Zurich University of Applied Sciences ZHAW

School of Management and Law

Master in Banking and Finance

Module: Master Thesis

Deep Reinforcement Learning for Trading Securities

Submitted by:
Fabio Bührer

Supervisor:
Dr. Bledar Fazlija
Department of Quantitative Finance

Submitted on:
Thursday, July 1, 2021

I. Table of Contents

II. List of Figures

III. List of Tables

1 Introduction

Machine learning methods find practical use in many modern-day scenarios, facilitating or improving the execution of tasks from simple to complex. While for some time the most prominent techniques were associated with algorithms solving problems through learning in a supervised or unsupervised manner, the paradigm of reinforcement learning has more recently also gained comparable fame, perhaps most notably through Google's AlphaGo breakthrough wins against distinguished Go player Lee Sedol in 2016 (DeepMind, n.d.). Owing to their generality and adaptability, reinforcement learning and its extension deep reinforcement learning can find use in a wide variety of fields including industrial automation, gaming, and healthcare (Chao, Jiming, & Nemati, 2020). This paper explores potential applications of deep reinforcement learning in the field of securities trading by analyzing the returns generated by variants of an agent acting autonomously in simplified trading environments. In particular, daily financial data for 17 listed securities representing six major asset classes and the 11 Global Industry Classification Standard sectors (MSCI Inc., 2021) are fed into the agent's environments to investigate how variating the neural network's architecture and the lookback time horizon considered for making trading decisions affects cumulative returns for a specific security.

1.1 Motivation

This study aims at establishing whether deep reinforcement learning techniques are suitable for applications in trading and the effects that some practical aspects such as the security being traded or implementation details have on the performance. The agents interacting with the trading environments consist of variants of a basic deep Q-network algorithm trained and tested on different timeseries of the same length. The entire software implementation is using the Python programming language.

1.2 Problem Statement and Research Question

Data generation and collection practices have been increasing exponentially for some time and so has the availability of extensive data sets. Correctly leveraging these vast amounts of data to automate tasks or to inform decision-makers can be a competitive advantage in trading, where top-players typically have comparable access to securities data. Deep reinforcement learning methods can help in selecting the most promising

course of action solely based on the data they are fed, providing the basis for this paper's research question:

"Can deep reinforcement learning methods suggest profitable decisions for trading securities?"

1.3 Scope

The broad spectrum of possible implementations, the fast pace of state-of-the-art research, and the limited computational resources available to the author impose limitations on the variety of aspects of reinforcement learning that can be investigated in this paper. The scope of the work presented in this study is broadly outlined by the points below:

1. The software libraries the used for the empirical part are developed by third parties and not by the author. The author makes use of the software as provided by the respective distributors without altering the source code, but by creating programs that take advantage of the Python application programming interface of each library. The range of methods available to the author is therefore limited by the prepackaged software.
2. The financial data is downloaded through the Yahoo Finance application programming interface and is assumed to be coherent and correct as of writing.
3. This paper is the author's master's degree thesis for the curriculum in banking and finance with specialization in capital markets and data science at the Zurich University of Applied Sciences. The variety of and the degree of thoroughness with which the topics are explained and explored reflects the author's level of skill, knowledge, and experience.

1.4 Structure of the Thesis

This thesis consists of four chapters, each being a logical step towards answering the research question and drawing concluding considerations for the study. This introductory part is followed by the second chapter, which explains the theory and explores the literature underpinning the main concepts implemented in the practical part of the paper. The empirical procedure is exposed in the third chapter, where the data, algorithm variations, and intermediate results analyzed. The fourth and last chapter summarizes and

discusses the empirical results, exposes methodological limitations, provides recommendations for similar research, and explores potential real-world applications for the methods used in this paper.

2 Theoretical Fundamentals

Since the goal of the thesis is to establish how effective deep reinforcement learning can be in trading securities, some aspects of machine learning that are not central to the implemented methods are not examined in depth. The theory in this chapter should provide the reader with an understating of reinforcement learning within the field of machine learning and explain the key mechanics behind some of the most common algorithms. The referenced literature provides additional reading material for more thorough and rigorous explanations.

2.1 Machine Learning

Machine learning is usually regarded as the sub-field of artificial intelligence concerned with studying computer algorithms that can solve specific problems by autonomously learning a set of rules from data. Besides the pure scientific interest for the role played by the various variables, often the goal is to obtain an algorithm that has a certain problem-solving performance when applied to unseen data, an ability referred to as generalization. The versatility and power of machine learning methods often makes them attractive candidates for automating specific, relatively complex tasks that can be expressed in terms of data (Nguyen & Zeigerman, 2018, pp. 7–8). Machine learning can also be seen as new programming paradigm; in classical programming, detailed rules have to be programmed manually and the data fed to the program to receive answers, whereas with machine learning the rules constitute the main objective of the program.

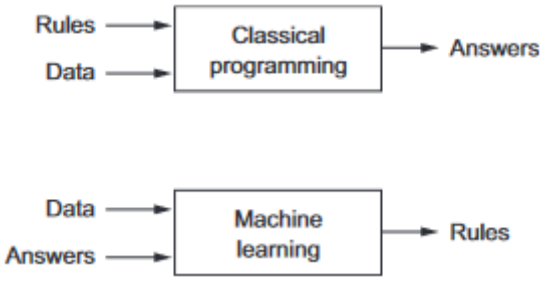

Figure 1: Machine learning as a new programming paradigm (Chollet, 2018, p. 5)

A common categorization of machine learning techniques is based on the type of learning that the algorithm is expected to go through, usually distinguishing between supervised, unsupervised, and reinforcement learning. Supervised learning is done by seeing examples of correct input-output pairs, as is the case for example in optical character recognition. By contrast, in unsupervised learning the algorithm is not fed the correct answers, as the notion of correct and incorrect answers is often not even included in problem, but rather the machine is expected to find patterns in the data and thus learn a representation of them; an example is customer segmentation based on a set of features. Reinforcement learning scenarios occur when agents ought to learn the actions that maximize a measure of reward from the environment with which they interact; an example is playing a videogame (Sutton & Barto, 2018, p. 2)

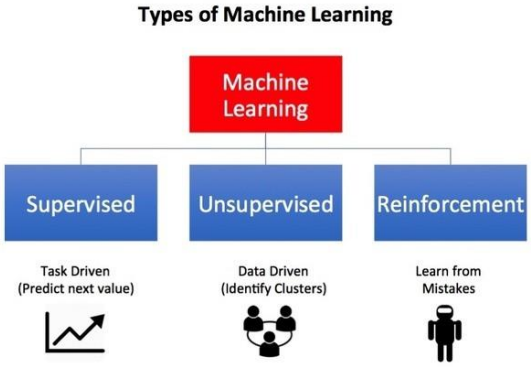

Figure 2: Learning paradigms within machine learning (Bhatt, 2018)

2.2 Deep Learning

Deep learning is a subset of machine learning concerned with using artificial neural networks to learn meaningful representations from data. Artificial neural networks are layers of artificial neurons organized in successive interconnected layers that make up the architecture. In general, every artificial neural network possesses an input layer, used for reading the data, one or more hidden layers where the representations are learned, and an output layer devoted to relaying the results. Deep learning as a field focuses on using artificial neural networks with multiple hidden layers (hence “deep”) to learn increasingly informative representations from complex and large sets of data (Chollet, 2018, pp. 8–11). Figure 3 provides a good illustration of how the data representations could work in the case of a digit-classification model.

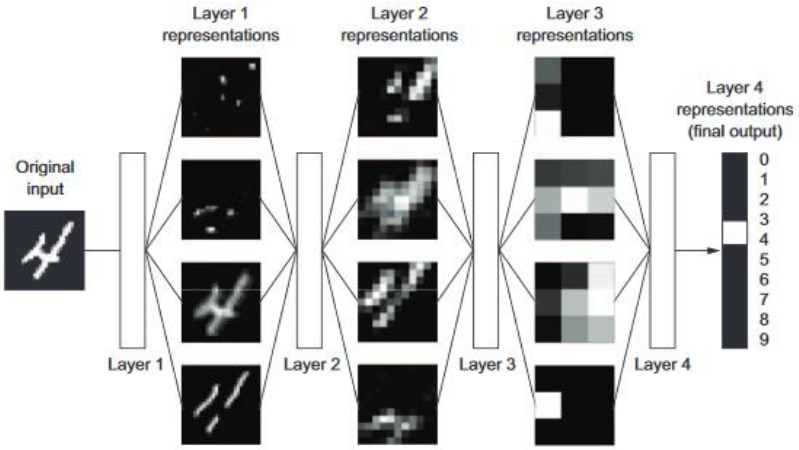

Figure 3: Layered representations as learned by a digit-classification model (Chollet, 2018, p. 9)

Learning in the context of deep learning means parametrizing the neurons in such a way that the artificial neural network reaches a certain measure of accuracy when mapping the inputs to the outputs. Each neuron typically has several parameters that can be changed to alter its output function: a bias b and a weight w for each of the inputs x from the neurons in the preceding layer to which it is connected. The number of parameters to be learned is therefore usually very high. It is common practice for neurons to have a non-linear activation function that is applied to the linear combination of it inputs such as the sigmoid function before the neuron's output is send to the next layer. This allows the network to model non-linearly behaving patterns found in the data; without non-linearity, an equivalent network without hidden units can be found that has the same modeling capability (Bishop, 2006, p. 229).

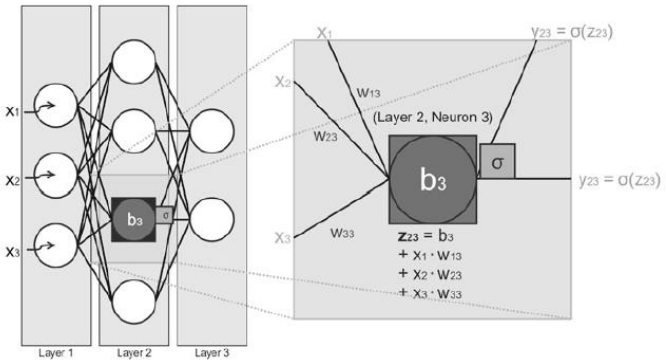

Figure 4: Inner workings of a hidden layer neuron (Skansi, 2018, p. 80)

Usually the parameters of the network are initialized with random values. Then, with each sample, the parameters are changed to reduce the error, calculated by a loss function,

between the network's output and the correct solution. Intuitively one step of the network's training can be explained as follows: the value of the error is calculated by a loss function; then, by applying the chain rule and partial derivation, a so-called backpropagation algorithm determines the role played by each parameter in the change of the loss function; finally, the adjustment necessary to reduce the error network is computed for each parameter (Bishop, 2006, p. 241).

2.3 Reinforcement Learning

Reinforcement learning is the subfield of machine learning that studies how agents can learn to make decisions that maximize a quantitative reward through experience.

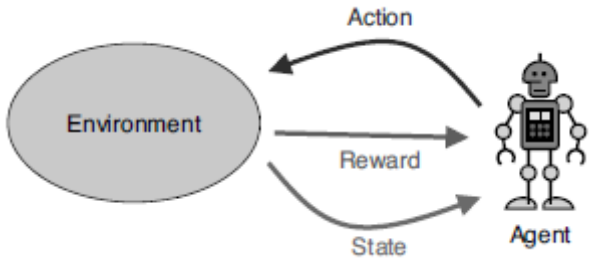

Figure 5: Schematic representation of reinforcement learning (Sugiyama, 2015, p. 4)

2.3.1 Main Characteristics of Reinforcement Learning

Although some tasks may be benefit from approaches based on any or combinations of supervised, unsupervised, and reinforcement learning, usually only one type of learning algorithms is chosen to tackle the problem. Supervised learning's requirement for correct and representative examples of correct behavior are often impractical; using chess as an example, it is difficult to imagine to have a complete training set that contains the ideal move of each piece for all possible chessboard configurations. Unsupervised learning on the other hand is not meant to directly suggest which action promises the most reward at each step. Since reinforcement learning does not suffer from the same limitations, it is usually the most suited for interactive tasks.

Another distinctive characteristic of reinforcement learning is the exploration-exploitation trade-off. The trade-off refers to the tension between repeating greedy actions that have proved to be profitable before, which is exploiting collected knowledge, and trying new nongreedy actions that may result in even better rewards, which is exploring

new behaviors. To succeed at a task, the agent has to strike a balance between exploration and exploitation.

Finally, reinforcement learning is conceptually straightforward to adopt for many common problems, as it does not necessitate to break the task into smaller units to be analyzed and solved singularly, but rather the starting assumption is to train an agent to reach a goal while making decisions in and interacting with a stochastic environment (Sutton & Barto, 2018, pp. 1–4).

2.3.2 Elements of Reinforcement Learning Systems

The primary elements of an interactive problem are the agent and the environment. The agent is the entity that tries to learn to behave optimally through its actions a, and the environment is the domain with which the agent interacts and is typically characterized by some degree of unpredictability (from the agent's perspective) of its states s. In many problems, the actions of the agent can have an effect of the future states of the environment, as is the case in many games. A reinforcement learning setting usually also includes four additional elements: a policy, a reward signal, a value function, and a model of the environment (Sutton & Barto, 2018, pp. 6–7).

A policy π determines all of the agent's decisions, therefore it can be seen as a state-action mapping. Basic approaches for building the mapping include simple functions or tables, whereas more involved processes may include complex constructs such as neural networks. The state-action mapping is usually formulated in terms of probabilities for each action, thus it can be denoted as a stochastic policy.

The reward signal r is a quantity that the environment emits at each state and that the sum of which the agent attempts to maximize. The reward is logically the main driver of changes in the policy, as the latter decides the actions the agent should take for each state.

The value function $v_*(s)$ or $q_*(s, a)$ (depending on the algorithm) can be described as the long-term equivalent of the reward signal. The value function maps the current state to the expected value of all future rewards, therefore quantifying how desirable a given state is in the long run. Although values are derived from rewards, it is values that matter for selecting actions, since only in this way can rewards be maximized over the long term. Given the indirect way a value function is estimated, their calculation has to be repeated

regularly based on the states of the environment. Efficiently estimating value functions is of primary importance for every reinforcement learning algorithm.

Model-based reinforcement learning methods also include a model of the environment $p(s' \mid s, a)$, which allows to infer future states of the environment before experiencing such situations, following a planning approach. Model-free methods on the other hand follow a trial-and-error approach. Current reinforcement learning methods cover a range between the extremes of trial-and-error and planning.

2.3.3 Overview of Reinforcement Learning Problems

In this part of the theory an overview of the reinforcement learning landscape is provided with the aim of helping the reader understand the classes of methods and the various types of formal problems that the methods attempt to solve.

2.3.3.1 Tabular Solution Methods

Tabular solution methods apply to the simplest form of problems where both environment states and agent actions can be modeled by the value function in the form of tables with discrete entries, for example in the game of tic-tac-toe.

The special case of a setting where the agent has to select an action out of k possibilities for only one specific state is named a k-armed bandit problem. The main difficulty in k-armed bandit problems is estimating the value $q_*(a)$ of each available action a in order to make the correct choice at each step; if the agent only exploits known actions, it cannot update and improve the estimates and will keep on making the same safe choice, while if it explores new actions, it may incur a lower reward in the short-term compared to following a greedy behavior. Since only one behavior can be followed at any one step, k-armed bandit problems offer a clear understanding of the exploration-exploitation trade-off common to all reinforcement learning problems (Sutton & Barto, 2018, pp. 25–27).

However, the general case of reinforcement learning problems can be described by Markov decision processes. In Markov decision processes, the value of each action depends both on the action and the current state s, therefore $q_*(s, a)$ has to be estimated; an alternative is estimating the value of the state provided that the optimal actions have been selected, $v_*(s)$. In the case of Finite Markov decision processes, where the spaces of states S, actions A, and rewards R are finite, the problem is usually solved by recurring

to any or a combination of dynamic programming, Monte Carlo methods, and temporal-difference methods (Sutton & Barto, 2018, p. 23).

Dynamic programming is a collection of techniques for iteratively determining the optimal policy given a perfect model of the environment. This means that the probability distributions $p(s', r \mid s, a)$, which determine the transitions from the current situation (s, a) to the new situation with state s' and reward r, are known for all $s \in A$, $a \in A$, and $r \in R$. This constraint and the computational overhead are the main reasons why classical dynamic programming algorithms alone find limited use in reinforcement learning (Sutton & Barto, 2018, p. 73).

Monte Carlo methods try to overcome the limiting assumption about perfect modeling of the environment in dynamic programming by using agent experience to estimate the values of specific states. In simple terms, the estimation is done by sampling and averaging the total discounted returns $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ (where $\gamma \in [0,1]$ is a factor for determining the preference for returns near in the future) for each state-action pair and the rewards for each action. Monte Carlo methods rely on the completion of the task, called an episode, to update the estimates (Sutton & Barto, 2018, p. 91).

Temporal-difference methods can conceptually be seen as an attempt to unite desirable qualities from both Monte Carlo and dynamic programming methods. Like for Monte Carlo, temporal-difference learns from experience and does not require a model of the environment's dynamics, and like dynamic programming, it can update the estimates before the conclusion of each episode (Sutton & Barto, 2018, p. 119) .

2.3.3.2 Approximate Solution Methods

Many reinforcement learning problems do not adhere to or cannot be feasibly translated into tabular form, since the state space may be too extensive, like in the case of a robot having to decide how to move. These problems often present the dual hurdle of too large tables to store all states, and the impossibility or impracticality of visiting all states to fill out the tables. Such cases require approximations that allow the algorithm to generalize its experience starting from a limited subset of all possible states. The generalization property is obtained from methods borrowed from supervised learning, including linear function approximations or artificial neural networks. (Sutton & Barto, 2018, p. 195).

2.4 Deep Reinforcement Learning

Deep reinforcement learning is an extension of classical reinforcement learning algorithms that exploits neural networks' ability to model non-linear relationship for approximating learnable functions of the reinforcement learning system.

2.4.1 Classifying Deep Reinforcements Learning Algorithms

Modern reinforcement learning algorithms are usually classified by what part of the reinforcement learning system they try to learn (Graesser & Wah Loon, 2020, pp. 12–14).

Policy-based methods try to maximize the expected cumulative discounted rewards by learning the ideal policy. The generality of this type of algorithms means that they can be used for problems where the action space is discrete, continuous or both. An additional advantage is that they are guaranteed to converge to locally optimal policy. The main disadvantages are high model variance and the inefficient use of the samples, which means that these algorithms need comparatively more samples to reach a certain level of performance.

Value-based algorithms attempt to learn the value function $v_*(s)$ or $q_*(s, a)$ to select the action that maximize the rewards. These algorithms are typically more sample-efficient than their policy-based counterparts, owing to their lower variance and thus make better use of information collected from the environment.

Model-based algorithms learn a model function $p(s' \mid s, a)$ of how the environment transitions from one state to the next or are provided with a known dynamics model. At each step, the agent tries to make various predictions of future states basing on the current state and a sequence of hypothetical actions. The attractiveness of a model-based algorithm resides in the fact that the agent is provided with predictive ability, making it a good candidate for situations where collecting sufficient experience is carries great expense in time or money. The predictive ability makes such an algorithm also very sample efficient: it can learn both from true experiences as well as through hypothetical scenarios. However, for many problems, including trading securities as treated in this thesis, the transition probabilities are unknown or not sufficiently accurate to model the environment several steps into the future.

In addition to the three main classes above, some algorithms combine approaches. A good overview of the various classes and corresponding algorithms is provided in Figure 6.

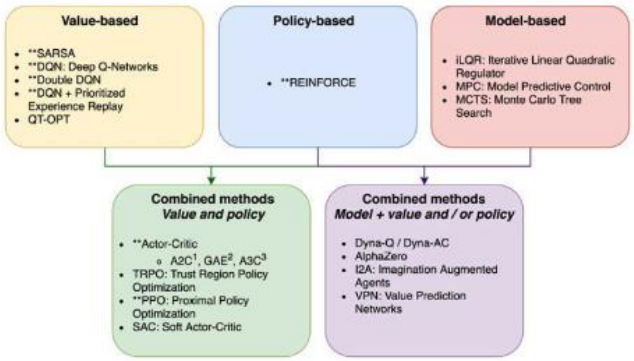

Figure 6: Classification of common deep reinforcement learning algorithms (Graesser & Wah Loon, 2020, p. 12)

Another distinction that is made when describing deep reinforcement algorithms is whether they are on-policy or off-policy. On-policy methods learn from the policy they are using as of training, making prior versions of the policy unusable and therefore being sample inefficient. Off-policy methods on the other hand do not learn directly on the current policy but use a separate, more exploratory policy called a behavior policy to collect experiences, therefore they are more sample efficient (Sutton & Barto, 2018, p. 103).

2.4.2 Overview of Common Deep Reinforcement Learning Algorithms

This sub-section provides an overview of the most widely known and implemented deep reinforcement learning algorithms that may be relevant and feasible candidates for implementation in the empirical part of this thesis.

2.4.2.1 SARSA

SARSA (short for State-Action-Reward-State-Action) is a value-based, on-policy algorithm that tries to estimate the value function for the current policy $Q^{\pi}(s, a)$ through a neural network, using the temporal difference technique and selecting the next action a' using the current policy. These properties may make is suitable for situations where risky behavior should be avoided already at training time, for example when reinforcement learning is applied to expensive, physical systems (Graesser & Wah Loon, 2020, pp. 53–58).

2.4.2.2 DQN

DQN, which stands for deep Q-network, is a value-based, off-policy algorithm that roughly employs the same techniques as SARSA with the crucial difference that it uses the action that would maximize the expected future returns to estimate $Q^{\pi}(s, a)$. Compared to SARSA, DQN being off-policy and the fact that it can resample past experiences from a replay buffer makes it an attractive candidate for problems where efficient, stable training is important and where an exploratory behavior during training is not a disadvantage. One key disadvantage of DQN is that it can only be applied to environments with discrete action spaces, which however holds true in the case of this thesis as explained in the methodology (Graesser & Wah Loon, 2020, pp. 81–83).

2.4.2.3 REINFORCE

REINFORCE is a policy-based, on-policy algorithm that is based on the idea that actions leading to good outcomes should become more likely, and the opposite should occur for actions leading to less desirable outcomes. REINFORCE uses neural networks to approximate the policy function, and in fact the same neural network is able to approximate various policies simply by changing its parameters. In the context of learning, the goal of REINFORCE is to find a set of parameter values that produce a good policy, which it pursues by following a gradient ascent approach called policy gradient (Graesser & Wah Loon, 2020, pp. 25–28).

2.4.2.4 A2C

A2C, short for advantage actor-critic, are algorithms that combines the concept of the learned value function from DQN and the policy gradient of REINFORCE. The key components of A2C are the actor, which learns the parametrized policy as explained for REINFORCE, and the critic, which approximates the value function as in DQN. A new concept in A2C is also that of advantage: how better a specific action is compared to other available actions, $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$. The motivation is that using the learned advantage can be more informative compared to the rewards provided by the environment, for example in tasks where rewards are only given when the agent reach a certain goal (Graesser & Wah Loon, 2020, pp. 135–136).

2.5 Python Libraries

The programming part of this thesis makes extensive use of Python libraries. The most important (and required) are listed here.

2.5.1 Pandas

Pandas is used for manipulating data structures, typically in the context of quantitative research and data analysis. Pandas is developed to help the user work with potentially large data structures in a flexible and intuitive way using DataFrame and Series objects (pandas, 2021).

2.5.2 NumPy

NumPy is a package for scientific computing that is meant to provide high-performance numerical objects through the Python interface. NumPy has a high-level syntax that allows it to be used by anyone and can be used to store arbitrary generic data (The NumPy community, 2021).

2.5.3 Matplotlib

Matplotlib is a Python package for producing customizable, high-quality plots, that can be extended through functionalities of third-party packages (The Matplotlib development team, 2021).

2.5.4 Seaborn

Seaborn is a Python visualization library that works on top of Matplotlib. It provides a high-level interface for generating statistical plots (Waskom, 2020).

2.5.5 PyArrow

The PyArrow library provides a Python API for handling Apache Arrow objects. The author leverages the Apache parquet file format for storing various data (Apache Software Foundation, 2019).

2.5.6 Stable Baselines3

Stable Baselines3 is the PyTorch re-implementation of Stable Baselines, a popular library for reinforcement learning. Most notably, it has a unified structure for accessing each algorithm and follows the Gym application programming interface for creating

environments (Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations, 2020).

3 Methodology

This chapter exposes the empirical methods used to investigate the answer to the research question and provides some interesting intermediate results. Following the methodological steps implemented in the computer program, this chapter is divided into two subchapters. In the first subchapter, an overview of the procedures applied to the data is given. In the second subchapter, the empirical research part to address the research question is exposed.

3.1 Data Procedures

In this subchapter, the author illustrates the steps the lead to the data used for the application of the reinforcement learning methods and provides some descriptive statistics.

3.1.1 Data Selection and Procurement

The data used in this thesis is selected to cover a good variety of assets classes and economic activity sectors to investigate how a trained reinforcement learning agent may perform trading various securities. Specifically, the selected securities are chosen to represent six major asset classes: equity, commodities, fixed income, foreign exchange, infrastructure, and real estate. Within the equity asset class, all 11 GICS (Global Industry Classification Standard) (MSCI Inc., 2021) sectors are represented. It should be noted that some of the selected securities represent a proxy for an investment in the respecting asset class; for example, traditionally real estate investments are not done via publicly listed securities. The following table provides an overview of the securities selected:

Table 1: Overview of the selected securities

Ticker	Name	Asset Class (GICS Sector)
AAPL	Apple Inc.	Equity (Information Technology)
AMT	American Tower Corporation (REIT)	Equity (Real Estate)
AMZN	Amazon.com, Inc.	Equity (Consumer Discretionary)
BA	The Boeing Company	Equity (Industrials)

Ticker	Name	Asset Class (GICS Sector)
CL=F	Crude Oil Aug 21	Commodities [Future]
EURUSD=X	EUR/USD	Foreign Exchange
GOOGL	Alphabet Inc.	Equity (Communication Services)
IGF	iShares Global Infrastructure ETF	Infrastructure [ETF]
IVV	iShares Core S&P 500 ETF	Equity (Index)
JNJ	Johnson & Johnson	Equity (Health Care)
JPM	JPMorgan Chase & Co.	Equity (Financials)
LIN	Linde plc	Equity (Materials)
NEE	NextEra Energy, Inc.	Equity (Utilities)
PG	The Procter & Gamble Company	Equity (Consumer Staples)
VNQ	Vanguard Real Estate Index Fund ETF Share	Real Estate [ETF]
XOM	Exxon Mobil Corporation	Equity (Energy)
ZN=F	10-Year T-Note Futures,Sep-2021	Fixed Income [Future]

Daily data for the period spanning approximately ten years between December 31, 2010, and December 30, 2020 (both dates inclusive) is queried and downloaded for the securities listed above using the Python library yfinance, which requests the data from Yahoo Finance. The library allows to download data for a variety of fields, but in the scope of this thesis, the financial data downloaded for each ticker is limited to the following columns five columns, explained below:

1. Open: day's opening price;
2. High: day's high price;
3. Low: day's low price;
4. Close: day's closing price; and
5. Volume: day's total volume of traded securities.

The downloaded data points for the four price columns are already adjusted for events like stock splits or dividends.

3.1.2 Data Cleaning

The data set obtained the previous step is cleaned by removing rows where not all columns have an entry for each column. While this step arguably removes some information from the data, it allows an easier and homogenous handling of all securities data. The cleaned data set spans the same period as originally but contains 141 rows less (2468 vs. 2609 before removal).

3.1.3 Descriptive Statistics

In this subchapter some securities are inspected in detail and compared on the basis of visualizations and financial considerations. These cursory analysis may be helpful in explaining potential differences in performance in the application of deep reinforcement learning for trading the securities. The final chart compares all securities' cumulative returns over the 10-year period.

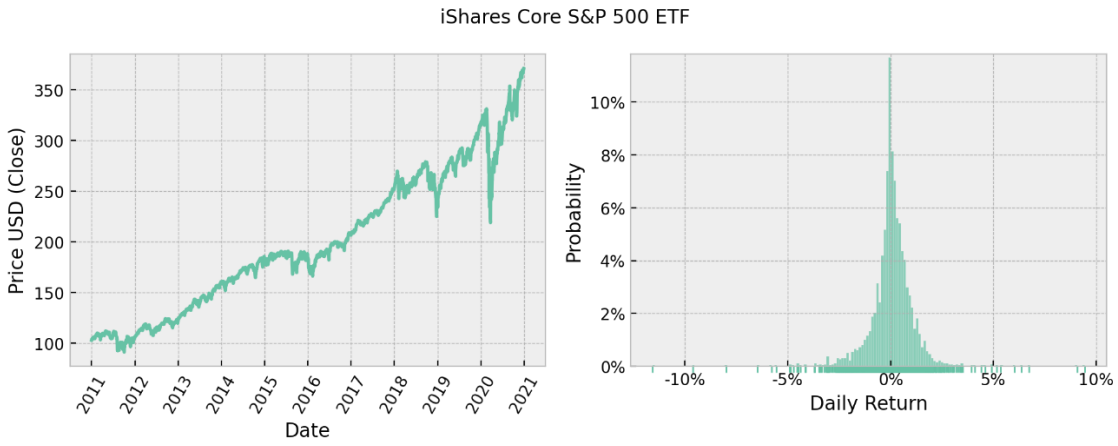

Figure 7: Price evolution and histogram of daily returns for the iShares Core S&P 500 ETF—own illustration

The S&P500 Index is often considered a good indicator of the status of the US economy (McFarlane, 2021), as it includes 500 of the largest publicly traded companies quoted on American stock exchanges and comprises about 80% of available market capitalization (S&P Dow Jones Indices, 2021). The quoted exchange traded fund (ETF) of the index is therefore a good reference point for comparing the other securities and having an idea of how the algorithm would "trade" the broader exchange-listed economy. It can be seen that over the period considered, the S&P500 ETF had an almost steady increase in price with a sharp drop around the beginning of the Covid-19 pandemic, and that daily returns are perhaps slightly positively skewed.

Figure 8: Price evolution and histogram of daily returns for Amazon.com, Inc.—own illustration

The stock price of Amazon.com Inc. followed and accelerating path towards the peak, with some rather volatile phases and a minor drop at the beginning of 2020. Daily returns reached up to 15% and are more likely to be slightly positive. These characteristics make this security a good buy-and-hold choice.

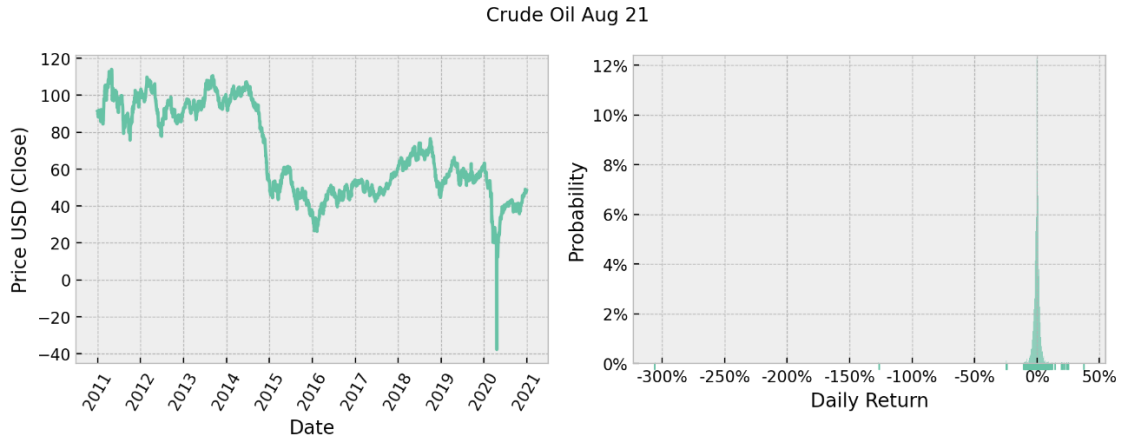

Figure 9: Price evolution and histogram of daily returns for the crude oil futures—own illustration

The crude oil future in Figure 9 illustrates a compelling case where trading the security successfully may prove extremely difficult. The persistent swinging of the price makes it difficult to identify promising entry points for a position, and a clear trend is not visible. Additionally, daily returns cover extreme values going as low as -300%, which can nullify even years of carefully cumulated profits.

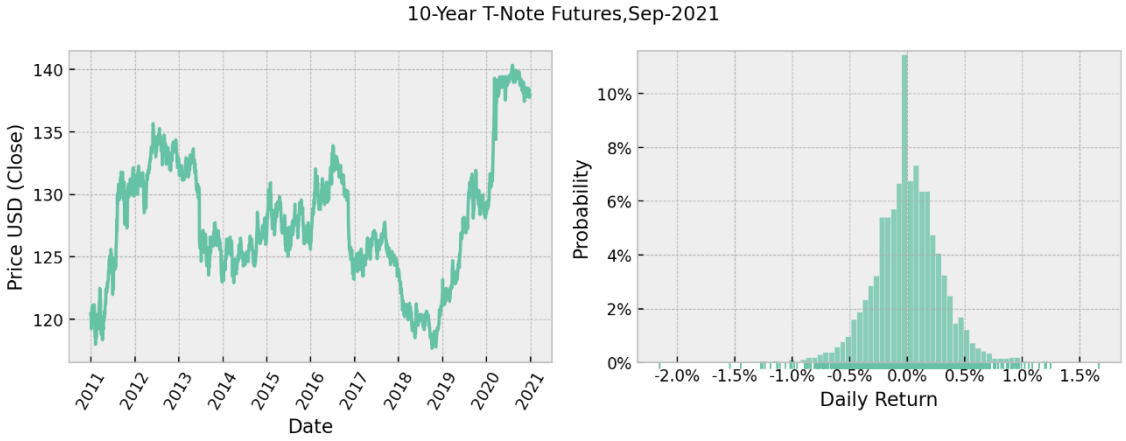

Figure 10: Price evolution and histogram of daily returns for US Treasury note futures—own illustration

Another intuitively difficult security to trade would be the futures on the ten-year US Treasury note. The price goes through very volatile phases and witnesses some sharp drops. A single, great entry period materializes towards the end of 2018, which however would be difficult to predict considering the price alone, since the subsequent growth does not follow a trend consistent with the rest of the chart. Daily returns are restricted to a very narrow range and follow an almost triangular distribution.

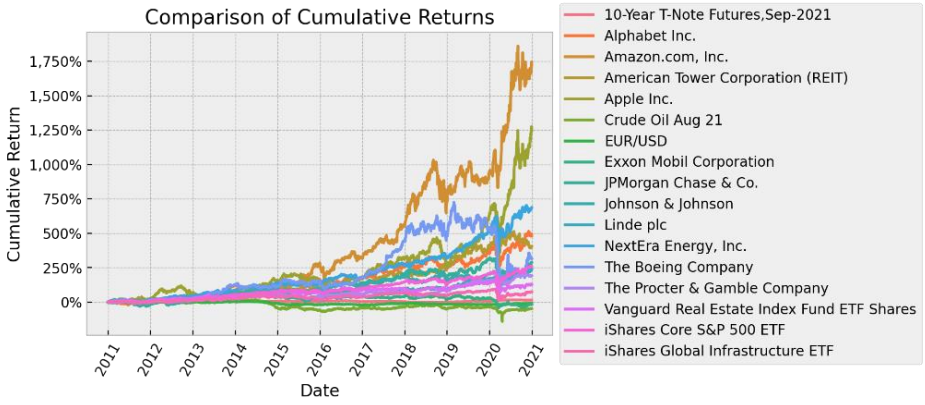

Figure 11: Cumulative returns visualized for all securities—own illustration

Figure 11 shows a comparison of all securities' cumulative returns over the entire period. Based on this comparison, it can be seen that a trading strategy more oriented towards entering long positions would generally be the safer choice.

3.1.4 Data Split and Normalization

As is common practice in the field of machine learning, the data is split into a training set used by the algorithm for learning and a test set on which the trained algorithm is tested;

the split following 80%-20% training-test proportions, respectively. Additionally, before usage with neural networks, some level of rescaling of the data to facilitate learning is applied. The author chooses a classical approach with a min-max scaler that forces each feature to have the normalized range $[0, 1]$. Care is taken to avoid normalizing the training data using parameters partially derived from the test data, as this data leakage could unduly improve the algorithm's performance out of sample.

3.2 Reinforcement Learning for Trading Securities

This subchapter exposes the steps of the empirical research part that is used to answer the research question formulated at the start of the thesis.

3.2.1 Planning the Implementation

The choice of algorithm to use has to be made before starting the actual computer implementation. Out of the numerous methods that fall within the scope of deep reinforcement learning, only some can be reasonably expected to deliver sensible results, given the author's limited resources and data. For example, all else being equal methods that are sample efficient are able to learn more with the same amount of training data compared to sample inefficient method. Another consideration for choosing the algorithm is whether it has limitations in terms of possible action and state spaces (Stable Baselines3, 2020). Software aspects also play a major role: some algorithms are available only in specific libraries, and the agent has to be compatible with environment it interacts with, therefore the pieces of software used to implement the two have to be compatible with one another. Finally, there is also the fundamental question whether the algorithm is well-suited for tackling the problem. After reviewing various literature (Zhang, Zohren, & Roberts, 2020) and software source (Stable Baselines3, 2020), the author decides to implement the analysis using the DQN algorithm from the Stable Baselines3 library, which uses a multi-layer perceptron (feed-forward) network with rectified linear unit activation, and the recommended environment setup to implement the reinforcement learning system.

3.2.2 Specification of the Environment

While implementing a custom environment using the Gym library interface is relatively easy from a development standpoint, it requires a sound understanding of how the agent should interact with the environment. The author opts for a model where at each step, the

environment reveals the financial data (open, high, low, close, and volume) of the security for a certain number of days leading up to the current day (the lookback window). Based on this, the agent has to decide what its position regarding the security should be: neutral, short, or long. The price at which the agent enters a long or short position is recorded each time the agent's position changes, and the profits from that position are realized only when the agent closes it. The reward (if any) at each step are the profits realized by closing the position. A schematic example can be:

1. The agent starts with a neutral position and no cumulative profits;
2. The agent decides to enter a long position, the current price of 0.5 is recorded;
3. The agent decides to be neutral, the current price of 0.6 is used to calculate the reward $r = 0.6 - 0.5 = 0.1$;
4. The agent decides to enter a short position, the current price of 0.7 is recorded;
5. The agent decides to enter a long position, the current price of 0.9 is used to calculate the reward $r = -(0.9 - 0.7) = -0.2$ and recorded;
6. The environment reaches the end of the episode (the last day in the data set), therefore potential profits or losses from the open long position are not assigned.

The algorithm is trained and tested within the environment using three different settings for the lookback window: five, 10, and 20 days, the latter approximately corresponding to the average number of trading days in a month (SwingTradeSystems.com, 2021).

3.2.3 Considerations for Choosing Algorithm Variants

The setting of the algorithm that may make a difference in how well the agent performs is the architecture of the neural networks devoted to approximating the value functions. It should be noted that the task of the networks is not to abstract or gain useful representations of the data as in the case of supervised learning, therefore it does not make sense to use very deep architectures, but rather to explore a few simple variants. In this thesis, three architecture variants are used:

1. Two hidden layers of 16 neurons each, denoted DQN[16, 16];
2. Two hidden layers of 32 neurons each, denoted DQN[32, 32]; and
3. Two hidden layers of 64 neurons each, denoted DQN[64, 64].

3.2.4 General Setup and Parametrization of the DQN Algorithm

Since the Stable Baselines3 DQN implementation does not support hyperparameter tuning as of writing (Raffin, 2021), the author recurs to tuning the hyperparameters manually by trial and error. The final non-default parameters as tuned by the author are reported in the table below:

Table 2: DQN parameters set by the author

Hyperparameter Name	Explanation	Value
learning_starts	Number of steps of the model to sample before learning starts	Number of steps in an episode (depends on lookback_window)
batch_size	The batch size for each gradient update	lookback_window
train_freq	Number of steps between each of the model's updates	loockback_window
target_update_interval	Number of steps between each of the target network's updates	Number of steps in an episode (depends on lookback_window)
exploration_fraction	Portion of steps over which the exploration rate is reduced	0.5
exploration_final_eps	Final value of the exploration rate	0.001

3.2.5 Training of the Algorithm

The author has established through iterative trials that training the algorithm for a number of steps equivalent to 40 episodes is a good compromise to avoid excessive fitting to the training data and too long training times, while still achieving some measure of convergence during training.

3.2.6 Trading Securities with DQN Variations

The next three subsections analyze the in-sample and out-of-sample results of each algorithm variant used with all three lookback windows.

3.2.6.1 Trading securities with DQN[16, 16]

3.2.6.1.1 5-Day Lookback Period

With a five-day lookback period, some degree of convergence of the episode mean reward is visible for some securities.

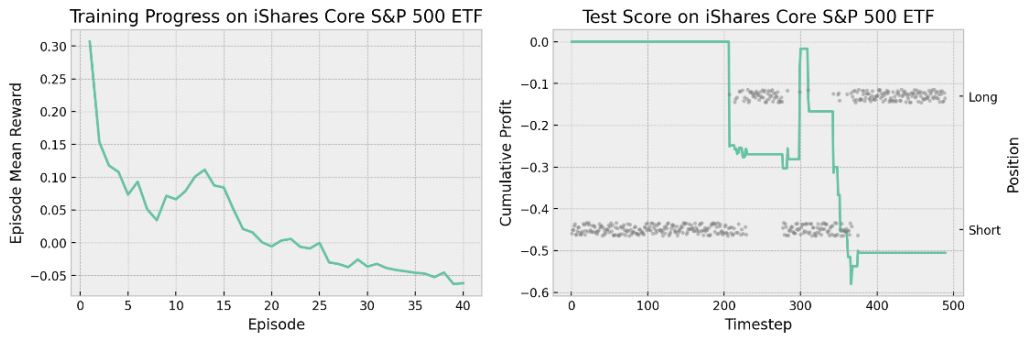

Figure 12: Training and test information for iShares Core S&P 500 ETF (DQN[16, 16], lookback 5)—own illustration

In the left figure above, for example, the episode mean reward seems to stabilize with each additional episode; however, the trend is decaying and not growing as would be desired. The trained algorithm for the same security also performs badly out of sample, as shown right, where the position at each step is also visible.

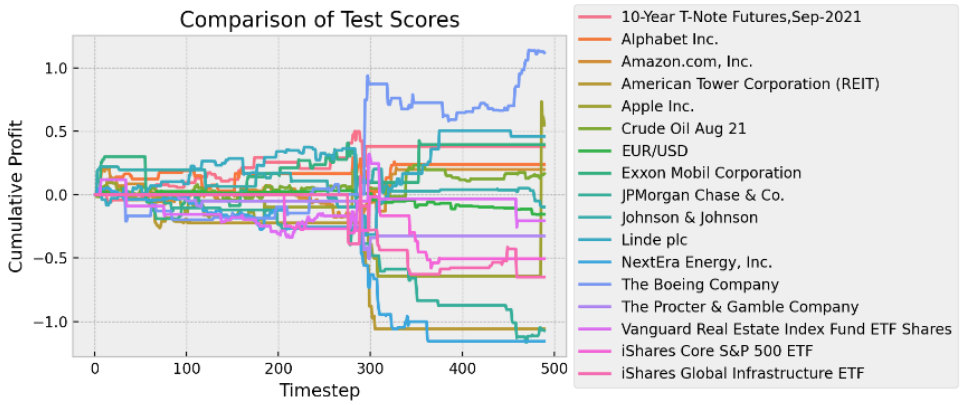

Figure 13: Out-of-sample cumulative profit for all securities (DQN[16, 16], lookback 5)—own illustration

The total cumulative loss for all securities at the end of the test episode is of 1.734.

3.2.6.1.2 10-Day Lookback Period

In the next run, the agent is able to observe the data of the previous 10 days.

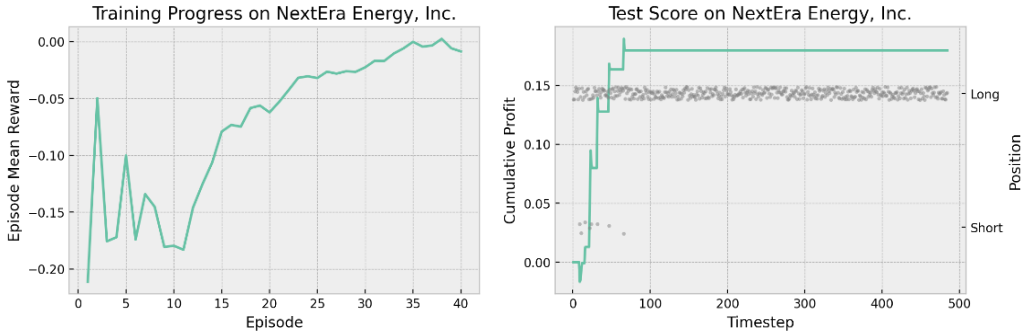

Figure 14: Training and test information for NextEra Energy, Inc. (DQN[16, 16], lookback 10)—own illustration

Figure 15 demonstrates that, for some securities, training can improve out-of-sample performance.

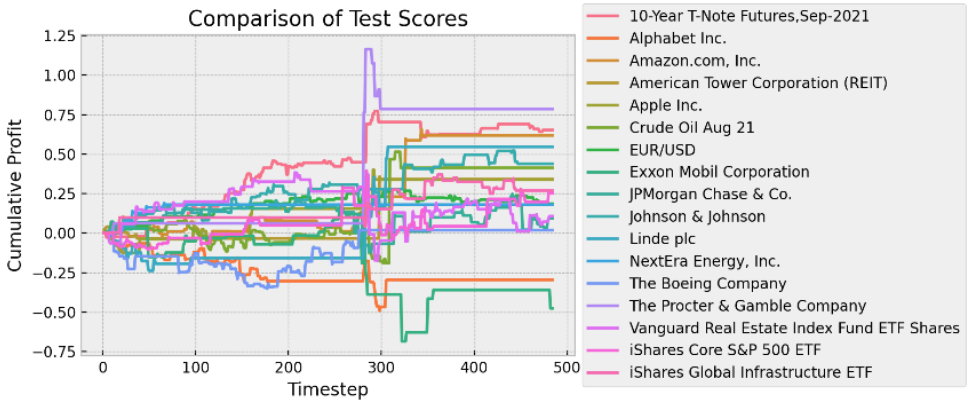

Figure 15: Out-of-sample cumulative profit for all securities (DQN[16, 16], lookback 10)—own illustration

With the longer 10-day lookback window, DQN[16, 16] is able to perform much better with a total cumulative profit of 4.466.

3.2.6.1.3 20-Day Lookback Period

In the final run for the [16, 16]-neuron DQN, 20 days of daily data are visible to the agent.

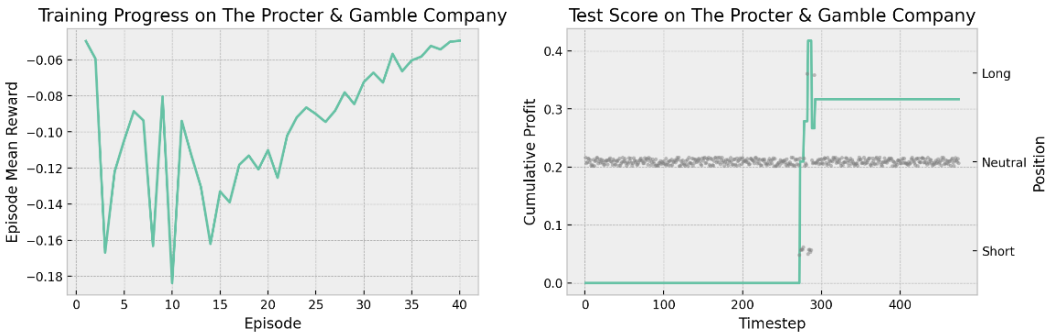

Figure 16: Training and test information for The Procter & Gamble Company (DQN[16, 16], lookback 20)—own illustration

The last figure displays an interesting case. The first half of training shows a high variance not seen with the other securities, then the improvement in score occurs. The right picture indicates that the training enabled the agent to make profits.

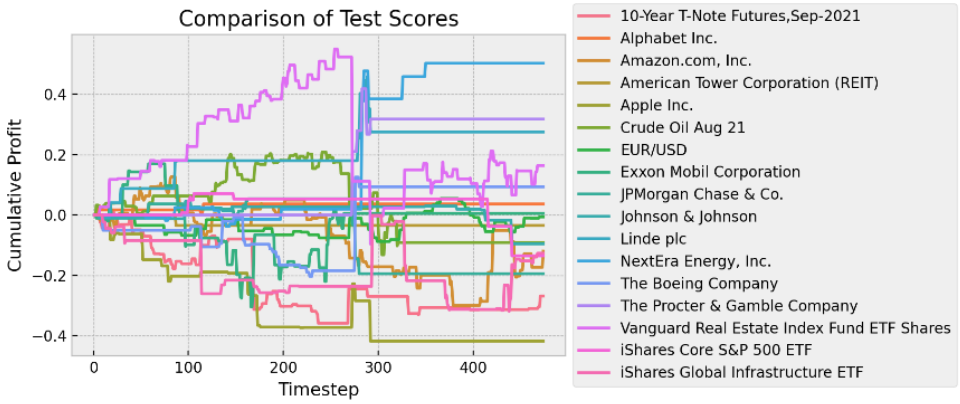

Figure 17:Out-of-sample cumulative profit for all securities (DQN[16, 16], lookback 20)—own illustration

The algorithm realizes an overall loss of 0.103.

3.2.6.2 Trading securities with DQN[32, 32]

3.2.6.2.1 5-Day Lookback Period

The same procedure outlined in the preceding subchapter is followed from here.

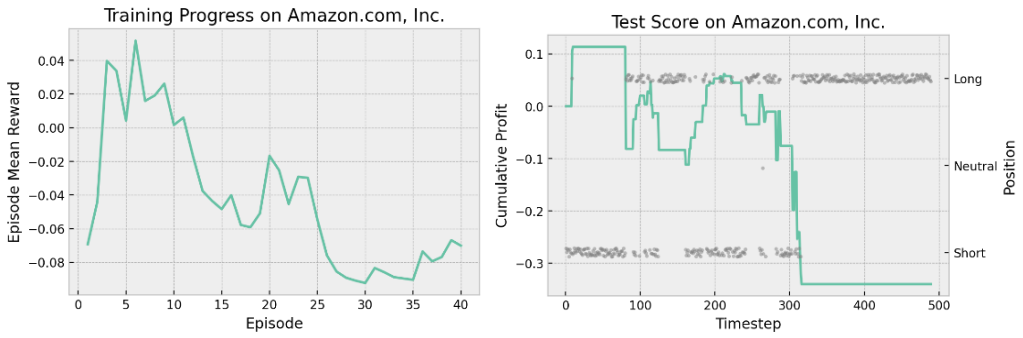

Figure 18: Training and test information for Amazon.com, Inc. (DQN[32, 32], lookback 5)—own illustration

What is rather surprising about Figure 18 is that we recall from the descriptive statistics that the underlying security for Amazon.com, Inc. has the best total returns over the entire period. This however does not seem to have been recognized by the algorithm on this occasion.

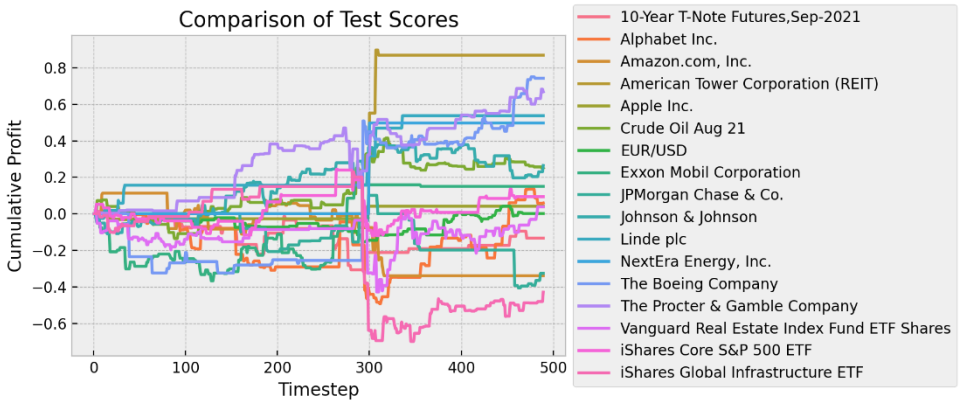

Figure 19: Out-of-sample cumulative profit for all securities (DQN[32, 32], lookback 5)—own illustration

In this case, DQN[32, 32] is able to generate an overall profit of 2.985.

3.2.6.2.2 10-Day Lookback Period

Next is a highlight from the 10-day lookback window environment. In Figure 20 below we can observe a desirable training progress obtained on the real estate security Vanguard Real Estate Index Fund ETF Shares. The positive effect of training effectively translates to the test data.

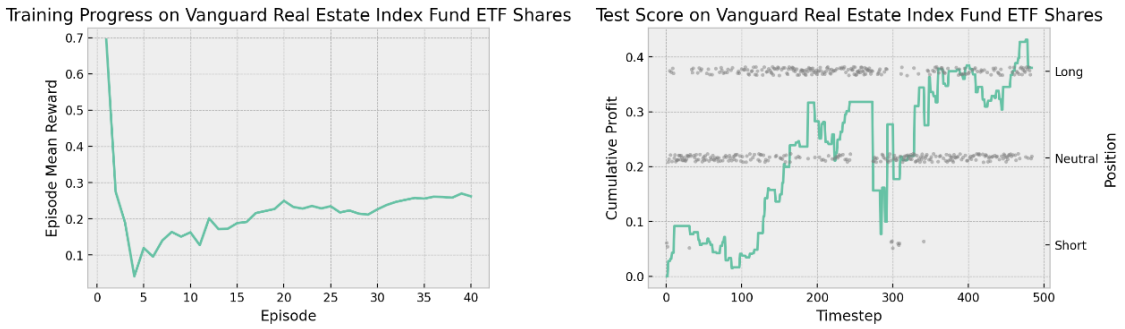

Figure 20: Training and test information for Vanguard Real Estate Index Fund ETF Shares (DQN[32, 32], lookback 10)—own illustration

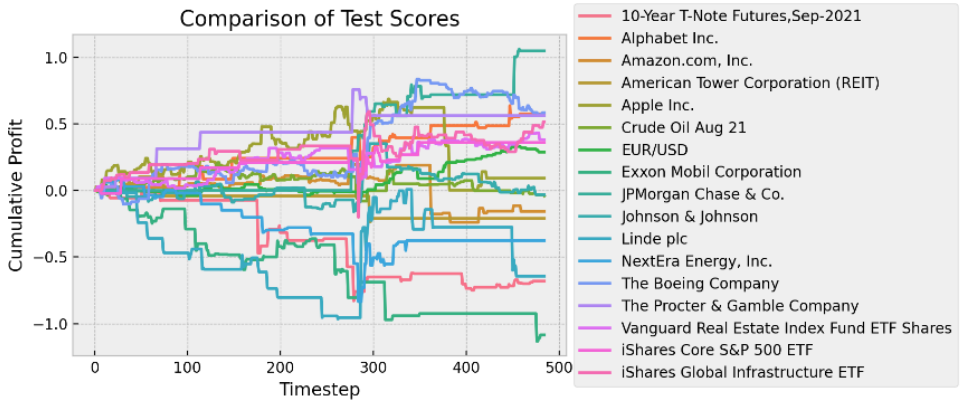

Figure 21: Out-of-sample cumulative profit for all securities (DQN[32, 32], lookback 10)—own illustration

The total profit over al securities realized by this DQN variant with a lookback window of 10 days is 1.165.

3.2.6.2.3 20-Day Lookback Period

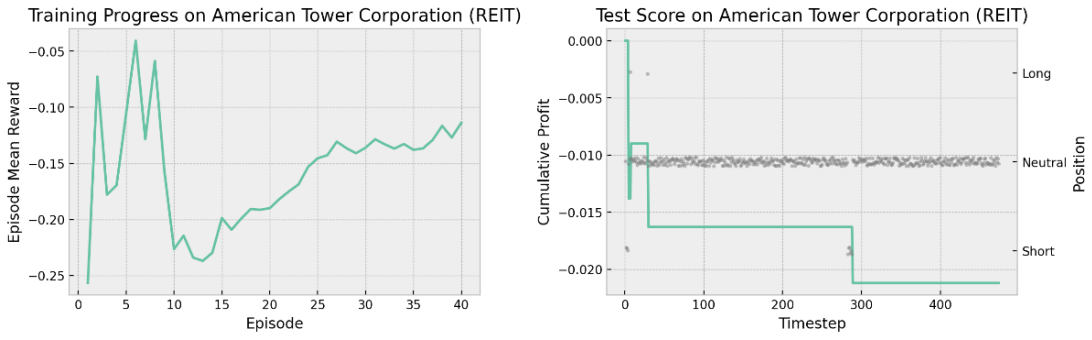

Figure 22: Training and test information for American Tower Corporation (REIT) (DQN[32, 32], lookback 20)—own illustration

The above figure is another case of large variance persisting in the model over a good part of the training phase, with a gradual improvement happening relatively late.

Unfortunately, the is not a positive transfer of the training knowledge to the out-of-sample situation.

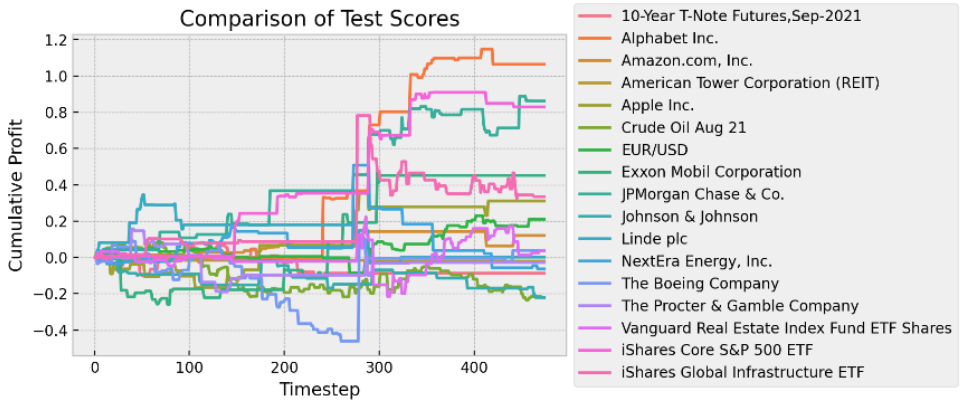

Figure 23: Out-of-sample cumulative profit for all securities (DQN[32, 32], lookback 20)—own illustration

As can be seen from the image, in this case the algorithm generates overall very good profits; the sum of cumulative profit at the end of the training episode is 3.610.

3.2.6.3 Trading securities with DQN[64, 64]

3.2.6.3.1 5-Day Lookback Period

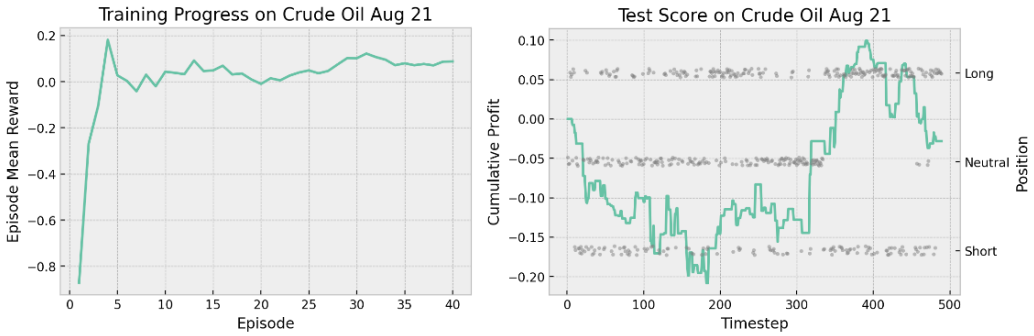

Figure 24: Training and test information for Crude Oil Aug 21 (DQN[64, 64], lookback 5)—own illustration

Training on the crude oil future in Figure 24 above displays an almost ideal shape. However, the algorithm incurs losses out of sample.

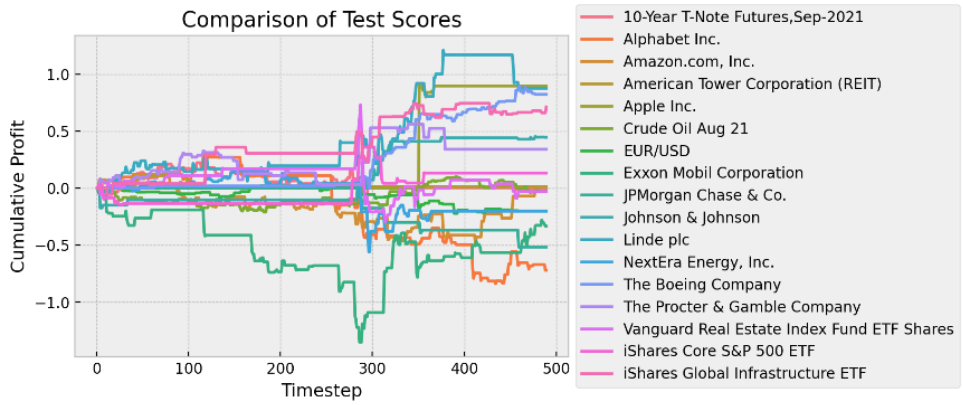

Figure 25: Out-of-sample cumulative profit for all securities (DQN[64, 64], lookback 5)—own illustration

Also in this case, the agent obtains a good result with a total cumulative profits at 2.213.

3.2.6.3.2 10-Day Lookback Period

Next is a case for the 10-day lookback window.

Figure 26: Training and test information for JPMorgan Chase & Co. (DQN[64, 64], lookback 10)—own illustration

Figure 26 shows a good training progress on the JPMorgan Chase & Co. stock. Out of sample, the agent escapes initial losses and is able to generate a small profit.

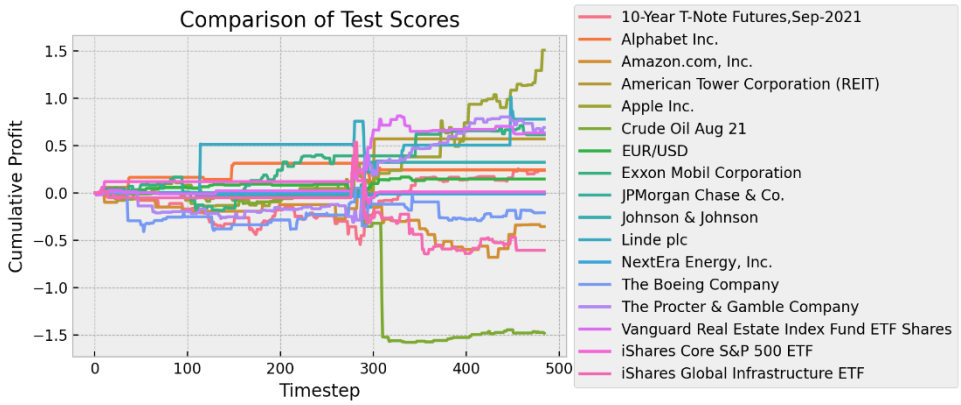

Figure 27: Out-of-sample cumulative profit for all securities (DQN[64, 64], lookback 10)—own illustration

Notwithstanding the single large loss visible in Figure 27, the algorithm generates a total profit of 3.105.

3.2.6.3.3 20-Day Lookback Period

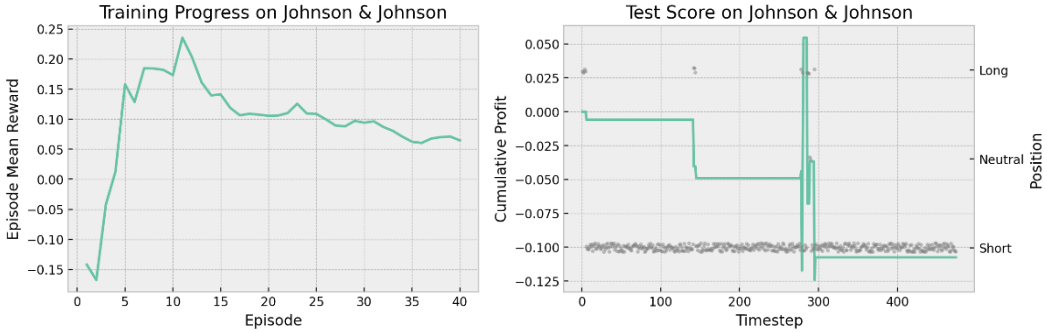

Figure 28: Training and test information for Johnson & Johnson (DQN[64, 64], lookback 20)—own illustration

Although not ideal, since the initial improvement is not sustained, Figure 28 displays some appreciable degree of convergence in training. However, the algorithm incurs losses for the same security during testing.

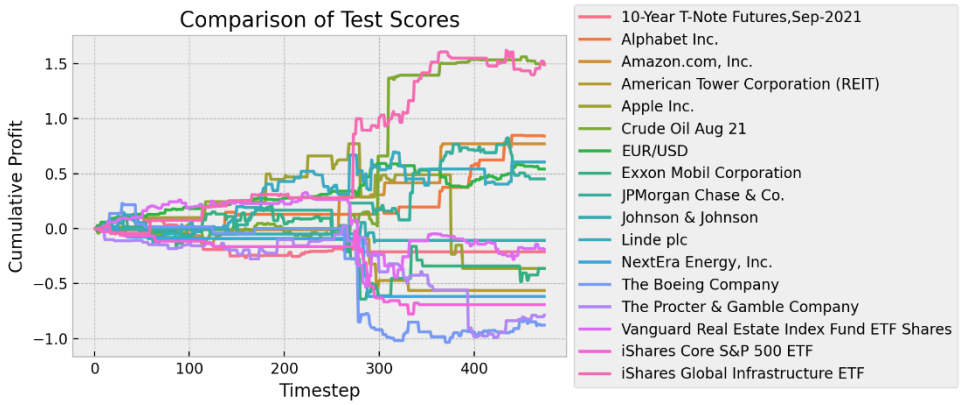

Figure 29: Out-of-sample cumulative profit for all securities (DQN[64, 64], lookback 20)—own illustration

The algorithm generates an out-of-sample profit over all securities of 1.432.

4 Discussion and Outlook

The results obtained in the empirical part of this thesis are summarized, explained, and discussed in the first part of this chapter, where the answer to research question is also stated. The second part is concerned with identifying the limitations of the study. The third part contains recommendations by the author for similar research. The fourth and conclusive part highlights the relevance of this thesis for practical application.

4.1 Summary and Discussion of Empirical Results

The following table presents a detailed view of all empirical results. Entries marked in green are the best results in the respective row.

Table 3: Empirical results by security, algorithm variant, and lookback window

	Algorithm Environment Lookback Window								
	DQN[16, 16]			**DQN[32, 32]**			**DQN[64, 64]**		
Security	**5**	**10**	**20**	**5**	**10**	**20**	**5**	**10**	**20**
10-Year T-Note Futures,Sep-2021	0.379	0.652	-0.268	-0.134	-0.681	-0.088	-0.015	0.238	-0.211
Alphabet Inc.	0.236	-0.296	0.036	0.059	0.574	1.063	-0.720	0.243	0.842
Amazon.com, Inc.	0.198	0.618	-0.120	-0.340	-0.159	0.121	0.013	-0.355	0.771
American Tower Corporation (REIT)	-1.059	0.341	-0.035	0.869	-0.211	-0.021	0.013	0.571	-0.563
Apple Inc.	0.547	0.339	-0.418	0.042	0.092	0.311	0.897	1.507	-0.363
Crude Oil Aug 21	0.165	0.413	-0.091	0.250	-0.042	-0.222	-0.028	-1.480	1.500
EUR/USD	-0.156	0.185	-0.006	0.001	0.288	0.209	-0.201	0.147	0.543
Exxon Mobil Corporation	0.394	-0.477	0.005	0.150	-1.085	0.452	-0.332	0.615	-0.363
iShares Core S&P 500 ETF	-0.506	0.252	-0.136	0.094	0.359	0.828	0.133	0.012	-0.692
iShares Global Infrastructure ETF	-0.650	0.269	-0.127	-0.430	0.515	0.335	0.712	-0.604	1.491
Johnson & Johnson	-0.097	0.438	-0.096	0.264	-0.032	-0.222	0.447	0.325	-0.107
JPMorgan Chase & Co.	-1.072	0.093	-0.195	-0.327	1.046	0.861	-0.517	0.004	0.452
Linde plc	0.458	0.546	0.275	0.538	-0.645	0.001	0.876	0.779	0.606
NextEra Energy, Inc.	-1.157	0.180	0.502	0.498	-0.378	-0.063	-0.202	-0.011	-0.617
The Boeing Company	1.118	0.019	0.093	0.743	0.583	0.038	0.823	-0.208	-0.879
The Procter & Gamble Company	-0.326	0.786	0.317	0.669	0.562	-0.028	0.342	0.690	-0.787
Vanguard Real Estate Index Fund ETF Shares	-0.206	0.107	0.163	0.038	0.380	0.036	-0.029	0.633	-0.189
Total	**-1.734**	**4.466**	**-0.103**	**2.985**	**1.165**	**3.610**	**2.213**	**3.105**	**1.432**

Before stating the answer to the research question, it can be interesting to explore the table above. As could already be inferred at the end of chapter three, almost every DQN variant is able to generate total positive cumulative profits out-of-sample with any of the three lookback window parametrizations. The only exception is DQN[16, 16], which generated both the best overall performance and the two worst losses. DQN[32, 32] and DQN[64, 64] take respectively the second and third place for single performances. If the average performance of the totals across the three environments is taken to judge each algorithm, DQN[16, 16] is actually by far the worst performer, and DQN[32, 32] the best. Although the above results do not clearly indicate that the role of neural network architecture in the DQN algorithm relates directly with the agent's cumulative profit, the overall performance from the DQN[32, 32] and DQN[64, 64] variants suggest that their learned approximations tend to have a better baseline thanks to more neurons in the artificial neural networks. This can also be seen if we count the number of green cells in the table for each algorithm, excluding the total row: while DQN[16, 16] and DQN[32, 32] each have four such entries, the remaining nine are produced by DQN[64, 64] This means that DQN[64, 64] is best-in-class for more than half of the cases when it comes to trading securities. This empirical "proof" and the fact that despite the same amount of training data received, DQN[64, 64] seems to match or even beat the performance of the two contestants under certain aspects, it seems safe to assume that, provided adequate training, more complex (perhaps not exaggeratingly so) network architectures in DQN algorithms can overall produce as good if not better results than their simpler counterparts. Another criterium that could be measured is the number of positive returns, where DQN[32, 32] wins as the "safest" algorithm, followed by DQN[16, 16]. One more consideration for the performances of the algorithms can be the variability between the various environments. In this regard, DQN[64, 64] has the most stable total performance, and DQN[32, 32] is a far second. Analyzing the role played by the lookback window does not give much insight, as there does not seem to be a strong preference common among the algorithms. An additional source for analysis are the securities: if we take the average of all columns for each algorithm, we build an idea of how difficult it was for the algorithm to interact with that security. The only commonality is Linde plc, where both DQN[16, 16], and DQN[64, 64] have their best average results.

When considering the above results, the answer to the research question "Can deep reinforcement learning methods suggest profitable decisions for trading securities?" is

yes. The positive answer however has to be balanced by the acknowledgement that, while deep reinforcement learning methods, as demonstrated here and in the literature, are able to engage profitably in securities trading, they usually do so in sufficiently simple environments where many of the notions of real-world trading, like risk management, are not realistically reproduced. To investigate whether a reinforcement learning algorithm would do well enough to be acceptable in a production role in finance, it would first have to be trained and tested in a sufficiently complex environment. The author also highlights the fact that while the totals in the table above are mostly positive, the algorithms still incur relatively large losses for single securities, which means that while the agent would to discretely well on a portfolio level, it would likely still not be very profitable with single positions.

4.2 Limitations of this Study

This thesis's investigation on whether deep reinforcement learning methods are viable for trading securities provides a good analysis and overview of some core concepts needed for applying reinforcement learning to many tasks. While the research question could be answered empirically, some relevant methodological and technical constraints can be identified.

One limitation is the fact that only DQN algorithms were implemented; other advanced algorithms exist that could be more performant or have desirable properties to tackle the problem of interacting with a trading environment. A further factor that may have reduced the potential for better empirical results is the limited way in which hyperparameter tuning was executed due to software library limitations. Another methodological constraint is the aforementioned simplification of the environment. To answer as to how concretely deep learning methods can help in trading securities, many variables have to be included in the agent's environment.

Technically, this thesis is limited by the computational resources and data available to the author. Using specialized equipment and industry-grade data sets would facilitate the training of the algorithms and in turn allow for more variety in the methods adopted.

4.3 Recommendations for Further Research

The main recommendations for further research rotate around the technical limitations. With the right resources, many different algorithms can be trained, sometimes even

parallelly, to determine what factors affect which method's performance. In this case, it would be interesting to attempt to implement state-of-the-art algorithms.

From a methodological standpoint, one key option for improving a study in deep reinforcement learning is making the environment as realistic as possible. In the case of trading securities, this could mean including notions of real finance professionals into the agent's environment. Some of these notions are risk management, liquidity concerns, transaction costs, the effect that a trader has on the market, and managing resources in the context of a portfolio.

4.4 Implications for Practice

Deep reinforcement learning has a multitude of application and perhaps some are yet to be discovered. Nonetheless, the findings made in this thesis confirm the utility that deep reinforcement learning can have in finance and specifically with applications to trading. At least one such concrete application has already been investigated by IBM (Srinivasan, 2018). One hypothetical use case would be to feed the agent's actions to a broader system that incorporates many inputs, such as centralized risk management system, to try to gauge how some financial industry entities would act. Another way of bringing deep reinforcement learning to use in the financial industry could be as bots used in simulated applications to improve employees' skills.

5 References

Apache Software Foundation. (2019). *Python bindings*. Retrieved from Apache Arrow: https://arrow.apache.org/docs/python/index.html

Bhatt, S. (2018). *Reinforcement Learning 101. Learn the essentials of Reinforcement Learning!* Retrieved from Towards Data Science: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

Bishop, C. M. (2006). *Pattern Recognition andMachine Learning.* New York: Springer Science+Business Media, LLC.

Chao, Y., Jiming, L., & Nemati, S. (2020). *Reinforcement Learning in Healthcare: A Survey.*

Chollet, F. (2018). *Deep Learningwith Python.* Shelter Island: Manning Publications Co.

DeepMind. (n.d.). *AlphaGo*. Retrieved from DeepMind: https://deepmind.com/research/case-studies/alphago-the-story-so-far

Graesser, L., & Wah Loon, K. (2020). *Foundations of Deep Reinforcement Learning: Theory and Practice in Python.* Upper Saddle River: Pearson Education, Inc.

McFarlane, G. (2021, June 21). *The S&P 500: The Index You Need To Know*. Retrieved from Investopedia: https://www.investopedia.com/articles/investing/090414/sp-500-index-you-need-know.asp

MSCI Inc. (2021). *The Global Industry Classification Standard (GICS®)*. Retrieved from MSCI: https://www.msci.com/gics

Nguyen, C. N., & Zeigerman, O. (2018). *Machine Learning – kurz & gut.* Heidelberg: dpunkt.

pandas. (2021). *About pandas*. Retrieved from pydata: https://pandas.pydata.org/about/

Raffin, A. (2021). *RL Baselines3 Zoo: A Training Framework for Stable Baselines3 Reinforcement Learning Agents*. Retrieved from GitHub: https://github.com/DLR-RM/rl-baselines3-zoo#hyperparameter-tuning

S&P Dow Jones Indices. (2021). *S&P 500®*. Retrieved from S&P Dow Jones Indices: https://www.spglobal.com/spdji/en/indices/equity/sp-500/

Skansi, S. (2018). *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence.* Cham: Springer International Publishing AG.

Srinivasan, A. (2018, July 26). *Reinforcement Learning: The Business Use Case, Part 2*. Retrieved from Medium: https://medium.com/ibm-data-ai/reinforcement-learning-the-business-use-case-part-2-c175740999

Stable Baselines3. (2020). *Reinforcement Learning Tips and Tricks*. Retrieved from Stable Baselines3: https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html

Stable Baselines3. (2020). *RL Algorithms*. Retrieved from Stable Baselines3: https://stable-baselines3.readthedocs.io/en/master/guide/algos.html

Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations. (2020). Retrieved from Stable-Baselines3: https://stable-baselines3.readthedocs.io/en/master/

Sugiyama, M. (2015). *Statistical Reinforcement Learning.* Boca Raton: CRC Press.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* Cambridge: The MIT Press.

SwingTradeSystems.com. (2021, June 20). *2021 Trading Days Calendar*. Retrieved from SwingTradeSystems.com: http://www.swingtradesystems.com/trading-days-calendars.html

The Matplotlib development team. (2021). *Matplotlib: Visualization with Python*. Retrieved from matplotlib: https://matplotlib.org/

The NumPy community. (2021). *What is NumPy?* Retrieved from NumPy: https://numpy.org/doc/stable/user/whatisnumpy.html

Waskom, M. (2020). *seaborn: statistical data visualization*. Retrieved from pydata: https://seaborn.pydata.org/

Zhang, Z., Zohren, S., & Roberts, S. (2020). Deep Reinforcement Learning for Trading. *The Journal of Financial Data Science*, 25-40.