskedő algoritmusok létjogosultságát a portfóliók kezelésében. Jelen kutatás célja, hogy hogyan lehet minél robusztusabb megerősítéses tanuló modellt készíteni, amely képes változó kockázatvállalási hajlandóságú kereskedést szimulálni a piac dinamikájához igazodva a portfólió értékének maximalizálása és a kockázat, illetve egyéb negatív következmények minimalizálása mellett.

A dolgozat a megerősítéses tanulás state-of-the-art módszereinek elméleti áttekintésével kezdődik, kiemelt figyelmet fordítva az egyes algoritmusok, modellek gyakorlati előnyeire és hátrányaira. Ezt követően bemutatásra kerül az új fejlesztések alapját biztosító kiinduló ensemble kereskedő modell a szimulációs részvénypiaci környezet, a piac dinamikájának modellezéséhez használt faktorok és adatok ismertetése mellett.

A TDK munka központi része egy új, a feature-ök szétosztásán és párhuzamosan tanuló ágensek döntés optimalizálásán alapuló ensemble technikát mutat be, amely képes az alapot jelentő fejlesztés eredményeinek felülteljesítésére. A kezdetben egyetlen ágenst, amely az összes bementi feature által alkotott térben tanult, leváltja több modell, amelyek bemeneti faktorai célzottan, statisztikai módszerek alapján összeállított, egy-egy technikai indikátor csoportot fednek le. A párhuzamosan tanuló modellek az egyes csoportokat felhasználva a piac dinamikájának egy-egy részét "látva" tudnak elfogulatlan döntést hozni, amelyek a Markov-folyamat legvégén aggregálásra kerülnek, így eredményezve egy robosztus döntést. További fejlesztések kerülnek tárgyalásra az egyes modellek jutalomfüggvényeit taglalva. Az ágensek elosztott működése egyben különböző jutalom maximalizálási lehetőségeket is lehetővé tett: eltérő célfüggvények segítségével rugalmasabban kezelhetőek a kereskedési paraméterek, amelyek ezáltal nagyobb fokú átláthatóságot is biztosítanak a modell működését illetően.

1 Introduction

One of the most popular applications of reinforcement learning is algorithmic stock trading. A profitable stock exchange trading strategy is vital for banks and investment funds, to the success of which the connections recognized by machine learning can greatly contribute.

In order to use reinforcement learning algorithms for algorithmic stock trading, we need to formulate the problem as a Markov decision process and our trading goal as a maximization problem. it is necessary to define the state space of the environment, the possible activities, and the reward function.

1.1 Research objectives

The effectiveness of reinforcement learning methods in trading has already been supported by numerous research and publications [1][9][11][14][25]. A common feature of all such methods is that they use a single complex feature space, i.e., the technical indicators used to determine the states are used „in bulk”.

One of the basic ideas of the present research is whether it would not be worthwhile to separate the factors describing the states into predefined function groups. The basic premise is that agents trained in parallel using segregated feature groups can extract more information using separate, sub-feature spaces, and can make significantly more robust decisions in predictions.

The research also seeks to solve a problem closely related to the first objective: how to construct a reward function and optimize the decisions of ensemble agents in terms of risk management so that they adapt to the volatile dynamics of the market while maximizing profits and minimizing negative factors (e.g., drawdown).

1.2 Trading a single stock

First, let’s take the case where we only want to trade the shares of a single company on the stock exchange and create our model for that. Here, the state space of the environment is determined by the share price of the given company, and we can also use the previous share prices as well as technical indicators as input data.

The model can basically perform three activities in this environment: sell, hold, or buy stock, which is usually given in the following form, respectively: {-1, 0, 1}.

However, we can also define the activities so that the model determines the number of shares to be purchased, for example, we can use the vector {-10, ..., -1, 0, 1,…, 10} to define the state space in which our model is able to sell or buy 10 shares [11].

Finally, we need a definition of the reward function, which is the incentive mechanism of our model to learn to act better. In stock exchange trading, we can simply determine the reward function by subtracting the value taken in the previous state from the new value of the portfolio [11]. The goal is to design a trading strategy that maximizes the positive change in the value of our portfolio in a real environment.

In one step, the model receives the state of the environment, which includes the current and previous share price as well as the technical indicators used. Based on this, the model selects an activity, and the next share price can be used to determine the goodness of the activity with which we can train the model.

1.3 Trading multiple stocks

In case we no longer want to trade not only shares of a single company, but also take into account the shares of several companies, we need to change our model and take into account that the size of the state space and the activity space increases exponentially in proportion to the number of shares.

In this case, the state space is no longer determined only by the share price of a company, but also by the share prices of all the companies taken into account, as well as the technical indicators calculated from them. Generally speaking, if we trade *c* shares of a company and describe each share with *l* features, then the state space is defined by a vector $(1 + l)^c + 1$ long (the number of shares we own for each company and our available capital) [11].

The scope of activities also changes by considering more stocks. By trading a stock, we were able to describe the space of the actions to be performed with the vector *{-k,…, -1, 0, 1,…, k}*, where *k* determined how many shares our model could trade at once. In general, by trading *c* shares, we can define the same set for each firm, so that a vector of length $(2k + 1)^c$ defines the space for activities.

However, the reward function does not change if we increase the number of shares. All we have to do is change the definition of the value of the portfolio, where we already have to multiply the number of all the shares we hold by the price of the shares and add our available cash to that.

In one step, the model determines for each stock how much to buy, sell, or just hold the shares based on the stock prices of the companies. Based on the new exchange rate, the goodness of the decision can be determined, with which we can train our model.

The environment thus defined can already be used for reinforcement learning. We can also train several reinforcement learning algorithms using the environment, which we can test to select the best one to start trading on the stock market.

2 Theory of reinforcement learning

2.1 Basic concepts

Machine learning has three basic paradigms: supervised learning, unsupervised learning, and reinforcement learning [17]. In the case of supervised learning, the goal of machine learning is to learn a function that determines the output data from the input data, all based on labeled examples. In contrast, for unsupervised learning, we look for unspecified patterns in the data. In the case of reinforcement learning, we place our machine learning model in an environment (as can be seen in Figure 2.2.1) in which the model learns how to maximize its reward in that environment. The reward received at time *t* is denoted by R_t, the state by S_t, and the action by a_t.

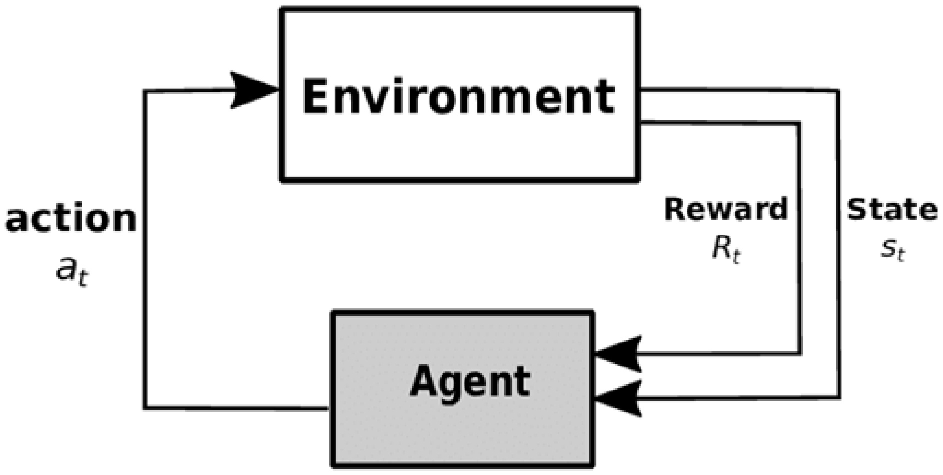


Figure 2.1.1. Process of reinforcement learning [2]

There are machine learning algorithms that try to build a model of the environment based on the information gathered about it and design their behavior based on that model. The other group is model-free algorithms, which try to determine the optimal strategy in the environment without building an environment model. Based on the machine learning model, we can classify reinforcement learning algorithms into several categories.

Utility-based models (utility-based or also known as value-based) learn the state-based utility function and use this to select their actions to maximize the expected value of the utility available.

Instead, policy-based models try to define an action-value function, also known as a Q-function, that can be used to clearly define how to act in a given state, since the Q-function assigns some expected reward in a given state for a given action. Utility-based algorithms must also have some model of the environment, but a Q-learning algorithm can compare its possible choices without knowing what they are leading to, so one doesn't need a model of the environment. Actor-critic algorithms [20] are both policy-based and utility-based, storing both the utility function and the action value function, and both are used to calculate the best decision.

2.2 Markov decision process and the Bellman equation

In the case of reinforcement learning, the best studied topic is when the problem can be formulated as a Markov decision process [2] (MDP). In the case of MDP, the machine learning model is able to visit a finite number of states, where it receives a reward (a negative number means the penalty). From each state s, other states can be accessed by performing actions. Each state has a constantly changing value, which indicates how much reward the model can collect from that state until it reaches the end state of the environment. The activities to be performed in the states are determined according to a strategy (policy), which can also change as the learning process progresses. The purpose of the model is to maximize the reward with the activities performed until it reaches the end state of the environment.

Each of the states in the Markov decision process is a Markov state. A state can be called Markov if the future is independent of the past, thus if our model is in a Markov state, it is not necessary to know the series of previous states, only the current state. Mathematically, a state s_t (at time t) is a Markov state if and only the state s_t carries the information of all previous states:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, \dots, s_t) \tag{2.2.1}$$

The transition probability between a Markov state s and a successor state s' can be determined as follows: $P_{ss'} = P(S_{t+1} = s'|S_t = s)$. This can be stored in the form of a matrix P, where $P_{i,j}$ is equal to the value of the transmission probability between s_i, s_j, so that the sum of each row of the matrix P becomes 1.

A Markov Reward Process (MRP) is a tuple of (S, P, R, γ), where S is the set of finite states with Markov states, P is the state transfer probability matrix, R is the reward function and γ is discount factor, where $\gamma \in [0, 1]$.

The reward function determines how much immediate reward the agent gets when it reaches a certain state. The purpose of the model is to maximize G_t, which is the total discounted reward, G_t can be determined using R and γ as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad (2.2.2)$$

This value is definitely finite, since the value of γ is between 0 and 1. In the case where γ is close to zero, the model prefers the immediate reward over future rewards, while if it is close to one, it is just the opposite.

The Markov decision process is a value of *(S, A, P, R, γ)* where *(S, P, R, γ)* forms an MRP and A (action) is a finite set of actions. Using the elements of this set, the model can move from one state of the environment to another. The behavior of the model used to select these activities can be determined by the π policy function, which can be written as follows:

$$\pi(a|s) = P[A_t = a|S_t = s] \qquad (2.2.3)$$

The machine learning model is designed to maximize the rewards it collects. To achieve this, we need to determine the optimal utility-value function, which can be done by the Bellman equation [3]. The Bellman equation describes that the value (utility value) of a state can be decomposed as the sum of the discounted value of the immediate reward and the following state:

$$\begin{aligned} V(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} \ldots |S_t = s] = \qquad (2.2.4) \\ = \mathbb{E}[R_{t+1} + \gamma \mathrm{V}(s_{t+1})|S_t = s] \end{aligned}$$

The value function thus shows how good it is for our model to be in a particular state of the environment, but it does not say anything about which activities to select. To do this, we need the action value function, which expresses how good it is to choose an activity in a given state s, and we can use it later to determine the best activity for the current state:

$$q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|S_t = s, A_t = a\right] \qquad (2.2.5)$$

The purpose of most reinforcement learning algorithms is to determine the action value function and select the optimal action from it in any state (optimal means that the activity with the highest value should be selected, as shown in the formula below):

$$q_*(s,a) = \max_\pi q_\pi(s,a) \qquad (2.2.6)$$

Using this function, we can determine the maximum reward following the π strategy. If we know the optimal action value function then the optimal strategy can be determined as follows:

$$\pi_*(a|s) = \begin{cases} 1, if\ a = \underset{a \in A}{argmax}\, q_*(s,a) \\ 0\ otherwise \end{cases} \tag{2.2.7}$$

The optimal action value function can be written recursively with the Bellman optimality equation in the following form, where R_s^a means the reward at state s taking action a and $P_{ss'}^a$ means the transition probability between state s and s' taking action a:

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s',a') \tag{2.2.8}$$

The Bellman optimality equation cannot be written in closed form, however, there are iterative approaches that can be applied, such as the SARSA and Q-learning algorithms, which are presented in the next section.

2.3 The main algorithms of reinforcement learning

2.3.1 TD-learner

Temporal difference (TD) learners [21] is a group of reinforcement learning algorithms without a model, i.e., these algorithms do not model their environment but take samples from it and update based on it.

The simplest algorithm for a TD learner is the TD(0) algorithm. In this case, the value function is updated after each sampling with the immediate reward and the discounted value of the next state. This algorithm is useful in cases where the environment has no end state because the value function is updated after each activity based on the following formula:

$$V(s_t) = V(s_t) + \alpha\big(R_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big) \tag{2.3.1}$$

Here, the parameter α is the learning rate, which has a value between 0 and 1. With this parameter we can determine how fast the model learns new information.

Another popular TD learning algorithm is TD(1). In this case, the value function is updated after an episode until the end state of the environment is reached. The update of the value function is described by the following formula:

$$V(s_t) = V(s_t) + \alpha\big(G_t - V(s_t)\big) \tag{2.3.2}$$

In case the environment has no end state but does not want to update the value function after each step, we can use the TD(λ) algorithm, where λ is between 0 and 1 and determines the length of the path used for the update. The higher the value of λ, the longer the length of the path used for the update.

2.3.2 SARSA

The SARSA (which stands for State, Action, Reward, State, Action) algorithm [16], unlike the TD learner, learns the action value function. The name of the algorithm suggests that the value of the q-function is determined by the current s_t state, the selected activity *t*, the collected *r* reward, the next s_{t+1} state, and the subsequent *t+1* activity. The policy function is updated according to the following formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \qquad (2.3.3)$$

In this algorithm, it is important to determine the initial values of the q-function. The most commonly used method is to use the first reward received when discovering the environment to determine its q-value.

2.3.3 Q-learner

The Q-learning algorithm [21], like SARSA, defines a q-function. The Q-learning algorithm belongs to the group of off-policy algorithms, which means that it does not learn exactly the same strategy as it uses to select action. (SARSA belongs to the on-policy group, meaning the strategy it learns and evaluate is the same.)

The Q-learner uses a so-called q-table that has the same number of rows as the number of states and the same number of columns as the activities that can be performed. A value in the table determines how good it is to select the activity corresponding to that column in the state corresponding to that row. The values in the q-table are updated according to the following formula:

$$Q(s_t, a_t) \leftarrow \; Q(s_t, a_t) + \; \alpha * \left[r_{t+1} + \gamma * \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \qquad (2.3.4)$$

where the parameters are the same as for SARSA.

The new q-value is determined based on the previous value and the maximum value of the immediate reward and the next status. The model should discover all state-action pairs to determine the optimal q-function and be careful not to get the model stuck in a local minimum.

The model updates its q-table based on interactions with the environment. The model can select the following activity in two ways: either it selects the activity that seems best based on the information gathered so far, so it has the highest value in the q-table, or it randomly selects an activity to explore the environment. The first method is called exploiting, while the second is called exploring. It is important to explore the environment at the beginning of the learning phase to find the best possible solution.

The rate of exploitation and exploration is most often determined using the ε-greedy strategy. In this case, the parameter *ε* (its value is between 0 and 1) determines in which part of the cases our model chooses a random activity, while with a probability of 1-ε it chooses the action that seems best. The value of *ε* is usually set to around 1, thus helping the model to explore its environment, and as time and the learning process progresses, its value decreases exponentially to a predetermined minimum level, thus achieving that the model maximizes its reward.

3 Main algorithms of deep reinforcement learning

3.1 Deep Q-learning

The Deep Q-Network (DQN) [13] is a special version of the Q-learner that is useful when many states and activities are observed in the environment. In this case, the size of the Q table required for the Q learner would be too large (e.g., it would not fit in the memory of computer). A solution to this problem is provided by the DQN, which uses a neural network (Q-network) instead of the Q table to determine Q values. The input of the network is the state of the environment, and the output gives the Q values for the possible activities, as shown in the following figure.

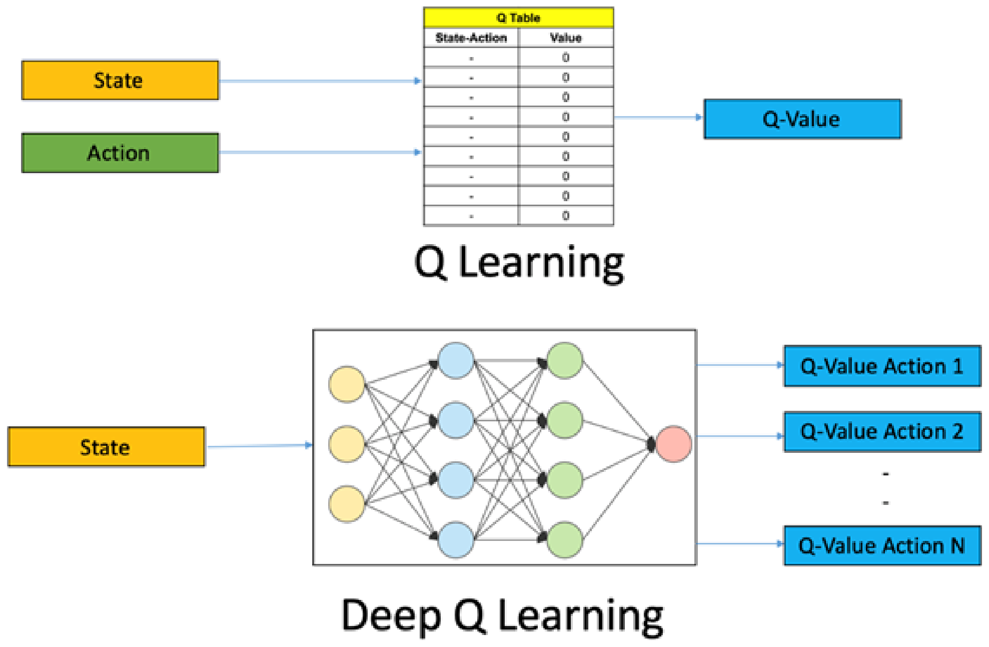


Figure 3.1.1. Difference between Q-learning and DQN [15]

When using DQN, the formula with the Q learner is simplified to the next:

$$Q(s_t, a_t) \leftarrow r_{t+1} + \gamma * \max_a Q(s_{t+1}, a) \tag{3.1.1}$$

In the case of DQN, the challenge is to map an ever-changing input to an ever-changing output. To solve this, there are two ways to keep the target variable temporarily in place. In the target network method, we use two neural networks for training: a target network and a main network. The Q-values predicted by the target network are used to train the main network. The parameters of the target network are not trained, however, they are periodically synchronized with the parameters of the main network.

Another solution to the same problem is experience replay [12]. Here, the response of the environment (status, activity, reward, next state quadruples) is not used immediately to train the network but is stored in a buffer. After a certain number of iterations, we randomly select elements from this buffer and train our network with these values. Random selection is important because samples that are usually similar to two adjacent or close states of the environment are usually characterized by a high correlation between two consecutive actions, which can lead to overfitting. By selecting the elements from the buffer by uniform sampling, we can avoid the above-mentioned problems.

3.2 Actor-critic methods

One of the biggest drawbacks of methods based on traditional policy optimization using Monte Carlo simulations is the high variance inherent in the process. Actor-critic methods [7] try to solve this problem, which has in common that they approximate optimal Q-values and actions at the same time by parameterizing neural networks. The critic component of the model is responsible for learning Q-values, which are used to systematically update the policy space parameters of the actor component. The task of the critic network is also to estimate the expected value of the reward that can be achieved by performing each state-action pair.

As described in the previous paragraph, the main advantage of the method is that the coordinated work of the two components allows: due to the fact that the critic continuously provides the actor with performance indicators of the execution of each action, the actor is able to define the actions to be performed without the need for the Q function.

Due to the nature of the method, the critic component is typically a function that approximates a state-value pair. It evaluates the function for each input pair and then decides whether the new values have improved or deteriorated the model:

$$A(s_t, a_t) = \mathrm{r}_{\mathrm{t+1}} + \gamma \mathrm{V}(\mathrm{s}_{\mathrm{t+1}}) - \mathrm{V}(\mathrm{s}_{\mathrm{t}}) \qquad (3.2.1)$$

where V is the current value function. Formula 3.2.1 is also called a TD (temporal difference) error, which can be used to evaluate the execution of an activity *t* in the state s_t. If the TD error is positive the activity a_t is more frequent if it is negative the less frequent execution of a_t is recommended in subsequent decisions.

3.2.1 Deep Deterministic Policy Gradient method (DDPG)

One of the most advanced versions of actor-critic type models is the DDPG method using a total of four networks [10]. The method is based on the Advantage Actor-Critic presented earlier but has also been extended to include a target network, which are periodically synchronized copies of the original networks. The components of the DDPG model are thus $\theta^{Q} and\, \theta^{Q'}$ (Q-value and target Q-value functions) and $\theta^{\mu}\ and\ \theta^{\mu'}$ (policy and target policy functions). The use of the two extra networks allows for more stable convergent learning, as periodic saving of parameters provides a solution to the divergence that occurs during the optimization process.

In addition to updating actor-critic networks, the DDPG uses three main techniques: experience replay, effective modification of target networks, and appropriate sampling of activity and parameter space (exploration). The first has been discussed before, but the other techniques are worth explaining briefly. The actor and critic networks are updated in a similar way to the Q-learning method, except that in the DDPG procedure, the Q-values of the following states are set by the $\theta^{\mu'}$ and $\theta^{Q'}$networks:

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'} \right) \mid \theta^{Q'} \right) \tag{3.2.2}$$

After determining the new Q-value, using the original Q-value calculated by the θ^{Q} network, we can obtain the cumulative error, i.e., the loss:

$$\text{Loss} = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i \mid \theta^{Q}) \right)^2 \tag{3.2.3}$$

Target networks are updated more easily by copying their parameters at a τ update rate:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \end{aligned} \tag{3.2.4}$$

In the case of reinforcement learning, balancing exploitation and exploration is an important consideration when sampling the activity that follows from the event space. In the case of sampling from a continuous event space with random selection, we add noise to the activity itself using the Ornstein-Uhlenbeck process [22]:

$$\mu'(s_t) = \mu\left(s_t \mid \theta_t^{\mu} \right) + N \tag{3.2.5}$$

3.2.2 A2C method

A2C (Advantage Actor Critic) [7] is a typical actor-critic algorithm, which is introduced to improve the policy gradient updates. A2C utilizes an advantage function to reduce the variance of the policy gradient. Instead of only estimating the value function, the critic network estimates the advantage function. Thus, the evaluation of an action not only depends on how good the action is, but also considers how much better it can be. So that it reduces the high variance of the policy networks and makes the model more robust.

A2C uses copies of the same agent working in parallel to update gradients with different data samples. Each agent works independently to interact with the same environment. After all the parallel agents finish calculating their gradients, A2C uses a coordinator to pass the average gradients over all the agents to a global network. So that the global network can update the actor and the critic network. The presence of a global network increases the diversity of training data.

Algorithm 3.2: Advantage Actor-Critic (A2C) [12]

```
Initialization of parameters: s (states), θ (policy function weights), w (Q-function weights)
for t = 1 ... T:
    $\boldsymbol{r_t} \sim \boldsymbol{R}(\boldsymbol{s}, \boldsymbol{a})$ reward and $\boldsymbol{s'} \sim \boldsymbol{P}(\boldsymbol{s'} \mid \boldsymbol{s}, \boldsymbol{a})$ $sampling\ of\ next\ state$
    $\boldsymbol{a' \sim \pi_\theta}(\boldsymbol{a'} \mid \boldsymbol{s'})$ $sampling\ of\ next\ action$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{a_\theta Q_w}(\boldsymbol{s}, \boldsymbol{a})\boldsymbol{\nabla_\theta \log \pi_\theta}(\boldsymbol{a} \mid \boldsymbol{s})$ policy weight update
    $\boldsymbol{\delta_t} = \boldsymbol{r_t} + \boldsymbol{\gamma Q_w}(\boldsymbol{s'}, \boldsymbol{a'}) - \boldsymbol{Q_w}(\boldsymbol{s}, \boldsymbol{a})$ TD error calculation for correction
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{a_w \delta_t \nabla_w Q_w}(\boldsymbol{s}, \boldsymbol{a})$ $\boldsymbol{Q - function\ update\ with\ respect\ to\ the\ TD\ error}$
    update: $\boldsymbol{a} \leftarrow \mathbf{a}'$ és $\mathbf{s} \leftarrow \mathbf{s}'$
end for
```

The synchronized gradient update is more cost-effective, faster and works better with large batch sizes. A2C is a great model for stock trading because of its stability. The objective function for A2C is:

$$\nabla J_\theta(\theta) = E\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t) \cdot A(s_t, a_t)\right] \tag{3.2.6}$$

where $\pi_\theta(a_t \mid s_t)$ is the policy network, $A(s_t, a_t)$ is the advantage function written as:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{3.2.7}$$

3.2.3 PPO method

PPO (Proximal Policy Optimalization) [18] is introduced to control the policy gradient update and ensure that the new policy will not be too different from the older one. PPO tries to simplify the objective of Trust Region Policy Optimization (TRPO) by introducing a clipping term to the objective function. The probability ratio between old and new policies is expressed as:

$$\mathrm{r_t}(\theta) = \frac{\pi_\theta(\mathrm{a_t} \mid \mathrm{s_t})}{\pi_{\theta_{\mathrm{old}}}(\mathrm{a_t} \mid \mathrm{s_t})} \tag{3.2.8}$$

The clipped objective function of PPO is:

$$J^{clip}(\theta) = E_t[min(r_t(\theta)A(s_t, a_t), clip(r_t(\theta), \varepsilon)A'(s_t, a_t))] \tag{3.2.9}$$

where $r_t(\theta)A(s_t, a_t)$ is the normal policy gradient objective, and $A'(s_t, a_t)$ is the estimated advantage function. The function $clip(r_t(\theta), \varepsilon)$ clips the ratio $r_t(\theta)$ to be within $[1-\varepsilon, 1+\varepsilon]$.

The objective function of PPO takes the minimum of the clipped and normal objective. PPO discourages large policy change move outside of the clipped interval. Therefore, PPO improves the stability of the policy networks training by restricting the policy update at each training step. We select PPO for stock trading because it is stable, fast, and simpler to implement and tune.

State-of-the-art algorithms for deep reinforcement learning have been presented in previous chapters. In the following, we will discuss the practical application of these algorithms in a stock trading environment.

4 Market model and environment descriptors

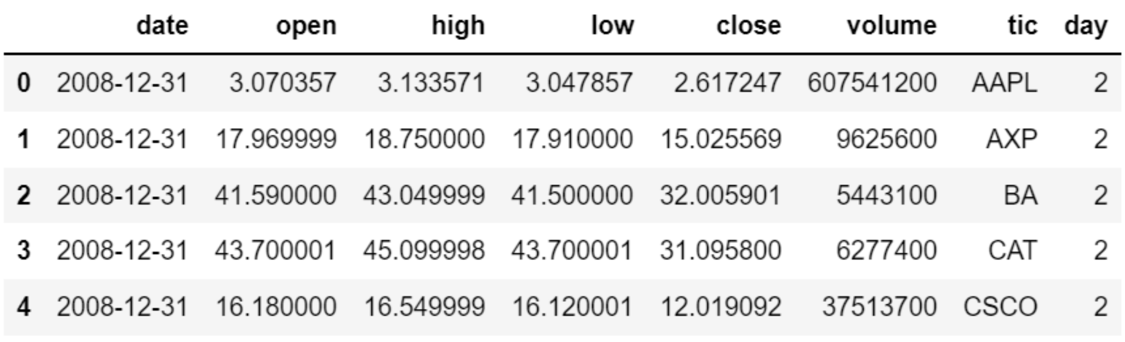
To model the stock trading task discussed below, an environment, more precisely the individual states of the environment, a set of multidimensional features had to be defined, through which the agent can learn to adapt to the behavior of the market [6] with the greatest possible efficiency.

4.1 Data description

Different model agents received the same set of money market data as input through Yahoo Finance. The dataset contains the following 7 features by default, downloaded in daily resolution: stock TIC ID, timestamp, daily opening price, daily closing price, daily highest price, daily minimum price, and finally the total daily volume traded.

For the main results of the research the DOW30 index were chosen [24], which at the beginning of the work were: AAPL, MSFT, JPM, V, RTX, PG, GS, NKE, DIS, AXP, HD, INTC, WMT, IBM, MRK, UNH, KO, CAT, TRV, JNJ, CVX, MCD, VZ, CSCO, XOM, BA, MMM, PFE, WBA, DD.

Data in this resolution was available in acceptable quality through the Yahoo Finance API until January 1, 2009, so the final dataset will cover approximately 12 and a half years by July 20, 2021. It is important to note that obtaining financial data in as much detail as possible will be an important step for further development.



	date	open	high	low	close	volume	tic	day
0	2008-12-31	3.070357	3.133571	3.047857	2.617247	607541200	AAPL	2
1	2008-12-31	17.969999	18.750000	17.910000	15.025569	9625600	AXP	2
2	2008-12-31	41.590000	43.049999	41.500000	32.005901	5443100	BA	2
3	2008-12-31	43.700001	45.099998	43.700001	31.095800	6277400	CAT	2
4	2008-12-31	16.180000	16.549999	16.120001	12.019092	37513700	CSCO	2

Figure 4.1.1 Features of the stocks

The improvements presented in the following chapters do not address the optimization of the time window of the training and test sets which in this research lasted from 2nd of April to 20th of July 2021.

4.2 Technical indicators

The state space, which is presented in more detail in the following chapters presenting the model, has been described with the help of widely used, popular technical indicators and utility quantities [19]. The key factors to characterize the stocks and the portfolio itself are:

- remaining balance: the amount of cash that can still be used in the trading process
- available number of shares: the number of shares already purchased from each share
- opening / closing share price
- adjusted closing share price: adjusted for end-of-day closing price

Technical indicators to characterize exchange rate dynamics for every stock in the portfolio:

- momentum indicators:
  - MACD, RSI 6 / 12 / 30, ADX, CCI 30
- volatility indicators:
  - VR, ATR, Bollinger-band
- traded volume indicators:
  - MFI, OBV, NVI, VPT, ADI

Further details about the technical indicators (14 factors) can be found in the *Appendix*.

4.3 Parameters of the trading process

Although the development of the trading process and strategy is carried out entirely by the agents with the goal of maximizing the reward function, to make realistic market trading, a few restrictions / facilitations had to be introduced [24] into the model:

- market liquidity, shortfall management: transactions take place with immediate effect and the transactions do not significantly affect the price of the asset
- non-negative balance: the transactions of the agents cannot result in a negative stock balance:

$$b_{t+1} = b_t + (p_t^s)^\top k_t^s - (p_t^B)^\top b_t^B \geq 0 \tag{4.3.1}$$

- transaction costs: transaction costs are incurred after each transaction, which are also implemented by the model. Such costs could be conversion, enforcement or even SEC (Securities and Exchange Commission) fees. For the sake of simplicity, only a single, fixed fee of 1/1000 of the value of the traded asset has been introduced in the model (both for sales and purchases).

$$c_t = p^\top b_t \times 0{,}1\% \qquad (4.3.2)$$

In the second half of the research, an indicator measuring market turbulence [8] was introduced in addition to the previous constraints. This factor can warn to hold back or even suspend trading in market situations where the market is showing unusual, overly volatile, unstable signals. A concrete example of such periods is the 2008 crisis, market bubbles or even wars.

5 Split feature space ensemble method

Due to the stochastic nature of stock trading tasks, the present trading task is modeled as a Markov Decision Process (MDP) problem. The decision-making process includes tracking stock price changes, defining sales activities, and calculating rewards. Taking into account the subtasks listed above, each agent develops their final trading strategy, i.e., policy.

Overall, agents interact with the environment to arrive at a trading strategy that maximizes the reward function over time. The above-mentioned environment, which is built from the data presented in the previous chapter, was simulated using the freely available OpenAI Gym framework.

The following subsections present the key elements of the reinforcement learning model that are designed to maximize portfolio value and minimize risk factors that occur during trading [25].

5.1 Standard ensemble model

The model used in the first half of the research used a single, coherent feature space in which 3 different models were trained. The following subsections discuss the key details of this baseline approach, which provided the basis for further developments.

5.1.1 Main elements of the model

In this chapter, the most important components of the agents implementing the deep learning methods detailed in the theoretical introduction are presented.

To describe the state space, we used a *14 * M* dimensional vector, in our case *M*, i.e. the number of the different shares was 30. In the initial development contained all the factors mentioned in the previous chapter to determine a state s: $\mathrm{s}_{\mathrm{t}} = [b_t, p_t, h_t, technical\ indicators_t]$, where

- $b_t \in R^{+}$: available cash balance at current time step t
- $p_t \in R^{M+}$: adjusted close price of each stock (*M* stocks)
- $h_t \in N^{\mathrm{M}}$: shares owned of each stock (*M* stocks)
- $technical\ indicators_t$: the 14 factors used to model the dynamics of the market

To describe the action space, the vector *{-k... -1, 0, 1, 2... + k}* is used, where *k* is the number of shares that can be sold in one state.

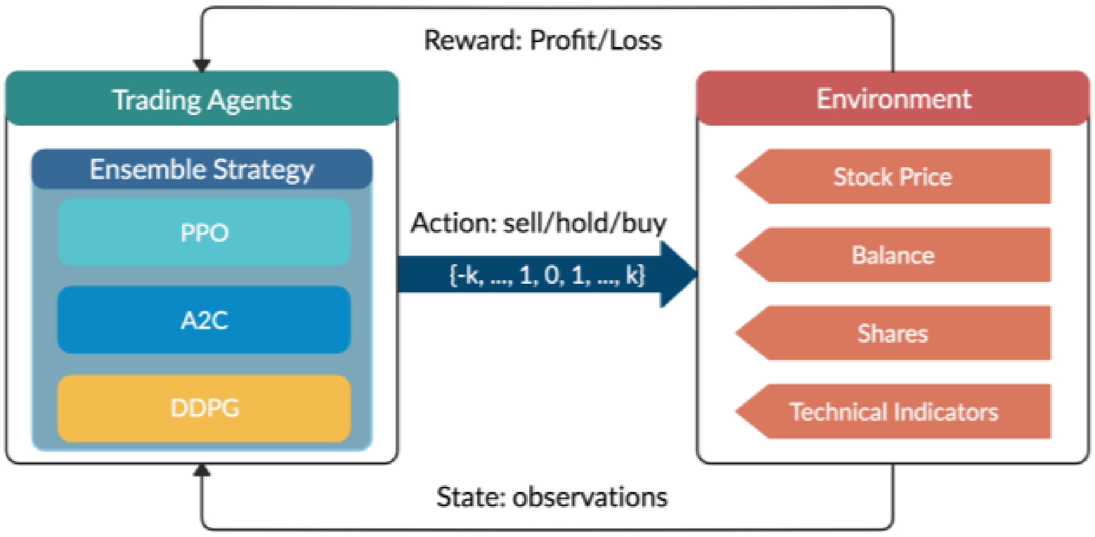


Figure 5.1.1 Trading process of an agent [24]

For the *M* shares used in the present research, the activity space is thus $(2k + 1)^M$ large and the reward function of the agents is the change in the value of the portfolio during the transition from s_t to s_{t+1}, which can be formalized as:

$$r(s_t, a_t, s_{t+1}) = \gamma\,[(b_{t+1} + p_{t+1}^T h_{t+1}) - (b_t + p_t^T h_t) - c_t] \tag{5.1.1}$$

Change in the number of shares held, where b_t^{sold} is the amount of shares the model sold and b_t^{bought} means the amount of shares the model bought:

$$h_{t+1} = h_t - b_t^{sold} + b_t^{bought} \tag{5.1.2}$$

The final decision of the models results in a sequence of actions for states that tells the probability distribution of each action in each state. The expected value of each state with transitions for the reward function can be expressed by the value of *Q*, i.e. how much expected portfolio value increase can be achieved in each state by performing each action. In each state, 3 possible events can be performed: sell, hold (no sell, nor buy) and buy as can be seen in Figure 5.1.2.

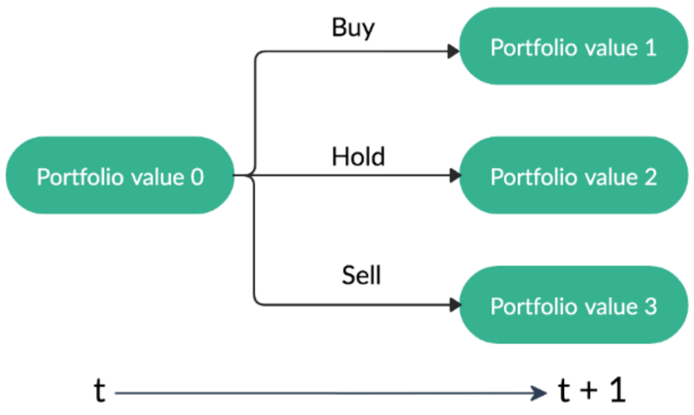


Figure 5.1.2 Effect of actions on the portfolio value [24]

Let d be a given stock (identity of the stock) in the portfolio, where $d = (1, \dots, M)$:

- selling $\boldsymbol{k}[d] \in [1, \boldsymbol{h}[d]]$ shares results in $\boldsymbol{h}_{t+1}[d] = \boldsymbol{h}_t[d] - \boldsymbol{k}[d]$ amount, where $\mathrm{k}[d] \in \mathrm{Z}^+$ and $d = (1, \dots, M)$
- in case of holding $\boldsymbol{h}_{t+1}[d] = \boldsymbol{h}_t[d]$
- buying $\boldsymbol{k}[d]$ shares results in $\boldsymbol{h}_{t+1}[d] = \boldsymbol{h}_t[d] + \boldsymbol{k}[d]$ amount.

The value of the portfolio throughout the process is $\boldsymbol{p}^T\boldsymbol{h} + b$.

5.1.2 Operating principles

The main goal of the initial model [24] is to develop a more robust trading strategy than previous methods using machine learning models. To do this, it uses a distributed, standard ensemble technique [24] that takes turns using the currently best-performing model as follows:

1. The entire trading period is divided into training and validation intervals without overlapping.
2. A PPO, DDPG, and A2C agents are initialized and learn in parallel using the same state descriptors.
3. We start training the models in an initial period, then evaluate them in a validation time window of 3 months after the training period and select the currently best performer for action prediction for the next quarter. Selection is based on the Sharpe ratio achieved, where $\overline{r_p}$ is the expected value of return, r_f is the risk-free interest rate and σ_p is the volatility (variance) of the stocks in the portfolio.

$$\text{Sharpe -ratio} = \frac{\overline{r_p} - r_f}{\sigma_p} \tag{5.1.4}$$

4. The training period is extended by 3 months and then the loop is restarted.

5.2 Splitting the feature groups

In the standard ensemble model, a set of features put into a single group was responsible for describing the state spaces of the parallel acting agents [24]. Factors, i.e., different aspects of each technical indicator, covered the dynamics of the market.

The basic idea of the first development idea was also given by the non-separate use of features: it would be worthwhile to parallelize agents [5] not only on a model basis, but also by feature function [4], i.e. to apply ensemble techniques [23] at the level of factor groups.

For this solution, the first step was to define the functional groups according to which the technical indicators could be grouped. In addition to the auxiliary factors not used in the model (such as the day of the week), three main groups were identified: momentum-type indicators, volatility indicators, and indicators that provide information on the volume traded in the market. The groups do not overlap, they form disjoint sets. The factors are also discussed in more detail in the *Technical indicators* section.

Dividing groups has several roles. One is to see the effectiveness of the initial ensemble development and extend the method to the feature space, thus giving the model a chance to find not only the best algorithm but also the most suitable feature set in a given trading period. With this extended method, we can get new information from each agent instance that can help us understand how each model can interact effectively with certain groups of factors, i.e., we can gain indirect knowledge about which methods are worth using in different market periods and what are the dynamic indicators that carry most of the useful information.

Separating groups and using them based on different goodness metrics can not only result in an even more robust model, but also alleviate the problem of transparency and algorithmic transparency, which is often treated as a critical point in the financial world. Although the application of deep learning methods continues to strengthen the black-box nature of developments, the ensemble decision presented here can be clearly traced back to all trading periods. Examining these decisions, we can clearly see in each period of the market what were the strong dynamic descriptors (momentum, volatility, traded volume) based on which the model decided to take or give up the position.

The selection of technical indicators for each group was mainly based on statistical methods. Function groups were constructed based on the most used factors in the literature, and after normalization of the data, indicators that explained each other too much were omitted based on Pearson correlation. It was mentioned earlier that not only did we strive for the greatest possible independence within the groups, but there was no overlap between the groups at all in order to increase robustness.

The initial feature space contained the factors presented in the chapter on technical indicators, the correlation heatmap of which is shown in the following figure:

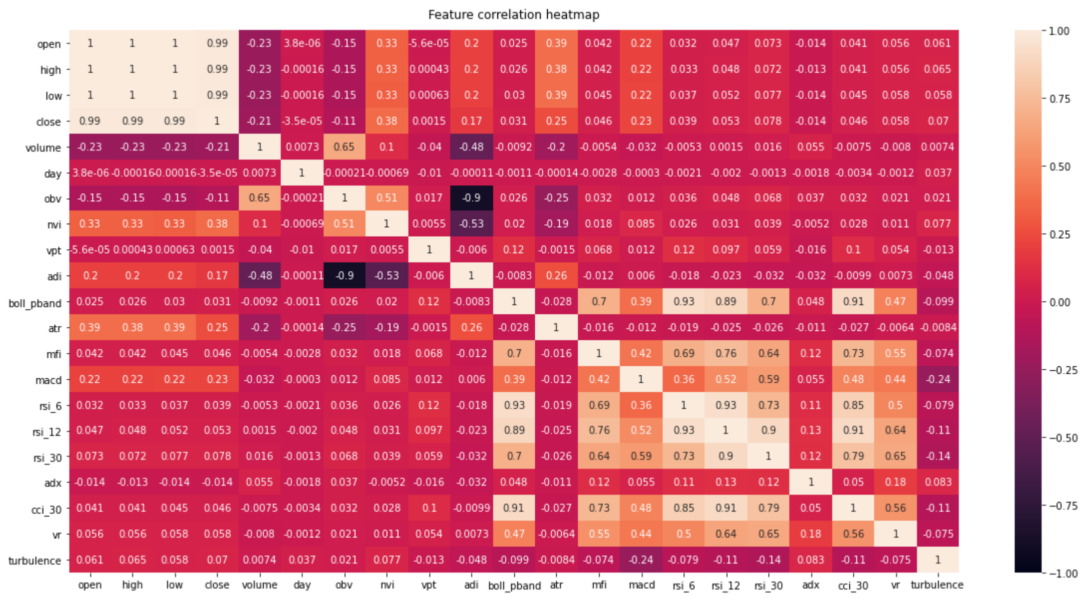


Figure 5.2.1 Correlation heatmap of the whole feature space

It can be seen that a significant number of factors explain each other, their positive correlation is high. To avoid overfitting and to achieve robust predictions, each function group was narrowed down using backward feature elimination. As a first step, a constant value was defined to define a strong positive correlation, the lower limit of which became 0.3. As a next step, the technical indicators that correlated strongly positively with most other indicators within a function group were selected. If two indicators correlated strongly positively with other indicators similar times, then the indicator which was referred more in the literature and used in practice stayed in place. In general, the goal was to keep as many factors as possible during feature elimination.

The following figures show the technical indicators of the original and reduced factor groups and their correlations. It can be noticed in the last two heatmaps containing momentum indicators that there are strongly correlated RSI indicators even after reduction. These factors have been intentionally left out, as they model the same market dynamics phenomenon in different time windows, the autocorrelation of which is inevitable, and these factors were too important to leave.

1. Volume indicators:

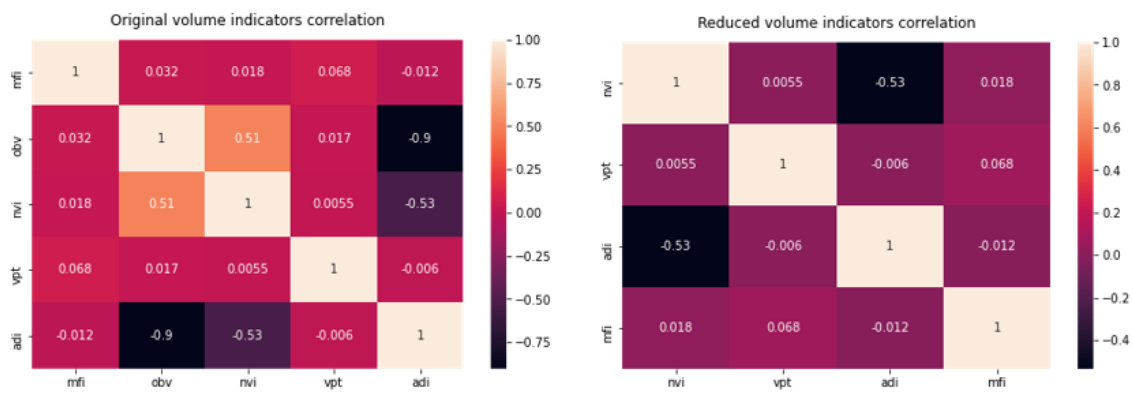


Figure 5.2.2 Volume indicators' correlation heatmaps before and after reduction

2. Volatility indicators:

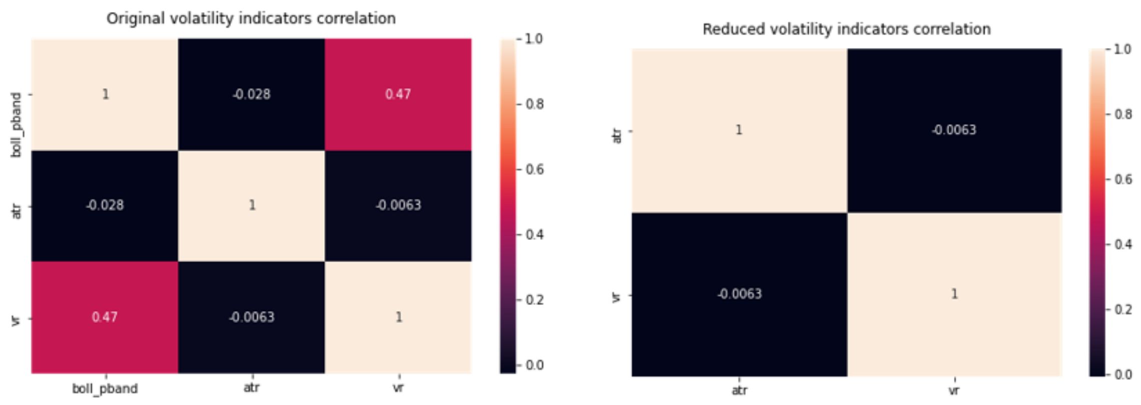


Figure 5.2.3 Volatility indicators' correlation heatmaps before and after reduction

3. Momentum indicators:

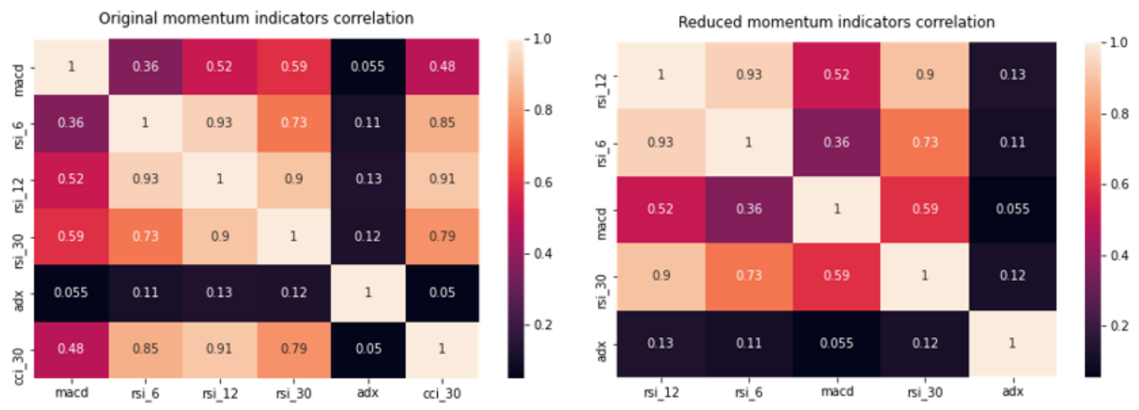


Figure 5.2.4 Momentum indicators' correlation heatmaps before and after reduction

5.3 Extended risk aware ensemble model space

The life cycle of the ensemble model follows the main points, similar to the initial development. At the initial time, let p_0 be the value of the shares and b_0 be the value of the capital invested. Denote by h the number of shares already purchased and in the active portfolio, which is initially 0. The initial value of the $Q_\pi(s_t, a_t)$ function is also 0, and the $\pi(s)$ policy defined in the state space is uniformly distributed.

The value of the $Q_\pi(s_t, a_t)$ function is updated during each interaction with the trading environment, as a function of the current state activity pair. The optimal trading strategy is obtained using the Bellman equation presented in the theoretical introduction, such that the expected reward of taking action at state s_t is the expectation of the summation of the direct reward $r(s_t, a_t, s_{t+1})$ and the future reward in the next state s_{t+1}. Furthermore, a discount factor for the future rewards is also introduced [24]:

$$Q_\pi(s_t, a_t) = E_{s_{t+1}}\left[r(s_t, a_t, s_{t+1}) + \gamma E_{a_{t+1}\sim\pi(s_{t+1})}[Q_\pi(s_{t+1}a_{t+1})]\right] \tag{5.3.1}$$

In the improved method, a 3x3 model was used instead of the initial 3 models. Feature groups covering 3 types of market dynamics were assigned 3 different models that were trained in parallel spaces. The higher an agent's Sharpe ratio, the better its returns have been relative to the amount of investment risk it has taken. Therefore, we pick the trading agent that can maximize the returns adjusted to the increasing risk.

At the end of each training period, out of the 9 models (initially), the policy and status space of the portfolio producing the best Sharpe ratio was selected for the next prediction (trading) period. The value and structure of the portfolio, i.e. the current position defined by the state space descriptors, was synchronized with the other models at the end of each cycle.

The technique of choosing from 9 models has had a number of other results in addition to increased profit and reduced risk. One of these was the demonstration of the sensitivity of the models to different market and trading situations. In addition to performance indicators, this sub-result is also discussed in the Measurements and results section.

One of the sub-tasks of the trading process is not only to increase the value of the portfolio, but also to minimize the drawdown, i.e., to reduce the risk. To solve this unavoidable problem, the following idea and methods have been used.

The previously mentioned market turbulence index was included in the more advanced model not only as a constant constraint, but as a kind of circuit breaker:

$$\text{turbulence}_{\,t} = (y_t - \mu)\Sigma^{-1} \cdot (y_t - \mu)^{\mathrm{T}} \in R \tag{5.3.2}$$

where y_t denotes the stock returns for current period t, μ denotes the average of historical returns, and Σ denotes the covariance of historical returns.

In highly volatile circumstances, when a predefined threshold is exceeded, the reward function changes to a target function that seeks to minimize [24] portfolio value loss by eliminating positions:

$$r_{sell} = (p_{t+1} - p_t)^T k_t \tag{5.3.3}$$

Aligning with the volatility indicator, the simple reward function that maximizes portfolio value and the ensemble decision responsible for selecting the model that provides predictions for the next trading period has been transformed into a weighted complex function that adapts to the most important trading subtasks. The decision between the models was made at the end of each 3-month period based on the following:

1. The Sharpe-ratio and maximum drawdown achieved by all 9 models were calculated.
2. Sharpe and drawdown values were scaled separately into the interval [0,1].
3. The scaled two values were aggregated per model (*Eq. 5.3.4*), resulting in a standard score.
4. The turbulence index was scaled to the [0,1] interval. This value, calculated per period and compared to the 90th percentile of the historical average, results in a weight factor in the final score. The two factors of the decision function, the Sharpe and drawdown ratios, separately control mainly the volatility-adjusted increase in the value of the portfolio and the loss in the value of the portfolio. With the introduction of the weighting factor, minimizing drawdown plays a greater role in turbulent periods, while increasing the value of the portfolio is more important in calmer, more stable periods.

The function of the final decision with weight factors adaptive to market turbulence can be written as follows:

$$P_{\text{final}} = \begin{bmatrix}\alpha \\ \beta\end{bmatrix} * \ [Sharpe_{scaled} \quad drawdown_{scaled}], \tag{5.3.4}$$

where $\begin{bmatrix}\alpha \\ \beta\end{bmatrix}$ in the ith trading period equals to $\begin{bmatrix} turbulence_i \,/\, turbulence_{mean} \\ 1 - \ (turbulence_i \,/\, turbulence_{mean}) \end{bmatrix}$.

Another useful by-product of the combined use of the above metrics is that even if the weighting factor of the drawdown member were 0 in some extreme case, risk management does not disappear from the model either, as volatility is included in the Sharpe-ratio.

6 Experiments and results

Five metrics are used to evaluate our results:

- cumulative return: $\frac{final\ portfolio\ value - initial\ portfolio\ value}{initial\ portfolio\ value}$
- annual return: $\sqrt[n]{\prod_1^n 1 + R_n} - 1$, where R_n is the cumulative return of year n
- annual volatility: $\sqrt{252} \times \sqrt{variance\ of\ annual\ return}$, where 252 is the number of trading days in a year
- maximum drawdown during the trading interval

Cumulative return reflects returns at the end of trading stage. Annualized return is the return of the portfolio at the end of each year. Annualized volatility and maximum drawdown measure the robustness of a model. The Sharpe ratio is a widely used metric that combines the return and risk together.

To evaluate the results, two main measurement scenarios were defined: in the first, a single set of features, while in the second, 3 shared feature sets were responsible for describing the states. During the measurements, the average efficiency and its standard deviation were calculated based on the results of 5 runs in each case.

- Sharpe-ratio:

feature / model	all features in one group	features in 3 separate groups
only PPO	1.638 ± 0.190578	1.644 ± 0.280054
only DDPG	**1.752 ± 0.221405**	1.814 ± 0.278083
only A2C	1.494 ± 0.291513	1.676 ± 0.106442
standard ensemble	1.726 ± 0.11371	-
3 model ensemble	-	2.194 ± 0.328831
9 model ensemble	-	**2.194 ± 0.197180**

Table 6.1.1 Sharpe-ratios of the different models

As of *Table 6.1.1*, in the first scenario, where the features were not separated, they were added as a bulk input to the models, the DDPG model achieved the highest Sharpe-ratio. The standard ensemble model was really close to the DDPG result. From the viewpoint of consistent efficiency, the standard ensemble can be considered better, due to its lower standard deviation in the results.

In the second scenario, where the features were handled in three separate groups, the highest Sharpe-ratio was achieved by the 9-model ensemble approach. However, scoring the same average as the 3-model version derived from 5 measurements the standard deviation of 9-model approach is lower, which is a sign of a more stable performance. Compared to the standard ensemble, which could be considered as a baseline, the improved development scored 0.45 more in average, which is quite a large improvement.

- Maximum drawdown:

feature / model	all features in one group	features in 3 separate groups
only PPO	**-0.08772 ± 0.010475**	-0.08886 ± 0.008784
only DDPG	-0.10472 ± 0.010284	-0.0906 ± 0.014662
only A2C	-0.09796 ± 0.016146	-0.09978 ± 0.013307
standard ensemble	-0.09704 ± 0.007469	-
3 model ensemble	-	**-0.08416 ± 0.017468**
9 model ensemble	-	-0.0875 ± 0.00882

Table 6.1.2 Maximum drawdown of the different models

As of *Table 6.1.2*, in the first scenario the PPO model achieved the lowest drawdown during the whole trading interval.

In the second scenario, where the features were handled in three separate groups, the lowest drawdown was achieved by the 3-model ensemble approach. The split feature PPO approach and the 9-model version was also really close to this result.

- Annual return:

feature / model	all features in one group	features in 3 separate groups
only PPO	0.24394 ± 0.029312	0.2548 ± 0.060038
only DDPG	0.2986 ± 0.037177	0.3022 ± 0.050149
only A2C	0.26072 ± 0.070203	0.28794 ± 0.023736
standard ensemble	**0.30504 ± 0.027429**	-
3 model ensemble	-	0.38524 ± 0.0453472
9 model ensemble	-	**0.40582 ± 0.030984**

Table 6.1.3 Annual return of the different models

As of *Table 6.1.3*, in the first scenario the standard ensemble model achieved the highest annual return.

In the second scenario, where the features were handled in three separate groups, the highest annual return was achieved by the 9-model ensemble approach. Compared to the standard ensemble, which could be considered as a baseline, the improved development scored 10% more in average, which is a great improvement considering the one-year trading interval.

- Annual volatility:

feature / model	all features in one group	features in 3 separate groups
only PPO	**0.13932 ± 0.006226**	**0.14384 ± 0.012929**
only DDPG	0.1565 ± 0.009719	0.15228 ± 0.010612
only A2C	0.16212 ± 0.014332	0.15892 ± 0.009106
standard ensemble	0.16206 ± 0.004021	-
3 model ensemble	-	0.15624 ± 0.020884
9 model ensemble	-	0.16198 ± 0.008748

Table 6.1.4 Annual volatility of the different models

As of *Table 6.1.4*, in the first scenario the PPO model achieved the lowest annual volatility during the whole trading interval.

In the second scenario, where the features were handled in three separate groups, the lowest annual volatility was also achieved by the PPO approach.

Further experiments were conducted using only a single feature group per a single model. These results can help understanding the behavior of each model with different factors. The following tables (from 6.1.5. to 6.1.8.) were evaluated column wise to obtain the best scoring models per feature group.

- Sharpe-ratio:

feature / model	volatility	volume	momentum
A2C	1.490	1.629	1.288
DDPG	1.747	**1.741**	**1.562**
PPO	**1.929**	1.357	1.357

Table 6.1.5 Sharpe-ratio of the different models

- Maximum drawdown

feature / model	volatility	volume	momentum
A2C	**-0.075**	-0.115	-0.118
DDPG	-0.100	**-0.078**	-0.120
PPO	-0.097	-0.103	**-0.088**

Table 6.1.6 Maximum drawdown of the different models

- Annual volatility

feature / model	volatility	volume	momentum
A2C	0.1393	0.1769	0.1671
DDPG	0.1604	0.1439	0.1607
PPO	**0.1368**	**0.1430**	**0.1310**

Table 6.1.7 Annual volatility of the different models

- Annual return

feature / model	volatility	volume	momentum
A2C	0.217	**0.311**	0.222
DDPG	**0.305**	0.270	**0.267**
PPO	0.288	0.201	0.183

Table 6.1.8 Annual return of the different models

At annual return the best model-feature pair was A2C – volume with 0.311, which can be compared with the previous best models, the standard ensemble and the 9-model ensemble methods. The Table 6.1.9 presents that 9-model ensemble algorithm surpasses the others.

model	Annual return
standard ensemble	0.305
9 model ensemble	**0.406**
best model-feature pair: A2C - volume	0.311

Table 6.1.9 Annual return of the best models

The following results present the performance of the best performing 9-model ensemble algorithm run based on the use of the Pyfolio backtest package measured from

2020.07.06 to 2021.06.30. As pictured on *Figure 6.1.1* the best performance of the models could increase the annual value of the portfolio with more than 44%. The portfolio value change is plotted against the turbulence index, which makes it easier to observe the different behavior of the portfolio during changing market dynamics based on equation 5.3.4.

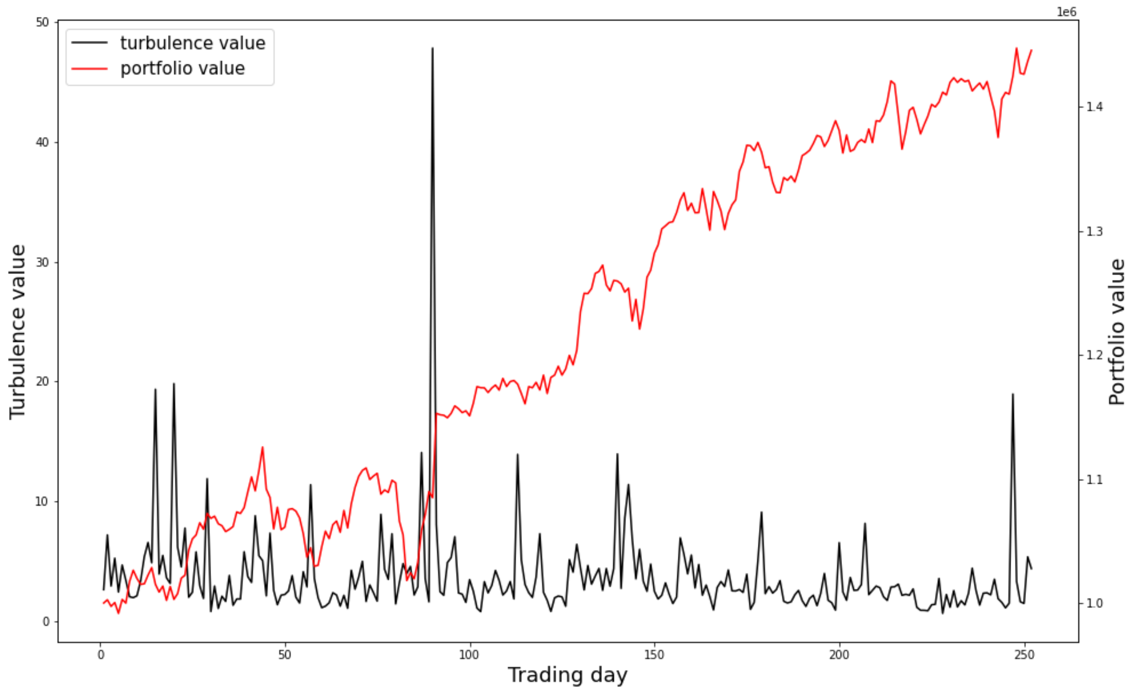


Figure 6.1.1 Performance of the portfolio with the turbulence index

Finally, the worst drawdown periods were the followings presented in *Table 6.1.10* and on *Figure 6.1.2* (drawdown periods and their respective loss values on the underwater plot), where net drawdown is the maximum loss in the portfolio, peak date is the last date where the performance was positive, valley date is the last date where the performance was negative, recovery date is the date when the portfolio value was the same again as at the peak date and the duration is the length of the drawdown period in trading days.

Worst drawdown period	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
1	9.56	2020-09-02	2020-10-28	2020-11-09	49
2	4.05	2021-01-14	2021-01-29	2021-02-04	16

Worst drawdown period	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
3	3.87	2021-05-07	2021-05-12	2021-06-02	19
4	3.38	2021-06-02	2021-06-18	2021-06-24	17
5	2.96	2021-03-17	2021-03-25	2021-04-09	18

Table 6.1.10 Drawdown periods during the whole trading interval

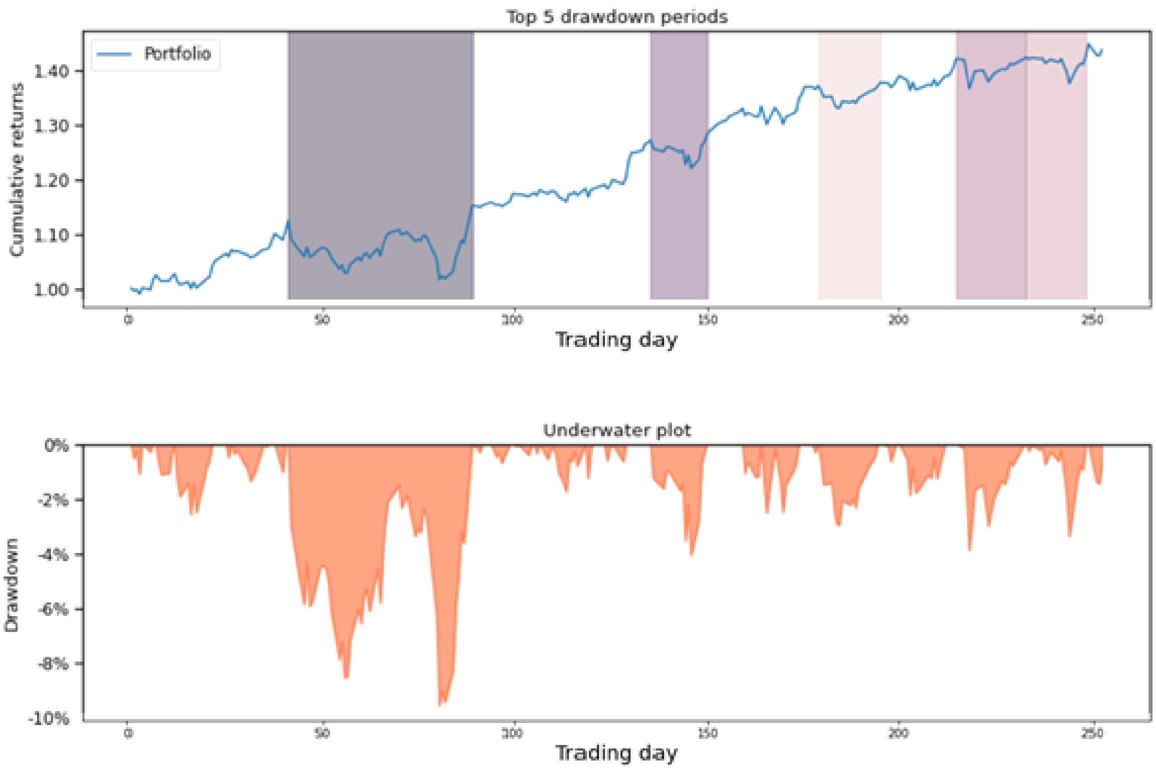


Figure 6.1.2 Drawdown periods and their coherent underwater plots

7 Summary

The dissertation details the theoretical background of algorithmic trading based on reinforcement learning methods, with a special focus on the possibilities of state-of-the-art deep reinforcement learning algorithms. After presenting the mathematical background of trading and deep reinforcement learning, I explored a new trading approach through a multi-share trading problem that uses ensemble methods to train multiple models simultaneously.

My research focused on assessing and further developing the performance of a standard ensemble agent architecture. This standard approach used technical indicators to describe market dynamics in a single bulk group and then selected a predictor for a subsequent trading period based on a rolling window ensemble method from 3 different algorithms. First, I performed individual performance measurements of each learner algorithms for this implementation, followed by the implementation of possible development ideas after setting up the performance profile.

In the first half of the research, an improved ensemble method was developed, which is able to increase the value of the portfolio even in highly turbulent market conditions while minimizing risk factors. The main development idea uses an extended ensemble method, one of the components of which is the split feature space. Dividing the feature space based on predefined function groups promotes the robustness of the model decision by always preferring the best algorithm and the group of factors that best fits it in the current market situation during the trading period.

In the second half of the research, the main goal was to effectively address risk factors. In addition to maximizing profits, a key task is to minimize portfolio impairment during volatile market situations. To solve this problem, an adaptive decision mechanism has been introduced that dynamically weights the reward function used in ensemble decision based on a turbulence index by varying the weights of the Sharpe-ratio and the maximum drawdown values.

In the final phase of the work, the performance of the new models was recorded by analyzing the different measurement scenarios and a number of financial metrics within them. The results show that the stock trading task aimed at the research project was achieved with outstanding efficiency even in turbulent market conditions.

During the implementation of the project, not only new results were achieved, but also a number of further issues were gathered. Of these, further research on reinforcement learning algorithms, such as the further development of multi-agent approaches (MARL, multi-agent RL) in terms of policy aggregation, may be a key topic for future developments. Another research topic could be the information theory analysis of the factors used in order to create a better and more informative feature space.

References

[1] Bao, W., & Liu, X. (2019). Multi-Agent Deep Reinforcement Learning for Liquidation Strategy Analysis. *ArXiv, abs/1906.11046*.

[2] Bellman, R. (1957). A Markovian Decision Process. Journal of Mathematics and Mechanics. 6 (5): 679–684. JSTOR 24900506.

[3] Bellman, R. (1957). Dynamic Programming. Dover. ISBN 0-486-42809-5.

[4] Dusparic I., Cahill V. (2009) Using Reinforcement Learning for Multi-policy Optimization in Decentralized Autonomic Systems – An Experimental Evaluation. In: González Nieto J., Reif W., Wang G., Indulska J. (eds) Autonomic and Trusted Computing. ATC 2009. Lecture Notes in Computer Science, vol 5586. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-02704-8_9

[5] Ghosh, S., Laguna, S., Lim, S.H., Wynter, L., & Poonawala, H.A. (2020). A Deep Ensemble Multi-Agent Reinforcement Learning Approach for Air Traffic Control. *ArXiv, abs/2004.01387*.

[6] Huang, C. (2018). Financial Trading as a Game: A Deep Reinforcement Learning Approach. *ArXiv, abs/1807.02787*.

[7] Konda, Vijay & Tsitsiklis, John. (2001). Actor-Critic Algorithms. Society for Industrial and Applied Mathematics. 42.

[8] Kritzman, Mark & Li, Yuanzhen. (2010). Skulls, Financial Turbulence, and Risk Management. Financial Analysts Journal. 66. 10.2469/faj.v66.n5.3.

[9] Lee, J. W., Park J., and Hong E., "A Multiagent Approach to Q-Learning for Daily Stock Trading," in *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 37, no. 6, pp. 864-877, Nov. 2007, doi: 10.1109/TSMCA.2007.904825.

[10] Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. CoRR, abs/1509.02971. https://arxiv.org/abs/1509.02971

[11] Liu, X.Y., Yang, H., Chen, Q., Zhang, R., Yang, L., Xiao, B., Wang, C.D. (2020). FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance, ArXiv abs/2011.09607

[12] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). PMLR.

[13] Mnih, V.; Kavukcuoglu, Koray; Silver, David; Rusu, Andrei A.; Veness, Joel; Bellemare, Marc G.; Graves, Alex; Riedmiller, Martin; Fidjeland, Andreas K. (2015). Human-level control through deep reinforcement learning. Nature. 518 (7540): 529–533.

[14] Mosavi, A.; Faghan, Y.; Ghamisi, P.; Duan, P.; Ardabili, S.F.; Salwana, E.; Band, S.S. Comprehensive Review of Deep Reinforcement Learning Methods and Applications in Economics. *Mathematics* 2020, *8*, 1640. https://doi.org/10.3390/math8101640

[15] Nováček, O., Vaiciukevičius, D., Koch, M. (2020). Connect X with DQN and PBT. Jun 15, 2020. (medium.com)

[16] Rummery, G. A., Niranjan, M. (1994). Online Q-learning using connectionist systems, Technical Report CUED/F-INFENG/TR 166.

[17] Russel, S. & Norvig, P. (2005). Mesterséges intelligencia modern megközelítésben. 2. kiadás. Panem.

[18] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *ArXiv, abs/1707.06347*.

[19] Širůček, Martin & Šíma, Karel. (2016). Optimized Indicators of Technical Analysis on the New York Stock Exchange. Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis. 64. 2123-2131. 10.11118/actaun201664062123.

[20] Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In NIPs (Vol. 99, pp. 1057-1063).

[21] Sutton, R., Barto, A. (1998). Reinforcement Learning. MIT Press. ISBN 978-0-585-02445-5. Archived from the original on 2017-03-30.

[22] Uhlenbeck, George E and Ornstein, Leonard S. (1930). On the theory of the brownian motion. Physical review, 36(5):823.

[23] Wiering, Marco & Van Hasselt, Hado. (2008). Ensemble Algorithms in Reinforcement Learning. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society. 38. 930-6. 10.1109/TSMCB.2008.920231.

[24] Yang, Hongyang and Liu, Xiao-Yang and Zhong, Shan and Walid, Anwar, Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy (2020) http://dx.doi.org/10.2139/ssrn.3690996

[25] Zhang, Z., Zohren, S., & Roberts, S.J. (2019). Deep Reinforcement Learning for Trading. *ArXiv*. https://arxiv.org/abs/1911.10107

Appendix

Technical indicators to characterize exchange rate dynamics for every stock in the portfolio. All the indicator details are from www.investopedia.com:

- momentum indicators:
  - MACD: Moving average convergence divergence (MACD) is a trend-following momentum indicator that shows the relationship between two moving averages of a security's price. The MACD is calculated by subtracting the 26-period exponential moving average (EMA) from the 12-period EMA.
  - RSI 6 / 12 / 30: The relative strength index (RSI) is a momentum indicator used in technical analysis that measures the magnitude of recent (6/12/60 days) price changes to evaluate overbought or oversold conditions in the price of a stock or other asset. The RSI will rise as the number and size of positive closes increase, and it will fall as the number and size of losses increase.
  - ADX: ADX is used to quantify trend strength. ADX calculations are based on a moving average of price range expansion over a given period of time. The default setting is 14 bars, although other time periods can be used.
  - CCI 30: The CCI compares the current price to an average price over a period of time. The indicator fluctuates above or below zero, moving into positive or negative territory. While most values, approximately 75%, fall between -100 and +100, about 25% of the values fall outside this range, indicating a lot of weakness or strength in the price movement.
- volatility indicators:
  - VR: The volatility ratio is a technical measure used to identify price patterns and breakouts. In technical analysis, it uses true range to gain an understanding of how a security's price is moving on the current day in comparison to its past volatility.
  - ATR: The ATR may be used by market technicians to enter and exit trades, and is a useful tool to add to a trading system. It was created to allow traders to more accurately measure the daily volatility of an asset by

using simple calculations. The indicator does not indicate the price direction; rather it is used primarily to measure volatility caused by gaps and limit up or down moves.

- Bollinger-band: A Bollinger Band® is a technical analysis tool defined by a set of trendlines plotted two standard deviations (positively and negatively) away from a simple moving average (SMA) of a security's price, but which can be adjusted to user preferences.

- traded volume indicators:
  - MFI: The Money Flow Index (MFI) is a technical oscillator that uses price and volume data for identifying overbought or oversold signals in an asset. It can also be used to spot divergences which warn of a trend change in price.
  - OBV: On-balance volume provides a running total of an asset's trading volume and indicates whether this volume is flowing in or out of a given security or currency pair. The OBV is a cumulative total of volume (positive and negative).
  - NVI: The negative volume index (NVI) is a technical indication line that integrates volume and price to graphically show how price movements are affected by down volume days.
  - VPT: The volume price trend (VPT) indicator helps determine a security's price direction and strength of price change. The indicator consists of a cumulative volume line that adds or subtracts a multiple of the percentage change in a share price's trend and current volume, depending upon the security's upward or downward movements.
  - ADI: The accumulation distribution indicator (ADI) is a momentum indicator that traders use to predict reversals in a trend by identifying tops and bottoms.