IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 July 2022, accepted 23 August 2022, date of publication 5 September 2022, date of current version 12 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3203697

RESEARCH ARTICLE

Deep Reinforcement Learning Approach for Trading Automation in the Stock Market

TAYLAN KABBANI AND EKREM DUMAN
Department of Industrial Engineering, Özyeğin University, 34794 Istanbul, Turkey
Corresponding author: Taylan Kabbani (taylankabbani96@gmail.com)

ABSTRACT Deep Reinforcement Learning (DRL) algorithms can scale to previously intractable problems. The automation of profit generation in the stock market is possible using DRL, by combining the financial assets price ''prediction'' step and the ''allocation'' step of the portfolio in one unified process to produce fully autonomous systems capable of interacting with their environment to make optimal decisions through trial and error. This work represents a DRL model to generate profitable trades in the stock market, effectively overcoming the limitations of supervised learning approaches. We formulate the trading problem as a Partially Observed Markov Decision Process (POMDP) model, considering the constraints imposed by the stock market, such as liquidity and transaction costs. We then solve the formulated POMDP problem using the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm reporting a 2.68 Sharpe Ratio on unseen data set (test data). From the point of view of stock market forecasting and the intelligent decision-making mechanism, this paper demonstrates the superiority of DRL in financial markets over other types of machine learning and proves its credibility and advantages in strategic decision-making.

INDEX TERMS Autonomous agent, deep reinforcement learning, MDP, sentiment analysis, stock market, technical indicators, twin delayed deep deterministic policy gradient.

I. INTRODUCTION

The prime objective of any investor when investing in any financial market is to minimize the risk involved in the trading process and maximize the profits generated. Investors can meet this objective by successfully predicting the prices or trends of the market assets and optimally allocating the capital among the selected assets. This process is very challenging for a human to consider all relevant factors in a complex and dynamic environment; therefore, the design of adaptive automated trading systems capable of meeting the investor's objective and bringing more stagnant wealth into the global market has been an intensive research topic. Many efforts have been made to design such trading systems in the past decade. The majority of these efforts focused on using *Supervised learning* (SL) techniques [1], [2], [3], [4], [9], which in essence train a predictive model (e.g., Neural Network, Random Forest,...) on historical data to forecast the trend direction of the market. Regardless of their popularity, these techniques suffered from various limitations, leading to sub-optimal results [5]. One primary limitation with SL is that it only considers one aspect of the trading problem, the prediction of the future price, and neglects the other important aspect which is the control problem of optimally diversifying the capital to be invested among the selected assets in the portfolio, this is a well-known problem known as the Portfolio Optimization Problem (POP) or Portfolio Selection Problem. Another major drawback of the methods that rely on SL is that they seek minimization of prediction error regardless of the risk involved, also exogenous constraints by the market environment (e.g., lack of liquidity, transaction cost) are not being considered at all in most cases.

Reinforcement Learning (RL) offers to solve the drawbacks of Supervised Learning approaches in trading financial markets by combining the financial assets price ''prediction'' step and the ''allocation'' step of the portfolio in one unified process to optimize the objective of the investor, where the trading agent (the algorithm) interacts with the environment (the model) to take the optimal decision [6]. In addition, financial data is highly time-dependent (function

The associate editor coordinating the review of this manuscript and approving it for publication was Emre Koyuncu.

of time), making it a perfect fit for Markov Decision Processes (MDP) [7], which is the core process of solving RL problems. MDP captures the entire past data and defines the whole history of the problem in just the agent's current state, and that's highly crucial when it comes to modeling financial market data [8].

Most works that studied the RL's applications in financial markets and particularly in trading stocks, considered discrete action spaces [9], [10], [11], [12], i.e., buy, hold, and sell a fixed number of shares to trade a single asset. In this work, a continuous action space approach is adopted to give the trading agent the ability to gradually adjust the portfolio's positions with each time step (dynamically re-allocate investments), resulting in better agent-environment interaction and faster convergence of the learning process. In addition, the approach supports the managing of a portfolio with several assets instead of a single one. We first propose a novel formulation of the stock trading problem or what is referred to as the trading *Environment* as a Partially Observed Markov Decision Process (POMDP) model considering the constraints imposed by the stock market, such as liquidity and transaction costs. More specifically, we design an environment that simulates the real-world trading process by augmenting the state (observation) representation with ten different technical indicators and sentiment analysis scores of news releases along with other state components. We then solve the formulated POMDP problem using the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, which can learn policies in high-dimensional and continuous action spaces like those typically found in the stock market environment. Finally, we evaluate our proposed approach by performing back-testing, which is the process used by traders and analysts to assert the viability of a trading strategy by testing it on historical data.

The key contribution of this work is the solution it provides to combine prediction and decision-making processes to fully automate the trading procedure in the stock market by utilizing DRL, which is a particular advantage over supervised learning. Compared with other studies investigating the RL applications in financial markets, this paper has the advantage of using technical indicators and news sentiments in the state representation and the use of the Twin Delayed DDPG algorithm for the first time to solve the trading problem which is capable of handling continuous action space.

II. BACKGROUND AND RELATED WORK

A. MDP IN REINFORCEMENT LEARNING

In essence, *Markov Decision Processes* [13] (MDP) is used to model stochastic processes containing random variables, transitioning from one state to another depending on certain assumptions and definite probabilistic rules. MDPs are a perfect mathematical framework to describe the reinforcement learning problem. In this framework, researchers call the learner or decision maker the *agent* and the surrounding which the agent interacts with (comprising everything outside the agent) the *environment*. The learning process ensues from the agent-environment interaction in MDP, at each time step $t \in \{1, 2, 3, \ldots, T\}$ the agent receives some representation (information) of its current state from the environment $s_t \in \mathcal{S}$, and on that basis selects an action $a_t \in \mathcal{A}$ to perform. One step later, due to its action, the agent finds itself in a new state, and the environment returns a reward $R_{t+1} \in \mathcal{R}$ to the agent as a feedback of its action's quality [14].

B. THE OBJECTIVE OF REINFORCEMENT LEARNING

The *objective* of any RL problem is to maximize the cumulative reward $\mathbb{G}_t$ it receives in the long run instead of the immediate reward R_t

$$\mathsf{E}[\mathbb{G}_t] = \mathsf{E}[R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T] \tag{1}$$

In the above reward equation (Eq. 1), the term R_T denotes the reward received at the terminal state T. meaning that the aforementioned equation is only valid when the problem at hand is an *Episodic task*, i.e., ends in a terminal state T. For the *Continuous tasks* i.e., no terminal state, $T = \infty$, a discount factor gamma is introduced to Eq. 1 ($0 \leq \gamma \leq 1$):

$$\begin{aligned}\mathbb{G}_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{k-1} R_{t+k} + \cdots \\ &= \sum_{0}^{\infty} \gamma^k R_{t+k+1}\end{aligned} \tag{2}$$

C. BELLMAN EQUATIONS

Value functions are being used by almost all RL methods to estimate how good (in terms of expected return) it is for the agent to be in a given state or to perform an action in a given state. This evaluation is being made based on the future expected sum of rewards. Accordingly, value functions are determined with respect to the future actions the agent will take. We call a particular way of acting a *Policy* (π) [14] which is a function that maps from the environment's states to probabilities of selecting each possible action.

Bellman equations [15] are the fundamental property of value functions used in *dynamic programming* as well as in reinforcement learning to solve MDPs, and they are essential to understand how many RL algorithms work. Bellman equation states that the value function of state s ($\mathcal{V}_\pi(s)$) can be calculated by finding the sum over all possibilities of expected returns, weighting each by its probability of occurring following a policy π:

$$\mathcal{V}_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma \mathcal{V}_\pi(s')], \ \forall s \in \mathcal{S} \tag{3}$$

In a similar way we define the action-value ($q_\pi(s, a)$) function as:

$$q_\pi(s, a) = \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')] \tag{4}$$

From Bellman equations (Eq. 3 and Eq. 4) we can derive what is called *The Bellman Optimality Equations*. Intuitively,

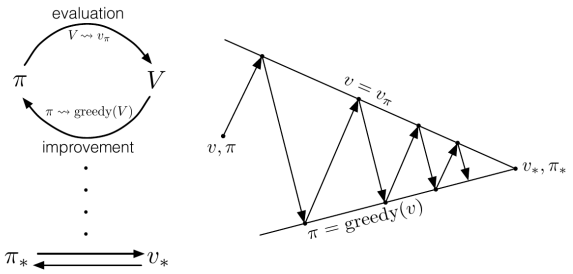

FIGURE 1. Generalized policy iteration [14].

the Bellman optimality equation expresses the fact that the value of a state under an optimal policy (π_*) must equal the expected return for the best action from that state [14], and the optimal state-value function ($\mathcal{V}_*$) equals to :

$$\mathcal{V}_*(s) = \max_a \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma \mathcal{V}_*(s')] \tag{5}$$

Similarly, we define optimal action-value (q_*) function as:

$$\begin{aligned} q_*(s, a) &= \max_\pi q_\pi(s, a) \\ &= \sum_{s'} \sum_r P(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned}$$

D. TAXONOMY OF RL ALGORITHMS

RL algorithms are classified based on how to represent and train the agent into three main approaches:

1) CRITIC-ONLY APPROACH

This family of algorithms learns to estimate the value function (State-value function or action-value function) by using what is called *Generalized Policy Iteration* (GPI). This concept refers to the interaction of two steps. The first step is the *policy-evaluation*. The main goal of this step is to collect information (value functions) under the given policy to determine how good it is. The second step is the *policy-improvement*. It is responsible of improving the policy by choosing greedy actions with respect to the value functions computed from the policy-evaluation step. The two steps alternate in a consecutive manner until the value functions and policies stabilize, which means that the process has reached an optimal policy, as illustrated in Fig. 1.

We distinguish between two different ways the agent learns the value function of the system. The first way is *Tabular Solution Method* where the value functions are represented as arrays or tables and updated with more accurate values after each iteration as the agent collects more experience. This way of learning often finds exact solutions. However, it does not generalize well, and the state and action spaces must be small enough to be stored in tables.

The second possible way in the critic-only approach is called *Approximate Solution Method*, which tends to generalize better than the Tabular Method but has lower discrimination, and it is capable of learning the value function of systems with enormous state and action spaces. Approximate methods achieve this generalization by combining RL with *supervised learning* algorithms. Deep Reinforcement Learning is considered an approximate method that combines Neural Networks with RL. *Mnih et al. [16]* is considered the father of DRL, where he trained an agent of Deep Q-network (DQN) to play Atari games, where pixels of the game screen were the input data (state), and the directions of the joystick were actions. He proved that DRL had outperformed all existing algorithms in 2015 [17].

2) ACTOR-ONLY APPROACH

All methods under the Critic-Only approach rely on the GPI framework to learn approximate action values to infer a good policy. *Actor-Only methods* (also called *Policy Gradient Methods*) estimate the gradient of the objective, which is maximizing rewards with respect to the policy parameters and adjust the policy parameters θ based on the estimate (Eq. 6). The parameterized policy function takes state and action as an input and returns the probability of taking that action in that state instead of taking the state only as an input and returning the value function as Critic-Only methods do. Note that in the below equation G_t represents the expected reward at time t.

$$\theta_{t+1} = \theta_t + \alpha \nabla \ln \pi(a_t|s_t, \theta_t) G_t \tag{6}$$

3) ACTOR-CRITIC APPROACH

In the *Actor-Critic approach*, the actor's job is to select actions at each time step to form the policy, whereas the critic's role is to evaluate these actions taken by the actor. So the approach is gradually adjusting the policy parameters θ of the actor to take actions that maximize the total reward predicted by the critic. The TD error (δ) calculated by the critic to evaluate the action is as follows:

$$\delta = R_{t+1} + \gamma \hat{V}(s_{t+1}, w) - \hat{V}(s_t, w) \tag{7}$$

The value function estimation of the current state $\hat{V}(s_t, w)$ is added as a baseline to make the learning faster. The parameter θ of the actor is being adjusted in the way of maximizing the total future reward from Eq. 6 and Eq. 7 we conclude the equation used by the Actor-Critic to update the gradient at each time step t as the following:

$$\begin{aligned} \theta_{t+1} = \theta_t + \alpha \nabla \ln \pi(a_t|s_t, \theta_t)(R_{t+1} \\ + \gamma \hat{V}(s_{t+1}, w) - \hat{V}(s_t, w)) \end{aligned}$$

Many researchers worked on improving the DQN algorithm. Van Hasselt *et al.* [18] proposed to use two networks instead of one Q-network to choose the action and the other to evaluate the action taken to solve the deviation problem in DQN. They called it Double-DQN. Lillicrap *et al.* [19] built on the top of Double-DQN, an algorithm based on the deterministic policy gradient (DDPG) that can operate over continuous action spaces. The Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm which will be discussed in section IV-A, was proposed by Fujimoto *et al.* [20] to tackle the problem of the approximation error in DDPG.

E. RL IN FINANCE

Bertoluzzo and Corazza [21] investigated the performance of different RL algorithms in day-trading for one selected Italian stock. Specifically, they compared the performance of Q-learning, and Kernal-based reinforcement learning, concluding that Q-learning performance outperformed Kernal-based RL. In a subsequent study (2014) [10], they explored the effect of different reward functions such as Sharpe ratio, average log return, and OVER ratio on the performance of Q-learning. By trading six selected Italian stocks, they reported that lagged return reward function has the best performance. Instead of approximating a value function (critic-only), Deng *et al.* [12] made one of the first attempts at combining Deep Learning with Recurrent Reinforcement Learning to directly approximate a policy function. This approach is called "deep recurrent reinforcement learning" (DRRL). In their proposed method, first, the DL part extracts 45 useful features from the market to be used as state representatives in the environment. Secondly, they use a Recurrent Neural Network (RNN) as a trading agent to interact with the deep-generated state features and make decisions. To investigate the potential advantage of Actor-Critic methods in solving the day trading problem, Conegundes and Pereira [22] used Deep Deterministic Policy Gradient (DDPG) algorithm to solve the asset allocation problem. Considering different constraints such as liquidity, latency, slippage, and transaction costs, they back-tested their approach on the Brazilian Stock Exchange datasets. They showed that their approach successfully obtained 311% cumulative return in three years with an annual average maximum drawdown around 19%.

III. PROBLEM DESCRIPTION

The stock trading problem is being modeled as *Partially Observed Markov Decision Process* (POMDP), which can be formulated by describing its State Space, Action Space, and Reward Function. The POMDP model of the problem is called the trading environment, and it's built to carefully mimic the real-world trading process.

A. STATE SPACE

The state-space in the proposed environment is designed to support multiple and single stock trading by representing the state as $(1 + 13 \times \mathcal{N})$-dimensional vector where $\mathcal{N}$ is the number of assets we consider to trade in the market. Hence the state space increases linearly with the number of assets available to be traded.

There are two main parts of the state presentation. The first part is the *Position State* $\in \mathbb{R}_+^{1+\mathcal{N}}$ which holds the current cash balance and shares owned by each asset in the portfolio, and the second part of the state is the *Market Signals* $\in \mathbb{R}^{12\times\mathcal{N}}$, which holds the necessary market features for each asset as a tuple, these features are the required information provided to the agent to make predictions of the market movement. The first type of information is based on the hypothesis of *technical analysis* [23], which states that the future behavior of financial markets is conditioned on its past; hence technical indicators are being used in the state space to help the agent interpret the market behavior. The second type of information is based on *fundamental analysis* [24], which studies everything from the overall economy and industry conditions to news releases. Therefore a Natural Language Processing (NLP) approach is used to measure the general sentiment from the news releases and integrate it with the state representation. The state (observation) vector at each time step is provided to the agent as follows:

$$\mathbf{S_t} = [[\mathbf{b_t}, \mathbf{h_t}], [\{(\mathbf{C_t^i}, \mathbf{SS_t^i}, \mathbf{T_t^i}) | \mathbf{i} \in \mathcal{N}\}]]$$

Each component of the state space is defined as follows:

- $\mathcal{N} \in \mathbb{Z}_+^{\mathcal{N}}$: Number of assets in the portfolio.
- $\mathbf{b_t} \in \mathbb{R}_+$: The available cash balance in the portfolio at time step t.
- $\mathbf{h_t} = \{h_t^i | i \in \mathcal{N}\} = \{h_t^0, h_t^1, \ldots, h_t^{\mathcal{N}}\} \in \mathbb{Z}_+^{\mathcal{N}}$: The number of shares owned for each asset i in $\mathcal{N}$ at time step t.
- $\mathbf{C_t^i} \in \mathbb{R}_+^{\mathcal{N}}$: The close price of asset i in $\mathcal{N}$ at time step t.
- $\mathbf{SS_t^i} \in (-1, 0, 1)$: An integer 1, 0 or -1 to indicate the sentiment of the news related to stock i at time step t.
- $\mathbf{T_t^i}$: The 10 different *Technical Indicators* vector for asset i in the portfolio at time step t using the past prices of the asset in a specified look-back window (most common window is 14 or 9).

To demonstrate the state space, let's assume that we have three different assets ($\mathcal{N} = 3$) in the trading environment and an initial capital of 1000$ to be invested, the state vector would be a 40-dimensional vector and the *initial state*(s_0) given by the environment would be:

$$s_0 = [[1000, 0, 0, 0][(p_0^1, SS_0^1, T_0^1), (p_0^2, SS_0^2, T_0^2), (p_0^3, SS_0^3, T_0^3)]]$$

B. ACTION SPACE

The designed agent in this study receives the state s_t at each time step t as input and sends back action in the range between 1 and -1 inclusive, $a_t \in [-1, 1]$, the action then is re-scaled using a constrain K_{max}, which represents the maximum allocation (buy/sell shares), transforming a_t to an integer $K \in [-K_{max}, \ldots, -1, 0, 1, \ldots, K_{max}]$, which stands for the number of shares to be executed, resulting in decreasing, increasing or holding of the current position of the corresponding asset [25]. There are two important conditions regarding the action execution in our approach:

- If the current capital (cash) in the portfolio is insufficient to execute the buy action, the action will be partially executed with what the current capital can buy of the requested stock.
- If the number of shares for a specific asset (h_t^i) in the portfolio is less than the number of shares to be sold ($a_t^i \in \mathbb{Z}^-$), the agent will sell all the remaining shares of this asset in the portfolio.

We can mathematically express the action space as the following:

$$\begin{aligned} A_t &= \{a_t^i | i \in \mathcal{N}\} = \{a_t^0, a_t^1, \ldots, a_t^{\mathcal{N}}\} \\ S.t. \quad & a_t^i \in \mathbb{Z}^{\mathcal{N}} \\ & -K_{max} \leq a_t^i \leq K_{max}, \ \forall i \in \mathcal{N} \\ & a_t^i = h_t^i \ \ if \ \ |a_t^i| > h_t^i, \ \forall a_t \in \mathbb{Z}^- \end{aligned} \tag{8}$$

where:

* $\mathcal{N}$: assets in the portfolio.
* A_t: the action vector sent by the agent to the environment.
* a_t^i: the action (number of shares) to buy/sell for asset i at time step t.
* K_{max}: the maximum number of shares the agent can re-allocate of an individual asset at each time step t.
* h_t^i: the portfolio position (number of shares) of asset i at time step t.

The action space depends on the number of assets available in the portfolio $\mathcal{N}$ and it's given as $(2 \times K_{max} + 1)^{\mathcal{N}}$; hence the action space increases exponentially by increasing $\mathcal{N}$.

C. REWARD FUNCTION

The difference between the portfolio value $\mathcal{V}_t$ at the end of period t and the value at the end of previous period $t-1$ represents the immediate reward $r(s, a, s')$ received by the agent after each action, and we denote the final investment return at a target time T_f as G.

$$r(s, a, s') = \mathcal{V}_t - \mathcal{V}_{t-1} \tag{9}$$

where the portfolio value $\mathcal{V}$ at each time step is calculated as:

$$\mathcal{V}_t = b_t + h_t.C_t \tag{10}$$

where:

* b_t: the available cash balance in the portfolio at time step t.
* $h_t = \{h_t^i | i \in \mathcal{N}\}$: the position vector (number of shares of each asset) at time step step t.
* $C_t = \{C_t^i | i \in \mathcal{N}\}$: the closing price of each asset in the portfolio at time step t.

The transition cost can be represented in many different ways in real life, and it varies from one broker to another. To better simulate the real-world trading process in the stock market, transaction costs (i.e., commission fees) are incorporated into the immediate reward ($r(s, a, s')$) calculation. In this study, we set the commission as a fixed percentage of the total closed deal cash amount, where d_{buy} represents the commission percentage when buying is performed, and d_{sell} is the commission percentage for selling:

$$\begin{aligned} d_t &= \{d_t^i | i \in \mathcal{N}\} = [d_t^0, d_t^1, \ldots, d_t^{\mathcal{N}}] \\ where: d_t^i &= \begin{cases} d_{buy}, & \text{if } a_t^i > 0 \\ 0, & \text{if } a_t^i = 0 \\ d_{sell}, & \text{if } a_t^i < 0 \end{cases} \end{aligned}$$

The commission vector d_t is incorporated into the immediate reward function by excluding the commission amount paid from the portfolio value calculated in Eq. 10, so the agent would avoid excessive trading that results in a high commission rate and therefore avoids a negative reward:

$$\mathcal{V}_t = b_t + h_t.C_t - h_t.(C_{t-1} \circ d_t) \tag{11}$$

In the above equation, the amount paid for the commission is calculated by taking the Hadamard product of the commission vector d_t and the closing price of the previous period C_{t-1}. That's because the action of buying/selling occurred in the previous state and therefore commission should be calculated using the closing prices on that state.

D. ENVIRONMENT CONSTRAINTS AND ASSUMPTIONS

We impose the following constraints and assumptions on the MDP environment for two main reasons. First, to idealize and simplify the complex financial market systems (e.g., via liquidity assumption) without losing the nature of the problem, and the second reason is to make the model closer to a real-world situation.

1) NON-NEGATIVE BALANCE CONSTRAINT

The cash balance in any state is not allowed to be negative, $b_t > 0$. Therefore, the actions should not result in a negative cash balance. To achieve that, the environment prioritizes the execution of sell actions ($a_t < 0$) in the action vector A_t (Eq. 8) to guarantee cash liquidity in the portfolio, so buy actions ($a_t > 0$) would be fulfilled afterward. If the buy action still results in a negative balance (i.e., not enough cash to fulfill the action), it is fulfilled partially with what remains in the portfolio's cash balance.

2) SHORT-SELLING CONSTRAINT

Short selling is prohibited in the designed environment, all portfolio's positions must be strictly non-negative:

$$\mathbf{h_t} = \{h_t^i | i \in \mathcal{N}\} = \{h_t^0, h_t^1, \ldots, h_t^{\mathcal{N}}\} \in \mathbb{Z}_+^{\mathcal{N}}$$

3) ZERO SLIPPAGE ASSUMPTION

When the market volatility is high, slippage occurs between the price at which the trade was ordered and the price at which it's completed [26]. In this study, the market liquidity is assumed to be high enough to meet the transaction at the same price when it was ordered [27]. This assumption is mostly valid in a real-world trading environment when trading in big stock markets.

4) ZERO MARKET IMPACT

In financial markets, a market participant impacts the market when it buys or sells an asset which causes the price change. The impact provoked by the agent in this study is assumed to have no effect on the market when it performs its actions. This assumption is mostly true even in real-life trading when the market volume is big enough to make the individual investment is insignificant [27].

IV. DETAILS OF IMPLEMENTATION

A. THE TRADING AGENT

Actor-Critic-based algorithms successfully solved the continuous action space by utilizing function approximation and policy gradient methods. One of the most famous actor-critic, off-policy algorithms is the Deep Deterministic Policy Gradient algorithm (DDPG) [28]. Still, despite the excellent performance DDPG achieved in continuous control problems, it has a significant drawback similar to many RL algorithms, which is the overestimation of action values ($\max_a Q(s_{t+1}, a_{t+1})$) as a result of function approximation error. This overestimation bias is unavoidable in RL as we use estimates instead of ground truth in the learning process. In this study, as our problem has a continuous space of actions, we use Twin Delayed Deep Deterministic Policy Gradient (TD3) [20] algorithm, which is a direct successor of DDPG but with improvements to tackle the overestimation problem. TD3 can reduce the overestimation bias, thus reducing the accumulation of errors in the learning process by introducing three main components to DDPG:

1) *Clipped Double Critic Networks*: The first component added is a novel clipped variant of Double Q-learning [18] to replace the single critic. Using two different and separate critic networks to make an independent estimate of the value function can be used to make unbiased estimates of the actions selected using the opposite value estimate. TD3 uses a clipped double Q-learning instead of the traditional one used in Double Q-learning where it takes the smallest value of the two critic networks estimates, that is, if we use the traditional Double Q-learning in actor-critic methods, the policy and target networks are updated so slowly that they make similar estimates and offered slight improvement.
2) *Delayed Updates*: The second component is added to solve the residual error accumulation formed due to the learning process without a fixed target (estimates instead). In Critic-Actor methods, this accumulation of errors is amplified due to the interaction between the policy (actor) and value (critic) networks, where the policy gradient is maximized over the value estimate. Delaying the policy network update, i.e., updating it less frequently than the value network, allows the value network to stabilize before it can be used to update the policy gradient. This results in a lower variance of estimates and, therefore, better policy.
3) *Target Policy Smoothing Regularization:* The final component is applying a regularization strategy to the target policy by adding a small random noise and averaging over mini-batches. This is important to reduce the variance of the target values when updating the critic, which causes by overfitting spikes in the value estimate.

The agent in this paper performs daily trading operations and to aid the agent in understanding its environment (the stock market), we augmented the state representation of ten different technical indicators and news sentiment scores.

Algorithm: Twin Delayed Deep Deterministic Policy Gradient (TD3) [20]

1. **Initialization**

Critic networks $Q(s, a|w_1)$, $Q(s, a|w_2)$ and actor $\pi(s|\theta)$, randomly, with weights W_1, W_2 and θ.
Target networks Q'_1, Q'_2 and π' with weights $W'_1 \longleftarrow W_1, W'_2 \longleftarrow W_2, \theta' \longleftarrow \theta$
Replay buffer $\mathcal{D}$

2. **foreach** *t=1 to T* **do**

Initialize a random process N for action exploration
Select action with exploration noise $a \sim \pi(s|\theta) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$
Observe reward r and next state s'
Store transition tuple (s, a, r, s') in $\mathcal{D}$
Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{D}$
$\tilde{a} \leftarrow \pi(s'|\theta) + \epsilon, \epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
$y \leftarrow r + \gamma \min_{i=1,2} Q(s', \tilde{a}|w_i)$
Update critics $W_i \leftarrow \arg\min_{W_i} N^{-1} \sum (y - Q_{W_i}(s, a))^2$
if *t mode d* **then**
Update θ by the deterministic policy gradient: $\nabla_\theta J(\theta) = N^{-1} \sum \nabla_a Q_{W_1}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)$
Update target networks:
$W'_i \leftarrow \tau W_i + (1 - \tau) W'_i$
$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$

B. TECHNICAL INDICATOR

We used the ten most famous indicators used by technical traders when trading in the stock market [23], we describe them briefly as follows:

1) **Relative Strength Index (RSI)** $\in \mathbb{R}_+^{\mathcal{N}}$: A momentum indicator to measure the magnitude of recent price changes and identify overbought or oversold conditions in the stock price.
2) **Simple Moving Average (SMA)** $\in \mathbb{R}_+^{\mathcal{N}}$: An important indicator to identify current price trends and the potential for a change in an established trend.
3) **Exponential Moving Average (EMA)** $\in \mathbb{R}_+^{\mathcal{N}}$: Like SMA, EMA is a technical indicator used to spot current trends over time. However, EMA is considered an improved version of SMA by giving more weight to the recent prices considering old price history less relevant; therefore it responds more quickly to price changes than SMA.
4) **Stochastic Oscillator (%K)** $\in \mathbb{R}_+^{\mathcal{N}}$: It's a momentum indicator comparing the closing price of the stock to a range of its prices in a look-back window period $\mathcal{W}$.
5) **Moving Average Convergence/Divergence (MACD)** $\in \mathbb{R}^{\mathcal{N}}$: Is one of the most used momentum indicators to

identify the relationship between two moving averages of the stock price and it helps the agent to understand whether the bullish or bearish movement in the price is strengthening or weakening [29].

6) **Accumulation/Distribution Oscillator (A/D)** $\in \mathbb{R}^{\mathcal{N}}$: A volume-based cumulative momentum indicator that helps the agent to assess whether the stock is being accumulated (bought) or distributed (sold) by measuring the divergences between the volume flow and the stock price.
7) **On-Balance Volume Indicator (OBV)** $\in \mathbb{R}^{\mathcal{N}}$: Another volume-based momentum indicator that uses volume flow to predict the changes in stock price [30]:
8) **Price Rate Of Change (ROC)** $\in \mathbb{R}^{\mathcal{N}}$: A momentum-based indicator that measures the speed of stock price changes over the look-back window $\mathcal{W}$.
9) **William's %R** $\in \mathbb{R}_{+}^{\mathcal{N}}$: Known also as Williams Percent Range, is a momentum indicator used to spot entry and exit points in the market by comparing the closing price of the stock to the high-low range of prices in the look-back window ($\mathcal{W}$).
10) **Disparity Index** $\in \mathbb{R}_{+}^{\mathcal{N}}$: Its value is a percentage that indicates the relative position of the current closing price of the stock to a selected moving average. In this study, the selected moving average is the EMA of the look-back window ($\mathcal{W}$).

C. SENTIMENT SCORES

The supply and demand fluctuations in the stock market are highly sensitive to the moment's news due to the impact of mass media on the investor's behavior. Hence many traders and investors consider the news reports in their stock-picking strategy. In our proposed approach, we believe that incorporating the general news sentence toward the asset being considered in the observation (state) definition will help the agent learn a better trading strategy. In Ding *et al.* [31] study, they showed that news headlines are more useful in forecasting than using the entire news article content. Therefore, we only consider news headlines as our input to calculate the sentiment score. We describe the process of calculating a sentiment score for each asset in the portfolio at time step t (day) as the following:

- We use a rule-based matching approach to search for the asset name, stock symbol, or other keywords in the headline news (ex. Microsoft or MSFT, tech,..) released on day t.
- Then we use a fine-tuned BERT model called FinBERT [32] to calculate the sentiment probability (Positive, Negative, or Neutral) of each news headline. FinBERT model is a pre-trained NLP model to analyze sentiments specifically for financial text.
- Finally, we take the average of the asset's news sentiment probabilities for each day and assign 1 if the positive probability is higher than the negative probability and -1 otherwise. We ignore the neutral probability as we believe that if an asset has been mentioned on the news, it will impact the asset price (positively or negatively). If the asset has no news on a given day, we assign 0 to the sentiment score.

V. EXPERIMENTS AND RESULTS

We evaluate our proposed approach by performing two different back-testing, which is the process used by traders and analysts to assess the viability of a trading strategy by testing it on historical data.

We perform two different back-testing experiments. The purpose of the first experiment (Section. V-B) is to validate the superiority of the continuous action space to solve the trading problem by comparing the results of the same experiments reported by Kaur [39]. In their paper, a discrete action space is adapted to solve the problem, where the agent can choose to buy, sell or hold action (i.e., discrete action space) of a fixed number of shares on each time step for a portfolio of two assets, namely; Qualcomm (QCOM) and Microsoft (MSFT). We back-test our approach on the same 5-years daily historical stock data (between 2011-2016) used in their study with the same amount of initial capital ($10,000).

The second experiment (Section. V-C) is conducted to validate the robustness of our model on a large space of actions and states by considering **10** different assets in the portfolio. In addition, considering that the first experiment was on training data set only, we evaluated the performance on an unseen market data (test data set) to check the agent's ability of generalization.

We use two metrics to evaluate our results: the first metric is the *cumulative sum of rewards*, i.e., the total profits at the end of the trading episode. The second metric is the annualized *Sharpe ratio* [38] which combines the return and the risk to give the average of the risk-free return by the portfolio's deviation. In general, a Sharpe ratio above 1.0 is considered to be *"good"* by investors because this suggests that the portfolio is offering excess returns relative to its volatility. A Sharpe ratio higher than 2.0 is rated as *"very good"* whereas a ratio above 3.0 is considered *"excellent"*.

A. DATA DESCRIPTION AND PREPROCESSING

In this work, We use Yahoo Finance [33] to retrieve historical market daily prices. The retrieved historical data consists of 7 columns; *Date, Volume, Open, Close, Adjusted Close, High* and *Low* prices. To prepare each dataset to be used by the model, we first perform timestamps processing by using the trading calendar (exchange-calendars package [34]) to check if the market was open on the given dates to the agent and exclude weekends and holidays from the dataset so the agent will not face gaps in the trading process. Further dataset processing is required to ensure that all financial assets (stocks) considered in the portfolio have an equal length of historical data points. Some stocks have been recorded for decades, while other newly listed stocks are only a few months. This time-dimension alignment of stocks' historical data will prevent the biased action of the agent toward the stock with more data. Once we have the timestamps, processed we use **Close,**

High, Low prices and **Volume** at each timestamp to calculate the technical indicators of each asset with a look-back window ($\mathcal{W}$).

To obtain comprehensive and accurate financial news, we combined headline news from *Benzinga, Seeking Alpha, Zacks* and other financial news websites [35], and crawled historical news headlines from *Reddit worldNews Channel* [36]. The final dataset consists of 3,288,724 news headlines ranging from 2009 to 2021, which we utilized to calculate the sentiment score.

B. FIRST EXPERIMENT

In the first experiment, we conduct three evaluations similar to the benchmark paper [39]. All three evaluations share the same configurations like the number of assets in the portfolio, initial capital, commission rates, etc. but with different components of the environment's state representation. We start with a baseline that only contains the close price as a market signal feature, we then add technical indicators in the second evaluation, and finally, we evaluate by adding sentiment analysis scores. In Table. 1, we summarize the three evaluation results of the experiment.

Due to the stochasticity in the learning process, the experiment results may change at each run depending on different factors such as the actions the agent randomly starts with and uses to explore or the random weight initialization. As suggested in [37], to ensure fairness and reliability of our results, we average multiple runs over different random seeds to have an insight into the population distribution of the algorithm performance in an environment. In this experiment's evaluations, we report and highlight results across several independent runs. While the recommended number of trials to evaluate an RL algorithm is still an open question in the field, we reported the mean and standard error across five trials (runs), which is the suggested number in many studies [37].

1) EVALUATION ON BASELINE ENVIRONMENT

To evaluate the continuous action approach in our model, we test it by solving the problem with only the close price of the assets as a market signal; hence the state representation in this baseline environment consists of only the *position state* and the close price of the asset at t (C_t) as a market signal, i.e., the agent will solely make its trading decisions based on merely the closing price of the stock as a market feature. We perform 5 experiment trials, each with 200 epochs (episodes) for the same hyperparameter configuration, only varying the random seed across trials.

Fig. 2 shows the average return (sum of rewards) at each trading episode and the standard error across the 5 runs. As can be observed, the agent's performance increases with more experience it gains with the number of epochs to successfully achieve 33960$ average return (profits) with a standard error equals to $\pm$4473$. From the commission spent by the agent, we can conclude that the agent was successfully able to find a balanced trading strategy by balancing between trading and holding positions. Finally, the average annual Sharpe ratio of our approach on the baseline environment was 1.43 with a standard error of $\pm$0.13. This is significantly higher than the reported Sharpe ratio 0.85 in [39] benchmark.

2) EVALUATION ON WithTechIndicators ENVIRONMENT

Using the same configurations used in baseline environment evaluation, we augment the state with technical indicators and run 5 independent experiments to report the average return, Sharpe ratio, and commission. We refer to this environment with technical indicators and close price in the state representation as *WithTechIndicators* environment.

The results in Fig. 3 demonstrate that augmenting the environment with technical indicators has brought more helpful information to the agent to make better decisions. The agent successfully achieved 89782$ average return (profits) with $\pm$18980$ standard error, and an average Sharpe ratio equals 2.75 with a standard error $\pm$0.43. We can also notice that the average amount of commission is almost two times the amount spent in the baseline environment, which means that the agent was significantly more active in buying/selling stocks and closed more successful deals. In addition, our approach outperformed the benchmark reported Sharpe ratio of 1.4.

3) EVALUATION ON WithSentiments ENVIRONMENT

We refer to this environment with sentiment analysis scores, technical indicators, and close price in the state representation as *WithSentiments* environment. We include the sentiment scores of news headlines for each asset in the state representation and repeat the experiment with the same configurations. The total average return profits increased to 115591$ with a standard error equals to $\pm$17721 across the five runs. Sharpe ratio increased to 3.14 and $\pm$0.40 standard error. The average amount of commission equals the amount spent in the environment with only technical indicators (WithTechIndicators environment), which means that the agent performed almost the same number of trades but with a better decision (policy). In the benchmark [39] study, they also reported an increase in the agent performance when adding sentiment scores to the state with a Sharpe ratio equal to 2.4. The plot showing the results in Fig. 4 demonstrates that augmenting the state with sentiment analysis along with technical indicators has improved the agent performance.

Experiment's Summary

We notice in all plots of the three evaluations that the policy improves over time, as the agent accumulates more reward, and thus the Sharp ratio increases. Towards the end, the slope is almost flat indicating that the policy has stabilized to the local optimum. As the stock trading problem has never been solved, we do not have a specified reward or Sharpe ratio threshold at which it's considered solved.

C. SECOND EXPERIMENT

In the second experiment, we evaluate our approach on a wider action and state spaces by considering ten assets to

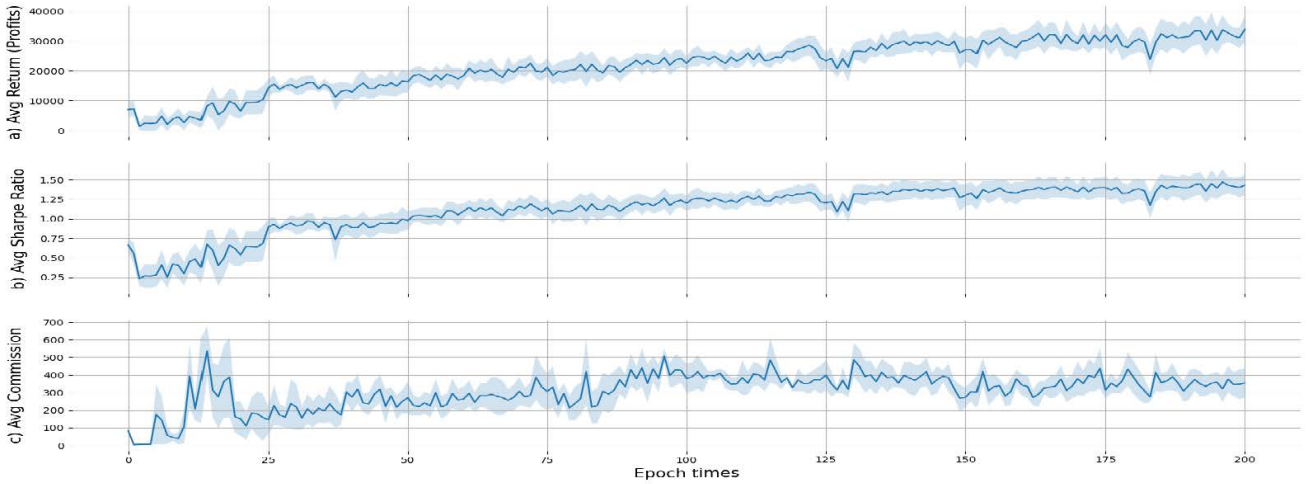

FIGURE 2. TD3 agent performance metrics on Baseline environment using the same hyperparameter configurations averaged over 5 different random seeds. a) Average return (Profits in dollars) at the end of each episode. b) The average annual Sharpe ratio at the end of each episode. c) The average amount of commission spent at the end of each episode.

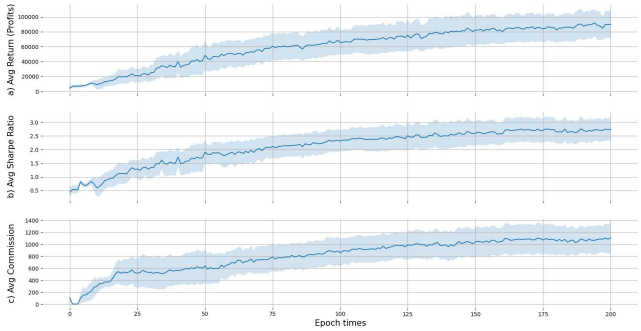

FIGURE 3. TD3 agent performance metrics on WithTechIndicators Environment using the same hyperparameter configurations averaged over 5 different random seeds. a) Average return (Profits in dollars) at the end of each episode. b) The average annual Sharpe ratio at the end of each episode. c) The average amount of commission spent at the end of each episode.

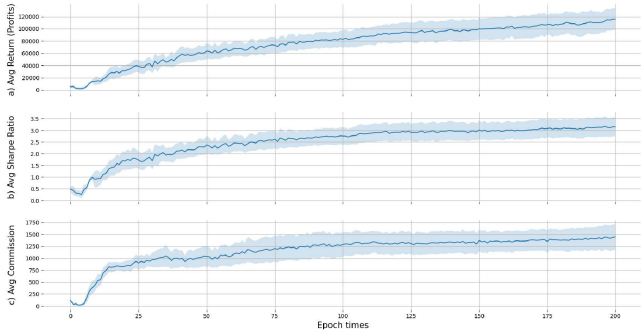

FIGURE 4. TD3 agent performance metrics on WithSentiments Environment using the same hyperparameter configurations averaged over 5 different random seeds. a) Average return (Profits in dollars) at the end of each episode. b) The average annual Sharpe ratio at the end of each episode. c) The average amount of commission spent at the end of each episode.

trade, AAPL, MSFT, QCOM, IBM, RTX, PG, GS, NKE, DIS and AXP.

TABLE 1. The performance evaluation comparison between three different evaluations and benchmark.

Evaluation Environment	Baseline	WithTechIndicators	WithSentiments
Accumulated Return	33960$ ±4473	89782$ ±18980	115591$ ±17721
Sharpe Ratio	1.43 ±0.13	2.75 ±0.43	3.14 ±0.4
Commission	355$ ±83	1109$ ±248	1447$ ±268
Sharpe Ratio benchmark	0.85	1.4	2.4

FIGURE 5. Train, and test data splits.

Our back-testing uses historical daily data from 01/01/2010 to 01/01/2018 with an initial capital of 100000$ for performance evaluation. We split the data set into two periods, the first period is to train the agent, the second is used to test the performance of the agent on unseen data (Fig. 5).

We notice that for our model to generalize better, we had to impose regularization by normalizing the observation space using *Batch Normalization*. This technique uses mini-batches from samples to have unit mean and variance. It maintains a running moving average of the mean and variance to normalize the observation vector during testing. We further normalized the rewards received by the agent as it makes the gradient steeper for better rewards. We also set the look-back window to 20 ($\mathcal{W} = 20$). We added action noise to encourage exploration during training to force the agent to try different actions and explore its environment more effectively, leading to higher rewards and more elegant behaviors.

Our approach successfully archived a 2.68 Sharpe ratio which is considered *"very good"* and 110308$ as total profits

(Rewards) on the test data. We let the agent keep learning on the testing set since this will help the agent better adapt to the market dynamics.

Disabling Sell Action: To investigate whether profits made on the test data (between 2016 and 2018) are a matter of the standard increase in the stocks and the market growth in general (primarily that tech stocks are known for their excellent performance in the past years) or are made due to the decision made by the agent, we disable the sell action and only let the agent buy and hold during the trading episode. As a result, the agent allocated the capital as follows:

Stock	Shares
AAPL	751
AXP	398
MSFT	398
IBM	200
DIS	199

and hold on to this position until the end of the episode. The Sharpe ratio has decreased to 2.00 with 66949$ as total profits (Rewards), which indicates that the decision made by the agent had a positive effect on the return and it was not merely due to the natural growth of the market.

VI. CONCLUSION AND FUTURE WORKS

This work presented a Deep Reinforcement Learning approach that combines technical indicators with sentiment analysis to find an optimal trading policy for assets in the stock market. Results show that the addition of technical indicators and sentiment scores of the news headlines to the state representation has significantly improved the agent's performance and the superiority of using a continuous action space over a discrete one to solve the trading problem. We also explored the potential of using an Actor-Critic algorithm (TD3) to solve the portfolio allocation problem. Our approach achieved an annual Sharpe ratio of 2.68 on test data, which is considered ''Good'' by investors. The approach can be improved in future work by having more computational power to run more experiences and better evaluate the approach. Our environment, agent, and learning process possess many hyperparameters that must be tuned. It will be interesting to see the model's performance with better-tuned parameters, which requires high computation power. In addition, we believe that training an NLP algorithm to process the financial news content instead of only the headline may positively affect the agent performance.

REFERENCES

[1] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, ''Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques,'' *Expert Syst. Appl.*, vol. 42, no. 1, pp. 259–268, 2015.

[2] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, ''Forecasting stock prices from the limit order book using convolutional neural networks,'' in *Proc. IEEE 19th Conf. Bus. Informat. (CBI)*, vol. 1, Jul. 2017, pp. 7–12.

[3] A. Ntakaris, J. Kanniainen, M. Gabbouj, and A. Iosifidis, ''Mid-price prediction based on machine learning methods with technical and quantitative indicators,'' *PLoS ONE*, vol. 15, pp. 1–39, Jun. 2020.

[4] Y. Hao and Q. Gao, ''Predicting the trend of stock market index using the hybrid neural network based on multiple time scale feature learning,'' *Appl. Sci.*, vol. 10, no. 11, p. 3961, Jun. 2020.

[5] M. López de Prado, ''The 10 reasons most machine learning funds fail,'' *J. Portfolio Manage.*, vol. 44, no. 6, pp. 120–133, Jun. 2018.

[6] T. L. Meng and M. Khushi, ''Reinforcement learning in financial markets,'' *Data*, vol. 4, no. 3, p. 110, Jul. 2019.

[7] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2010.

[8] S. Chakraborty, ''Capturing financial markets to apply deep reinforcement learning,'' 2019, *arXiv:1907.04373*.

[9] M. R. Vargas, C. E. M. dos Anjos, G. L. G. Bichara, and A. G. Evsukoff, ''Deep learning for stock market prediction using technical indicators and financial news articles,'' in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–8.

[10] M. Corazza and F. Bertoluzzo, ''Q-learning-based financial trading systems with applications,'' Dept. Econ., Univ. Venice Ca' Foscari, Venice, Italy, Working Papers 2014:15, 2014.

[11] Z. Tan, C. Quek, and P. Y. K. Cheng, ''Stock trading with cycles: A financial application of ANFIS and reinforcement learning,'' *Expert Syst. Appl.*, vol. 38, no. 5, pp. 4741–4755, May 2011.

[12] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, ''Deep direct reinforcement learning for financial signal representation and trading,'' *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2017.

[13] O. Alagoz, H. Hsu, A. J. Schaefer, and M. S. Roberts, ''Markov decision processes: A tool for sequential decision making under uncertainty,'' *Med. Decis. Making*, vol. 30, no. 4, pp. 474–483, Jul. 2010.

[14] R. S. Sutton, F. Bach, and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[15] R. E. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 2010.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, ''Playing Atari with deep reinforcement learning,'' 2013, *arXiv:1312.5602*.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, ''Human-level control through deep reinforcement learning,'' *Nature*, vol. 518, no. 7540, pp. 529–533, 2019.

[18] H. van Hasselt, A. Guez, and D. Silver, ''Deep reinforcement learning with double Q-learning,'' 2015, *arXiv:1509.06461*.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, ''Continuous control with deep reinforcement learning,'' 2019, *arXiv:1509.02971*.

[20] S. Fujimoto, H. van Hoof, and D. Meger, ''Addressing function approximation error in actor-critic methods,'' 2018, *arXiv:1802.09477*.

[21] F. Bertoluzzo and M. Corazza, ''Testing different reinforcement learning configurations for financial trading: Introduction and applications,'' *Proc. Econ. Finance*, vol. 3, pp. 68–77, Jan. 2012.

[22] L. Conegundes and A. C. M. Pereira, ''Beating the stock market with a deep reinforcement learning day trading system,'' in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–8.

[23] C. D. Kirkpatrick and J. R. Dahlquist, *Technical Analysis: The Complete Resource for Financial Market Technicians*. London, U.K.: Pearson, 2006.

[24] J. R. Nofsinger, ''The impact of public information on investors,'' *J. Banking Finance*, vol. 25, no. 7, pp. 1339–1366, Jul. 2001.

[25] X.-Y. Liu, Z. Xiong, S. Zhong, H. Yang, and A. Walid, ''Practical deep reinforcement learning approach for stock trading,'' 2018, *arXiv:1811.07522*.

[26] *Investopedia—Slippage Definition*. Accessed: Oct. 2, 2021. [Online]. Available: https://www.investopedia.com/terms/s/slippage.asp

[27] Z. Jiang, D. Xu, and J. Liang, ''A deep reinforcement learning framework for the financial portfolio management problem,'' 2017, *arXiv:1706.10059*.

[28] A. Akhmetzyanov, R. Yagfarov, S. Gafurov, M. Ostanin, and A. Klimchik, ''Continuous control in deep reinforcement learning with direct policy derivation from Q network,'' in *Human Interaction, Emerging Technologies and Future Applications II*. Cham, Switzerland: Springer, 2020, pp. 168–174.

[29] T. Chong, W.-K. Ng, and V. Liew, "Revisiting the performance of MACD and RSI oscillators," *J. Risk Financial Manage.*, vol. 7, no. 1, pp. 1–12, Feb. 2014.
[30] J. Granville, *Granville's New Key to Stock Market Profits*. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.
[31] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Using structured events to predict stock price movement: An empirical investigation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*. Doha, Qatar: Assoc. Comput. Linguistics, Oct. 2014, pp. 1415–1425.
[32] D. Araci, "FinBERT: Financial sentiment analysis with pre-trained language models," 2019, *arXiv:1908.10063*.
[33] *Yahoo Finance*. Accessed: Nov. 15, 2021. [Online]. Available: https://finance.yahoo.com/
[34] G. Manoim. *Exchange-Calendars*. Accessed: Nov. 15, 2021. [Online]. Available: https://pypi.org/project/exchange-calendars/
[35] *Kaggle—Daily Financial News for 6000+ Stocks*. Accessed: Nov. 15, 2021. [Online]. Available: https://www.kaggle.com/miguelaenlle/massive-stock-news-analysis-db-for-nlpbacktests
[36] (Aug. 2016). *Daily News for Stock Market Prediction*. Accessed: Nov. 15, 2021. [Online]. Available: https://www.kaggle.com/aaron7sun/stocknews
[37] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 1–8.
[38] W. F. Sharpe, "The Sharpe ratio," *J. Portfolio Manage.*, vol. 21, no. 1, pp. 49–58, 1994.
[39] S. Kau, "Algorithmic trading using reinforcement learning augmented with hidden Markov model," Stanford Univ., Stanford, CA, USA, Working Paper, 2017. [Online]. Available: https://www.semanticscholar.org/paper/Algorithmic-Trading-using-Sentiment-Analysis-and-Kaur/3ed53a69019089c467289f5e43486785a8fcd856#citing-papers

TAYLAN KABBANI received the B.Sc. degree in biomedical engineering from Yıldız Technical University, in 2018, and the master's degree in data science, in 2021. He currently joins the Department of Industrial Engineering, Özyeğin University, as a Teaching Assistant. Prior to coming to Özyeğin, he was a Teaching Assistant at the Computer Science Department, Marmara University, in 2019. He worked in multiple roles as a Data Scientist and an AI Engineer. His research interests include AI applications in financial markets, bioinformatics, and mathematical optimization.

EKREM DUMAN was born in Afyon, Turkey, in 1967. He received the B.S. degree in electrical and electronical engineering and the M.S. and Ph.D. degrees in industrial engineering from Boğaziçi University. Since 2011, he has been working with the Industrial Engineering Department, Özyeğin University. He has conducted a number of data analytics and optimization projects mostly for the banking sector. His research interests include industrial applications of operations research, scheduling, and data analytics.