BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

REINFORCEMENT LEARNING FOR AUTOMATED STOCK PORTFOLIO ALLOCATION

VYUŽITÍ ZPĚTNOVAZEBNÉHO UČENÍ PRO AUTOMATICKOU ALOKACI AKCIOVÉHO PORTFOLIA

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **ZDENĚK LAPEŠ**
AUTOR PRÁCE

SUPERVISOR **doc. RNDr. MILAN ČEŠKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

148202

Institut:	Department of Intelligent Systems (UITS)
Student:	**Lapeš Zdeněk**
Programme:	Information Technology
Specialization:	Information Technology
Title:	**Reinforcement Learning for Automated Stock Portfolio Allocation**
Category:	Artificial Intelligence
Academic year:	2022/23

Assignment:

1. Study the state-of-the-art methods for automated stock portfolio allocation. Focus on the methods based on reinforcement learning and planning in Markov Decision Processes.
2. Experimentally evaluate selected open access tools for automated portfolio allocation including e.g. FinRL-Meta and identify their weak points.
3. Propose and implement improvements of a selected method/tool allowing to mitigate theseweak points.
4. Using suitable benchmarks and datasets, perform a detailed experimental evaluation of the implemented improvements with the focus on the portfolio allocation returns.

Literature:
Rao A., Jelvis T., Foundations of Reinforcement Learning with Applications in Finance. 1st Edition, Taylor & Francis 2022
* Li, Xinyi and Li, Yinchuan and Zhan, Yuancheng and Liu, Xiao-Yang, Optimistic Bull or Pessimistic Bear: Adaptive Deep Reinforcement Learning for Stock Portfolio Allocation, In ICML 2019.
* Liu X.-Y. Rui J. Gao J. aj.: FinRL-Meta: A Universe of Near-Real Market Environments for Data-Driven Deep Reinforcement Learning in Quantitative Finance. Workshop on Data Centric AI 35th Conference on Neural Information Processing Systems at NeurIPS 2021.
* Mao Guan and Xiao-Yang Liu. 2021. Explainable Deep Reinforcement Learning for Portfolio Management: An Empirical Approach. In ICAIF 2021.

Requirements for the semestral defence:
Items 1, 2, and partially 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Češka Milan, doc. RNDr., Ph.D.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	10.5.2023
Approval date:	3.11.2022

Abstract

Portfolio allocation is about selecting a set of assets to invest in. The goal is to maximize the expected return for a given level of risk for an investing horizon selected by us. In former times, this the process is usually done manually by an expert investor. Nowadays, there are many portfolio allocation methods that are not successful in the real world or can be improved and the potential of current technologies is not fully explored in the financial market. To solve the problem of portfolio allocation, I propose a methods based on reinforcement learning. The agent will learn the exact strategies of selecting assets and their weight for the portfolio based on which the human expert would select it, like fundamental analysis and the health of the company. The thesis deals with the problem of portfolio allocation using reinforcement learning, which helps in selecting the best assets and their importance for the portfolio.

Abstrakt

Alokace portfolia se zaměřuje na výběr souboru aktiv, do kterých investujete. Cílem je maximalizovat očekávaný výnos pro danou míru rizika pro námi zvolený investiční horizont. V dřívějších dobách tohle proces je obvykle prováděn ručně zkušeným investorem. V dnešní době existuje mnoho metod alokace portfolia, které nejsou úspěšné v reálném světě nebo je lze zlepšit a potenciál současných technologií není na finančním trhu plně prozkoumán. Pro řešení problému alokace portfolia navrhuji metody založené na posilovacím učení. Agent se naučí přesné strategie výběru aktiv a jejich váhu pro portfolio, na základě kterého by je lidský expert vybíral, jako je fundamentální analýza a zdraví společnosti. Diplomová práce se zabývá problémem alokace portfolia pomocí posilovacího učení, které pomáhá při výběru nejlepších aktiv a jejich důležitosti pro portfolio.

Keywords

artificial intelligence, AI, reinforcement learning, stock portfolio allocation, modern portfolio theory, Q-learning, neural networks, stock market

Klíčová slova

umělá inteligence, AI, posilované učení, alokace akciového portfolia, moderni teorie portfolia, Q-learning, neuronové sítě, akciový trh

Reference

LAPEŠ, Zdeněk. *Reinforcement Learning for Automated Stock Portfolio Allocation*. Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. RNDr. Milan Češka, Ph.D.

Reinforcement Learning for Automated Stock Portfolio Allocation

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Milan Češka, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Zdeněk Lapeš
April 11, 2023

Acknowledgements

I would like to thank my supervisor, Mr. Milan Češka, for his guidance and support during the preparation of this thesis. I would also like to thank my family and friends for their support.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The **Portfolio allocation problem** is to spread appropriate finite cash budget into financial instruments [3]. Under the financial instruments, we can imagine Stocks, Bonds, Mutual Funds, Commodities, Derivatives, Real Estate Investments Trusts (REITs), Exchange-Traded Funds (ETFs), and many more. The outcome should be to increase the initial capital over the course of a selected investing horizon, which can vary from a few days to decades. Portfolio management is essential for investors, particularly those who manage large sums of money such as institutional investors, pension funds, and wealthy individuals. While allocating assets instead of cash one must think about minimizing risk and maximizing the expected return on the investment. For that, the key considered strategy is **diversification**, which involves spreading investments across different instrument classes and markets in order to reduce the overall risk of the portfolio. A portfolio full of different assets can change over time due to market conditions, where the value of other assets may increase or decrease, which may cause the portfolio to become imbalanced. **Rebalancing** ensures that the portfolio remains aligned with the investor's goals and risk tolerance.

Among other portfolio allocation strategies could be mentioned **Modern portfolio theory (MPT)** to optimally allocate assets in a portfolio [9]. MPT uses statistical tools to determine the efficient frontier, which is the set of optimal portfolios that offer the highest expected return for a given level of risk, or the lowest risk for a given level of expected return. [11] Another approach is **Mean-variance optimization**, which uses mathematical models to determine the optimal portfolio based on an investor's risk tolerance and expected returns [6].

These approaches are not too appropriate for portfolio management, because the stock market is stochastic, volatile, quickly changing, and uncertain environment. These strategies are not flexible enough to adapt to the changing environment like the stock market, because they assume the future will be similar to the past, which may not always be accurate.

So, the most recent state-of-the-art portfolio management strategies are based on machine learning techniques. **Reinforcement learning (RL)** is a type of machine learning that is well-suited for solving problems involving decision-making and control [7]. In the context of portfolio allocation, RL can be used to optimize the allocation of assets in a portfolio in order to maximize returns or minimize risk. RL algorithms can learn from historical data and adapt to changing market conditions, which can lead to more efficient

and profitable portfolio management. The benefits of RL have been used in many different fields, such as robotics, games, and finances.

In the last decade, RL has become popular, because of its ability to learn difficult tasks in a variety of domains without knowing the environment model [7]. RL has advantages, such as flexibility, adaptability, and utilization of various information like e.q. experience gained from the environment under certain conditions. The agent is trained under a certain policy in a particular environment, which is modeled using **Markov Decision Process (MDP)**. MDP is a mathematical framework for modeling sequential decision-making problems [4]. MDP can be used to model the fully observable environment, where the agent can observe the state of the environment. If the environment is not fully observable, then the agent can observe only a part of the state of the environment, which is called **partially observable Markov decision process (POMDP)** [7]. In finances, the environment is usually fully observable, because the agent can observe the state of the environment. MDP is composed of the following elements:

1. **State:** The state is the current situation of the environment.
2. **Action:** The action is the decision that the agent can take.
3. **Reward:** The reward is the feedback that the agent receives after taking an action.
4. **Transition:** The transition is the change of the state after taking an action.

In agent training we handle the following problems:

- **State space**
  The state space is a finite set of all possible configurations of the environment. In the context of portfolio allocation, the state space can be defined as the finite set of all possible instrument features (fundamental and technical analysis) and their weights in the portfolio.
- **Action space**
  Action space should be designed so that the agent weights the assets in the portfolio. Here the question is: Should be this asset in the portfolio and if yes, what is the weight of this asset in the portfolio? These decisions are crucial for the performance of the agent. It is really difficult to find the optimal policy for the portfolio allocation because the agent has to choose between multiple assets with various differences in information about the assets. Also, actions should be considered profitable and safe in the long term, which means that the agent usually has to make decisions based on long-term rewards or on the defined investment horizon.
- **Reward function**
  The reward should reflect the agent's performance in the environment. Is the current portfolio value increasing or decreasing after the agent takes actions proposed by the policy?

When the state space is too large, then is merely impossible to be explored with the limited computational resources **Deep Reinforcement Learning (DRL)** can be used. DRL is a subfield of Reinforcement Learning (RL) that combines the use of deep neural networks with RL algorithms. In traditional RL, the agent's policy and value functions are typically represented by simple, hand-designed features or a small number of parameters. In contrast, DRL uses deep neural networks to represent these functions, allowing

the agent to learn from high-dimensional and complex inputs. DRL algorithms are used to train agents to perform a wide range of tasks, such as playing video games, controlling robotic arms, and driving cars. There are several popular algorithms in DRL, such as: **Deep Q-Network (DQN)**, **Deep Deterministic Policy Gradient (DDPG)**, **Proximal Policy Optimization (PPO)**, **Soft Actor-Critic (SAC)**, and **Twin Delayed Deep Deterministic Policy Gradient (TD3)**.

1.2 Limitations

1. **Data availability:** DRL models require large amounts of historical data to train effectively, which may be difficult to obtain for certain assets or markets.
2. **Model Overfitting:** DRL models can easily overfit to the training data, leading to poor performance on unseen data.
3. **High computational cost:** DRL models can require significant computational resources to train agents.
4. **Risk management:** DRL models may not be able to effectively handle risk management, such as different market situations (Market sentiment, Bull and Bear markets).

1.3 Aim of the Thesis

We will evaluate the performance of portfolio allocation methods based on DRL and compare them to traditional portfolio optimization techniques (MPT, Mean-Variance). Our goal is to determine the potential of DRL for portfolio allocation and identify the limitations of DRL-based portfolio allocation methods for future research.

The thesis objectives are:

- **Experimental evaluation & Benchmarks**
  Compare existing portfolio allocation agents. Evaluate the performance of the RL agents by comparing them with the baseline portfolio management strategies, such as MPT, Mean-Variance Optimization, and indexes (DJI, Nasdaq-100).
- **Dataset**
  Create a suitable dataset for the portfolio allocation problem. Datasets will be focused on the company's financial data, such as fundamental and technical analysis data.
- **Reimplementation**
  Try to improve current agents (Portfolio Allocation agent from **FinRL** [2]) with new datasets, focusing on Data Engineering and different DRL algorithms.

The thesis will be implemented using the programming language Python3 and open-source libraries such as NumPy, Pandas, Stable Baselines3, and OpenAI Gym.

Chapter 2

Reinforcement Learning

The motivation for this chapter comes from the influential book on reinforcement learning by Richard Sutton and Andrew Barto and Foundations of Reinforcement Learning with Application in Finance by Ashwin Rao and Tikhon Jelvis [4, 7].

This chapter explore the application of Reinforcement Learning techniques in the context of portfolio allocation. In section 2.2 we will provide an introduction to Markov models, first explore *Markov Processes*, *Markov Reward Processes* and *Markov Decision Processes*, which are the fundamental building blocks of Reinforcement Learning. Afterward, we divide Reinfocement Learning into two main categories. In section 2.3, we will discuss *Model-based* Reinforcement Learning methods, and in section 2.6 we will discuss *Model-free* Reinforcement Learning methods in details with advantages and disadvantages of each approach. Let start with introduction to AI in general and its different types of learning:

Supervised Learning In supervised learning (SL), a model is trained on a labeled dataset, where the input data is paired with corresponding output labels. The model learns to make predictions based on the labeled examples, and the goal is to minimize the error between predicted outputs and actual labels. Common applications of supervised learning include image classification, speech recognition, and sentiment analysis.

Semi-supervised Learning Semi-supervised learning (SSL) is a combination of supervised and unsupervised learning. It uses a small labeled dataset along with a large unlabeled dataset for training. The model leverages the limited labeled examples to learn patterns from the unlabeled data, and then makes predictions on unseen data. SSL is useful when obtaining labeled data is expensive or time-consuming. It is often used in scenarios where obtaining a large labeled dataset is challenging, such as in medical diagnosis or fraud detection.

Unsupervised Learning In unsupervised learning (UL), the model learns from unlabeled data without any predefined output labels. The goal is to find underlying patterns, structures, or relationships within the data. Common unsupervised learning tasks include clustering, dimensionality reduction, and anomaly detection. UL is used in scenarios where labeled data is scarce or not available, and the model needs to discover patterns autonomously from the data.

The last type is Reinforcement Learning and this entire chapter will be devoted to it, let's dive into it in more detail.

2.1 Introduction

Reinforcement learning (RL) is an exciting field at the intersection of artificial intelligence (AI) and machine learning (ML) that deals with training agents to make optimal decisions in dynamic environments. RL is inspired by the way humans learn from experience, like `trial-and-error`, and an agent interacts with an environment the same way and receives feedback in the form of *rewards* (typically positive number, e.g.: 1) or *penalties* (typically negative number, e.g.: -1), and uses this feedback to learn and improve its decision-making abilities.

At the heart of RL lies the concept of an agent, which that takes actions in an environment to achieve specific goals. The environment is typically modeled as a Markov decision process (MDP), which is a mathematical framework that describes how an agent interacts with an environment in discrete time steps. An MDP is defined by a tuple (S, A, P, R), where S is the state space, A is the action space, P is the transition probability function, and $R \in \mathbb{R}$ is the reward.

The goal of an RL agent is to learn a policy, denoted by π, which is a mapping from states to actions that maximizes the cumulative reward G_t over time T. The agent uses this policy to select actions at each time step, and the environment responds with a new state and a reward. The agent then updates its policy based on the observed rewards and states, aiming to improve its decision-making abilities and achieve higher rewards in the long run. RL does not require labeled data, because learns from observations and rewards via interaction with the environment.

The sequence of states, actions and rewards that the agent experiences is called a trajectory, and it look like this:

$$(S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots, S_{T-1}, A_{T-1}, R_T, S_T) \tag{2.1}$$

This sequence of *state-action-reward* can be finite or infinite, depending on the environment and the agent's goal. Pretty good example of this is the game of chess, where the game ends when one of the players wins or the game is a draw. In this case, the trajectory is finite, and the agent's goal is to maximize the cumulative reward over time T. On the other hand, the self-driving car example is an infinite-horizon problem, where the agent's goal is to maximize the cumulative reward over an infinite time horizon or until the car reaches its destination [8].

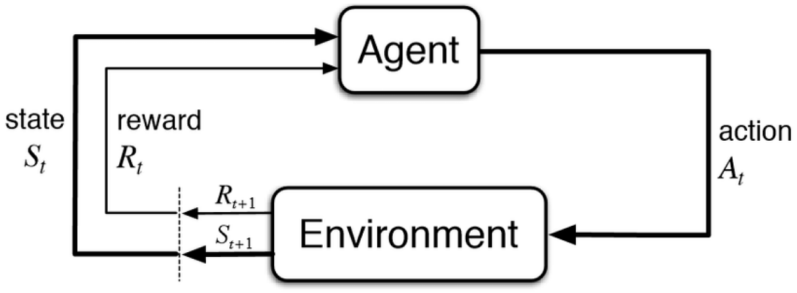

Figure 2.1: The agent interacts with the environment and learns to maximize the cumulative reward over time T.

Because RL agent is categorized by the way it learns, there are many different types of RL agents. We will try to differentiate them in this chapter, because it is important to

understand teh differences between them, to understand the behavior of the agent and to choose the right algorithm for the problem. Based on the different computable approaches we can categorize RL agents:

- **Value-based**
  - No Policy (implicit)
  - Value function
- **Policy-based**
  - Policy
  - No Value function
- **Actor-critic**
  - Policy
  - Value function
- **Model-based**
  - Policy and/or Value function
  - No model
- **Model-free**
  - Policy and/or Value function
  - Model

Model-based algorithm are closely relate to *Planning algorithms* which are trying to create a model of the environment and use it to make decisions, they have knowledge of the transition probabilities and rewards from the beginning, we will discuss them in section 2.3. *Value-based* algorithms can implicitly infer a policy, but do not explicitly compute it, we will discuss them in section 2.4, and *Policy-based* algorithms compute a policy function by themselves, and are discussed in section 2.5. Another agents like *Actor-critic methods* compute a policy function and a state-value function at the same time [5, 7].

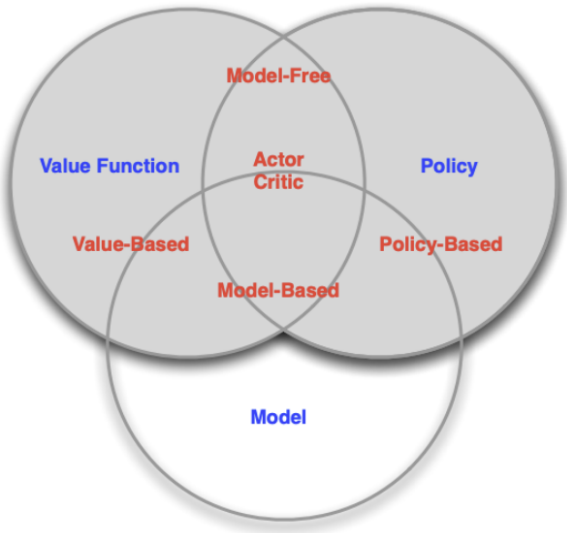

Figure 2.2: The relationship between model, value function and policy. Source: [8]

2.2 Markov Theory

This section provides introduction to Markov theory, which is the fundamental building blocks of Reinforcement Learning. We begin by *Markov Process*, described in section 2.2.1,

which is a stochastic process that satisfies the Markov property. We then move on to *Markov Reward Process*, described in section 2.2.2, which is a Markov process with a reward function. Finally, we describe the most important for RL *Markov Decision Process*, described in section 2.2.3, which is a Markov Reward process with a decision-making ability.

2.2.1 Markov Process

The Markov process (MP) (also called Markov Chains) describes the states of an environment and models the dynamics of state transitions. In an MP, an agent can only observe the changing states of the environment and has no influence over them. Markov processes have two key properties. Firstly, state transitions are non-deterministic and influenced by randomness. States are modeled as realizations of random variables, defined in section 2.2.1. Secondly, the future state is only dependent on the current state, and not on previous states, simplifying causality with the Markov property [8].

Probability Functions

The first property of an MP states that each concrete state of an environment is the realization of a discrete random variable X from set V, with a certain probability $P(X = x)$, where $x \in V$. A state is a realization of a random experiment that the environment assumes with a certain probability, and this can be represented as a probability function:

$$P(X = X(\omega)) = P(X = x) \tag{2.2}$$

The repeated successive execution of a random experiment can be represented as a stochastic process, which is a sequence of random variables, e.g., $X_t(\omega), X_{t+1}(\omega), \dots X_n(\omega)$, where a single term can be shortened to X_t and it represents the state of the environment at time t [8, 4].

Stochastic Process (Random Process)

A stochastic process is defined as a collection of random variables defined on a common probability space $(\Omega, \mathcal{F}, P)$, where Ω is a sample space, $\mathcal{F}$ is a σ-algebra, and P is a probability measure; and the random variables, indexed by some set T, all take values in the same mathematical space S, which must be measurable with respect to some σ-algebra Σ.

In other words, for a given probability space $(\Omega, \mathcal{F}, P)$ and a measurable space (S, Σ), a stochastic process is a collection of S-valued random variables, which can be written as: [10]

$$\{X(t) : t \in T\}.$$

In stochastic processes the probability that the environment assumes a certain state depends on the realized states of previous random variables. For example, if the weather forecast is assumed to be a stochastic process, then the yesterday's weather may still have an influence on tomorrow's weather. To represent this causality complicates the modeling of stochastic processes, so that with definition of Markov property the dependence of future states is assumed only on the current state. This is the second important property of Markov process [8].

Markov Property

The Markov property, which is defined using conditional probability, states that "The future is independent of the past, given the present.„

The stochastic process has the Markov property if and only if, for all time steps $t \in N$, the conditional probability of the next state given the current state is equal to the conditional probability of the next state given all the previous states $X_1, \ldots, X_t$. $\mathbb{P}[X_{t+1}|X_t] = \mathbb{P}[X_{t+1}|X_1, \ldots, X_t]$

This property has several advantages in practical reinforcement learning, including the uniqueness and distinctiveness of states, as well as the ability to precisely formulate the probability of state transitions, defined as: [8]

$$\mathcal{P}(x'|x) = \mathbb{P}(X_{t+1} = x'|X_t = x) \tag{2.3}$$

Given n possible states, $s \in S$, then the probability of transitioning from state s to state s' can be represented as a matrix P, and because probability summation rule, the sum of transition probabilities from state s to any other state s' must equal to 1.

Definition 1. *The Markov process is stochastic process that satisfies the Markov property and is described as tuple* $(\mathcal{S}, \mathcal{P})$ *for which holds:[1]*

- $\mathcal{S} = s_1, s_2, ..., s_n$ *is a finite set of states*
- $\mathcal{P}$ *is an* $n \times n$ *transition probability matrix which sums to 1 for each row, so each value* p_{ij} *is the probability of transitioning from state* s_i *to state* s_j *in interval* $\langle 0; 1 \rangle$

$$\mathcal{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1n} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & p_{n3} & \cdots & p_{nn} \end{bmatrix}$$

Starting States

The probability distribution of start states is denoted as $\mu : N \to [0, 1]$ in order to perform simulations and compute the probability distribution of states at specific future time steps. A Markov Process is fully specified by the transition probability function P, which governs the complete dynamics of the process.

- Specification of the transition probability function P.
- Specification of the probability distribution of start states (denote this as $\mu : N \in [0, 1]$).

Given μ and P, we can generate sampling traces of the Markov Process and answer questions such as probability distribution of states at specific future time steps or expected time of first occurrence of a specific state, given a certain starting probability distribution μ. The separation of concerns between P and μ is key to the conceptualization of Markov Processes [4].

Terminal States

Markov Processes can terminate at specific states (e.g., based on rules for winning or losing in games). Termination can occur after a variable number of time steps (episodic) or after a fixed number of time steps (as in many financial applications). If all sampling traces of the Markov Process reach a terminal state, they are called episodic sequences. The notion of episodic sequences is important in Reinforcement Learning. In some financial applications, the Markov Process terminates after a fixed number of time steps T, and states with time index $t = T$ are labeled as terminal states. States with time index $t < T$ transition to states with time index $t + 1$ [4].

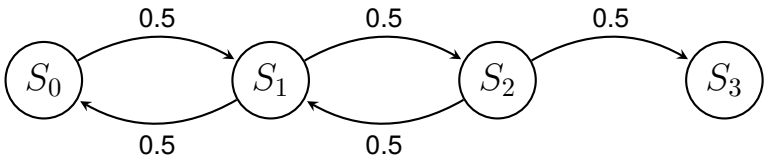

Figure 2.3: Markov Process with Start state S_0 and Terminal state S_3, because there is no edge from S_3.

The examples shown here only include states and transition probabilities. To fully define an environment within the framework of RL, actions and rewards also need to be defined.

2.2.2 Markov Reward Process

Markov Reward Process (MRP) is a Markov Process with rewards. These rewards are random, and all we need to do is to specify the probability distributions of these rewards as we make state transitions. The main purpose of Markov Reward Processes is to calculate how much reward we would accumulate (in expectation, from each of the non-terminal states) if we let the process run indefinitely, bearing in mind that future rewards need to be discounted appropriately γ (otherwise, the sum of rewards could blow up to ∞). In order to solve the problem of calculating expected accumulative rewards, defined in section 2.2.2, from each non-terminal state, we will first set up some formalism for Markov Reward Processes and develop some theory on calculating rewards accumulation.

The main objective of an RL agent is to maximize the sum of rewards from each time-step. The agent can observe different episodes in the Markov process, but lacks the means to determine the actual quality of an episode. By calculating the reward, we can precisely measure the goodness of an episode or even a single state using the *state-value function*, defined in section 2.2.2. This allows the agent to actively transition to favorable states and maximize the reward.

Definition 2. *Markov Reward Processes is a tuple* $(\mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma)$ *for which holds:[4]*

- $\mathcal{S} = s_1, s_2, ..., s_n$ *is a finite set of states*
- $\mathcal{P}$ *is an* $n \times n$ *transition probability matrix which sums to 1 for each row, so each value* p_{ij} *is the probability of transitioning from state* s_i *to state* s_j *in interval* $\langle 0; 1 \rangle$
- $\mathcal{R}$ *is a sequence of random variables* $R_1, R_2, ..., R_n$ *where* R_t *is a random variable that represents the reward for transitioning from state* s_t *to state* s_{t+1}

- γ *is a discount factor in interval* $\langle 0; 1\rangle$

$$\mathcal{P}(s, r, s') = I\!P\left[R_{t+1} = r, S_{t+1} = s' | S_t = s\right] \text{ for time steps } t = 0, 1, 2, \ldots \tag{2.4}$$

$$\text{for all } s \in N, r \in D, s' \in S, \text{ such that } \sum_{s' \in S} \sum_{r \in D} \mathcal{P}(s, r, s') = 1 \text{ for all } s \in N$$

Reward function

The reward function $\mathcal{R}(s)$ is a function that maps a state s to a reward r and specify how much reward and agent expects from the environment given current state s. If an agent is in a state s at time t, the agent receives reward R_{t+1} at time $t + 1$, when it transitions to a subsequent state s'. Rewards of an episode can be represented as a sequence $(R_1, R_2, \ldots, R_t)$ [8].

When to receive reward? An RL agent, defined in `\cref{TODO}`, should receive a reward for a good action and a penalty for a bad action. Good actions are those that lead to the agent's main goal, while bad actions are those that lead to a state that is not desirable for the agent. That is, an agent should not receive (one small) reward when it can then receive a large penalty, for example in chess by taking one piece, when it can then lose the game by getting checkmate [7].

Expected reward

The expected reward, denoted as G_t at time t is the discounted sum of rewards in a single episode. Can be used to calculate the sum of discounted rewards in an episode.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.5}$$

We can also calculate the expected rewards for state-action pairs as a two-argument function $r : S \times A \to R$. The expected reward of a state-action pair as two-argument function $r : \mathcal{S} \times \mathcal{A} \leftarrow \mathbb{R}$ is defined as:

$$r(s, a) = \sum_{t=0}^{\infty} \gamma^t R_{t+1} P(S_{t+1} = s | S_t = s, A_t = a) \tag{2.6}$$

Discount factor γ The calculation also involves the discount factor γ which is a value in the interval $\langle 0, 1\rangle$. If γ is equal to one, the series value goes to infinity. The agent can only calculate the reward in the case of always terminating episodes. If γ is less than one, the reward has a finite value, allowing the agent to determine the quality of an episode. The discount factor is not only useful mathematically but also for tuning the agent's rewards. If early rewards in an episode are more significant than later ones, γ should be close to zero. If the rewards represent monetary gains, then it is the case, as early rewards earn additional interest. On the other hand, the closer γ is to one, the more important later rewards are.

State-value Function

The state-value function provides information about the long-term expected reward for each state in an environment. With this information, an agent can determine which state to transition to in order to maximize the reward of an episode. Specifically, the agent should choose the state that has the highest long-term expected reward. If the agent observes a sequence of states and rewards, it can remember the subsequent rewards for each state, calculate the reward of an episode, and iteratively update the probabilities for higher occurrence of rewards over multiple episodes. As the number of episodes approaches infinity, the estimated probabilities converge to the true probabilities, and the long-term expected reward of a state can be accurately determined. Intuitively, the more episodes and consequent rewards an agent observes, the better it can estimate the value of each state. We will introduce the methods for determining the state-value function and the action-value function in the next subsection, along with a recursive iterative approach for calculating the state-value function based on Bellman's equation [8].

Since episodes may start with different states due to state transitions probabilities, the expected reward of a particular state is the expected value of the conditional density function over the probabilities of rewards for that state. Thus, G_t can be treated mathematically as a continuous random variable. To derive *Bellman's equation*, defined later in this subsection, we need to make use of the definition of the *(Recursive) State-value function* [8, 4].

$$\begin{aligned} v(s) &= E[G_t|S_t = s] \\ &= E[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= E[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \\ &= \mathcal{R}(s) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s) v(s') \end{aligned} \tag{2.7}$$

where γ is the discount factor, $\mathcal{R}(s)$ is the immediate reward for state s, $\mathcal{P}(s'|s)$ is the transition probability from state s to state s', and $\mathcal{S}$ is the set of all possible states in the environment. The last equation expresses that the long-term expected reward of a state depends only on the immediate reward and the long-term expected reward of the subsequent states.

In the case of Finite Markov Reward Processes, let's assume that the state space is denoted as $\mathcal{S} = \{s_1, s_2, ..., s_n\}$, and the subset of states of interest is denoted as $\mathcal{N}$ with $m \leq n$ states. We can use bold-face notation to represent functions as column vectors and matrices, since we are dealing with finite states/transitions. So, V is a column vector of length m, P is an $m \times m$ matrix, and R is a column vector of length m (with rows/columns corresponding to states in $\mathcal{N}$). We can express the equation in vector and matrix notation as follows:

$$\begin{aligned} V &= R + \gamma P \cdot V \\ &= (I_m - \gamma P)^{-1} \cdot R \end{aligned} \tag{2.8}$$

where I_m is the identity matrix of size $m \times m$, and γ is the discount factor [4].

By extending the equation to include actions, we arrive at the most important equation in all of reinforcement learning: the Bellman equation, defined in **??**. It states that to calculate the long-term expected reward from a state, an agent only needs to add together the reward of the current state and the long-term expected reward of the next state [8].

2.2.3 Markov Decision Process

The Markov Decision Process (MDP) allows an agent to actively influence changes in the state of the environment through its actions. Within the MDP, the agent has the ability to jointly determine the subsequent state to which the environment should transition. The agent's primary goal is to strategically choose actions that maximize expected payoff. In the previous subsections, we addressed the aspect of *sequential uncertainty (e.g., MRP)* using the Markov process framework and extended it to include the uncertain reward $R_t \in \mathcal{R}$ at each state transition $p(s', r, s)$, referred to as Markov reward processes. However, this framework lacks the notion of *sequential decision making* and in this section this notion is introduced in terms of MDP, a generalization of MRP that includes the notion of *sequential decision making* [4].

Definition 3. *Markov decision process (MDP) is Markov reward process with actions. It is an tuple* $(S, A, \mathcal{P}, \mathcal{R}, \gamma)$*, where:*

- $\mathcal{S}$ *is finite set of states* S *(known as the State Space), a set* $T \subseteq S$ *(known as the set of Terminal States)*
- $\mathcal{A}$ *is finite set of actions* A *(known as the Action Space)*
- $\mathcal{P}$ *is a transition probability function* $p(s', r, s, a)$*, which is a function that maps a state* s*, an action* a*, a next state* s' *and a reward* r *to a probability* $p(s', r, s, a)$
- $\mathcal{R}$ *is a reward function* $r(s, a)$*, which is a function that maps a state* s *and an action* a *to a reward* $r(s, a)$
- $\gamma \in [0, 1]$ *is the discount factor*

Stochastic Policy

A policy, denoted by π, in the context of MDPs, is a function that maps states to actions. It represents the agent's strategy or decision-making rule for selecting actions based on the current state.

Definition 4. *A policy is defined as:* $\pi : S \to A$*, where* S *is the set of states and* A *is the set of actions. Notation for a policy is as follows:*

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s) \tag{2.9}$$

Policy refers to the specification of an Agent's actions based on the current state in a Markov Decision Process. The policy can be deterministic, meaning it selects a single action for each state. It is represented as a function $\pi_D : N \to A$, where $\pi_D(s)$ represents the action to be taken in state s. Or it can be stochastic, meaning it selects actions probabilistically based on some probability distribution over actions for each state. A policy is a function that maps states to actions and represents the agent's decision-making strategy. Mathematically, a Policy is represented as a function $\pi : N \times A \to [0, 1]$, where N represents the state space and A represents the action space. The function $\pi(s, a)$ represents the probability of taking action a in state s at time step $t = 0, 1, 2, \ldots$, for all $s \in N$ and $a \in A$. It is assumed that the sum of probabilities for all actions in a given state is equal to 1. A Policy is usually assumed to be Markovian, meaning that the action probabilities depend only on the current state and not the history. It is also assumed to be stationary, meaning that the

action probabilities do not change over time. However, if the policy needs to depend on the time step t, we can include t as part of the state, which would make the policy stationary but may increase computational cost due to the enlarged state space. In the more general case, where states or rewards are uncountable, the same concepts apply except that the mathematical formalism needs to be more detailed and more careful. Specifically, we'd end up with integrals instead of summations, and probability density functions (for continuous probability distributions) instead of probability mass functions (for discrete probability distributions). For ease of notation and more importantly, for ease of understanding of the core concepts (without being distracted by heavy mathematical formalism), we've chosen to stay with discrete-time, countable S, countable A [4].

Policies and action-value functions are closely related and are used interchangeably in many reinforcement learning algorithms, but they are conceptually distinct.

State-value Function for Stochastic Policy π

The value function $V^\pi(s)$ for a stochastic policy π is the expected cumulative discounted reward the agent can obtain from state s by following policy π and then continuing to follow π thereafter. It can be computed recursively using the Bellman equation, which relates the Value Function of a state to the rewards and transitions of the MDP. It is defined as:

$$\begin{aligned} v^\pi(s) &\doteq {}_\pi\left[g_t | S_t = s\right] \\ &= \left[r_{t+1} = \gamma g_{t+1} | S_t = s\right] \\ &= {}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(r, s'|s, a)\left[r + \gamma v_\pi(s')\right], \text{ for all } s \in \mathcal{S} \end{aligned} \tag{2.10}$$

where $\mathbb{E}_\pi$ denotes the expectation with respect to the states and rewards generated by following policy π.

Action-value Function

Action-Value Function $q^\pi(s, a)$, which maps a (state, action) pair to the expected reward originating from that pair when evaluated with policy π. Mathematically, an action-value function is defined as $Q : S \times A \to \mathbb{R}$, where S is the set of states and A is the set of actions. The Action-value function $a^\pi(s, a)$ represents the expected cumulative reward from taking action a in state s, following the policy π thereafter. In other words, it quantifies the value of selecting a particular action in a particular state under a specific policy. The Action-Value Function is denoted by q^π and is crucial in developing various Dynamic Programming and Reinforcement Learning algorithms for the MDP Prediction problem.

Definition 5. *The action-value function $Q^\pi(s, a)$ for a policy π is the expected cumulative discounted reward the agent can obtain from state s, taking action a, and then following policy π thereafter. It is defined as:*

$$q^\pi(s, a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s, A_0 = a\right]$$

To avoid confusion, v^π is referred to as the State-Value Function, while q^π is referred to as the Action-Value Function. The Action-Value Function provides information about the

expected returns from specific state-action pairs and is useful for making decisions about which actions to take in an MDP. The relationship between the state-value function and the action-value function can be defined as:

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

Optimal State-value function

The goal of an agent is to maximize the reward, which means finding a policy that yields the highest possible return. To compare two policies, let π and π' be two different stochastic policies. Then π' is considered better or equal to π if $v_{\pi'}(s) \geq v_\pi(s)$ holds for all states s. It can be shown that there is always at least one policy that is better or equal to all other policies, and this policy is called the optimal policy π^*. If an agent uses the optimal policy, then for all states and actions the agent will use the optimal state-value function and the optimal action-value function:

Definition 6. *The Optimal State-value function $v^*(s)$ is the maximum value function over all possible policies and is defined as:*

$$v^*(s) = \max_\pi v^\pi(s) \tag{2.11}$$

Optimal Action-value function

Conversely, if the optimal action-value function is found, the optimal policy can be derived.

Definition 7. *The Optimal Action-value function $q^*(s, a)$ is the maximum action-value function over all possible policies. It is defined as:*

$$q^*(s, a) = \max_\pi q^\pi(s, a) \tag{2.12}$$

The optimal policy function $\pi^*(a|s)$ always selects an action a with a selection probability of 1 for which $q_*(s, a)$ is maximal for a given state s. It also follows that the optimal policy is deterministic, meaning for the same state, the optimal policy function always selects exactly the same action. In contrast, Definition 2.6 allows for stochastic behavior of the policy. The transition from stochastic to deterministic policy is explained by the greedy selection of actions in the last equation. As will be shown in the following sections, all reinforcement learning algorithms presented here use a

Bellman's Optimality Equation

The optimal state-value function $v^*(s)$ must satisfy Bellman's equation, which provides an recursive rule for determining the long-term expected return of each state. In order to incorporate actions, the State-value function for a Markov Reward Process (MRP) is extended to include actions, leading to Bellman's optimality equation for $v^*(s)$:

$$\begin{aligned} v^*(s) &= \max_{a \in \mathcal{A}} \in \mathcal{A}\mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_{a \in \mathcal{A}} \sum_{s', r} p(r, s'|s, a) \left[r + \gamma v^*(s') \right] \end{aligned} \tag{2.13}$$

This equation derives the recursive relation to the subsequent state, allowing the reference to the optimal policy to be omitted. Similarly, the Action-value function $q^*(s, a)$ can be derived as [4]:

$$\begin{aligned} q^*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q^*(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s',r} p(r, s'|s, a) \left[r + \gamma \max_{a' \in \mathcal{A}} \in \mathcal{A} q^*(s', a') \right] \end{aligned} \tag{2.14}$$

The Bellman Optimality equation can be used to recursively calculate value functions by updating the value of each state based on the immediate reward and the long-term expected reward of the next state [4].

2.3 Model-based methods

As mentioned in Section section 2.1, RL algorithms can be broadly categorized into two main types: model-free and model-based. Model-based methods rely on *planning* as they primary component, while model-free methods rely on *learning* [7]. Despite their distinctions, both approaches involve computing value functions and using them to update approximate value functions based on future events.

One popular approach is to use a probabilistic model of the environment, which can be represented as a transition function $P(s_{t+1}|s_t, a_t)$ that describes the probability of transitioning from state s_t to state s_{t+1} when taking action a_t. Some popular probabilistic model-based RL methods include Monte Carlo Tree Search, Neural Network Dynamics, and Probabilistic Ensembles with Trajectory Samplin.

Another approach in model-based RL is to use a learned deterministic model of the environment, which can be represented as a function $f(s_t, a_t)$ that directly maps states and actions to next states. Deterministic model-based RL methods, such as World Models, learn an encoder to represent states, a recurrent neural network (RNN) to model dynamics, and a decoder to generate predicted next states.

2.3.1 Planning

A model of the environment refers to anything that an agent can use to predict the outcome of its actions. It can be either stochastic, where there are multiple possible outcomes with associated probabilities, or deterministic, where the outcome is fixed. Distribution models provide a description of all possible outcomes and their probabilities, while sample models generate a single outcome sampled from the probabilities.

Planning, in the context of artificial intelligence, refers to a computational process that takes a model of the environment as input and produces or improves a policy for interacting with the environment. There are two main approaches to planning: *state-space* planning and *plan-space* planning. In *state-space* planning, the focus is on searching through the **space of states** to find an optimal policy or path to a goal. Value functions are computed over states to guide the search. On the other hand, in *plan-space* planning, the search is conducted through the **space of plans**, where operators transform one plan into another, and value functions, if any, are defined over the space of plans. Plan-space methods are challenging to efficiently apply to the stochastic sequential decision problems that are the primary focus in reinforcement learning [7].

The common structure in *state-space* planning and learning methods, as presented in this chapter, is that value functions are computed as intermediate steps using simulated experience. Value functions are used to estimate the expected future rewards of different actions or states, and they play a key role in improving the policy of an agent. Simulated experience is generated by using the model to simulate the environment, and the agent can update its value functions based on this simulated experience to guide its decision-making process.

$$\text{model} \longrightarrow \text{simulated experience} \xrightarrow{\text{backups}} \text{values} \longrightarrow \text{policy}$$

Dynamic programming methods fit the structure of making sweeps through the space of states, generating possible transitions, computing backed-up values, and updating state estimates. Other state-space planning methods also fit this structure, with differences in updates, order, and retention of backed-up information. Planning methods are related to learning methods in estimating value functions through backing-up operations. Learning methods use real experience from the environment, while planning uses simulated experience from a model. Ideas and algorithms can be transferred between planning and learning. Planning in small, incremental steps allows for efficient interruption and redirection, which is beneficial for intermixing planning with acting and learning of the model. Planning in small steps may be the most efficient approach for large planning problems [7].

Input : S: Set of states
A: Set of actions
α: Learning rate
γ: Discount factor

while *True* **do**
  Select a state $s \in S$, action $a \in A(s)$ at random;
  Send s, a to a sample model, and obtain a sample next reward r and a sample next state s';
  Apply one-step tabular Q-learning update to s, a, r, s':

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a) \right)$$

end

Algorithm 1: Random-sample one-step tabular Q-planning

2.3.2 Trajectory Sampling

2.3.3 Summary

Model-based RL offers several advantages over model-free methods. One major advantage is sample efficiency, as the agent can use the learned model of the environment to plan and generate simulated trajectories for learning, reducing the need for costly real-world interactions. Additionally, model-based RL can enable the agent to handle complex, high-dimensional state spaces and long-horizon tasks more effectively.

However, model-based RL also faces challenges. One challenge is the accuracy of the learned model, as any errors in the model can lead to suboptimal policies. Another challenge is the computational cost of planning and decision-making using the learned model, as it requires additional computation compared to direct action selection in model-free methods.

In conclusion, model-based RL is a promising approach that can offer sample-efficient learning and improved performance in complex environments. Various methods, such as probabilistic models and deterministic models, have been proposed in the literature. Despite some challenges, model-based RL continues to be an active area of research in machine learning.

2.4 Value-based methods

2.5 Policy-based methods

2.6 Model-free methods

2.6.1 Value-based methods

TODO

Dynamic Programming

TODO

Policy Iteration TODO

Value Iteration TODO

Monte Carlo Methods TODO

Temporal Difference Methods TODO

2.6.2 Policy-based methods

Policy Gradient Methods

TODO

Actor-Critic Methods

TODO

Monte Carlo Policy Gradient Methods

TODO

Chapter 3

Stock Portfolio Allocation

TODO

3.1 Environment

TODO

3.1.1 Reward Function

3.2 Data Engineering

TODO

3.2.1 Datasets

TODO

Chapter 4

Experiments and Results

TODO

4.1 Agent

4.2 Computational Resources

TODO

4.3 Experiments

TODO

4.4 Summary

Chapter 5

Conclusions

TODO

Bibliography

[1] Abate, A., Andriushchenko, R., Češka, M. and Kwiatkowska, M. Adaptive formal approximations of Markov chains. *Performance Evaluation*. 2021, vol. 148, p. 102207. DOI: https://doi.org/10.1016/j.peva.2021.102207. ISSN 0166-5316. Available at: https://www.sciencedirect.com/science/article/pii/S0166531621000249.

[2] Guan, M. and Liu, X.-Y. *Explainable Deep Reinforcement Learning for Portfolio Management: An Empirical Approach*. arXiv, 2021. DOI: 10.48550/ARXIV.2111.03995. Available at: https://arxiv.org/abs/2111.03995.

[3] Oshingbesan, A., Ajiboye, E., Kamashazi, P. and Mbaka, T. *Model-Free Reinforcement Learning for Asset Allocation*. arXiv, 2022. DOI: 10.48550/ARXIV.2209.10458. Available at: https://arxiv.org/abs/2209.10458.

[4] Rao, A. and Jelvis, T. *Foundations of Reinforcement Learning with Applications in Finance*. CRC Press, 2022. ISBN 9781000801101. Available at: https://books.google.cz/books?id=n_-VEAAAQBAJ.

[5] Silver, D. *Introduction to Reinforcement Learning with David Silver* [online]. 2015 [cit. 2023-04-08]. Available at: https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver.

[6] Soeryana, E., Fadhlina, N., Firman, S., Rusyaman, E. and Supian, S. Mean-variance portfolio optimization by using time series approaches based on logarithmic utility function. *IOP Conference Series: Materials Science and Engineering*. january 2017, vol. 166, p. 012003. DOI: 10.1088/1757-899X/166/1/012003.

[7] Sutton, R. and Barto, A. *Reinforcement Learning, second edition: An Introduction*. MIT Press, 2018. Adaptive Computation and Machine Learning series. ISBN 9780262039246. Available at: https://books.google.cz/books?id=5s-MEAAAQBAJ.

[8] Vosol, D. *Aplikace posilovaného učení v řízení autonomního vozidla*. Brno, CZ, 2022. Master's thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://www.fit.vut.cz/study/thesis/25127/.

[9] Wikipedia contributors. *Modern portfolio theory — Wikipedia, The Free Encyclopedia*. 2022. [Online; accessed 28-December-2022]. Available at: https://en.wikipedia.org/w/index.php?title=Modern_portfolio_theory&oldid=1126978098.

[10] Wikipedia contributors. *Stochastic process — Wikipedia, The Free Encyclopedia*. 2023. [Online; accessed 10-April-2023]. Available at: https://en.wikipedia.org/w/index.php?title=Stochastic_process&oldid=1148510872.

[11] Širůček, M. and Křen, L. Application of Markowitz Portfolio Theory by Building Optimal Portfolio on the US Stock Market. *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*. Mendel University Press. september 2015, vol. 63, no. 4, p. 1375–1386. DOI: 10.11118/actaun201563041375. Available at: http://dx.doi.org/10.11118/actaun201563041375.