T H E R E P U B L I C O F T U R K E Y

B A H C E S E H I R U N I V E R S I T Y

TRADING USING REINFORCEMENT LEARNING

Master of Science Project

GÖKÇE SARIÇİYİL

İSTANBUL, JANUARY 2023

T H E R E P U B L I C O F T U R K E Y
B A H C E S E H I R U N I V E R S I T Y

GRADUATE SCHOOL

TRADING WITH REINFORCEMENT LEARNING

MASTER'S PROGRAM IN ARTIFICIAL INTELLIGENCE

ARTIFICIAL INTELLIGENCE MASTER PROJECT

Gökçe SARIÇİYİL

Project Supervisor: Assist. Prof. Tarkan AYDIN

İSTANBUL, JANUARY 2023

ABSTRACT

TRADING USING REINFORCEMENT LEARNING

Sarıçiyil, Gökçe

Master's Program in Artificial Intelligence

Supervisor: Assist. Prof. Tarkan Aydın

January 2023, 40 pages

Automated Financial Trading Systems is a research topic of interest to both the academic and financial community due to the increasing power of real computers and their potential for self-learning. Many technical and academic studies are carried out in the field of artificial intelligence to perform more effective operations in these systems. Reinforcement Learning, a subcomponent of artificial intelligence, is also widely used in these studies. In this article, we have benefited from the A2C (Advantage Actor Critic) algorithm, which is the Reinforcement Learning (RL) Policy Gradient Method, and the PPO (Proximal Policy Optimization) algorithm. We compared these two algorithms, which optimize their behaviour based on the responses they receive, without the need for a supervisor.

Keywords: Reinforcement Learning, Trading, Policy Gradient, Advantage Actor Critic, Proximal Policy Optimization.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
DL	Deep Learning
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
A2C	Synchronous Actor Critic
A3C	Asynchronous Actor Critic
PPO	Proximal Policy Optimization
SGD	Stochastic Gradient Descent
SAC	Soft Actor Critic
DDPG	Deep Deterministic Policy Gradient
DPG	Deterministic Policy Gradient
DQN	Deep Q Network
TD3	Twin Delayed Deep Deterministic Policy Gradient
TD	Temporal Difference
ReLU	Rectified Linear Unit

1. INTRODUCTION

The financial industry is an area where it is aimed to manage resources in the most effective way to make a profit. In the stock market, with the opportunity to realize profit maximization, investors were trading stocks according to their own methods and perceptions. The reason for the low yield in this form of trading is because of the irrational actions of the investors. Quantitative trading, which is carried out automatically with algorithms beyond the typical commercial understanding, by revolutionizing traditional methods with the rapidly developing technology of our age, is profit mining made by making use of transactions and behaviours that have been made in the past. In addition, quantitative trading strategies adapt to the ever-evolving stock market. Among the methods used in quantitative trading, the purpose of reinforcement learning methods is to maximize a portion of the cumulative reward (Wu, et al., 2020). In addition, with new developments in Reinforcement learning, it can use large amounts of data and improve decisions in complex economic environments (Singh, Chen, Singhania, Nanavati, & Gupta, 2022).

In this paper, we compare the performance of the Proximal Policy Optimization strategy, which combines 3 deep reinforcement learning algorithms, against the Advantage Actor Critic strategy. Using these two different deep reinforcement algorithms, we find the optimal one on the trading environment. In Figure 1, we visualized our deep reinforcement learning approach.

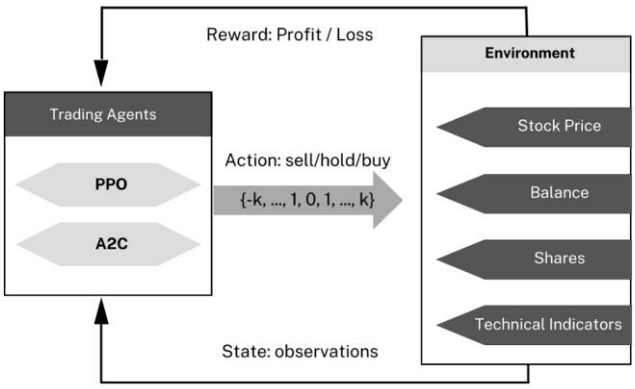

Figure 1. Reinforcement Learning-based Trading Structure

As we mentioned before, we will use Actor Critic and Proximal Policy Optimization algorithms, which are policy gradient methods. First, we aimed to explain the basics and elements of reinforcement learning, then we will summarize the deep reinforcement learning method because the algorithms we use for benchmarking (A2C & PPO) are deep reinforcement learning algorithms that arise from the combination of deep learning and reinforcement learning methods. In addition, since the policy gradient method is under the sub-title of policy-based methods, we will also touch on policy-based methods. We will also touch on other policy gradient algorithms, Deep Deterministic Policy Gradient (DDPG), Twin Delayed Deep Deterministic Policy Gradient (TD3), Soft Actor Critic (SAC) and Asynchronous Actor Critic (A3C). In final section we explained the methods and conclusions. The relationship between AI, ML, RL, DL and DRL is as shown in the figure2 (Ji, Alfarraj, & Tolba, 2020).

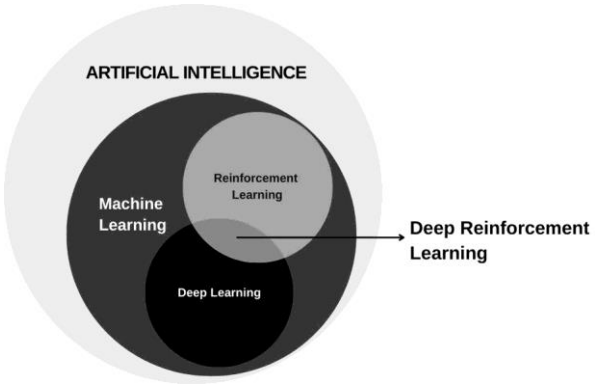

Figure 2. The Relationship Between AI, ML, RL, DL and DRL. (Ji, Alfarraj, & Tolba, 2020)

2. REINFORCEMENT LEARNING

This chapter summarizes the relevant fundamentals for further understanding of the topic of the project. In chapter 2, basic concepts of Reinforcement Learning are presented.

Machine learning, a subfield of artificial intelligence, generally has a complex classification of algorithms grouped into three main categories. These are Supervised Learning, Unsupervised Learning, and Reinforcement Learning methods. Reinforcement Learning is different from Supervised Learning and Unsupervised-Learning methods, which are Machine Learning techniques.

In the Supervised Learning method, a mapping is provided between the labelled dataset given as input and the output, and it is based on new inputs predicting the outputs based on this mapping (Cunningham, Matthieu, & Delany, 2008).

Unsupervised Learning is a learning method used to discover similar sample groups known as clustering within a set of data given as input, or to determine the data distributions of the data in the sample space (Greene, Pádraig , & Mayer, 2008). Reinforcement learning is a reward-based learning mechanism. Rather than telling what actions to take-in a given environment, an agent is expected to discover each action himself and determine how to achieve the highest efficiency in the environment where it collects digital reward signals. In the most interesting and challenging situations, actions can affect not only the instant reward, but also the next state and thus all subsequent rewards. These two features (trial and error research and delayed reward) are the two most important distinguishing features of reinforcement learning (Sutton & Barto, 2018).

In reinforcement learning, the learning tool that interacts with time and environment encounters the decision-making problem. The agent must use what they have already experienced to obtain rewards but must also explore to make better action choices in

the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing the mission. The agent should try various actions and gradually choose the ones that look best. This brings up exploration/exploitation trade-off. (Sutton & Barto, 2018). Markov Decision Processes (MDP), defined as the decision-making process in Reinforcement Learning, are controlled stochastic processes. This process is defined by 5 bundles (S, A, T, p, r), -S: state space, A: set of possible actions, T: set of time steps, p: state transition probability function, r: reward function (Sigaud & Buffet, 2013).

While applying the reinforcement learning method, four main sub-elements of the learning system of the RL agent, a policy, a reward signal, a value function, and optionally, an environment model are defined. A policy is a mapping of the actions that an RL agent should take in its environment. A reward signal represents the reward that the reinforcement learning agent collects per action taken while performing the goal of maximizing total reward. The reward signal is the key element for the policy update. If the policy followed promises a low reward, the policy is updated to maximize the reward. (Sutton & Barto, 2018).

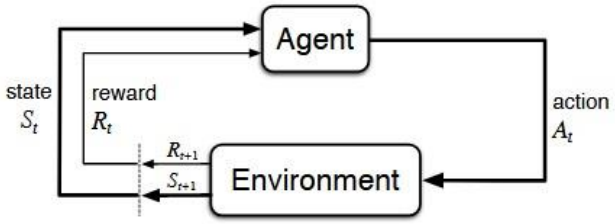

Figure 3. The Agent-environment Interaction in a Markov Decision Process (Sutton & Barto, 2018).

In reinforcement learning, the agent learns optimal behaviour by maximizing future reward by maintaining a balance between exploration (possibilities) and exploitation (experiences). This cumulative future reward (known as the value / Q function) is calculated using an approximate dynamic programming formulation that starts with an initial estimate when not known in advance. RL methods are divided into value-based methods and policy-based methods according to the value function estimation and the way the policy is derived. Value-based methods are constructs represented by

Q (s, a) (value function) and find the optimal value function by iteratively calculating and updating the state-action value function. In simple situation and small cases, value iteration is ideal, but when the data encoding the value function is found in too many dimensions, it significantly impedes the efficiency of value iteration. Value-based methods aim to estimate the value or Q function using the greedy approach. The Q function is usually represented as a table of values for each state-action pair in discrete state and action spaces. As states and actions increase in size or range, it causes the problem of dimensionality curse for discrete spaces, which leads to exponential computation time in the Q-learning algorithm, and one way to solve this situation is to use a deep neural network. The application of deep learning technique to solve RL problems has led to a field called deep reinforcement learning (Kumar, 2021).

3. DEEP REINFORCEMENT LEARNING

Chapter 3 covers the basics of Deep Reinforcement Learning. Deep Learning in Section 3.1 and Section 3.2. Policy Gradient Methods are also presented. Section 3.2.1 Proximal Policy Optimization (PPO) Advantage Actor Critic (A2C) in section 3.2.2, Synchronous Actor Critic (A2C) in section 3.2.3, Asynchronous Actor Critic (A3C) in section 3.2.4, section 3.2. 5 Soft Actor Critic (SAC), section 3.2.6 Deep Deterministic Policy Gradient (DDPG), section 3.2.7 Twin Delayed DDPG (TD3) deep learning methods are explained.

Deep Reinforcement Learning consists of a combination of Deep Learning and Reinforcement Learning methods. We mentioned that the value function: $v(s; \theta)$ or $q(s, a; \theta)$, policy: $\pi(a|s; \theta)$ and model are reinforcement learning components. When we use deep neural networks to approximate any of these components, we get the deep reinforcement learning method (Li, 2017). Research in the field of DRL has shown that this structure is successful in a wide variety of complex decision-making tasks.

3.1. Deep Learning

Deep learning, which is the opposite of shallow learning, has one or more hidden layers between the input and output layers. Each neuron in these layers

applies the weighted sum from the previous layer and applies one of the activation functions to the result. The most used activation functions are Rectified Linear Unit (ReLU), Softmax, Sigmoid and Hyperbolic Tangent (tanh).

ReLU is the activation function that is most frequently used in deep learning. ReLU returns 0 if the input is negative and the entered value if it is positive, for nonlinearity it consists of two linear segments. Since a function will not be linear when the slope is not constant, the ReLU function is non-linear around 0, and the slope is always 0 for negative inputs and 1 for positive inputs. This function is continuous and not differentiable for negative inputs. A feature that helps with gradient descent is that the ReLU output is not saturated, so it has no maximum value.

The Softmax function, another activation function, converts the inputs from 0 to 1 in order to calculate them as probabilities, even when they are negative, positive, or greater than 1. Depending on whether the inputs are large or small, this activation function transforms these inputs into linearly proportional large and small probabilities, and these converted values are always between 0 and 1.

The sigmoid function and its derivative are simple and reduce the time required to make the model, but the short range of the derivative has the problem of information loss. The more multi-layered or deep the neural networks are, the more information loss is experienced, leading to a disadvantageous situation. It is generally preferred to be used in the last layer. It has the disappearing or exploding gradient problem.

Tanh function also has the disappearing or exploding gradient problem. It is more convenient than Sigmoid Function and Tanh is 0-centered unlike Sigmoid.

After one of these activation functions is applied, neurons in the next layer separate the classes with a curve, which allows the hidden layers to understand hierarchical features.

A neural network is formed by combining many artificial neurons. The mathematical model of an artificial neuron is as shown in the figure3.

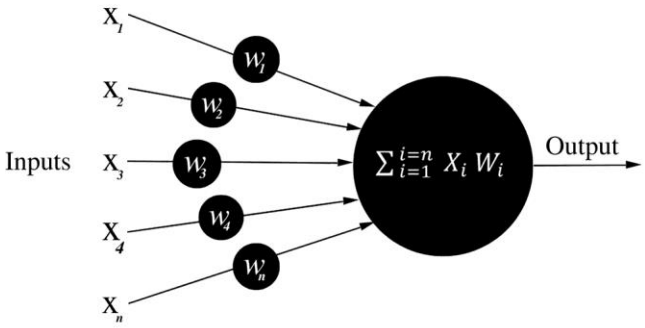

Figure 4. Model of Artificial Neuron

The input vector X is X = {x_1, x_2, …, x_n} and these inputs are either raw inputs or scaled inputs and are normalized if these inputs are part of a very large network. The W weight vector is represented as W = {w_1, w_2, …, w_n} and the inputs are weighed with this weight vector. To reach the best output, it is aimed to produce the output that produces the least error/loss (Sewak, 2019).

3.2. Policy Gradient Algorithms.

The policy gradient, which belongs to the class of policy search techniques, is a reinforcement learning approach that aims to directly model and optimize the control policy by gradient descent. Policy gradient methods are parameterized policy learning methods that can select actions without consulting a value function, instead of taking actions with action-value methods as others do. It is not required for action selection but can use a value function to learn the policy parameter. Usually, modelling is based on θ, $\pi\theta$ (a | s) and the reward function depends on this policy. Depending on this policy, various algorithms can be applied to optimize the θ. All methods that follow this general scheme are policy gradient methods. Methods that simultaneously learn approaches to policy and value functions are generally known as actor-critical methods (Sutton & Barto, 2018).

In policy gradient,

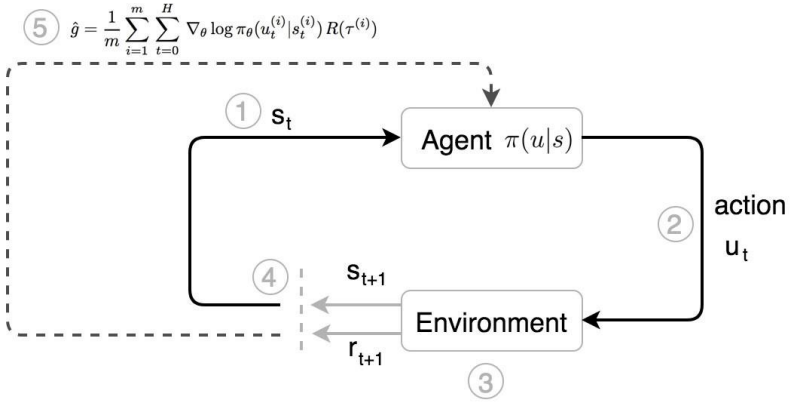

Figure 3. In Policy Gradient, Agent-environment Relationship *(Hui, 2018)*.

- When the agent is on the state-s, it decides an action(u) according to its instinct (a policy π).
- The agent moves and a new state is formed.
- Depending on the situation he observes, the agent takes other actions.
- The agent adjusts his instincts according to the total reward ***R(t)***, this happens after a ***t*** trajectory.

The probability of performing the action u given in a policy π in RL can be expressed mathematically as:

$$\boldsymbol{\pi(u|s)}$$

Formula 1. Probability of performing the action u given in a policy (Hui, 2018)..

The expected rewards are equal to the sum of the probability of the rewards corresponding to a trajectory:

$$\boldsymbol{J(\theta) = E\,[\sum_{t=0}^{H} R(s_t, u_t);\ \pi_\theta\,] = \sum_{\tau} P(\tau;\theta)R(\tau)}$$

Formula 2. Expected rewards (Hui, 2018).

Our aim is to find a policy θ that creates a trajectory $\boldsymbol{\tau}$ with the aim of maximizing expected rewards:

$$\boldsymbol{max_\theta J(\theta) = max_\theta \sum_\tau P(\tau;\theta)\, R(\tau)}$$

Formula 3. Maximizing expected rewards (Hui, 2018).

3.2.1. Proximal Policy Optimization. Proximal Policy Optimization is a deep reinforcement learning algorithm that uses the Deep-Q learning method developed by the OpenAI team and introduced in 2017. The PPO algorithm is called the RL method on-policy. It is in the act of directly optimizing the policies of the agent by applying a stochastic gradient rise to the policy parameters. Deep Neural Network (DNN) is used to facilitate action and state mapping. In the Proximal Policy Optimization Algorithm, any training data is used once and then discarded, so it requires moderate main memory for inference (Schulman, Chen, & Abbeel, 2017). Proximal Policy Optimization does not change sequential policies much, causing less variance in training, but contributes to smooth training by preventing the agent from taking an irreversible path because of meaningless actions. PPO was developed to avoid potentially catastrophic changes to policy while making as much policy improvement as possible to make training effective (Nahhas, Kharitonov, & Turowski, 2022).

PPO-Penalty and PPO-Clip are the two main variants of PPO (OpenAI, 2020). PPO-Penalty penalizes KL deviation without making it a harsh constraint and automatically adjusts the penalty coefficient throughout the training (OpenAI, 2020).

There is no PPO-Clip restriction and no Kl deviation. It uses special breaking in the objective function to remove incentives for the new policy to diverge from the old policy (OpenAI, 2020).

We will apply the PPO-Clip approach in our project.

$$\theta_{k+1} = \arg max_\theta \, E_{s,a\sim\pi_{\theta_k}} [L(s,a,\theta_k,\theta)]$$

Formula 4. PPO-clip updates policies (OpenAI, 2020).

PPO-clip takes multiple stochastic gradient descent steps by updating policies via delta and aims to maximize the target. Here L tells how far the policy is allowed to go from the old policy.

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s,a)\right)$$

Formula 4. PPO-clip updates policies (OpenAI, 2020).

The algorithm is summarized below as pseudocode:

Algorithm PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters $\emptyset_0$

for k = 0,1,2, … **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running

policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards to go $\widehat{R_t}$.

Compute advantage estimates, $\widehat{A_t}$ (using any method of advantage estimation) based on the current value function $V_{\emptyset_k}$.

Update the policy by maximizing the PPO-Clip objective:

$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|\, s_t)}{\pi_{\theta_k}(a_t|\, s_t)} A^{\pi_{\theta_k}}(s_t,a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t,a_t))\right)$,

typically, via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$\emptyset_{k+1} = \arg\min_\emptyset \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \left(V_\emptyset(s_t) - \widehat{R_t}\right)$,

typically, via some gradient descent algorithm.

end for

3.2.2. Actor-Critic. As we mentioned earlier, value-based methods try to find the optimal value function that maps between value and action to reach higher value. Policy-based methods try to optimize the policy. Actor-critic methods have emerged by combining policy-based and value-based methods and aim to benefit from the good sides of both methods (Karagiannakos, 2018). In more detail, the Q-learning algorithm family is known to be cutting edge among value-based learning approaches, and the only good solution among policy-gradient-based reinforcement

learning approaches is the Reinforce with baseline method. The actor critical method was created by combining these two approaches (Sewak, 2019).

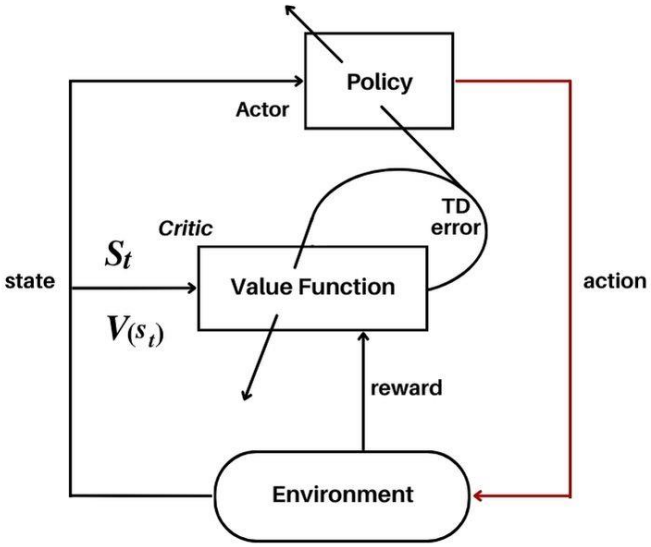

Figure 4. Actor Critic Method *(Güldenring, 2019)*.

Critic measures how well the action taken was(value-based), the actor controls how the agent behaves (policy-based) (Simonini, 2018), as the name suggests is to take an action. Since in the context of reinforcement learning a policy is needed to take any action, in the case of actor-critic the actor implements a policy to take an action. The critic provides feedback on the good and bad of the action taken. The rewards the actor collects by interacting with the environment are received by the critic to improve the next step, and the critic corrects and updates the actor's estimates. In addition to the instant reward in the face of the changing environment, the actor also sends the new state reached as a result of the action to the critic. The actor has a (stochastic) policy at each step of the training, and his policy is constantly updated during training and uses (state) value estimates from the critic to update this policy. The critic's estimates are the basis for the actor to update his policy using the policy-gradient approach. The reinforce-with-baseline algorithm doesn't update itself at every iteration like in Q-Learning, it uses state value estimates, but the state value used is not preloaded which means it can't update itself on every refresh. It is essential that the critic's value estimates are updated at each iteration, so the reinforce-with-baseline algorithm cannot be called a critic. The error between the predictions of the next state values, with the discount factor γ, is calculated using the instant reward and discounted state value of the state. For V is the state-value

estimator function, for $\boldsymbol{W_t}$ is weight/parameter vector at the t.th step, the equation corresponds to one-step return updates (Sewak, 2019).

$$\boldsymbol{\delta_t = R_t + \gamma V(S_{t+1}; W_t) - V(S_t; W_t)}$$

Formula 4. One-step return updates (Sewak, 2019).

The value estimator function parameterized by the weight vector $\boldsymbol{W}$ updates from $\boldsymbol{W_t}$ to $\boldsymbol{W_{t+1}}$ at each iteration, as well as updates the value estimator function.

$$\boldsymbol{W_{t+1} = W_t + \alpha_w \delta_w \Delta_w V(S; W_t)}$$

Formula 5. Weight vector update (Sewak, 2019).

The actor's policy estimator function is also updated with each iteration.

$$\boldsymbol{\theta_{t+1} = \theta_t + \alpha_\theta \delta_t \Delta_\theta log\pi(A|S; \theta_t)}$$

Formula 6. Actor's policy estimator function update (Sewak, 2019).

3.2.3. Advantage Actor-Critic Model (A2C). There are two main variants of Advantage Actor Critic: Asynchronous Advantage Actor Critic (A3C) and Advantage Actor Critic (A2C).

The Asynchronous Advantage Actor-Critic Method has been proposed by the Google DeepMind team (Mnih V. , et al., 2016). As the basic working principle of the method, while the advantage function used updates the critic, the actor also shares some parameters with the critic. The Advantage function provides us with a basis for evaluating the quality of action-state. The critic approaches the advantage function and regulates the actor, accordingly, thus providing stability by reducing the variance of the policy gradient. In this algorithm, the authors also found that adding the policy entropy to the actor's loss function prevents the agent from converging to non-optimal solutions. Many worker threads copy the parameters of the actor-critical network held by a main thread to their local memory and after executing a segment, they calculate the gradients of the network and send them to the parameter server

(Rastogi, 2017). So, at any time, it is possible that they have out-of-date local parameters.

Finding that the asynchronous update does not contribute to rapid convergence or stability, and that the resulting noise is of no appreciable benefit, OpenAI researchers have introduced a synchronous version of A3C called A2C. The biggest disadvantage is that some agents of A3C use outdated parameters, so we have a version of A2C that waits for all agents to finish their segments and then update their global network weights and reset all agents (Kyriakides & Margaritis, 2018) (Kyriakides & Margaritis, 2018).

The algorithm is summarized below as pseudocode:

Algorithm Advantage Actor-Critic

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 $t = t_{start} == \; t_{max}$

$R = \begin{cases} 0 & for\ terminal\ s_t \\ V(s_t, \theta_v) & for\ non-terminal\ s_t\ //Bootstrap\ from\ last\ state \end{cases}$

for $i \in \{t-1, \dots, \; t_{start}\}$

$R \leftarrow r_i + \gamma R$

Accumulate gradients wrt θ: $d\theta \leftarrow \; d\theta + \; \nabla_\theta log\pi(a_i|s_i;\theta)\left(R - V(s_t;\theta_v)\right) +$ $\beta_e \partial H\left(\pi(a_i|s_i;\theta)\right)/\,\partial\theta_v)$

Accumulate gradients wrt θ_v: $d\theta_v \leftarrow \; d\theta_v + \; \beta_v\left(R - V(s_i;\theta_v)\right) \; \partial V(s_i;\theta_v)/\,\partial\theta_v)$

end for

Perform update of θ using $d\theta$ and of θ_v using $d\theta_v$.

$E \leftarrow E + 1$

until $E > E_{max}$

3.2.4. Asynchronous Actor-Critic Method(A3C). A3C (Asynchronous Advantage Actor-Critic) is a reinforcement learning algorithm that uses multiple independent agents to learn and adapt to complex environments. These agents

interact with different copies of the environment in parallel, allowing for efficient exploration of the state-action space. A3C uses a self-weighted network, which means that each agent's contribution to the overall learning process is weighted according to its performance. This allows the algorithm to quickly adapt to changes in the environment and find optimal solutions more efficiently than traditional reinforcement learning algorithms. The most obvious difference between A2C (Advantage Actor-Critic) and A3C (Asynchronous Advantage Actor-Critic) is that A3C is asynchronous, while A2C is synchronous. This means that in A3C, each agent can learn and make decisions independently without being tied to the same timeline as the other agents. This allows for faster and more efficient exploration of the state-action space, as the agents can explore different parts of the environment simultaneously. In contrast, in A2C, all agents must learn and make decisions in lockstep, which can slow down the overall learning process. This can be particularly problematic in complex environments where a large number of agents are required to explore the state-action space effectively.

Agents trained in parallel periodically update the global network holding the parameters, and the updates do not occur simultaneously, so they are called asynchronous. Agents reset their parameters to that of the global network after each update and continue their independent exploration and training until they are updated again. Information is transferred from agents to the global network and also to other agents, so that all agents have knowledge of each other.

The algorithm is summarized below as pseudocode (Mnih V. , et al., 2016):

Algorithm Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^{-}, and counter $T = 0$,

Initialize thread step counter $t \leftarrow 0$

Initialize target network weights $\theta^{-} \leftarrow \theta$

Initialize network gradients $d\theta \leftarrow 0$

Get initial state s

Repeat

Take action a with ϵ −greedy policy based on $Q(s, a; \theta)$

Receive new state s' and reward r

$$y = \begin{cases} r & \text{for terminal s}' \\ r + \gamma max_{a'} Q(s', a'; \theta^{-}) & \text{for non} - \text{terminal s}' \end{cases}$$

Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$

$s = s'$

$T \leftarrow T + 1$ and t$\leftarrow t + 1$

if $T\ mod\ I_{target} == 0$ **then**

Update the target network $\theta^{-} \leftarrow \theta$

end if

if $t\ mod\ I_{AsyncUpdate} == 0$ or s is terminal **then**

Perform asynchronous update of θ using $d\theta$.

Clear gradients $d\theta \leftarrow 0$.

end if

until $T > T_{max}$

3.2.5. Soft Actor-Critic Model (SAC). Soft Actor Critic (SAC) is an off-policy actor-critic deep RL algorithm, mainly based on a maximum entropy reinforcement learning framework, jointly developed by UC Berkely and Google. SAC method combines the strengths of two popular algorithms, namely the actor-critic and soft Q-learning methods. In the actor-critic method, the "actor" is responsible for deciding what actions to take in a given environment, while the "critic" evaluates the actor's actions and provides feedback to help improve future decision making. The soft Q-learning method, on the other hand, uses a "soft" version of the traditional Q-function, which allows for the optimization of the expected long-term return of an agent's actions, instead of just the immediate reward.At the same time, this allows the algorithm to find solutions that balance exploration and exploitation, and to learn policies that are both efficient and robust. This makes SAC a powerful tool for solving a wide range of reinforcement learning tasks. In this algorithm, the actor aims to maximize the expected reward and entropy, thus increasing entropy provides further exploration that can accelerate learning. Including the clipped double-Q trick, this algorithm also benefits from something like target policy smoothing. It is considered one of the most efficient RL algorithms to be applied in the field of robotics. (Haarnoja, et al., 2018).

The algorithm is summarized below as pseudocode:

Algorithm Soft Actor-Critic

Initialize parameter vectors $\psi, \bar{\psi}, \theta, \phi$.

for each iteration **do**

for each environment step **do**

$a_t \sim \pi_\phi(a_t|s_t)$

$s_{t+1} \sim p(s_{t+1}|s_t, a_t)$

$\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$

end for

for each gradient step **do**

$\psi \leftarrow \psi - \lambda_V \widehat{\nabla_\psi} J_V(\psi)$

$\theta_i \leftarrow \theta_i - \lambda_Q \widehat{\nabla_{\theta_i}} J_Q(\theta_i)$ for $i \in \{1,2\}$

$\phi \leftarrow \phi - \lambda_\pi \widehat{\nabla_\phi} J_\pi(\phi)$

$\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$

end for

end for

3.2.6. Deep Deterministic Policy Gradient (DDPG). It is a non-policy actor-critic algorithm that combines a deep deterministic policy gradient algorithm, Deterministic Policy Gradient (DPG) and Deep Q Learning (DQN) approach, designed for use in environments with continuous action. DPG learns a deterministic rather than stochastic policy, and DQN continually expands it into the field of action. The deterministic definition in the name of this algorithm, which produces the exact action instead of the probability distribution on the actions, expresses the calculation of the action directly. It uses two different policies for discovery and updates. In DDPG, the actor-critical algorithm, the TD error from the critic is used during the update process of the actor. The critical network is updated according to the TD error, this rule is similar to the Q-learning update rule (Siver, et al., 2014).

The algorithm is summarized below as pseudocode:

Algorithm 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 N for action exploration

Receive initial observation state s1

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 at 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_t, a_t, r_t, s_{t+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:

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

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

3.2.7. Twin Delayed DDPG (TD3). The successor to the Deep Deterministic Policy Gradient (DDPG), TD3 focuses on solving one of the disadvantages of the DDPG algorithm, the overestimation bias (Lillicrap, et al., 2020). Due to the critical network constantly overestimating the Q values, DDPG can be heavily dependent on finding the correct hyperparameters. Accumulation of forecast errors over time can cause the tool to drop to a local minimum or experience forgetfulness. The TD3 algorithm uses a dual critical mesh to overcome this problem, it also uses a delayed update of the actor mesh, updating only in two-time steps instead of each time step, preferring higher values by adding clipped noise to the selected action.

The algorithm is summarized below as pseudocode:

Algorithm TD3

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ

Initialize target networks $\theta_1' \leftarrow \theta_1, \quad \theta_2' \leftarrow \theta_2, \quad \phi' \leftarrow \phi$

Initialize replay buffer $\mathcal{B}$

for $t = 1, T$ **do**

Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$,

observe reward r and new state s'

Store transition tuple (s, a, r, s') in $\mathcal{B}$

Sample mini batch of $\mathcal{N}$ transition (s, a, r, s') from $\mathcal{B}$

$\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$

$y \leftarrow r + \gamma min_{i=1,2}\, Q_{\theta_i'}(s', \tilde{a})$

Update critics $\theta_i \leftarrow argmin_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$

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:

$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$

$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$

end if

end for

---

4. METHODOLOGIES

In this research, we compared the performance of two different deep reinforcement learning algorithms, A2C and PPO, on the financial stock trading system. Before going into details, if we explain why we focus on these two algorithms (A2C, PPO), which are both policy gradient algorithms, directly improve the policy. These two policy gradient algorithms update the weights of the network to increase good overall reward and reduce bad outcomes.

Google Stock Dataset was used from August 2004 to March 2022 to compare PPO and A2C algorithms in financial stock trading system. NumPy was used for working with arrays, pandas for data analysis, TensorFlow, stable-baseline3 for A2C and PPO algorithms and gym – gym-anytrading collection for environment and matplotlib for plotting the data results.

QuanStats and Tensorboard library were used for in-depth analytics and risk measurement and performance analysis.

FinTa (Financial Technical Analysis) library, which supports more than 80 trading indicators, was used in the project to add SMA (Simple Moving Averages), RSI (Relative strength index), OBV (Equilibrium Trading Volume) on our dataset.

NumPy is a Python library used for computing in Python, providing multidimensional array objects, various routines on arrays, including mathematical, logical, shape manipulation, sorting, selection, I/O, discrete. It is also used in Fourier transforms, linear algebra, statistical operations, random simulation, and more. The purpose of using NumPy in this project is to increase the array dimensions by adding new axes in the test environment (Developers, n.d.).

Pandas is a popular open-source library in the Python programming language for working with structured data. It is designed to provide powerful, easy-to-use data structures and data analysis tools for working with tabular data, such as spreadsheets or datasets in CSV format. Pandas provides data structures such as the DataFrame

which is similar to a spreadsheet or SQL table, and tools for working with these structures, such as indexing, filtering, and aggregation.

One of the key benefits of pandas is its ability to handle large datasets efficiently. It provides fast, flexible, and expressive data structures that make it easy to work with complex data, and offers a variety of built-in functions for cleaning, transforming, and analysing data. Additionally, pandas integrate seamlessly with other popular Python libraries for data analysis, such as NumPy and matplotlib, which makes it a popular choice for data scientists and analysts working with large datasets in Python.

TensorFlow is a multidimensional array stream in a computational graph used in deep learning models, using data flow graphs that can be shared and executed on many different platforms. It is an open-source code library provided by Google.

Stable Baselines is a set of implementations of reinforcement learning algorithms based on an OpenAI Baselines. It is intended to be used in the project to implement Deep Reinforcement Learning algorithms.

AnyTrading, a Reinforced learning-based collection of OpenAI Gym environments, was used in the project. This collection, which provides convenience for trading algorithms applied in Forex and Stock markets, aims to improve, and facilitate the implementation of Reinforcement Learning based algorithms. For its purpose, it provides three different Gym environments named TradingEnv, ForexEnv and StocksEnv. These environments are intangible and support any trading environment. In this environment, sell=0 and buy=1 transactions are also sufficient to train an agent. (Haghpanah, 2020).

4.1. Dataset

Stock data from the American multinational technology company Google LLC was used.

The stock market data used in the study were downloaded from the web address ww.kaggle.com as a data file with .csv extension. The daily data in the obtained data are ordered from the oldest to the newest date. Stock data from Google LLC, an American multinational technology company specializing in Internet-related services and products, including software and hardware, was used. These data were downloaded as daily data between August 2004 and March 2022. The parameter list

in the downloaded file consists of columns Date, Open, High, Low, Close, Adj Close and Volume. The Date column shows the date information of the stock. The Open column means the starting price of the stock. The High column shows the highest price of the stock during the day. The Low column shows the lowest price of the stock during the day. The Close column shows the closing price of the stock at the end of the day. Adj Close is a value of the closing price announced at the end of the day for a stock, reflecting all adjustments and significant events applicable for that day. The Volume column is the sum of the trade (buy and sell) (Öztoprak & Orman, 2022).

An example portion of the dataset is as follows:

Table 1. Google LLC Stock Market Data

	Date	Open	High	Low	Close	Adj Close	Volume
0	19/08/2004	50.05005	52.08208	48.02803	50.22022	50.22022	50.22022
1	20/08/2004	50.55556	54.5946	50.3003	54.20921	54.20921	54.20921
2	23/08/2004	55.43043	56.7968	54.57958	54.75475	54.75475	54.75475
3	24/08/2004	55.67567	55.85586	51.83684	52.48749	52.48749	52.48749
4	25/08/2004	52.53253	54.05405	51.99199	53.05306	53.05306	53.05306
5	26/08/2004	52.52753	54.02903	52.38238	54.00901	54.00901	54.00901
6	27/08/2004	54.1041	54.36437	52.8979	53.12813	53.12813	53.12813
7	19/08/2004	50.05005	52.08208	48.02803	50.22022	50.22022	50.22022

After sorting the data from the oldest to the newest and specifying the date column as the index, we created the stock environment using the OpenAI-gym environment. As step two, we build the environment.

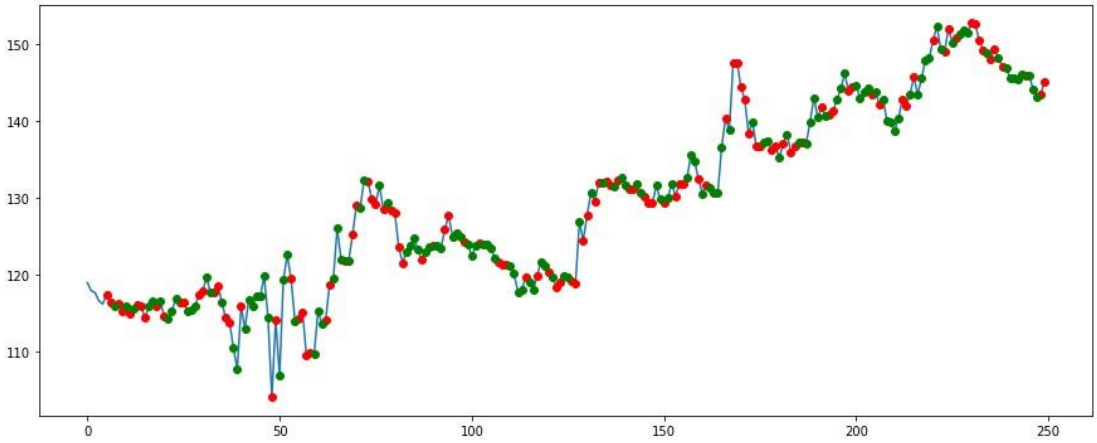

Figure 5. Google LLC Stock Data Environment

We used the Finta library to add Simple Moving Averages (SMA), Equilibrium Trading Volume (OBV), Relative strength index (RSI) indicators on stock data.

4.2. Quantitative Comparison

After a total training period of 1,000,000 steps for PPO and A2C algorithms, the results were logged, and the average values were calculated. We also shared the Cumulative Returns for each test phase of the PPO model in detail in below. For the PPO model, the total reward and profit information returned at each test step are also shared as a table.

***Table 2*. PPO Model Cumulative Returns**

	Cumulative Returns %					
Year	**Test Run 1**	**Test Run 2**	**Test Run 3**	**Test Run 4**	**Test Run 5**	**Average**
2004	30.31	39.55	39.13	68.64	34.66	42.458
2005	37.73	20.83	6.8	17.41	36.45	23.844
2006	-16.38	-28.32	-25.19	-36.08	-22.75	-25,744
2007	-17.07	-27.63	-5.39	-6.89	13.46	-8.704
2008	-76.6	-73.39	-75.39	-56.05	-68.64	-70.014
2009	6.3	13.33	12.96	8.94	-7.94	6.718
2010	-25.36	-43.8	-20.99	-42.05	-28.01	-32.042
2011	-32.92	-48.81	-37.57	-28.73	-42.65	-38.136
2012	-34.57	-24.65	-18.79	-26.86	-30.66	-27.106
2013	-7.94	-25.09	-12.63	-16.69	-19.39	-16.348
2014	-25.95	-36.86	-39.39	-25.17	-43.22	-34.118
2015	-28.32	-26.98	-13.82	-25.24	-24.51	-23.774
2016	-43.75	-39.91	-31.6	-26.29	-34.3	-35.17
2017	-23.84	-24.23	-28.37	-30.3	-16.24	-24.596

2018	-43.5	-35.3	-36.9	-31.95	-42.31	-37.992
2019	-24.7	-20.88	-13.93	-20.96	-27.23	-21.54
2020	-36.45	-2.28	-10.12	-37.45	-28.63	-22.986
2021	-15.36	-3.42	-4.03	-21.47	-12.12	-22.986
2022	-19.78	-9.24	-9.34	-12.12	-5.32	-11.16

Performance metrics for test one data as an example of other test steps are shared in figure 6.

Figure 6. PPO Performance Metrics

Also, for all test information for PPO Model is as follows:

Table 3. PPO Model Test Phase Rewards

Testing	Total Reward	Total Profit	Position

1	1697.0140109999993	0.00389713157702084	1
2	2247.913564	0.003943372393268917	1
3	2731.964072	0.011121355792610328	1
4	2404.5710809999996	0.009471938624154115	1
5	2153.617059999999	0.0063090153438074585	1

In a few test runs, different batch_size parameter values were tried in the PPO model and the best result was added. For the aim of the project, the parameters of the PPO model have not been analysed in detail, but it is aimed to analyse the detailed analysis study in a different project according to the parameter performances. Since the PPO model is a later learning algorithm than the A2C model, the learning_rate and bach_size parameters should be evaluated with different values to maximize the reward, especially in line with the data and target used.

The analyses for the A2C model are as follows.

Cumulative Returns for each test phase of the A2C model in detail in below. Performance metrics for test one data as an example of other test steps are shown in figure 8.

Table 4. A2C Model Cumulative Returns

	Cumulative Returns %					
Year	**Test Run 1**	**Test Run 2**	**Test Run 3**	**Test Run 4**	**Test Run 5**	**Average**
2004	51.7	43.98	29.79	40.74	45.25	42.292
2005	-41.42	-30.02	-29.48	-10.52	-29.71	-28.23
2006	-68.65	-59.94	-50.35	-50.18	-36.94	-53.21
2007	-40.46	-46.65	-51.63	-59.19	-41.59	-47.904
2008	-65.78	-72.83	-76.43	-61.23	-67.8	-68.814
2009	-32.85	-27.26	-34.47	-32.1	-21.21	-29.578
2010	-52.69	-64.17	-45.96	-41.51	-57.74	-52.414
2011	-51.87	-40.13	-31.22	-35.52	-40.06	-39.76
2012	-58.08	-49.16	-46.53	-53.82	-42.58	-50.034
2013	-37.3	-28.39	-46.64	-33.5	-28.61	-34.888
2014	-49.37	-48.28	-38.16	-56.71	-44.78	-47.46
2015	-22.23	-22.85	-13.91	-17.53	-38.53	-23.01
2016	-50.04	-51.22	-51.37	-39.42	-43.79	-47.168
2017	-29.84	-39.56	-35.39	-39.16	-40.51	-36.892
2018	-54.11	-47.39	-59.27	-41.99	-53.3	-51.212
2019	-41.39	-37.62	-17.52	-27.19	-29.18	-30.58
2020	-36.25	-43.73	-38.45	-29.85	-34.38	-36.532
2021	-10.39	-12.98	-30.62	-18.56	-19.28	-18.366
2022	-22.05	-20.31	-28.84	-11.21	-19.81	-20.444

Performance metrics for test one data as an example of other test steps are shared in figure 7.

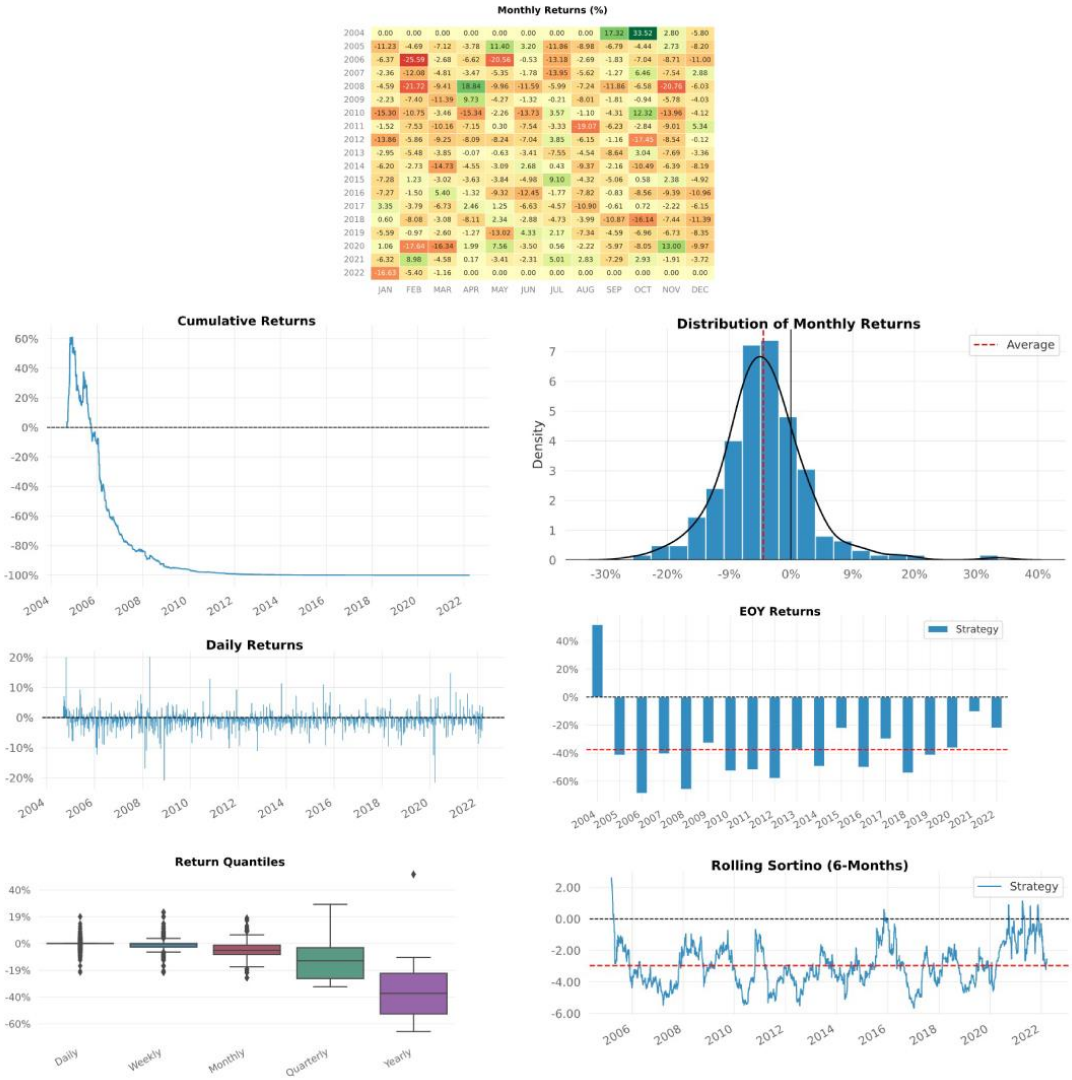

*Figure 7.*A2C Performance Metrics

Also, for all test information for A2C Model is as follows:

Table 5. A2C Model Test Phase Step Reward

Testing	Total Reward	Total Profit	Position
1	2360.9313119999993	3.77162201427628e-05	0
2	2034.2908469999993	5.00553571804741e-05	0
3	2042.537521000001	6.148270991402652e-05	0

4	2410.1356190000006	0.00022010779800390258	1
5	2035.066068000002	0.00015795801341918285	1

As a result of the analysis, the following graphs were obtained by using the analysis data in the comparison of the two DRL algorithms.

The average and comparison of the awards received by the models for a series of steps are listed in chart 7. As we have obtained from the analysis and the inferences made from the graphics, the PPO model shows a higher performance than the A2C model.

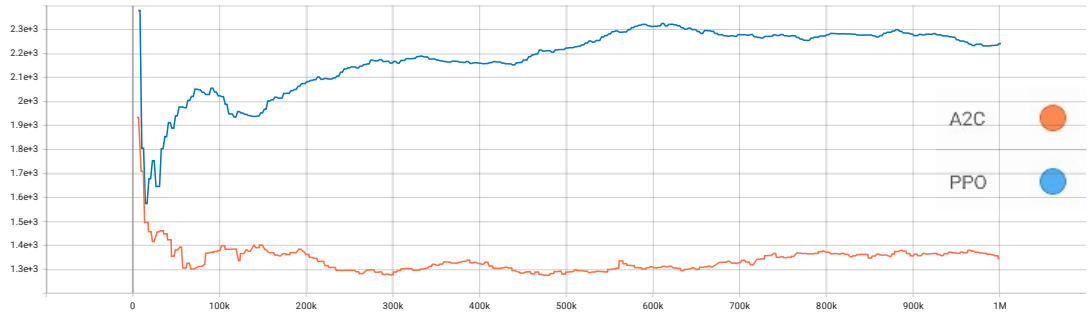

Figure 8. A2C and PPO Rollout Episode Reward Mean

The Time/FPS graph, which is the time and frame rate used by the models for a series of steps, is as follows.

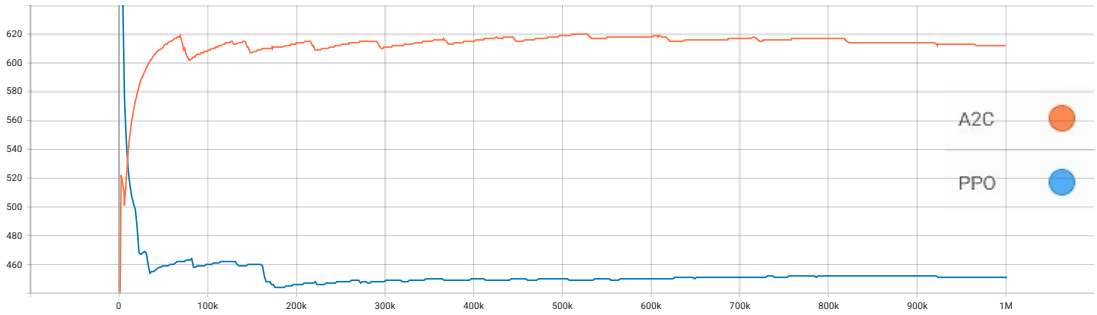

Figure 9. A2C and PPO Time/FPS Graph

Train Entropy Loss values for A2C and PPO models are as follows.

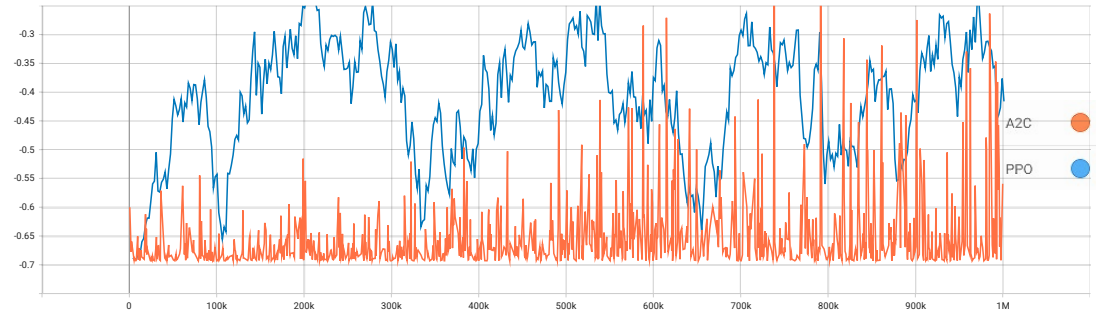

Figure 10. A2C and PPO Train Entropy Loss

Train Explained Variance values, which is a learning criterion used by PPO and A2C Models during training, are as follows.

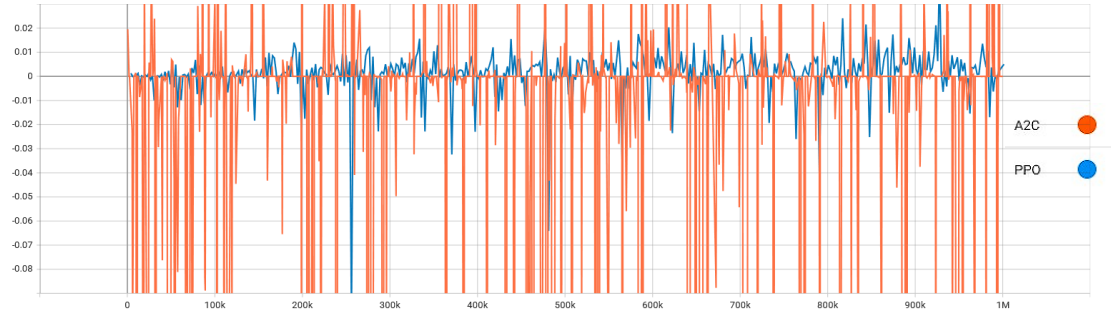

Figure 11. A2C and PPO Train Explained Variance

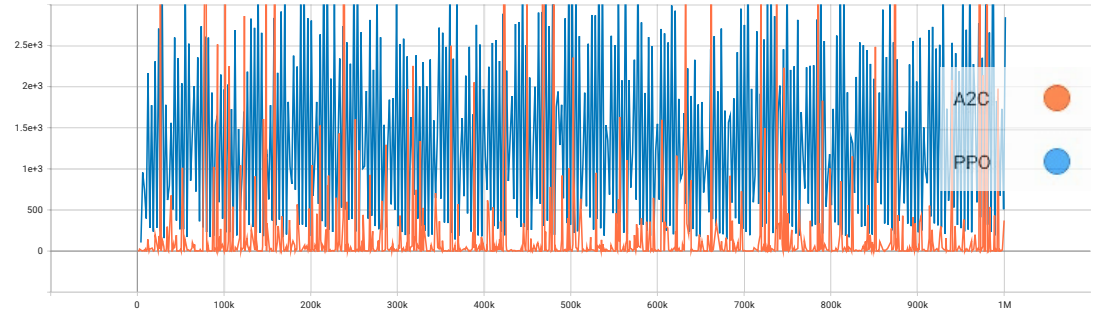

Figure 12. A2C & PPO Train Value Loss

The final graphic analysis of the comparison of the two models, the Train Value Loss values for the A2C and PPO models, are as shown in the graphic above.

In the OpenAI stock environment we created, two different models trained using two deep reinforcement learning algorithms, A2C and PPO were tested, the results

obtained during the test stages were reported in detail, and the total benefit calculation was made using these reports. According to the observed test results, the PPO model showed higher performance than A2C model. Results have obtained using Google LLC stock data between 2004-2022 and OpenAI stock environment.

5. CONCLUSION

In this article, we analysed the applicability and performance of Reinforcement Learning methods in Financial Trading Systems. We applied the PPO and A2C models, which are the Deep Reinforcement Learning model, to the financial system and compared the performance metrics.

When comparing the performances of the applied models, it was observed that the PPO model was higher than the A2C model. Considering that these performance values may change with the detailed application of various parameters, we carried out detailed performance analysis and application of our models to the stock environment within the scope of the project's purpose. While it is observed that the parameters of these two models can affect the model performances to a certain extent, it is possible to experiment with more various parameters for future studies and analyses.

REFERENCES

Carapuço, J., Neves, R., & Horta, N. (2018). Reinforcement learning applied to Forex trading. *Applied Soft Computing*, 783-794.

Cobbe, K. W., Hilton, J., Klimov, O., & Schulman, J. (2021). Phasic policy gradient. *International Conference on Machine Learning* (s. 2020--2027). PMLR.

Cunningham, P., Matthieu, C., & Delany, S. (2008). *Machine learning techniques for multimedia.* Berlin, Heidelberg.: Springer.

Developers, N. (tarih yok). *NumPy documentation*. https://numpy.org/: https://numpy.org/doc/stable/ adresinden alındı

Fan, J., Wang, Z., Xie, Y., & Yang, Z. (2020). A theoretical analysis of deep Q-learning. J. Fan, Z. Wang, Y. Xie, & Z. Yang içinde, *Learning for Dynamics and Control* (s. 486--489). PMLR.

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. S. Fujimoto, H. Hoof, & D. Meger içinde, *International conference on machine learning* (s. 1587-1596). PMLR.

Gao, Z., Gao, Y., Hu, Y., Jiang, Z., & Su, J. (2020). Application of deep q-network in portfolio management. *2020 5th IEEE International Conference on Big Data Analytics (ICBDA)* (s. 268--275). IEEE.

Greene, D., Pádraig , C., & Mayer, R. (2008). *Unsupervised learning and clustering.* Berlin, Heidelberg: Springer.

Güldenring, R. (2019). *Applying Deep Reinforcement Learning in the Navigation of Mobile Robots in Static and Dynamic Environments.* Hamburg: informatik.uni-hamburg.de.

Haarnoja, T., Pong, V., Hartikaine, K., Zhou, A., Dalal, M., & Levine, S. (2018, Dec 14). *Soft Actor Critic—Deep Reinforcement Learning with Real-World Robots*. https://bair.berkeley.edu/: https://bair.berkeley.edu/blog/2018/12/14/sac/ adresinden alındı

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. T.

Haarnoja, A. Zhou, & P. Abbeel içinde, *International conference on machine learning* (s. 1861--1870). PMLR.

Haghpanah, M. A. (2020, July 31). *gym-anytrading*. github.com: https://github.com/AminHP/gym-anytrading adresinden alındı

Hui, J. (2018, September 12). *Rl-policy gradient explained*. Medium: https://jonathan-hui.medium.com/rl-policy-gradients-explained-9b13b688b146 adresinden alındı

Ji, H., Alfarraj, O., & Tolba, A. (2020). *Artificial intelligence-empowered edge of vehicles: architecture, enabling technologies, and applications.* IEEE.

Karagiannakos, S. (2018). *The idea behind Actor-Critics and how A2C and A3C improve them.* Sensors.

Kumar, S. (2021). *Controlling an Inverted Pendulum with Policy Gradient Methods - A.* arXiv.

Kyriakides, G., & Margaritis, K. G. (2018). Neural Architecture Search with Synchronous Advantage. *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*, 1--7.

Li, Y. (2017). *Deep reinforcement learning: An overview.* arXiv preprint arXiv:1701.07274.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N. M., Erez, T., Tassa, Y., . . . Wierstra, D. P. (2020). *Continuous control with deep reinforcement learning.* Google Patents.

Liu, G., Li, X., Sun, M., & Li, P. (2020). An advantage actor-critic algorithm with confidence exploration for open information extraction. G. Liu, X. Li, M. Sun, & P. Li içinde, *Proceedings of the 2020 SIAM International Conference on Data Mining* (s. 217-225). SIAM.

McFarlane, R. (2018). *A survey of exploration strategies in reinforcement learning.* McGill University. Montreal, Canada: McGill University.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., . . . Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement

Learning. *International conference on machine learning* (s. 1928--1937). içinde PMLR.

Mnih, V., Badia, P. A., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., . . . Kavukcuoglu, K. (2016). *Asynchronous Methods for Deep Reinforcement Learning.* Montreal: University of Montreal.

Mnih, V., Kavukcuoglu, K., Siver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning.* and Graves, Alex and Antonoglou, Ioannis and Wierstra, Daan and Riedmiller, Martin: arXiv preprint arXiv:1312.5602.

Nahhas, A., Kharitonov, A., & Turowski, K. (2022). Deep Reinforcement Learning Techniques for Solving Hybrid Flow Shop Scheduling Problems: Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C). *Proceedings of the 55th Hawaii International Conference on System Sciences.* (s. 1666-1675). hawaii.edu.

OpenAI. (2020, January 30). *Proximal Policy Optimization*. spinningup.openai.com: https://spinningup.openai.com/en/latest/algorithms/ppo.html#id6 adresinden alındı

Öztoprak, S., & Orman, Z. (2022). Finansal Verilere İlişkin Tahminleri Açıklamaya Yönelik Yeni Bir Model-Agnostik Yöntem ve Uygulaması. *Avrupa Bilim ve Teknoloji Dergisi*, 32-39.

Peters, J., & Schaal, S. (2006). Policy gradient methods for robotics. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (s. 2219--2225). içinde IEEE.

Rastogi, D. (2017). *Deep Reinforcement Learning for Bipedal Robots.* Holland: Delf University of Technology.

Schulman, J., Chen, X., & Abbeel, P. (2017). *Equivalence between policy gradients and soft q-learning.* arXiv preprint arXiv:1704.06440.

Sewak, M. (2019). *Deep Reinforcement Learning Frontiers of Artificial Intelligence.* Singapore: Springer.

Sigaud, O., & Buffet, O. (2013). *Markov Decision Processes in Artificial Intelligence.* John Wiley \& Sons.

Simonini, T. (2018, July 26). *An intro to Advantage Actor Critic methods: let's play Sonic the Hedgehog!* freecodecamp: https://www. freecodecamp. org/news/an-intro-toadvantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d adresinden alındı

Singh, V., Chen, S.-S., Singhania, M., Nanavati, B., & Gupta, A. (2022). How are reinforcement learning and deep learning algorithms used for big data based decision making in financial industries–A review and research agenda. *International Journal of Information Management Data Insights*.

Siver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). *Deterministic Policy Gradient Algorithms.* London: PMLR.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction. MIT press*. Cambridge, Massachusetts: The MIT Press.

Weber, C., Elshaw, M., & Mayer, N. M. (2008). *Reinforcement Learning.* BoD--Books on Demand.

Weng, L. (2018). *Policy Gradient Algorithms*. lilianweng.github.io: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/ adresinden alındı

Wu, X., Chen, H., Wang, J., Troiano, L., Loia, V., & Fujita, H. (2020). *Adaptive stock trading strategies with deep reinforcement learning methods.* IJCNN.

Yang, H., Liu, X.-Y., Zhong, S., & Walid, A. (2020). Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy. *Proceedings of the First ACM International Conference on AI in Finance* (s. 1-8). içinde

Yang, H., Zhong, S., Liu, X.-Y., & Walid, A. (2020). Deep Reinforcement Learning for Automated Stock Trading. *Proceedings of the First ACM International Conference on AI in Finance*, (s. 1--8).

Yoon, C. (2019, February 6). *Understanding actor critic methods and a2c.* towardsdatascience: https://towardsdatascience. com/understanding-actor-critic-methods-931b97b6df3f adresinden alındı

Zhou, K., Wang, W., Hu, T., & Deng, K. (2021). Application of Improved Asynchronous Advantage Actor Critic Reinforcement Learning Model on Anomaly Detection. *Entropy*, 274.