INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA DE INTELIGÊNCIA ARTIFICIAL




Portfolio Allocation using Deep Reinforcement Learning

JOSÉ ALEXANDRE QUINTELA DA SILVA
Outubro de 2022

P.PORTO

Portfolio Allocation using Deep Reinforcement Learning

Cryptocurrency trading as a Markov Decision Process

José Alexandre Quintela da Silva

1150371

Dissertação para obtenção do Grau de

Mestre em Engenharia de Inteligência Artificial

Orientador: Dra. Isabel Praça, Professora Coordenadora do Instituto Superior de Engenharia do Instituto Politécnico do Porto

Júri
Presidente:

Doutora Maria Goreti Carvalho Marreiros, Professora Coordenadora com Agregação do Instituto Superior de Engenharia do Instituto Politécnico do Porto

Vogais:

Doutor Tiago Manuel Campelos Ferreira Pinto, Professor Adjunto Convidado do Instituto Superior de Engenharia do Instituto Politécnico do Porto e Professor Auxiliar da Universidade de Trás-os-Montes e Alto Douro

Doutora Isabel Cecília Correia da Silva Praça Gomes Pereira, Professora Coordenadora do Instituto Superior de Engenharia do Instituto Politécnico do Porto

Porto, Outubro 2022

Acknoledgements

At the end of this longed-for stage of training, only possible with the support, affection, and resilience of several people with whom I had the privilege of meeting along my journey, if merit is manifested in it, it is only fair that it be shared. With all those who, in one way or another, have taught that success sometimes owes less to ability than to enthusiasm, a teaching transmitted along this path by a group of people to whom a heartfelt gratitude is required.

Thus, I would like to thank the entire faculty of the Artificial Intelligence Engineering Master for availability, opportunities and support shown during training.

To Dr. Isabel Praça, advisor, for her constant concern, for her repeated availability, who helped me in crucial moments.

Finally, a special thanks, because marked by naturally stronger and older ties, to the family and friends, for those who put complete certainty in this path of realization, accompanying each stage with renewed confidence, while constituting a stronghold of the stability needed to overcome the vicissitudes to achieve the goals pursued. THANK YOU SO MUCH!

Resumo

A gestão de portefólio é um problema em que, em vez de olhar para ativos únicos, o objetivo é olhar para um portefólio ou um conjunto de ativos como um todo. O objetivo é ter o melhor portefólio, a cada momento, enquanto tenta maximizar os lucros no final de uma sessão de trading. Esta tese aborda esta problemática, empregando algoritmos de Deep Reinforcement Learning, num ambiente que simula uma sessão de trading. É também apresentada a implementação desta metodologia proposta, aplicada a 11 criptomoedas e cinco algoritmos DRL. Foram avaliados três tipos de condições de mercado: tendência de alta, tendência de baixa e lateralização. Cada condição de mercado em cada algoritmo foi avaliada, usando três funções de recompensa diferentes, no ambiente de negociação, e todos os diferentes cenários foram testados contra as estratégias de gestão de portefólio clássicas, como seguir o vencedor, seguir o perdedor e portefólios igualmente distribuídos. Assim, esta estratégia foi o benchmark mais performativo e os modelos que produziram os melhores resultados tiveram uma abordagem semelhante, diversificar e segurar. Deep Deterministic Policy Gradient apresentou-se como o algoritmo mais estável, junto com seu algoritmo de extensão, Twin Delayed Deep Deterministic Policy Gradient. Proximal Policy Optimization foi o único algoritmo que não conseguiu produzir resultados decentes ao comparar com as estratégias de benchmark e outros algoritmos de Deep Reinforcement Learning.

Palavras-chave: Deep Reinforcement Learning, Machine Learning, Gestão de portfolios, Trading, Criptomoedas

Abstract

The problem with portfolio management is that, instead of looking at single assets, the goal is to look at a portfolio or a set of assets as a whole. The objective is to have the best portfolio at each given time while trying to maximize profits at the end of a trading session. This thesis addresses this issue by employing the **Deep Reinforcement Learning** algorithms in a cryptocurrency trading environment which simulates a trading session. It is also presented the implementation of this proposed methodology applied to 11 cryptocurrencies and five Deep Reinforcement Learning algorithms. Three types of market conditions were evaluated namely, up trending or bullish, down trending or bearish, and lateralization or sideways. Each market condition in each algorithm was evaluated using three different reward functions in the trading environment and all different scenarios were back tested against old school portfolio management strategies such as following-the-winner, following-the-loser, and equally weighted portfolios. The results seem to indicate that an equally-weighted portfolio is an hard to beat strategy in all market conditions. This strategy was the most performative benchmark and the models that produced the best results had a similar approach, diversify and hold. Deep Deterministic Policy Gradient presented itself to be the most stable algorithm along with its extension algorithm, Twin Delayed Deep Deterministic Policy Gradient. Proximal Policy Optimization was the only algorithm that could not produce decent results when comparing with the benchmark strategies and other Deep Reinforcement Learning algorithms.

Keywords: Portfolio management, Deep Reinforcement Learning (DRL), Cryptocurrency, trading

Agradecimentos

Ao terminar esta almejada etapa da formação, só possível com o apoio, o carinho e resiliência por parte de várias pessoas com quem tive o privilégio de me cruzar ao longo da minha caminhada, se mérito nela se manifesta, é justo que a mesma seja repartida com todos aqueles que, de uma forma ou de outra, ensinaram que, por vezes, o sucesso deve menos à capacidade do que ao entusiasmo, ensinamento transmitido, ao longo deste percurso, por um conjunto de pessoas a quem se impõe um sentido agradecimento.

Assim, agradeço a todo o corpo de docentes que constitui o Mestrado de Engenharia de Inteligência Artificial a disponibilidade, pelas oportunidades e pelo apoio demonstrados durante a formação.

À Doutora Isabel Praça, orientadora da Dissertação, pela constante preocupação, pela reiterada disponibilidade com que me ajudou em momentos cruciais.

Por último, um agradecimento especial, porque marcado por laços naturalmente mais vincados e antigos, à família e amigos, para os que depositaram a plena certeza neste percurso de realização, acompanhando cada etapa com confiança renovada, ao mesmo tempo que se constituíam reduto da estabilidade necessária para ultrapassar as vicissitudes, para alcançar os objetivos perseguidos. MUITO OBRIGADO!

Index

List of Figures

List of Tables

Abreviations

List of Abbreviations

A2C	*Advantage*
ANN	Artificial Neural Networks
DDPG	Deep Deterministic Policy Gradient
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
MLP	Multi-Layer Perceptron
PPO	*Proximal Policy Optimization*
RL	Reinforcement Learning
SAC	*Soft Actor Critic*
TD3	Twin Delayed Deep Deterministic Policy Gradient

1 Introduction

1.1 Context

The financial community has long been interested in predicting and analysing financial economic indicators, but recently the machine learning community (ML) has shown great interest in using and benefiting from more advanced techniques, such as deep networks and deep reinforcement learning. In addition, the cryptocurrency industry has gained popularity worldwide, with Bitcoin making headlines almost weekly.

When the concept and invention of Bitcoin were announced in 2009 [1] there were many doubts, but also a lot of enthusiasm for this new kind of cash that you could not touch. Thousands of new cryptocurrencies later, the obscurity about this virtual money has faded and been replaced by the desire to become a whale, whether through the tweets of Elon Musk or local governments quickly trying to control these unregulated forms of investment. Bitcoin's popularity and profit margins are increasing exponentially. Unlike traditional stock markets, the Bitcoin market and its exchanges are not controlled by a central authority. This, combined with the fact that trading on these platforms involves very low entry criteria, makes it a particularly attractive investment opportunity for people who are new to trading and not yet ready to invest large sums of money. This means that a smart agent can make a lot of money, but a speculative trader can also lose a lot of money. Moreover, the market capitalization is almost constant, which indicates that new players are entering and actively investing in the crypto marketplaces.

Every cryptocurrency investor faces an internal dilemma every day. Is it better to HODL or not? HODL stands for "Hold on to Dear Life" and refers to the practice of holding crypto assets for a long period of time. For some, holding on has led to tenfold gains, while for others it's just been a waiting game. Some investors may hit the jackpot by quickly engaging in pump-and-dump methods where the price of a coin is driven up primarily by trading robots, but there has to be a more effective way to play in this highly volatile but extremely rewarding market. This leads to the problem of portfolio management: when dealing with many assets, the

problem of portfolio management arises. An investment portfolio is a collection of assets that includes stocks, bonds, cash, and other financial instruments such as cryptocurrencies. In today's market, a balanced portfolio is critical to an investor's success. When a certain number of resources need to be allocated among a variety of simultaneous assets, we need to choose how to allocate those resources. Originally, crypto markets were highly correlated (when Bitcoin went up, most other cryptocurrencies went up as well, so the difference did not matter much), but recently it has been shown that the correlation between different pairs is decreasing significantly, leading to a greater need for dynamic trading. Due to the obvious 24/7 nature of the cryptocurrency market, actions taken during trading often have long-term consequences that cannot be quantified immediately. Even when we are asleep, we need someone or something to act on our behalf. Deep reinforcement learning uses deep learning to help an agent learn how to approximate value or policy functions, which in turn helps the agent learn how to meet its goals. These algorithms are often diverse to be applicable to a wide range of contexts and, in this example, to commercial enterprises. This paper presents a new method for building a multi-asset portfolio trading strategy that uses Deep Reinforcement Learning (DRL) to address these limitations. In this paper, we consider a multi-asset trading strategy and create a simple set of trading actions that traders can use as direct investment guidance or for automated trading.

1.2 Research Questions

Algorithmic trading has promised a lot a people free money without requiring any skill. Some say that even technical analysis is like astrology and many of the presented trading bots present fake results such as 50%+ in 6 months or +75% a year, hiding the bad results. Some algorithms might even outperform the market with some luck but can’t keep that performance over a medium-long term. The truth is if any algorithm can constantly have these results, in some years, anyone can become richer than Elon Musk or Warren Buffet. If someone invested 10 000 USD and constantly gained 75% a year, in 5 years it would result in 93000 USD, almost a 10x of the initial investment, and in 20 years it would be more than 400 million USD. This section has the goal of exposing the questions that lead to development of this project.

Q1: What strategy is the most reliable to manage a portfolio?

To address Q1 we are going to compare the performance of old-school portfolio management strategies to the ones produced by DRL algorithms.

Q2: How different Deep Reinforcement Learning algorithms perform on various market conditions?

This question has the object of assessing if the algorithms are stable across various market conditions.

Q3: What is the impact of having suggestive reward functions?

In Q3 we are going to analyse how models behave across different goals. For example, if it is better to trying to maximize profits or simply outperforming Bitcoin.

Q4: Is cryptocurrency algorithmic trading a fallacy?

To answer Q4 we are going to compare the results of the algorithms and see if an honest trading bot can outperform the market.

1.3 Contribution

The development of Deep reinforcement learning for portfolio management contributes with the following:

C1: Inform how different trading strategies perform across different market conditions

C2: A literature review of state-of-art DRL models.

C3: Provide an overview of the behaviour of different DRL algorithms

C4: Provide a comparison between different types of reward functions and how reward engineering can be useful.

C5: Provide an overview of the portfolio management problem and its constituents.

1.4 Report Structure

A brief outline of the project structure and chapters is provided below:

Chapter 2: Literature Review: This chapter introduces specific terms related to portfolio management that are necessary to understand the following sections in Subchapter 2.1. Subchapter 2.2 analyses related work that inspired this thesis. Sub-chapter 2.3 is a general explanation on how reinforcement learning works. Sub-chapter 2.4 introduces Neural Networks, and the final sub-chapter (2.5) explains all the Deep Reinforcement Learning algorithms used.

Chapter 3: Materials and Methods: This chapter has the objective of combining the portfolio management problem with deep reinforcement learning. Starts by introducing several portfolio management strategies and technical indicators. Sub-chapter 3.3 explains in a mathematical standpoint how portfolio management can be achieved through reinforcement learning. The next sub-chapter presents the architecture for the proposed methodology. In sub-chapter 3.4 we expose the data used in this project, how the data was processed and split. Subchapter 3.8 presents the performance of the benchmark strategies introduced in 3.1. The final sub-chapter lists all the tools and relevant libraries used.

Chapter 4: Case Studies description and Analysis: In this chapter we benchmark and discuss the results of each algorithm for three case studies.

Chapter 5: Conclusions: In this chapter we reflect about the overall results of this project including its contributions and limitations.

Chapter 6: Future Work: In this final chapter we discuss the future roadmap of this project

2 Literature Review

2.1 Financial Terms and Concepts

Financial engineering is the application of mathematical methods to solve financial problems. It also known as financial mathematics, mathematical finance, and computational finance. The authors in [2] stated that the process of financial engineering could be described in several ways:

- the creation of a financial product ab initio that yields a specific financial return over a period of time
- the "fine-tuning" of an existing financial product to improve its return or risk characteristics in light of changing market conditions.
- a process to revise and restructure existing financial products to take advantage of changing tax, legal or general economic conditions.

2.1.1 Asset

Any asset owned by an entity and reported on its balance sheet. Examples of assets include cash (held in an account or at a bank), inventories, loans, and advances, accrued assets, and other assets. This report primarily addresses cryptocurrencies, but the general principles apply to other types of assets as well.

2.1.2 Cryptocurrency

A cryptocurrency is a digital or virtual currency that is protected by encryption, making counterfeiting or double-spending practically impossible. Many cryptocurrencies are decentralized networks built on blockchain technology, which is a distributed ledger enforced by a network of computers. Cryptocurrencies are distinguished by the fact that they are 6 generally not issued by any central authority, making them potentially impervious to government meddling or manipulation.

2.1.3 Portfolio

A portfolio is a collection of financial assets such as stocks, bonds, commodities, currencies, and cash equivalents, and their counterparts in the form of open-end, exchange-traded, and closed-end funds. Non-publicly traded securities such as real estate, art, and private investments may also be included in a portfolio. Portfolios are held directly by investors and/or managed by financial specialists and asset managers. Investors should construct an investment portfolio based on their risk tolerance and investment objectives. Investors may

also maintain multiple portfolios for different purposes. It all depends on one's investment goals.

2.1.4 Volume

The number of shares or contracts traded in a security or market in a given period is called volume. For each customer, there is one seller, and each transaction is added to the total volume. That is, it is considered a transaction when the buyer and seller agree to make a transaction at a certain price. The daily volume is five if there are only five transactions in a day.

2.1.5 Return

Investors do not benefit directly from absolute asset prices. Price trends over time, on the other hand, are extremely important because they show the profit and loss of an investment or, in other words, its return.

2.1.5.1 Simple Return

The additional net income anticipated from a potential investment opportunity is divided by the amount invested to arrive at the simple rate of return. To determine whether a business should invest in a fixed asset and any related incremental change in working capital, the simple rate of return is employed in capital planning.

2.1.5.2 Gross Return

The overall rate of return on an investment before any fees, commissions, or costs are subtracted is called the gross rate of return. A month, quarter, or year is used as the unit of measurement for the gross rate of return. On the other hand, the net rate of return eliminates fees and charges from the calculation to provide a truer picture of return.

2.1.5.3 Log Return

The logarithmic return is a way of calculating the rate of return on an investment. To calculate it you need the initial value of the investment Vi, the final valueVf and the number of time periods t. You then take the natural logarithm of Vf divided by Vi, and divide the result by t:

$$\mathrm{R} = \ln(\mathrm{VfVi})\mathrm{t} \times 100\%\mathrm{R} = \ln(\mathrm{VfVi})\mathrm{t} \times 100\%$$

This value is normally expressed as a percentage, so you also multiply by 100. The calculated rate will depend on the value of *t* that you use. If *t* is the number of years, then you get an annual rate. This then gives you the continuously compounded annual interest rate that you would need to receive to match the return on this investment.

2.1.6 Transaction Cost

Transaction costs are the costs incurred to purchase or sell a product or service. Transaction costs are the labour required to bring a good or service to market, giving rise to an entire industry dedicated to facilitating exchange. From a financial standpoint, transaction costs include brokerage fees and spreads, which is the difference between the price the dealer pays for the security and the price the buyer pays.

2.1.7 Portfolio Value

The total monetary value of the portfolio obtained by multiplying the weights of the assets with the daily prices.

2.2 Related Work

Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) are commonly used in dynamic portfolio optimization nowadays. In the literature, there are a rising number of published works. This section examines existing techniques of both value-based and policy-based model-free RL for the portfolio optimization issue, outlining some important unanswered problems and difficulties facing today's portfolio managers for a review of RL and deep RL approaches.

The authors in [3] present an extensible reinforcement-learning framework solving the general financial portfolio management problem. The purpose of this system is to deal with multi-channel market inputs and immediately output portfolio weights as market actions. It can be used with a variety of deep neural networks and is linearly scalable as the portfolio size grows. The framework was built utilizing three distinct underpinning networks in the experiments: a CNN, a simple RNN, and an LSTM. In contrast to other trading algorithms, all three versions outperformed the others in terms of final cumulative portfolio value. A novel extension to reinforcement learning in this problem is presented in [4], where it’s presented a generic state augmented RL framework that can integrate heterogeneous data sources into standard RL training pipelines for learning portfolio management strategies.

With the objective of improving tradition reinforcement algorithms the authors in [5] proposed a new portfolio management that uses the Robust Log-Optimal Strategy (RLOS). The Robust Log-Optimal Strategy (RLOS) is a novel portfolio management strategy that improves on the General Log-Optimal Strategy (GLOS) by approximating the standard reward function using quadratic Taylor expansion. It eliminates GLOS's complicated estimate method, therefore avoiding the "Butterfly Effect" produced by estimation mistake. According to the authors on multiple back tests, the RLOSRL's performance is compared to that of several classic strategies, in which the authors chose a random sample of CSI300 index member stocks as assets under management and the test results support its profitability and stability.

A new approach to the above papers is presented in [6] using Deep Q-Learning techniques (DQN), combining deep learning with reinforcement learning. The proposed general framework is composed by two main ingredients: A set of local agents that are responsible for trading the single asset in the portfolio and a global agent that rewards each local agent based on the same reward function (global reward). The local agent was based on three different deep RL approaches: deep Q-learning, double deep Q-learning and duelling double deep Q-learning. All frameworks in this paper were tested on a crypto portfolio composed by four assets (currency): Bitcoin (BTC), Litecoin (LTC), Ethereum (ETH) and Ripple (XRP). ROI (on average) is always above the 2% considering an initial investment of 10,000 USD dollars. The authors backed by the empirical study suggest that, while considering a D-DQN as local agent, the most suitable global reward function is a linear combination of Sharpe ratio and portfolio net return. The portfolio management framework based on deep Q network achieved the best value in terms of average daily returns (4.67%) with the DQN-RF2 model and it was compared with other two alternative approaches: an equally weighted portfolio technique (EW-P) and portfolio management technique based on GA optimization (PS-GA).

A deterministic deep reinforcement learning technique for tackling the portfolio management problem is provided [7] , which constructs the portfolio vector directly using raw market data and previous prices as input. The authors claim that this technique is very extendable and does not rely on any financial theory. On a cryptocurrency market, a back-test experiment is conducted. The CNN approach is compared to three benchmarks and three different portfolio management strategies and proved to be profitable.

An open-source framework named FINRL is proposed in [8]. The goal of this study is to use a multi-agent platform based on Deep Deterministic Policy Gradient (DDGP) and back tested against existing DRL techniques such as DQN to help quantitative traders overcome the steep learning curve. The DDGP turned out to be successful, with an annual percent return (APR) of 36%.

A Deep Reinforcement Learning Approach for Automated Cryptocurrency Trading is presented in [9], in which multiple Deep Reinforcement Learning trading algorithms were evaluated on hourly cryptocurrency values. Double and Duelling Double Deep Q-learning Networks are used in the trading systems. They were each put to the test with two distinct reward functions. The first function was based on the Sharpe ratio, which is a measure of an investment's risk adjusted return, while the second was tied to profit. The trading systems based on Double Q-learning and the Sharpe ratio reward function (SharpeD-DQN) obtained higher return values, according to the findings. SharpeD-DQN was also evaluated across the whole period under consideration, yielding a positive percentage return (8 percent average return).

In [10] is developed a digital currency portfolio trading system in which deep networks are used to optimize virtual currency portfolio management. Deep-learning networks and reinforcement-learning networks are combined in the suggested strategy. Its goal is to grow total assets by buying or selling appropriate currencies. Furthermore, the strategy not only raises asset value but also regulates risk, may significantly boost average return, and it is

resistant to long-term risk. The results of the experiments show that this method may be employed for long-term investing in the virtual currency market.

A cryptocurrency recommendation system is presented in [11] based on a deep reinforcement learning approach. The framework based on historical price movements, acts on real-time and it is tested on three cryptocurrencies namely, Bitcoin (BTC), Litecoin (LTC), and Ethereum (ETH). The experiment on Bitcoin via DRL application shows that the investor got 14.4% net profits within one month. Similarly, tests on Litecoin and Ethereum also finished with 74% and 41% profit, respectively.

Another DQN approach is introduced in [12] with the goal of designing long-short trading strategies for futures contracts. The state space in this paper is made up of volatility-normalized daily returns, with buying or selling serving as the reinforcement learning action and the total reward being the sum of our actions' gains. According to the authors, the results were unsatisfactory and required more development, but one essential point jumps out: choosing the correct generator for (fake) financial market data to train the RL agent is critical. In [13] a similar study that applies DQN in the PM problem. The tested DQN agent achieved an average annual portfolio return of 65.98% %, although it showed extreme volatility over the 2,000 runs. Despite the high volatility of deep reinforcement learning, the experiment demonstrated that it has exceptionally high potential to be employed and provides a solid foundation on which to build further research.

A novel cost-sensitive portfolio policy network is given in [14] to tackle the financial portfolio management challenge. The suggested network can extract both price sequential patterns and asset correlations by designing a novel two-stream architecture. In addition, the authors built a novel cost-sensitive reward function and used the direct policy gradient technique to optimize it to maximize the accumulated return while managing both transaction and risk costs.

[15] presents a portfolio management approach based on adaptive sampling. According to the authors, the agent may learn the general trading technique more efficiently in a short amount of time using this method. According to the results of the experiment, the sampling strategy model outperforms the random learning approach. The Sharpe ratio is raised by 6% to 7%. The experiment's results show that the suggested learning framework, together with the sampling technique, is effective in establishing accurate trading rules.

A portfolio management system, called PMS designed to support human decision-making is presented in [16] with a focus on equity market neutral (EMN). EMN is a hedging method with risk management capabilities that works by balancing long and short positions, EMN can control risk and decrease investment risk. In this work, two NN models are combined with the planned RL system to acquire spatial and temporal information, and two reward functions are constructed to maximise profitability while taking risk into account. According to the findings of the experiments, the created PMS with Sharpe ratio reward function outperforms the reward function with trading return, increasing the return by 39.0 percent, and lowering the drawdown by 13.7 percent on average.

A Deep Deterministic Policy Gradient-based (DDPG) is studied in [17] where a novel DDPG strategy makes profits in a portfolio set with 4 low volatility U.S stocks and 4 high volatility U.S stocks. This system was benchmarked according to Compound Annual Return Rate, Sharpe Ratio, and Maximum Drawdown performance metrics. In our experiment, the Cost DDPG strategy outperforms seven other traditional strategies in all three-performance metrics. We also compare the performance of our DDPG strategy with and without transaction cost and provide a plausible explanation to the result that DDPG strategy with transaction cost is more profitable and more robust than the Cost-free strategy.

A framework for implementing DRL algorithms in the context of high frequency trading is proposed in [18]. The findings of this study demonstrated that a DRL agent may perform effectively in a highly stochastic and nonstationary environment. The capacity to generate successful strategies out of the blue demonstrates that market inefficiencies exist and may be exploited to generate long-term profits, and therefore contributes to the growing body of evidence refuting the long-held notion that markets are innately efficient.

2.3 Reinforcement Learning Problem

First books on Markov Decision Processes are Bellman [19] and Howard [20]. The term 'Markov Decision Process' has been coined by Bellman in 1954 [21]. Shapley [22] was the first study of Markov Decision Processes in the context of stochastic games. Mathematical rigorous treatments of this optimization theory were presented by Blackwell [23], Bertsekas and Shreve [24] and Dynkin and Yushkevich [25] .More recent textbooks on this topic are Puterman [26], Hernandez-Lerma and Lasserre [27] , Powell [28] and more recently Sutton & Barto [29]. The following chapter is based on these authors.

2.3.1 Common elements to all control tasks

The first element common to all control tasks is the **State (St).** The state is all the relevant information that describes the situation the task environment is currently in. For example, in a chess match the state is the location of the pieces on the board and the remaining time for each player. A state is always linked to a moment in time in which we refer the state in time *t* (St).

The second element we find in all control tasks are the **Actions (At)** that are carried out during the task. Actions are moves that an agent can perform at each moment in time. For example, in the game of chess the actions are every legal move that a player can make. Actions are also tied to a moment in time and are chosen based on the state of the task. By that, we mean that the agent observes the states and based on the characteristics of the state in which the task finds itself, it will take some actions or others.

The third element is the **Reward (Rt),** which is the mechanism that informs the agents about the effectiveness of its decision making. It is a numerical value that an agent receives after

carrying out an action and it determines the immediate effect of having carried out that action. For example, in the game of chess, where the goal is to checkmate an opponent if we perform an action that creates a state where the opponent is in check and has no legal moves then its reward must be the highest possible. Contrarily, if an action that hangs mate in one is performed the reward must be the lowest possible.

The fourth element is the **Agent.** An agent is an entity that participates in a control task by observing the state and carrying out actions at each time step. In chess, the agent can be a human being who observes the state of the task with his eyes and takes the necessary actions based on it.

The final element is the **Environment,** which comprises all the aspects of the task that the agent doesn't control fully. In the game of chess, the time is part of the environment since a player cannot decide to increase it arbitrarily. Also, the opponent's moves are part of the environment and the rules that state a move as valid or not. For example, a rook cannot move diagonally, and a bishop cannot move perpendicularly.

2.3.2 Markov Decision Process (MDP)

A Markov Decision Process is a discrete time stochastic control process. It is a control process because it consists of a decision process in which the objective is to solve a given task according to a certain reward function. It is a stochastic process because the agent's actions affect only partially the evolution of the task. And finally, it is a discrete time process because time progresses in finite time intervals. The diagram below represents the Markov Decision Process.

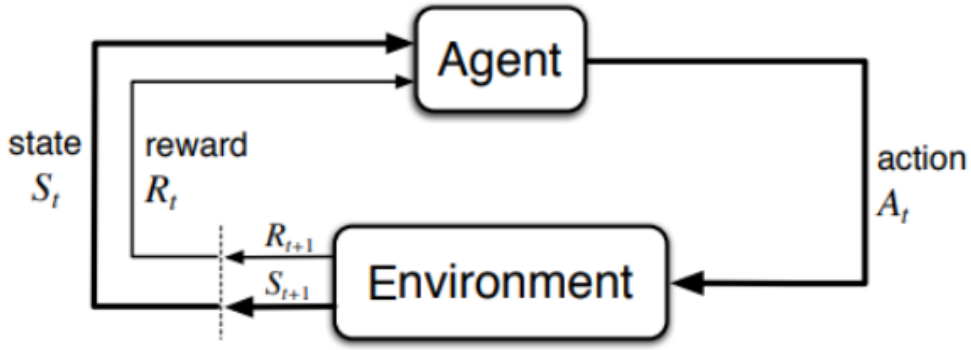


Figure 1 - Markov Decision process diagram [30]

In this process, an agent interacts with the environment. Initially, the agent observes the state of the environment in the first time-step (**t**) denoted as **St** and chooses the action that

according to his knowledge (policy) is best suited (**At**). This action produces an effect on the environment which modifies its state. As a result, the agent observes the new state (**St+1**) in which the task finds itself and receives a reward from the environment which gives feedback on the quality of the previous act (**Rt**). Then based on the new state, the agent chooses another action, and the cycle repeats itself until the task is solved or the last time step is reached.

Thanks to the Markov decision Process we can describe a control task in the simple form of a quadruple of objects: **S, A, R, P.** The first object is the **state space**, which consists of all the possible states in which the task can find itself. The second is the **action space**, which consists of the set of all the actions that an agent can execute in the environment. The third is the set of all **rewards** obtained by performing each action in each state. The last object is the **set of probabilities** of changing from one state to another by taking an action. If we know these for elements, we can perfectly describe a control task.

2.3.2.1 Types of Markov decision Process

A Markov decision process can be classified as **finite** or **infinite**. In a finite decision process, both the number of states, the number of actions and the number of rewards are finite. In an infinite MDP one or more of these three values is infinite.

We can also distinguish a MDP as **episodic** or **continuing**. An episodic process has a terminal condition while a continuing process does not have a terminating condition.

2.3.2.2 Reward and Return

As stated in previous sections the goal of the task is represented by the rewards that the environment gives the agent in response to an action. To maximize a task in the best way possible, it is necessary to maximize the sum of those rewards. However, taking an action to obtain a short-term reward can lead to lower rewards in the long run. The best-case scenario is to maximize the long-term sum of rewards. Reward is the immediate result that an action produces while return is the sum of rewards from the first time-step until a task is concluded. Since we want to maximize the long-term sum of rewards, we can also say that we want to maximize the return.

2.3.2.3 Policy

The policy is a function that takes as input as state and returns the action to be taken in that state and it is represented by the Greek letter **π.** Policies can be stochastic or deterministic. A deterministic policy, $\pi(s)$,always choose the same action in a given state while a stochastic policy, $\pi(a|s)$, chooses the action based on probabilities. Since the objective of a control task is to maximize the return and the actions that we take are based on the policy, to maximize the sum of rewards we must find the optimal policy denoted as π^*.

2.3.2.4 Value Function

To be able to decide what action to take at a particular instant, it is important for the agent to know how good it is for the agent to be in a particular state. A way of measuring the "goodness" of a state is the value function. It is defined as the expected sum of rewards that the agent will receive while following a particular policy π from a particular state s. The value function, Vπ(s) for policy π is given by

$$V\pi(s) = E(Gt|st = s)$$

The value of state will be defined as the return that we expect to obtain starting from that state and interacting with the environment following policy π until the end of the episode. The value of state is linked to the policy that we decide to follow since different policies that different actions and will achieve different returns.

On the other hand, we can also evaluate an action in a particular state by estimating the q-value. The q-value of an action in a state is the return we expect to obtain if we start in state **s** and, we take action **a**, and then we interact with the environment following policy π until the end of the episode. The action value function, Qπ(s, a) is given by

$$Q\pi(s,a) = E(Gt|St = s, At = a)$$

Q-values are used extensively because they simplify the search for the optimal policy.

2.3.2.5 Bellman Equation

The Bellman equations formulate the problem of maximizing the expected sum of rewards in terms of a recursive relationship of a value function. A policy π is considered better than another policy π' if the expected return of that policy is greater than π' for all s ∈ S, which implies, V π (s) ≥ V π' (s) for all s ∈ S. Thus, the optimal value function, V∗(s) can be defined as,

$$V*(s) = \max\pi V\pi(s), \forall s \in S$$

Similarly, the optimal action value function, Q∗(s, a) can be defined as,

$$Q*(s,a) = \max\pi Q\pi(s,a), \forall s \in S, a \in A.$$

The optimal q-value for an action in a state is the weighted sum of the rewards obtained upon reaching each of possible successor states weighted by the probability of reaching that successor state.

2.3.3 Finding the optimal policy

2.3.3.1 Monte Carlo Methods

This is a family of methods that learn optimal $\mathbf{V}*(\mathbf{s})$ or $\mathbf{Q}*(\mathbf{s},\mathbf{a})$ based on samples of experiences collected by the agents while interacting with the environment. At the beginning of the learning process the agent will follow an arbitrary policy. Then the agent will try to perform the task using that policy until the end of the episode, generating a trace of

experience containing the states visited, the actions taken, and the rewards obtained. At the end of the episode, it is calculated the return from every state visited. Every time it is observed a new return for a specific state the estimated value of that state is updated as the average of all the returns that the agent has collected. The same process can be done using q-values, except now we must average the returns produced after taking a specific action in that state. Using these two steps iteratively, it can be shown that the algorithm converges to the optimal value function and policy. Though Monte Carlo methods are straightforward in their implementation, they require a large number of iterations for their convergence and suffer from a large variance in their value function estimation.

2.3.3.2 Temporal Difference Methods

Temporal difference methods are a cluster of algorithms that learn the optimal values based on experience. Temporal difference methods combine features of Monte Carlo methods with features of dynamic programming. As in Monte Carlos Methods, the agent will face the environment, generating a trajectory with the states visited, the actions taken and the rewards obtained. At the end of the episode, the q-values estimates are updated based on the experience and the policy. Similarly to Monte Carlos methods, the environments dynamics are not modelled. Q-values implicitly estimate the environment dynamics, in that way the policy will simply choose the actions with the highest q-value. Also, as in dynamic programming, bootstrap is used (estimates of q-values are used to produce new estimates that will be more accurate). The update equation for the value function is given by:

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s\,0) - V(s)]$$

where, α is the learning rate, r is the reward received at the current time instant, s' is the new state and s is the old state. Two TD algorithms which have been widely used to solve RL problems are SARSA (State-Action-Reward-State-Action) and Q-Learning.

SARSA is an on-policy temporal difference algorithm which tries to learn an action value function instead of a value function. The policy evaluation step uses the temporal error for the action value function, which is like the value function. The algorithm is summarized below:

Algorithm 1 SARSA

Initialize Q(s, a) randomly
repeat
  Observe initial state s1
  Select an action a1 using policy derived from Q
  for t=1:T **do**
    Carry out action at
    Observe reward rt and new state st+1
    Choose next action at+1 using policy derived from Q
    Update Q using
      Q(st , at) ← Q(st , at) + α[rt + γQ(st+1, at+1) − Q(st , at)]
    end for
 until terminated

As in Monte Carlo methods, SARSA follows the generalized policy iteration, in which policy evaluation and policy improvement will alternate as described in the figure below.

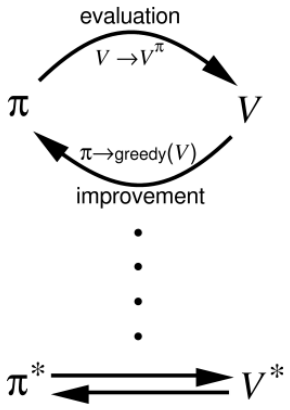


Figure 2 generalized policy iteration [31]

The evaluation process and the improvement process compete with each other, but they also drive each other towards the optimal q-values and optimal policy. Unlike Monte Carlo methods that update once per episode, temporal difference methods perform this cycle of policy evaluation and improvement every time an action is taken during the episode.

Another common algorithm that learns from temporal differences is Q-learning. This method follows an off-policy learning strategy. This means that two policies exist, the first is called a target policy that has the objective of participating in the optimization process and the other called exploratory policy to explore the environment. Q-learning is off-policy since the action is chosen with respect to an -greedy policy while the Q-function is updated by using a policy which is directly greedy with respect to the current Q-function. Q-learning differs from SARSA in the way the Q-function is updated. The update rule is given by

$$\mathrm{Q(s,a)} \leftarrow \mathrm{Q(s,a)} + \alpha[\mathrm{r} + \max \mathrm{a}\, \gamma \mathrm{Q(s\,0\,,a)} - \mathrm{Q(s,a)}]$$

The algorithm is summarized below:

Algorithm 2 Q-learning

Π <- greedy policy
b <- exploratory policy
Initialize Q(s, a) randomly
Repeat
Restart environment and observe initial state s1
for t=1:T **do**
Select an action at using policy derived from Q (e.g. -greedy)
Carry out action at
Observe reward rt and new state st+1

```
    Update Q
  end for
until episode < N
```

The algorithms start by initiating two separate policies, the target policy which is greedy, and the exploratory policy as well as the q-value table. Entering the main loop that is repeat for the number of episodes. In each episode the environment is restarted, and the initial state is observed. The inner loop that is executed for every moment in time, an action is chosen using the exploratory policy. That action is then carried out, the environment is observed, and the q-values are updated according to the update rule.

2.3.3.3 Continuous State Spaces

When working with continuous state spaces, there are two widely used strategies. The first one is called **state aggregation** and consists of cutting the range of valid values for a given state into a finite number of intervals and then aggregate all the values inside those intervals into a single state. The computation complexity of state aggregation grows exponentially as the number of state dimension grows and the precision of the estimates is limited since several states are being aggregated into one.

Function approximators is an alternative technique used in continuous state spaces that is more precise than state aggregators and has a limited complexity. In this algorithm instead of storing an estimation on the value table, a series of parameters decides the shape of a function. During the learning process those parameters are modified so that the function fits as well as possible the real q-value function.

2.3.3.4 Policy Gradient Methods

Policy Gradient methods are a are a type of reinforcement learning techniques where a function approximator is used to estimate the probability of taking each action instead of estimating the q-values. Generally, a neural network takes as input state values, a based on the state, outputs a vector of probabilities containing the probabilities of taking each action. In policy gradient methods the neural network is the policy (stochastic). In this type of algorithms, if the parameters of neural networks are changed, they will change the policy. The policy gradient methods target at modelling and optimizing the policy directly. The policy is usually modelled with a parameterized function respect to θ, $\pi\theta(a|s)$. The value of the reward (objective) function depends on this policy and then various algorithms can be applied to optimize θ for the best reward. The reward function is defined as:

$$J(\theta) .= v\pi\theta\ (s0),$$

where $v\pi\theta$ is the true value function for $\pi\theta$, the policy determined by θ.

It is natural to expect policy-based methods are more useful in the continuous space. Because there is an infinite number of actions and (or) states to estimate the values for and hence value-based approaches are way too expensive computationally in the continuous space. For example, in generalized policy iteration, the policy improvement

step argmaxa∈AQπ(s,a) requires a full scan of the action space, suffering from the curse of dimensionality.

Using gradient ascent, we can move θ toward the direction suggested by the gradient ∇θJ(θ) to find the best θ for πθ that produces the highest return.

Computing the gradient ∇θJ(θ) is tricky because it depends on both the action selection (directly determined by πθ) and the stationary distribution of states following the target selection behavior (indirectly determined by πθ). Given that the environment is generally unknown, it is difficult to estimate the effect on the state distribution by a policy update. Fortunately, there is an excellent theoretical answer to this challenge in the form of the policy gradient theorem introduced in [29], which provides us an analytic expression for the gradient of performance with respect to the policy parameter. The policy gradient theorem establishes that

$$\nabla J(\boldsymbol{\theta}) \propto \sum_{s} \mu(s) \sum_{a} q_{\pi}(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}),$$

where the gradients are column vectors of partial derivatives with respect to the components of θ, and π denotes the policy corresponding to parameter vector θ. The symbol ∝ here means “proportional to”. In the episodic case, the constant of proportionality is the average length of an episode, and in the continuing case it is 1, so that the relationship is actually an equality.

2.3.3.5 Actor Critic Methods

Actor critic methods are TD methods which store the policy explicitly. Two main components in policy gradient are the policy model and the value function. It makes a lot of sense to learn the value function in addition to the policy, since knowing the value function can assist the policy update, such as by reducing gradient variance in vanilla policy gradients, and that is exactly what the Actor-Critic method does.

Actor-critic methods consist of two models, which may optionally share parameters:

- **Critic** updates the value function parameters w and depending on the algorithm it could be action-value or state-value .
- **Actor** updates the policy parameters for, in the direction suggested by the critic.

In Actor-Critic methods exists two learning rates and are predefined for policy and value function parameter updates respectively. The algorithm below is the pseudo-code for simple action-value actor-critic algorithm.

Algorthm 3 simple action-value actor-critic algorithm.

Initialize s,θ,w at random; sample $a\sim\pi\theta(a|s)$.
For t=1…T:
  Sample reward $r_t\sim R(s,a)$ and next state $s'\sim P(s'|s,a)$;
  Then sample the next action $a'\sim\pi\theta(a'|s')$;
  Update the policy parameters: $\theta\leftarrow\theta+\alpha\theta Qw(s,a)\nabla\theta\ln\pi\theta(a|s)$;
  Compute the correction (TD error) for action-value at time t: $\delta t=rt+\gamma Qw(s',a')-Qw(s,a)$
  and use it to update the parameters of action-value function: $w\leftarrow w+\alpha w\delta t\nabla wQw(s,a)$
 Update $a\leftarrow a'$ and $s\leftarrow s'$.

2.4 Deep Neural Networks

2.4.1 Introduction

Deep Neural Networks are the driving force behind some of the major developments in artificial intelligence and machine learning in the recent years. They are at the heart of most interesting products like self-driving cars [32], cutting edge cancer detection [33] and other diseases analysing X-ray images [34], and most recently contributing to ground breaking advances in understanding the protein folding problem, a 50-year-old grand challenge in biology [35]. They have achieved state-of-the-art performances in these tasks and are becoming ubiquitous for these purposes.

Neural networks (NNs), which form the theoretical architecture of deep learning, were inspired by the primary visual cortex of cats where neurons are organized in hierarchical layers of cells to process visual stimulus [36]. A strength of deep learning is that features of the data are built hierarchically, which enables the representation of complex functions. Thus, deep learning can accurately fit functions without hand-designed features or the user choosing a good basis.

This chapter will give a brief introduction about the neural network architecture, the algorithms used in training the neural network and will introduce a regularization technique, dropout, which improves the prediction accuracy of the neural networks.

2.4.2 Architectures of Neural Networks

Artificial Neural Networks (ANNs) are mathematical models that have been motivated by the functioning of the brain. However, the models we discuss in the following do not aim at providing biologically realistic models. Instead, the purpose of these models is to analyse data.

2.4.2.1 Model of an Artificial Neuron

The basic entity of any neural network is a model of a neuron. In Figure 3, the basic layout of a perceptron is presented.

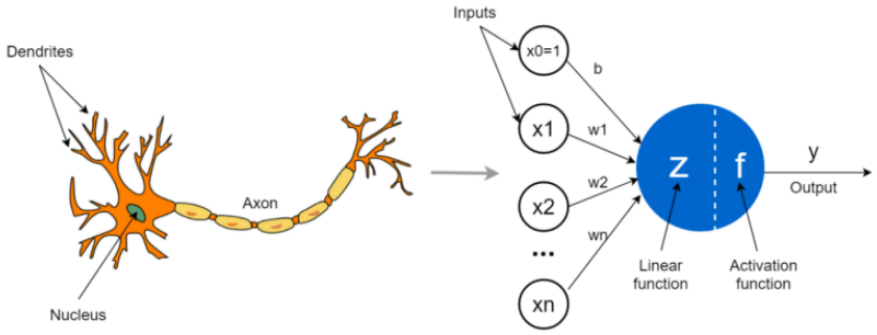


Figure 3 Artificial Neuron (Perceptron) [37]

Simplifying a lot, one can say a neuron is a cell of the nervous system capable of receiving, processing, and sending electrical or chemical signals to other neurons with which it is connected. Neurons are connected to other neurons through dendrites and axons. Dendrites are the connections through which the neurons receive the signals. The received signals are aggregated and process in the cell body and when they exceed a certain intensity, the cell emits electrical pulses through the axons to other cells. Simply put, dendrites are the part of the cell that receives stimuli and axons are part that propagate the signal.

The basic idea of an artificial neuron model is that an input, **x**, together with a bias, *b,* is weighted by, **w**, and then summarized together. The bias, *b*, is a scalar value whereas the input **x** and the weights **w** are vector valued, i.e., $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^n$ with $n \in \mathbb{N}$ corresponding to the dimension of the input.

$$y = \phi(z) = \phi(w^T x + b)$$

Considering only the argument of φ one obtains a linear discriminant function [38].

2.4.2.2 Activation Functions

The activation function introduces non-linearity into the output of a neuron. This helps the network to learn non-linear representations from the training data. Several activation functions have been introduced in literature, like sigmoid, hyperbolic tangent and a rectified linear unit. The commonly used activation functions are explained below:

- Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Hyperbolic Tangent

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- Rectified Linear Unit (ReLu)

$$f(x) = \max(0, x)$$

From the activation functions discussed above, ReLu provides the best performance over a wide range of tasks, as it does not suffer from saturation as compared to the other two functions.

2.4.3 Feed Forward Networks

The feedforward neural network also known as Multi-layered Network of Neurons (MLN) was the first and simplest type of artificial neural network devised [39]. In this network, the information moves in only one direction—forward—from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

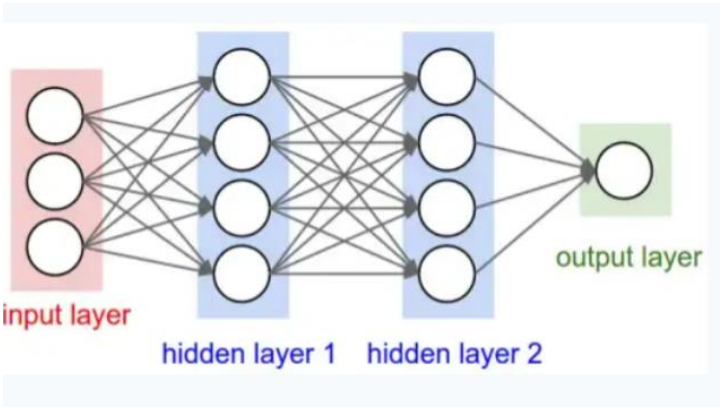


Figure 4 Feed Forward Network [40]

This architecture usually has three kinds of layers: an input layer, a few hidden layers and an output layer. The flow of information takes place from the input layer to the hidden layers and finally to the output layer which computes the final output.

The following defines a prototypical m-layer (meaning m−2 hidden layers) MLP that computes a one-dimensional output o on an n-dimensional input x = [x1,…,xn], assuming that:

1. The output perceptron has an activation function g_o and hidden layer perceptrons have activation functions g.

2. Every perceptron in layer l_i is connected to every perceptron in layer l_{i-1},therefore layers are "fully connected." Thus, every perceptron depends on the outputs of all the perceptrons in the previous layer.
3. There are no connections between perceptrons in the same layer.

Computation of the MLP's output
1.Initialize the input layer l_0 Set the values of the outputs o_i^0 for nodes in the input layer l_0 to their associated inputs in the vector x 2. Calculate the product sums and outputs of each hidden layer in order from l_1 to l_{m-1} For k from 1 to m-1 Compute product sum plus bias for perceptron i in layer l_k Compute output for node i in layer l_k 3. Compute the output y for output layer l_m

Neural networks can be used in control tasks. For example, in Figure 4, if we want to solve a control task where the state has three dimensions and one dimension for the actions, we can use this neural network to train to estimate the Q-values of the actions for each state that we give as input.

2.5 Deep Reinforcement Learning

2.5.1 Introduction

Deep reinforcement learning is the combination of reinforcement learning (RL) and deep learning. Deep Reinforcement Learning (DRL) has received a lot of interest among the AI community in the last couple of years. DRL refers to the use of Deep Neural Networks as function approximators for value functions or policy in a RL framework. This field of research has been able to solve a wide range of complex decision-making tasks that were previously out of reach for a machine. DRL opens up many new applications in domains such as finance [41] [42] [43], recommendation systems [44], energy management systems [45] [46], robotics [47] [48] and many more fields of research. The fact that DRL can handle high dimensional state and action space makes it extremely suitable for the purpose of tackling the portfolio allocation problem. DRL subdivides into two main categories namely **Value-based methods,** which aim to build a value function, which subsequently lets us define a policy such as Deep-Q-Learning and **Policy gradient methods.** Policy gradient methods optimize a performance objective (typically the expected cumulative reward) by finding a good policy (e.g a neural network parameterized policy) thanks to variants of stochastic gradient ascent with respect to the policy parameters

2.5.2 Deep-Q-Learning

Deep-Q-learning results from the combination of temporal difference algorithm Q-learning with Neural Networks. This novel approach was introduced in [49] where the authors presented the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. In this paper Q-learning is combined with Convolutional Neural networks with the objective of beating Atari 2600 games and the algorithm showed human level performance on seven games using only raw image pixels as the input. In this paper the authors trained a Q-network by minimizing a sequence of loss functions Li(θi) defined as:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right],$$

"where $y_i = \mathbb{E}_{s'\sim\mathcal{E}}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a]$ is the target for iteration i and ρ(s, a) is a probability distribution over sequences s and actions a that we refer to as the behaviour distribution" [49].

To minimize the loss function, the gradient of the loss function is computed with respect to the weights and is given by

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right].$$

The loss function is minimized using stochastic gradient descent. The behaviour policy is an ε-greedy policy to ensure sufficient exploration. The pseudocode for the proposed algorithm is presented below.

Algorithm 4 Deep Q-learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = 1, M do
  Initialise sequence s1 = {x1} and pre-processed sequenced ϕ1 = ϕ(s1)
  for t = 1, T do
    With probability select a random action at
    otherwise select at = maxa Q∗(ϕ(st), a; θ)
    Execute action at in emulator and observe reward rt and image xt+1
    Set st+1 = st, at, xt+1 and pre-process ϕt+1 = ϕ(st+1)
    Store transition (ϕt, at, rt, ϕt+1) in D
    Sample random minibatch of transitions (ϕj , aj , rj , ϕj+1) from D
    Set yj = ⟨$\begin{pmatrix} \text{rj} & \text{for terminal } \varphi j+1 \\ \text{rj} + \gamma \max a0\, Q(\varphi j+1, a0\,;\,\theta) & \text{for non-terminal } \varphi j+1 \end{pmatrix}$⟩
    Perform a gradient descent step on $(yj - Q(\varphi j\,, aj\,;\,\theta))2$
  end for
end for

In order to break the temporal correlation of data points while training, the agent's experience at each time step, (st , at , rt , st+1), is saved in a replay buffer. At every instant, a fixed number of samples are extracted from the replay buffer at random and used to train the network. The one major drawback of the above algorithm is the need to calculate the maximum over actions and this prohibits the use of of the above algorithm for tasks with continuous actions spaces.

2.5.3 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is a method for self-control tasks with continuous action spaces introduced in [50]. In order to solve tasks where the action space is infinite the DDPG algorithm assumes that the Q-function is differential with respect to the action. This means that Q-function (that is the value of an action) will change very smoothly as the values of the actions change. This actor-critic framework where the actor and the policy are described by two separate neural networks uses stochastic gradient decent to update the Q-value function:

$$Q(s, \pi(s))$$

In order to the π policy, an action in a give state is selected and then use another neural network to compute the Q-value of that state and that action. Then, the gradient is computed with respect to the parameters of the policy. In this algorithm there is a neural network for the policy and another one for the q-network for computing q-values. The policy is updated through the q-network. The complete DDPG algorithm pseudocode is given below:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 5 DDPG pseudocode from the original paper [50]

2.5.4 Twin Delayed DDPG (TD3)

Twin Delayed DDPG (TD3) is a variant of the DDPG algorithm with some improvements that, in theory, makes it more stable and capable of solving more complex tasks [51]. DDPG has some downturns such as being very sensitive to hyperparameter choice, suffers from maximization bias and which results in estimating wrong q-values leading to a suboptimal policy. TD3 introduces three improvements to the DDPG algorithm. The first is called Clipped double Q-learning which creates two separated neural networks to compute the Q-values, each one of them with their own target networks. In the network update process, to avoid overestimation TD3 uses both neural networks to estimate independently the q-value of the future state and the future action, and then selects the most conservative of those estimates. Having two independent q-networks allows for each one of them to correct the other overestimation. The second improvement consists of delayed policy updates where the policy is only updated once for every n updates of the Q-network, keeping the policy more stable. The last improvement is called target policy smoothing, which aims to solve the problems that come with the deterministic feature of DDPG. Since DDPG always takes the same action for a given state, this creates a problem in cases where the Q-Network overestimates the value of a given action creating a peak around it. This results in the policy exploiting that peak and it will always assign that overestimated action in that state and it will be trapped in that error forever. To solve this problem, when computing the target of the Q-network, the authors

suggest slightly pushing the action selected by the policy into a nearby action by adding some noise. TD3 pseudocode is presented below:

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ
with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
  $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

Figure 6 TD3 pseudocode from the original paper [51]

2.5.5 Soft Actor Critic (SAC)

Soft Actor Critic is a method with a stochastic based policy introduced in [52], where the policy assigns the probability of taking a given action in each state. Since, in a continuous action space, there are infinite possible actions, this policy would have to compute an infinite number of probabilities. This method tackles this problem by forcing the probability distribution to be a normal distribution, therefore making only necessary computing the mean and the standard deviation to known everything about the probability of the policy selecting each action. In this algorithm the policy network outputs Mu and Sigma, and then these two values will construct the probability function and will sample the action from this probability. This will make sure that the algorithm exploits the actions that are best, but sometimes it will deviate to explore the environment. A problem appears when the policy network finds success very early, resulting in the thinning of the probability distribution so that it always selects the same actions. In other words, this problem pushes the standard deviation to zero and it will stop exploring relatively early. Maximum entropy reinforcement learning is the technique proposed by the authors to complement the algorithm because it optimizes policies to maximize both the expected return and the expected entropy of the policy. This means that

an agent will be rewarded for keeping the exploration high. SAC pseudocode is presented below:

Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  end for
end for

Figure 7 SAC pseudocode from the original paper [52]

2.5.6 Proximal Policy Optimization (PPO)

Introduced in [53], Proximal Policy Optimization is a family of policy gradient methods for reinforcement learning. PPO aims to strike a balance between important factors like ease of implementation, ease of tuning, sample complexity, sample efficiency and tries to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small. PPO is in fact, a policy gradient method that learns from online data as well. It merely ensures that the updated policy isn't too much different from the old policy to ensure low variance in training. The most common implementation of PPO is via the Actor-Critic Model which uses 2 Deep Neural Networks, one taking the action(actor) and the other handles the rewards(critic). The mathematical equation of PPO is shown below:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

"Where, epsilon is a hyperparameter, say, ε = 0.2. $clip(rt \ldots)$ modifies the surrogate objective by clipping the probability ratio, which removes the incentive for moving r_t outside of the interval [1 − ε, 1 +ε]. Finally, take the minimum of the clipped and unclipped objective, so the final objective is a lower bound on the unclipped objective" [53].

Pseudo-code for PPO Actor Critic is presented below:

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 8 PPO pseudocode from the original paper [53]

2.5.7 Advantage Actor Critic (A2C)

An actor-critic algorithm is a policy gradient algorithm that uses function estimation in place of empirical returns Gt in the policy gradient update. The actor is the policy πθ and the critic is usually a value function Vφ. The actor is trained using gradient ascent on the policy gradient, and the critic is trained via regression. The regression target can either be the empirical returns Gt, or can be the Bellman target rt+γVφ(st). An advantage actor-critic is just an actor-critic that uses the advantage A(st) instead of Vπθ. Advantage function is nothing but difference between Q value for a given state-action pair and value function of the state and its presented in the equation below:

$$A(s, a) = Q(s, a) - V(s)$$

This can be intuitively taken as the difference of q value and the average of actions which it would have taken at that state. It tells about the extra reward that could be obtained by the agent by taking that particular action. Pseudo-code for A2C is presented in the figure below:

Algorithm 1 Advantage actor-critic - pseudocode

// Assume parameter vectors θ and θ_v
Initialize step counter $t \leftarrow 1$
Initialize episode counter $E \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta)$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t+1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta_v) & \text{for non-terminal } s_t \text{ //Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \nabla_\theta \log \pi(a_i|s_i;\theta)(R - V(s_i;\theta_v)) + \beta_e \partial H(\pi(a_i|s_i;\theta))/\partial\theta$
    Accumulate gradients wrt θ_v: $d\theta_v \leftarrow d\theta_v + \beta_v(R - V(s_i;\theta_v))(\partial V(s_i;\theta_v)/\partial\theta_v)$
  end for
  Perform update of θ using $d\theta$ and of θ_v using $d\theta_v$.
  $E \leftarrow E + 1$
until $E > E_{max}$

Figure 9 A2C pseudocode [54]

2.6 Summary

This chapter started by explaining basic financial terms and concepts and some related papers were analysed revealing that are several ways at looking at the portfolio management problem. One can use function approximators to predict the actions that maximize the reward in a given environment. The actions in this problem can be viewed as a vector of the weights representing the allocation for each asset or as a discrete problem were the function approximator predicts one of three actions, buy, sell, or hold. The sections following the related work expose the background information that led to deep reinforcement learning. It starts by reviewing Reinforcement Learning basics, followed by a succinct explanation of neural networks that are used as function approximators and lastly five deep reinforcement learning algorithms are analysed. The following chapter exposes how the portfolio management problem can be converted to a Markov decision process, how the DRL algorithms interact with the environment and how to measure the quality of the models.

3 Materials and Methods

The methodology proposed for the asset allocation problem follows the following steps:

a) **Review Classical Portfolio Management Strategies.** Explain some old school portfolio management strategies.
b) **Overview of technical indicators.** Briefly description of the technical indicators used in this project.
c) **Portfolio Management as a Markov Decision Process.** Expose Portfolio Management in a mathematical standpoint.
d) **Methodology Architecture Overview.** Overview of the interaction between models and the environment.
e) **Select the assets that will form the portfolio**.
f) **Feature Selection and Data Pre-processing**. Technical indicators are added to the selected assets to aid in the performance of the agent;
g) **Train and Test Data Split**. We split the data into train-data and test-data which is used to train the model and test its performance respectively;
h) **Benchmark Portfolios**. We construct portfolios based on equal weights allocation and mean-variance (maximum Sharpe) allocation, follow the winner and follow the loser strategies.
i) **Tools and Libraries.** In this section is described main tools used in this project.

3.1 Portfolio Management Strategies

3.1.1.1 Follow the winner

Follow the Winner approach is characterized by transferring portfolio weights from the under-performing assets (experts) to the outperforming ones.

3.1.1.2 Follow the loser

The Follow the Loser approach assumes that the under-performing assets will revert and outperform others in the subsequent periods. Thus, their common behaviour is to move portfolio weights from the outperforming assets to the under-performing assets.

3.1.1.3 Equal Weighted Portfolio

An equally weighed portfolio of assets with the capital invested equally among the total number of assets is constructed.

3.1.1.4 Markowitz's Mean-Variance Portfolio (Modern Portfolio Theory)

A Nobel prize winner strategy proposed by Harry Markowitz in [55], a ground-breaking investment strategy based on his realization that the performance of an individual stock is not as important as the performance and composition of an investor's entire portfolio. In MPT,

the **efficient frontier** is the set of portfolios that are optimal in the sense of the risk-return tradeoff, that is, every portfolio in the efficient frontier minimizes the risk for a given return. Given n number of assets with expected return vector of R and the covariance matrix of ∑ the mean-variance optimization is the solution to the following quadratic problem:

$$arg_w min(\frac{1}{2}) w^T \sum w$$

Subject to:

$$R^T \, \mathrm{w} \geq \, R_b$$

r_b is the acceptable baseline expected rate of return.

3.2 Technical Indicators

Technical indicators are used by financial analysts and portfolio managers to gauge the performance of the assets that they are interested in. A multitude of indicators exist in the financial domain, but we are only going to talk about the ones' mentioned below.

Technical Indicator	Abbreviation	Description
Moving Average Convergence Divergence	MACD	Moving average convergence divergence (MACD, or MAC-D) is a trend-following momentum indicator that shows the relationship between two exponential moving averages (EMA's) of a security's price. The MACD is calculated by subtracting the 26-period exponential moving average (EMA) from the 12-period EMA.
Fast Exponential Moving Average	EMA	The EMA is a type of weighted moving average (WMA) that gives more weighting or importance to recent price data. Like the simple moving average (SMA), the EMA is used to see price trends over time, and watching several EMAs at the same time is easy to do with moving average ribbons.
Relative Strength Index	RSI	RSI measures the speed and magnitude of a security's

		recent price changes to evaluate overvalued or undervalued conditions in the price of that security.
Bollinger Band Width	BBW	Keltner Channels are volatility-based bands that are placed on either side of an asset's price and can aid in determining the direction of a trend.
Fast Simple Moving Average	SMA	The simple moving average (SMA) is a straightforward technical indicator that is obtained by summing the recent data points in a given set and dividing the total by the number of time periods.
On-balance Volume	OBV	On-balance volume (OBV) is a technical trading momentum indicator that uses volume flow to predict changes in stock price.
Chaikin Money Flow	CMF	Chaikin Money Flow (CMF) developed by Marc Chaikin is a volume-weighted average of accumulation and distribution over a specified period.

3.3 Portfolio Allocation as a Markov Decision Process

In a mathematical standpoint, we can define a portfolio as a composition of **N** financial assets, usually described by a set of price information such as the opening price at the start of a certain instant **t**, the highest and lowest price within **t**, and the closing price at the end of **t**.

Considering a period of **T** composed by **t** trading instants and each instance being characterized by **N** closing prices **p**, we can construct the portfolio matrix **P** as in the matrix below,

$$P = \begin{matrix} p1,1 & \dots p1,t & p1,T \\ p2,1 & \dots p2,t & p2,T \\ \dots pN,1 & \dots pN,t & pN,T \end{matrix}$$

where, pN, T is the closing price at time T of a financial asset N.

The t-**th** column is the price vector of the t-**th** trading instant, denoted by $\mathbf{V}t$. The n **- th** row is the price time sequence of an asset. At this point, considering $\mathbf{V}t$**+1** and $\mathbf{V}t$, the price change vector can be defined by **yt** as follows:

$$\mathrm{yt} = \mathrm{Vt} + 1 - \mathrm{V}t$$

At time t, an agent invests on the market by a portfolio weights vector wt. wnt represents the proportion of financial exposure on a trading period t for asset n.

Now, the portfolio management problem can be described as a Markov Decision Process(MDP). Then we need the following elements:

1. The State (S): the covariance matrix of close prices and the identified market financial indicators that are used as inputs;
2. The action (A): the desired weights allocation.
3. The reward function (R): is given based on the total portfolio value adjusted for the transaction costs.
4. Policy ($\boldsymbol{\pi}(\boldsymbol{s}, \boldsymbol{a})$) : The policy determines the action to take that maximizes the reward at each state. In our case the policy is made by a Deep Neural network using a Reinforcement Learning algorithm.

3.4 Proposed Methodology

The RL algorithms listed below are used to implement five DRL models:

1. Deep Deterministic Policy Gradient (DDPG). This algorithm is constantly learning the State-action function (Q function) and employing it to determine the best policy.
2. Advantage Actor Critic (A2C) Algorithm. The policy is the actor in the algorithm, and it acts on the environment. The critic computes the value function based on a predefined reward function, which assists the actor in determining the best policy.
3. Proximal Policy optimization (PPO). The PPO algorithm is an on-policy algorithm that can be used in either discrete or continuous action spaces.
4. Twin Delayed DDPG (TD3). DDPG extension that solves the problem where the learned Q-function begins to dramatically overestimate Q-values.
5. Soft Actor-Critic(SAC). Algorithm that optimizes a stochastic policy in an off-policy way, forming a bridge between stochastic policy optimization and DDPG-style approaches.

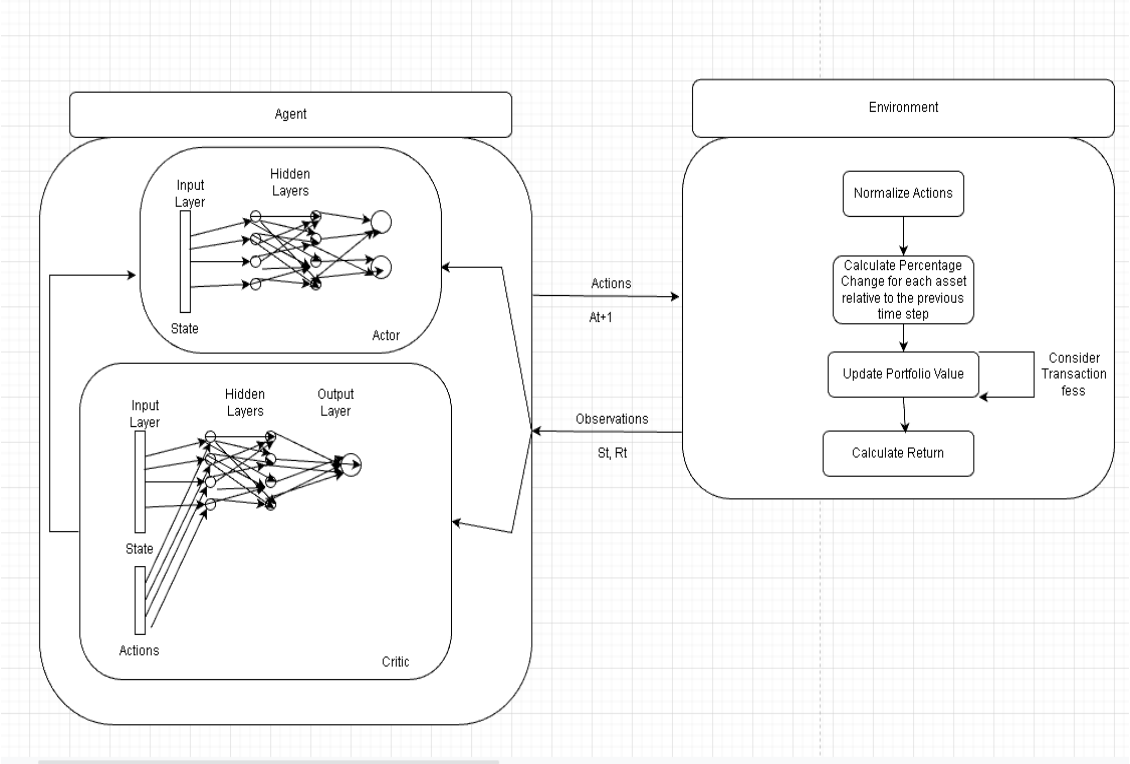


Figure 10 Proposed Methodology

The environment setup within which we implement the algorithms is made up of the Crypto constituents, the prices, the technical indicators (features) and the covariance matrix. The DRL agent interacts with the environment in an exploration-exploitation manner. This involves making decisions which will maximize rewards by considering whether to make previous good decisions (exploitation) or to try new decisions with potential for greater rewards (exploitation).

The actions which the agents make are the portfolio weights and the policy is learned at each step that maximizes the portfolio value (reward function).

3.5 Data Description

We make use of the YahooDownloader API [56] in the yfinance library to get hourly data for the 20 highest fully diluted market cap for the period of 2021-06-01 to 2022-09-11. The figure bellow summarizes the dataset structure.

	date	tic	close	high	low	open	volume
0	2022-01-23 22:45:00+00:00	ADA-USD	1.080542	1.080965	1.075063	1.075261	248832
1	2022-01-23 22:45:00+00:00	ALGO-USD	0.954671	0.955360	0.952273	0.952431	50656
2	2022-01-23 22:45:00+00:00	ATOM-USD	32.515247	32.665077	32.507679	32.665077	0
3	2022-01-23 22:45:00+00:00	AVAX-USD	61.676456	61.676456	61.457947	61.457947	837952
4	2022-01-23 22:45:00+00:00	BCH-USD	290.401917	290.401917	289.706787	290.140839	980992
...	...	...	...	...	...	...	...
1099	2022-01-24 09:51:00+00:00	TRX-USD	0.054055	0.054055	0.054055	0.054055	0
1100	2022-01-24 09:51:00+00:00	USDC-USD	1.000611	1.000611	1.000611	1.000611	0
1101	2022-01-24 09:51:00+00:00	USDT-USD	1.000576	1.000576	1.000576	1.000576	0
1102	2022-01-24 09:51:00+00:00	XLM-USD	0.182000	0.182000	0.182000	0.182000	0
1103	2022-01-24 09:51:00+00:00	XRP-USD	0.582463	0.582463	0.582463	0.582463	0

Figure 11 Data Structure

In order to assess the highest fully diluted market cap, the website Coin Gecko [57] was consulted. In the figure below we can see an ordered list of the top cryptocurrencies.

#	Moeda		Preço	1 h	24 h	7 d	Volume em 24 h	Capitalização de Mercado
☆ 1	Bitcoin BTC	Buy	US$ 19.680,47	-0.4%	-0.8%	1.7%	US$ 29.026.814.400	US$ 376.926.188.972
☆ 2	Ethereum ETH	Buy	US$ 1.447,19	-0.4%	-3.1%	-11.5%	US$ 18.350.739.204	US$ 174.760.845.488
☆ 3	Tether USDT		US$ 1,00	0.3%	0.2%	0.3%	US$ 38.495.631.953	US$ 67.847.323.122
☆ 4	USD Coin USDC	Buy	US$ 1,00	0.4%	0.4%	0.3%	US$ 7.840.666.945	US$ 50.361.646.788
☆ 5	BNB BNB		US$ 274,97	-0.0%	1.6%	-2.0%	US$ 578.602.129	US$ 44.879.620.831
☆ 6	Binance USD BUSD		US$ 0,999934	-0.2%	-0.3%	-0.0%	US$ 8.193.444.723	US$ 20.644.727.789
☆ 7	XRP XRP		US$ 0,333536	0.3%	2.4%	-2.1%	US$ 1.003.191.664	US$ 16.595.687.176
☆ 8	Cardano ADA		US$ 0,462142	0.0%	-1.4%	-3.7%	US$ 433.281.381	US$ 15.618.159.084
☆ 9	Solana SOL	Buy	US$ 32,25	-0.3%	-2.4%	-4.0%	US$ 761.666.066	US$ 11.455.889.682
☆ 10	Dogecoin DOGE	Buy	US$ 0,059738	0.0%	1.1%	-2.1%	US$ 250.527.204	US$ 7.920.523.224
☆ 11	Polkadot DOT		US$ 6,85	-0.1%	-1.4%	-7.4%	US$ 277.282.977	US$ 7.896.339.918
☆ 12	Shiba Inu SHIB	Buy	US$ 0,00001162	-0.2%	-0.8%	-5.0%	US$ 234.005.826	US$ 6.856.145.635
☆ 13	Dai DAI		US$ 1,00	0.5%	0.4%	0.3%	US$ 474.168.663	US$ 6.367.204.886
☆ 14	Lido Staked Ether STETH		US$ 1.429,79	-0.4%	-2.7%	-9.2%	US$ 59.200.802	US$ 6.176.429.463
☆ 15	Polygon MATIC		US$ 0,803762	-0.2%	-2.9%	-4.8%	US$ 335.365.396	US$ 5.983.439.033
☆ 16	TRON TRX		US$ 0,061153	0.0%	-0.0%	-0.2%	US$ 315.051.944	US$ 5.645.566.106
☆ 17	Avalanche AVAX		US$ 18,00	-0.2%	-2.2%	-5.9%	US$ 282.377.750	US$ 5.328.296.129
☆ 18	Wrapped Bitcoin WBTC		US$ 19.655,30	-0.3%	-0.5%	1.7%	US$ 219.242.309	US$ 4.899.951.594
☆ 19	Cosmos Hub ATOM		US$ 16,14	-0.4%	9.7%	16.4%	US$ 774.886.411	US$ 4.727.855.343
☆ 20	Ethereum Classic ETC	Buy	US$ 34,03	-0.0%	-9.1%	-8.4%	US$ 1.313.376.283	US$ 4.658.149.985
☆ 21	LEO Token LEO	Buy	US$ 4,93	-0.4%	-4.1%	-1.1%	US$ 2.049.181	US$ 4.595.184.164
☆ 22	Uniswap UNI		US$ 5,80	-0.2%	-1.9%	-6.0%	US$ 67.617.766	US$ 4.378.655.972

Figure 12 Top 22 Cryptocurrencies from CoinGecko

From this figure the following Crypto-USD pairs were extracted:

Table 1 Top 20 Cryptocurrencies

USD Pair	**Asset Name**
BTC-USD	Bitcoin
ETH-USD	Ethereum
USDT-USD	Tether
USDC-USD	USD Coin

BNB-USD	Binance Coin
BUSD-USD	Binance USD
XRP-USD	Ripple
ADA-USD	Cardano
SOL-USD	Solana
DOGE-USD	Dogecoin
DOT-USD	Polkadot
SHIB-USD	Shiba INU
DAI-USD	DAI
SETH-USD	Sether
MATIC-USD	Polygon
TRX-USD	Tron
AVAX-USD	Avalanche
WBTC-USD	Wrapped Bitcoin
ATOM-USD	Cosmos
ETC-USD	Ethereum Classic

Since cryptocurrencies are a very recent theme, and some of the top cryptos this year didn't exist last year, only assets that exist for the period of 2021-06-01 to 2022-09-11 were considered, resulting in a total of 12 assets represented in the table below:

Table 2 Filtered Cryptocurrencies

USD Pair	**Asset Name**
BTC-USD	Bitcoin
ETH-USD	Ethereum
USDT-USD	Tether
USDC-USD	USD Coin
XRP-USD	Ripple
ADA-USD	Cardano
SOL-USD	Solana
DOGE-USD	Dogecoin
MATIC-USD	Polygon
TRX-USD	Tron
AVAX-USD	Avalanche
ETC-USD	Ethereum Classic

In the table below, one can see that two Stable coins were included in the top cryptocurrencies. A stable coin is an asset that has its value pegged to the US dollar. The value for both USDT-USD pair and USDC-USD pair, most of the time, should be one. Considering them both in the data will only add noise, so an analysis was conducted in order to see which should be considered. The USDT-USD was excluded from the asset list since USDC-USD had a lower correlation and standard deviation than USDT-USD and USDT-USD hit a min of 0.015096 while USDC-USD hit a min of 0.964171. After this selection the asset list is reduced to a final of

11 cryptocurrencies, including only one stable coin (riskless asset). In the figure below is represented the heatmap that allowed this conclusion.

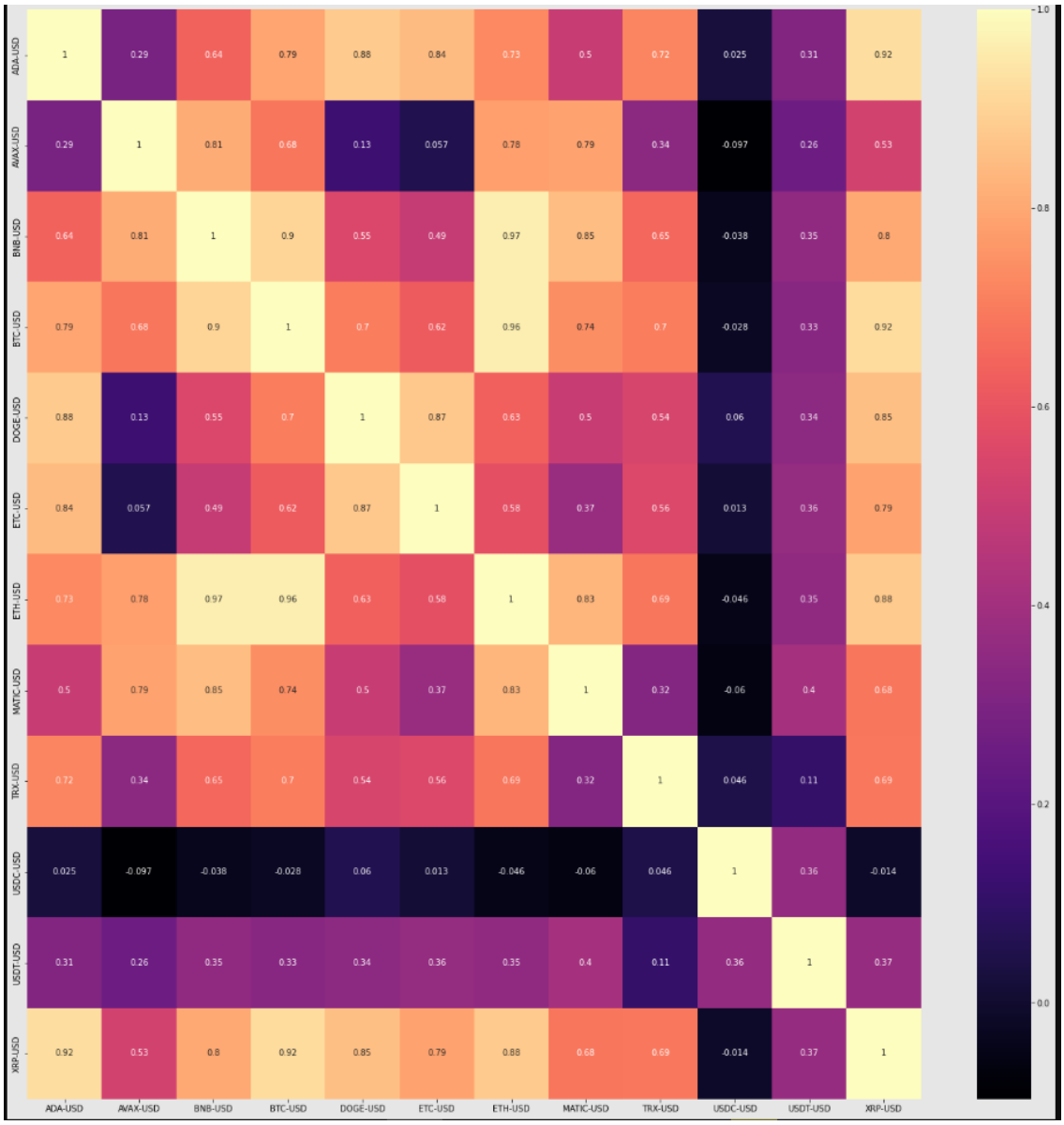


Figure 13 Correlation Heatmap for the complete dataset

As we can see, the market is very highly correlated except for stable coins. Analysing the bitcoin, which is the asset with the highest market cap, the asset with the lowest correlation (excluding stable coins) is Ethereum Classic (ETC), with a correlation of 0.62, while the one with the highest is Ethereum(ETH) with a correlation of 0.96. XRP,BNB and ADA with a correlation of 0.92, 0.90 and 0.79 which were the highest after ETH while AVAX, DOGE and TRX were the next in line for the lowest correlation after ETC with 0.68, 0.7 and 0.7 respectively. It looks like the higher the market cap of an asset the more correlation it has with BTC (excluding stable coins).

3.6 Feature Selection and Data Pre-processing

The following technical indicators are generated to be used as feature:

- Moving Average Convergence Divergence (MACD)
- Relative Strength Index (RSI)
- Bollinger Band Width (BBW)
- Keltner Channel (KCW)
- Fast Simple Moving Average (SMA)
- Fast Exponential Moving Average (EMA)
- On-balance Volume (OBV)
- Chaikin Money Flow (CMF)

These features were selected mainly because MACD and RSI are the most popular indicators in financial markets, and with the objected of adding two indicators about trend, two about volume and two about volitility.

Dimensionality reduction using unsupervised stacked restricted Autoencoder model is performed to reduce the number of input features from eight to four.

3.7 Train and Test Data Split

First let's have a look on how the Bitcoin behaves overtime.

Figure 14 Bitcoin price evolution

In order to better understand the behaviour of the algorithm, the data was split into three categories:

- Bull Market: Characterized by a rally in the price of an asset.
- Bear Market: Characterized by a down peak of an asset.
- Sideways Market: Characterized by a price lateralization

As one can see from the bitcoin price chart, we can define for each category:

- Bull Market: Starts 2021-05-31 22:00:00 (first data point) and ends 2021-11-10 14:00:00 (highest high). Bitcoin price for this market is presented in Figure 15.
- Bear Market: Starts 2021-11-10 15:00:00 (highest high) and ends 2022-06-18 20:00:00 (lowest low). Bitcoin price in the bear market is presented in Figure 16.
- Sideways Market: Starts 2022-06-18 21:00:00 and ends 2022-09-10 21:00:00 (last data point). Bitcoin price in the lateralization period is represented in Figure 17.

Figure 15 Bitcoin price Bull Market Data

Figure 16 Bitcoin price Bear Market Data

Figure 17 Bitcoin price sideways Data

After splitting the data into these three categories, an 80% train and 20% test split was performed for each category.

3.8 Benchmark Portfolio

In order to assess the quality of the strategies produced by DRL algorithms, we are going to benchmark the test data for each market and each strategy. For all the benchmark strategies the algorithm will start with 1 million dollars' worth of bitcoin as shown in the figure below.

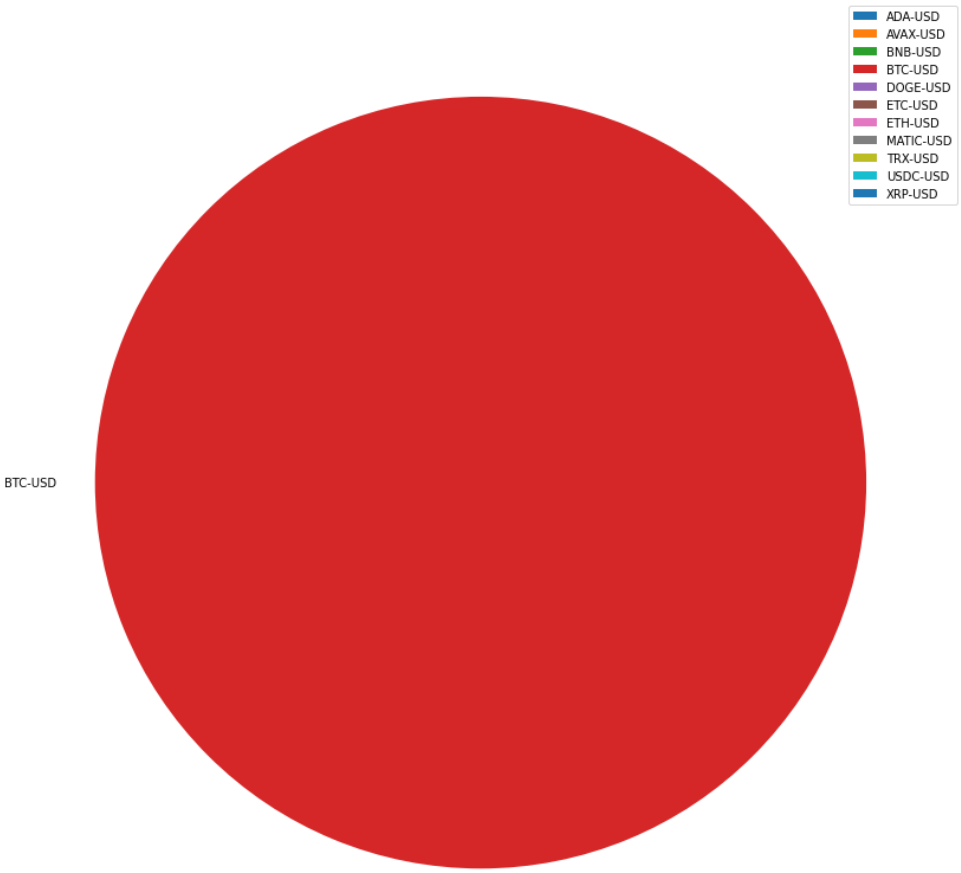


Figure 18 Only Bitcoin portfolio allocation

According to Binance's website [58], the trading fees are 0.1% for a maker and taker. In the benchmark strategies, each time the portfolio is rebalanced, we are going to deduct 0.1% of the portfolio value.

Fee Rate

Spot Trading | Margin Borrow Interest | USDⓈ-M Futures Trading | COIN-M Futures Trading | Cross Collateral Interest | Swap Farm

Level	30d Trade Volume (BUSD)	and/or	BNB Balance	Maker / Taker	Make BNE
Regular User	< 1 000 000 BUSD	or	≥ 0 BNB	0.1000% / 0.1000%	0.0750%

Figure 19 Trading fees extracted from Binance

3.8.1 Bull Data

3.8.1.1 Follow-The-Winner

In this strategy, each timestep, we are going to rebalance the portfolio to hold 100% of the highest performing asset relative to the previous timestep. The performance of this strategy is presented in Table 3.

Table 3 Follow the Winner performance of bull data

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
744083.046	-25.9%	451	108

From the table above, this is a non-profitable strategy with a return of -25.9%. This is due to the high number of rebalances since the winner may change every timestep. The portfolio value overtime is presented in Figure 20.

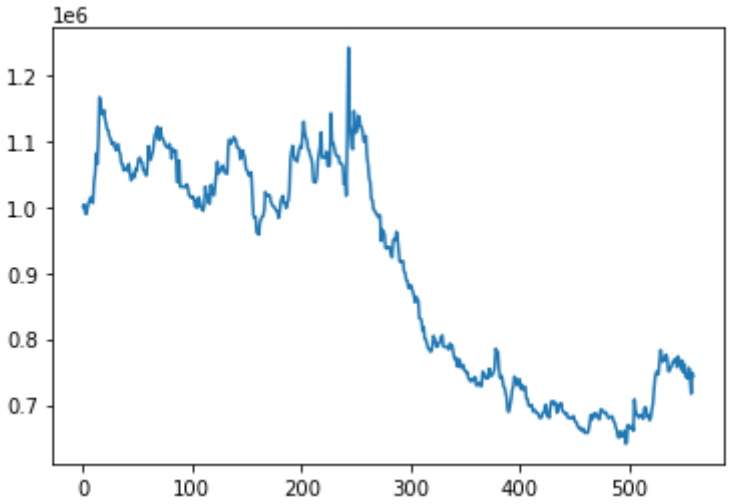


Figure 20 Follow the winner portfolio value on bull data

As we can see from the figure above, this strategy is very unstable. It starts making about 20%, but by the 200th timestep it is no longer viable.

3.8.1.2 Follow-The-Loser

In this strategy, each timestep, we are going to rebalance the portfolio to hold 100% of the lowest performing asset relative to the previous timestep. The performance of this strategy is presented in Table 4.

Table 4 Follow the loser perfomance on bull data

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
1089979.88	8,99%	488	71

Following the loser on a bull market seems to be a profitable strategy. Surprisingly it has a profit of 8.99% despite having a very high number of rebalances. The portfolio value for this strategy is presented in Figure 21.

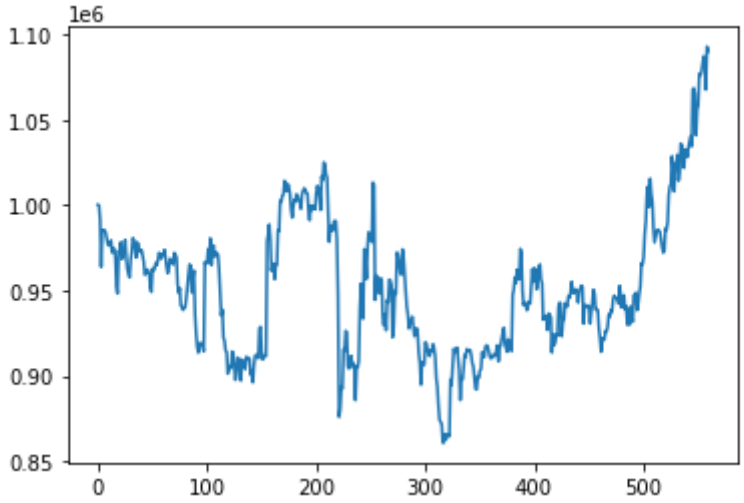


Figure 21 Follow the loser portfolio value on bull data

Similarly to follow-the-winner, this strategy requires a lot of portfolios rebalances, therefore it is very unstable despite the market being in an uptrend. As we can see from the figure above, this strategy starts losing money hitting a bottom around the 300th timestep with about 15% loss and then starts to be a profitable strategy.

3.8.1.3 Equal Weight Portfolio

In this strategy, we are going to distribute the allocation equally for all assets as shown in the figure below.

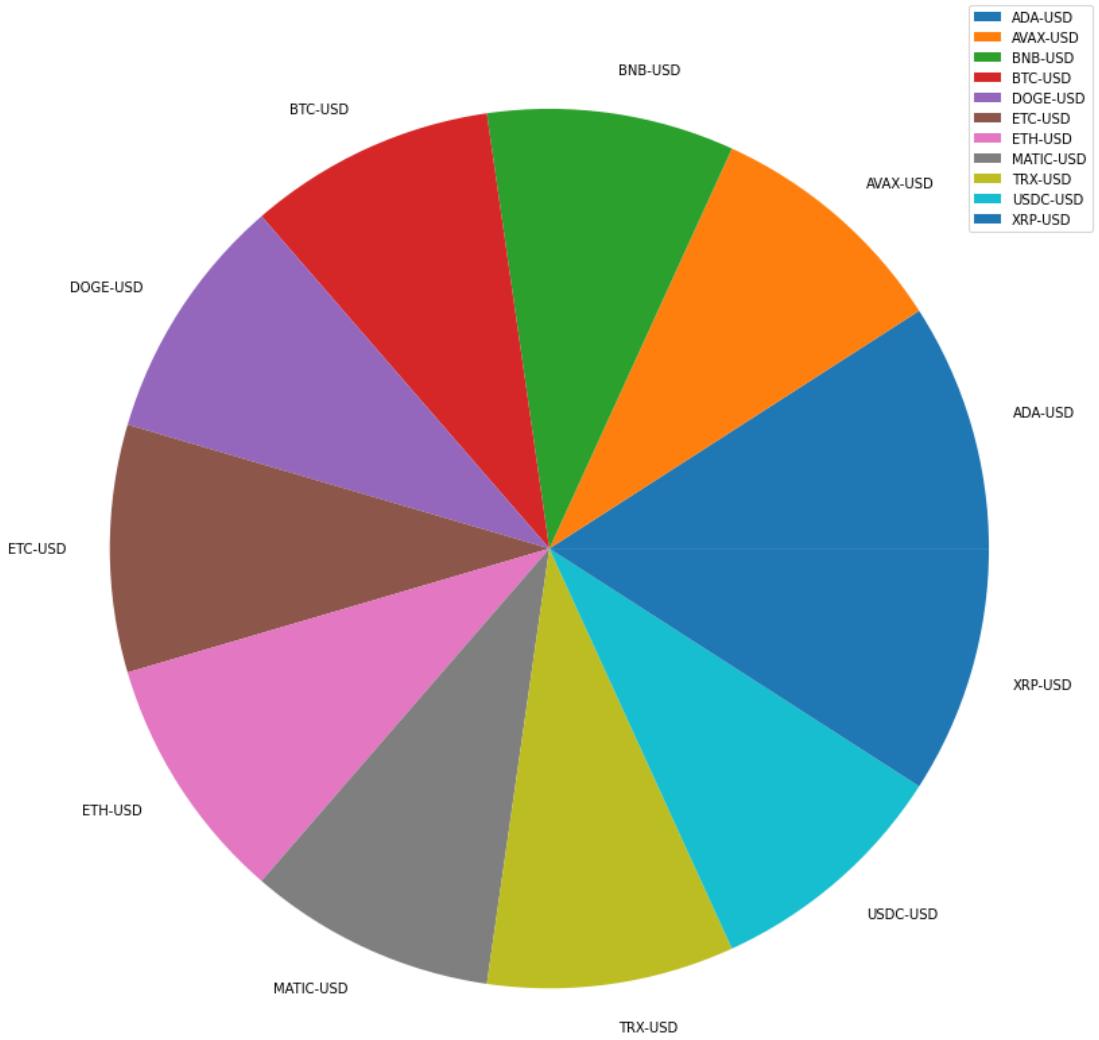


Figure 22 Equal Weight Portfolio allocation

As one can see from Figure 22, this portfolio is equally distributed across the 11 assets, each one having 9,09% allocation. The performance of this strategy is presented in Table 5.

Table 5 Equal Weight perfomance on bull data

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
1220823.006	22.08%	0	599

From the previously evaluated strategies, this is by far the most performative on a bull market with a return of 22.08%. Obviously, the number of rebalances is 0 since the assets are equally distributed in all the timesteps. The performance overtime of this strategy is presented in the figure below.

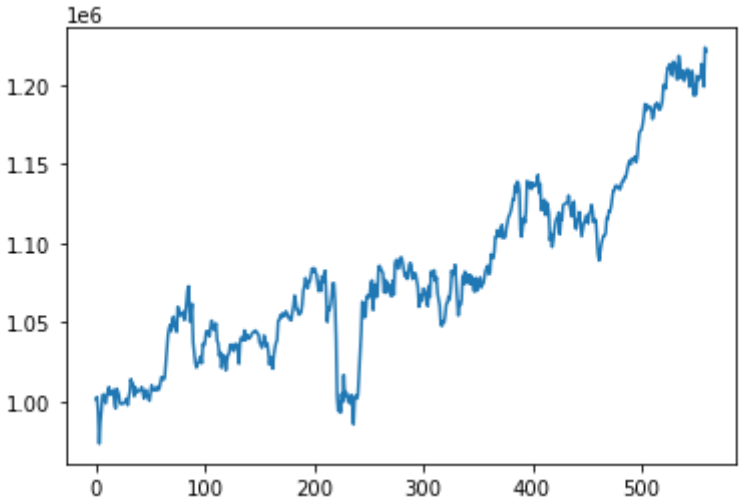


Figure 23 Equal Weight portfolio value on bull data

This strategy is very stable in this type of market. As we can see from the figure above, it almost never loses money turning this strategy into the most viable benchmark so far.

3.8.1.4 Mean-Variance Portfolio

This strategy based on the Modern Portfolio Theory had the following performance:

Table 6 Mean Variance performance on bull data

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
797105.39	-20.28%	408	151

From the table above, we can observe that this strategy is not profitable with a return of -20.28%, probably due to the high number of portfolio rebalances. The portfolio value overtime for this strategy is presented below:

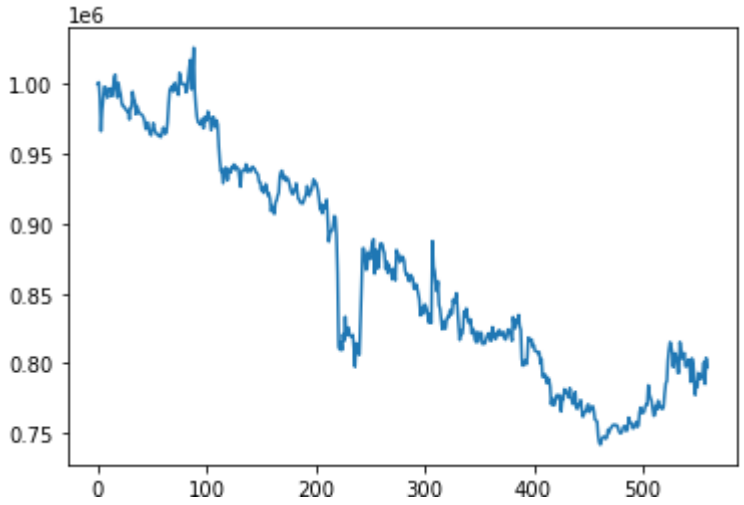


Figure 24 Mean Variance Performance of Bull data

3.8.2 Bear Data

3.8.2.1 Follow-The-Winner

The strategy where we rebalance portfolios according to the most performant asset in the previous timestep had the following performance on a bear market:

Table 7 Follow the winner bear data performance

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
438032.30	-56.20%	709	141

As we can see from the table above, this strategy loses about 56.20% of the initial portfolio value, with a very high number of rebalances. The portfolio value overtime is presented in the figure below.

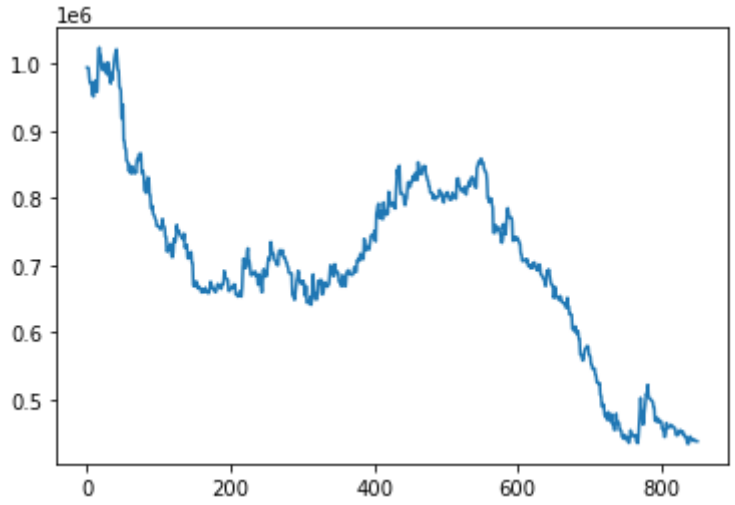


Figure 25 Follow the winner portfolio value on bear data

As we can observe, this strategy loses about 30% of the portfolio value in the first 200 timesteps, then it recovers about 15% of initial amount between the 400th and the 600th timestep and then loses money almost linearly until the final steps.

3.8.2.2 Follow-The-Loser

The strategy where we swap allocation to the least performative assets has the following performance on a bear market:

Table 8 Follow the loser bear data performance

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
265919.73	-73.40	707	143

As one can see from the table above, this strategy is even worse than follow the winner with a return of -73.40. Similarly to the previously evaluated strategy, it has a very high number of rebalances. The portfolio value for the trading period is presented in the figure below.

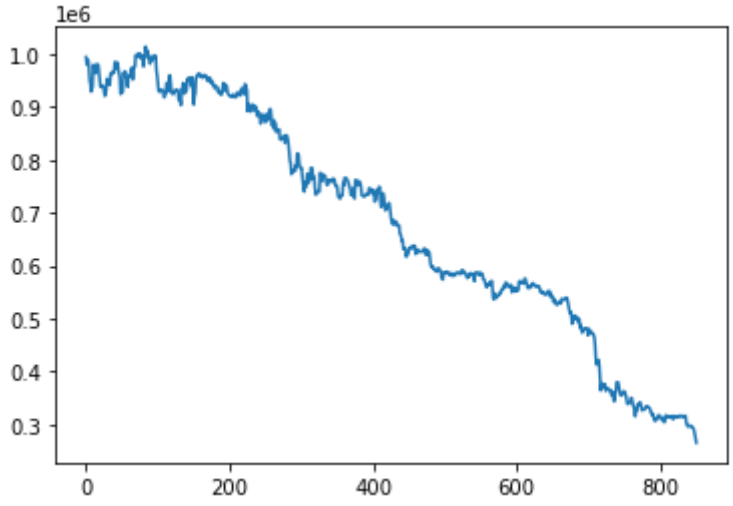


Figure 26 Follow the loser portfolio value bear data

From this figure, we can conclude that it is never able to make money and constantly loses almost in a linear way until the final timesteps.

3.8.2.3 Equal-Weight

An equally weight portfolio on a bear market performed as follows:

Table 9 Equal weight portfolio value on bear data

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
619621.24	-38.03%	0	850

From Table 9 it is possible to conclude that this is the best strategy so far, not only for bull markets but for bear markets as well with a return of -38.03%. The portfolio value over the trading period is presented in the figure below.

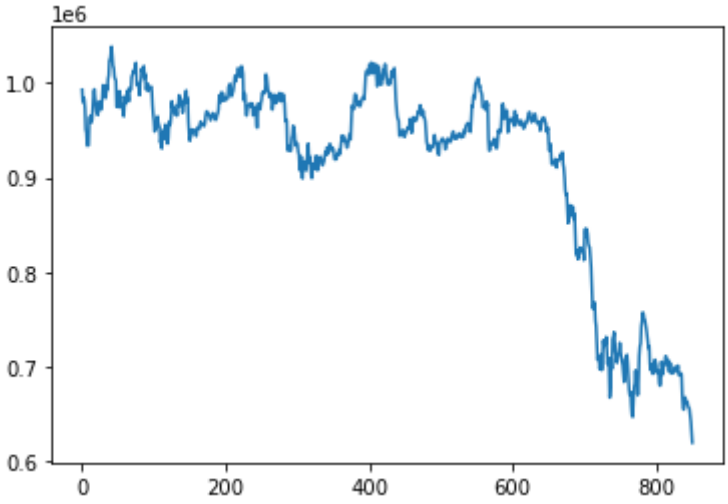


Figure 27 Equal weight portfolio value on bear data

From Figure 27 we can observe that this strategy keeps the portfolio value close to the initial amount until the 650th timestep an then it has a massive loss of near 35%. Near the 800th timestep it has a rebound of 10% and then keeps falling until the last timestep.

3.8.2.4 Mean Variance

This strategy where the objective is trying to maximize rewards where minimizing risk based on Modern portfolio theory had the following performance.

Table 10 Mean variance performance on bear data

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
320409.63	-67.95	619	231

From Table 10 this strategy is the second worst, following Follow-the-Winner on a bear market. It returns -67.95% and has a very high number of rebalances. The portfolio value overtime is presented below.

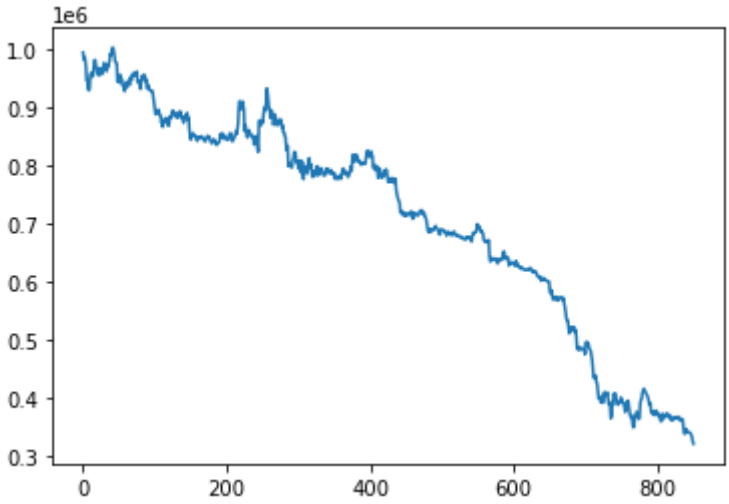


Figure 28 Mean variance portfolio value on bear data

Similarly to follow-the-loser strategy, Mean-variance loses money almost constantly although it seems to have a lower relative portfolio value drop around the 650th timestep comparing to equally weighted portfolios.

3.8.3 Sideways Data

3.8.3.1 Follow-The-Winner

Following the winner on a sideways/lateralization market performs as following:

Table 11 Follow the winner performance on sideways data

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
823927.12	-17.60	181	32

Table 11 shows that this strategy is non-profitable on a sideways market with a return of -17.60% with a very high number of rebalances. The portfolio value during this trading period is presented below:

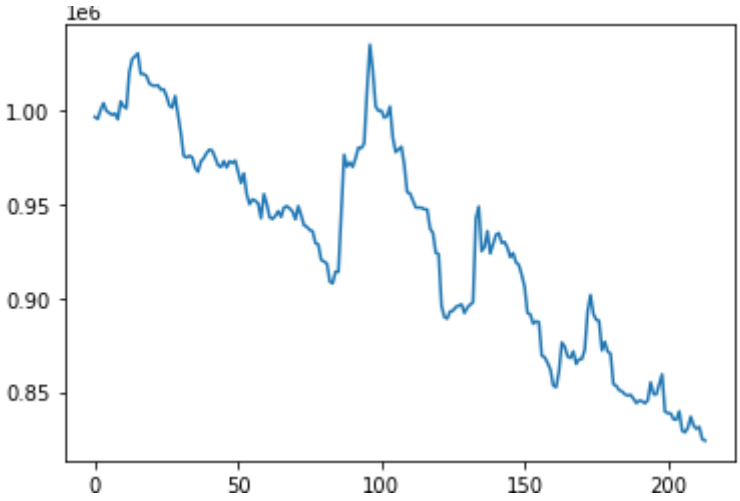


Figure 29 Follow the winner portfolio value on sideways data

As one can see from the figure above, this trading strategy loses about 10% during the first 80 timesteps and then recovers until the 100th timestep. From the 100th timestep until the end, it loses money in cycles and then bounces back a bit and keeps on losing money.

3.8.3.2 Follow-the-Loser

Tracking the loser strategy performance on sideways data is presented below:

Table 12 Follow the loser performance on sideways data

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
881304.44	-11.86	192	21

As we can see from the table above, this strategy outperforms follow the winner but still has a very high number of rebalances and returns -11.86% of the initial portfolio value, which is presented in the figure below.

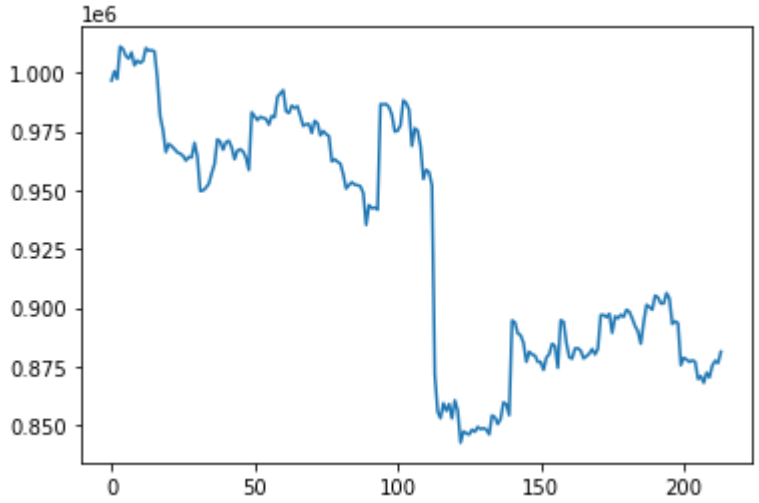


Figure 30 Follow the loser portfolio value on sideways data

From Figure 30 we can see that this strategy is never profitable and around the 100th timestep it has a massive drop of value, nearly 10% in just one timestep.

3.8.3.3 Equal-Weight

An equally weighted portfolio performs as follows:

Table 13 Equal weight performance on sideways data

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
1066539.84	6.65	0	213

Similar to other market conditions, it has the strategy with the best performance. On a sideways data it returns a 6.65% profit being the only profitable strategy so far. The portfolio value overtime for this strategy is presented below:

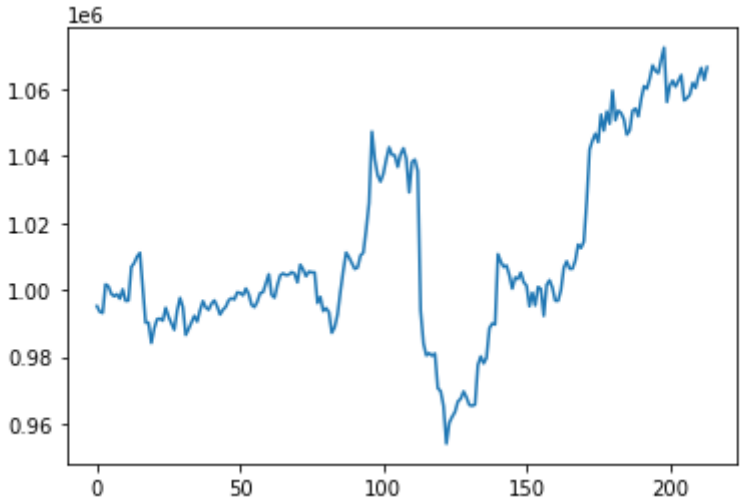


Figure 31 Equal weight portoflio value on sideways data

As we can conclude from the figure above, this strategy keeps the portfolio value until near the 80th timestep, where it starts gaining about 4%, and then it has a significant drop of 8% until the 120th timestep hitting a bottom of -4% of the initial portfolio value and then it rises in value until the end.

3.8.3.4 Mean-Variance

The performance for mean-variance strategy on sideways market conditions is presented in the table below:

Table 14 Mean variance performance on sideways data

Final Portfolio Value	Profit and Loss (%)	Number of Rebalances	Number of Holds
955719.75	-4.42	156	57

We can see that this strategy is the second best following equally weighted portfolio, despite having a negative return of 4.42% and a high number of rebalances. The portfolio value over this trading period is presented below:

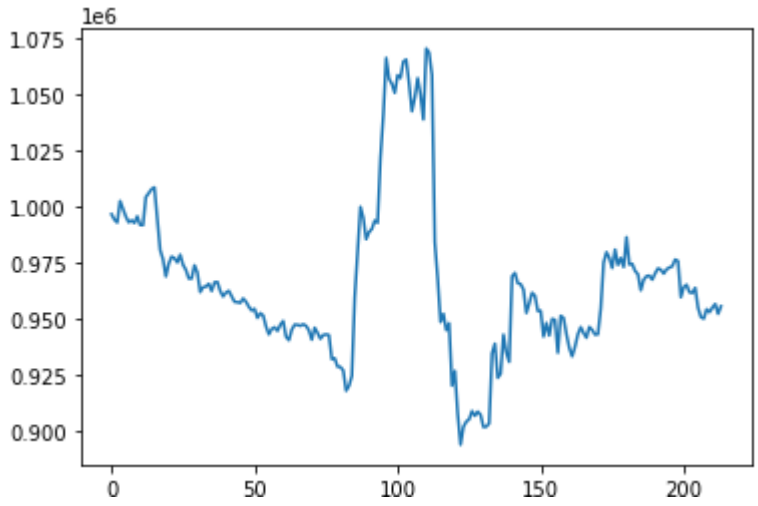


Figure 32 Mean Variance portfolio value on sideways data

We can conclude that this strategy on these market conditions seems to perform better than in other more unstable market conditions. It loses around 7.5% until the 80th timestep where it has a rise of almost 15% hitting a 7.5% profit from the initial portfolio value near the 100th timestep followed by a massive drop of 15% hitting a bottom at 120th timestep, followed by an increase and lateralization until the end of the trading period.

3.9 Tools and Libraries

This project was developed using Python 3.10.8 [59] and in this section all the relevant libraries used are going to be listed below:

- Yahoo! Finance's API (yfinance) [56]. It's an open-source utility that downloads financial historical data from Yahoo's publicly accessible APIs.
- OpenAI gym [60]. Gym is an open-source Python package that provides a common API for communicating between learning algorithms and environments, allowing for the development and comparison of reinforcement learning algorithms.
- Stable-Baselines3 [61]. Stable Baselines3 (SB3) is a collection of trustworthy PyTorch implementations of reinforcement learning methods.
- TensorFlow [62]. TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across many tasks but has a focus on training and inference of neural networks.
- Technical Analysis (ta) [63] . Technical Analysis is a library that leverages the Pandas package with more than 130 Indicators and Utility functions and more than 60 Candlestick Patterns.
- PyPortfolioOpt [64] . PyPortfolioOpt is a library that implements portfolio optimization methods, including classical efficient frontier techniques.

3.10 Summary

This Chapter exposed several classical portfolio management strategies, namely follow-the-winner, follow-the-loser, equally-weighted and mean variance portfolios based on modern portfolio theory. Explained the technical indicators that were used in the state of the environment. Portfolio Management was explained in a mathematical standpoint and how it can be viewed as a Markov Decision Process. An overview of the proposed methodology is presented, as well as all the data description and preprocessing that was used in the following sections. The classical portfolio management strategies were benchmarked to be compared with DRL algorithms in the following sections. Finally, this chapter shows all the libraries and tools used in next chapter.

4 Case Studies description and Analysis

4.1 Case Study Description

The main objective of this thesis is to understand how different Deep Reinforcement Learning algorithms behave according to different market conditions, learning rates and reward functions, and if it is possible to produce a strategy that can outperform the benchmark strategies. A table with the different parameters under evaluation in this case study is presented below.

Table 15 Different testing scenarios

Algorithm	Market Condition	Learning Rate	Reward Function
A2C	Bull	0.0001	R1 (Profit)
DDPG	Bear	0.001	R2 (Outperform Btc)
SAC	Sideways	0.01	R3(Outperform Btc and USDC)
PPO	-	0.1	-
TD3	-	-	-

As discussed in section above, we are going to explore five different DRL algorithms under three different market conditions, namely Bull, Bear and Sideways. For each algorithm four different learning rates are going to be explored to assess which algorithms converge faster. On top of all that, three different reward functions are going to be explored in order to see if smart/suggestive rewarding can help DRL algorithms. The different reward functions are explained below.

- **R1**: This reward consists of the simple profit relative to the last time step. For example, if the portfolio value in the previous timestep was 500 and in the current timestep is 700, the reward is going to be 200.
- **R2**: In this reward instead of going for pure profit each time step, the reward is going to be the difference between the current portfolio value and the value of the portfolio if the allocation consisted 100% of Bitcoin. Simply put, the objective of this reward is to outperform Bitcoin instead of going for profits. For example, if the current value of the portfolio is 700, the last timestep was 500 and this timestep, if we had an 100% Btc allocation the value was 900, then the reward would be -200 (700-900) instead of a positive reward of 200 (700-500) that would be obtained with R1.
- **R3**: This reward is an extension of **R2** where the objective is to outperform both BTC and the riskless asset USDC. This reward is the sum of the difference in portfolio value of the action taken with holding 100% BTC and the difference of the action taken with holding 100% USDC. This might be useful because if BTC is falling in price and the

portfolio value is falling at an approximate rate, instead of having a near zero reward, the reward would be negative.

The difference scenarios evaluated in this study will be 5 (models) * 3 (market conditions) * 4 (learning rates) * 3 (reward function) being a total of 180 scenarios.

The following sections will be divided in three main sections according to the market conditions. Each one of these sections will be subdivided according to the reward function used. Inside each of these subsections 5 models with 4 learning rates are going to be presented. In the figure below is presented the colour of each asset in the portfolio.

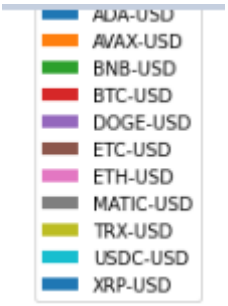


Figure 33 Asset Colour for portfolio images

4.2 Bull Market

4.2.1 Data Visualization

In the figure below it is possible to see the price change in BTC for the training data.

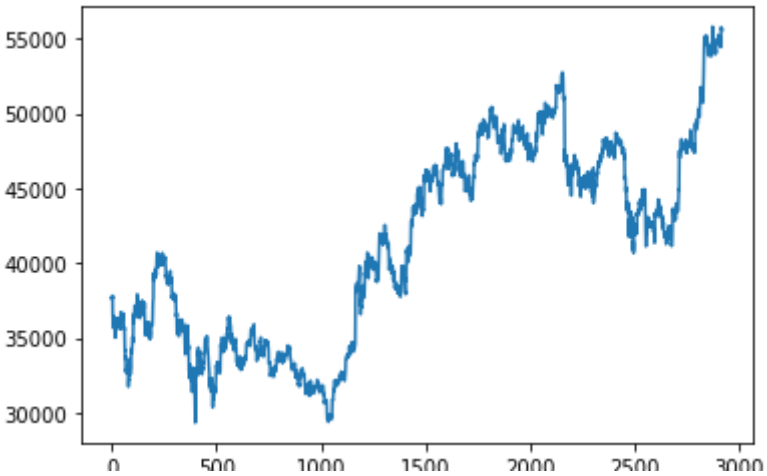


Figure 34 Bitcoin price evolution on training bull data

From this picture it is possible to see that there was a big spike in price. In the first timesteps the price falls a bit but then it has a massive rally. In the table below we can see the price variation from the first timestep to the last for each asset.

Table 16 Training bull-data asset percent change

Asset	Price at first timestep	Price at last timestep	% change
ADA-USD	1.76	2.27	28,97727
AVAX-USD	19.00	60.35	217,6316
BNB-USD	417.01	421.76	1,141487
BTC-USD	37735.25	55836.95	47,97027
DOGE-USD	0.38	0.24	-36,8421
ETC-USD	66.90	56.08	-16,1734
ETH-USD	2790.22	3597.54	28,91457
MATIC-USD	1.67	1.35	-19,1617
TRX-USD	0.07	0.10	42,85714
USDC-USD	1.00	0.99(9)	-0,1
XRP-USD	0.98	1.21	23,46939

From this table we can see that bitcoin had a price change of 47.97%, it is safe to say that models will be trained on bullish data. The best performance assets are AVAX with 217% price change, followed by BTC with 47.97% and the third one is TRX with 42.85%. The assets with the poorest performance were DOGE with -36.84%, followed by MATIC with -19% and USDC with a minor change in price of -0.1%.

For the test data, it is important that the split maintains an upward trend, and we can observe the BTC price evolution for the testing data in the figure below.

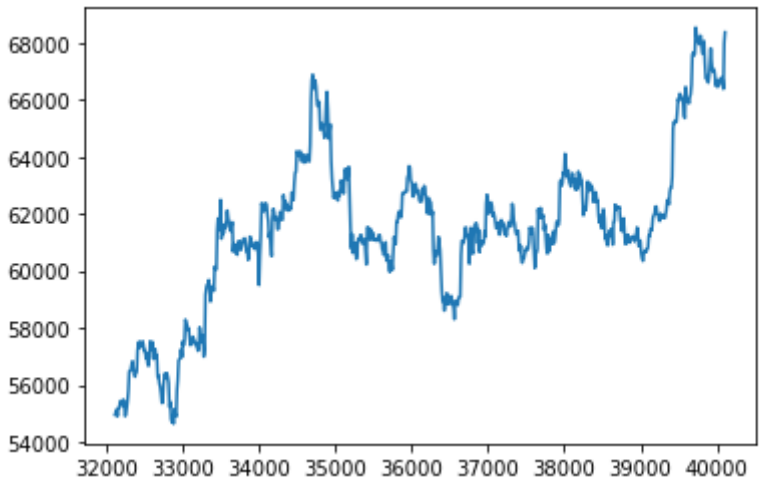


Figure 35 Testing bull data bitcoin price evolution

As we can see, one can say that this data is still a bull market, although it is not as upward as the training data. We can see the price variation in the table below.

Table 17 Testing bull data asset percent change

Asset	Price at first timestep	Price at last timestep	% change
ADA-USD	2,257	2,256	-0,044306602
AVAX-USD	59,85	89,66	49,80785297
BNB-USD	419,17	653,71	55,95343178
BTC-USD	54949,88	68363,5	24,41064475
DOGE-USD	0,24	0,27	12,5
ETC-USD	55,33	62,42	12,81402494
ETH-USD	3572,88	4843,04	35,55003247
MATIC-USD	1,33	1,83	37,59398496
TRX-USD	0,1	0,11	10
USDC-USD	0,9999	1,0001	0,020002
XRP-USD	1,193	1,31	9,807208718

From this table we can see that the most performative asset was BNB with a 55% increase in its price, followed by AVAX and MATIC. Only ADA had a negative price change of -0.04%.

4.2.2 A2C

4.2.2.1 Reward R1

In the figure below it is possible to see the portfolio value over the testing data using the reward R1 with the objective of maximizing the profit per timestep.

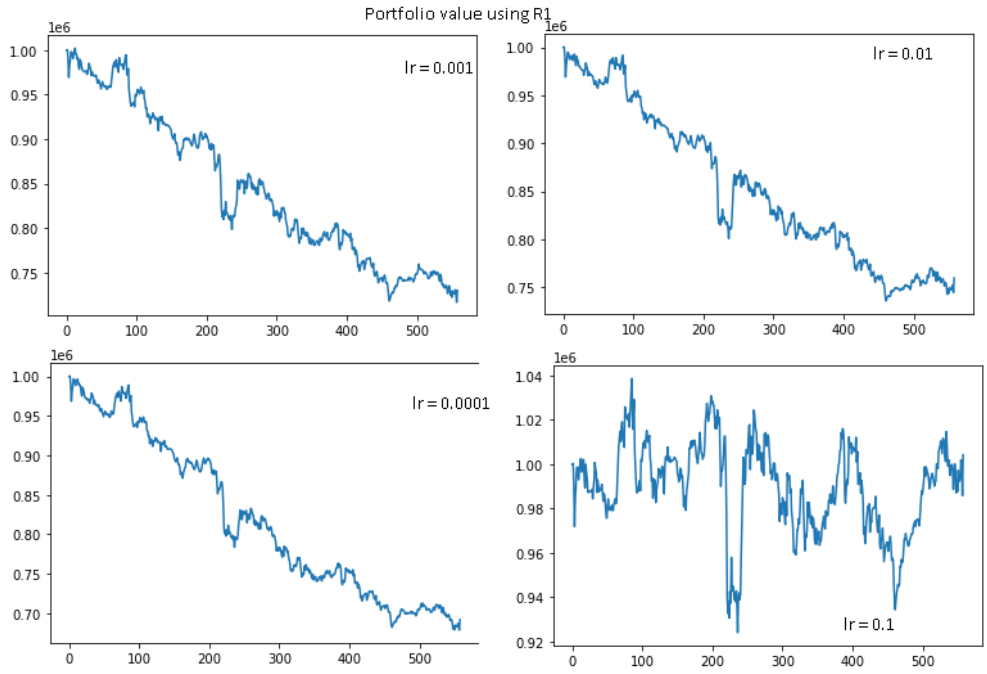


Figure 36 A2C portfolio value using R1 on Bull data

From this figure we can see that the results were very negative for the first three learning rates with the exception of utilizing a learning rate of 0.1. In order to understand better what is going on it is important to take into account the number of rebalances made in this trading period. The table below summarizes the profits and losses and the number of rebalance and holds.

Table 18 A2C Performance using R1 on a bull market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	691923.52(-30.80 %)	558	0
0.001	730147.82(-26.98%)	557	1
0.01	759477.25 (-24.05 %)	450	108
0.1	1004266.09 (+0.42 %)	190	368

We can see that only with a learning rate of 0.1 the model had profit and the higher the amount of rebalances, the lower the performance of a model is. One important thing to remember is that a rebalance cost of 0.1% at first glance seems a small loss, but if the model does it every timestep it turns into compound loss or exponential decay of the portfolio value. Let's look at the first example with the learning rate of 0.0001 and 558 rebalances. With the

trading cost of 0.1%, if all assets maintained their prices, just by rebalance at all timesteps it would result in a loss of 74.67% [(1.001^558)-1]. It seems that with a learning rate of 0.01 and 0.1 the models start to learn the art of holding but they are still not very effective.

4.2.2.2 Reward R2

Utilizing the second reward function, with the objective of outperforming bitcoin each timestep we obtained the following results expressed in the figure below.

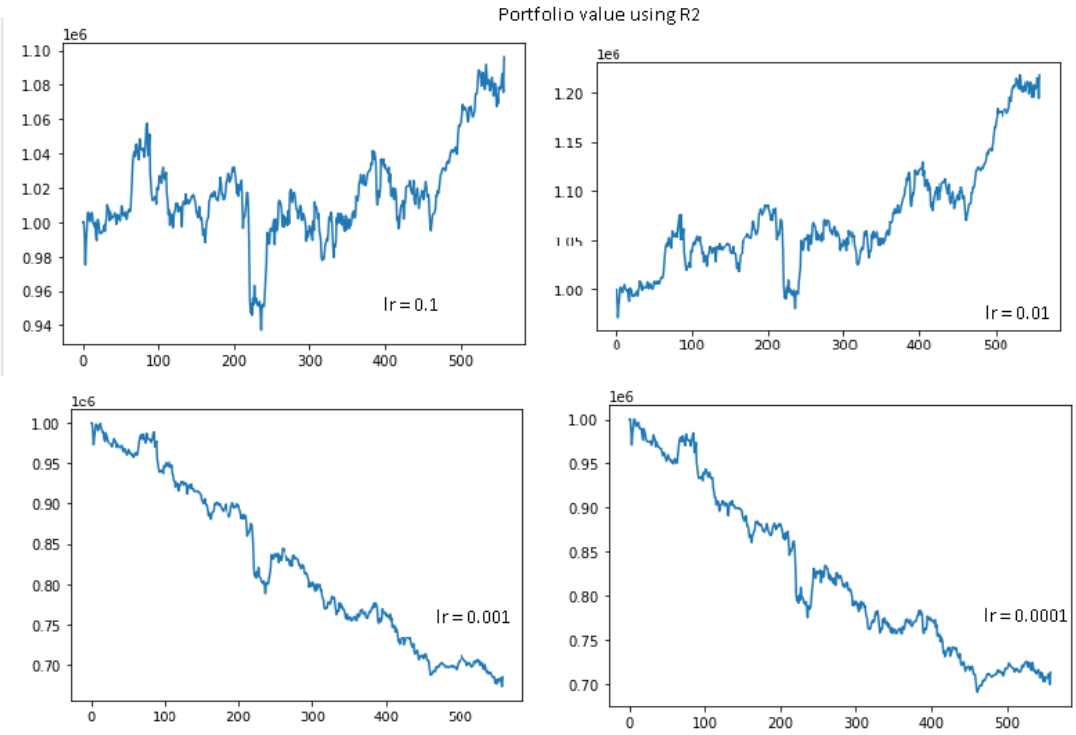


Figure 37 A2C portfolio value using R2 on bull market

Comparing with the first reward, the results for lr=0.1 and lr=0.01 look much better. With the lower learning rates the model still only loses money. In the table below we can see how this reward behaved.

Table 19 A2C perfomance using R2 on bull market

Learning Rate	**Final Portfolio Value (+ profit%)**	**Number of portfolio rebalances**	**Number of holds**
0.0001	713298.17(-28.67)	558	0
0.001	683319.12(-31.66)	558	0

0.01	1218790.21(+21.87)	7	551
0.1	1096222.72(+9.62)	95	463

For the lower learning rates the model still rebalances the portfolio at each timestep, therefore it loses about 30% of the initial portfolio value. The learning rates of 0.01 and 0.1 are now performing much better and we can observe the number of portfolio rebalances has dropped significantly. With this reward the model that performed the best was with a learning rate of 0.01 with a performance of 21.87%, very close with the best performance benchmark strategy equal-weight which had a return of 22.08%.

4.2.2.3 Reward R3

Reward R3 has the objective of trying to outperform BTC and USDC at each timestep. The results using this reward are presented in the figure below.

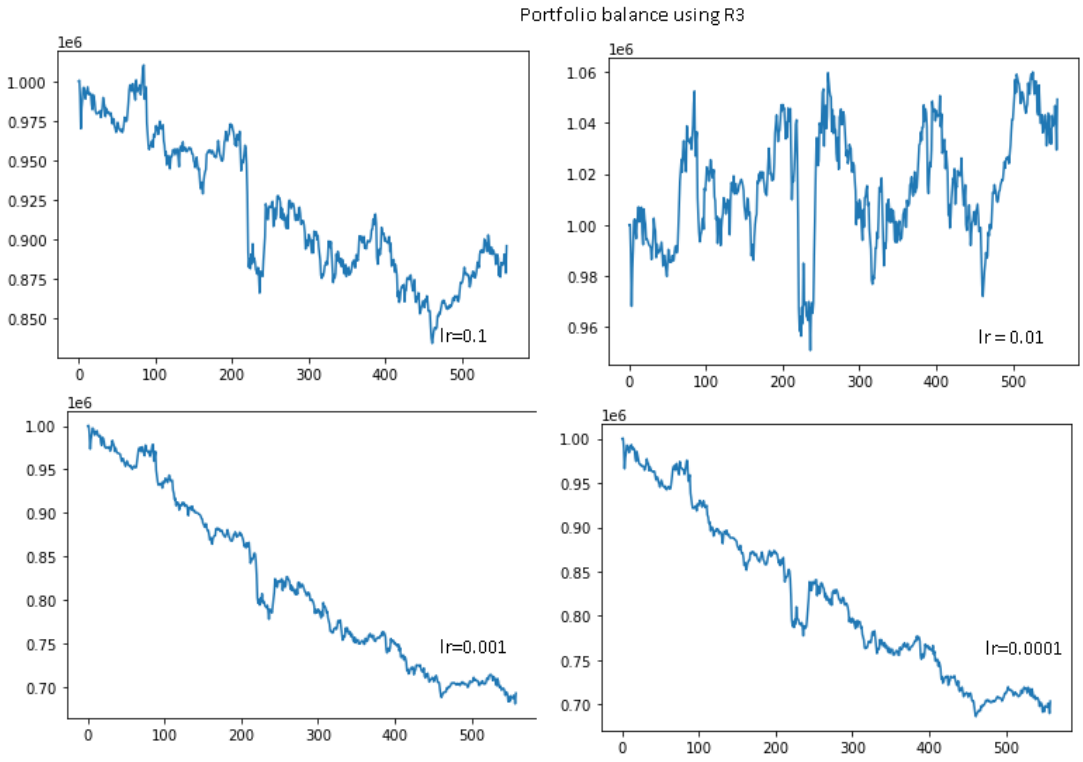


Figure 38 A2C portfolio value using R3 on a bear market

The results look more unstable than using R2, with only one model having a positive performance. In the table below we can see the summary for all models.

Table 20 A2C performance using R3 on bull market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	703682.24(-29.63)	558	0
0.001	693274.58(-30.67)	549	9
0.01	1049232.86(+4.92)	184	374
0.1	895681.77(-10.43)	270	288

Similarly to the previous rewards, the lower learning rates have a very high number of rebalances while for learning rates of 0.01 and 0.1 they start to hold with a lot more consistency. The most performative model was produced using a learning rate of 0.01 and obtained a performance of 4.9%.

4.2.3 DDPG

4.2.3.1 Reward R1

Deep Deterministic Policy Gradient obtained the following results when trying to maximize profit per timestep.

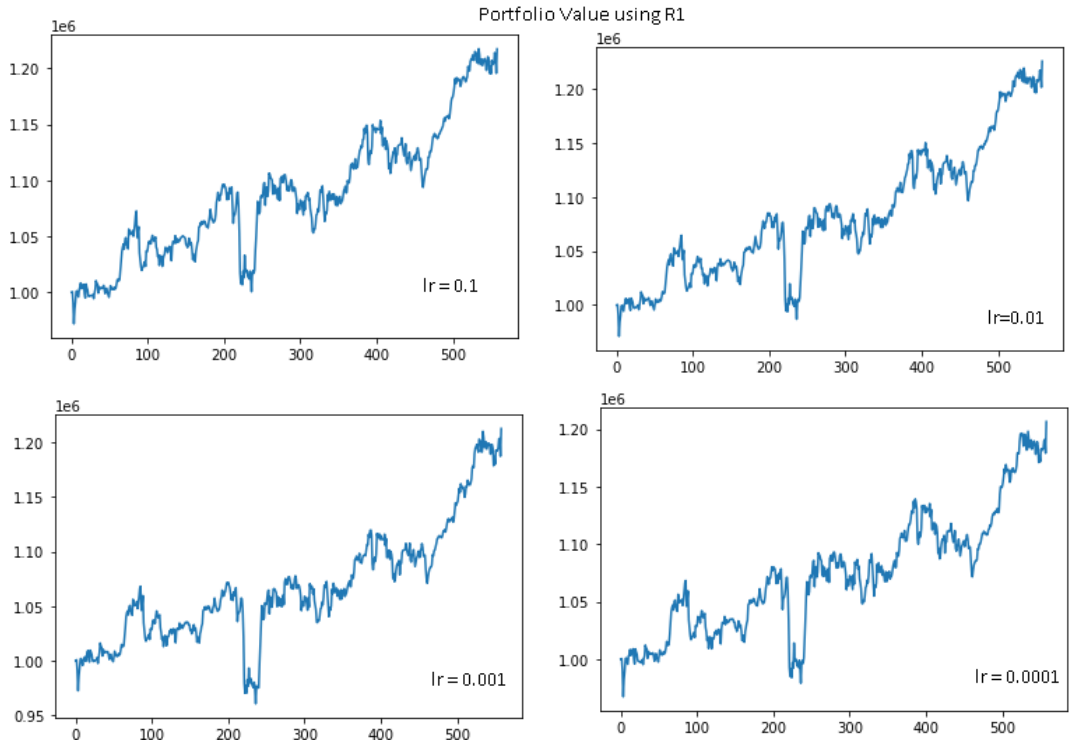


Figure 39 DDPG portoflio vale using R1 on bull market

The results look very stable, and all learning rates produced a return above 20%. In the table below we can see what they did different than A2C.

Table 21 DDPG performance with R1 on bull market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	1206485.95(+20.64)	1	557
0.001	1212559.02(21.25)	1	557
0.01	1225705.44(+22.57)	1	557
0.1	1217422.54(+21.74)	1	557

For all the models, only one rebalance was executed. For the first time we outperformed the best benchmark strategy. With a learning rate of 0.01, the model obtained a performance of 22.57%, an increase of 0.49% comparing with an equal-weight portfolio. Remembering that we started with a portfolio of 100% allocation to BTC, the model at the first timestep rebalanced the portfolio to the ones presented in the figure below and held until the end of the trading period.

As

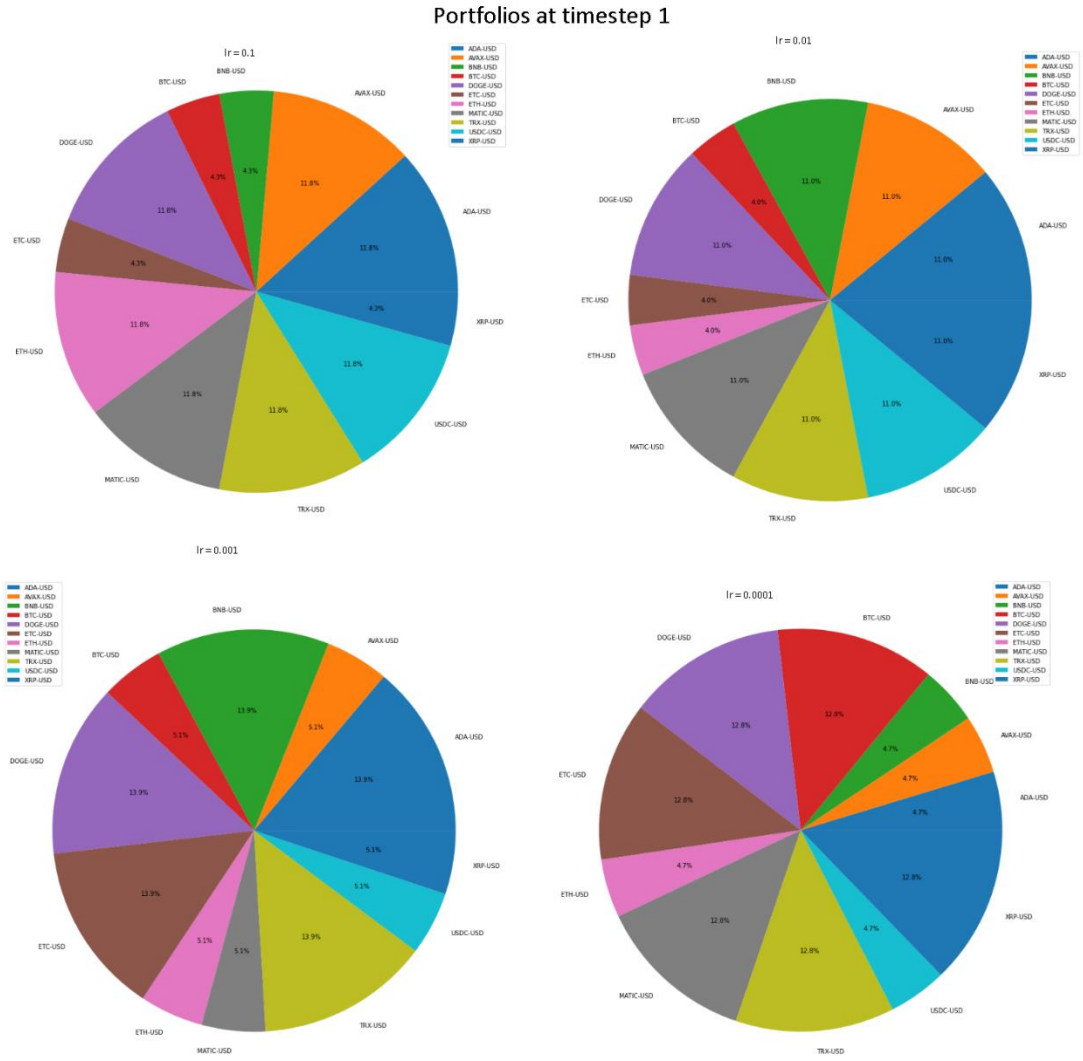


Figure 40 DDPG portoflios with R1 on a bull market

As we can see from this figure, all models are following approaches like the equal weighted portfolios, but instead of diving equally among all assets, it chooses divides into two categories. The high allocation assets and the lower allocation assets and divides them equality inside those categories. Optimally the highest performing assets in the training data, AVAX, BTC, TRX (orange, red, yellow), should be included in the highest allocation category while DOGE, MATIC (purple,grey) should be included in the lowest allocation. None of the models follow this policy meaning there is still room for improvement.

4.2.3.2 Reward R2

Trying to outperform BTC, DDPG obtained the following results.

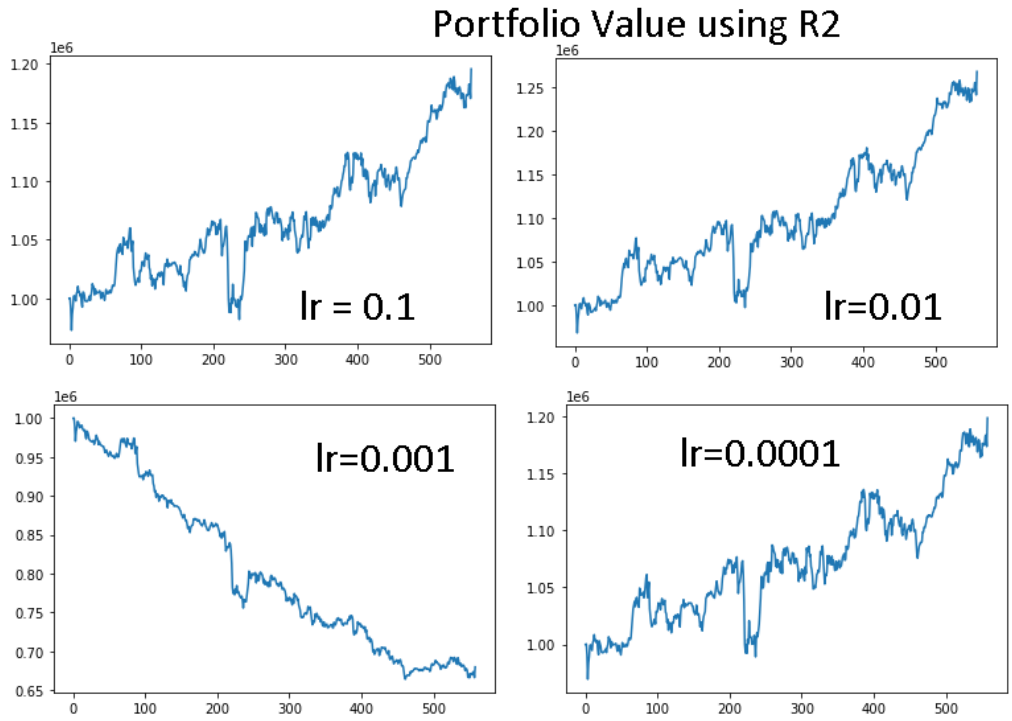


Figure 41 DDPG portfolio value using R2 on a bull market

This reward seems to be a bit more difficult for the models to understand, with now a model with a learning rate of 0.001 performing very poorly. We can see the overall performance in the table below.

Table 22 DDPG perfomrance using R2 on a bull market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	1198663.36(+19.86)	1	557
0.001	679869.42(-32.01)	558	0
0.01	1267896.18(+26.78)	1	557
0.1	1195720.91(+19.57)	1	557

From this table, we can see that learning rate 0.001 could not learn how to hold and tried to rebalance the portfolio at each timestep, while the others remained like the first reward. With a learning rate of 0.01 the performance was 26.78% improving significantly compared to the previous best model and the equal-weight portfolio strategy. In the figure below we can observe the allocations for the models that hold through the trading period.

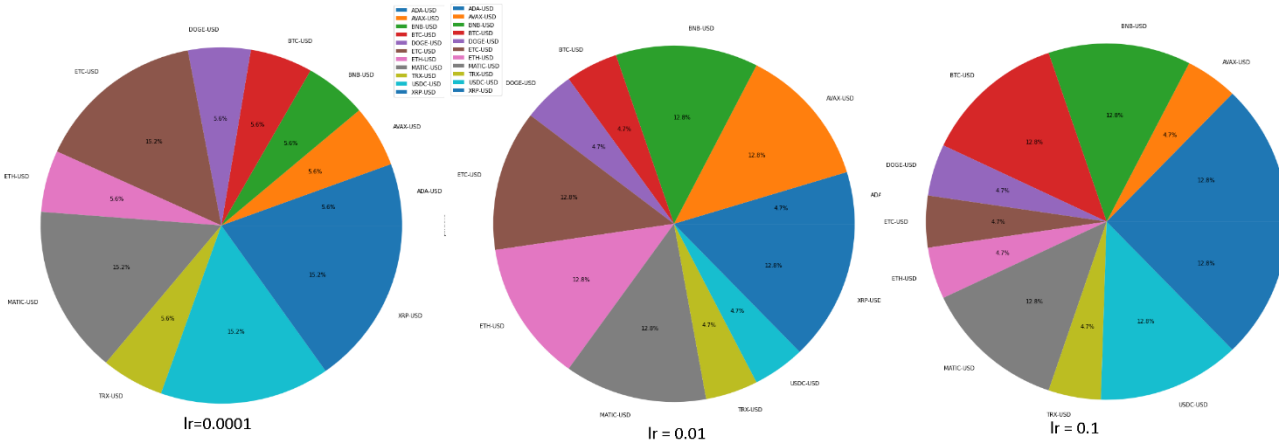


Figure 42 DDPG portoflios using R2 on a bull market

From this figure we can conclude that 0.01 obtained a better performance than the other models because it chose a low allocation for USDC-USD pair represented with the light blue colour. The allocation is still not optimal considering the training data. Only with a learning rate of 0.01 the most performative asset AVAX represented with the colour orange was included in the high allocation category. For all the portfolios above the lowest performative asset in the training data, DOGE (purple) was included in the low allocation category.

4.2.3.3 Reward R3

Trying to outperform BTC and USDC, DDPG obtained the following results during the trading period.

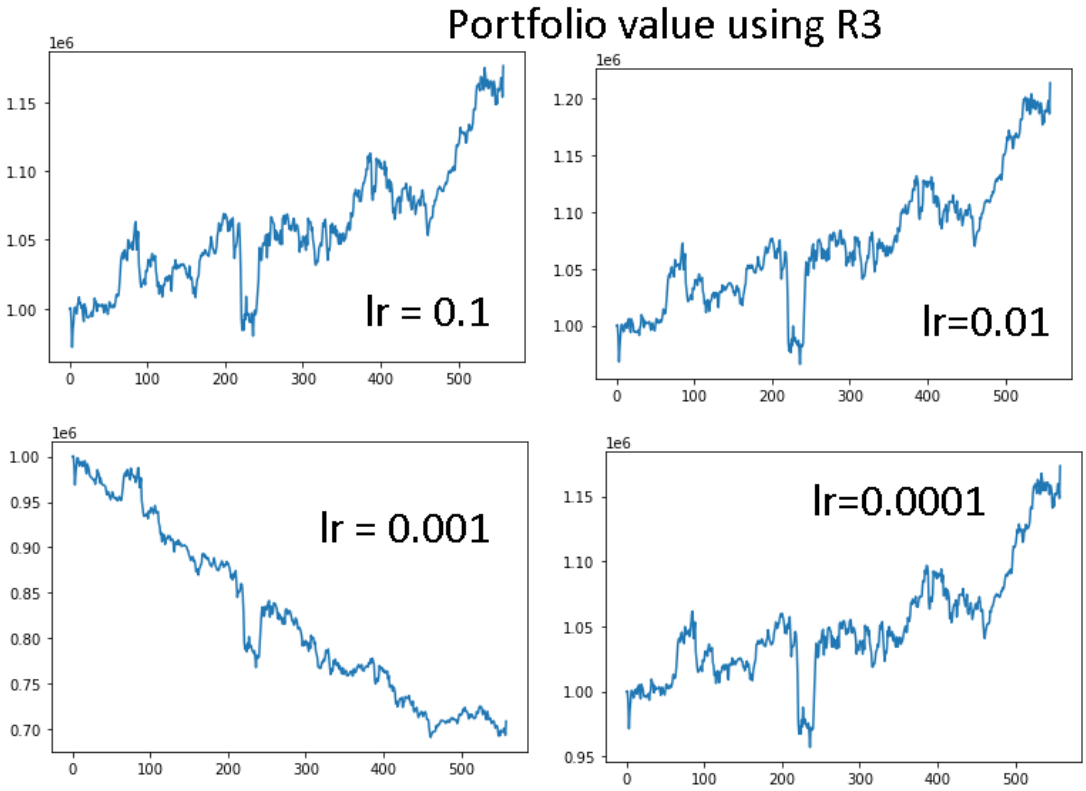


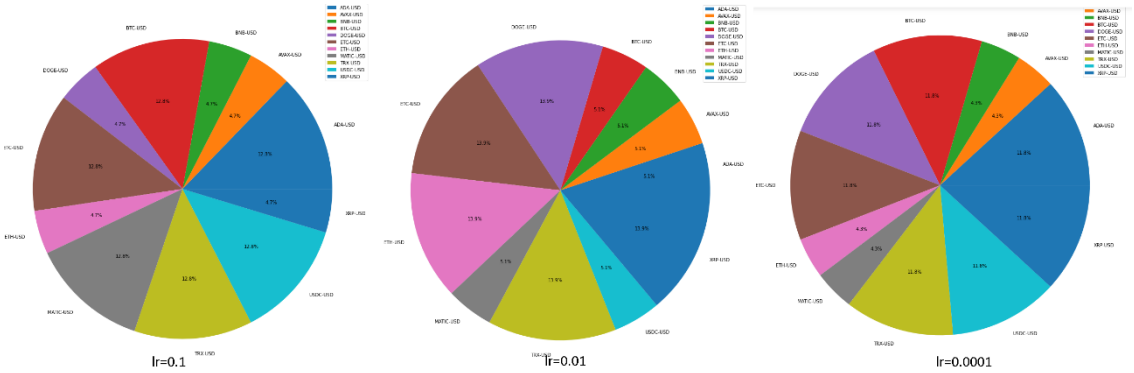
Figure 43 DDPG portfolio value using R2 on a bull market

Similarly to R2, all models perform relatively good with the exception of the model that uses a learning rate of 0.001. The performance of each model is presented below:

Table 23 DDPG performance using R3 ona bull market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	1173284.76(+17.32)	1	557
0.001	709190.95(-29.08)	557	1
0.01	1213301.58(+21.33(	1	557
0.1	1176564.93(+17.65)	1	557

From this table we can see that this reward produced an overall worst performance among the models that learnt how to hold. Once more a learning rate of 0.01 seems to produce the best results, but none of the models could outperform equal-weight strategy. The allocation is presented in the figure below.



Similarly to previous conditions, the model with a learning rate of 0.01 has better results because it holds less of USDC than other models.

4.2.4 PPO

4.2.4.1 Reward R1

Proximal policy optimization obtained the following results when trying to maximize profit each timestep.

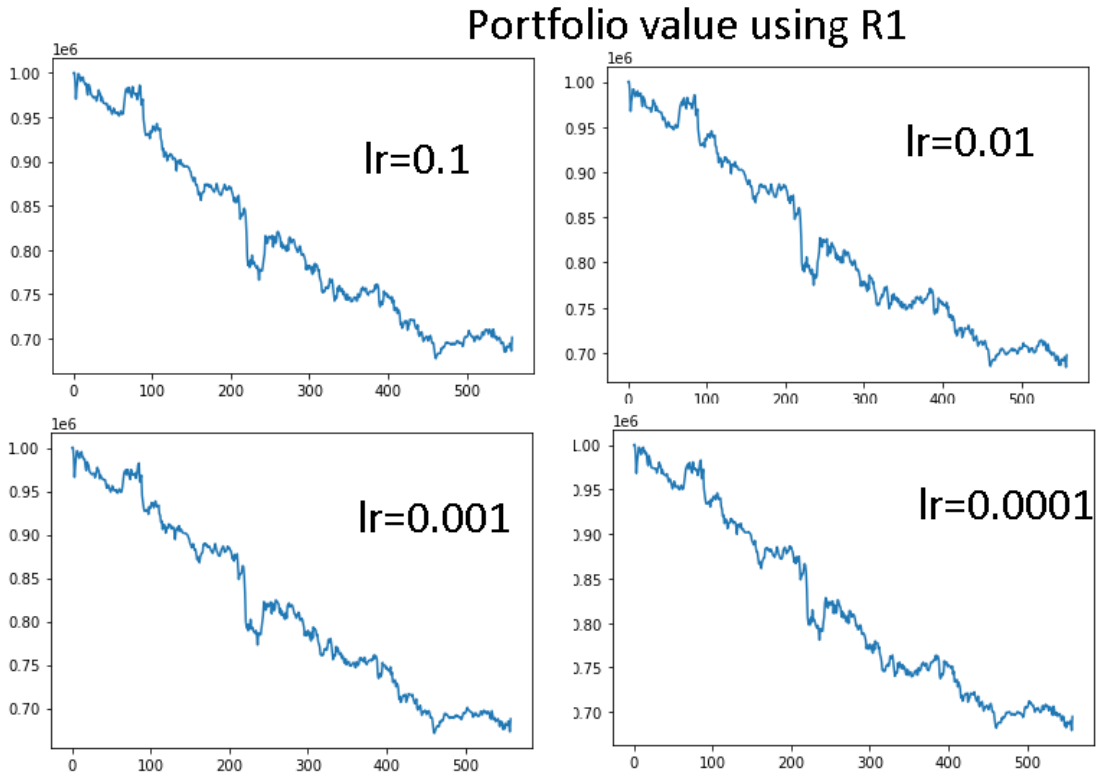


Figure 44 PPO portfolio value using R1 on a bull market

From this figure we can see that PPO produced the worst results so far. In the table below it is a list the summary of the results.

Table 24 PPO performance uysing R1 on a bull market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	694741.28(-30.52)	558	0
0.001	687641.81(-31.23)	558	0
0.01	697603.64 (-30.23)	558	0
0.1	700785.17(-29.92)	542	16

We can see that none of the learning rates learned how to hold properly, although we can see that with a learning rate of 0.1 it holds 16 out of the 558 steps, but the results are still very poor.

4.2.4.2 Reward R2

Proximal policy optimization obtained the following results when trying to outperform Bitcoin.

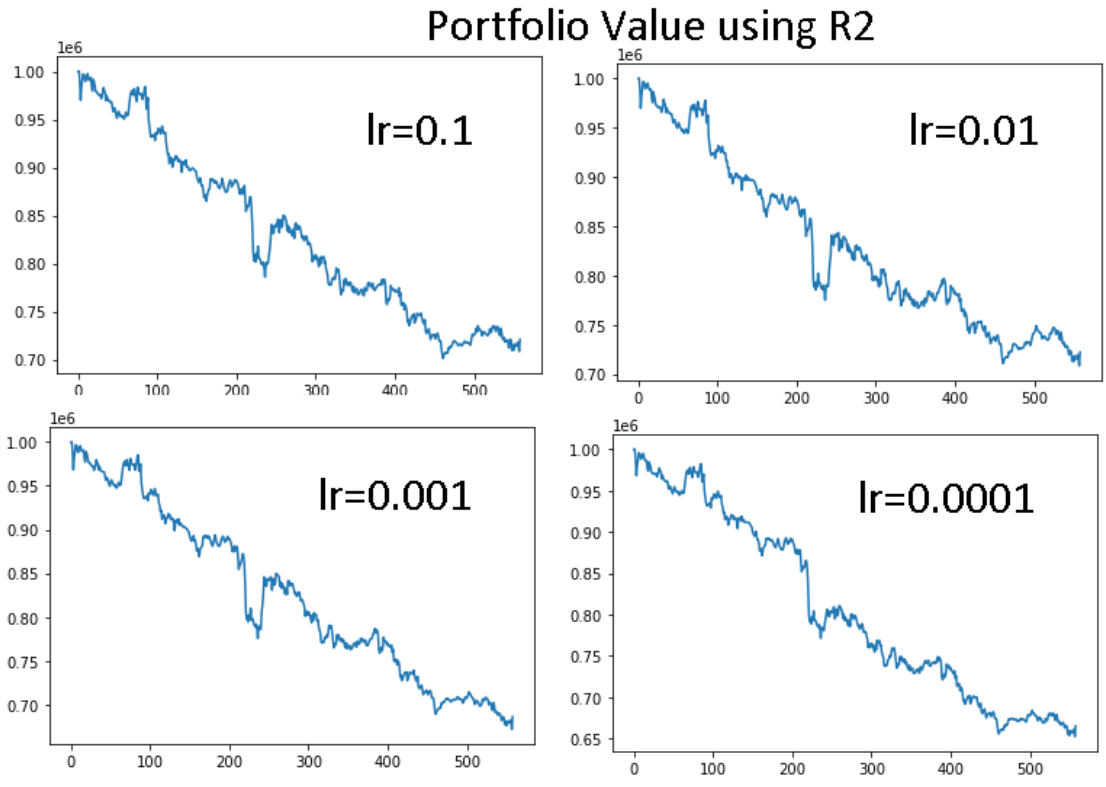


Figure 45 PPO portfolio value using R2 on bull market

Again, the results are very similar to the previous reward, and the model seems like it learns slower than any other model. In the table below it is the summary of the results.

Table 25 PPO performance using R2 on a bull market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	664780.16(-33.52)	558	0
0.001	685977.26(-31.40)	558	0
0.01	722196.47(-27.78)	558	0
0.1	720717.25(-27.92)	558	0

All models produced very poor results due to the fact that it is rebalancing the portfolio every timestep.

4.2.4.3 Reward R3

Proximal policy optimization obtained the following results when trying to outperform BTC and USDC.

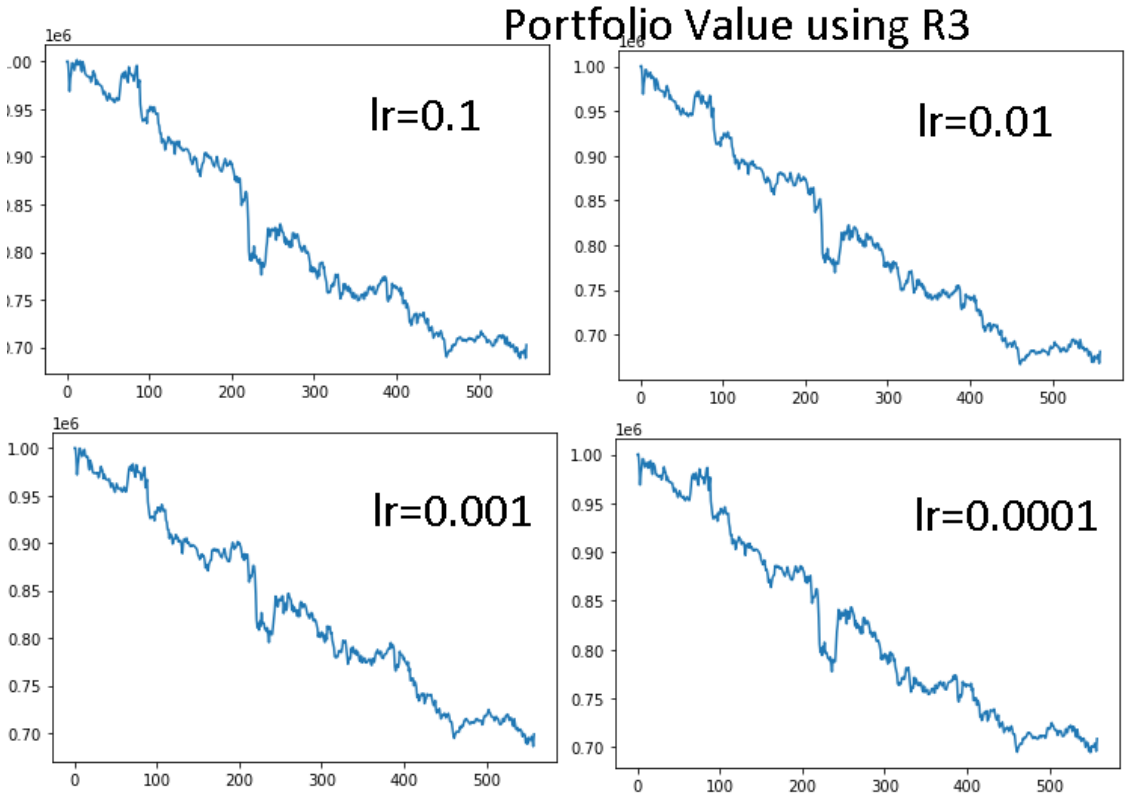


Figure 46 PPO portfolio value using R3 on a bull market

Once again, the results are very bad. This is expectable since, if the model can't learn a basic reward like R1, it certainly cannot learn from a more complex reward. Table below shows the results for all the models.

Table 26 PPO performance using R3 on a bull market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	708499.68 (-29.15)	558	0
0.001	698698.70 (-30.13)	558	0
0.01	681204.46 (-31.88)	558	0
0.1	702510.30 (-29.74)	558	0

The model failed to learn every reward including R3. Comparing with other models looks like it learns slower. The only holds were produced with R1 and it still was a very low number.

4.2.5 SAC

4.2.5.1 Reward R1

Soft-actor-critic trying to maximize profit produced the following results:

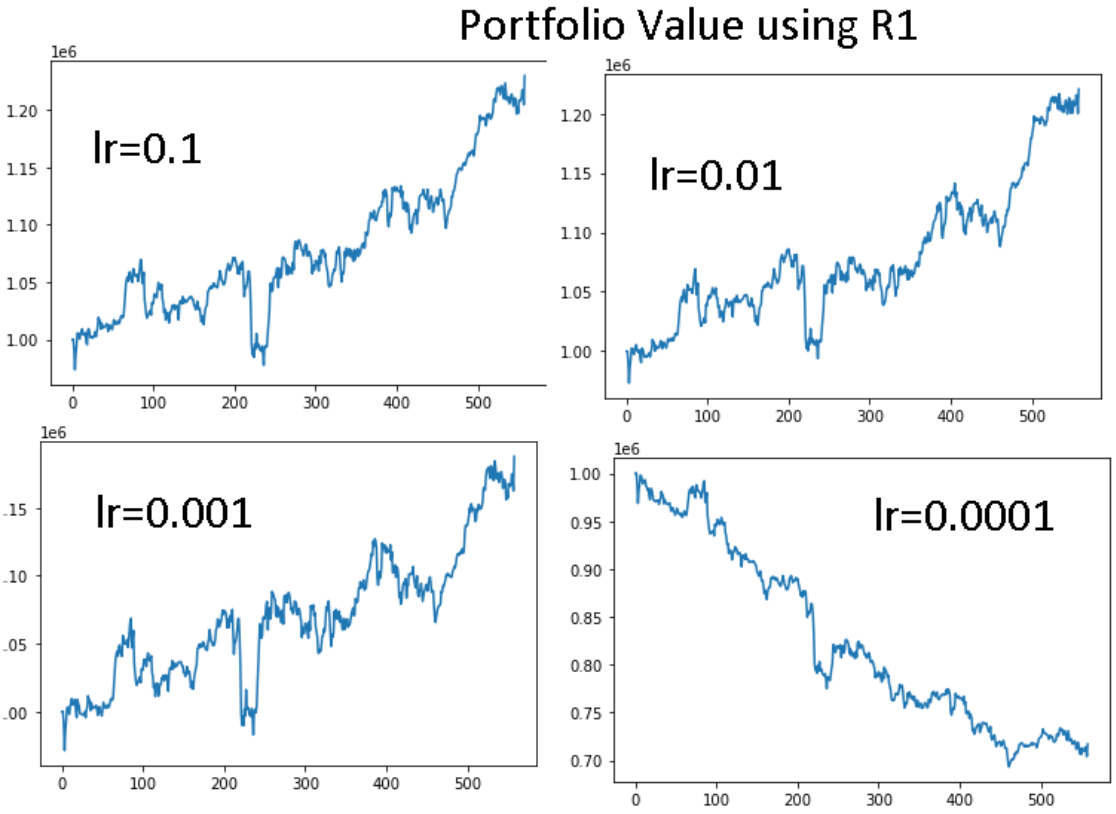


Figure 47 SAC portfolio value using R1 on a bull market

All models seem to have profit with the exception of a learning rate of 0.0001. A detailed description of the results is presented below:

Table 27 SAC performance with R1 on a bull market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	717184.92 (-28.28)	558	0
0.001	1187765.00 (+18.77)	1	557
0.01	1220621.39 (+22.06)	1	557
0.1	1229887.39 (+22.98)	1	557

Similarly to profitable models presented above, SAC learns that the best strategy is to rebalance the portfolio at the first timestep and not rebalance to avoid losing money to fees. The portfolio allocations according to the learning rate is presented below.

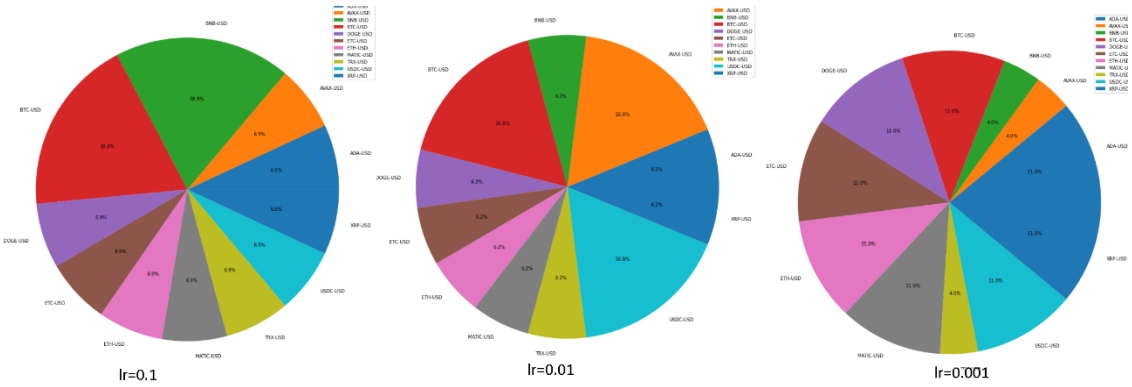


Figure 48 SAC portfolios using R1 on a bull market

From Figure 48, we can see that the most performant model with a learning rate of 0.1 has a very high amount of BNB represented by the green colour when compared to other portfolios. This probably happened randomly since BNB is one of the most underperforming assets in the training data. Bitcoin, which was the second most performative in the training data, in all cases, has a very high allocation.

4.2.5.2 Reward R2

Trying to outperform BTC, SAC produced the following results.

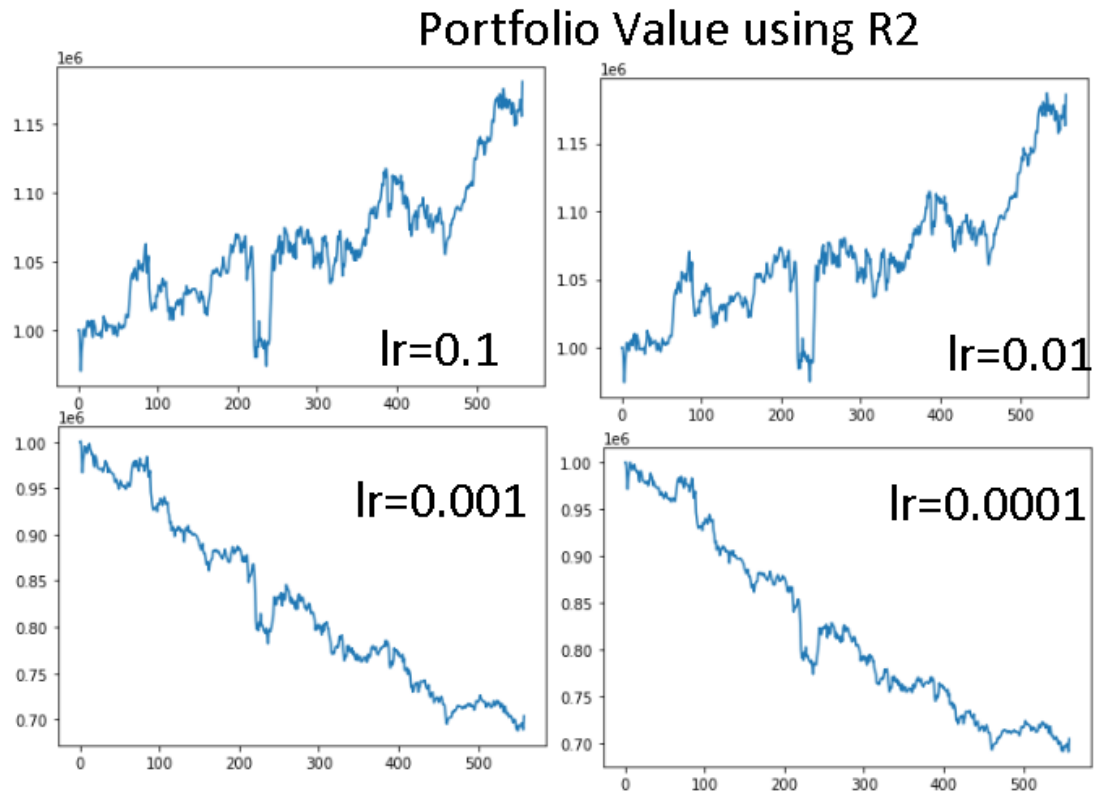


Figure 49 SAC portfolio value using R2 on a bull data

With R2, the model seems to have more difficulty to understand how to hold. With R1, and a learning of 0.001, the model learns how to hold and with R2 it fails. We can analyse the behaviour of the models below:

Table 28 SAC performance using R2 on bull data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	705095.31 (-29.49)	558	0
0.001	703737.60 (-29.62)	558	0
0.01	1185860.55 (+18.58)	1	557
0.1	1180449.87 (+18.04)	1	557

This reward produced worst results than R1, and in the cases where it learnt how to hold, namely lr=0.1 and lr=0.01 it produced about 4% lower returns. The portfolio allocations according to the learning rate is presented below.

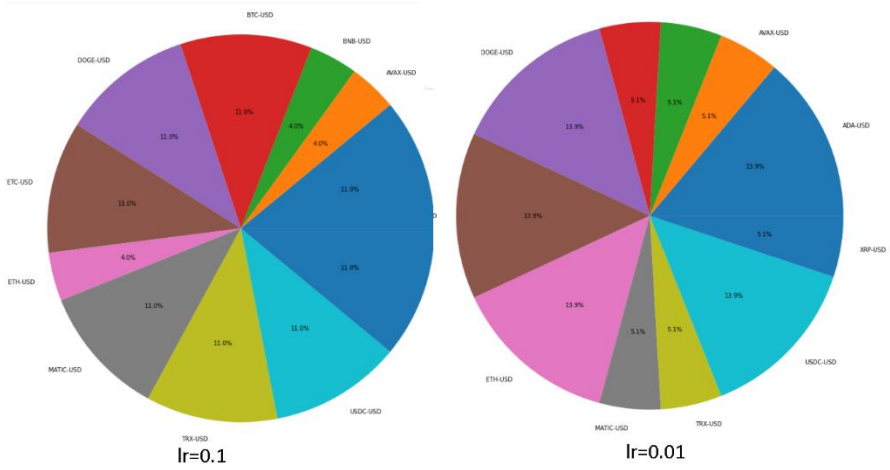


Figure 50 SAC portfolios using R2 on bull data

In both cases, the portfolios don't seem to have learnt which of the assets were the most performative from the training data. The portfolio with a learning rate of 0.1 has a high allocation of BTC(red) and TRX (yellow), which performed very good on training data, but also has a high allocation for MATIC(grey) and USDC (light blue) which didn't perform well. Also, with a learning rate of 0.01 the most performative assets have a low allocation and some of the least performative assets have high allocations such as DOGE (purple) and ETC (brown).

4.2.5.3 Reward R3

SAC with the objective of outperforming BTC and USDC produced the following results:

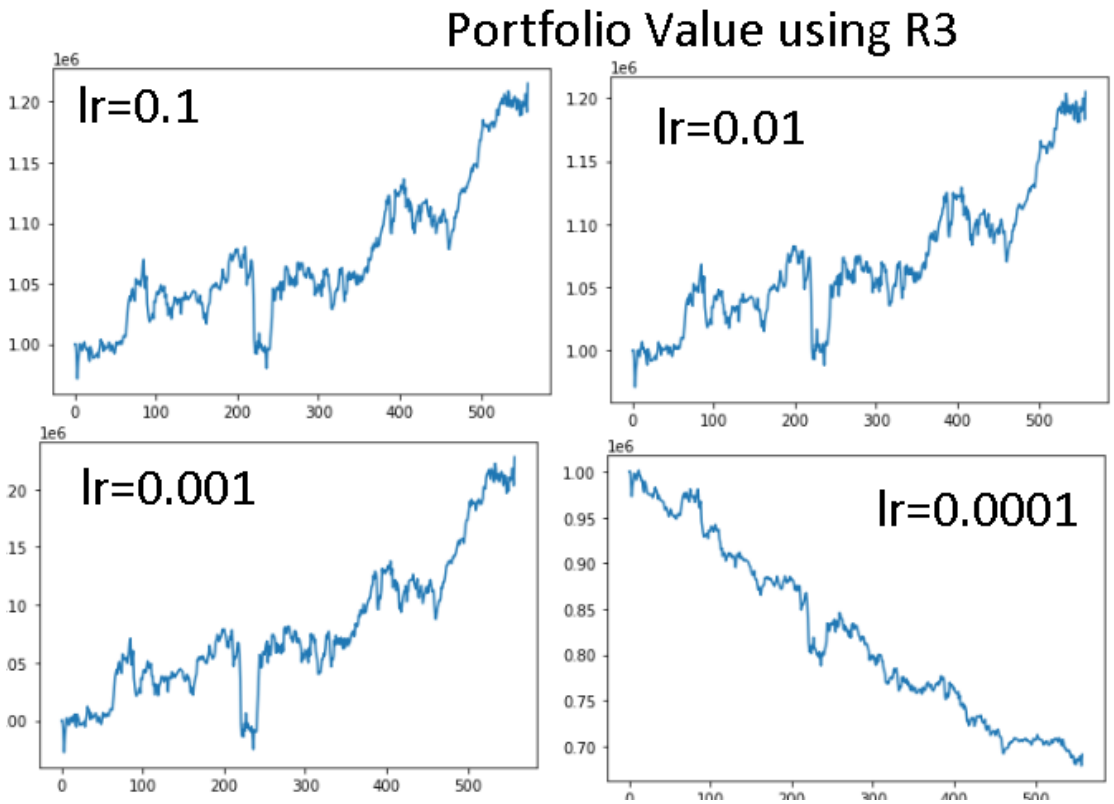


Figure 51 SAC portfolio value using R3 on bull data

This algorithm seems to learn faster from this reward than R2. Similarly to R1, only with a learning rate of 0.0001, this algorithm wasn't able to learn the holding strategy.

Table 29 SAC performance with R3 on bull data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	690878.73 (-30.91)	558	0
0.001	1227820.63 (+22.68)	1	557
0.01	1204866.09 (+20.48)	1	557
0.1	1215078.78 (+21.50)	1	557

This reward produced results very similar to R1. The best model outperformed equal-weight portfolios with a learning rate of 0.001, but still was unable to outperform the DDPG model with a learning rate of 0.01. The portfolio allocations for the models that learned how to hold is presented in the figure below:

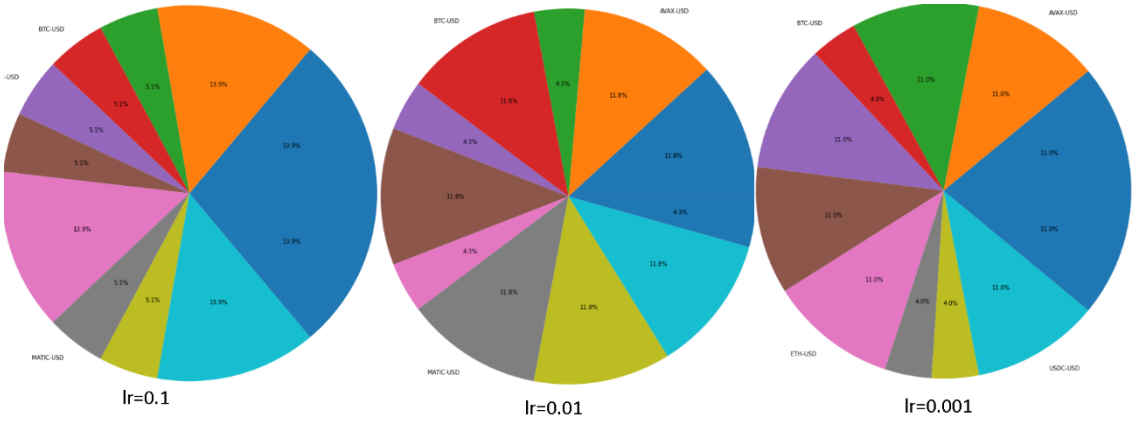


Figure 52 SAC portfolio using R3 on bull data

The portfolios look very similar with all learning rates and all included the most performative asset on training data AVAX (orange) on a high allocation category. The learning rate of 0.0001 performed better than other models probably because it was the only one who included BNB (green), which was the most performative on test data, on a high allocation category.

4.2.6 TD3

4.2.6.1 Reward R1

TD3 maximizing profit each timestep performance is presented in the figure below:

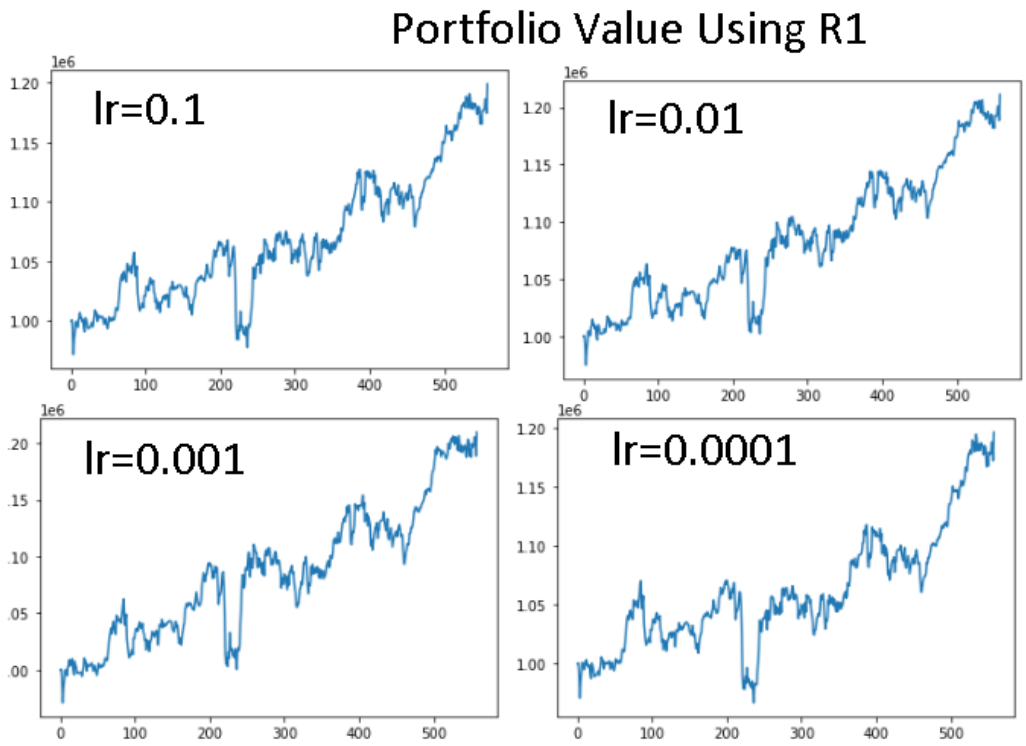


Figure 53 TD3 portfolio value uysing R1 on bull data

Similarly to DDPG this algorithm learns holding with all learning rates and produced profitable strategies across all the learning rates. The performance of TD3 on a bull market is presented below:

Table 30 TD3 performance with R1 on bull data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	1196034.35 (+19.60)	3	555
0.001	1209485.52 (+20.94)	17	541
0.01	1211168.44 (21.11)	1	557
0.1	1199038.98 (19.90)	1	557

As one can observe from the table above, the learning rates of 0.0001 and 0.001 didn't hold across all learning steps but have a very low number of rebalances and still were able to produce a profitable strategy. The most performative model was with a learning rate of 0.01. The portfolio allocations for the models that learned how to hold is presented in the figure below:

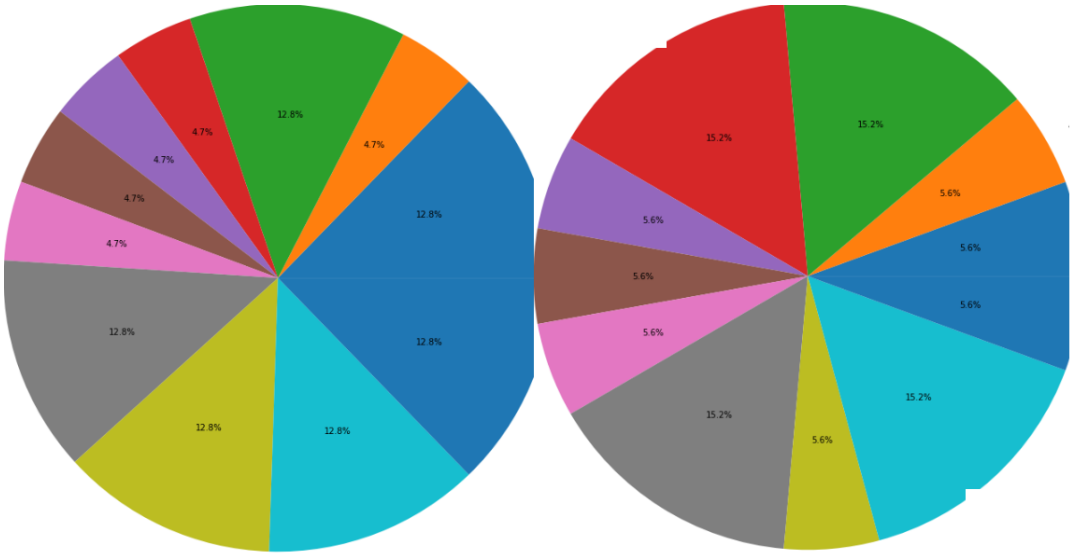


Figure 54 TD3 portfolios using R1 on a bull market

The portfolio with the learning rate of 0.01 seems to outperform the other because it has a slightly higher allocation on the high performance assets in test data namely BNB (green) and MATIC(grey).

4.2.6.2 Reward R2

TD3 with the objective of outperform BTC each timestep performed accordingly to the figure below:

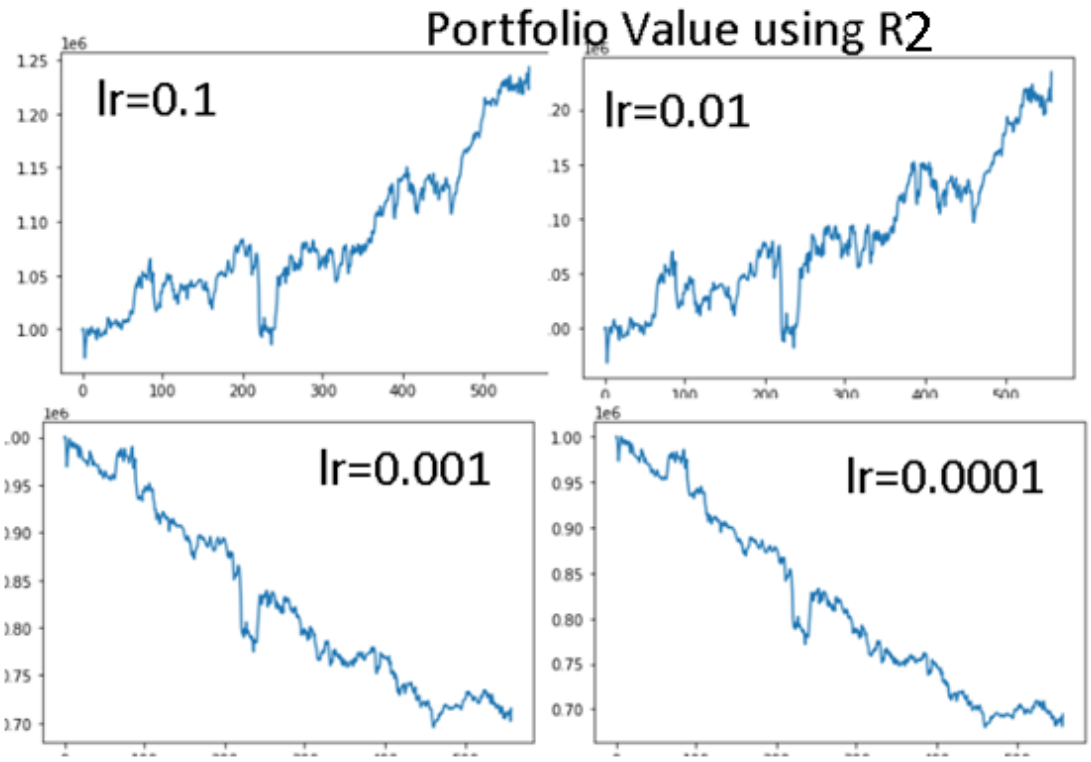


Figure 55 TD3 portfolio value using R2 on bull market

As we can observe from Figure 55 the only profitable strategies were produced with learning rates of 0.01 and 0.1. We can see the performance of each model on the table below:

Table 31 TD3 performance with R3 on bull data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	693922.93 (-30.60)	546	12
0.001	715152.99 (-28.48)	558	0
0.01	1234167.36 (+23.41)	1	557
0.1	1242945.40 (24.29)	1	557

We can conclude from this table that the models that didn't produce profitable strategies had a very high number of rebalances, and the models that learned how to hold produced better results than with the previous reward. The portfolio allocations are presented below:

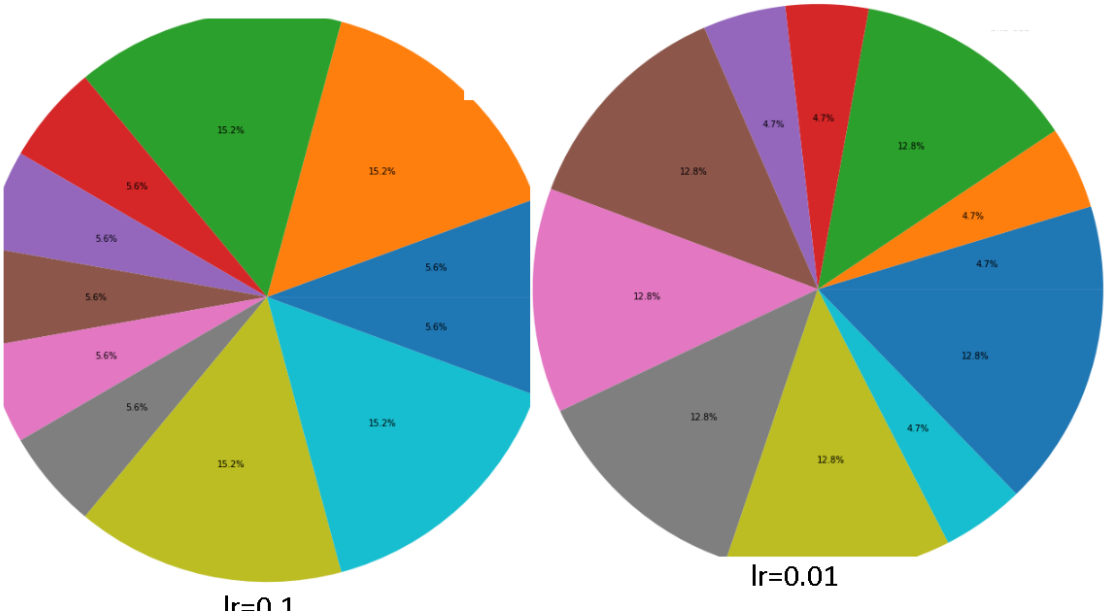


Figure 56 TD3 portfolios allocation using R2

4.2.6.3 Reward R3

TD3 trying to outperform BTC and USDC produced the following results:

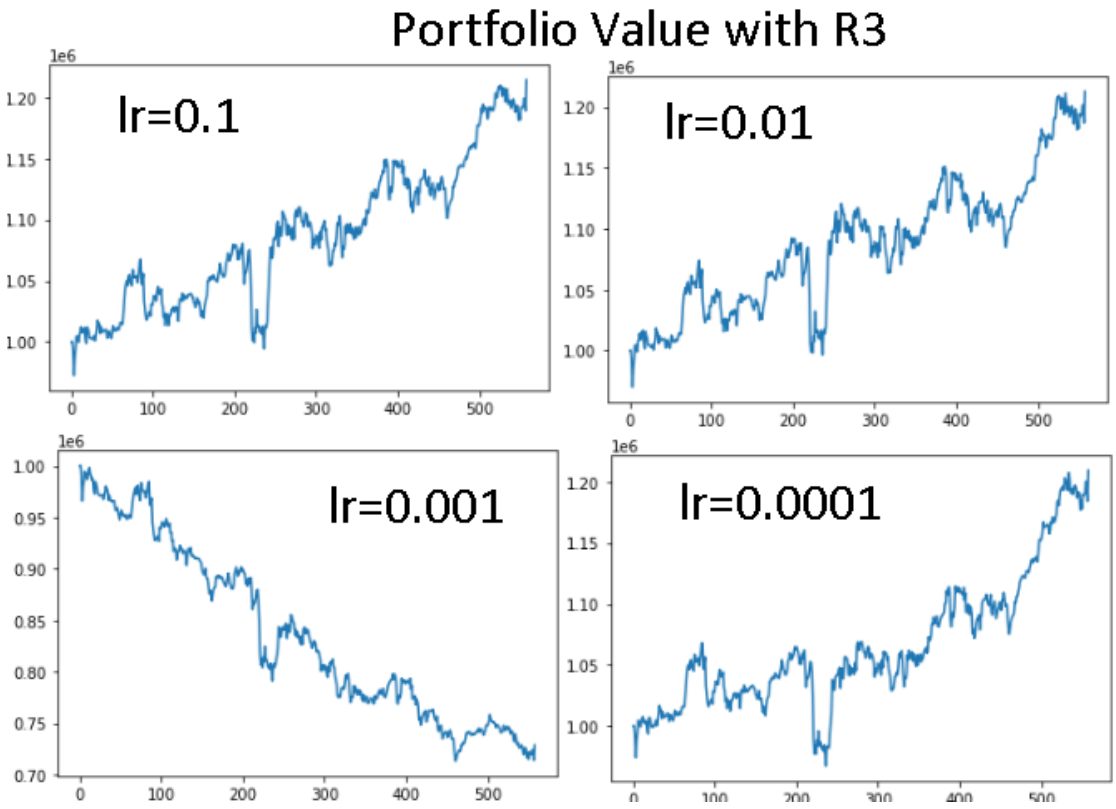


Figure 57 TD3 Portfolio value using R3 on a bull market

From Figure 57 only a learning rate of 0.001 didn't produce a profitable strategy on this type of market. The performance for each strategy is presented below:

Table 32 TD3 performance using R3 on bull data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	1209235.69 (+20.92)	1	557
0.001	728401.105 (-27.15)	558	0
0.01	1212651.19 (21.26)	1	557
0.1	1214208.55 (21.42)	1	557

From this table, one can conclude that the model that used a learning rate of 0.001 isn't profitable because of the high number of rebalances. None of the models was able to outperform equal-weighted portfolios and all models were very similar. The allocation for each profitable strategy is presented below.

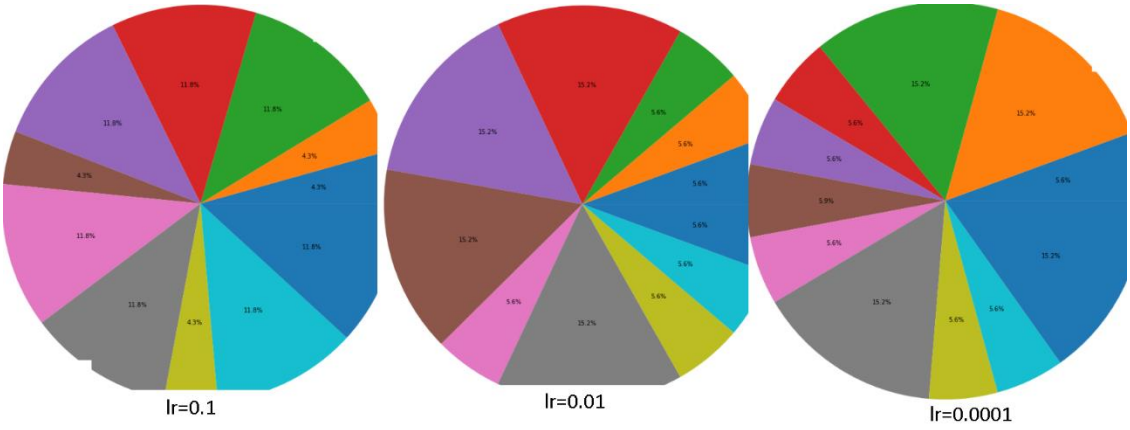


Figure 58 TD3 portfolio allocation using R3 on bull data

4.3 Bear Market

4.3.1 Data Visualization

In this section it is possible to see the price changes for all assets in the training and testing data on a bearish market. Bitcoin price evolution on training data is presented below:

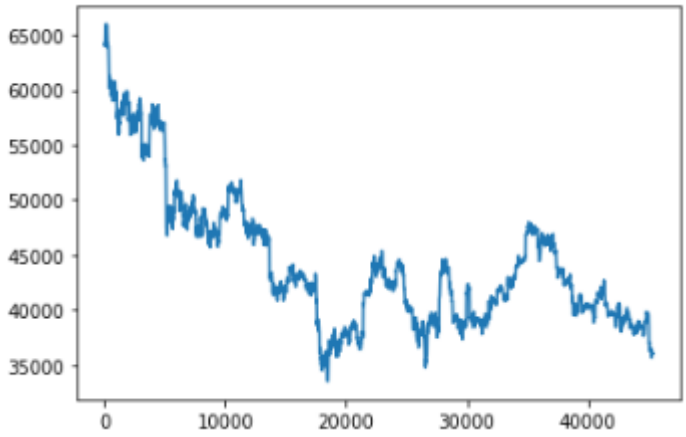


Figure 59 Bitcoin price evolution training data on a bear market

As we can see from this figure, the asset with the highest market cap falls almost 50% and it has a significant down trend. Usually bear markets are characterized by a 20% or more down fall. The price for each asset included is presented in the table below:

Table 33 training bear data percent change

Asset	Price at first timestep	Price at last timestep	% change
ADA-USD	2,028501	0,78	-61,54796078
AVAX-USD	93	57,31	-38,3763
BNB-USD	646,97	381,25	-41,0715
BTC-USD	64196,29	36008,78	-43,9083
DOGE-USD	0,26	0,12	-53,8462
ETC-USD	55,72	27,94	-49,8564
ETH-USD	4556,78	2697,81	-40,7957
MATIC-USD	1,7	1,05	-38,2353
TRX-USD	0,11	0,08	-27,2727

USDC-USD	1	1,0001	0,01
XRP-USD	1,18	0,59	-50

From Table 33 we can observe that bitcoin has dropped 43.90% and all of the other assets have a percent change of near -40% or more with the exception of TRX with -27,27%. USDC being a stable coin is clearly the most performative asset in a market where all the other assets are melting. In the figure below we can observe the bitcoin price for the testing data.

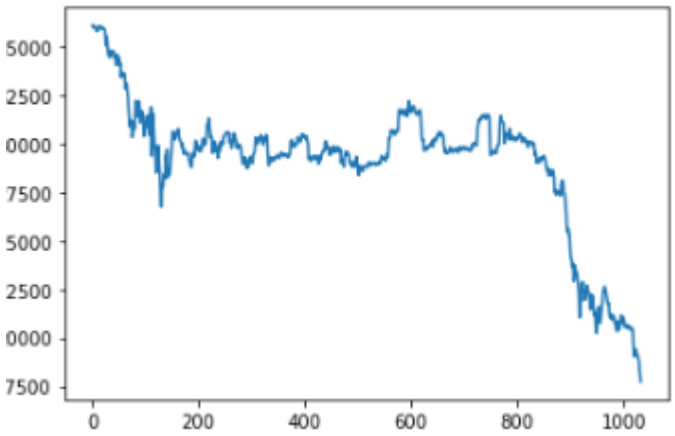


Figure 60 Bitcoin price evolution testing data on a bear market

In Figure 60 we can see that the down trend continues and Bitcoin seems to lose about 50% of its value during this period. In the table below we can see the percent change for each asset.

Table 34 Bear market asset percentage change on testing data

Asset	Price at first timestep	Price at last timestep	% change
ADA-USD	0,78	0,43	-44,8718
AVAX-USD	57,57	13,92	-75,8207
BNB-USD	381,34	186,02	-51,2194
BTC-USD	36098,34	17744,89	-50,8429
DOGE-USD	0,127	0,04	-68,5039
ETC-USD	27,98	12,75	-54,4317
ETH-USD	2702,38	896,57	-66,8229
MATIC-USD	1,05	0,32	-69,523

TRX-USD	0,08	0,05	-37,5
USDC-USD	0,99	1	1,010101
XRP-USD	0,6	0,29	-51,6667

From the table above we can see that the percent change is even more significant than in the training data. All assets are losing more than 50% of their value with exception of TRX and ADA with a return of -37,5% and -44,8 respectively. USDC once again, and as expect, was the most performative asset.

4.3.2 A2C

4.3.2.1 Reward R1

Advantage Actor Critic with the goal of maximizing profits each timestep obtained the following results:

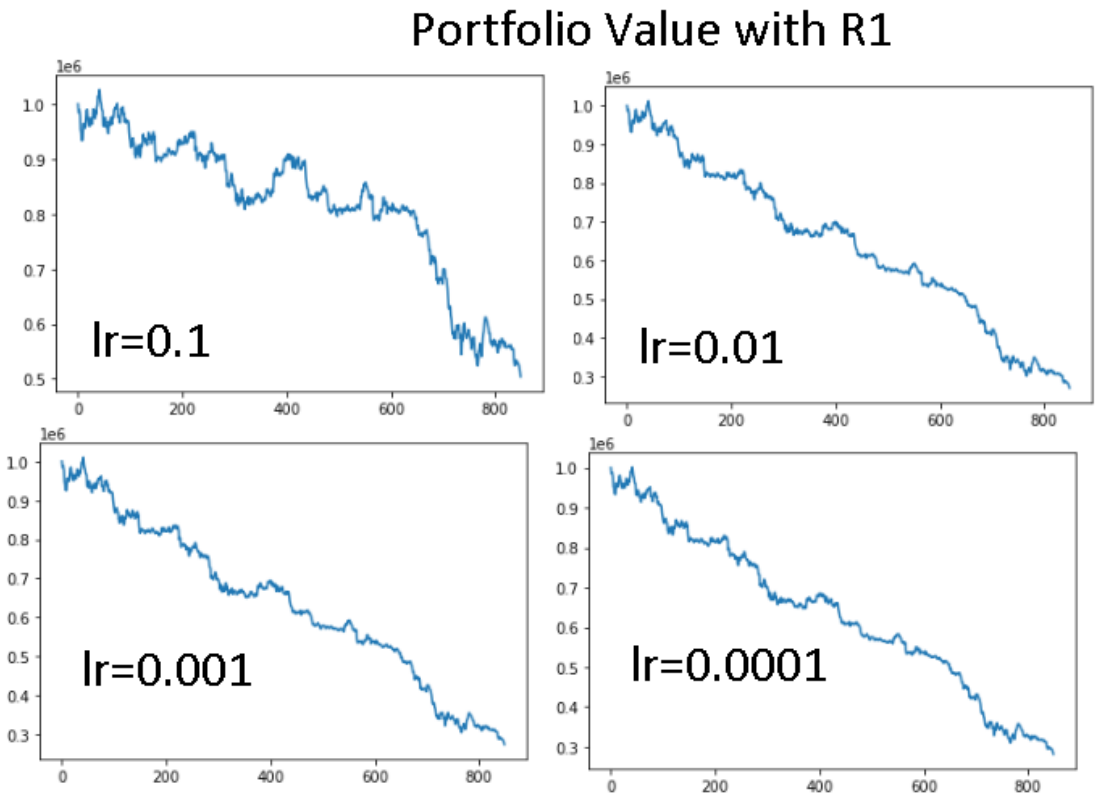


Figure 61 A2C using R1 on a bear market

From the figure above, we can see that all the models are losing nearly 70% of the initial portfolio value with the exception of a learning rate of 0.1. The performance for each model is presented in the table below:

Table 35 A2C performance using R1 on bear data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	281903.25 (-71.80)	849	0
0.001	273941.13 (-72.60)	798	51
0.01	269380.97 (-73.06)	824	25
0.1	504009.91 (-49.59)	255	594

It is possible to confirm that the most performative model was obtained with a learning rate of 0.1. Only the model with a learning rate of 0.0001 rebalanced the portfolio each timestep and the model with a learning rate of 0.1 was the only one with a high number of holds, but still has a high number of rebalances. This model was unable to outperform the best benchmark strategy for this type of market named equal-weighted portfolios.

4.3.2.2 Reward R2

When trying to outperform bitcoin A2C on a bearish market obtained the following results:

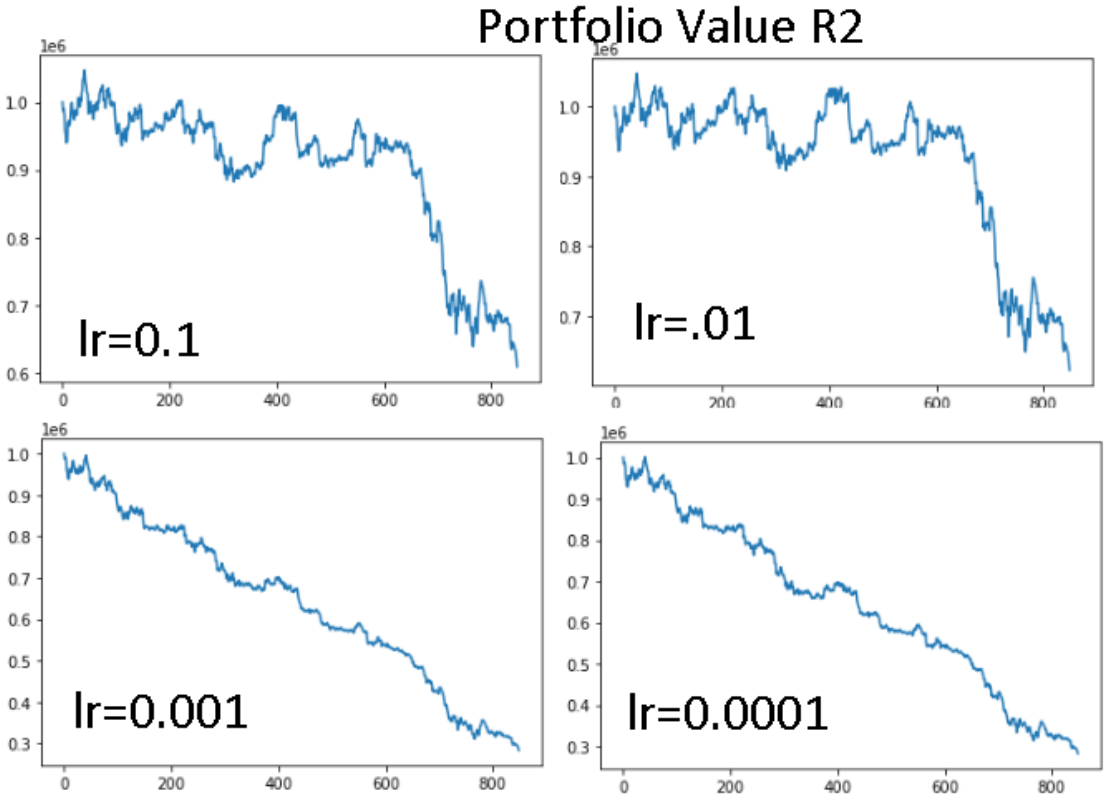


Figure 62 A2C potfolio value using R2 on a bear market

From the figure above it is possible to observe that the results seem better for the learning rates of 0.01 and 0.1 and look very similar in the lower learning rates. We can see the performance for each model below:

Table 36 A2C performance using R2 on a bear market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	282492.46 (-71.75)	849	0
0.001	283845.32 (-71.61)	846	3
0.01	623000.94 (-37.69)	1	848
0.1	608854.95 (-39.11)	57	792

From Table 36 we can see that the lower learning rates perform relatively similary comparing to R1 with a loss close to 72% while with higher learning rates the models seem to perform better. The learning rate of 0.01 strategy consisted of rebalancing the portfolio at the first timestep and holding until the last timestep and the model with a learning rate of 0.1 held a lot more times than using R1 and consequently kept about 10% of the portfolio value. The most performative model was produced with a learning rate of 0.01 and was able to outperform equal-weighted portfolios by a slight margin. The portfolio allocation for this model is presented below.

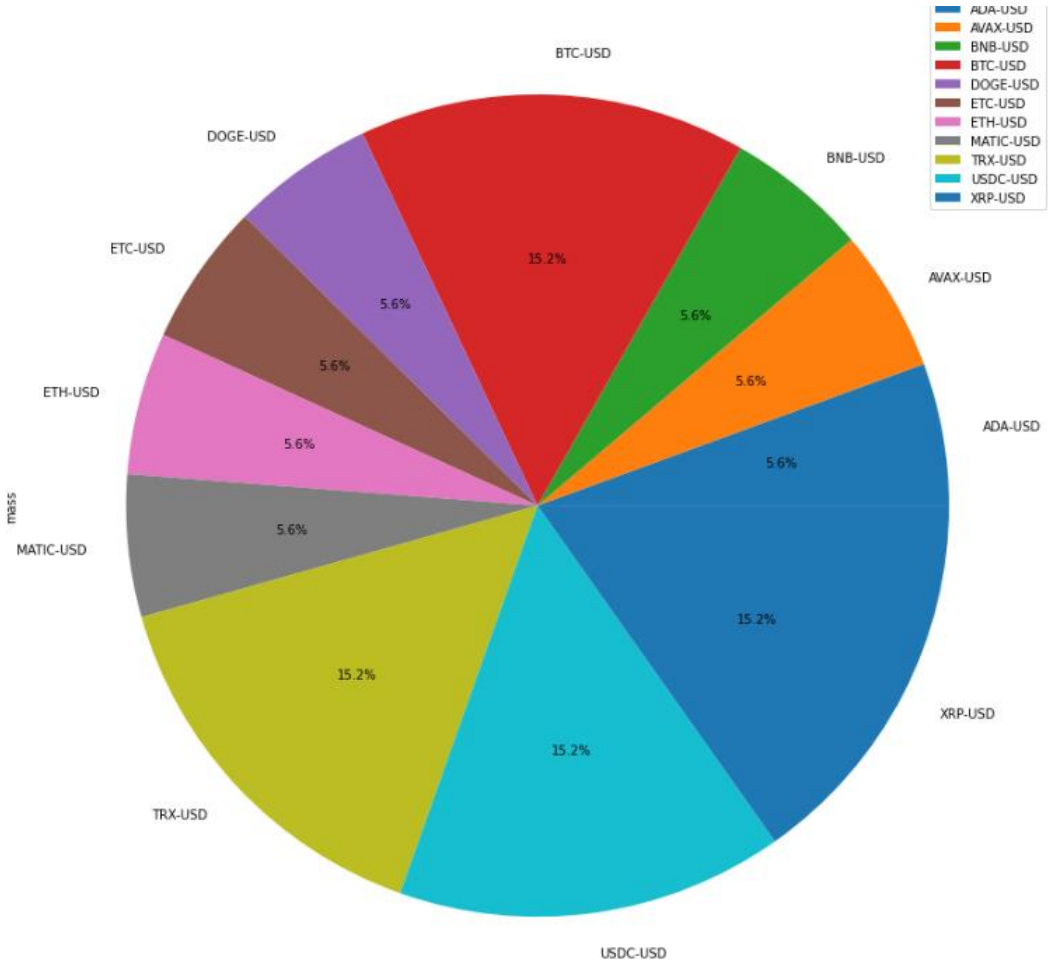


Figure 63 A2C using R2 on a bear data (lr =0.01)

From Figure 63 the learning rate of 0.01 seemed to fit well on the training data since it included the most performative assets USDC (light blue) and TRX (yellow) in the high allocation category and included the least performative ones in the low allocation category namely, ADA (blue) and DOGE(purple). This is still viable on the testing data because USDC and TRX kept being the most performative and DOGE one of the least performative.

4.3.2.3 Reward R3

A2C when trying to outperform both BTC and USDC at the same time produced the following portfolio values with different learning rates:

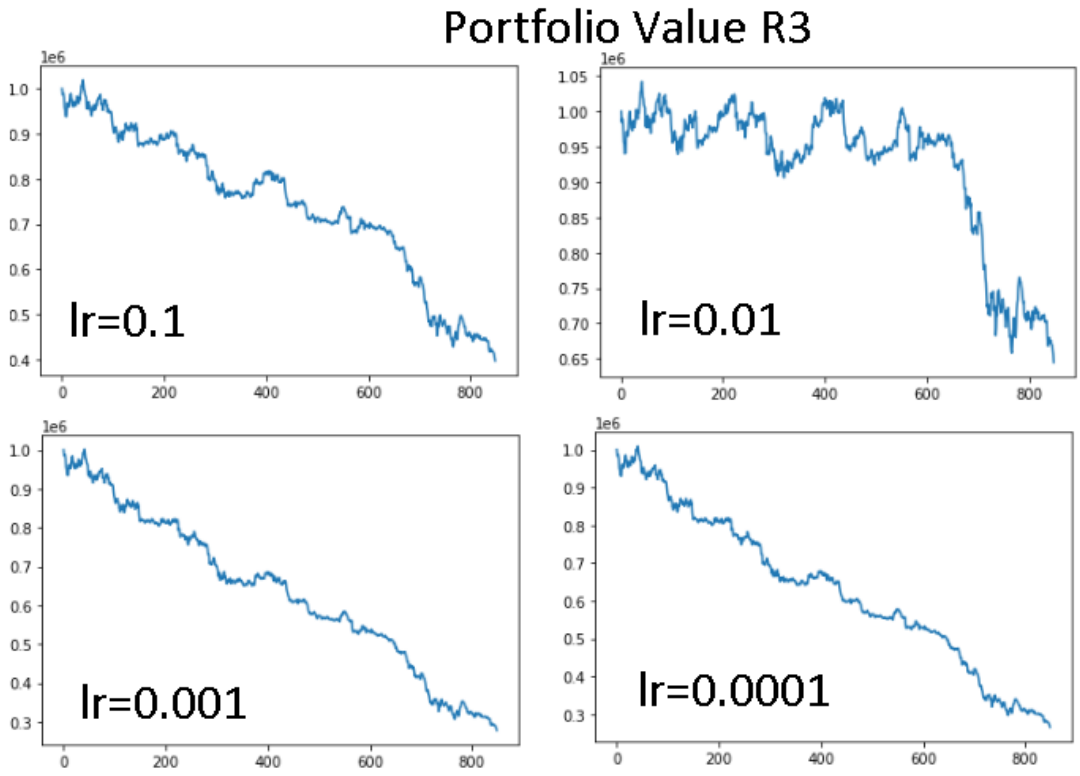


Figure 64 A2C using R3 on bear market

From the figure above we can see that the lower learning rate of 0.1 seems to perform worse than with other rewards while the lower learning rates are still unable to produce decent strategies. Only with a learning rate of 0.01 the results seem to be viable. The performance for each learning rate is presented in the table below:

Table 37 A2C performance using R3 on bear data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	265379.41 (-73.46)	849	0
0.001	277893.13 (-72.21)	847	2
0.01	644704.59 (-35.52)	1	848
0.1	396436.75 (-60.35)	501	348

We can observe from Table 37 that only the learning rate of 0.01 was able to consistently learn not to rebalance the portfolio a high number of times and produced the best model so far in this type of market with a return of -35.52%, more than a 2% improvement over A2C with R2 and a learning rate of 0.01. The allocation for this model is presented below:

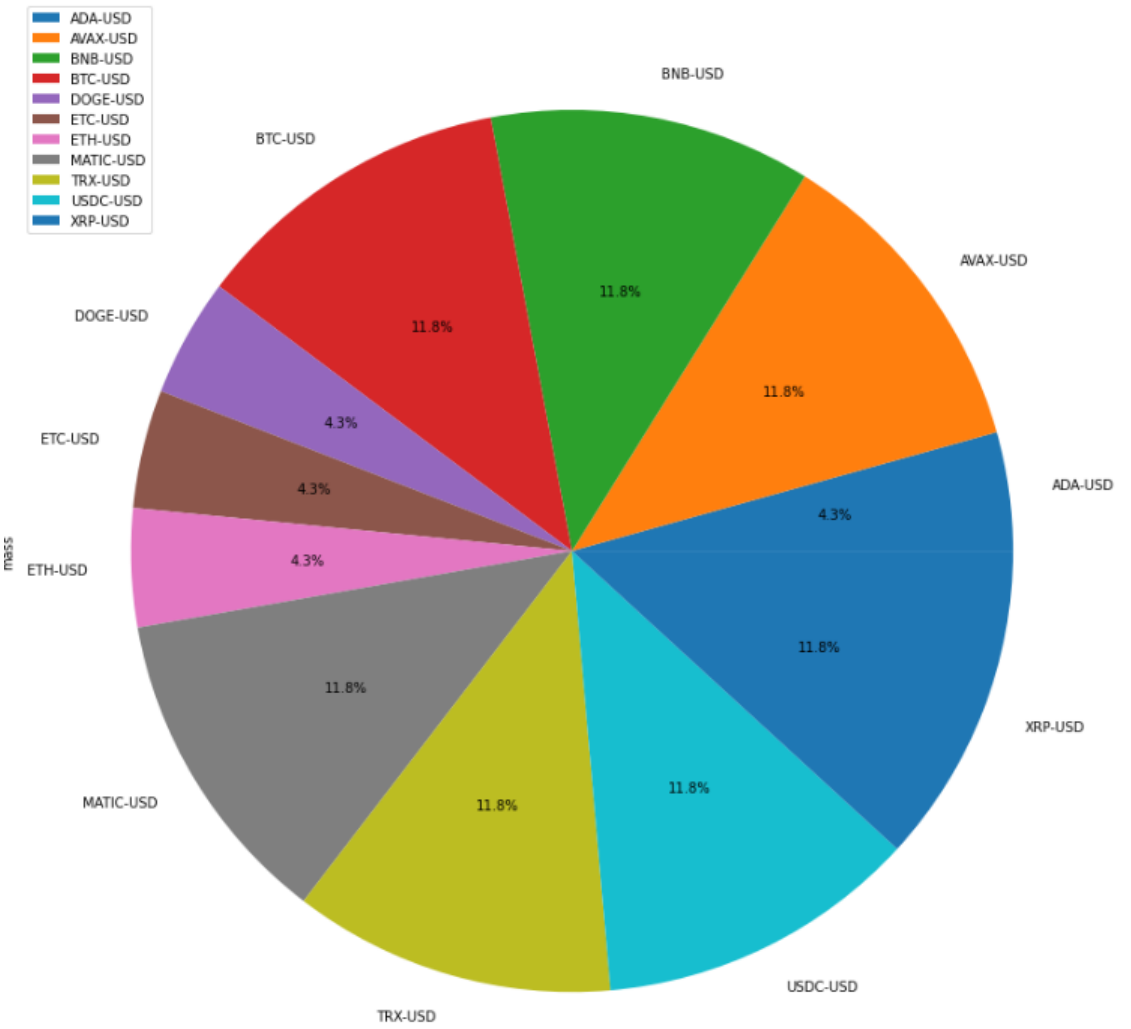


Figure 65 A2C portfolio allocation using R3 on bear data (lr =0.01)

This model has chosen a high allocation for the most performative assets in the training data namely USDC and TRX and kept a low allocation for DOGE and ADA which performed worse.

4.3.3 DDPG

4.3.3.1 Reward R1

Deep Deterministic Policy Gradient when maximizing profits obtained the following portfolio values:

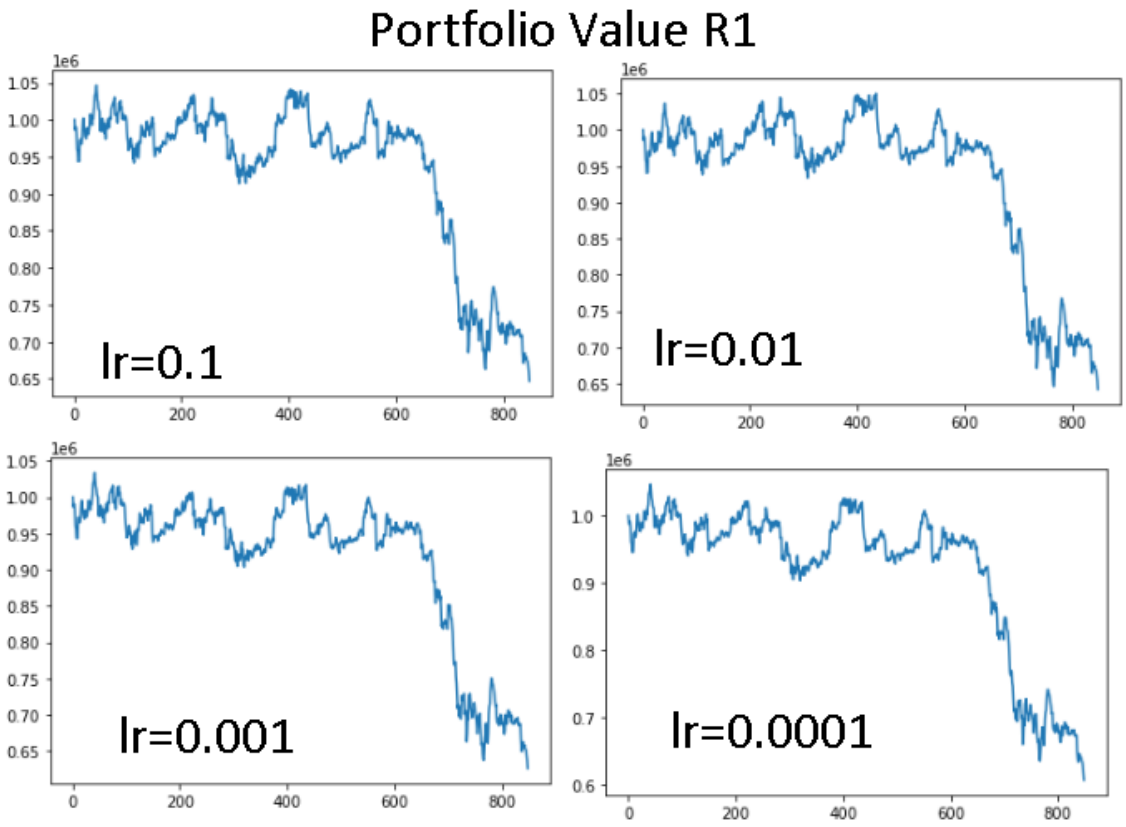


Figure 66 DDPG portfolio value using R1 on bear market

DDPG, similarly to a bull market condition, is able to produce very decent results across all the learning rates. The performance for each learning rate is presented below:

Table 38 DDPGG poerformance using R1 on bear market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	607173.17 (-39.28)	1	848
0.001	625948.58 (-37.40)	7	842
0.01	642088.25 (-35.79)	1	848
0.1	647077.07 (-35.29)	1	848

From Table 38 we can observe that every model learned that holding throughout the trading period produced excellent results. Only the model with a learning rate of 0.001 had more than 1 portfolio rebalance. The model with 0.1 beat the previous best model (A2C with R3) by a small margin of 0.23%. The portfolio allocation for each model that only rebalance 1 time is presented in the figure below:

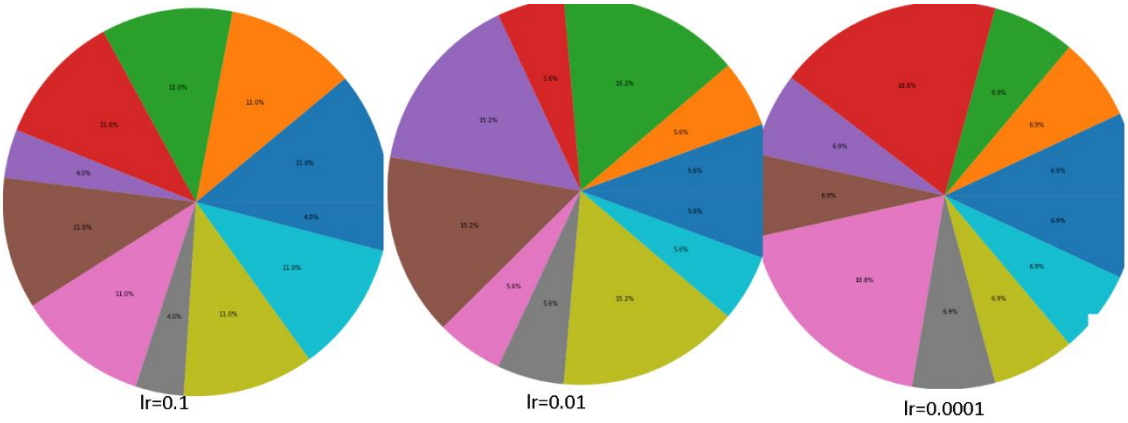


Figure 67 DDPG portfolio allocations using R1

From this figure we can see that only a learning rate of 0.1 USDC (light blue) was included in the high allocation category.

4.3.3.2 Reward R2

DDPG trying to outperform BTC obtained the following portfolio values during the trading period:

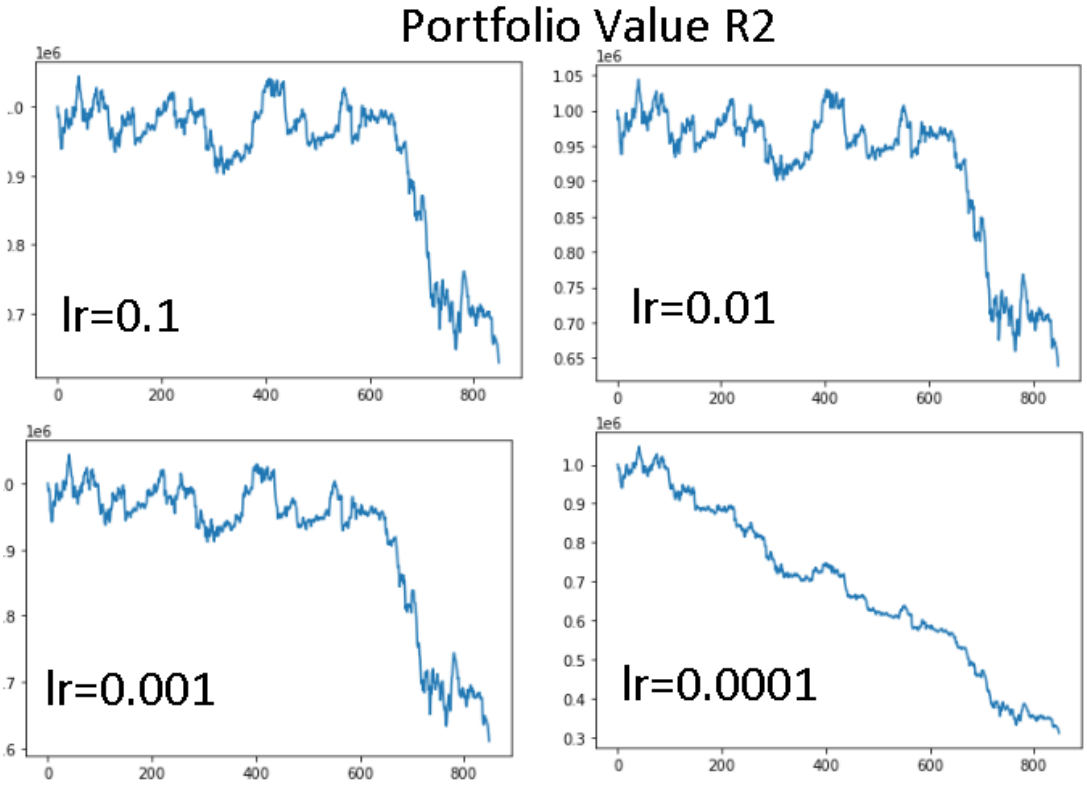


Figure 68 DDPG portfolios values using R2 on Bear data

From Figure 68 this algorithm with this reward produced good results except for a learning rate of 0.0001. The performance for each learning rate is detailed below:

Table 39 DDPG performance using R2 on bear data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	311303.87 (-68.86)	644	205
0.001	610940.31 (-38.90)	1	848
0.01	638687.55 (-36.13)	1	848
0.1	627839.40 (-37.21)	1	848

Table 39 confirms that once again the model that performs worse has a very high number of rebalances. All the other learning rates stick to the strategy of holding the rebalanced portfolio at the first timestep. The allocation for each learning rate is presented below:

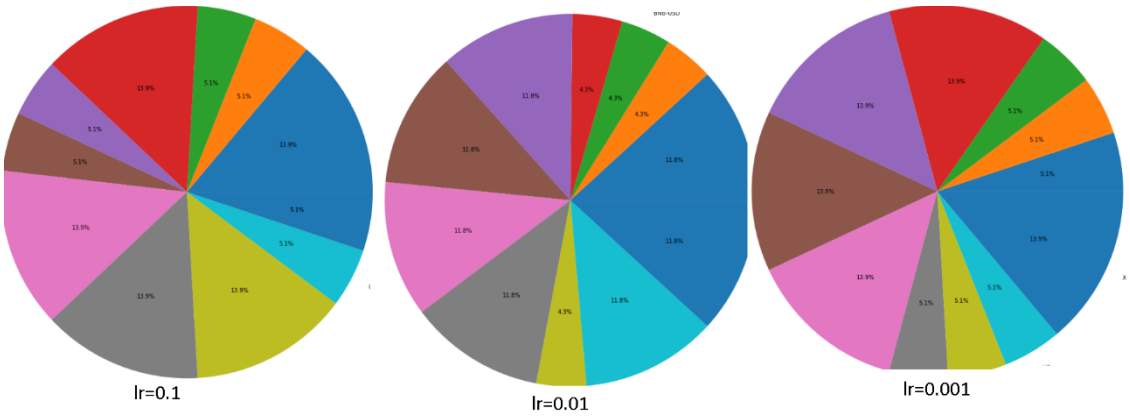


Figure 69 DDPG portfolio allocations using R2 on bear market

The model with a learning rate of 0.01 was the most performative and was also the only one who included the most performative asset on this type of market USDC (light blue) in the high allocation category.

4.3.3.3 Reward R3

When trying to outperform BTC and USDC, Deep Deterministic Policy Gradient obtained the following portfolio values:

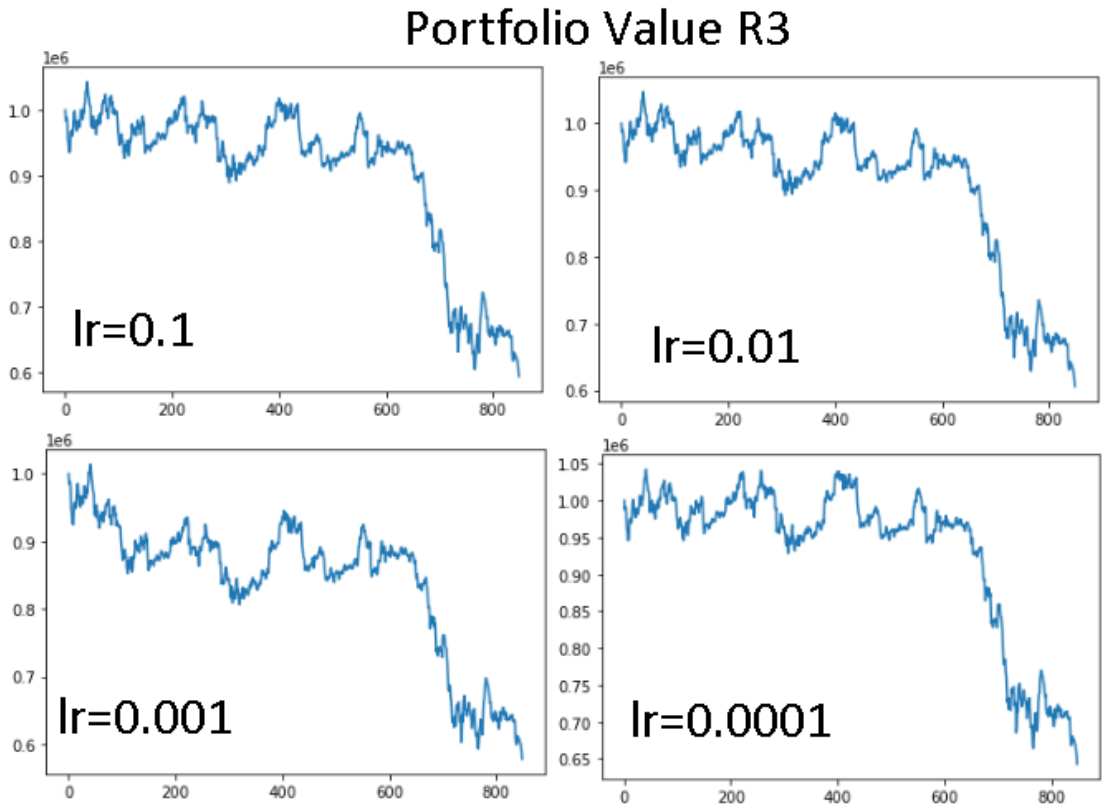


Figure 70 DDPG portfolio values using R3 on a bear market

From the figure above it looks like every model performed reasonably. The detained performance of each learning rate is presented in the table below:

Table 40 DDPG performance using R3 on bear market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	643037.75 (-35.69)	1	848
0.001	577838.91 (-42.21)	91	758
0.01	606065.28 (-39.39)	1	848
0.1	593519.10 (-40.64)	1	848

Table 40 shows that all models learn how to hold. Only with a learning rate of 0.001 this algorithm didn't hold throughout the whole trading period. Despite that, it has a very high number of holds but was the least performative model. Surprisingly with a learning rate of 0.0001 the results were best producing a return of -35.69% and very close to outperform DDPG with R1. The portfolio allocations are presented in the figure below:

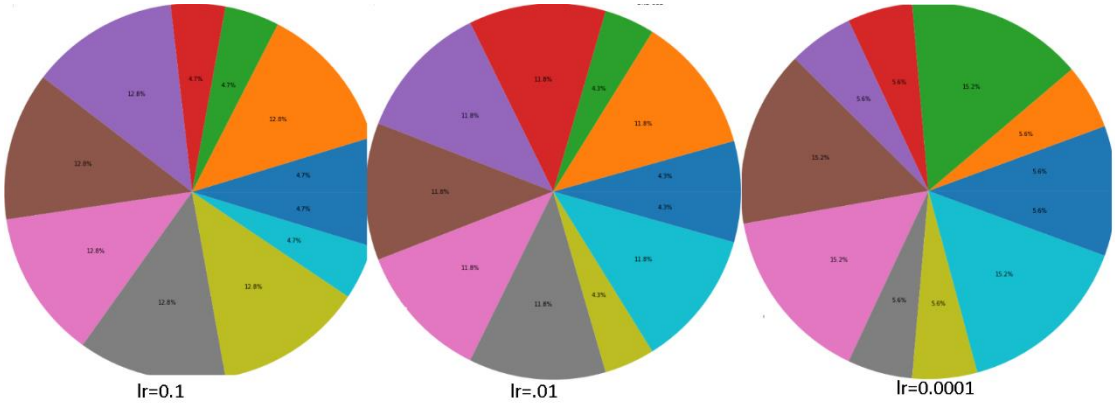


Figure 71 DDPG protfolio allocation using R3 on bear data

From this figure, we can see that the model with a learning rate of 0.0001 produced the best results because not only it attributed USDC (light blue) to the high allocation category, but it was also the one with the highest allocation of this asset when comparing to the other models.

4.3.4 PPO

4.3.4.1 Reward R1

Proximal policy optimization obtained the following portfolio values when trying to maximize profits:

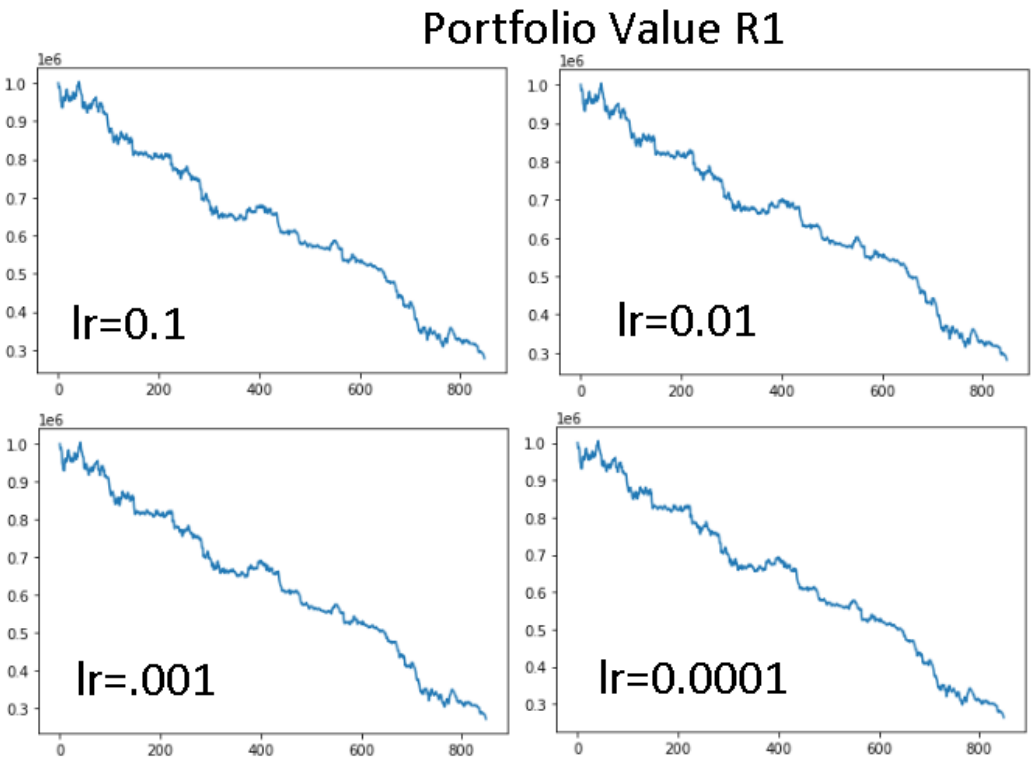


Figure 72 PPO protfolio value using R1 on bear market

Figure 72 shows that PPO using this reward produced very poor results across all the learning rates. The detained performance is presented in the table below:

Table 41 PPO performance using R1 on a bear market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	263621.43 (-73.63)	849	0
0.001	271518.68 (-72.84)	849	0
0.01	281564.38 (-71.84)	849	0
0.1	277787.39 (72.21)	849	0

From this table it is possible to see that all the models have very poor results and rebalanced the portfolio each timestep.

4.3.4.2 Reward R2

PPO when trying to outperform BTC obtained the following portfolio values.

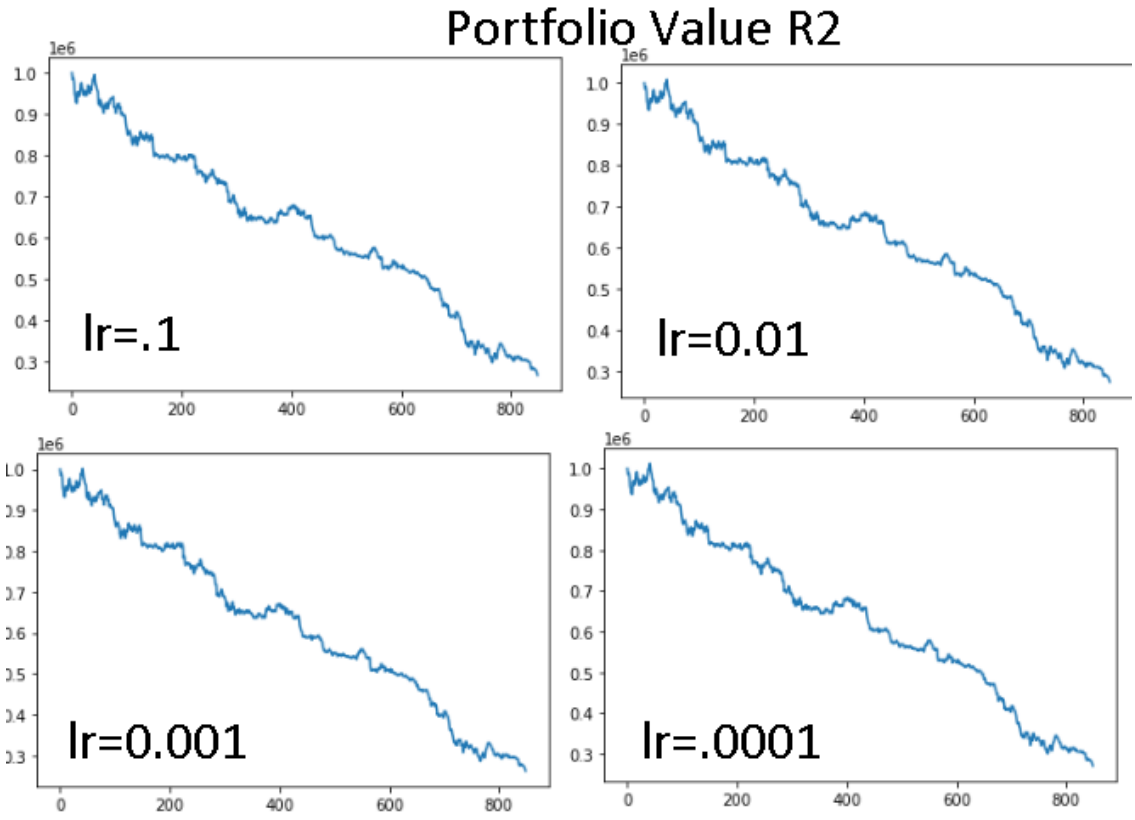


Figure 73 PPO portfolio value using R2 on a bear market

Similarly to R1, the results don't look good at all. The detained performance is presented below:

Table 42 PPO performance using R2 on bear market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	270541.370 (-72.94)	849	0
0.001	261876.28 (-73.81)	849	0
0.01	275283.23 (-72.47)	849	0
0.1	268684.35 (-73.13)	849	0

Table 42 shows that once again all models rebalanced the portfolios every timestep and consequently lost a great amount of the initial portfolio value.

4.3.4.3 Reward R3

PPO when trying to outperform USDC and BTC obtained the following portfolio values:

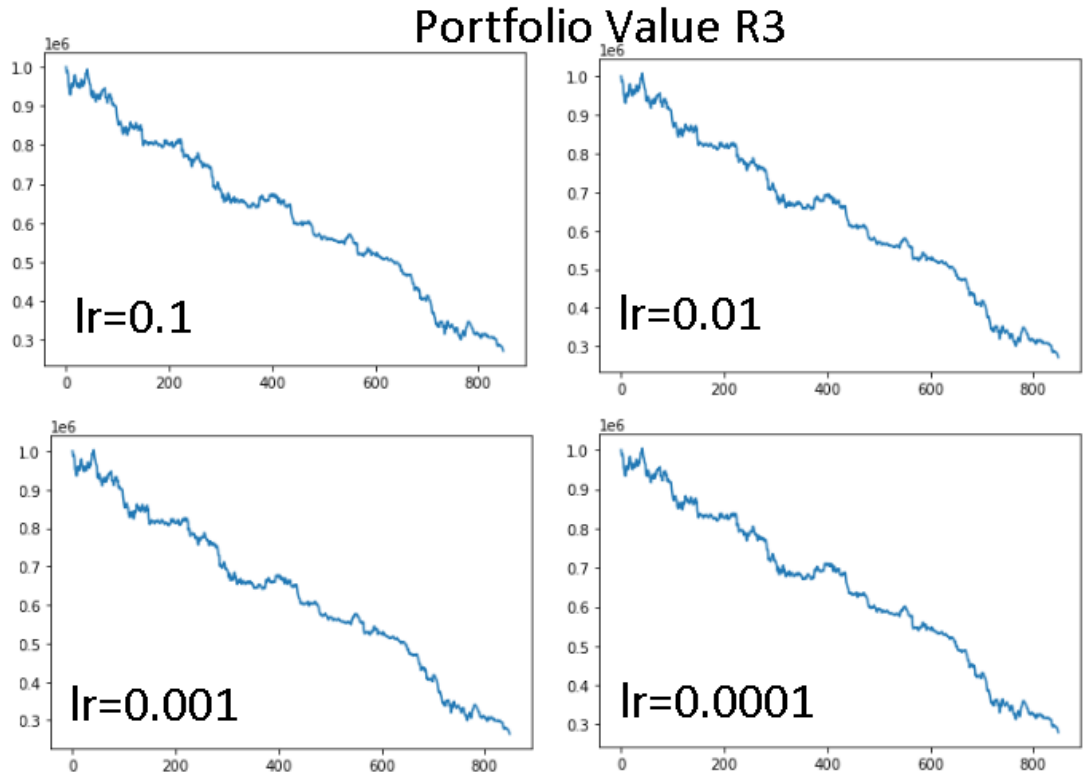


Figure 74 PPO portfolio values using R3 on a bear data

R3, similarly to R1 and R2, was unable to produce a decent strategy. The performance for each learning rate is presented in the table below:

Table 43 PPO performance using R3 on a bear market

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	280013.43 (-71.98)	849	0
0.001	264648.18 (-73.53)	849	0
0.01	271649.10 (-72.83)	849	0
0.1	270909.08 (-72.90)	849	0

Table 43 confirms that PPO has a difficult time learning how to hold, and just as in R1 and R2, R3 rebalanced the portfolio every timestep of the trading period.

4.3.5 SAC

4.3.5.1 Reward R1

Soft-actor-critic with the goal of maximize profits, has the following portfolio values:

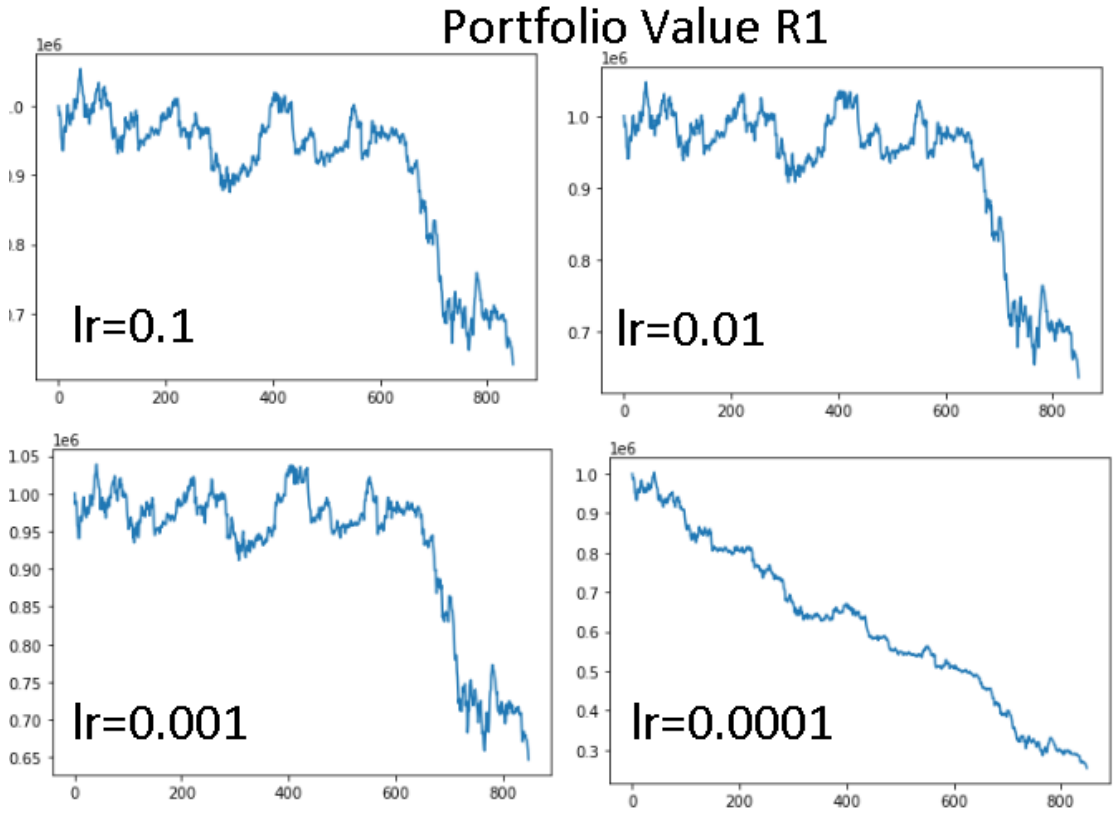


Figure 75 SAC portfolio values using R1 on bear data

From Figure 75 we can see that all models except the one with a learning rate of 0.0001 performed relatively similarly. The performance summary is presented below:

Table 44 SAC performance using R1 on bear data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	253527.48 (-74.64)	849	0
0.001	646899.16 (-35.31)	1	848
0.01	635601.34 (-36.43)	1	848
0.1	626260.55 (-37.37)	1	848

Similarly to other scenarios, the model that insists on rebalancing the portfolio every timestep loses a lot more of the initial portfolio values than those who have a high number of holds. The model which performed the best was the one with a learning rate of 0.001. The portfolio allocation is presented below:

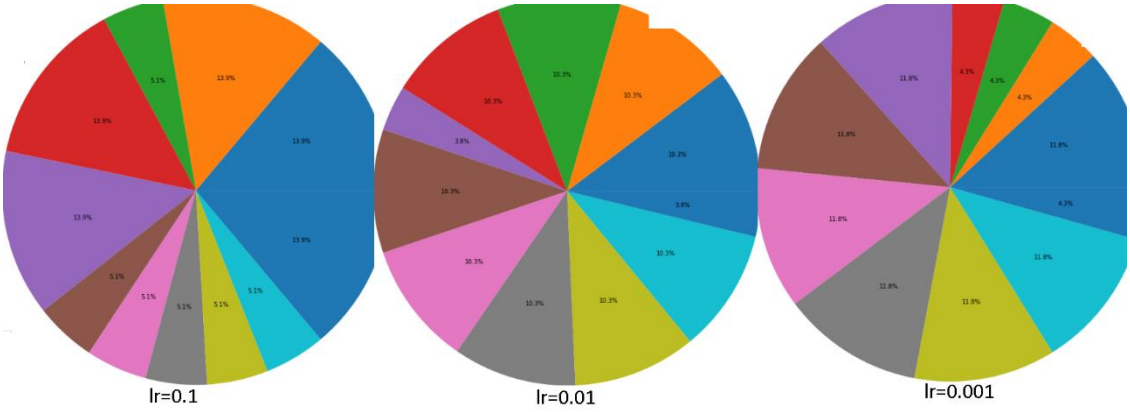


Figure 76 SAC portoflios using R1 on bear data

Figure 76 shows that the model with a learning rate of 0.001 performs slightly better compared to the others because it has the highest allocation for USDC (light blue).

4.3.5.2 Reward R2

SAC with a reward that has the goal of outperforming bitcoin performs as follows:

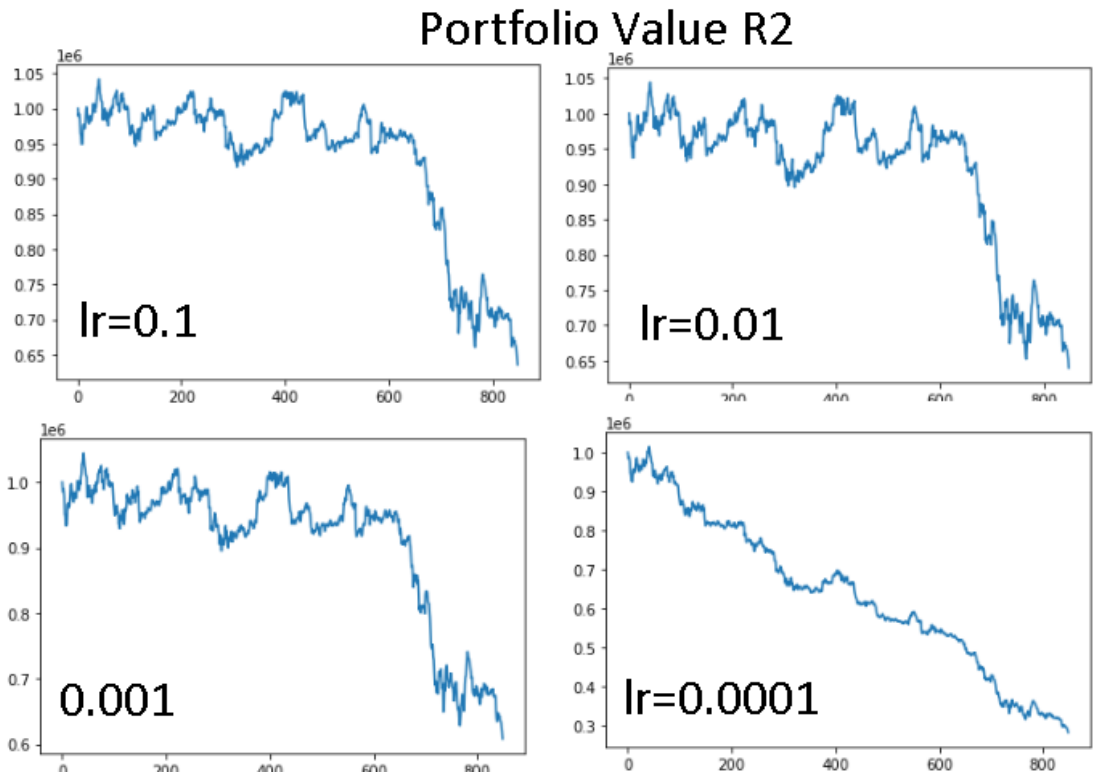


Figure 77 SAC portfolio values using R2 on bear data

The results look similar to the ones with R1, where all models seem to perform relatively well with the exception of the model with a learning rate of 0.0001 which loses about 50% more than other models. The summary for the performance of these models is presented below:

Table 45 SAC performance using R2 on bear data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	283579.26 (-71.64)	849	0
0.001	608245.96 (-39.17)	3	846
0.01	639691.76 (-36.03)	1	848
0.1	635591.96 (-36.44)	1	848

Table 45 shows that the model with the lowest learning rate rebalances the portfolio each trading instant. The model with a learning rate of 0.001 rebalances the portfolio two more times than the models with higher learning rates and has a lower performance than those models. The allocation for the most performative models is presented below.

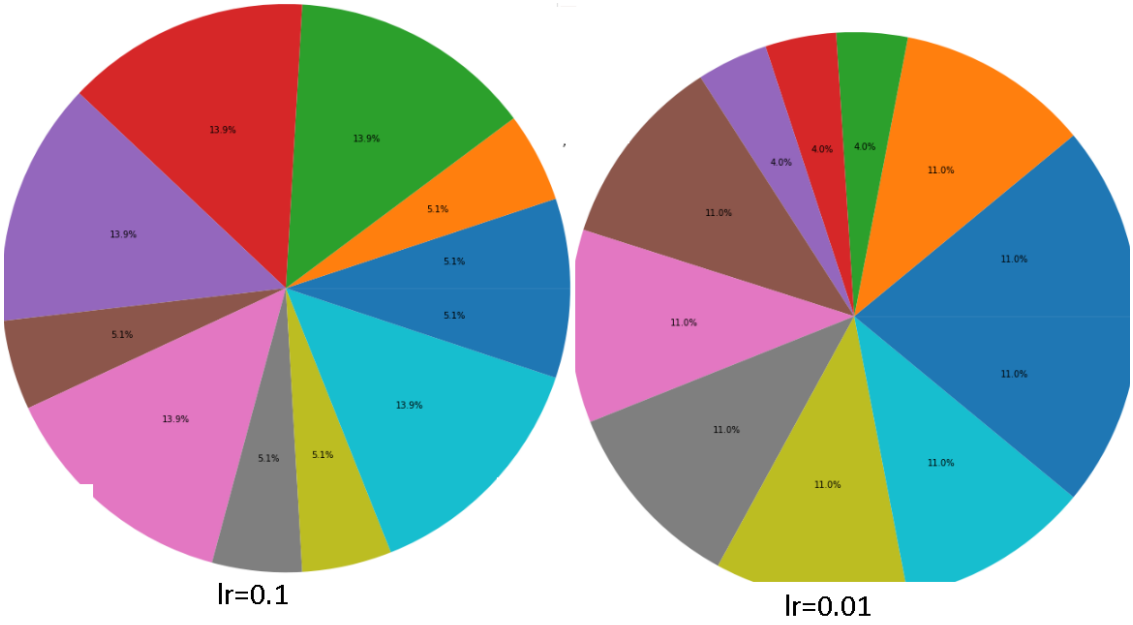


Figure 78 SAC portfolio allocations using R2 on bear data

From the figure and table above, we can see that models with learning rates of 0.01 and 0.1 perform very similary, but the one with 0.01 performs slightly better although the highest learning rate has a slightly higher allocation for USDC (light blue). The model on the right has a high allocation for TRX (yellow) and ADA(blue) which are the next most performative assets on test data after USDC while the one on the left has low allocation for those assets.

4.3.5.3 Reward R3

SAC when trying to outperform USDC and BTC has the following portfolio values:

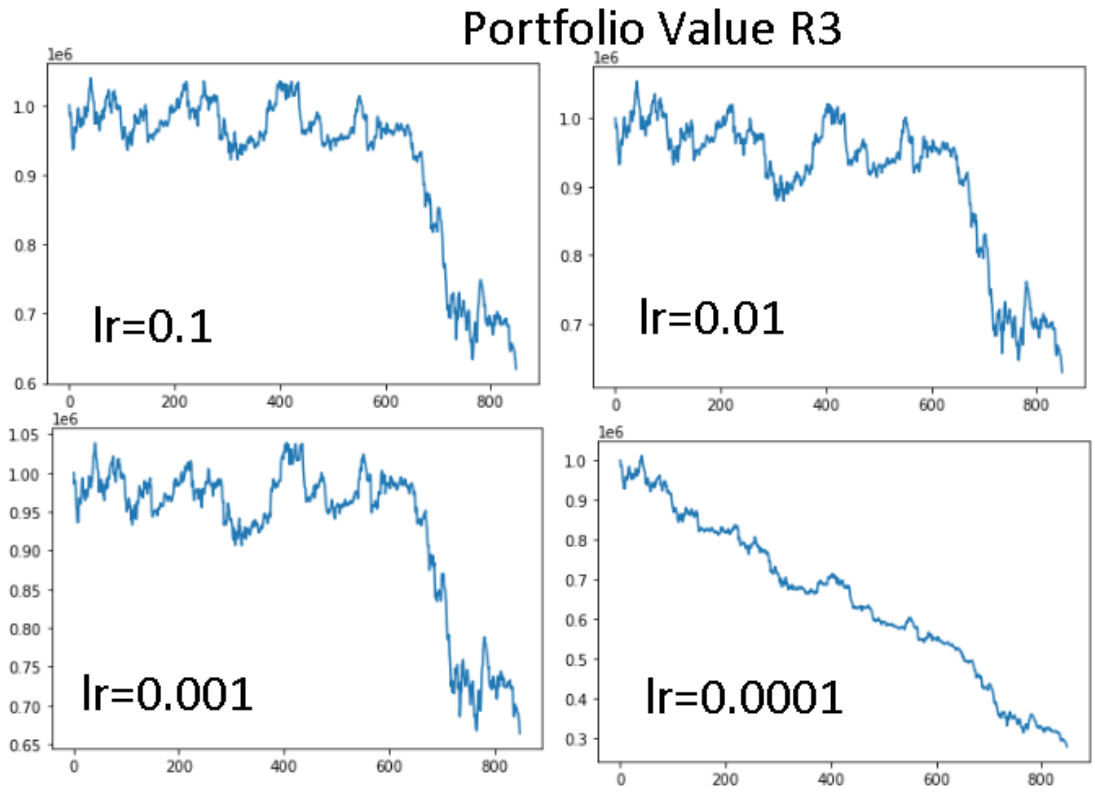


Figure 79 SAC portfolio values using R3 on bear data

Figure 79 shows that the results are very similar to R1 and R2, where only the model with the lowest learning rate performs way below the others. Summary for this scenario is presented below

Table 46 SAC performance using R3 on bear data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	277291.58 (-72.27)	849	0
0.001	663721.77 (-33.62)	1	848
0.01	628793.52 (-37.12)	1	848
0.1	620614.19 (-37.93)	1	848

From the table above we can observe that the model with a learning of 0.001 outperforms DDPG with R1 which was the previous best model. It outperformed DDPG-R1 and lost 1,67% less of the initial portfolio value and when comparing with the best benchmark strategy it lost about 4,41% of the initial value. The allocation for the models that performed well are presented in the figure below:

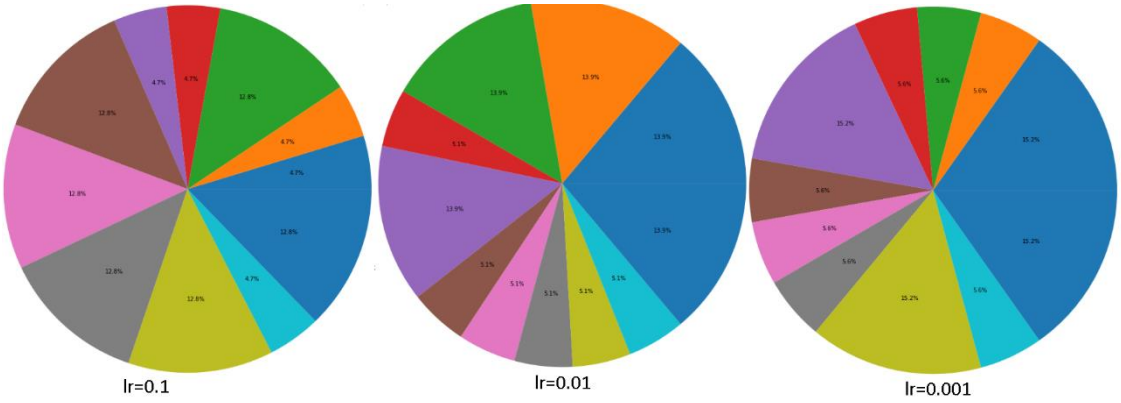


Figure 80 SAC portfolio allocation with R3 on bear data

As we can see from Figure 80 all models surprisingly have a low allocation for USDC(light blue). The best model (lr=0.001) probably outperformed the others out of luck. It is the one with the highest allocation for TRX(yellow) which is coherent with the training data but it also has a high allocation of ADA (blue) which is one of the most performative assets on test data, but one of the worst on training data.

4.3.6 TD3

4.3.6.1 Reward R1

Twin delayed DDPG when trying to maximize profit has the following portfolio values:

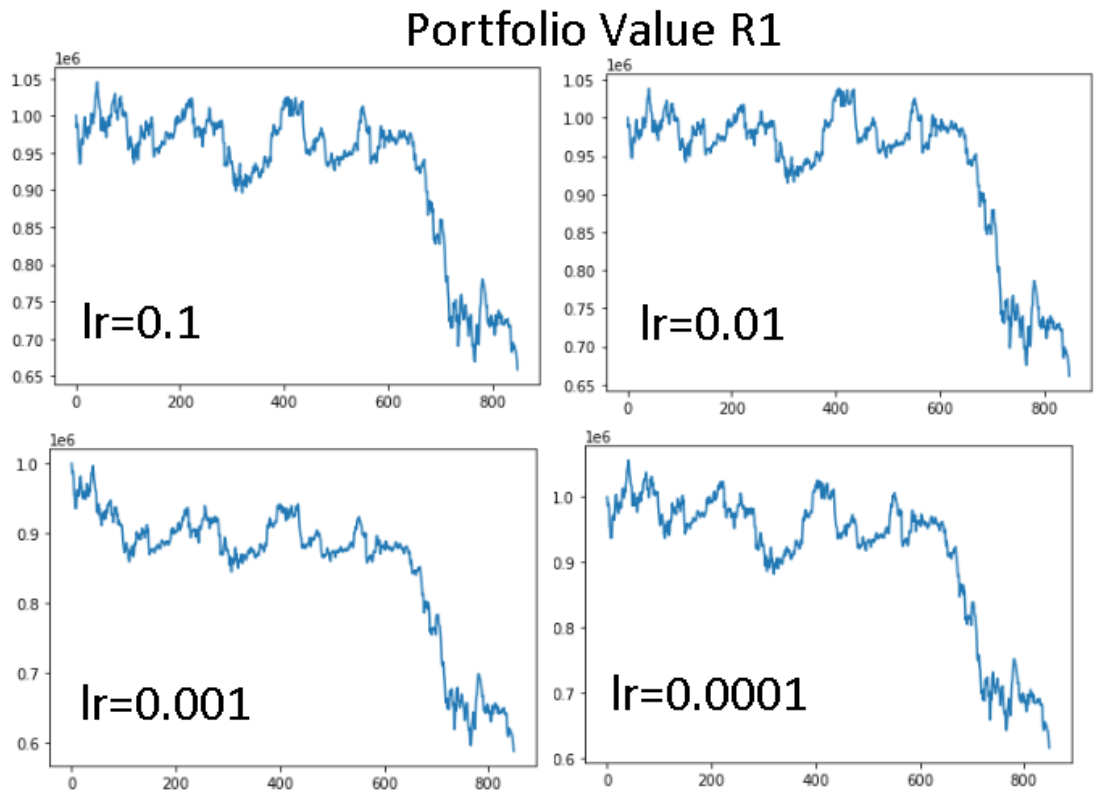


Figure 81 TD3 portfolio values using R1 on bear data

From the figure above it seems all models perform accordingly to the best ones so far. The detained performance is presented in the table below:

Table 47 TD3 performance using R1 on bear data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	616646.22 (-38.33)	1	848
0.001	587968.63 (-41.20)	747	102
0.01	661142.85 (-33.88)	1	848
0.1	658552.99 (-34.14)	1	848

Table 47 shows that the model with 0.001 has a very high number of rebalances despite of having a decent performance. It still is the least performative model when comparing to the others on the table but, when comparing with models from other algorithms with similar number of rebalances, it has a much lower loss. The model with a learning rate of 0.01 has a performance very similar to the best model so far, SAC-R3. The allocation for the models that hold is presented below:

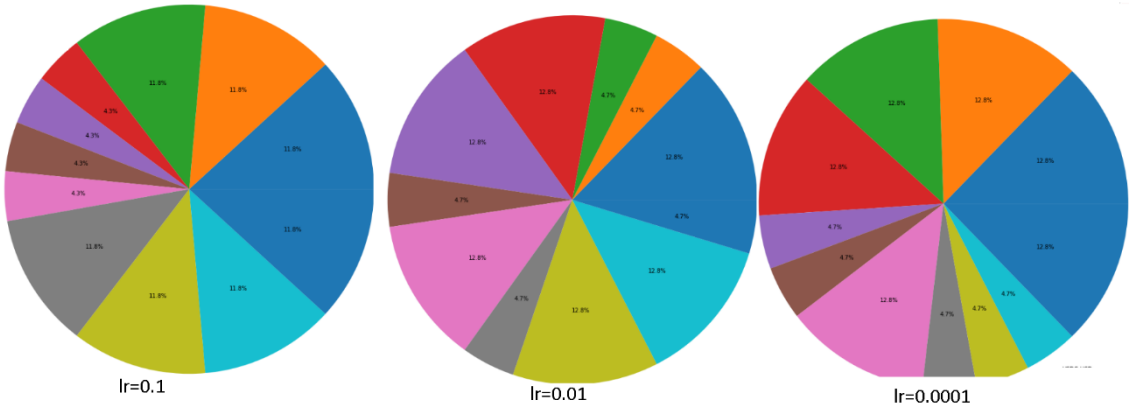


Figure 82 TD3 Portfolio allocations using R1 on bear data

From Figure 82 it is observable that the model with a learning rate of 0.01 which was the best one has the highest allocation for the two best performative assets USDC(light blue) and TRX (yellow).

4.3.6.2 Reward R2

TD3 with a goal of outperforming BTC has the following portfolio values:

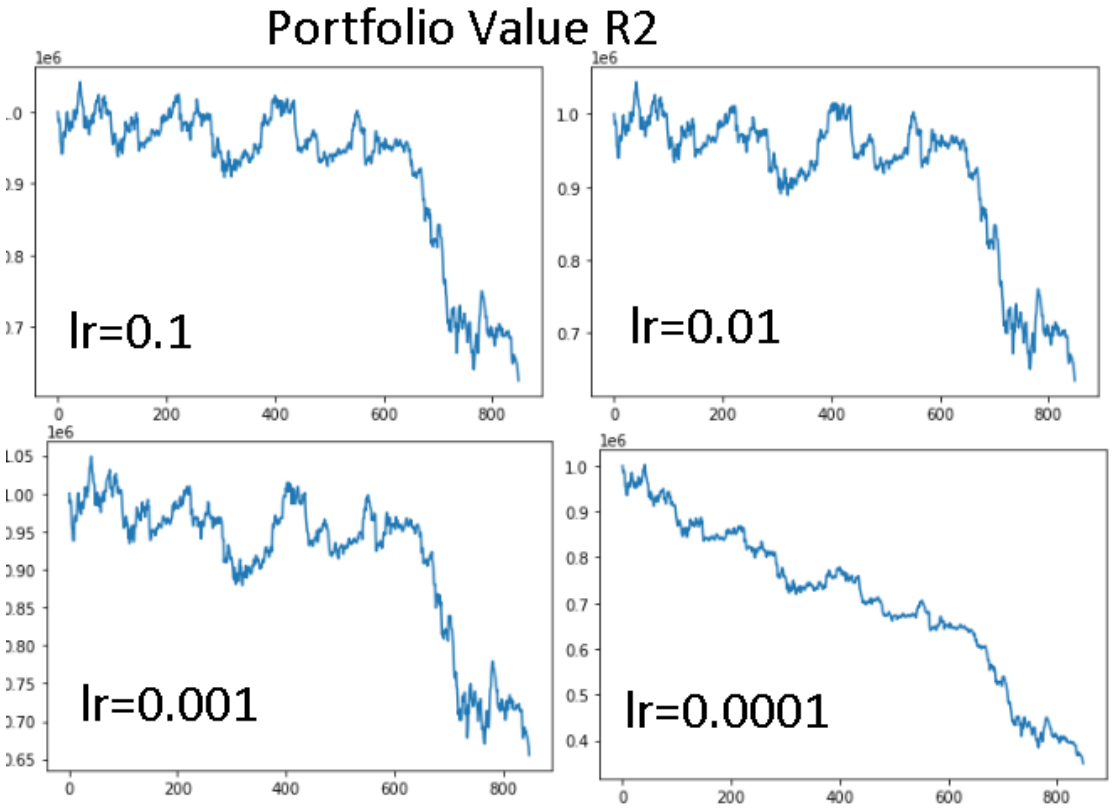


Figure 83 TD3 portfolio values using R2 on bear data

The figure above shows that the three highest learning rates seem to perform in the norms while the lowest learning rate has a very high loss. Below is presented the summary:

Table 48 TD3 perfomance using R2 on bear data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	349247.80 (-65.07)	548	301
0.001	655452.27 (-34.45)	1	848
0.01	634579.68 (-36.54)	1	848
0.1	624425.73 (-37.55)	1	848

From this table we can conclude that the lowest learning rate didn't perform as the other models due the high number of portfolio rebalances. The allocation for the other models is presented in the figure below:

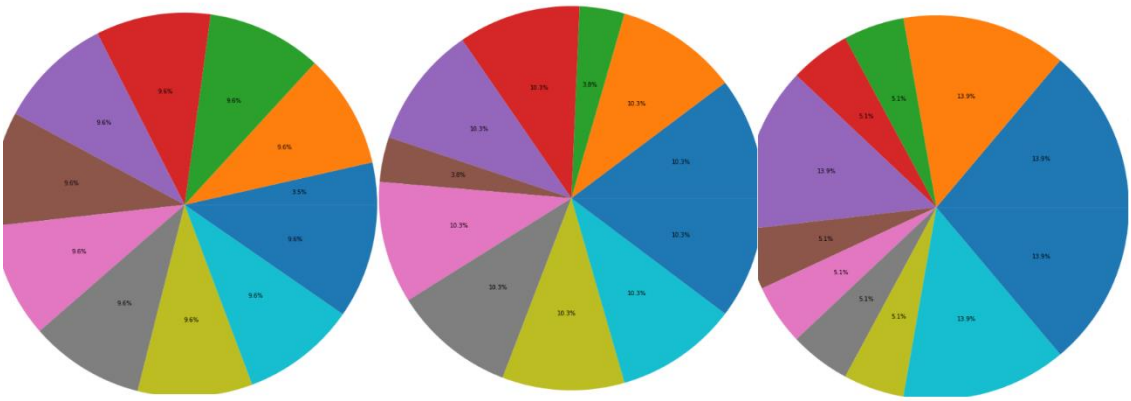


Figure 84 TD3 portfolio allocation using R2 on bear data(lr=0.1,lr=0.01,lr=0.001)

As we can see from Figure 84 the model with a learning rate of 0.001 has the highest performance and the highest allocation for the most performative asset USDC (light blue).

4.3.6.3 Reward R3

TD3 with the objective of outperforming BTC and USDC at the same time has the following performance across four different learning rates:

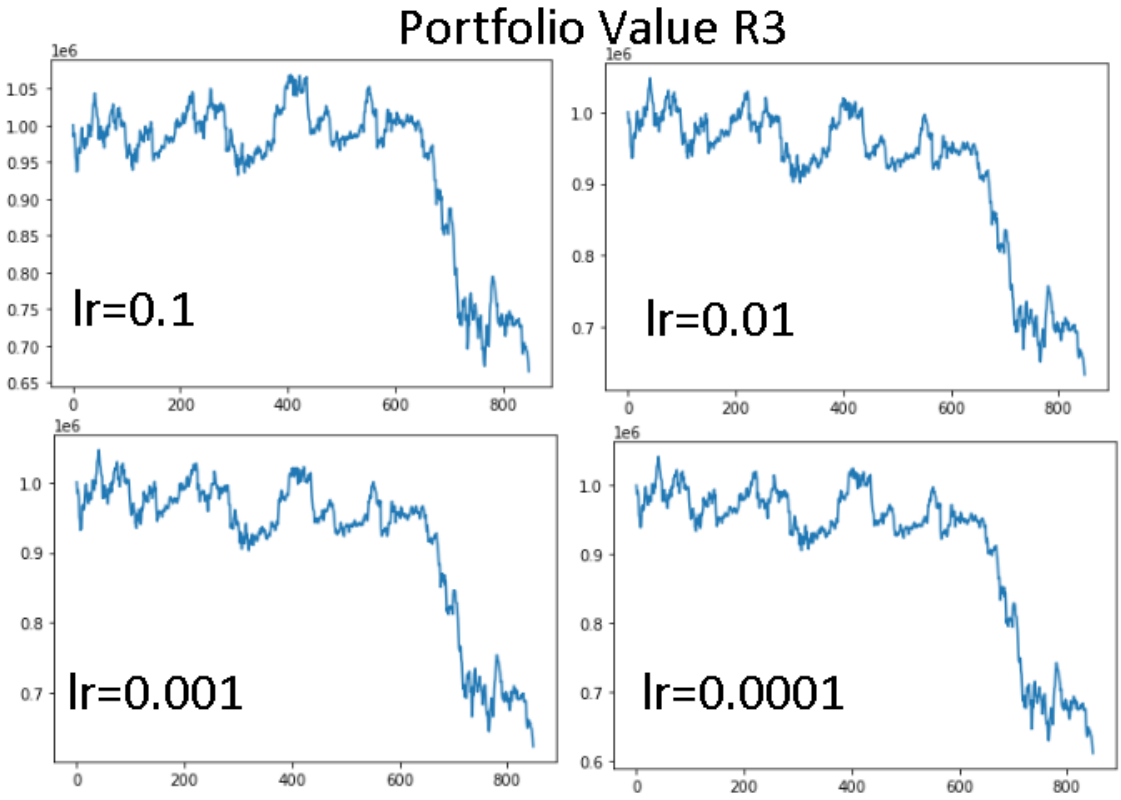


Figure 85 TD3 portfolio values using R3 on bear data

From Figure 85, it is observable that all models perform relatively similarly to one another. The performance summary for these models is presented in the table below.

Table 49 TD3 performace using R3 on bear data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	610432.61 (-38.95)	1	848
0.001	623707.00 (-37.62)	1	848
0.01	632679.08 (-37.63)	1	848
0.1	665295.69 (-33.47)	1	848

From this table we can see that all models rebalance at the first timestep and hold that allocation until the end of the trading period. The allocation for these models is presented in the figure below:

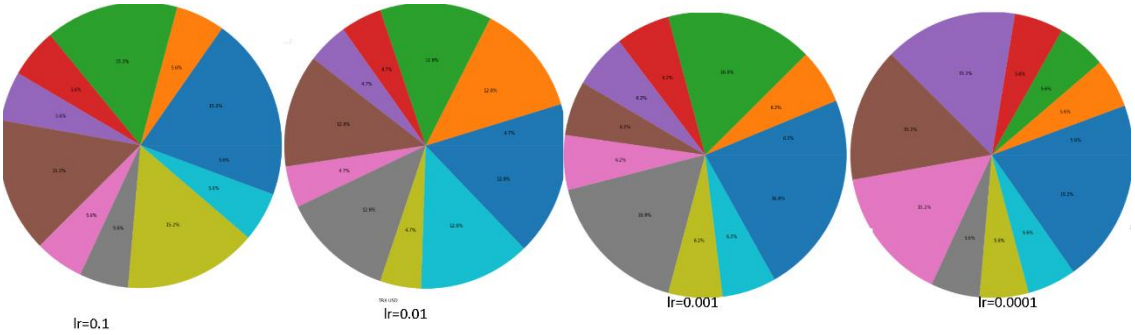


Figure 86 TD3 allocations using R3 on bear data

From this figure, we can see that the most performative model has the lowest allocation for the highest performative asset USDC (light blue), but it also has its highest allocation for the following two best performative assets namely TRX (yellow) and ADA (blue) while other models have low allocations for these assets. This model was able to outperform SAC-R3 which had a return of -33.62% while TD3-R3 has a return of -33.47.

4.4 Sideways Market

4.4.1 Data Visualization

In this section it is possible to see the price changes for all assets in the training and testing data on a sideways/lateralization market. Bitcoin price evolution on training data is presented below:

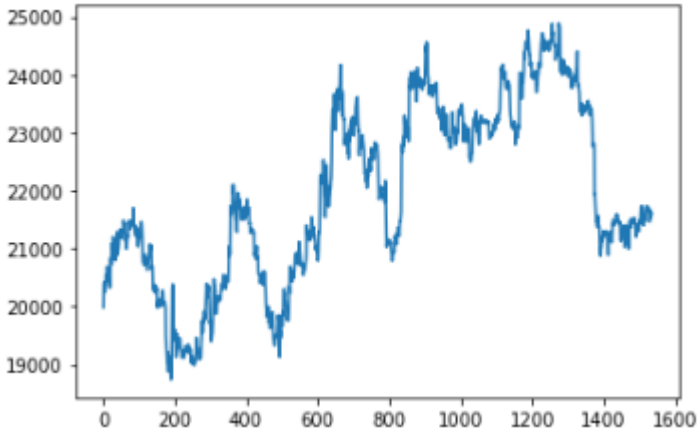


Figure 87 Bitcoin price evolution on training data for sideways data

As we can see from Figure 87 the price goes up and down but it doesn't show a massive upward trend or a lower one. This type of market keeps the price between a relatively small range of values. The price change for each asset is presented in the table below:

Table 50 Training data price change on sideways market

Asset	Price at first timestep	Price at last timestep	% change
ADA-USD	0,458723	0,464303	1,21642
AVAX-USD	16,08681	22,97812	42,83824
BNB-USD	214,5179	301,63	40,60828
BTC-USD	19987,61	21596,9	8,051441
DOGE-USD	0,061721	0,069	11,79339
ETC-USD	15,22751	37,1	143,638
ETH-USD	1051,499	1697,01	61,38961
MATIC-USD	0,45858	0,81	76,63221
TRX-USD	0,063158	0,0654	3,549827
USDC-USD	1	0,9999	-0,01
XRP-USD	0,322	0,348	8,074534

From Table 50 we can see that we have some very bullish assets while bitcoin only grows about 8%. The most performative assets are MATIC with an increase of 76.63, followed by ETH with a growth of 61,38%. The least performative assets were USDC with -0.01% followed by ADA with 1.21%. The bitcoin price for the test data is presented below:

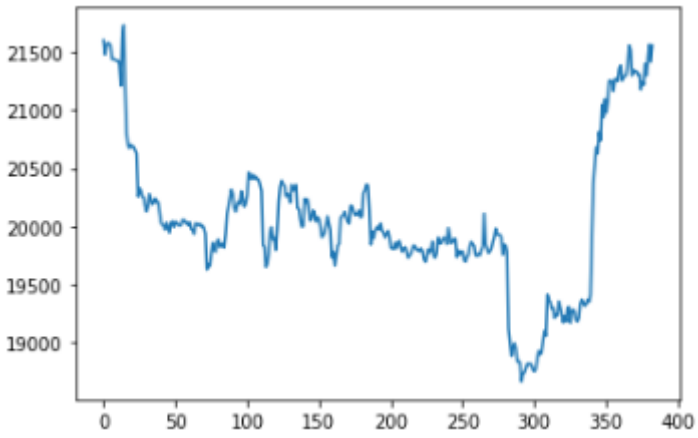


Figure 88 Bitcoin price on test data in a sideways market

Similarly to the price in the training data, BTC in the test data keeps its price between a relatively small range of value. The individual asset performance is presented in the table below:

Table 51 Asset percent change for testing data on sideways data

Asset	Price at first timestep	Price at last timestep	% change
ADA-USD	0,465028	0,469865	1,040152
AVAX-USD	23,02	19,94766	-13,3464
BNB-USD	301,6633	242,082	-19,7509
BTC-USD	19987,61	21809,52	9,115217
DOGE-USD	0,069096	0,069129	0,04776
ETC-USD	37,137	15,67489	-57,7917
ETH-USD	1696,6	1235,38	-27,185
MATIC-USD	0,818	0,605	-26,0391
TRX-USD	0,065	0,069	6,153846
USDC-USD	0,9999	1	0,010001
XRP-USD	0,349	0,342	-2,00573

Table 51 shows that the majority of assets are much more stable when comparing with the training data. BTC similarly to training data has an increase of near 9%. The most performative assets were BTC, followed by TRX with 6,15% increase. The least performative assets were ETC with a return of -57.79% followed by ETH and MATIC with a loss near to 27%.

4.4.2 A2C

4.4.2.1 Reward R1

Advantage Actor Critic obtained the following portfolio values trying to maximize profits.

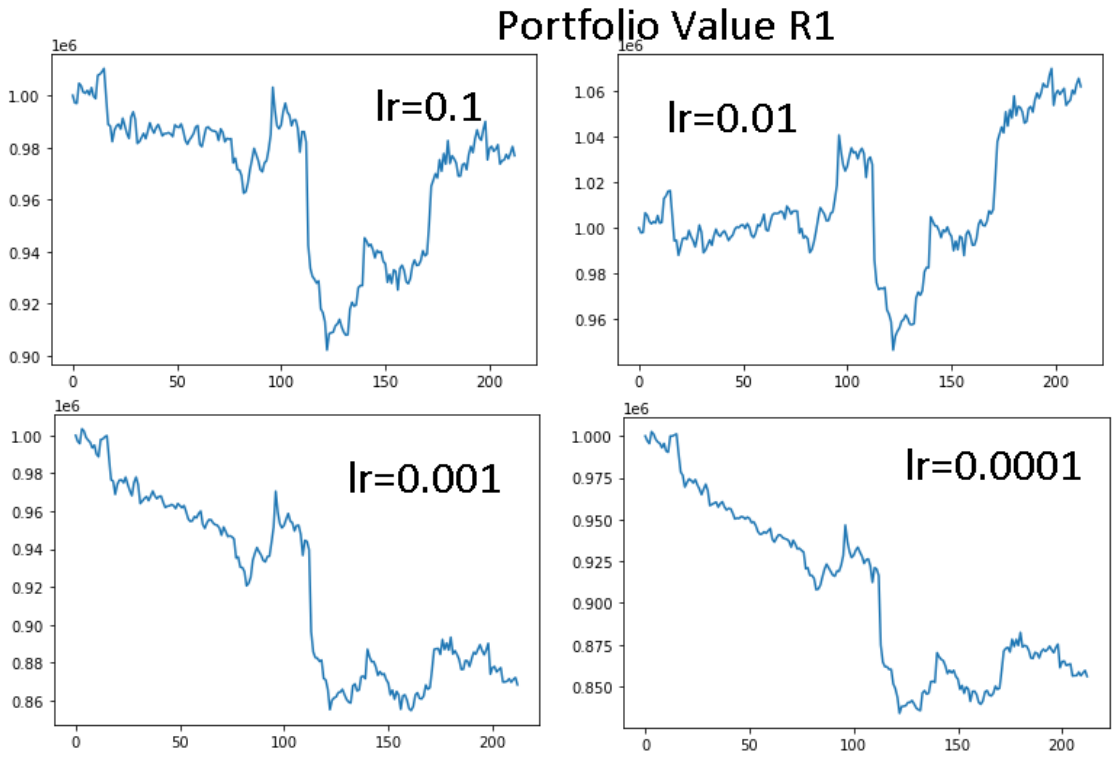


Figure 89 A2C portfolio values using R1 on sideways data

Figure 89 shows that the highest two learning rates (0.1 and 0.01) have a higher return than models with learning rates of 0.001 and 0.0001. The model with a learning rate of 0.01 shows to be the one that loses less money when the market fall and gain more when the market rises. The performance for each learning rate is presented in the table below.

Table 52 A2C performance using R1 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	855905.98 (-14.40)	212	0
0.001	868346.23 (-13.16)	205	7
0.01	1061955.61 (+6.19)	1	211
0.1	976848.97 (-2.31)	96	116

From the table above, the model with a learning rate of 0.01 was the only one that used the strategy of rebalance at the first timestep and hold until the end of the trading period and turned out to be the only profitable model with a return of 6.19%, unable to outperform the best benchmark strategy equal-weighted portfolios that had a return of 6.65% on this market conditions. The portfolio allocation for this model is presented in the figure below.

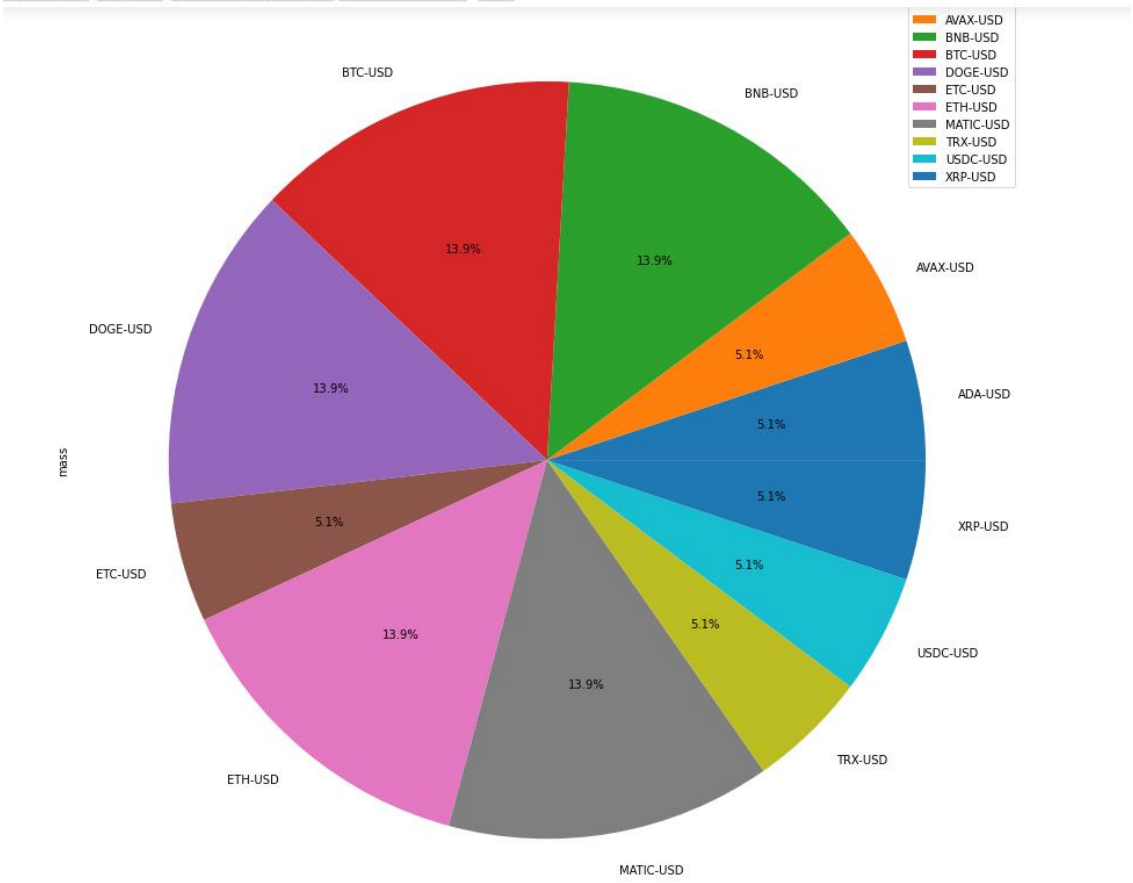


Figure 90 A2C allocation using R1 a learning rate of 0.01

Figure 90 shows that the model seems to have adapted well to the training data, including high allocation for the most performative assets MATIC(grey) and ETH(pink) while keeping low allocations for the least performative ones USDC(light blue) and ADA(blue). On test data this strategy is not optimal since ETH and MATIC are among the worst performing assets.

4.4.2.2 Reward R2

With the goal of outperforming BTC, A2C produced the following portfolio values.

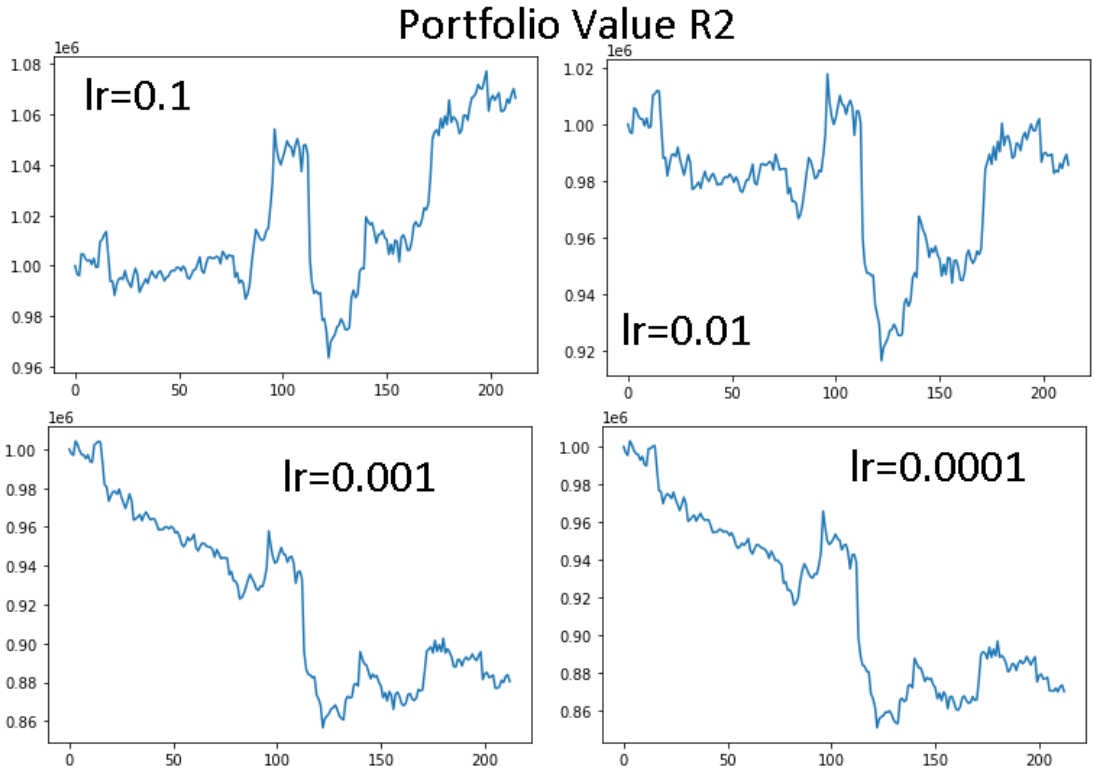


Figure 91 A2C portfolio values using R2 on sideways data

Figure 91 shows that similarly to R1 only the two models with the highest learning rate were able to produce viable or near viable strategies. The performance for each learning rate is presented in the table below.

Table 53 A2C performance using R2 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	870351.39 (-12.96)	212	0
0.001	880355.96 (-11.96)	181	31
0.01	985625.83 (-1.43)	85	127
0.1	1066566.20 (+6.65)	1	211

From the table above, it is observable that the model with a learning rate of 0.1 was able to match the strategy of equally weighted portfolios, with a return of 6.65%. Models started learning how to hold with a learning rate of 0.001, but it only holds fully after the first timestep with a learning rate of 0.1. None of the other learning rates produced positive returns. The portfolio allocation for this strategy is presented in the figure below.

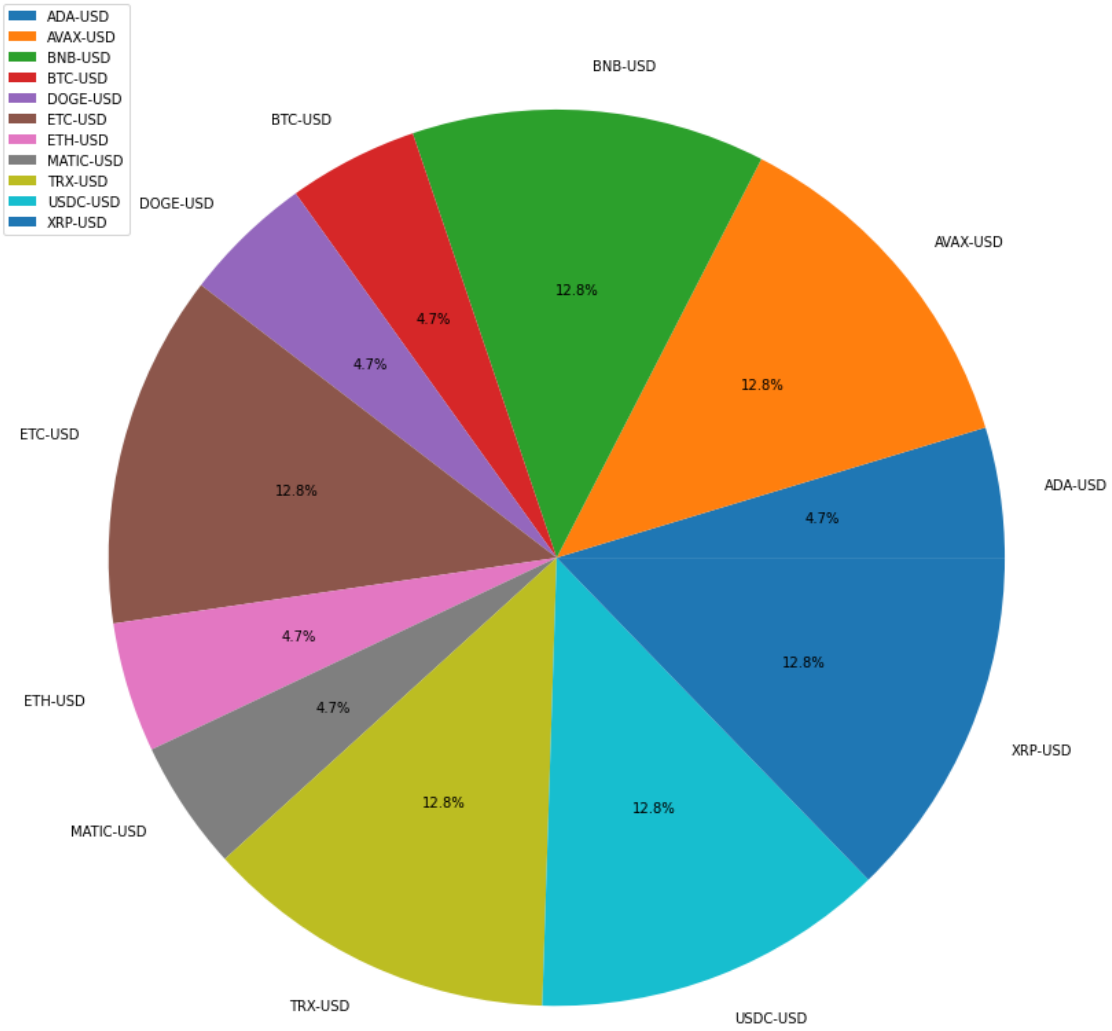


Figure 92 A2C with R1 and a learning rate of 0.1

Figure 92 shows that this strategy included one of the best assets TRX (yellow) in a high allocation category. The most underperforming asset, ETC(brown) ,was also included in the high allocation and the model was still able to have a higher return than holding an 100% of the second most performing asset TRX, which had a return of 6.15%.

4.4.2.3 Reward R3

A2C with a goal of outperforming BTC and USDC obtained the following portfolio values:

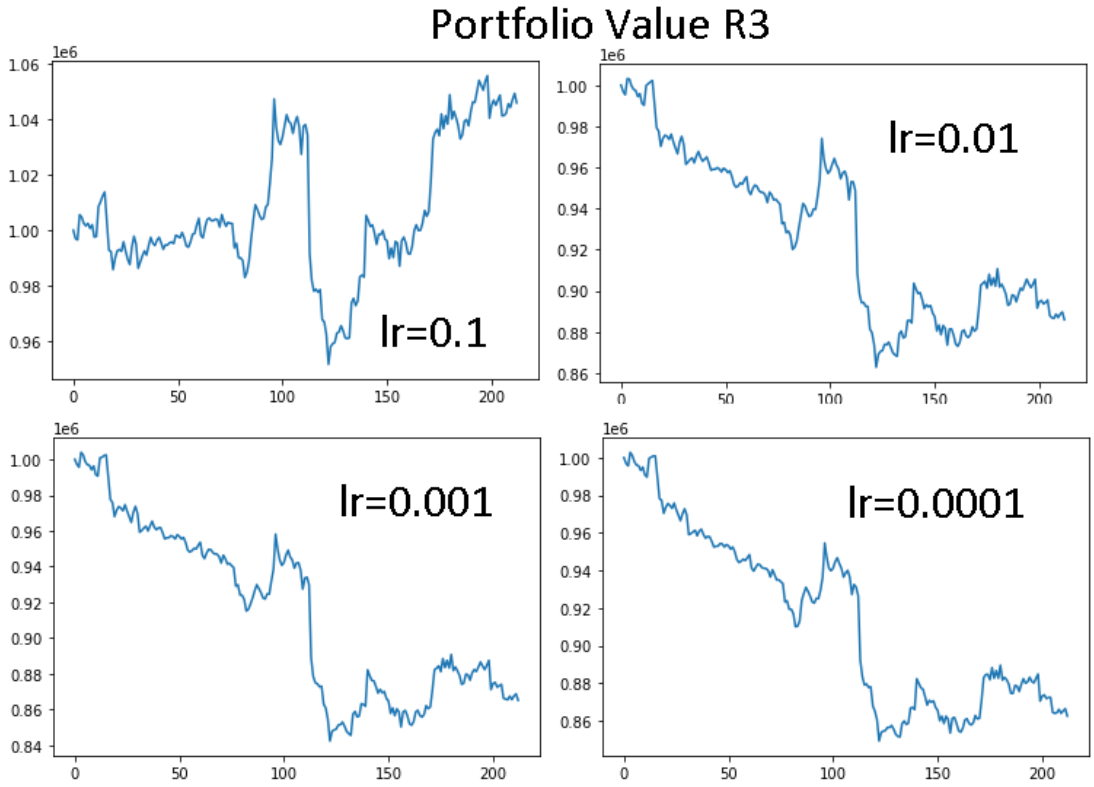


Figure 93 A2C portfolio values using R3 on sideways data

Figure 93 shows that only the model with a learning rate of 0.1 was stable. All the other models constantly lose value. The performance for each model is presented below:

Table 54 A2C portfolio values with R3 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	862522.73 (-13.74)	212	0
0.001	865140.26 (-13.48)	212	0
0.01	885921.53 (-11.40)	194	18
0.1	1046085.27 (4.60)	23	189

From the table above we can confirm that only with a learning rate of 0.1 was achieved a positive reward. None of the models adopted a strategy of buy and hold until the end of the trading period and models only started learning how to hold with a learning rate of 0.01.

4.4.3 DDPG

4.4.3.1 Reward R1

Deep deterministic policy gradient going for the maximization of profits each timestep obtained the following portfolio values.

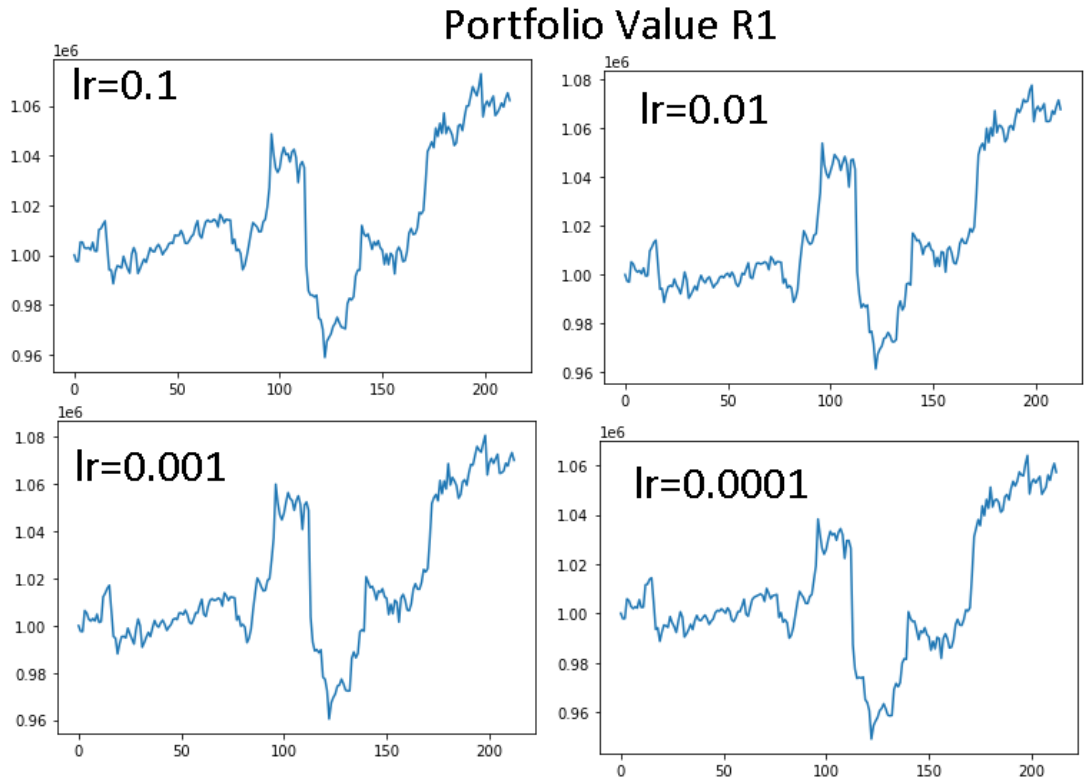


Figure 94 DDPG portfolio values using R1 on sideways data

Similarly to other market conditions, DDPG seems very stable with different learning rates, producing only profitable models. The performance of this algorithm is presented in the table below.

Table 55 DDPG performance using R1 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	1057167.52 (5.71)	1	211
0.001	1070192.66 (7.01)	1	211
0.01	1067629.89 (6.76)	1	211
0.1	1062172.87 (6.21)	1	211

Table 55 confirms that all models had a positive return and all of them used the strategy of rebalance on the first timestep and holding until the end of the ending period. The most profitable model was obtained with a learning rate of 0.001 with a return of 7.01, outperforming equally weighted portfolios. The allocation for each model is presented in the figure below.

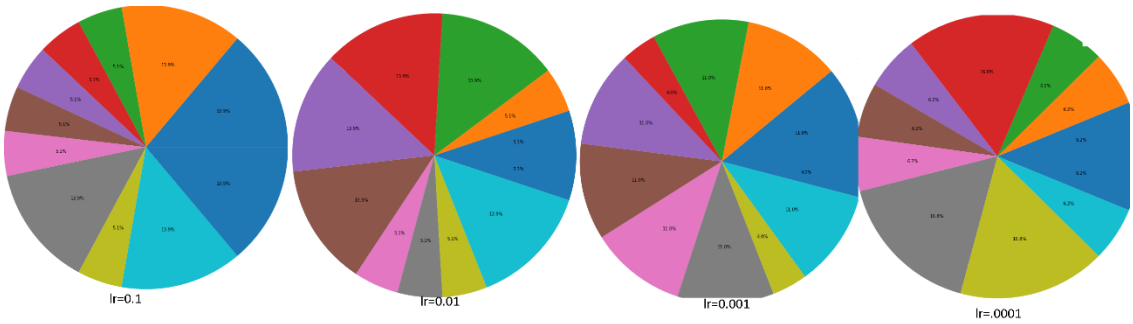


Figure 95 DDPG allocations using R1 on sideways data

4.4.3.2 Reward R2

DDPG with the goal of outperforming BTC obtained the following portfolio values.

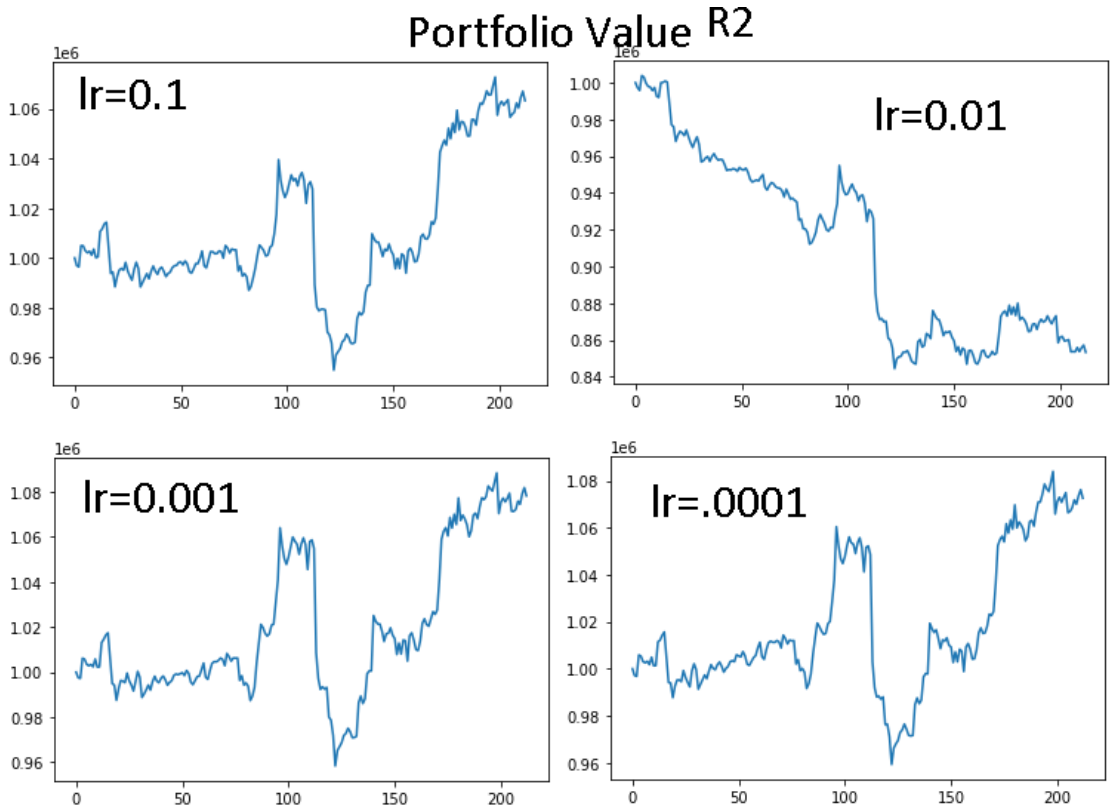


Figure 96 DDPG portfolio values using R2 on sideways data

The figure above shows that only the model with a learning rate of 0.01 produced a losing strategy. The performance for each learning rate is presented in the table below.

Table 56 DDPG performance using R2 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	1072573.64 (7.25)	1	211
0.001	1078229.82 (7.82)	1	211
0.01	853265.46 (-14.67)	212	0
0.1	1063234.88 (6.32)	1	212

Table 56 confirms that the mode with a learning rate of 0.01 was the only one with a negative reward of 14.67%. The model with the highest performance was produced with a learning rate of 0.001, outperforming the previous best model DDPG-R1 with a loss 0.81% lower of the initial portfolio value. The allocations for the models that hold are presented in the figure below.

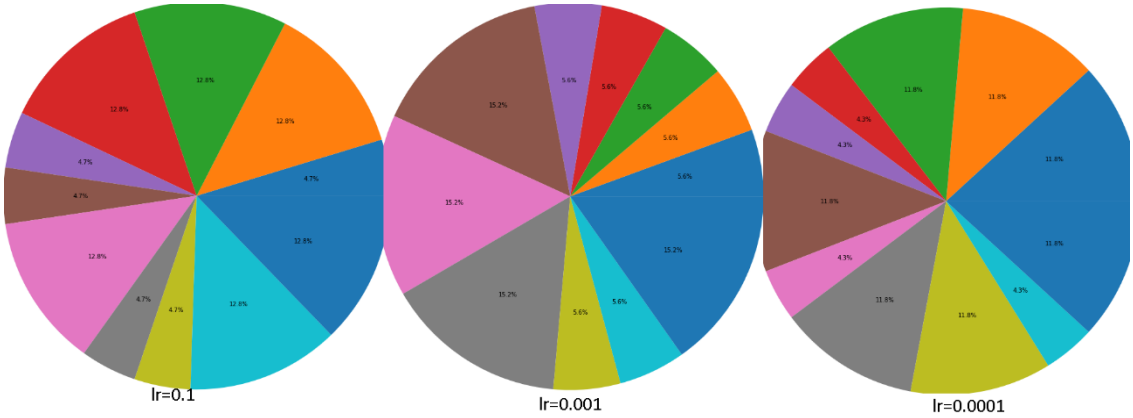


Figure 97 DDPG allocation using R2 on a sideways market

4.4.3.3 Reward R3

DDPG trying to outperform BTC and USDC obtained the following results.

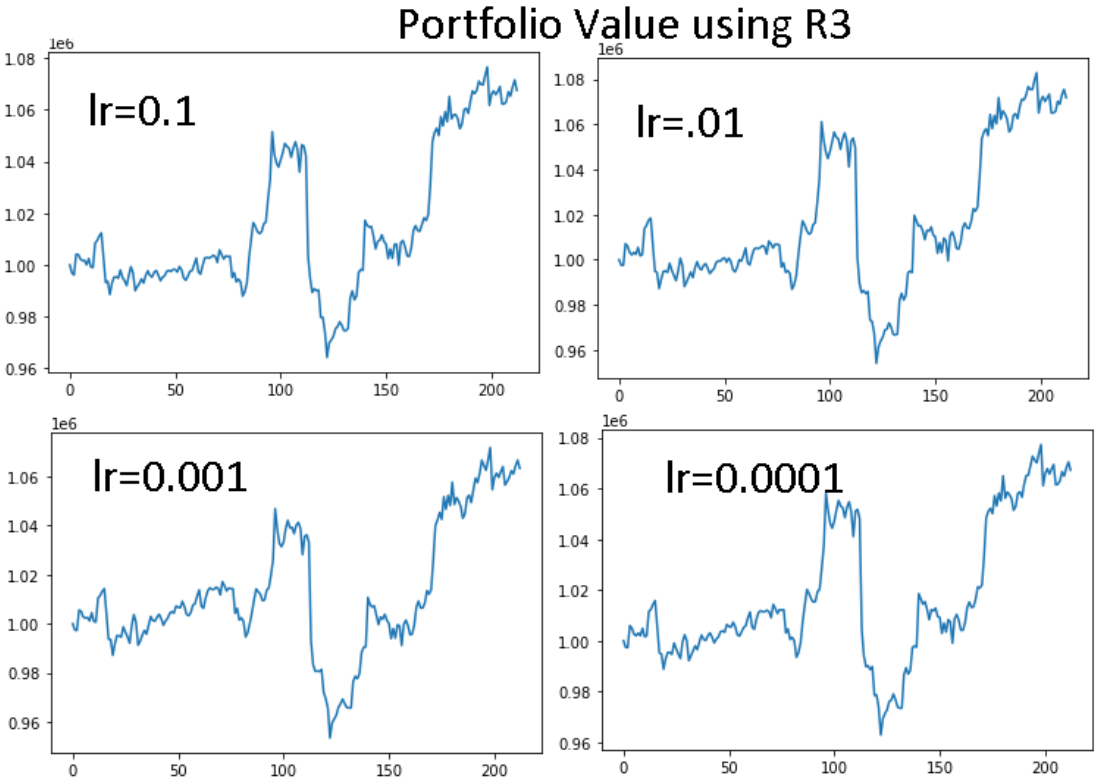


Figure 98 DDPG portfolio value using R3 on sideways data

Similarly to R1, R3 is able to produce profitable strategies with all learning rates. The performance for each model is presented in the table below.

Table 57 DDPG perfomrnac eusing R3 non sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	1067230.95 (6.72)	1	211
0.001	1063308.87 (6.33)	1	211
0.01	1071874.72 (7.18)	1	211
0.1	1067509.96 (6.75)	1	211

Table 57 shows that every model has a positive return and all of the models learned the strategy of only rebalance at the first timestep to obtain a diversified portfolio and then hold it until the end of the trading period. The allocation for each model is presented in the figure below.

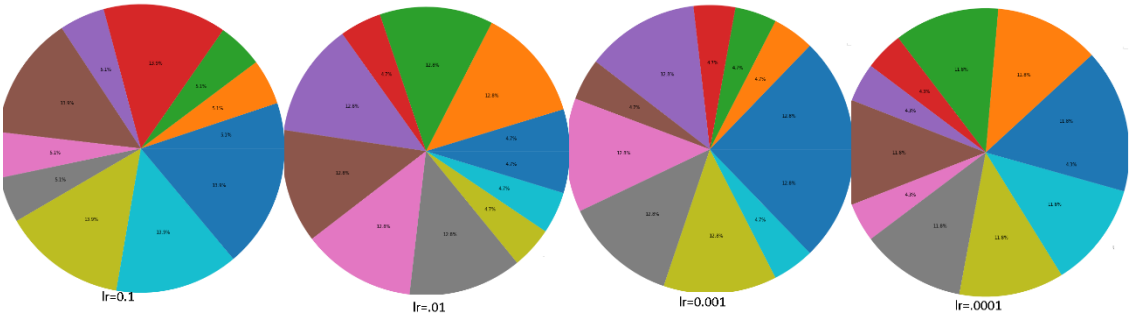


Figure 99 DDPG allocation with R3 on sideways data

4.4.4 PPO

4.4.4.1 Reward R1

Proximal policy optimization obtained the following portfolio values when trying to maximize profits.

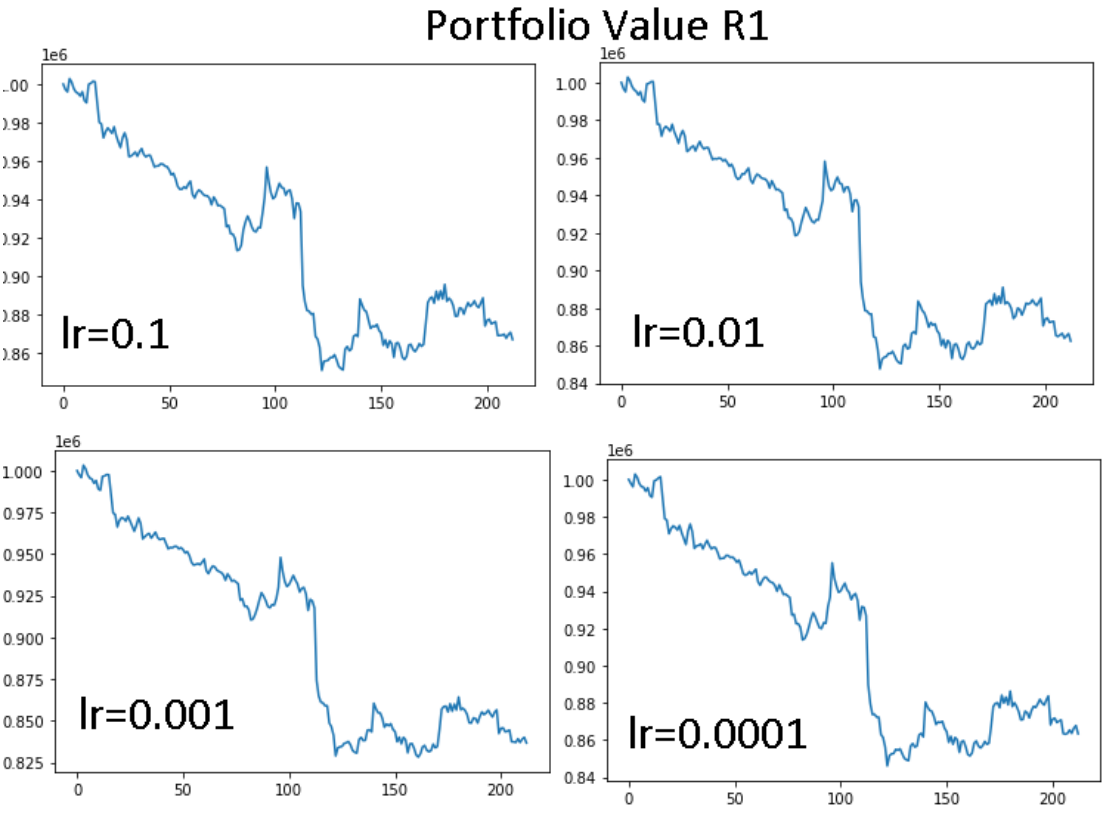


Figure 100 PPO portfolio values using R1 on sideways data

Similarly to other market conditions, PPO seems very underperforming and wasn't able to produce a viable strategy. The summary of the performance of each learning rate is presented in the table below.

Table 58 PPO performance using R1 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	863427.36 (-13.65)	212	0
0.001	836667.20 (-16.33)	212	0
0.01	862438.01 (-13.75)	212	0
0.1	866712.04 (-13.28)	212	0

Table 58 confirms that none of the models have been able to produced profitable strategies and none of them held the same portfolio two timestep in a row.

4.4.4.2 Reward R2

PPO trying to outperform BTC obtained the following results.

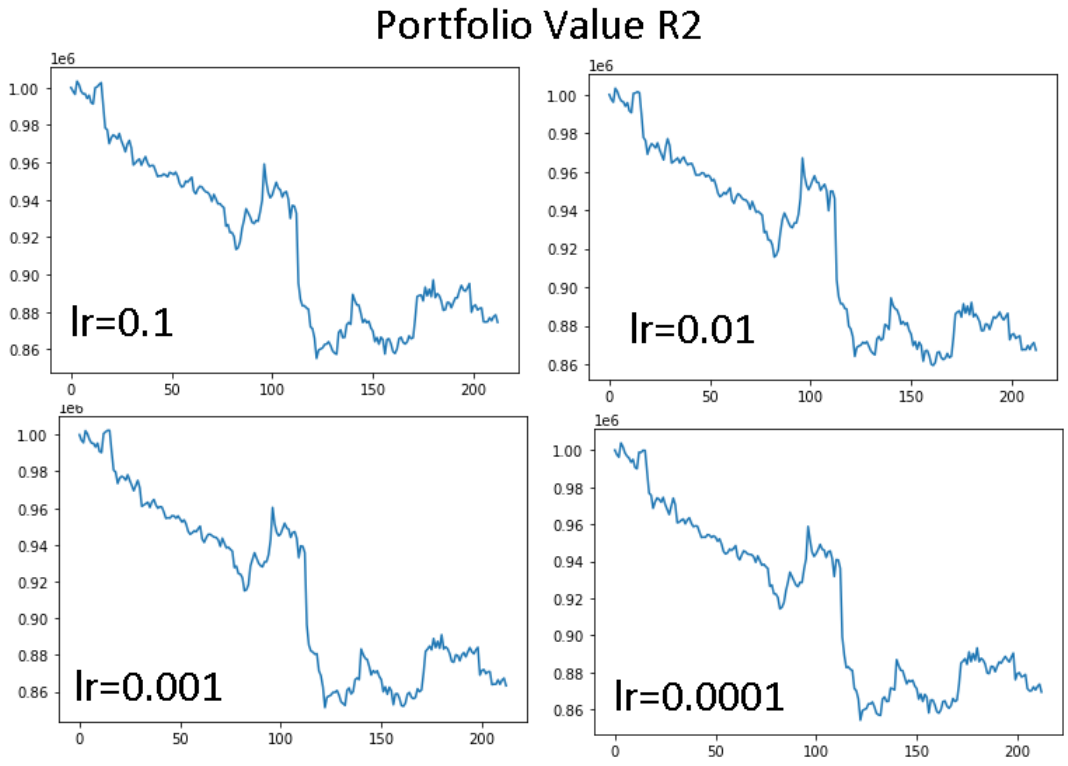


Figure 101 PPO portfolio values using R2 on sideways data

Just as in the section above, PPO wasn't able to produce a viable strategy. The summary of the performance of this algorithm is presented in the table below.

Table 59 PPO performance using R2 oon sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	869451.076 (-13.05)	212	0
0.001	863282.70 (-13.67)	212	0
0.01	867337.65 (-13.26)	212	0
0.1	874402.50 (-12.65)	212	0

Table 59 shows that across all the learning rates the portfolios were rebalanced during the trading period.

4.4.5 SAC

4.4.5.1 Reward R1

Soft-actor Critic with the goal of maximizing profits obtained the following results.

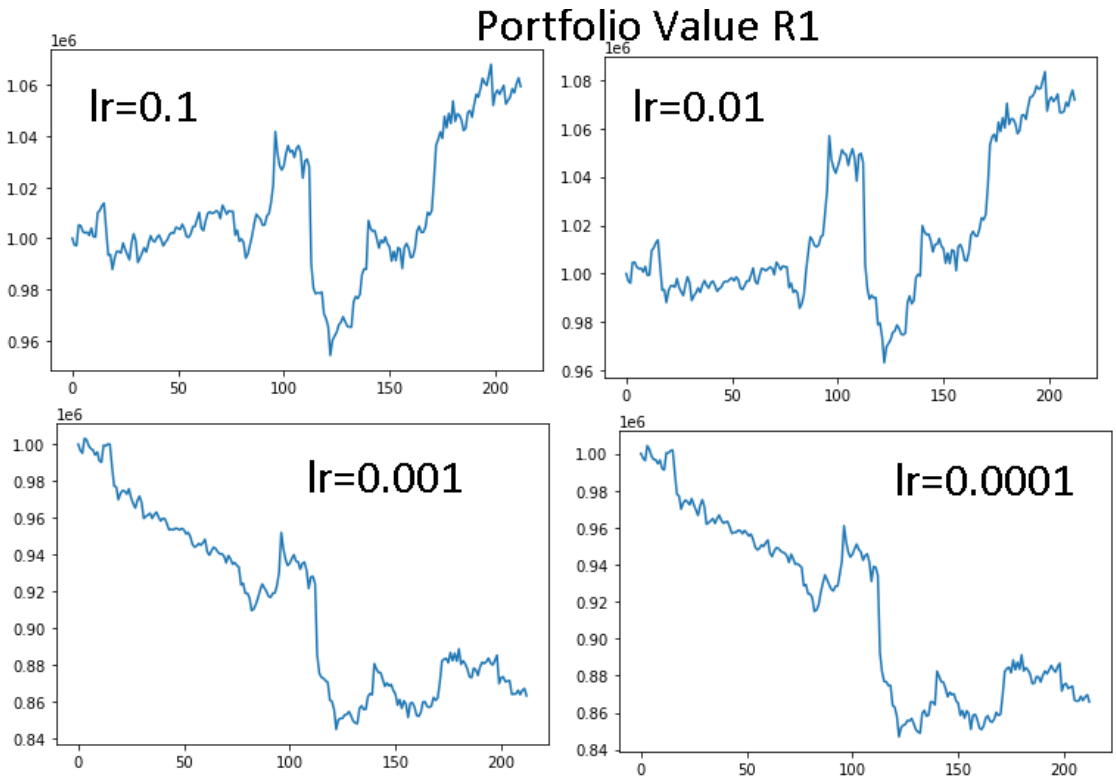


Figure 102 SAC portfolio values using R1 on sideways data

Figure 102 shows that SAC only produced profitable strategies with the two highest learning rates. The performance summary is presented in the table below.

Table 60 SAC performance using R1 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	865779.54 (-13.42)	212	0
0.001	863201.26 (-13.67)	212	0
0.01	1071765.73 (7.17)	1	211
0.1	1059379.96 (5.93)	1	211

From the table above it is observable that the two models with the lower learning rates rebalanced the portfolios every timestep of the trading session while the two models that had a positive return only rebalanced once. The highest performance model was obtained with a learning rate of 0.01 with a return of 7.17% better than equal weighted portfolios but worse than DDPG-R2. The allocation for the profitable models is presented below.

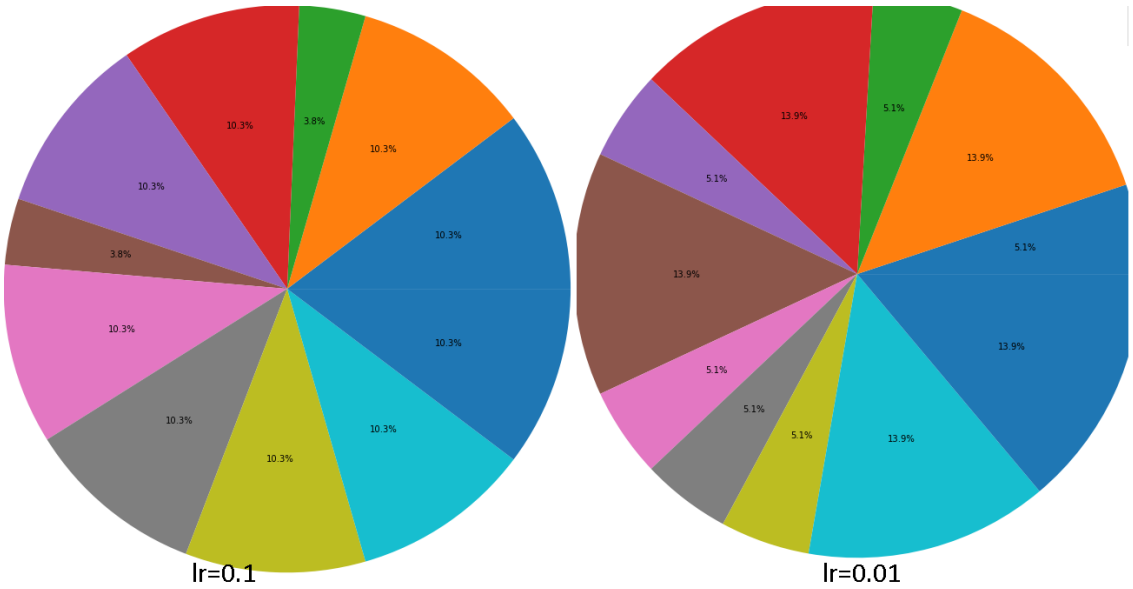


Figure 103 SAC portfolio allocation using R1 on sideways data

4.4.5.2 Reward R2

SAC with the goal of outperforming BTC performed as follows:

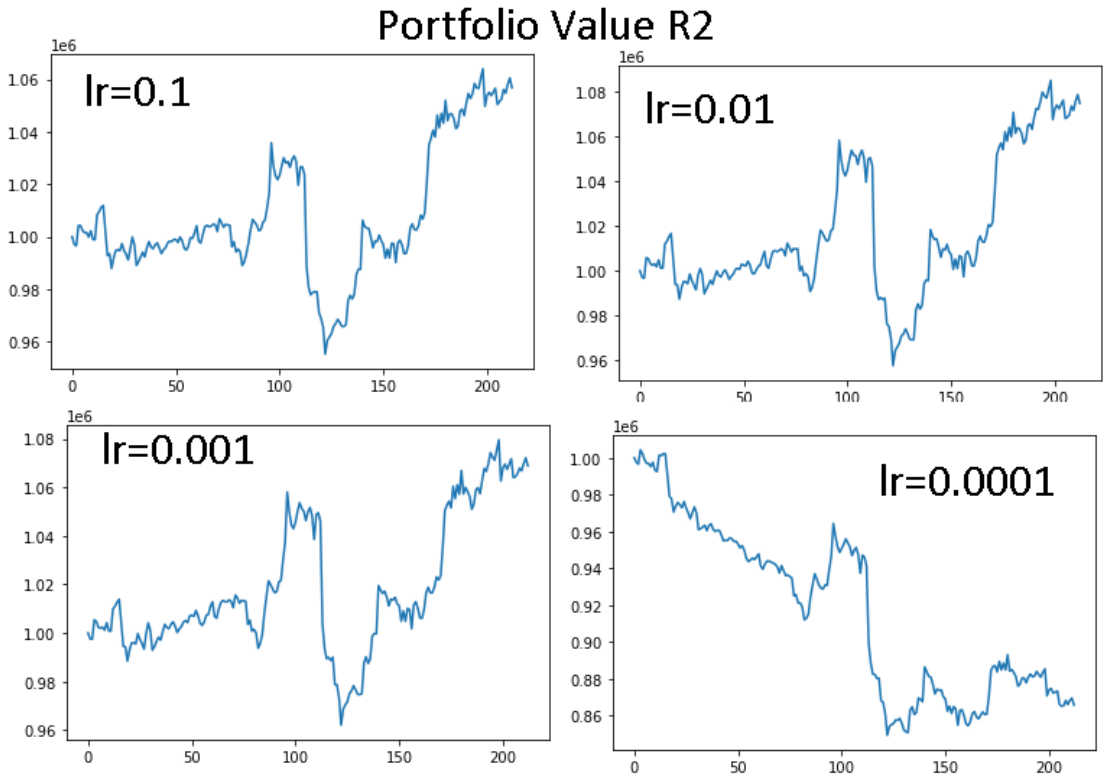


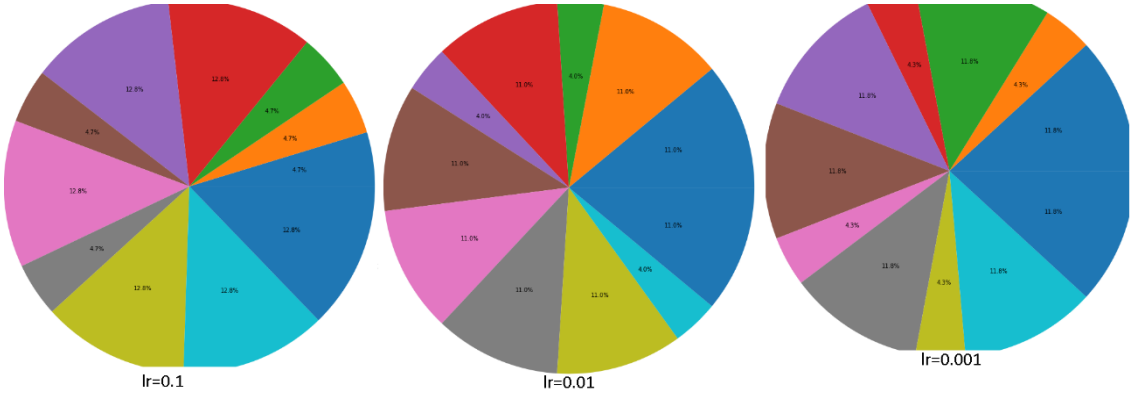
Figure 104 SAC portfolio value with R2 on sideways data

From this figure, we can see that only the three highest learning rates produced profitable strategies while the lower learning rate had a significant loss. The summary for each learning rate is presented below

Table 61 SAC performance with R2 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	865753.69 (-13.42)	212	0
0.001	1068899.72 (+6.88)	1	211
0.01	1074747.00 (+7.47)	1	211
0.1	1056822.56 (5.68)	1	211

Table 61 shows that only the model with a learning rate of 0.0001 rebalanced the portfolio every timestep while all the others rebalanced only on the first timestep and held until the end of the trading period. The allocations for the profitable models are presented in the figure below.



4.4.5.3 Reward R3

Soft-actor critic presented the following results when trying to outperform BTC and USDC at the same time:

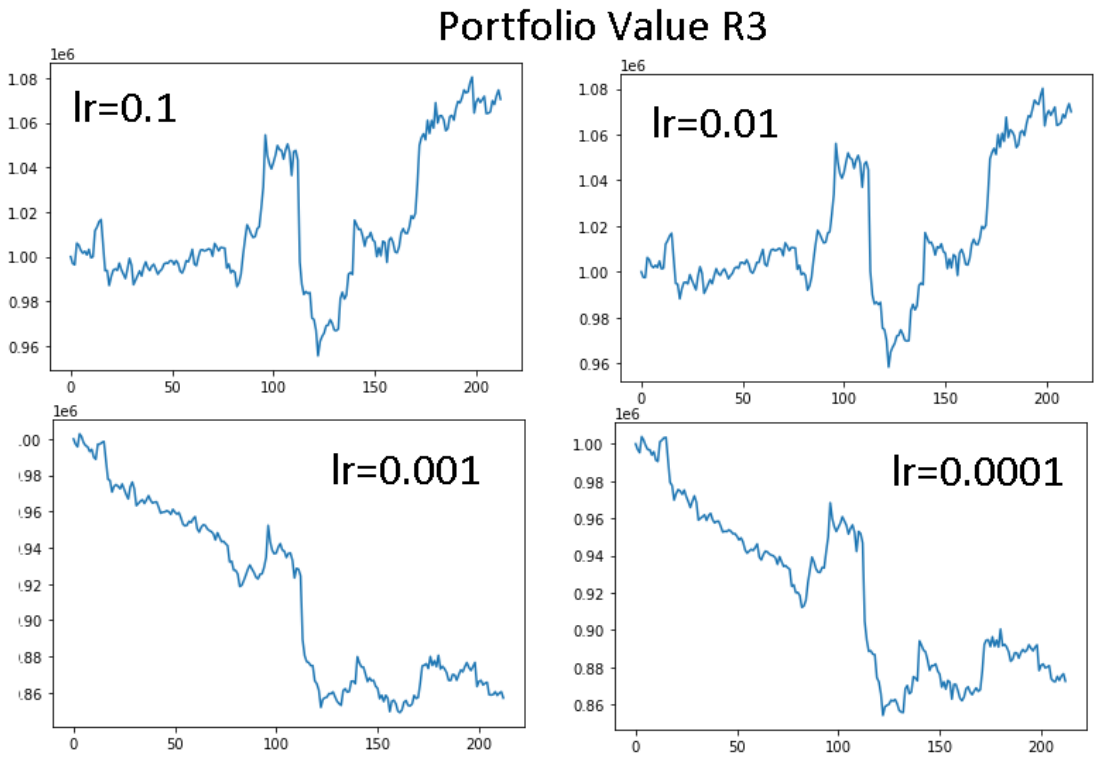


Figure 105 SAC portfolio values using R3 on sideways data

It is observable from Figure 105 only the learning rates of 0.1 and 0.01 produced profitable strategies. The summary for the performance of each learning rate is presented in the table below.

Table 62 SAC performance with R3 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	872415.45	212	0

	(-12.75)		
0.001	857018.63 (-14.29)	212	0
0.01	1069962.44 (6.99)	1	211
0.1	1070483.49 (7.04)	1	211

From this table we can see that the models that weren't profitable rebalanced the portfolio every timestep while others rebalanced the portfolio on the first timestep and held until the end. The allocation for the models that only rebalanced on the first timestep is presented in the figure below.

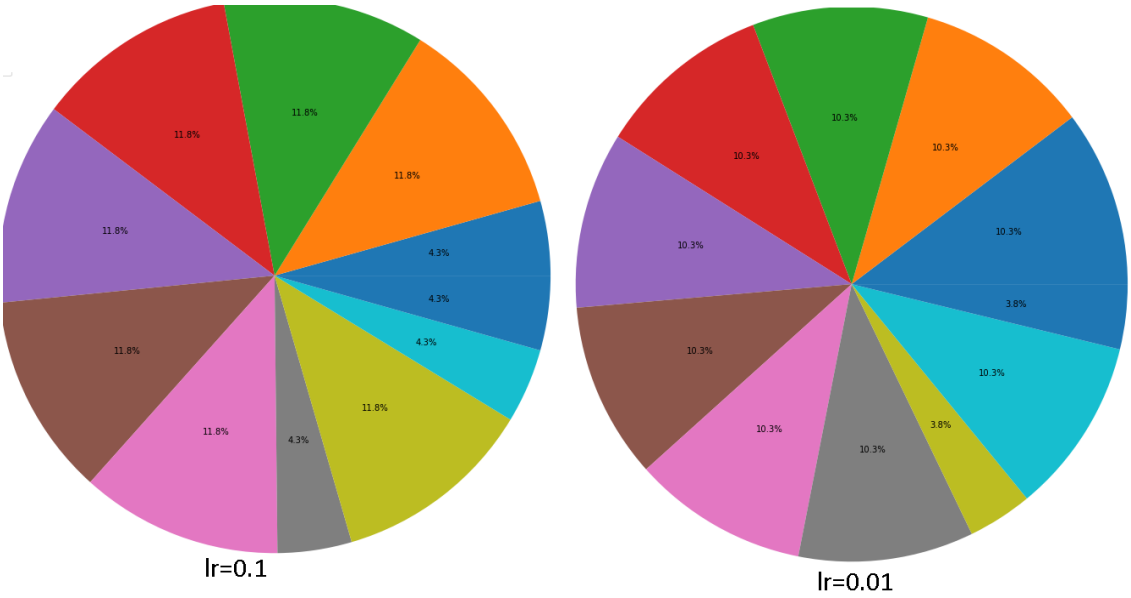


Figure 106 SAC allocaiton using R3 on sideways data

4.4.6 TD3

4.4.6.1 Reward R1

TD3 maximizing profit obtained the following portfolio values during the trading session.

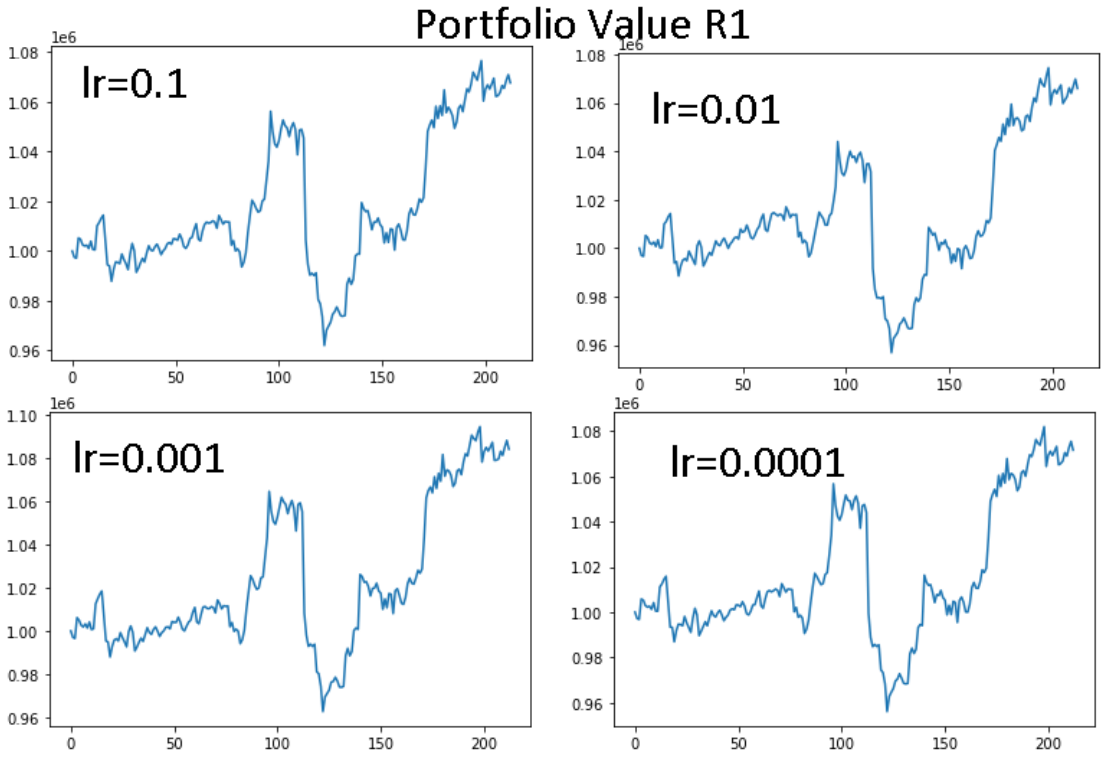


Figure 107 TD3 portfolio values using R1 on sideways data

From the figure above, we can see that similarly to his brother DDPG, TD3 is able to produce viable strategies across all the learning rates. The summary for the performance is presented in the table below.

Table 63 TD3 performance using R1 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	1071694.96 (7.16)	1	211
0.001	1084139.93 (8.41)	1	211
0.01	1066006.39 (6.60)	1	211
0.1	1067701.30 (6.77)	1	211

Table 63 shows that every model adopted the strategy of rebalancing on the first timestep and holding until the end. The mode with a learning rate of 0.001 had a positive return of 8.41% outperforming DDPG-R2. The allocation of each learning rate is presented below.

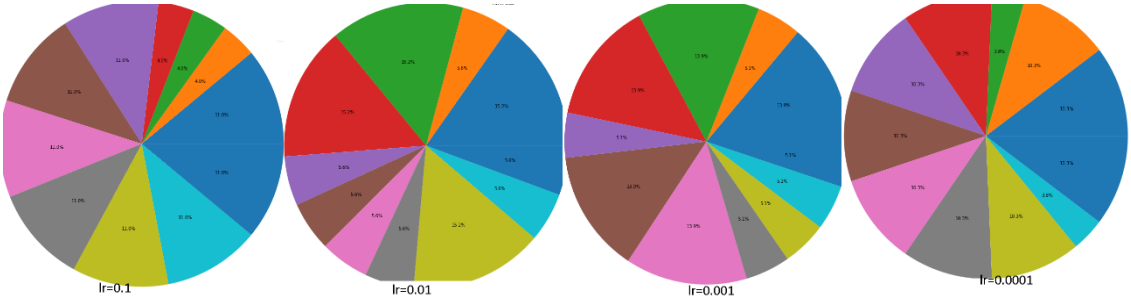


Figure 108 TD3 allocation using R1 on sideways data

4.4.6.2 Reward R2

When trying to outperform BTC, TD3 obtained the following portfolio values.

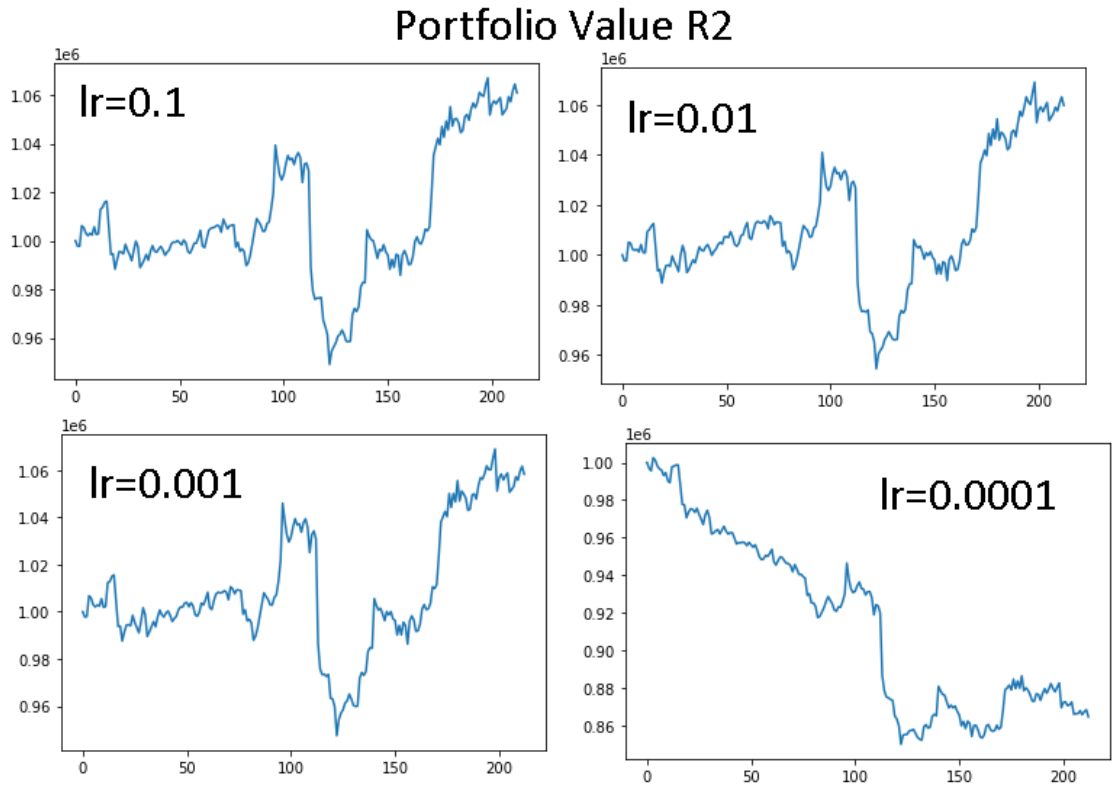


Figure 109 TD3 protfolio values using R2 on sideways data

From the figure above it seems that this reward is harder to learn and the model with a learning rate of 0.0001 couldn't produce a profitable strategy. The summary for each learning rate is presented in the table below.

Table 64 TD3 performance using R2 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	864469.43 (-13.55)	206	6

0.001	1058286.33 (5.82)	1	211
0.01	1059692.53 (5.96)	1	211
0.1	1060921.00 (6.09)	1	211

We can see from the table above that, similarly to R1, the highest three learning rates rebalanced on the first timestep and held but now the model with the lowest learning rate only holds six times. The allocation for each profitable model is presented below.

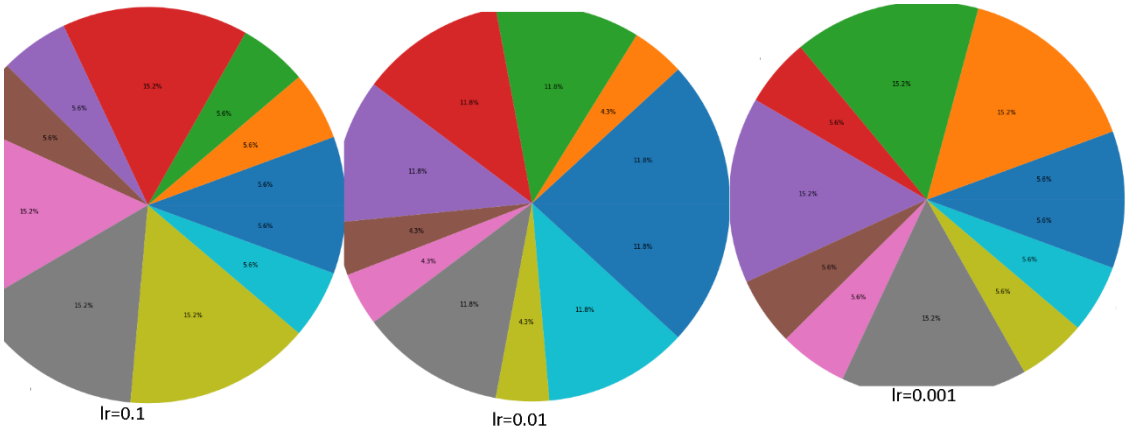


Figure 110 TD3 allocation using R2 on sideways data

4.4.6.3 Reward R3

With the goal of outperforming BTC and USDC, TD3 produced the following portfolio values during the trading period.

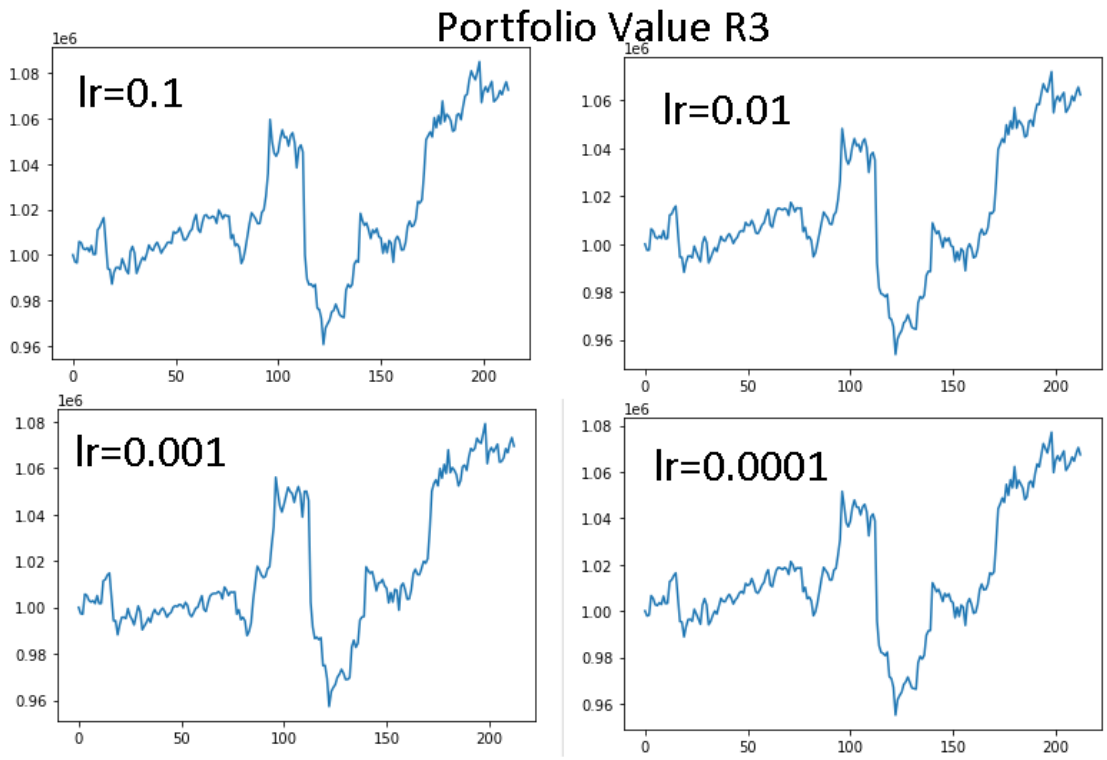


Figure 111 TD3 protfolio values using R3 on sideways data

Figure 111 shows that similarly to R1, this algorithm was able to have positive returns across all the learning rates. The performance for each learning rate is presented below.

Table 65 TD3 performance using R3 on sideways data

Learning Rate	Final Portfolio Value (+ profit%)	Number of portfolio rebalances	Number of holds
0.0001	1067435.49 (6.74)	1	211
0.001	1069382.93 (6.93)	1	211
0.01	1062381.08 (6.23)	1	211
0.1	1072487.09 (7.24)	1	211

From this table we can observe that TD3 was able to produce good results and just rebalanced the portfolio once across all the learning rates. The rebalanced portfolios are presented in the figure below.

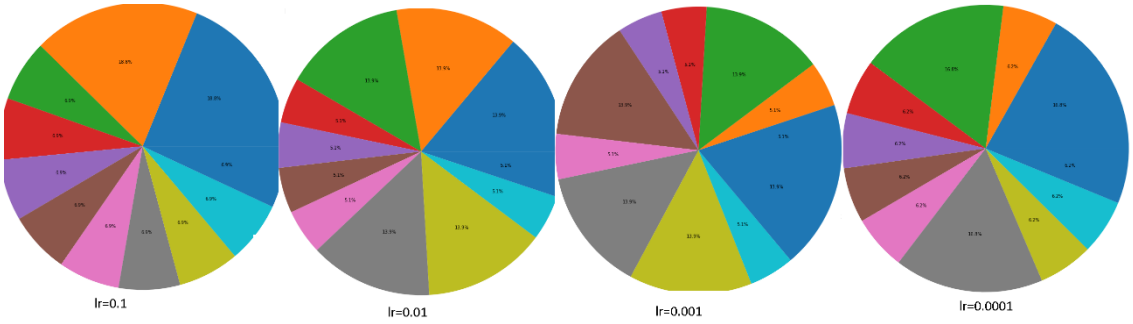


Figure 112 TD3 allocation using R3 on sideways market

4.5 Summary

This chapter outlined the results across all the different scenarios. Five DRL algorithms were analysed considering 4 different learning rates and three different reward functions. On top of that, three market scenarios were analysed and all the scenarios were compared with the benchmark strategies from sub-chapter 3.8. DDPG and TD3 appear to be the most stable models among the ones that were tested while PPO wasn't able to produce viable results in any market condition. The next chapter, the results obtained are going to be discussed and the main conclusions are going to be outlined.

5 Conclusion and Result Discussion

The implemented DRL models do perform relatively better than the best benchmark strategy equal-weighted portfolios. Strategies and models that go for a constant rebalance of the portfolio almost always lose money overtime. Considering bull markets, equal-weighted portfolios produced a return of 22.08% and were outperformed by DDPG using R1 and R2, SAC with R3 and TD3 using R2 and R3. The most profitable model on a bull market was DDPG using R2 with a learning rate of 0.01 producing a return of 26.78, an improvement of 4,7%. A2C in the best performative bull market models was starting to avoid portfolio rebalance but still did not learn to hold through the end. PPO rebalanced at all timesteps across all the rewards. In a bear market, once again, the highest return benchmark strategy was equal-weighted portfolios with -38.03% return, in a market where BTC lost 50% of its value. The highest performance model was TD3 using R2 with a learning rate of 0.001 returning -34.45%, saving 3,58% of the portfolio value when comparing to equal weights. On a sideways market, equal-weighted portfolio returns 6.65% over the trading period and was outperformed by TD3 with a learning rate of 0.0001, using R1 that produced a return of 8.41%, improving 1,76% over equal-weighted portfolios. Proximal policy optimization could not produce a model that was able to learn how to hold a portfolio, in fact it always rebalanced the portfolio every timestep resulting in the worst performance model. DDPG proved to be a very stable model being the only model that learnt the power of holding in almost every scenario. When comparing rewards, outperforming BTC (R2) seems to be more effective than going for pure profit (R1) in bull markets and bear markets. More complex rewards such as R2 and R3 seem to be harder for models to learn but, at the same time, they also seem to promote exploration.

From the results, we can answer the first research question **Q1** (“What strategy is the most reliable to manage a portfolio?”) and say that the most reliable strategy to manage a portfolio is somewhere along the lines of an equally weighted portfolios. The results showed that some models could outperform this strategy by optimizing it and pick a higher allocation for most performative models, but in most of the cases it is a matter of luck. We can speculate and attribute a higher allocation to the assets we think are going to perform better but at the same time if we fail our predictions we are going to lose more when comparing to equally weighted portfolios, therefore increasing the risk. Following the old maxim “don't put all your eggs in one basket” and holding while we want to stay in the market seems to be the balance between risk and reward.

To answer **Q2** (“How different Deep Reinforcement Learning algorithms perform on various market conditions?”), the results show that the algorithms perform in a similar way in different market conditions. For all of the three market conditions, the most performative algorithms stick with the same strategy, a near equally distributed portfolio and not rebalancing until the end of the trading period.

In **Q3**(“What is the impact of having suggestive reward functions?

"), the results from different rewards seem very similar and are inconclusive. We can say that more complex rewards seem to promote exploration and seem to be harder to learn, but in future worker this question needs to be addressed with more depth.

From these results and to answer **Q4**("Is cryptocurrency algorithmic trading a fallacy?"), one can only conclude that day trading the crypto market is like gambling. The results show that if the market is up, we make money and if it is down we lose money. With algorithmic trading we were only able to slightly minimize the loss when the market is down and perform better than most assets when the market is up by not constantly trading the market. Models that tried to trade at each timestep had a massive loss when comparing to other models. Some investors like Warren Buffet even say that technical analysis is like astrology and if a model is really able to consistently make even 1% per day the person who developed this model wouldn't have the necessity to sell it to others since he would rapidly become one of the richest persons on earth.

To conclude, the greater enemy of a trader or investor seems to be the trading fees. Even though a small number like 0.1% seems harmless, if a trader is constantly rebalancing its portfolio, it will result in an exponential decay of the portfolio value. The HODL philosophy proved to be efficient and equal-weighted portfolios proved to be a hard to beat strategy. The produced models seem to use a variant of equal weights where, instead of splitting equally among all assets, they divide the assets into two categories, one for high allocations, and another for low allocations, and divide equally inside those categories. While the presented results indicate the above conclusions, there is still an almost infinite room for improvement that are going to be discussed in the next chapter.

6 Future Work

Further works should be explored to improve the performance of the DRL models to be able to significantly outperform the equal-weighted portfolios. Some possible actions to achieve this goal might be:

- Developing a reward where the objective each timestep is to take an action capable of beating equal-weighted allocation
- Increase the amount of assets used.
- Explore more technical indicators
- Increase the amount of training steps
- Incorporate sentiment analysis as part of the state of the environment

In this work, there are almost limitless scenarios that we can explore. We can test numerous reward functions, a vast number of assets including assets from different markets, like commodities, stock market and bonds and add other technical indicators making the model aware of the state of the global economy. With the increasing dimensionality, we most likely will need a massive amount of training steps so that the models start learning useful information and consequently very large amounts of computational power. The sentiment analysis is a dangerous indicator since nowadays there is an increasing amount of bots that try to manipulate social media in order to create speculation around an asset.

7 Referências

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2009.

[2] M. D. P. M. P. Constantin Zopounidis, Handbook of Financial Engineering, 2008.

[3] Z. Jiang, “A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem”.

[4] H. P. B. W. P.-Y. C. Y. Z. J. X. Yunan Ye, Reinforcement-Learning Based Portfolio Management with Augmented Asset Movement Prediction States, Proceedings of the AAAI Conference on Artificial Intelligence, 2020.

[5] X. F. Y. S. M. L. Yifeng Guo, “Robust Log-Optimal Strategy with Reinforcement Learning,” 1 May 2018.

[6] G. L. &. M. Borrott, “A deep Q-learning portfolio management framework for the cryptocurrency market,” em *Neural Computing and Applications volume*, 2020, p. 17229–17244.

[7] J. L. Zhengyao Jiang, “A deterministic deep reinforcement learning technique for tackling the,” em *Conference: IntelliSys 2017*, London, UK, 2017.

[8] H. Y. J. G. C. D. W. Xiao-Yang Liu, “FinRL: Deep Reinforcement Learning Framework to Automate Trading in Quantitative Finance,” em *ACM International Conference on AI in Finance*, New York City, USA, 2021.

[9] M. B. Giorgio Lucarelli, “A Deep Reinforcement Learning Approach for Automated Cryptocurrency Trading,” em *15th IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI)*, Hersonissos, Greece, 2019.

[10] L. W. M. X. Xudong Sun, “A Portfolio Trading System of Digital Currencies,” *The International Journal of Engineering and Science,* vol. 9, nº 03, pp. 36-43, 2020.

[11] A. M. W. L. K. K. O. A. J. O. a. H. S. J. Otabek Sattarov, Recommending Cryptocurrency Trading Points with Deep Reinforcement Learning Approach, Korea, 2020.

[12] J. O. B. H. M. J.-A. P. Ali Hirsa, DEEP REINFORCEMENT LEARNING ON A MULTI-ASSET ENVIRONMENT FOR TRADING, 2021.

[13] Z. W. a. A. F. Yu chien (Calvin) Ma, “Deep Q-Learning for Trading Cryptocurrency,” *The Journal of Financial Data Science Summer ,* 2021.

[14] P. Z. W. B. L. J. H. M. T. Yifan Zhang, “Cost-Sensitive Portfolio Selection via Deep Reinforcement Learning,” *IEEE Transactions on Knowledge and Data Engineering ,* vol. 34, nº 1, pp. 236 - 248, 2022.

[15] Y.-H. Miao, Y.-T. Hsiao e S.-H. Huang, “Portfolio Management based on Deep Reinforcement Learning with Adaptive Sampling,” em *International Conference on Pervasive Artificial Intelligence (ICPAI)*, Taipei, Taiwan, 2020.

[16] J.-H. Syu, M.-E. Wu e J.-M. Ho, “Portfolio Management System with Reinforcement Learning,” em *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Toronto, ON, Canada, 2020.

[17] J. Z. J. S. Huanming Zhang, “A Deep Deterministic Policy Gradient-based Strategy for Stocks Portfolio Management,” em *IEEE 6th International Conference on Big Data Analytics (ICBDA)*, 2021.

[18] J. T. R. M. T. A. Antonio Briola, “Deep Reinforcement Learning for Active High Frequency Trading,” 2021.

[19] R. Bellman, “Dynamic programming and stochastic control processes,” *Information and Control,* vol. 1, nº 3, pp. 228-239, 1958.

[20] R. A. HOWARD, “Dynamic Programming and Markov Processes,” 1960.

[21] R. Bellman, “The theory of dynamic programming,” *Bull. Amer. Math. Soc.,* pp. 503-515, 1954.

[22] L. S. Shapley, “A Value for n-Person Games,” em *Contributions to the Theory of Games*, 1953.

[23] D. Blackwell, “Discounted Dynamic Programming,” em *The Annals of Mathematical Statistics*, 1965.

[24] S. E. S. Dimitri P. Bertsekas, Stochastic Optimal Control: The Discrete Time Case, Academic Press, 1978, p. 323.

[25] E. a. Y. Dynkin, “Controlled Markov Processes,” em *Springer-Verlag*, New York, 1979.

[26] M. Puterman, “Markov Decision Processes: Discrete Stochastic Dynamic Programming,” *Wiley Series in Probability and Statistics,* 1994.

[27] O. a. L. J. Hernandez-Lerma, “Discrete-Time Markov Control Processes: Basic Optimality Criteria,” em *Springer*, New York, 1996.

[28] W. B. Powell, Approximate Dynamic Programming: Solving the Curses of Dimensionality, Wiley, 2007.

[29] R. S. S. a. A. G. Barto, Reinforcement Learning:An Introduction, London, England: The MIT Press, 2014.

[30] blackburn, “Towards Data Science,” [Online]. Available: https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da.

[31] “incomplete ideas,” [Online]. Available: http://incompleteideas.net/book/ebook/node46.html.

[32] L. Q. S. Y. J. L. H. L. F. P. Zhihan Lv, “Memory-augmented neural networks based dynamic complex image segmentation in digital twins for self-driving vehicle,” *Pattern Recognition,* vol. 132, 2022.

[33] R. T.-M. A. R. T. S. S. A. Fatemeh Hamedani-KarAzmoudehFar, “Breast cancer classification by a new approach to assessing deep neural network-based uncertainty quantification methods,” *Biomedical Signal Processing and Control,* vol. 79, nº 1, 2023.

[34] R. K. P. M. T. K. R. Rajagopal, “Deep Convolutional Spiking Neural Network optimized with Arithmetic optimization algorithm for lung disease detection using chest X-ray images,” *Biomedical Signal Processing and Control,* vol. 79, nº 2, 2023.

[35] S. I. E. T. M. J. S. Alessia David, “The AlphaFold Database of Protein Structures: A Biologist’s Guide,” *Journal of Molecular Biology,* vol. 434, nº 2, 2022.

[36] T. N. W. D. H. Hubel, “Receptive fields, binocular interaction and functional architecture in the cat's visual cortex,” 1 January 1962.

[37] R. Pramoditha, “Towards Data Science,” 26 Dec 2021. [Online]. Available: https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-

neural-networks-fab22249cbfc.

[38] A. R. Webb, Statistical Pattern Recognition, West Sussex, England: Butterworth-Heinemann, 2002.

[39] JürgenSchmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks,* vol. 61, pp. 85-117, 2015.

[40] “Brilliant,” [Online]. Available: https://brilliant.org/wiki/feedforward-neural-networks/.

[41] W.-H. C. Carlos Betancourt, “Deep reinforcement learning for portfolio management of markets with a dynamic number of assets,” *Expert Systems with Applications,* vol. 164, 2021.

[42] X. S. M. X. J. L. Y. X. Liguo Weng, “Portfolio trading system of digital currencies: A deep reinforcement learning with multidimensional attention gating mechanism,” *Neurocomputing,* vol. 402, pp. 171-182, 2020.

[43] E. P. Farzan Soleymani, “Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder—DeepBreath,” *Expert Systems with Applications,* vol. 156, 2020.

[44] K. C. P. L. C. Q. P. Z. X. W. Huiting Liu, “REDRL: A review-enhanced deep reinforcement learning model for interactive recommendation,” *Expert Systems with Applications,* 2022.

[45] M. R. Z. Y. G. Y. Y. G. L. C. C. W. Xiangfei Liu, “A multi-step predictive deep reinforcement learning algorithm for HVAC control systems in smart buildings,” *Energy,* vol. 259, 2022.

[46] Z. N. C. C. Yuting An, “Smart control of window and air cleaner for mitigating indoor PM2.5 with reduced energy consumption based on deep reinforcement learning,” *Building and Environment,* vol. 224, 2022.

[47] F. Z. H. L. Xudong Zhu, “Swarm Deep Reinforcement Learning for Robotic Manipulation,” *Procedia Computer Science,* vol. 198, pp. 472-479, 2022.

[48] Z. H. Y. Z. Qixin Wang, “Learn to swim: Online motion control of an underactuated robotic eel based on deep reinforcement learning,” *Biomimetic Intelligence and Robotics,* vol. 2, nº 4, 2022.

[49] K. K. D. S. A. G. I. A. D. W. M. R. Volodymyr Mnih, “Playing Atari with Deep Reinforcement Learning,” em *NIPS Deep Learning Workshop* , 2013.

[50] J. J. H. A. P. N. H. T. E. Y. T. D. S. D. W. Timothy P. Lillicrap, “Continuous control with deep reinforcement learning,” em *arXiv*, 2015.

[51] H. v. H. D. M. Scott Fujimoto, “Addressing Function Approximation Error in Actor-Critic Methods,” em *ICML* , 2018.

[52] A. Z. P. A. S. L. Tuomas Haarnoja, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” 2018.

[53] F. W. P. D. A. R. O. K. John Schulman, “Proximal Policy Optimization Algorithms,” *CoRR,* 2017.

[54] J. &. K.-N. Z. &. K. D. &. T. D. &. S. H. &. L. J. &. H. D. &. B. M. Wang, “Prefrontal cortex as a meta-reinforcement learning system,” *Nature Neuroscience,* 2018.

[55] H. Markowitz, “Portfolio Selection,” *The Journal of Finance,* vol. 7, nº 1, pp. 77-91, 1952.

[56] “Yahoo Finance,” [Online]. Available: https://finance.yahoo.com/.

[57] “Coin Gecko,” [Online]. Available: https://www.coingecko.com/pt.

[58] “Binance,” [Online]. Available: https://www.binance.com/en/fee/schedule.

[59] “Python,” [Online]. Available: www.python.org.

[60] “OpenAI Gym,” [Online]. Available: https://github.com/openai/gym.

[61] “stable-baselines3,” [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/.

[62] “TensorFlow,” [Online]. Available: www.tensorflow.org.

[63] “readthedocs,” [Online]. Available: https://technical-analysis-library-in-python.readthedocs.io/en/latest/.

[64] “readthedocs,” [Online]. Available: https://pyportfolioopt.readthedocs.io/en/latest/.

[65] A. P. B. M. M. A. G. T. P. L. T. H. D. S. K. K. Volodymyr Mnih, “Asynchronous Methods for Deep Reinforcement Learning,” 2016.