Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Bachelor Thesis

Reinforcement Learning in Algorithmic Trading

Bold-Erdene Bayaraa

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Bold-Erdene Bayaraa

Informatics

Thesis title

Reinforcement learning in Algorithmic Trading

Objectives of thesis

This thesis will explore the application of deep reinforcement learning (DRL) techniques to develop automated trading strategies. The work will begin with a theoretical foundation of reinforcement learning concepts, including Markov Decision Processes (MDPs), Q-learning, and relevant DRL architectures. Several DRL-based trading agents will be developed and evaluated within a simulated trading environment. Their performances will be compared not only to each other but also some baseline strategies. The thesis will conclude with a discussion of the strengths, limitations, and overall potential of deep reinforcement learning in the financial trading domain.

Methodology

This thesis will explore the application of deep reinforcement learning (DRL) to trading, building upon a theoretical RL foundation. Multiple DRL agents will be developed, incorporating historical data and technical indicators into their state representations. A simulated trading environment will facilitate the evaluation of these agents against both baseline strategies and each other. Performance analysis will employ key trading metrics, leading to insights on the strengths, limitations, and potential of DRL within the financial trading domain.

The proposed extent of the thesis

30-40

Keywords

Reinforcement Learning, Machine Learning, Algorithmic Trading

Recommended information sources

LAPAN, Maxim. Deep Reinforcement Learning Hands-On. 1. Packt Publishing, 2020. ISBN 9781838826994.
SUTTON, Richard S. a BARTO, Andrew G. Reinforcement Learning. 2. MIT Press, 2018. ISBN 0262352702.
SZEPESVÁRI, Csaba. Algorithms for Reinforcement Learning. 1. Morgan & Claypool, 2010. ISBN 9781608454921.

Expected date of thesis defence

2023/24 SS – PEF

The Bachelor Thesis Supervisor

Ing. Martin Pelikán, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 11. 3. 2024

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 15. 3. 2024

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 15. 03. 2024

Declaration

I declare that I have worked on my bachelor thesis titled ***"Reinforcement Learning in Algorithmic Trading"*** by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break any copyrights.

In Prague on 15th of March 2024 ______________________________

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Ing. Martin Pelikán, Ph.D., for his invaluable guidance and support throughout the development of this thesis. I am also deeply grateful to my family and girlfriend for their unwavering encouragement and belief in me during my studies.

Reinforcement Learning in Algorithmic Trading

Abstract

The field of algorithmic trading has traditionally relied on established financial models and technical analysis. However, the rise of deep reinforcement learning (DRL) offers a compelling new paradigm for developing trading strategies. This thesis explores the potential of DRL for algorithmic stock trading by implementing and comparing the performance of five advanced DRL algorithms (A2C, DDPG, PPO, TD3, SAC) against a traditional benchmark, the Dow Jones Industrial Average (DJIA) index.

The study utilizes historical market data to train and evaluate these agents. Key performance metrics, including Sharpe ratio, drawdown, and risk-adjusted returns, are employed to assess their effectiveness. Results demonstrate that DRL-based trading strategies hold the potential to outperform traditional benchmark-following approaches, particularly during periods of market volatility. Notably, the DDPG agent displayed remarkable resilience during downturns, highlighting DRL's adaptability.

This thesis contributes to the growing body of research on the intersection of machine learning and finance. It provides insights into the strengths and limitations of different DRL algorithms for trading applications, paving the way for further research into the development of robust and profitable algorithmic trading systems.

Keywords: Reinforcement Learning, Deep Reinforcement Learning, Machine Learning, Algorithmic Trading, Financial Markets, Trading.

Posilovací učení v algoritmickém obchodování

Abstrakt

Oblast algoritmického obchodování se tradičně opírá o zavedené finanční modely a technickou analýzu. Rozvoj hlubokého posilovacího učení (deep reinforcement learning, DRL) však nabízí nové přesvědčivé paradigma pro vývoj obchodních strategií. Tato práce zkoumá potenciál DRL pro algoritmické obchodování s akciemi implementací a porovnáním výkonnosti pěti pokročilých algoritmů DRL (A2C, DDPG, PPO, TD3, SAC) s tradičním benchmarkem, indexem Dow Jones Industrial Average (DJIA).

Studie využívá historická tržní data k tréninku a vyhodnocení těchto agentů. K hodnocení jejich účinnosti jsou použity klíčové ukazatele výkonnosti, včetně Sharpeho poměru, čerpání a výnosů očištěných o riziko. Výsledky ukazují, že obchodní strategie založené na DRL mají potenciál překonat tradiční přístupy sledující benchmark, zejména v obdobích volatility trhu. Zejména agent DDPG vykazoval pozoruhodnou odolnost během poklesů, což zdůrazňuje přizpůsobivost DRL.

Tato práce přispívá k rostoucímu počtu výzkumů v oblasti propojení strojového učení a financí. Poskytuje vhled do silných stránek a omezení různých algoritmů DRL pro obchodní aplikace a otevírá cestu k dalšímu výzkumu vývoje robustních a ziskových algoritmických obchodních systémů.

Klíčová slova: Posilovací učení, Hluboké posilovací učení, Strojové učení, Algoritmické obchodování, Finanční trhy, Obchodování.

Table of content

1 Introduction 10

2 Objectives and Methodology 11

2.1 Objectives 11

2.2 Methodology 11

3 Literature Review 12

3.1 Financial Trading 12

3.1.1 Financial trading 12

3.1.2 Algorthmic trading 12

3.1.3 The Moving Average Convergence and Divergence (MACD) 13

3.1.4 Relative strength index (RSI) 13

3.1.5 Limitations of pre-defined rule based algorithmic trading. 14

3.2 Reinforcement Learning 14

3.2.1 Markov property 15

3.2.2 Markov decision process 15

3.2.3 Return 17

3.2.4 Policy 17

3.2.5 Value functions 18

3.2.6 Bellman equation 19

3.2.7 Q-learning 19

Temporal difference 20

Exploration vs. Exploitation 21

Epsilon-greedy strategy 21

Hyperparameters in Q-learning 22

Practical Limitations 22

3.3 Deep Q-network (DQN) 22

3.3.1 Neural networks 23

3.3.2 Activation function 25

3.3.3 Optimization Algorithm 26

3.3.4 Experience Replay 27

3.3.5 Target Network 27

3.3.6 Practical Limitations 28

3.4 Policy Gradient Method 28

3.4.1 PPO (Proximal Policy Optimization) 29

3.5 Actor-critic methods 29

3.5.1 A2C (Advantage Actor-Critic) 30

3.5.2 SAC (Soft Actor-Critic) 30
3.6 Continuous Control Methods 31
3.6.1 DDPG (Deep Deterministic Policy Gradient) 31
3.6.2 TD3 (Twin Delayed DDPG) 32

4 Practical Part 32
4.1 Data 32
4.1.1 Data Acquisition and Preparation 32
4.1.2 Data Preprocessing 33
4.1.3 Data Transformation 34
4.1.4 Dataset Split and Storage 34
4.2 Train 34
4.2.1 Package Installation 35
4.2.2 Market Environment Creation 35
4.2.3 DRL Agent Training 36
4.3 Agent Evaluation 37
4.3.1 DJIA Index 37
4.3.2 Metrics for Evaluating Performance 38

5 Results and Discussion 39
5.1 Comparison of DRL agents 39
5.2 DDPG vs. DJI Benchmark 41
5.3 Limitations and future directions 43

6 Conclusion 45

7 References 46

8 List of figures and abbreviations 49
8.1 List of figures 49
8.2 List of Abbreviations 50

1 Introduction

Financial markets present a complex environment where investors strive to optimize returns while managing risk. Traditional algorithmic trading strategies often rely on technical indicators and established financial models. However, these approaches can be limited in their ability to adapt to rapidly changing market dynamics. The emergence of deep reinforcement learning (DRL) offers a compelling alternative, empowering algorithms to learn directly from market data and make adaptive trading decisions.

This thesis investigates the potential of DRL for algorithmic trading. It centres on a comparative analysis of five cutting-edge DRL algorithms (A2C, DDPG, PPO, TD3, SAC), benchmarked against the Dow Jones Industrial Average (DJIA). Using historical market data, these agents are trained and evaluated on their ability to generate profitable trading strategies. Performance is assessed through key financial metrics, including Sharpe ratio, maximum drawdown, and risk-adjusted returns.

Existing research suggests that DRL holds the potential to navigate market complexities and outperform traditional approaches. This thesis aims to contribute to this growing body of knowledge by offering a rigorous comparison of different DRL algorithms in the trading domain. The results of this study will shed light on the strengths, weaknesses, and practical applicability of DRL for algorithmic trading.

This thesis is structured to first establish a theoretical foundation in financial trading and reinforcement learning, with a specific focus on Deep Q-Networks and policy gradient methods. It then delves into the practical implementation, detailing the data processing, agent training process, and evaluation criteria for the DRL agents. Subsequently, it presents the experimental results and discussions, analysing the agents' performance against the DJIA benchmark.

Ultimately, this research aims to illuminate the potential of reinforcement learning for developing intelligent and robust algorithmic trading strategies.

2 Objectives and Methodology

2.1 Objectives

The primary objective of this thesis is to evaluate the effectiveness of various deep reinforcement learning (DRL) algorithms for algorithmic stock trading. Through a comparative analysis, this study will examine the performance of agents A2C, DDPG, PPO, TD3, and SAC. Their performance will be benchmarked against a traditional market index, such as the DJIA, using key financial metrics to assess their potential for generating profitable trading strategies.

2.2 Methodology

This study leverages the FinRL library to construct a stock trading environment and obtains historical stock market data from Yahoo Finance. Data preparation involves calculation of technical indicators and turbulence index. Five deep reinforcement learning agents (A2C, DDPG, PPO, TD3, SAC) are implemented using the Stable Baselines3 library, employing default hyperparameter settings for initial exploration. The agents interact with the environment, taking actions based on market features and receiving rewards that reflect their trading decisions. Key financial metrics such as Sharpe ratio, maximum drawdown, and risk-adjusted returns are utilized for evaluation. The DRL agents' performance is benchmarked against the DJIA index, and backtesting on out-of-sample data validates the strategies' robustness.

3 Literature Review

This part of the thesis provides a foundational exploration of key concepts essential for understanding the application of deep reinforcement learning (DRL) in algorithmic trading. It begins by outlining the principles of financial trading and the core components of reinforcement learning, including Markov Decision Processes and Q-learning. Subsequently, the review delves into Deep Q-Networks (DQN), a seminal breakthrough in DRL, and examines various policy gradient methods. This comprehensive groundwork sets the stage for understanding the specific challenges of the financial trading domain and how the DRL algorithms investigated in this thesis address them.

3.1 Financial Trading

This section of thesis introduces the concept of algorithmic trading, in which the execution of trades is automated through rule-based systems. It underscores the prominence of technical analysis within this domain, where traders utilize historical price data and statistical indicators to identify patterns and inform their strategies. Key technical indicators such as the Moving Average Convergence Divergence (MACD) and Relative Strength Index (RSI), due to their significance in the agent's decision-making, will be defined. This establishes the groundwork for understanding how traditional algorithmic approaches function and provides context for the motivation behind exploring machine learning-driven techniques.

3.1.1 Financial trading

Financial trading means buying and selling assets like stocks, bonds, currencies, or commodities to profit from short-term price changes. Unlike investing, which focuses on long-term gains, trading aims to capitalize on quick market movements. Day trading (buying and selling within the day), scalping (small, frequent trades), and swing trading (holding for a few days or weeks) are all examples of financial trading strategies. (Dodd, 2020)

3.1.2 Algorthmic trading

Algorithmic trading involves using computer programs that follow specific rules to make trading decisions, place orders, and even manage the trades after they've been executed. These programs are designed to analyse market data, identify potential opportunities, and act according to pre-programmed instructions. The goal of algorithmic trading is to remove human emotion from the process, increase the speed of execution, and potentially gain an edge in the ever-changing markets. (Johnson, 2010)

3.1.3 The Moving Average Convergence and Divergence (MACD)

The Moving Average Convergence/Divergence (MACD) is a technical analysis tool used to identify trend changes, assess trend strength, and spot potential overbought/oversold conditions. It works by analysing the relationship between two different exponential moving averages (EMAs).

MACD line is calculated as:

$$MACD\ line = EMA_{12} - EMA_{26} \tag{1}$$

The signal line is then the exponential moving average (EMA) of the MACD line:

$$Signal\ line = EMA_{9}\ (MACD\ line) \tag{2}$$

Where *EMA* is exponential moving average and subscript indicates number of past days.

Traders often look for crossovers between the MACD line and signal line. A bullish crossover (MACD above signal) may indicate a buy signal, while a bearish crossover (MACD below signal) may indicate a sell signal. (Dolan, 2024)

3.1.4 Relative strength index (RSI)

The Relative Strength Index (RSI) is a momentum oscillator used in technical analysis. Developed by J. Welles Wilder Jr., it measures the speed and magnitude of a security's recent price changes to assess overbought or oversold conditions. RSI values range from 0 to 100. (Fernando, 2024)

Relative strength index is calculated as:

$$RSI = 100 - \left[\frac{100}{1 + RS}\right] \tag{3}$$

Where RS is relative strength factor. (Fernando, 2024)

3.1.5 Limitations of pre-defined rule based algorithmic trading.

Pre-defined rule-based algorithmic trading systems are built upon a static set of instructions. This makes them inflexible when faced with changing market conditions or unexpected events like sudden volatility or news-driven price swings. Their inability to adapt can lead to missed opportunities or significant losses, highlighting the need for constant monitoring and refinement to remain effective. (How Is Machine Learning Used in Trading?, 2022)

As market dynamics evolve or new rules are added to address shortcomings, rule-based systems often become increasingly complex. This complexity makes optimization a significant challenge, as even small changes can have unintended consequences across the interconnected rules. There's also the risk of overfitting, where the system becomes overly tailored to historical data and performs poorly in real-time trading environments as it fails to generalize well to new market situations.

The limitations of pre-defined rule-based algorithmic trading, such as their lack of adaptability to changing market conditions, their complexity, and the risk of overfitting, underscore the potential advantages of reinforcement learning (RL) approaches. RL focuses on agents that learn and improve through interaction with their environment, allowing them to identify patterns, adapt to shifting dynamics, and make decisions that weren't explicitly programmed. This adaptability, along with RL's ability to handle complexity and mitigate overfitting, make it a promising avenue for developing more robust and flexible trading strategies in the complex financial landscape.

3.2 Reinforcement Learning

This section of thesis introduces the concepts of Reinforcement Learning. We will discuss the Markov property, Markov decision processes (MDPs), value functions, the Bellman equation, and Q-learning. These concepts lay the groundwork for understanding more sophisticated RL algorithms, such as the Deep Q-Network (DQN) employed later in this work.

Reinforcement learning is a branch of machine learning concerned with how intelligent agents should take actions within an environment to maximize a cumulative reward signal. Unlike supervised learning, RL agents do not receive explicit labelled examples of correct behaviour. Instead, the agent learns through trial and error, interacting with the environment and receiving feedback in the form of rewards or punishments.
(Sutton, Barto, 2018)

This learning process is rooted in the principle that actions yielding positive rewards should be reinforced, while those with negative consequences should be discouraged. Over time, the agent aims to discover an optimal policy, a mapping from states to actions that maximizes its expected long-term return. RL draws from principles of psychology, neuroscience, and optimal control, offering a framework for sequential decision-making in dynamic and uncertain environments. (Sutton, Barto, 2018)

3.2.1 Markov property

In the context of reinforcement learning, the Markov property signifies that all information necessary to predict future states and rewards is encapsulated in the current state of the environment. In other words, the history of past states and actions does not provide additional predictive power beyond what is contained within the present state. This property underpins many reinforcement learning algorithms, as it allows the agent to make decisions based solely on its current situation rather than requiring a complete record of past interactions. (Sutton, Barto, 2018)

Markov property can be formulated as:

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, S_2 \dots, S_t] \qquad (4)$$

Which states that probability of moving to next state S_{t+1} only depends on current state S_t and is independent on previous states before S_t.

3.2.2 Markov decision process

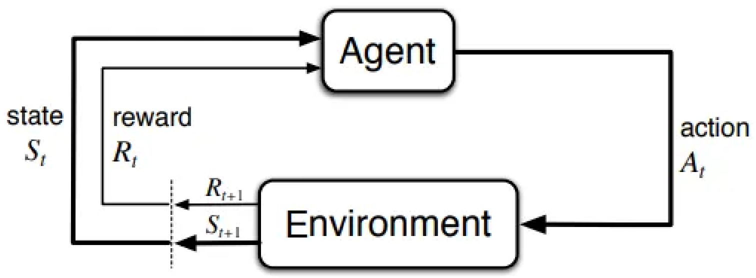
A Markov Decision Process (MDP) is a mathematical framework for modelling decision-making in dynamic systems where outcomes are partially random and partially controlled. It helps determine the optimal actions for an agent based on the current state of the system and potential rewards. MDPs are widely used in artificial intelligence. In probabilistic planning, they guide agents with known models to achieve goals, while in reinforcement learning (RL), they allow agents to learn from environmental feedback, even with uncertain outcomes. (Puterman, 1994)

MDP framework has following components:

- S: set of states.
- A: set of actions
- P: state transition probability matrix $P^{a}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
- γ: discount factor
- R: reward

The agent-environment interface in a Markov Decision Process (MDP) is characterized by a cyclical interaction. At each time step, the agent selects an action from the available set. This action triggers a state transition within the environment, and the agent receives a corresponding reward signal. The agent's subsequent decisions are informed solely by this new state and reward, upholding the Markov property. This interface forms the foundation upon which reinforcement learning algorithms seek to optimize the agent's decision-making policy.

Figure 1 Agent-Environment interface in a MDP



Source: www.towardsdatascience.com (Singh, 2019)

3.2.3 Return

In reinforcement learning, the "return" typically refers to the cumulative sum of rewards obtained by an agent over a sequence of time steps. It represents the overall measure of the agent's performance in an environment, considering both immediate and future rewards. The return is often denoted as G and is defined as the sum of rewards discounted by a factor γ over time:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (5)$$

Where G_t is return at time step t, R_{t+k+1} is reward obtained at time step $t + k + 1$ and γ is discount factor, determining the importance of future rewards.

The return is central to RL as it guides the agent's learning process. The agent's objective is to find a policy that maximizes the expected value of the return. (Lee, 2005)

3.2.4 Policy

In reinforcement learning, a policy is a strategy or a set of rules that an agent employs to make decisions in an environment. It defines the mapping from states to actions, specifying the agent's behaviour at each point in its interaction with the environment. Policy is denoted by π and can be deterministic or stochastic.

- Deterministic policy directly maps each state to a specific action. $\pi : S \rightarrow A$
- Stochastic policy specifies a probability distribution over actions for each state. In other words, it describes the likelihood of taking each possible action in a given state. $\pi : S \times A \rightarrow [0,1]$

The central goal of many RL algorithms is to find an optimal policy (π*) that maximizes the expected cumulative reward the agent receives from the environment. Policies can be learned:

- Implicitly (value-based methods), where the agent learns value functions and then derives actions that lead to more favourable states.
- Explicitly (Policy-Based Methods), where the policy function itself is directly modelled and optimized. (Carr, 2023)

3.2.5 Value functions

Value functions in reinforcement learning serve as powerful tools for predicting how much future reward an agent can expect to receive by being in a specific state or taking a particular action within that state. They form a core concept by allowing the agent to evaluate the long-term consequences of its decisions. By learning to identify states (or state-action pairs) with higher associated values, agents can develop optimal strategies to maximize their cumulative rewards. (Szepesvári, 2010)

State-Value Function (V-function)

The state-value function, denoted as V(s), predicts the total discounted reward an agent can expect to accumulate starting from a state 's' and then subsequently following a given policy. Mathematically, it's the expected sum of future rewards, with those further in the future discounted using a factor (gamma) to prioritize immediate gains.
(Bertsekas, Tsitsiklis, 1996)

$$V_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_{t+1} |\, S_0 = s] \qquad (6)$$

Where $V(s)$ is the state value for state s, $\mathbb{E}_\pi$ denotes the expectation under a policy π, γ is the discount factor, R_{t+1} is the reward obtained at time step t+1, and $S_0 = s$ indicates that the agent starts in state *s*.

Action-Value Function (Q-function)

The action-value function, denoted as Q(s,a), focuses on the value of taking a specific action 'a' while in state 's', and then continuing according to the policy. Similar to the state-value function, it predicts the expected cumulative discounted reward. This function allows the agent to evaluate the potential outcomes of each action in a given state.
(Bertsekas, Tsitsiklis, 1996)

$$Q_\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_{t+1} |\, S_0 = s\, , A_0 = a\,] \qquad (7)$$

Here, $Q(s, a)$ is the action value for state-action pair $(s,\ a)$, $\mathbb{E}_\pi$ denotes the expectation under a policy π, γ is the discount factor, R_{t+1} is the reward obtained at time step $t + 1$, $S_0 = s$ and $A_0 = a$ indicates initial state and action.

3.2.6 Bellman equation

The Bellman equation, named after Richard E. Bellman, is a fundamental equation in dynamic programming and reinforcement learning. It expresses the value of a particular state (or state-action pair) in terms of the immediate reward received and the discounted value of successor states. The Bellman equation can be written for state-value functions $V(s)$ or action-value functions $Q(s, a)$. The Bellman equation provides a recursive structure for calculating value functions. Iterative methods based on this equation form the basis of many dynamic programming and reinforcement learning algorithms. (Bellman, 1957)

Bellman expectation equation of the state-value and action-value functions are represented as:

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s] \tag{8}$$

$$Q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s,\ A_t = a] \tag{9}$$

Where:

- $V_\pi(s)$ is the value of state, representing the expected cumulative future rewards following policy π.
- $Q_\pi(s, a)$ is the action-value function, representing the expected cumulative future rewards for taking action a in state s following policy π.
- R_{t+1} is the immediate reward obtained after transitioning from state s to the next state S_{t+1}.
- S_{t+1} is the next state reached after taking action a in state s.
- A_{t+1} is the action chosen in the next state.
- γ is discount factor, emphasizing the importance of future rewards.

These equations capture the recursive relationships between the current value (or action-value), the immediate reward, and the discounted expected value (or action-value) of the next state. These equations serve as fundamental building blocks for devising reinforcement learning algorithms such as Q-learning and DQN. (Singh, 2019)

3.2.7 Q-learning

Q-learning is a core reinforcement learning algorithm designed to find the optimal action-selection strategy within a Markov Decision Process (MDP). It focuses on learning the Q-function, $Q(s, a)$, representing the expected long-term reward of taking action a in state s and then acting optimally from there on. Q-learning is model-free, meaning it doesn't need a model of the environment beforehand, making it widely applicable. (Lapan, 2020)

The algorithm learns through experience. Using temporal difference learning, it iteratively updates its Q-values based on the rewards received and the estimated values of the next states encountered. Q-learning is off-policy, allowing it to improve its estimates of the optimal policy even while following a different behaviour. Through repeated updates, Q-values gradually become consistent with the Bellman equation, guiding the agent to select actions that eventually maximize its total reward. (Lapan, 2020)

Bellman update equation in Q-learning:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma \max_a' Q(s',a') - Q(s,a)] \tag{10}$$

Where: $Q(s,a)$ is the current estimated value for state s and action a. α is the learning rate, a value between 0 and 1, controls how much the existing Q-value is adjusted with each update. $R(s,a)$ represents immediate reward received for taking action a in state s. γ is discount factor, value between 0 and 1, balances the importance of immediate rewards vs. the potential for future rewards. Higher gamma value places more emphasis on long-term gains, while a lower gamma prioritizes immediate rewards. $\max_a' Q(s',a')$ represents the estimated maximum future reward achievable from the next state s' by taking any possible action a'. (Bellman, 1957)

Temporal difference

Temporal difference (TD) learning is a central concept in reinforcement learning. It differs from traditional dynamic programming methods by not requiring a full model of the environment and differs from Monte Carlo methods by not needing to wait until the end of an episode to learn. Instead, TD methods learn by updating their estimates of value functions based on other estimates within the process. (Dutta, 2018)

Temporal difference is difference between the current estimate of a value and a slightly improved estimate obtained from the immediate reward and the estimated value of the next state. TD learning algorithms use this difference to drive updates, gradually adjusting the current estimates towards more accurate values reflecting the long-term rewards the agent can expect.

Notice that the part of bellman update equation $R(s,a) + \gamma \max_a' Q(s',a') - Q(s,a)]$ represents temporal difference, where $R(s,a) + \gamma \max_a' Q(s',a')$ is improved estimate and $Q(s,a)$ is the current estimate.

Exploration vs. Exploitation

A core challenge in reinforcement learning is the trade-off between exploration and exploitation. Exploitation refers to the agent leveraging its current knowledge to select the actions that it believes will yield the highest rewards based on past experience. Exploration, on the other hand, involves trying new actions that may seem suboptimal in the short term but could potentially lead to the discovery of better long-term strategies.

In most RL problems, the agent begins without a perfect understanding of the environment or how its actions lead to rewards. Exploration is crucial for discovering higher-rewarding states and actions. f the environment's reward structure changes over time, an agent that solely engages in exploitation might get stuck in a suboptimal strategy. Continued exploration allows it to adapt. (Russell, Norvig, 2020)

Epsilon-greedy strategy

The epsilon-greedy strategy is a fundamental exploration-exploitation technique in reinforcement learning. It guides an agent's action selection by balancing between exploration of unknown options and exploitation of known optimal choices. With probability epsilon ϵ, the agent explores a random action, allowing it to discover potentially better alternatives. On the other hand, with probability 1-ϵ, the agent exploits the current best-known action based on its learned Q-values. This approach ensures a trade-off between trying new possibilities to improve the agent's understanding of the environment and leveraging established knowledge to maximize cumulative rewards, making epsilon-greedy a versatile and widely used strategy in reinforcement learning algorithms. Value of epsilon parameter is between 0 and 1 and gradually decreases over time to favour exploitation as learning progresses. (Ravichandiran, 2018)

Hyperparameters in Q-learning

Hyperparameters play a crucial role in Q-learning, influencing the algorithm's behaviour, convergence, and overall performance. The learning rate α determines the step size during updates, affecting how much the current Q-values are adjusted based on new information. A higher α gives more weight to recent experiences, potentially leading to faster adaptation but risking instability. The discount factor γ balances immediate rewards against future gains, impacting the trade-off between short-term and long-term decision-making. Selecting an appropriate ϵ for the epsilon-greedy strategy is vital, as it dictates the agent's exploration-exploitation behaviour. Careful tuning of these hyperparameters is necessary to ensure Q-learning converges effectively and generalizes well across various environments. (Habib, 2019)

Practical Limitations

Traditional Q-learning relies on storing Q-values in a table, mapping each possible state-action pair to its expected reward value. However, this approach becomes infeasible in environments with large or continuous state spaces, like those involving image-based input or complex control tasks. The number of entries in the Q-table explodes, hindering learning and preventing generalization to unseen but similar states.

Deep Q-Networks (DQN) address this limitation by replacing the Q-table with a neural network. This neural network serves as a powerful function approximator, learning to estimate Q-values directly from the state input. This allows DQN to handle the high-dimensional and complex state spaces common in real-world applications. Since neural networks can learn patterns, DQN can generalize knowledge across similar states, leading to more efficient learning and better performance in unseen scenarios. (Kvartalnyi, 2023)

3.3 Deep Q-network (DQN)

Traditional Q-learning, while a foundational reinforcement learning algorithm, encounters limitations when dealing with complex environments characterized by large or continuous state spaces. The tabular approach to storing Q-values becomes infeasible, hindering learning and generalization. Deep Q-Networks (DQN) elegantly address this challenge by replacing the Q-value table with a powerful deep neural network. This network learns to approximate the Q-function, estimating the expected future rewards of taking actions directly from the state representation. (Kvartalnyi, 2023)

This section will delve into the components and techniques that underpin DQN's success. We will explore the architecture of the neural network Q-value approximator, the innovative experience replay mechanism, and the use of a target network to promote training stability. Furthermore, we'll examine optimization considerations, loss functions, and the role of activation and regularization techniques within the network. These elements collectively enable DQN to achieve state-of-the-art performance in domains where traditional Q-learning falters.

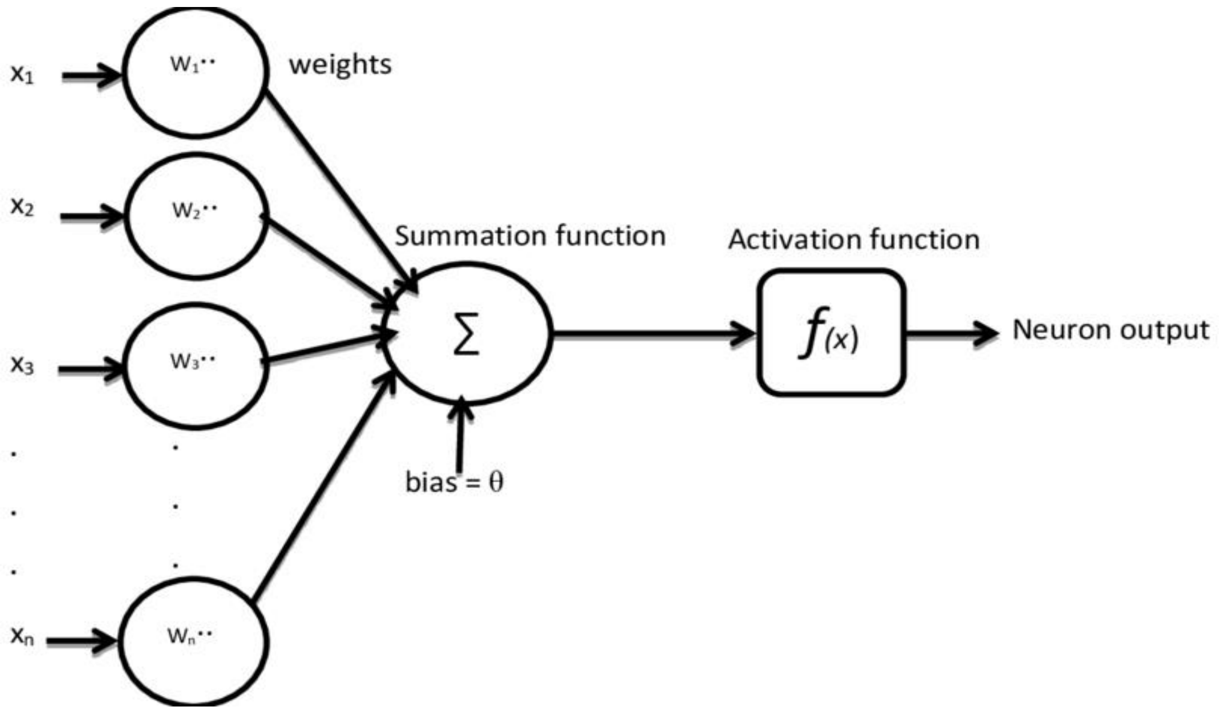
3.3.1 Neural networks

Neural networks (NNs) are a cornerstone of deep learning, inspired by the structure and function of the human brain. They consist of interconnected layers of artificial neurons, which process information and learn from data. A typical NN has an input layer that receives data, hidden layers that perform computations, and an output layer that produces results. Each layer contains multiple artificial neurons, connected by weighted links. NNs learn by adjusting the weights of these connections based on the data they are trained on. This process, called backpropagation, iteratively minimizes the difference between the network's predictions and the desired outputs. (Hardesty, 2017)

Artificial neuron

A neuron, the basic building block of a neural network, processes information through a series of steps. It receives multiple inputs, each associated with a weight that determines its importance. The neuron calculates a weighted sum of its inputs, adds a bias term, and then applies a non-linear activation function to this sum. The result of the activation function becomes the neuron's output, influencing neurons in subsequent layers or forming part of the network's final prediction. (Tucci, 2013)

Figure 2 Structure of the artificial neuron

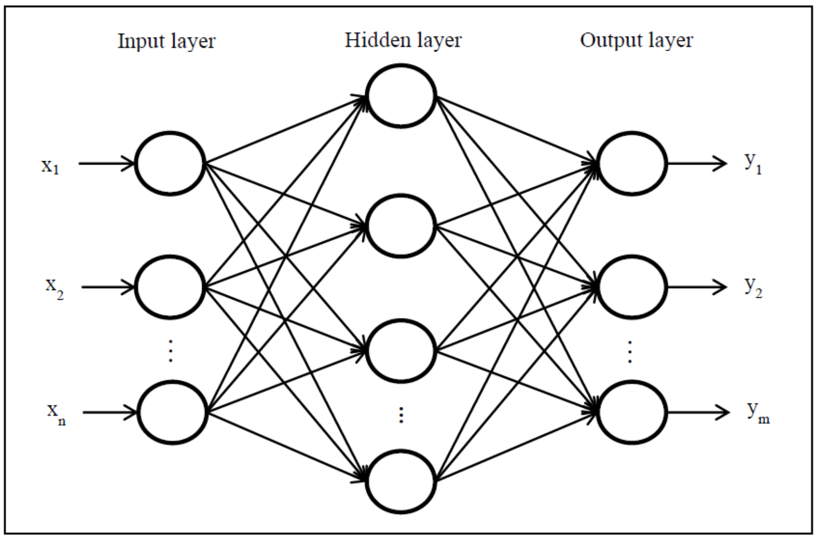


Source: www.researchgate.net

Feedforward Neural Network FNN

A feedforward neural network is a type of artificial neural network where information flows strictly in one direction, from the input layer, through hidden layers where computations occur, and finally to the output layer. Neurons in a feedforward network do not form loops or connections back to earlier layers. Feedforward neural networks learn through a process called backpropagation. During training, errors between the network's outputs and the desired targets are calculated. Backpropagation systematically determines how much each weight in the network contributed to those errors and adjusts the weights accordingly, with the goal of minimizing future errors. This iterative adjustment process allows the network to gradually learn complex patterns and relationships within the data. (Goodfellow et al., 2016)

Figure 3 Structure of Feedforward neural network



Source: www.researchgate.net

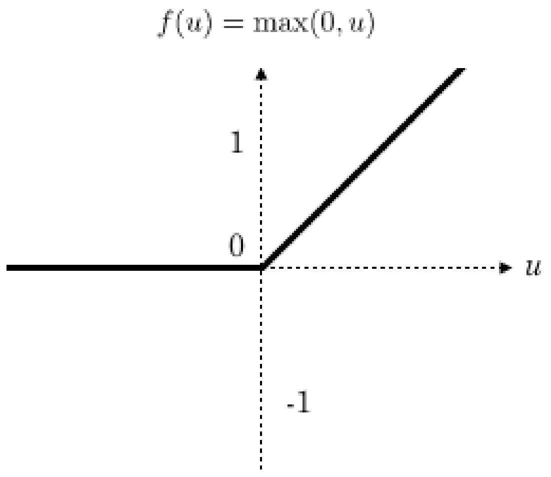
3.3.2 Activation function

An activation function is a non-linear function applied to the weighted sum of a neuron's inputs (plus its bias) within a neural network. It determines the neuron's output, influencing the signal passed to the next layer. Activation functions are crucial because they introduce non-linearity into neural networks, enabling them to learn complex, real-world relationships that aren't simple linear combinations of their inputs. (Baheti, 2021)

ReLU (Rectified Linear Unit)

The ReLU activation function is a popular choice in neural networks due to its computational efficiency and effectiveness in mitigating the vanishing gradient problem. It's defined as $f(x) = \max(0, x)$, meaning it outputs the input directly if it's positive and zero otherwise. ReLU's simplicity speeds up computations compared to functions like sigmoid and tanh. Additionally, its non-saturating gradient for positive inputs helps preserve learning signals across deep network layers. (Brownlee, 2020)

Figure 4 ReLU Activation function



Source: www.researchgate.net

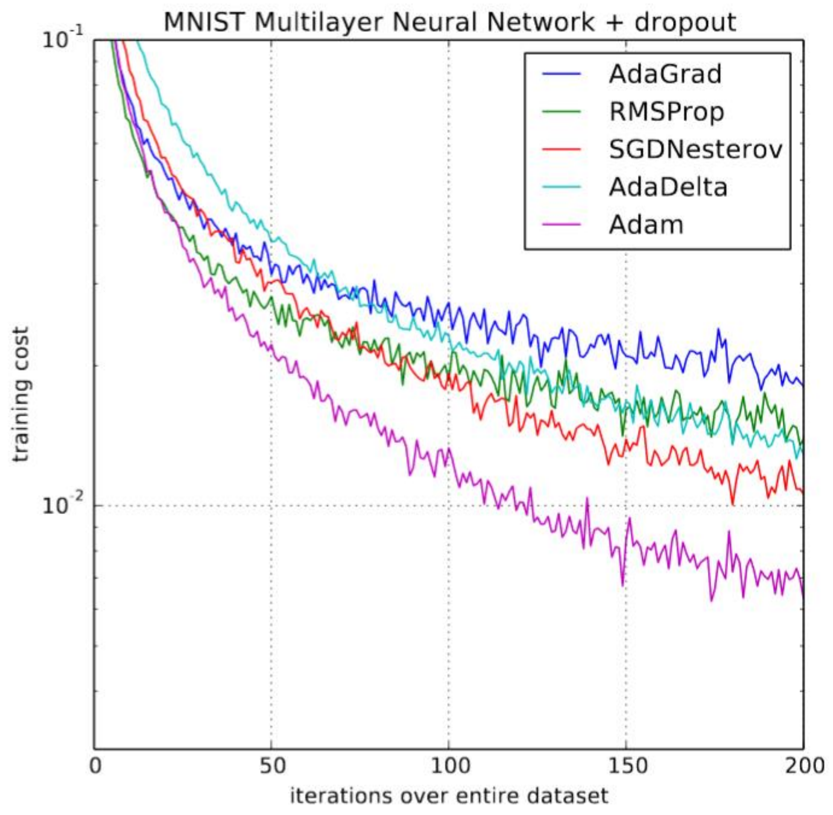
3.3.3 Optimization Algorithm

Optimization algorithms are the engines driving the learning process in neural networks. They guide how the network's weights (and sometimes biases) are updated to minimize a loss function. This loss function quantifies the error between the network's predictions and the true target values. The optimization algorithm's goal is to find the weights that produce the best possible performance on the task. Common optimization strategies include gradient descent (with variations like stochastic or mini-batch gradient descent), momentum-based methods that enhance convergence, and algorithms like AdaGrad, RMSprop, and Adam that utilize adaptive learning rates for individual parameters. (Walia, 2021)

Adam Optimizer (Adaptive Moment Estimation)

Adam is an optimization algorithm widely used in deep learning. It combines aspects of momentum optimization and RMSprop to adaptively adjust learning rates during training. Adam maintains two moving averages, the first moment (mean) and the second moment (uncentered variance) of the gradients, providing adaptive learning rates for each parameter. This adaptability allows Adam to handle various types of data and optimize neural network models efficiently. The algorithm has become a popular choice for its robust performance, effective handling of sparse gradients, and the ability to converge quickly during training, making it well-suited for tasks like training Deep Q-Networks (DQN) in reinforcement learning. (Walia, 2021)

Figure 5 Comparison of optimizers training on MNIST images.



Source: www.fast.ai

3.3.4 Experience Replay

Experience Replay is a reinforcement learning technique where the agent's past experiences (state, action, reward, next state) are stored in a memory buffer. During training, the algorithm randomly samples batches from this buffer instead of relying solely on the most recent experience. This breaks the correlation between consecutive samples, allows for the reuse of valuable experiences, and generally leads to more efficient learning and greater stability in the training process. (Lapan, 2020)

3.3.5 Target Network

A target network in the context of deep reinforcement learning refers to a separate neural network used to stabilize and improve the training process. Typically associated with algorithms like DQN, the target network is a copy of the primary network, often referred to as the Q-network, but its parameters are updated more slowly. This delayed updating helps mitigate the issues of training instability and divergence that can arise when using a single network to estimate both current and target values. By periodically updating the target network's parameters, the learning process becomes more robust, as it introduces a smoother and more consistent set of target values for the agent to optimize towards during the reinforcement learning training. This concept aids in achieving a more reliable convergence and improved training efficiency, contributing to the overall stability and effectiveness of the reinforcement learning algorithm. (Sutton, Barto, 2018)

3.3.6 Practical Limitations

Critic-only methods like DQN, while powerful, have limitations. They struggle with continuous action spaces where actions aren't discrete choices. Additionally, they cannot directly learn stochastic policies where the agent outputs a probability distribution over actions.

This motivated the evolution towards actor-only methods. These excel at continuous actions and stochastic policies but can suffer from high variance and slow convergence. Actor-critic methods emerged as a balance, combining the strengths of both. The critic provides guidance for policy improvement, reducing variance, while the actor enables the flexibility necessary for complex environments.

3.4 Policy Gradient Method

Policy gradient methods form a powerful class of reinforcement learning algorithms that directly optimize the policy, the agent's decision-making strategy. Unlike value-based methods that first learn a value function, policy gradient methods directly update the policy parameters to maximize expected rewards. They are particularly well-suited for problems with continuous action spaces or when a stochastic policy (a distribution over actions) is desirable. (Goodfellow et al., 2016)

At their heart, policy gradient algorithms seek to increase the probability of actions that lead to high rewards and decrease the probability of those that don't. This is often achieved through a gradient ascent procedure, where the policy parameters (often denoted by θ) are updated in a direction that improves the expected long-term return. A common policy gradient update rule takes the form:

$$\theta(t+1) = \theta(t) + \alpha\nabla[J(\theta)] \tag{11}$$

Where α is learning rate, J(θ) is an objective function measuring expected return, and ∇[J(θ)] is the gradient of the objective function with respect to the policy parameters.

Policy gradient methods offer advantages such as their ability to handle continuous action spaces effectively and their potential to learn stochastic policies. Additionally, they have the potential to converge to better local optima in some problems compared to value-based methods. However, a notable drawback is that they can suffer from high variance in the gradient estimates, which can lead to instability in the learning process. Furthermore, their convergence might be slower compared to value-based methods in certain scenarios.

3.4.1 PPO (Proximal Policy Optimization)

PPO is an influential policy gradient algorithm designed to improve upon the stability and sample efficiency of traditional policy gradient methods. To understand PPO, let's denote the ratio between the new policy and the old policy as r(θ). PPO's clipped surrogate objective can then be expressed as:

$$L^{CLIP}(\theta) = \widehat{E_t}\left[min\left(r_t(\theta)\widehat{A_t}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A_t}\right)\right] \tag{12}$$

Where $\widehat{E_t}$ is the empirical average over time, $\widehat{A_t}$ is an advantage estimator, and ϵ is a clipping parameter. (Lapan, 2020)

This objective function encourages updates when the ratio r(θ) leads to improvement but clips the objective to prevent excessively large policy changes.

The techniques employed by PPO have significantly advanced the applicability of policy gradient methods in complex domains, including trading, due to increased stability and robustness.

3.5 Actor-critic methods

Actor-critic methods offer a powerful paradigm within reinforcement learning, strategically blending the strengths of value-based and policy-based approaches. The "actor" component represents the agent's policy, directly responsible for selecting actions based on the current state of the environment. In contrast, the "critic" component learns a value function, which serves to estimate the expected future rewards from a given state or state-action pair. The critic's evaluation provides a more informed signal for updating the actor's policy, going beyond the simple success/failure feedback of pure policy gradient methods.

This combination offers several key advantages. The incorporation of a learned value function helps reduce the variance often present in policy gradient updates, leading to increased stability in the learning process. Additionally, actor-critic methods can improve sample efficiency, meaning they can learn effectively from less environmental interaction. Furthermore, many actor-critic architectures are well-equipped to handle continuous action spaces, making them directly applicable to the complexities of algorithmic trading, where precise order sizes might be required. (Karunakaran, 2020)

3.5.1 A2C (Advantage Actor-Critic)

A2C (Advantage Actor-Critic) is a seminal actor-critic algorithm known for its effectiveness and relative simplicity. A core concept in A2C is the utilization of the advantage function, which quantifies how much better an action performed, relative to expectations. (Simonini, 2022)
Mathematically, the advantage function can be expressed as:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{13}$$

Where $\mathrm{Q(s_t,\ a_t)}$ is the Q-value (expected future reward for taking action a_t in state s_t) and $V(s_t)$ is the state value (expected future reward from state s_t).

This calculated advantage serves to focus the learning process on surprisingly good or surprisingly bad action choices. A2C updates both its actor (policy) and critic (value function) components in tandem.

3.5.2 SAC (Soft Actor-Critic)

SAC (Soft Actor-Critic) builds upon the successes of A2C, specifically targeting continuous action space problems. A key innovation of SAC is the introduction of entropy regularization. By explicitly adding an entropy term to its objective function, SAC encourages a degree of exploration within its policy. This helps it avoid premature convergence to suboptimal strategies, which can be a risk in complex environments.
(Karunakaran, 2020)
Mathematically, the core objective of SAC can be described as:

$$J(\pi) = \sum_{t=1}^{T} E(s_t, a_t) \backsim \rho\pi \left[r(s_t, a_t) + \alpha H\big(\pi(.\,|s_t)\big) - Q(s_t, a_t) \right] \quad (14)$$

Where $H\big(\pi(.\,|s_t)\big)$ is the entropy of the policy and α is a temperature parameter controlling the importance of entropy.

Actor-critic methods, due to their balance of stability, sample efficiency, and performance, have become highly popular choices for a wide range of reinforcement learning problems. Their applicability to algorithmic trading makes them particularly compelling tools for developing intelligent trading strategies.

3.6 Continuous Control Methods

The world of algorithmic trading often goes beyond simple, discrete choices like buying, selling, or holding a stock. To effectively navigate this complex landscape, reinforcement learning offers a subset of algorithms specifically designed for problems where the action space is continuous. This means the agent must choose actions from a range of possible values, such as determining the exact order size, rather than a limited set of options. (Chow, 2021)

3.6.1 DDPG (Deep Deterministic Policy Gradient)

DDPG is a foundational algorithm in continuous control, extending the groundbreaking concepts of Deep Q-Networks (DQN) to handle continuous actions. It employs an actor-critic architecture, where the critic learns to estimate the long-term value of being in a certain state and taking an action, while the actor directly learns a policy to select these continuous actions. To enable stable learning within this complex setting, DDPG borrows crucial techniques from DQN and adds its own core contribution. (Lapan, 2020)

DDPG uses a few key techniques to enhance stability and efficiency within the learning process. Target networks, which are slowly updated versions of the actor and critic networks, provide a more stable target for value and policy updates. Experience replay involves storing past experiences in a buffer and sampling from it; this breaks up correlations within the data, which can make training less erratic. Finally, DDPG differs from standard policy gradient methods by using a deterministic policy, meaning it directly outputs a specific action instead of a probability distribution over actions.

3.6.2 TD3 (Twin Delayed DDPG)

TD3 (Twin Delayed DDPG) is an improved version of DDPG designed to be more stable and avoid overestimating values. Key changes include delaying the frequency of policy updates in comparison to critic updates, using two critic networks instead of one (taking the minimum of their outputs to reduce overestimation), and injecting a bit of controlled noise into target actions to encourage exploration and make policy updates smoother. (Santhosh, 2022)

The ability to model continuous outputs directly makes these methods powerful tools for algorithmic trading. Whether it's dynamically determining order sizes, optimizing bid-ask spreads, or managing complex portfolio allocations, continuous control methods allow for fine-grained decision-making that better reflects the nuances of financial markets.

4 Practical Part

Building upon the theoretical foundation, this chapter delves into the practical application of deep reinforcement learning for algorithmic trading. It describes the process of training multiple DRL agents within a FinRL environment, fed with historical data from Yahoo Finance. The chapter lays out the evaluation framework and anticipates how the performance metrics will illuminate the strengths, weaknesses, and potential promise of these DRL-driven trading strategies.

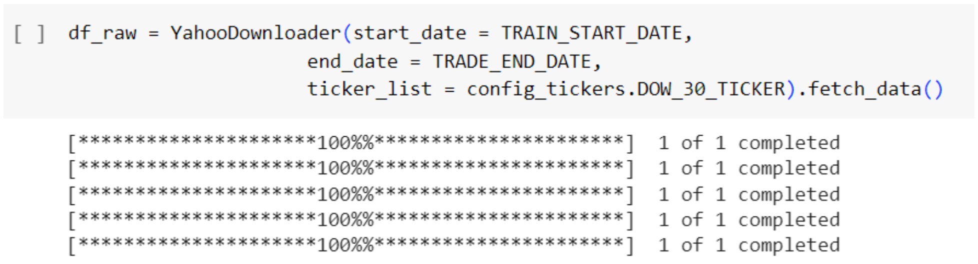
4.1 Data

This section outlines the fundamental processes of creating a suitable dataset for training and evaluating a reinforcement learning model in the context of financial trading. It describes the strategic acquisition of historical stock market data for the Dow Jones 30 constituents, covering essential OHLCV components. The importance of data preprocessing is emphasized, including the handling of potential inconsistencies and the use of technical indicators to generate valuable insights. Furthermore, the transformation of data to represent distinct states compatible with reinforcement learning principles is explained.

4.1.1 Data Acquisition and Preparation

This project aimed to apply reinforcement learning to financial trading, and reliable data formed the foundation. The FinRL library's YahooDownloader class was used to acquire historical financial data from the Yahoo Finance API. The focus was on the Dow Jones 30 constituents, providing a diverse representation of the stock market. Daily OHLCV data (open, high, low, close, volume) was chosen because it encapsulates essential price and volume dynamics. The dataset was divided into training (2010-01-01 to 2020-01-01) and testing (2020-01-02 to 2023-01-01) sets, ensuring the model would be trained on historical data and evaluated on unseen data for a realistic assessment of performance.

Figure 6 Data fetch from Yahoo finance



```
[ ]  df_raw = YahooDownloader(start_date = TRAIN_START_DATE,
                             end_date = TRADE_END_DATE,
                             ticker_list = config_tickers.DOW_30_TICKER).fetch_data()

     [*********************100%%**********************]  1 of 1 completed
     [*********************100%%**********************]  1 of 1 completed
     [*********************100%%**********************]  1 of 1 completed
     [*********************100%%**********************]  1 of 1 completed
     [*********************100%%**********************]  1 of 1 completed
```

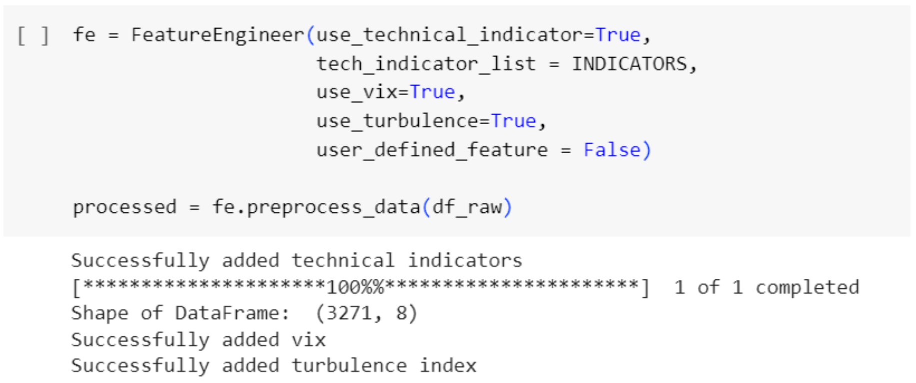
Source: Own processing.

4.1.2 Data Preprocessing

Raw financial data often contains inconsistencies such as missing values. There were no missing values in our fetched data.

Next, feature engineering was performed to augment the raw data with insights and patterns relevant to trading decisions. Technical indicators provide valuable signals for market analysis. The FinRL library was leveraged to calculate trend-following indicators like MACD and RSI, aiding in identifying potential buy or sell opportunities based on momentum shifts. Furthermore, FinRL's turbulence index calculation was incorporated, enabling the model to measure extreme price fluctuations in the market and adjust its behaviour in response to differing volatility conditions.

Figure 7 Date feature engineering



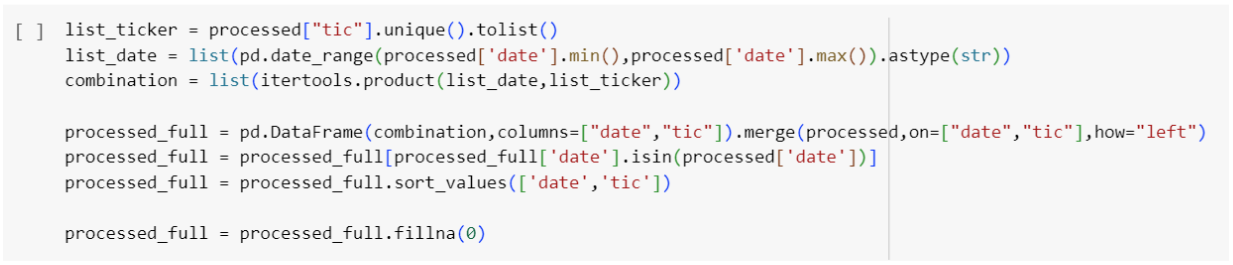
```
[ ]  fe = FeatureEngineer(use_technical_indicator=True,
                          tech_indicator_list = INDICATORS,
                          use_vix=True,
                          use_turbulence=True,
                          user_defined_feature = False)

     processed = fe.preprocess_data(df_raw)

     Successfully added technical indicators
     [*********************100%%**********************]  1 of 1 completed
     Shape of DataFrame:  (3271, 8)
     Successfully added vix
     Successfully added turbulence index
```

Source: Own processing.

4.1.3 Data Transformation

Reinforcement learning models require data to be structured in a way that defines distinct states upon which the agent will make decisions. To achieve this, the pre-processed data was organized such that each row represented a unique state, combining a specific date, the stock ticker, and relevant features. This transformation ensured compatibility with the reinforcement learning environment.

Figure 8 Data transformation for state space



```
[ ]  list_ticker = processed["tic"].unique().tolist()
     list_date = list(pd.date_range(processed['date'].min(),processed['date'].max()).astype(str))
     combination = list(itertools.product(list_date,list_ticker))

     processed_full = pd.DataFrame(combination,columns=["date","tic"]).merge(processed,on=["date","tic"],how="left")
     processed_full = processed_full[processed_full['date'].isin(processed['date'])]
     processed_full = processed_full.sort_values(['date','tic'])

     processed_full = processed_full.fillna(0)
```

Source: Own processing.

4.1.4 Dataset Split and Storage

Dividing the dataset into distinct training and testing sets is a standard practice in machine learning. The data_split function facilitated this, using the predefined date ranges. The segregated datasets were saved as CSV files for later use during model training and evaluation.

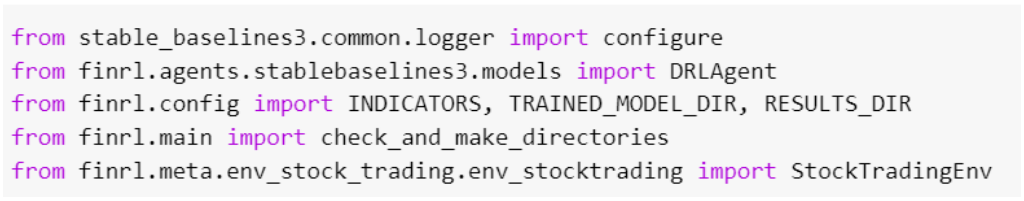
4.2 Train

This section explores the process of training the reinforcement learning trading agent. It begins by outlining the setup of necessary tools for model development. Next, it describes the transformation of financial data into a reinforcement learning-compatible market environment, emphasizing how this environment guides the agent's choices. The section then highlights the diverse deep reinforcement learning agents used for training, explaining the core principles behind their distinct approaches to learning and optimizing trading strategies. These agents leverage methods such as policy gradients, actor-critic techniques, and strategies suited for continuous control or improved exploration.

4.2.1 Package Installation

Key packages were installed to facilitate the training of a reinforcement learning trading agent. The FinRL library provided core components for constructing a trading environment and defining reinforcement learning models. Additionally, the Stable Baselines 3 library was crucial, offering implementations of several popular deep reinforcement learning (DRL) algorithms.

Figure 9 Libraries and imports for DRL agent to train



```python
from stable_baselines3.common.logger import configure
from finrl.agents.stablebaselines3.models import DRLAgent
from finrl.config import INDICATORS, TRAINED_MODEL_DIR, RESULTS_DIR
from finrl.main import check_and_make_directories
from finrl.meta.env_stock_trading.env_stocktrading import StockTradingEnv
```

Source: Own processing.

4.2.2 Market Environment Creation

To apply reinforcement learning, it was necessary to structure the financial data into an environment adhering to the OpenAI Gym standard. Let's break down the key elements:

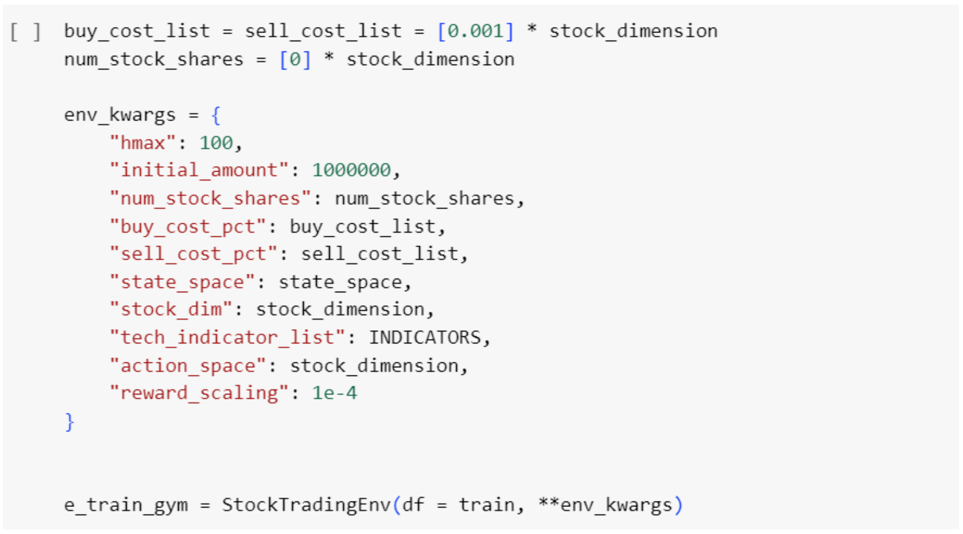
State (s): The state represents the agent's current view of the market, encapsulating historical price data, technical indicators, and other relevant factors calculated from the training dataset. This state definition enables the agent to make informed decisions based on observed market conditions.

Action (a): The action space comprises the actions the agent can take (e.g., buy, sell, hold). When an action operates multiple shares, $a \in \{-k, ..., -1, 0, 1, ..., k\}$, e.g. "Buy 10 shares of AAPL" or "Sell 10 shares of AAPL" are 10 or -10, respectively.

Reward Function ($R(s, a, s')$): The reward function is critical in guiding the agent's learning. It provides feedback on the outcomes of actions taken in a given state. Here, the reward is designed around changes in portfolio value, encouraging the agent to develop profitable trading strategies.

The pre-processed training data (train_data.csv) was loaded, and essential parameters were calculated for the environment's construction. The $StockTradingEnv$ class within the FinRL library handled the creation of this environment, incorporating transaction costs, allowed actions, state and reward definitions, and more.

Figure 10 Creation of stock trading environment for DRL agents



```
[ ] buy_cost_list = sell_cost_list = [0.001] * stock_dimension
    num_stock_shares = [0] * stock_dimension

    env_kwargs = {
        "hmax": 100,
        "initial_amount": 1000000,
        "num_stock_shares": num_stock_shares,
        "buy_cost_pct": buy_cost_list,
        "sell_cost_pct": sell_cost_list,
        "state_space": state_space,
        "stock_dim": stock_dimension,
        "tech_indicator_list": INDICATORS,
        "action_space": stock_dimension,
        "reward_scaling": 1e-4
    }


    e_train_gym = StockTradingEnv(df = train, **env_kwargs)
```

Source: Own processing.

4.2.3 DRL Agent Training

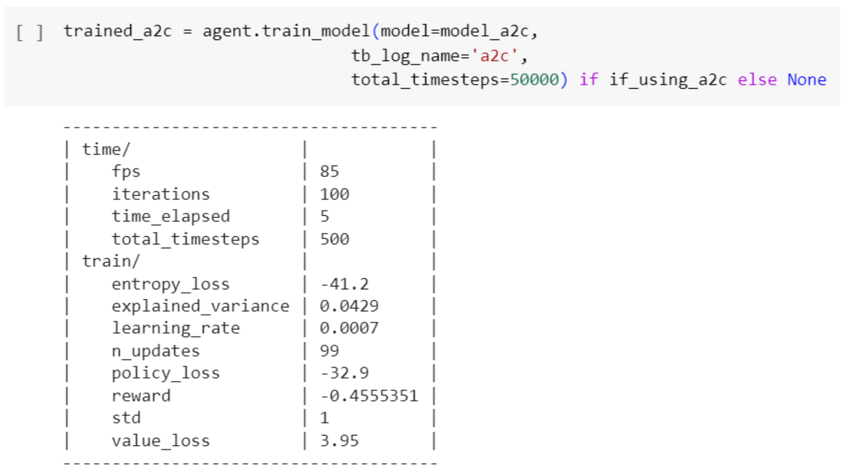
Stable Baselines 3 library provided implementations of several advanced DRL algorithms. A $DRLAgent$ object was created to manage the training and interaction with the trading environment. The following algorithms were selected for training:

- A2C (Advantage Actor-Critic): An on-policy algorithm that combines policy gradients and value function estimation for improved stability.
- DDPG (Deep Deterministic Policy Gradient): Suited for continuous action spaces, useful when fine-grained control of buying/selling quantities is required.
- PPO (Proximal Policy Optimization): An on-policy algorithm that ensures policy updates remain within a controlled range to avoid drastic changes in behavior.
- TD3 (Twin Delayed DDPG): Enhances DDPG with techniques to mitigate overestimation bias in value functions.

- SAC (Soft Actor-Critic): An off-policy algorithm that emphasizes entropy for greater exploration, aiding in discovering diverse strategies.

The training process involved each algorithm interacting repeatedly with the market environment. Hyperparameters, such as learning rate and network size used their default values within the library's implementation. Logging was configured to measure performance metrics during training. Finally, the trained models were saved for subsequent use in backtesting and analysis.

Figure 11 Training process of DRL agent



```
[ ]  trained_a2c = agent.train_model(model=model_a2c,
                                 tb_log_name='a2c',
                                 total_timesteps=50000) if if_using_a2c else None
```

```
--------------------------------------
| time/                 |            |
|    fps                | 85         |
|    iterations         | 100        |
|    time_elapsed       | 5          |
|    total_timesteps    | 500        |
| train/                |            |
|    entropy_loss       | -41.2      |
|    explained_variance | 0.0429     |
|    learning_rate      | 0.0007     |
|    n_updates          | 99         |
|    policy_loss        | -32.9      |
|    reward             | -0.4555351 |
|    std                | 1          |
|    value_loss         | 3.95       |
--------------------------------------
```

Source: Own processing.

4.3 Agent Evaluation

This section focuses on evaluating the performance of the trained reinforcement learning agents within a simulated market environment. This backtesting process is essential to understand how well the agents have learned to make profitable trading decisions and how they might perform in a real-world scenario. By observing the agents' actions within the simulation, we can track their account values over time, gaining valuable insights into their performance.

4.3.1 DJIA Index

The Dow Jones Industrial Average (DJIA) is a stock market index that measures the performance of 30 large, publicly traded companies listed on U.S. stock exchanges. Calculated since 1896, it provides a snapshot of the overall health of the U.S. stock market, influencing investment decisions worldwide. The Dow Jones Industrial Average (DJIA) index was incorporated as an additional benchmark to gauge the performance of the reinforcement learning models. Historical DJIA price data was fetched for the testing period.

4.3.2 Metrics for Evaluating Performance

Metrics for Evaluating Performance refer to quantifiable measures used to assess the effectiveness and efficiency of a system, process, or entity. These metrics provide objective insights into key aspects, such as financial returns, risk, and operational efficiency. Common examples include Sharpe Ratio for risk-adjusted returns, Annual Return for overall performance, and Max Drawdown for measuring downside risk. Effective performance metrics enable informed decision-making, support goal alignment, and help stakeholders evaluate success or identify areas for improvement in various domains, including finance, business, and technology. (Groette, 2023)

- **Sharpe ratio:** The Sharpe Ratio measures the risk-adjusted performance of an investment by assessing the excess return per unit of risk. A higher ratio indicates better risk-adjusted returns.

$$S_T = \frac{mean(R_t) - r_f}{std(R_t)} \tag{15}$$

  Where $R_t = \frac{v_t - v_{t-1}}{v_{t-1}}$, r_f is risk-free rate and $t = 1, .. T$.

- **Cumulative return:** It represents the total gain or loss of an investment over a specified period, expressed as a percentage. It shows the overall performance, incorporating all changes in value.

$$R = \frac{v - v_0}{v_0} \tag{16}$$

  Where v_0 is initial capital and v is final portfolio value.

- **Max Drawdown:** Measures the largest decline in investment value from peak to trough during a specified period. It indicates the maximum loss an investor could have experienced.

- **Annual return:** Expresses the percentage change in the value of an investment over a one-year period. It summarizes the overall performance and provides a simple metric for assessing investment success.

$$r = (1 + R)^{\frac{365}{t}} - 1 \quad (17)$$

Where, t is number of trading days.

- **Annual volatility:** Annual volatility quantifies the degree of variation in the value of an investment over a one-year period. It reflects the level of risk or uncertainty associated with the investment's returns.

$$\sigma_a = \sqrt{\frac{\sum_{i=1}^{n} (r_i - \bar{r})^2}{n-1}} \quad (18)$$

Where, r_i is annualized return in year i, n is number of years and $\bar{r}$ is the average annualized return.

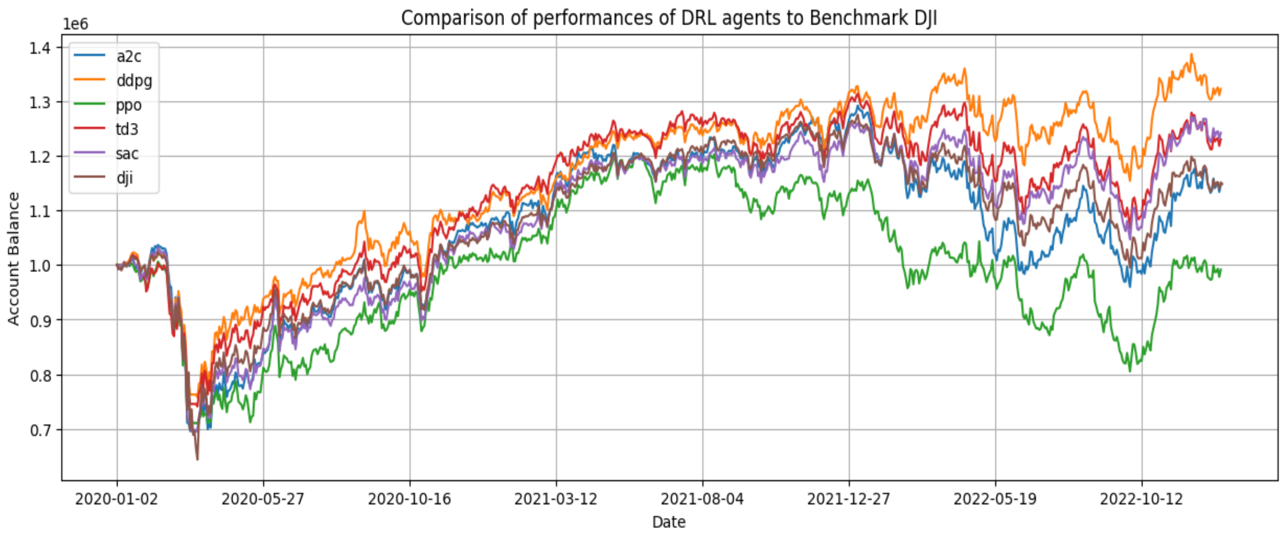
5 Results and Discussion

In this section, we carefully analyse the performance of different Deep Reinforcement Learning (DRL) agents applied to financial trading. We examine key metrics like annual return, cumulative returns, max drawdown, and Sharpe ratio for several agents (A2C, DDPG, PPO, TD3, SAC). This analysis reveals the strengths of each agent, highlighting DDPG's superior performance. We then compare DDPG in detail to the Dow Jones Industrial Average (DJI) benchmark. The goal of this section is to provide clear insights into how well DRL strategies work in the complex world of financial markets.

5.1 Comparison of DRL agents

Our experimental results reveal significant discrepancies in the performance of the evaluated deep reinforcement learning (DRL) agents. Visual analysis of the account balance over time *figure 12* and the *figure 13* of evaluation metrics highlights the following key observations.

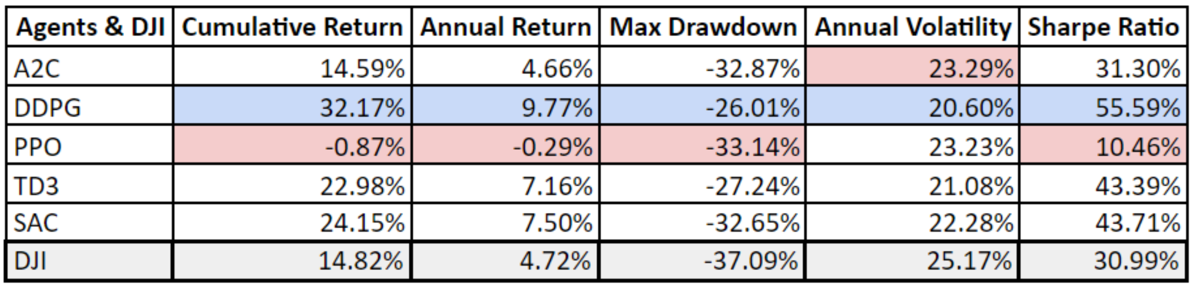
Figure 12 Comparison graph of DRL agents along with DJI



Source: Own processing.

The DDPG agent emerged as the top performer in our evaluation. It demonstrated the largest cumulative returns (32.17%), along with impressive risk-adjusted returns as evidenced by its strong Sharpe ratio. This success could be attributed to DDPG's design, combining actor-critic methods and off-policy learning, making it potentially well-suited for continuous control problems like trading. In contrast, the PPO agent exhibited the weakest performance with negative annual returns and the worst Max Drawdown. This underperformance suggests sensitivity to hyperparameters or potential mismatch between the algorithm's strengths and our trading environment. A2C, TD3, and SAC occupied a middle ground, showing promise but possibly requiring refinement for this specific problem. It's essential to consider not only returns but also the risk profiles of these agents; analysing metrics like Annual Volatility and Max Drawdown would provide a more comprehensive picture.

Figure 13 Performance stats



Agents & DJI	Cumulative Return	Annual Return	Max Drawdown	Annual Volatility	Sharpe Ratio
A2C	14.59%	4.66%	-32.87%	23.29%	31.30%
DDPG	32.17%	9.77%	-26.01%	20.60%	55.59%
PPO	-0.87%	-0.29%	-33.14%	23.23%	10.46%
TD3	22.98%	7.16%	-27.24%	21.08%	43.39%
SAC	24.15%	7.50%	-32.65%	22.28%	43.71%
DJI	14.82%	4.72%	-37.09%	25.17%	30.99%

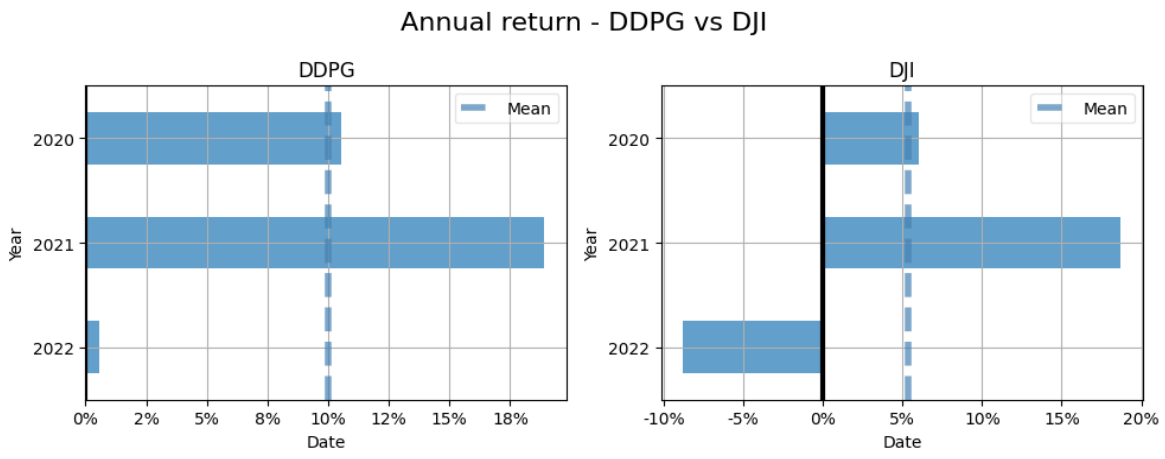
Source: Own processing.

DDPG's significant outperformance in this experiment makes it a compelling choice for further analysis. Its ability to surpass other DRL agents suggests the potential to compete with traditional financial benchmarks, like the Dow Jones Industrial Average (DJI). In the next section, we'll take a closer look at how DDPG stacks up against the DJI index.

5.2 DDPG vs. DJI Benchmark

A deeper dive into DDPG's performance against the Dow Jones Industrial Average (DJI) reveals notable strengths, particularly during market chaos.

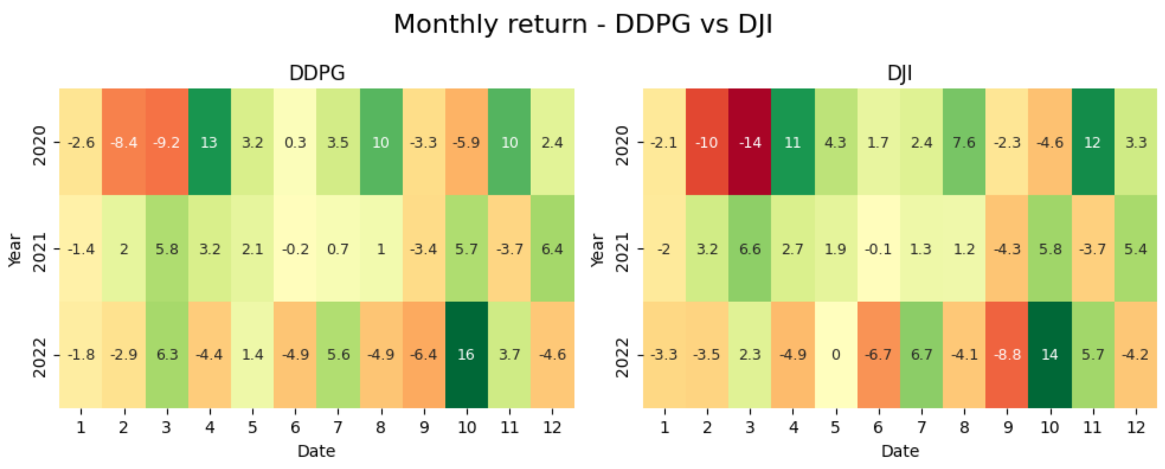
Figure 14 Bar chart, annual return



Source: Own processing.

The bar graph *figure 14* of annual returns shows that DDPG had positive returns in 2022, a year when the DJI declined significantly (around -8%). Even during the COVID-related market downturn, DDPG's performance held up relatively well. This suggests that DDPG can adapt to difficult market conditions better than strategies that simply track an index.

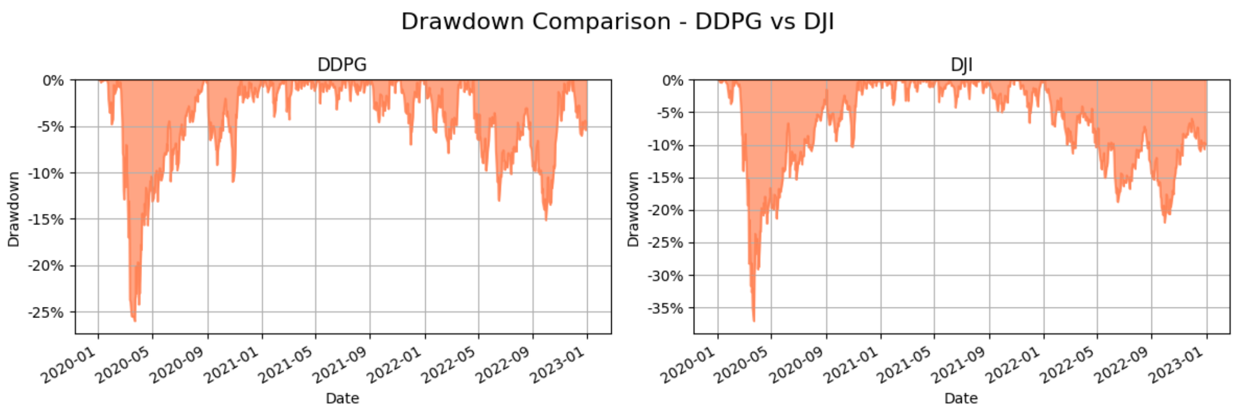
Figure 15 Heatmap, monthly return



Source: Own processing.

As shown in the *figure 15* of monthly returns, both DDPG and the DJI struggled in March 2020 (the COVID crash). However, both saw a strong performance in October 2022, which might suggest that DDPG can identify and react to positive market trends.

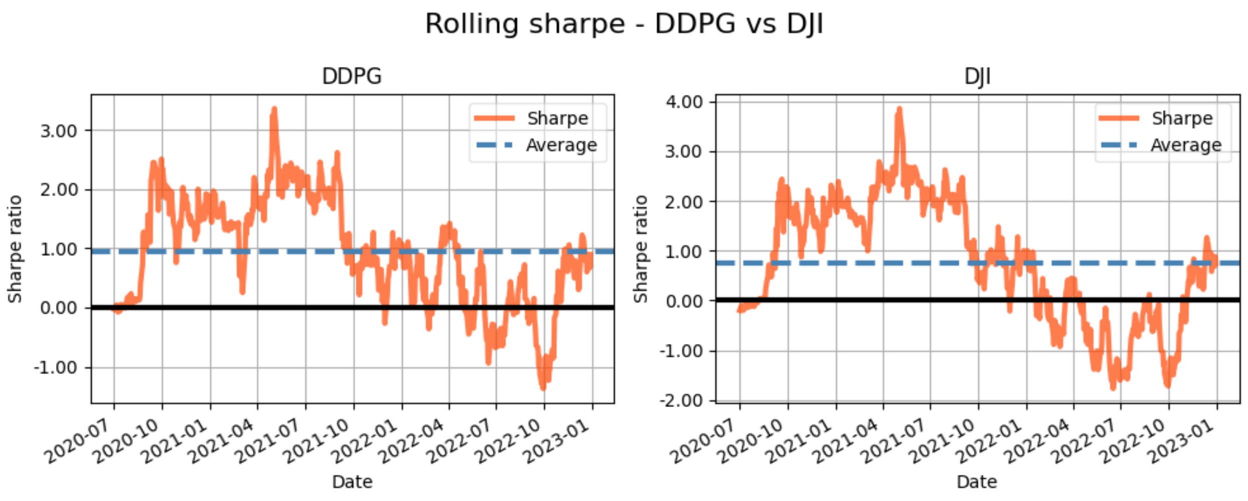
Figure 16 Underwater graph, Max Drawdown



Source: Own processing.

In the underwater graph *figure 16* of Max Drawdown, we can see that DDPG's drawdown is much smaller than the DJI's. Even during their worst period in March 2020, DDPG's maximum drawdown was only -25% compared to -35% for the DJI. This suggests that DDPG may be better at managing risk than traditional index tracking.

Figure 17 Rolling Sharpe graph



Source: Own processing.

As it appears in *figure 17*, DDPG's ability to maintain a Sharpe ratio around its mean even after 2022 demonstrates consistency and risk control in a volatile environment. The declining Sharpe ratio of the DJI, on the other hand, signals increasing risk relative to returns.

DDPG's performance compared to the DJI benchmark suggests that it offers distinct advantages for trading strategies. Its ability to generate positive returns and minimize losses during market downturns demonstrates the power of reinforcement learning in the financial domain. DDPG's apparent capacity to adapt to market changes and prioritize risk management further highlights the potential limitations of traditional index-tracking approaches.

5.3 Limitations and future directions

While our findings demonstrate the potential of DRL in algorithmic trading, it's important to acknowledge key limitations. Real-world financial markets are incredibly complex, and RL agents may not fully grasp every market dynamic. Additionally, the quality and quantity of training data significantly impact agent performance. Therefore, ensuring robust data sourcing and addressing potential biases are essential. It's also crucial to test an agent's generalizability across diverse market conditions to assess its true adaptability.

To build upon these findings, future research should prioritize hybrid approaches that combine RL with traditional financial models or sentiment analysis. Explicitly incorporating risk management into the RL framework is vital for real-world deployment. Moreover, realistic modelling of transaction costs, investigating wider algorithmic market impact, and exploring the potential of newer RL algorithms offer exciting avenues for further advancement in this field.

6 Conclusion

This thesis embarked on a comprehensive investigation into the potential of deep reinforcement learning (DRL) for algorithmic stock trading. The primary objective was to evaluate the effectiveness of various DRL algorithms and benchmark their performance against the traditional DJIA index. Through rigorous experimentation, the study has demonstrated both the promise and challenges inherent in applying DRL to this complex domain.

A key finding of this research is the ability of DRL agents to exhibit adaptability, particularly during market downturns. Notably, the DDPG agent displayed greater resilience compared to the benchmark, maintaining positive returns in periods when the DJIA experienced significant losses. This highlights the potential of RL-driven strategies to navigate market volatility more effectively than purely index-tracking approaches.

While the results of this study are encouraging, it's crucial to acknowledge limitations and outline avenues for future research. The complexity of real-world markets necessitates further exploration of factors such as transaction costs, market impact, and the integration of diverse data sources. Additionally, advancements in explainable RL could pave the way for greater transparency and trust in these algorithmic trading systems.

This thesis contributes to the ongoing dialogue on the intersection of machine learning and finance. By systematically evaluating DRL algorithms and offering insights into their performance characteristics, this work lays a foundation for the development of increasingly sophisticated, intelligent, and robust trading strategies powered by reinforcement learning principles.

7 References

BAHETI, Pragati. Activation Functions in Neural Networks [12 Types & Use Cases]. Online. 2021. Available at: https://www.v7labs.com/blog/neural-networks-activation-functions.

BELLMAN, Richard. Dynamic Programming. 1. Princeton University Press, 1957. ISBN 978-0691079518.

BERTSEKAS, Dimitri P. a TSITSIKLIS, John N. Neuro-Dynamic Programming (Optimization and Neural Computation Series, 3)1. 1. Athena Scientific, 1996. ISBN 978-1886529106.

BROWNLEE, Jason. A Gentle Introduction to the Rectified Linear Unit (ReLU). Online. 2020. Available at: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

CARR, Thomas. Deterministic vs. Stochastic Policies in Reinforcement Learning. Online. 2023. Available at: https://www.baeldung.com/cs/rl-deterministic-vs-stochastic-policies.

DODD, Randall. Financial Market. Online. 2020. Available at: https://www.imf.org/en/Publications/fandd/issues/Series/Back-to-Basics/Financial-Markets.

DOLAN, Brian. What Is MACD? Online. 2024. Available at: https://www.investopedia.com/terms/m/macd.asp.

DUTTA, Sayon. Reinforcement Learning with TensorFlow: A Hands-On Introduction to Deep Q-Learning using the TensorFlow. 1. O'Reilly Media, 2018. ISBN 978-1788835725.

FERNANDO, Jason. Relative Strength Index (RSI) Indicator Explained With Formula. Online. 2024. Available at: https://www.investopedia.com/terms/r/rsi.asp.

GOODFELLOW, Ian; COURVILLE, Aaron a BENGIO, Yoshua. Deep Learning. 1. MIT Press, 2016. ISBN 978-0262035613.

GROETTE, Oddmund. Trading Performance: Strategy Metrics, Risk-Adjusted Metrics, And Backtest. Online. 2023. Available at: https://www.quantifiedstrategies.com/trading-performance/.

HABIB, Nazia. Hands-On Q-Learning with Python: Practical Q-learning with OpenAI Gym, Keras and TensorFlow. 1. Apress, 2019. ISBN 978-1484243568.

HARDESTY, Larry. Explained: Neural networks. Online. 2017. Available at: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

CHOW, Adrian. Solving Continuous Control using Deep Reinforcement Learning (Policy-Based Methods). Online. 2021. Available at: https://ahtchow.medium.com/solving-continuous-control-using-deep-reinforcement-learning-policy-based-methods-64a871832496.

JOHNSON, Barry. Algorithmic Trading and DMA: An introduction to direct access trading strategies. 1. 4Myeloma Press, 2010. ISBN 978-0956399205.

KARUNAKARAN, Dhanoop. The Actor-Critic Reinforcement Learning algorithm. Online. 2020. Available at: https://medium.com/intro-to-artificial-intelligence/the-actor-critic-reinforcement-learning-algorithm-c8095a655c14.

KARUNAKARAN, Dhanoop. Soft Actor-Critic Reinforcement Learning algorithm. Online. 2020. Available at: https://medium.com/intro-to-artificial-intelligence/soft-actor-critic-reinforcement-learning-algorithm-1934a2c3087f.

KVARTALNYI, Nazar. Deep Q-Learning Explained: A Comprehensive Guide. Online. 2023. Available at: https://inoxoft.com/blog/deep-q-learning-explained-a-comprehensive-guide/.

LAPAN, Maxim. Deep Reinforcement Learning Hands-On. 1. Packt Publishing, 2020. ISBN 9781838826994.

LEE, Mark. Returns. Online. 2005. Available at: http://incompleteideas.net/book/ebook/node30.html.

PUTERMAN, Martin L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. 1. Wiley-Interscience, 1994. ISBN 978-0471619771.

RAVICHANDIRAN, Sudharsan. Hands-On Reinforcement Learning with Python: Master reinforcement and deep reinforcement learning using OpenAI Gym and TensorFlow. 1. Packt Publishing, 2018. ISBN 978-1788836524.

RUSSELL, Stuart a NORVIG, Peter. Artificial Intelligence: A Modern Approach. 4. Pearson Education Limited, 2020. ISBN 978-0134610993.

SANTHOSH, Sthanikam. Reinforcement Learning(Part-7): Twin Delayed Deep Deterministic Policy Gradient(TD3) in Tensorflow2. Online. 2022. Available at: https://medium.com/@sthanikamsanthosh1994/reinforcement-learning-part-7-twin-delayed-deep-deterministic-policy-gradient-td3-in-tensorflow2-726fb9a53ae6.

SIMONINI, Thomas. Advantage Actor Critic (A2C). Online. 2022. Available at: https://huggingface.co/blog/deep-rl-a2c.

SINGH, Ayush. Reinforcement Learning: Bellman Equation and Optimality (Part 2). Online. 2019. Available at: https://towardsdatascience.com/reinforcement-learning-markov-decision-process-part-2-96837c936ec3.

SINGH, Ayush. Reinforcement Learning : Markov-Decision Process (Part 1). Online. 2019. Available at: https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da.

SUTTON, Richard S. a BARTO, Andrew G. Reinforcement Learning: An Introduction. 2. MIT Press, 2018. ISBN 0262352702.

SZEPESVÁRI, Csaba. Algorithms for Reinforcement Learning. 1. Morgan & Claypool, 2010. ISBN 9781608454921.

TUCCI, Linda (ed.). Artificial neuron. Online. 2013. Available at: https://www.techtarget.com/searchcio/definition/artificial-neuron.

WALIA, Anish Singh. Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent. Online. 2021. Available at: https://medium.com/nerd-for-tech/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-descent-1e32cdcbcf6c.

How Is Machine Learning Used in Trading? Online. 2022. Available at: https://www.spiderrock.net/how-is-machine-learning-used-in-trading/.

8 List of figures and abbreviations

8.1 List of figures

Figure 1 Agent-Environment interface in a MDP 16
Figure 2 Structure of the artificial neuron 24

Figure 3 Structure of Feedforward neural network 25

Figure 4 ReLU Activation function 26

Figure 5 Comparison of optimizers training on MNIST images. 27

Figure 6 Data fetch from Yahoo finance 33

Figure 7 Date feature engineering 34

Figure 8 Data transformation for state space 34

Figure 9 Libraries and imports for DRL agent to train 35

Figure 10 Creation of stock trading environment for DRL agents 36

Figure 11 Training process of DRL agent 37

Figure 12 Comparison graph of DRL agents along with DJI 40

Figure 13 Performance stats 40

Figure 14 Bar chart, annual return 41

Figure 15 Heatmap, monthly return 42

Figure 16 Underwater graph, Max Drawdown 42

Figure 17 Rolling Sharpe graph 43

8.2 List of Abbreviations

A2C	: Advantage Actor Critic
ADAM	: Adaptive Moment Estimation
CSV	: Comma Separated Values
DDPG	: Deep Deterministic Policy Gradient
DJIA	: Dow Jones Industrial Average
DQN	: Deep Q-Networks
EMA	: Exponential Moving Average
FNN	: Feedforward Neural Network
MACD	: The Moving Average Convergence Divergence
MDP	: Markov Decision Process
ML	: Machine Learning
NN	: Neural Network
OHLCV	: Open, High, Low, Close and Volume
PPO	: Proximal Policy Optimization
ReLU	: Rectified Linear Unit
RL	: Reinforcement Learning
RSI	: Relative Strength Index
SAC	: Soft Actor Critic
TD3	: Twin Delayed DDPG