Indian Institute of Technology, Kanpur

MTH698A Report

Algorithmic Trading using Deep RL

Author
Keshav Ranjan

Supervisor
Prof. Amit Mitra

31st October 2024

CERTIFICATE

This is to certify that the project entitled "Algorithmic Trading using Deep RL" submitted by Keshav Ranjan (Roll no.: 208170508) as a part of MTH698A course offered by the Indian Institute of Technology, Kanpur, is a Bonafide record of the work done by him under my guidance and supervision from 1^{st} August 2024 to 31^{st} October 2024.

Amit Mitra

Dr. Amit Mitra
Professor,
Dept. of Mathematics & Statistics
Indian Institute of Technology Kanpur

Annexure-II

DECLARATION

I/We hereby declare that the work presented in the project report entitled Algorithmic Trading using Deep RL is written by me in my own words and contains my own or borrowed ideas. At places, where ideas and words are borrowed from other sources, proper references and acknowledgements, as applicable, have been provided. To the best of my knowledge this work does not emanate from or resemble work created by person(s) other than those mentioned and acknowledged herein.

Name and Signature Keshav Ranjan Keshav Ranjan

Date: 30/10/2024

Acknowledgement

I would like to express my heartfelt gratitude to the Indian Institute of Technology Kanpur and my project supervisor, **Prof. Amit Mitra**, for the opportunity to pursue my MS research through the **MTH698A** course. The knowledge and experience I gained from this project have been invaluable for my future endeavors. I am particularly grateful to **Prof. Amit Mitra** for his mentorship, guidance, and the constructive feedback he provided on my technical reports. His support has significantly enriched my research journey, and I truly appreciate the time and effort he dedicated to my development.

Abstract

This report presents a novel approach to algorithmic trading using Deep Reinforcement Learning (DRL) through a Trading Deep Q-Network (TDQN) algorithm. The TDQN builds on the popular DQN framework and is tailored for financial markets to maximize the **Sharpe ratio**. The trading problem is modeled as a sequential decision-making task, with actions—buy, sell, or hold—optimized based on market observations. The algorithm is trained daily using historical OHLCV (Open-High-Low-Close-Volume) data, generating artificial trajectories to supplement real market scenarios and handle data scarcity. ϵ-greedy strategies are employed to maintain a balance between exploration and exploitation during training. The TDQN algorithm is tested on six selected stocks from Indian and US markets, covering diverse sectors. Key metrics such as annualized returns, volatility, and maximum drawdown are used for performance evaluation. Results indicate that TDQN outperforms traditional strategies like trend following and mean reversion in stable markets with a Sharpe Ratio of **1.572** for Apple Stock and **1.322** for TCS Stock, but faces challenges with high volatility scenarios as observed in Tesla Stock, showing increased variance and reduced profitability. While the TDQN algorithm demonstrates significant potential, it encounters challenges with overfitting, transaction costs, and market regime shifts. Our future work will includes the integration of news sentiment analysis and transfer learning to improve adaptability and performance in dynamic financial environments.
Keywords- Algorithmic Trading, Deep Reinforcement Learning, Sharpe Ratio, AI, Trading Policy, Stocks

Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, financial markets have grown increasingly dynamic and complex, posing new challenges for traders. Traditional trading strategies often struggle to keep up with this volatility, as they rely on fixed mathematical rules and assumptions that do not adapt well to changing environments. On the other hand, the emergence of artificial intelligence (AI) and deep learning has opened new possibilities for developing automated trading algorithms that can make better decisions based on real-time data.

Within this landscape, deep reinforcement learning (DRL) stands out for its ability to handle sequential decision-making problems—an essential requirement in trading. Unlike classical algorithms, DRL agents learn by interacting with the environment, gradually improving their performance through feedback. As financial markets are inherently uncertain and noisy, DRL provides an opportunity to design robust trading strategies capable of both maximizing returns and managing risks efficiently. The motivation for this project lies in leveraging these advancements to explore a novel AI-driven approach to trading.

1.2 Problem Statement

In the financial markets, determining the optimal trading position—whether to buy, sell, or hold an asset—at any given moment is a complex and dynamic challenge. Traditional trading strategies, such as **trend following** and **mean reversion**, are often limited by rigid rules and assumptions, struggling to adapt to rapidly changing market conditions and high volatility. Moreover, financial markets are characterized by **stochastic behavior, partial observability**, and the influence of numerous factors, including macroeconomic trends, news, and sentiment, making it difficult to predict future price movements accurately.

This project addresses the limitations of traditional trading models by implementing a **Trading Deep Q-Network (TDQN)** algorithm, a specialized variant of deep reinforcement learning(DRL). The TDQN algorithm is designed to learn optimal trading actions (buy, sell, or hold) by interacting with the market environment. In addition, trading involves practical complexities such as transaction costs and slippage, which must be considered to ensure the model's performance aligns with real-world conditions.

1.3 Objectives

The main objective of this project is to design and test a deep reinforcement learning-based trading strategy using the TDQN algorithm. Specifically, the project involves evaluating the model on **six stocks from various sectors**, with stocks selected from both **Indian** and **US markets,** to assess the generalizability of the strategy. The algorithm aims to **maximize the Sharpe ratio**, a widely used metric in finance that measures the trade-off between risk and return.

The key objectives are summarized as:

- Develop a novel DRL-based trading policy adapted from the Deep Q-Network (DQN) algorithm.
- Train the algorithm using daily historical data, with the goal of dynamically adjusting positions based on evolving market conditions.

- Benchmark the performance of the TDQN algorithm against classical strategies such as **Buy-and-Hold**, **Trend Following**, and **Mean Reversion**.
- Incorporate realistic constraints like **transaction costs** and **market slippage** to ensure the model performs well under real-world conditions.

1.4 Scope of the Project

This project focuses on evaluating the TDQN algorithm on six carefully selected stocks across **three different sectors**: technology, finance, and energy. These sectors were chosen to reflect the diversity and volatility found in financial markets. The project emphasizes building a model that not only generates profit but also manages risk effectively. The test environment includes both **bullish** and **bearish** market conditions to understand how the model performs under different scenarios.

A crucial aspect of this work is the **daily retraining** of the TDQN model to incorporate new market data, allowing it to adapt to dynamic market conditions. The success of the model will be measured primarily by its Sharpe ratio, with secondary metrics like **maximum drawdown** and **profitability ratio** providing additional insights.

Chapter 2

Literature Review

A significant advancement in trading is algorithmic trading, which uses computer algorithms to make financial decisions. However, this field presents two critical challenges. First, much innovative research is still unavailable because private financial firms frequently conceal their findings because of the enormous financial stakes involved[1]. Second, comparing different approaches is challenging due to the absence of a uniform evaluation framework. Since researchers usually create evaluation frameworks, biases may be introduced. Comparisons are further complicated because trading costs are either completely ignored or stated inconsistently.

Even now, many conventional trading methods that do not use artificial intelligence (AI) are still in use. [2, 3, 4] highlights mean-reversion and trend-following algorithms as notable examples. Over the years, traders and financial analysts have evolved these traditional strategies, which still have significance. However, machine-learning approaches have become more popular nowadays, particularly for market prediction. If market movements can be predicted accurately, traders can make optimal decisions to maximize profits.

In recent years, deep learning (DL) approaches have shown positive developments in this field. Arevalo et al.[5] had proposed a High-Frequency trading strategy that utilizes Deep Neural Networks(DNNs). Similarly, Bao et al.[6] used long short-term memory (LSTM) networks, stacked autoencoders, and wavelet transformations for financial forecasting. These hybrid models successfully capture the complex structure of financial time series data. Paiva et al.[7] presented a strategy that combines mean-variance optimization for portfolio selection with support vector machines (SVMs) for return prediction.

Algorithmic trading has also explored reinforcement learning (RL) approaches in addition to machine learning (ML) models. Unlike predictive-based methods, reinforcement learning (RL) methods do not require explicit predictive models. Instead, they enable agents to learn trading positions through market interaction. One of the first works in the RL field was created by Moody and Safell[8], who developed a recurrent RL algorithm to find trading strategies. Dempster and Leemans[9] applied adaptive RL techniques for foreign exchange trading.

Recent years have seen the rise of deep reinforcement learning (DRL) techniques, which combine RL and deep learning. Deng et al.[10] used fuzzy logic to manage uncertainty in time series data and developed a trading system that does not utilize technical indicators, using a fuzzy recurrent deep neural network. Carapuço et al.[11] used deep Q-learning to trade in the foreign exchange market to show that DRL can handle volatile markets.

Various innovative DRL extensions have been proposed. Jeong and Kim[12] addressed the problem of insufficient financial data by combining RL with deep networks and transfer learning. Almahdi and Yang[13] developed a constrained portfolio management system by combining recurrent RL and particle swarm optimization. A simultaneously time-driven, feature-aware DRL model was presented by Lei et al.[14] to enhance trading decision-making and signal representation. Park et al.[15] investigated the application of deep Q-learning to combinatorial action spaces in portfolio trading.

Despite these developments, there are still some doubts about the reliability of the studies in this field. Ioannidis[16] argues that many published findings may need to be more reliable, especially in a sensitive field like market prediction. Bailey et al.[17] criticized financial research for often lacking scientific precision. Our project aims to overcome these obstacles by creating a novel DRL algorithm.

Chapter 3

Problem Formalization

3.1 Algorithmic Trading

Algorithmic trading, often referred to as quantitative trading, is a specialized field within finance that enables the automation of trading decisions through predefined mathematical rules executed by machines. This approach allows for rapid, data-driven trading, often leveraging high-frequency trading (HFT) capabilities to capitalize on short-term market fluctuations. While some distinctions exist in literature between algorithmic trading (execution of trades) and quantitative trading (strategic decision-making), this project uses these terms interchangeably to encompass a full algorithmic trading system.

Algorithmic trading has demonstrated significant benefits for financial markets, particularly in enhancing liquidity. Various markets support algorithmic strategies, from traditional stock exchanges to FOREX (foreign exchange) for currency trading, and commodities markets. The emergence of cryptocurrencies, such as Bitcoin, has opened up new avenues, further broadening the scope of algorithmic trading applications. While the DRL (Deep Reinforcement Learning) algorithms in this report are adaptable across various markets, the primary focus will be on stock trading, with possible future applications in other asset classes.

Trading in this context involves managing a portfolio, which may consist of a diverse set of assets like stocks, bonds, commodities, and currencies. In this report, the portfolio is simplified to include a single stock and cash held by the trading agent. The portfolio's value at any time t, denoted by v_t, reflects both the cash available (v_t^c) and the value of held shares (v_t^s), both of which fluctuate with market conditions. Trades involve straightforward exchanges between cash and shares, governed by the structure of an order book(Table 3.1), which records all buy (bid) and sell (ask) orders in the market.

Buy		**Sell**	
Quantity(q)	**Price(p)**	**Price(p)**	**Quantity(q)**
5,000	99	99	4,000
8,000	98	100	10,000
10,000	97	101	1,000
15,000	95	103	15,000

(a)

Buy		**Sell**	
Quantity(q)	**Price(p)**	**Price(p)**	**Quantity(q)**
1,000	99	100	10,000
8,000	98	101	1,000
10,000	97	103	15,000
15,000	95	104	3,000

(b)

Table 3.1: An example trade order book for ABC Inc. showing (a) before matching a trade and (b) after matching a trade. Buy orders are ranked with the highest price at the top, while sell orders are ranked with the lowest price at the top.

An order represents the willingness of a market participant to trade and is composed of a price p, a quantity q and a side (bid(buy) or ask(sell)). For a trade to occur, a match between bid and ask orders is required, an event which can only happen if $p_{max}^{bid} \geq p_{min}^{ask}$, with p_{max}^{bid} (p_{min}^{ask}) being the maximum(minimum) price of a bid (ask) order. Within this framework, the trading agent faces a complex challenge, typical of high-stakes financial markets: deciding precisely **what, when, and how much to trade, and at what price**. This ongoing decision-making process forms the core problem addressed in this report, framing algorithmic trading as a **sequential decision-making** task that is well-suited for reinforcement learning techniques.

$$\begin{tikzcd}
\cdots & i_t \arrow[r, "\pi(a_t|i_t)"] & a_t \arrow[r, "t \rightarrow t+1"] & i_{t+1} \arrow[r, "\pi(a_{t+1}|i_{t+1})"] & a_{t+1} & \cdots
\end{tikzcd}$$

Figure 3.1: Trading strategy execution(a sequential process)

We know that the trading activity is a continuous process. So, to analyze the algorithmic trading problem in this report we will **discretize** the continuous timeline. This process divides the trading timeline into numerous discrete time steps t, each with a constant duration Δt. For clarity, the notation $t+1$ and $t-1$ is employed to represent the transition from time step t to $t+\Delta t$ and from t to $t-\Delta t$, respectively.

The duration Δt is intricately tied to the trading frequency that the trading agent aims to achieve, such as very high trading frequency, intraday, daily, or monthly. This discretization imposes a constraint on trading frequency; specifically, the duration Δt cannot be infinitely small due to technical limitations, which caps the maximum attainable trading frequency at $1/\Delta t$. For our report, the constraint is addressed with a focus on a daily trading frequency, where the trading agent executes a decision once each day.

Trading Strategy & execution

A trading strategy can be seen as a programmed policy $\pi(a_t|i_t)$, which may be deterministic or stochastic. This policy defines the trading action a_t based on the information i_t accessible to the trading agent at time step t. A defining feature of trading strategies is their sequential nature, as illustrated in Fig 3.1. When executing a trading strategy, an agent follows these iterative steps:

1. **Market Information Update**: Gather and update the available market information, i_t.
2. **Policy Execution**: Apply the policy $\pi(a_t|i_t)$ to determine the trading action a_t.
3. **Action Application**: Execute the chosen trading action a_t.
4. **Advance Time Step**: Move to the next time step $t \to t+1$ and return to step 1.

The above algorithmic trading decision-making process follows a sequential step and thus, it shares similarities with various other problems which are successfully tackled by RL. We will formally describe a RL problem for Algorithmic Trading in the next section.

3.2 Reinforcement Learning for Algorithmic Trading

Reinforcement learning (RL) is an advanced type of machine learning known for its effectiveness across diverse applications, including games and robotics. In RL, an agent learns by taking actions in an environment with the goal of maximizing cumulative rewards. This learning process involves developing a policy($\pi(a_t|s_t)$) that maps states(including information present) to actions, aiming to maximize the agent's expected cumulative reward over time.

In recent years, RL has gained quite an attention in quantitative trading, a field traditionally dominated by rule-based systems. These rule-based strategies, typically designed by human experts, rely on technical indicators like moving averages and Relative Strength Index (RSI) to make trading decisions. While effective, they often lack flexibility, especially when adapting to rapidly shifting market conditions.

RL-based approaches offer a promising alternative, allowing trading algorithms to learn from historical data and adapt dynamically to the ever-changing market landscape. This adaptability enables RL algorithms to make informed trading decisions in real time, offering potential advantages over static, rule-based systems. For instance, research by Moody and Saffell[8] demonstrated that an RL-based trading strategy outperformed both buy-and-hold and moving average strategies for S&P 500 futures.

3.2.1 Reinforcement Learning in Sequential Decision-Making

Reinforcement learning (RL) is a powerful framework for developing adaptable trading strategies, enabling an agent to make sequential decisions based on ongoing feedback from its environment. In the context of algorithmic trading, the environment represents the financial market, and the agent is the trading system, which learns by

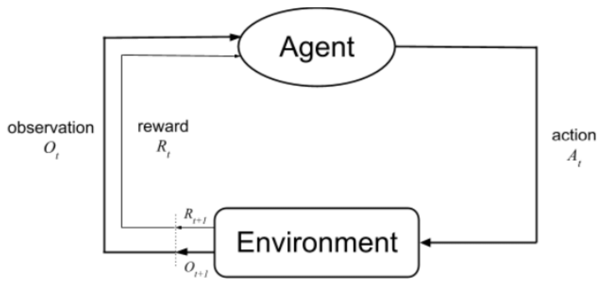


Figure 3.2: RL Model

receiving feedback in the form of rewards (e.g., profit or loss) based on its trading actions. The RL process involves a sequence of interactions, illustrated in Fig 3.2. The reinforcement learning (RL) framework for sequential trading is based on a Markov Decision Process (MDP), which provides a structured way to model decision-making in an uncertain environment. An MDP is characterized by the following components:

Markov Decision Processes(MDPs)

In the context of RL for algorithmic trading, a Markov Decision Process (MDP) serves as a fundamental framework for modeling decision-making in uncertain environments. MDPs are particularly well-suited for this application due to their ability to capture the sequential nature of trading decisions and the stochastic dynamics of financial markets. The key reasons for utilizing MDPs include:

- **Structured Decision-Making:** MDPs provide a formal structure for making decisions where outcomes are uncertain. They facilitate the analysis of various policies and their consequences over time.
- **Markov Property:** The decision-making process adheres to the Markov property, meaning that the future state depends only on the current state and action, not on the history of past states. This simplifies the decision-making process and allows for effective policy formulation.
- **State Space:** The collection of all possible states that the agent can encounter in the environment. Each state reflects the market conditions at a given time.
- **Action Space:** The set of all possible actions the agent can take in response to its current state, influencing future states and rewards.
- **Transition Probabilities:** The probabilities of transitioning from one state to another, given a specific action. These probabilities are crucial for understanding the dynamics of the environment and for optimizing the policy.
- **Optimal Policy Formulation:** MDPs enable the derivation of optimal policies that maximize cumulative rewards over time. By formalizing the relationship between states, actions, and rewards, MDPs facilitate the development of algorithms that can learn effective trading strategies.

Given above understanding of MDPs, we can now outline the specific components that define the RL framework for our sequential trading:

1. **Observation of Environment**: At each time step t, the agent looks at the market, getting a state s_t and an observation o_t. This observation gives the agent important information for making decisions.
2. **Action Selection**: Using its past experiences stored in history h_t, the agent chooses an action a_t based on its policy $\pi(a_t|h_t)$, which guides it on the best move to make.
3. **Reward Feedback**: The agent receives a reward r_t, which shows how well its chosen action worked. This reward helps the agent improve its future decisions.

4. **History Update**: The agent updates its history h_t to include the latest observation, action, and reward, represented as:
$$h_t = \{(o_\tau, a_\tau, r_\tau) \mid \tau = 0, 1, \ldots, t\}.$$
This history allows the agent to continuously improve its policy by keeping a record of its past experiences to inform future decisions.

3.2.2 Optimal Policy and Reward Maximization

The goal in RL is to design policies that maximize rewards over time. This cumulative reward is evaluated through an **expected discounted sum of rewards**, balancing both short and long-term outcomes:

Optimal Policy Definition

The optimal policy π^*, which maximizes this cumulative reward, can be expressed as:

$$\pi^* = \arg\max_\pi \mathbb{E}[\mathbf{R} \mid \pi],$$

where the cumulative reward $\mathbf{R}$ over an infinite time horizon is given by:

$$\mathbf{R} = \sum_{t=0}^{\infty} \gamma^t r_t, \quad \text{where } \gamma \text{ is the discount factor.}$$

Discount Factor γ

The discount factor γ (where $\gamma \in [0, 1]$) controls the importance of future rewards:

- **Short-term Focus**: A discount factor $\gamma = 0$ creates a "myopic" agent that prioritizes immediate rewards, disregarding future consequences.
- **Long-term Orientation**: As γ approaches 1, the agent considers future rewards more heavily, adopting a long-term perspective on decision-making(we can consider this as equivalent to our traditional buy & hold strategy).

Therefore, we need to optimally tune the discount factor γ for balancing short-term responsiveness with a strategic, long-term perspective. This balance is crucial for profitability while maintaining effective risk management. We will now look more deeply into each components required for the RL framework. Also, we will do some reductions on each of these components for our model to work in the next few sections.

3.2.3 RL Agent Observations o_t for Trading

In algorithmic trading using reinforcement learning (RL), the "environment" in which the RL agent operates reflects the complex ecosystem of the stock market. This environment serves as the agent's world, encompassing trading mechanics, price fluctuations, market trends, macroeconomic indicators, and even the psychology driving market participants.
A significant challenge for the agent is that it has only a limited view of this environment. Essential details-such as confidential company information, competitors' trading strategies, and unforeseen news events-remain hidden, creating an incomplete picture. Consequently, the agent must navigate and make decisions based on partial information, which introduces considerable risk and uncertainty into its trading strategy.
The data available to the agent is varied, ranging from pure numbers (like prices and volumes) to qualitative factors (like news sentiment). The RL agent has to process these data points carefully, translating them into useful metrics for trading decisions. This process of quantifying and interpreting data also requires reducing any bias, ensuring that the information remains objective and actionable.
An added layer of complexity is the "sequential" nature of trading: the market is constantly moving and evolving over time, so the agent's understanding of it is cumulative. At each trading time step t, the RL agent observes a partial representation of the stock market's state, denoted $s_t \in \mathbf{S}$. The specific information that the agent accesses at this time step is captured as $o_t \in \mathbf{O}$, ideally encompassing all market-influencing data. Each observation o_t consists of data accumulated over the prior τ time steps, along with new insights gained at time t. In this context, we can express the RL agent's observations mathematically as follows:

$$o_t = \bigcup_{t'=t-\tau}^{t} \{S(t'), D(t'), T(t'), I(t'), M(t'), N(t'), E(t')\}$$

where each component represents critical elements of the trading environment[18]:

- $S(t)$: The agent's current position, including the number of shares it holds, its available cash, and any ongoing trades. This helps the agent understand its exposure and risk at any given time.
- $D(t)$: OHLCV data summarizing the stock price movement, where:
  - p_t^O & p_t^C: Opening & Closing price for the time frame [t-1, t],
  - p_t^H & p_t^L: Highest price & Lowest price over the time frame [t-1, t],
  - V_t: Volume of shares traded over time frame [t-1, t].
- $T(t)$: Temporal information like the date, weekday, and exact time, aiding the agent in situational awareness.
- $I(t)$: Technical indicators, such as MACD, RSI, and ADX, which provide deeper insights into market trends and patterns.
- $M(t)$: Macroeconomic indicators (e.g., interest rates, exchange rates) that can forecast market direction.
- $N(t)$: News data from multiple sources, processed through sentiment analysis to generate metrics like sentiment polarity and subjectivity, which are valuable for assessing market sentiment.
- $E(t)$: Additional data relevant to trading, potentially including information on competitors' strategies, confidential insights, or speculative trends.

For our report we only have information about the OHLCV data, thus from now on we will assume that the RL agent will only have information about the OHLCV data $D(t)$ and the state information $S(t)$. Hence, with this assumption, we can express o_t as the following:

$$o_t = \left\{ \{p_t^O, p_t^C, p_t^H, p_t^L, V_t\}_{t'=t-\tau}^{t}, P_t \right\}$$

where P_t is the trading position of RL agent at time t(either long or short, explained in the next subsection).

3.2.4 RL Agent Actions a_t for Trading

In the context of reinforcement learning (RL) for algorithmic trading[18], the agent makes decisions at each time step t by executing a trading action a_t based on its programmed policy $\pi(a_t|h_t)$. The core challenges for RL agent involve determining **whether to trade, how to trade, and the volume of shares to trade**. These decisions can be formalized through the quantity of shares transacted by the agent at time t, denoted as $Q_t \in \mathbb{Z}$. Thus, the trading actions can be expressed as:

$$a_t = Q_t \tag{1}$$

Depending on the value of Q_t, three distinct scenarios arise:

- **Buying**: If $Q_t > 0$, the agent is purchasing shares by placing bid orders on the order book.
- **Selling**: If $Q_t < 0$, the agent is selling shares by placing ask orders on the order book.
- **Holding**: If $Q_t = 0$, the agent chooses to hold its position without engaging in any buying or selling.

The RL agent's actions ultimately result in orders that are executed on the order book. It relies on an external module, referred to as the trading execution system, to convert its intended actions into actual trades based on the value of Q_t. While various execution strategies can be employed, these are not the primary focus of this paper.

The actions taken by the agent affect two crucial components of its portfolio value: the cash value (v_c) and the share value (v_s). Assuming trades are executed close to market closure at price $p_t \approx p_t^C$, the updates for these components can be represented by the following equations:

$$v_{t+1}^c = v_t^c + Q_t p_t \tag{2}$$

$$v_{t+1}^s = (n_t + Q_t) p_{t+1} \tag{3}$$

In these equations, $n_t \in \mathbb{Z}$ indicates the number of shares owned by the agent at time t. Interestingly, the agent can hold a negative quantity of shares, which signifies that shares are borrowed and sold, obligating the agent to repay the lender in the future. This feature opens up additional trading strategies for the agent.

Constraints on Trading Actions

Two significant constraints are imposed on the quantity of shares Q_t traded by the agent. First, while the share value v_s can vary in sign, the cash value v_c must always remain positive throughout all trading time steps t. This requirement establishes a limit on the maximum number of shares the agent can purchase, as derived from equation (2).

Second, there is a risk of being unable to repay the borrowed shares if the agent experiences substantial losses. To mitigate this risk, the cash value v_c must be adequately high when the agent holds a negative number of shares, ensuring it can return to a neutral position ($n_t = 0$).

The RL agent assumes a maximum expected price change, denoted as $\epsilon \in \mathbb{R}^+$, before initiating trading activities. This parameter signifies the maximum anticipated fluctuation in the market over the entire trading horizon, allowing the agent to repay the share lender provided that market movements remain within this predefined limit.

The constraints on the RL actions at time step t can be formulated mathematically as follows[19]:

$$v_{t+1}^c \geq 0 \tag{4}$$

$$v_{t+1}^c \geq -n_{t+1} p_t (1 + \epsilon) \tag{5}$$

Moreover, the following condition must hold:

$$\left| \frac{p_{t+1} - p_t}{p_t} \right| \leq \epsilon \tag{6}$$

But the above equations are some what inaccurate because of the negligence of the trading costs occur during a trade. To make our simulations comparable to real life scenarios, we need to account for these costs when trading. Below we will explain these costs in more details and will correct our equations accordingly thereafter.

Trading Costs

The trading costs can be subdivided into two costs, namely explicit costs and implicit costs. Explicit costs are induced by transaction costs and taxes while implicit costs also called Slippage costs which consists of three main elements as given below:

- **Spread Costs**: These costs arise from the difference between the minimum ask price (p_{min}^{ask}) and the maximum bid price (p_{max}^{bid}), collectively known as the spread. Due to the complexity of the order book's complete state, trading decisions are often based on the mid price, defined as

  $$p_{mid} = \frac{p_{max}^{bid} + p_{min}^{ask}}{2}.$$

  However, executing a buy trade at p_{mid} results in a price of at least p_{min}^{ask}, while a sell trade results in a price of at most p_{max}^{bid}. The significance of these spread costs increases in low liquidity environments, where the volume of shares traded is substantial.

- **Market Impact Costs**: These costs are incurred when a trader's actions influence market prices. Each trade, whether a buying or selling order, has the potential to affect the stock price, particularly in markets with low liquidity relative to the trading volume.

- **Timing Costs**: Timing costs are associated with the delay in executing trades after a decision is made, as market prices are constantly fluctuating. These delays can be attributed to two primary factors:

- The inevitable latency which delays the posting of orders to the market order book.
- The intentional delays introduced by trading execution systems. For instance, a large trade might be broken into smaller trades executed over time to mitigate market impact costs.

Correction

While explicit costs are relatively easy to take into account, the valid modeling of slippage costs is a truly complex task. For our algorithm, the integration of both costs in RL environment is performed. When a trade is executed, a certain amount of capital equivalent to a percentage C of the amount of money invested is lost. This parameter was realistically chosen to be 0.1% for our simulations. Thus following these corrections, we will rewrite the equation (2) with a corrective term taking into account for the trading costs:

$$v_{t+1}^c = v_t^c + Q_t p_t - C|Q_t|p_t, \quad \text{where the last term is subtracted for the trading costs} \tag{7}$$

Consequently, equation (5) can be rewritten as:

$$v_{t+1}^c \geq -n_{t+1} p_t (1+\epsilon)(1+C) \tag{8}$$

Hence, In the context of algorithmic trading, the RL agent's action space, denoted by **A**, represents the discrete set of allowable values for the quantity of traded shares Q_t[18].

$$\mathbf{A} = \left\{ Q_t \in \mathbb{Z} \cap \left[Q_t^{min}, Q_t^{max} \right] \right\}. \tag{9}$$

where:

- Q_t^{min} and Q_t^{max} represent the minimum and maximum allowable trade quantities at time t, determined as:

$$Q_t^{max} = \frac{v_t^c}{p_t(1+C)}$$

and

$$Q_t^{min} = \begin{cases} \frac{\Delta_t}{p_t \epsilon (1+C)} & \text{if } \Delta_t \geq 0 \\ \frac{\Delta_t}{p_t(2C + \epsilon(1+C))} & \text{if } \Delta_t < 0 \end{cases}$$

- Here, Δ_t is defined as:

$$\Delta_t = -v_t^c - n_t p_t (1+\epsilon)(1+C)$$

Now, we will again reduce this action space for the scope of our report and simulations in order to reduce the time complexity of our algorithm. The reduced action space **A** contains only two RL actions which can be mathematically written as:

$$a_t = Q_t \in \{\text{Long}_t, \text{Short}_t\}$$

The first RL action Long_t maximises the number of shares owned by the RL agent, by converting as much cash value v_t^c as possible into share value v_t^s . It can be mathematically expressed as follows:

$$\text{Long}_t = \begin{cases} \left\lfloor \frac{v_t^c}{p_t(1+C)} \right\rfloor & \text{if } a_{t-1} \neq \text{Long}_{t-1}, \\ 0 & \text{Otherwise.} \end{cases}$$

As we can see Long_t action is always valid as it is included in the original action space **A**. Hence, the number of shares owned by the RL agent at time t, $N_t^{long} = n_t + \text{Long}_t$.

The second RL action Short_t aims to convert share value v_t^s into cash value v_t^c. Hence, this action not only close the Long position i.e. now the number of shares RL agent own will be equal to $-N_t^{long}$ but also opens a short position for n_t shares. It can be mathematically expressed as follows:

$$\text{Short}_t = \begin{cases} -2n_t - \left\lfloor \frac{v_t^c}{p_t(1+C)} \right\rfloor & \text{if } a_{t-1} \neq \text{Short}_{t-1}, \\ 0 & \text{Otherwise.} \end{cases}$$

However, the above equation may violate the lower bound for the action space **A**. So we will consider

$$\text{Short}_t = \max\{\text{Short}_t, Q_t^{min}\}$$

It should be noted that the two reduced RL actions are actually related to the next trading position of the agent, designated as P_{t+1}. As, the first action Long_t induces a *long* **trading position** because the number of shares is positive while the second action Short_t induces a *short* **trading position** as the number of shares is negative.

3.2.5 RL Agent Rewards & Objective

For this algorithmic trading problem, a natural choice for the RL rewards is the strategy daily returns. The RL rewards can be mathematically expressed as the following:

$$r_t = \frac{v_{t+1} - v_t}{v_t}, \quad \text{where } v_t \text{ is the portfolio value.}$$

But, objectively accessing the performance of the model just based on profitability is a dangerous task. A well performed trading strategy should not only be considering the profit but also to mitigate the risk associated with it. For instance, a well performing strategy just based on profit may give a very poor performance when a sudden change in market condition happens. Thus, we need to balance out these two goals simultaneously. For that we will try to maximize the *Sharpe Ratio*, a performance indicator widely used in the fields of finance and algorithmic trading. *Sharpe Ratio* is very well suited for our performance assessment task as it considers both profit generation and risk associated with trading activity. It is mathematically defined as:

$$\text{Sharpe Ratio}, S_r = \frac{\mathbb{E}[\mathbf{R}_s - \mathbf{R}_f]}{\sigma_r} = \frac{\mathbb{E}[\mathbf{R}_s - \mathbf{R}_f]}{\sqrt{\text{Var}[\mathbf{R}_s - \mathbf{R}_f]}}$$

where

- $\mathbf{R}_s$ is the trading strategy return over a certain time period, modeling its profitability.
- $\mathbf{R}_f$ is the risk-free return from a safe investment.
- σ_r is the std. deviation of trading strategy excess return from the risk-free return, modeling its risk.

The *Sharpe Ratio* often quoted by these carrying out trading strategies is the Annualized Sharpe Ratio, the calculation of which depends upon the trading period of which the returns are measured. Assuming there are N trading periods in a year, then the Annualized Sharpe Ratio is calculated as follows:

$$\text{Sharpe Ratio} = \sqrt{N}\left(\frac{\mathbb{E}[\mathbf{R}_s - \mathbf{R}_f]}{\sigma_r}\right)$$

For our report, we will consider the risk-free return to be negligible since we are trading daily, so it will not significantly change the value of *Sharpe Ratio*. thus, we can write *Sharpe Ratio(S_r)* as follows:

$$S_r \simeq \frac{\mathbb{E}[\mathbf{R}_s]}{\sqrt{\text{Var}[\mathbf{R}_s]}}$$

In practice, to compute the *Sharpe Ratio* S_r, we will first calculate the daily returns ρ_t achieved by the trading strategy which is computed as follows:

$$\rho_t = \frac{v_t - v_{t-1}}{v_{t-1}},$$

where v_t represents the portfolio value at time t.

Next, the *Sharpe ratio* is calculated by evaluating the ratio between the mean and the standard deviation of the daily returns. To obtain the Annualized Sharpe Ratio, this value is multiplied by the square root of the number of trading days in a year (typically 252 days).

NOTE: Generally, a *Sharpe Ratio* **greater than 1** indicates a **good** trading strategy.
A well-performing trading strategy should demonstrate acceptable performance across diverse market conditions, each exhibiting distinct patterns. For example, the strategy should be robust enough to handle both bull markets (characterized by strong upward price trends) and bear markets (characterized by strong downward price trends), as well as varying levels of market volatility.

The primary objective of this research is to develop a novel trading strategy leveraging deep reinforcement learning (DRL) techniques, aimed at maximizing the average Sharpe ratio across a comprehensive range of stock markets.

Chapter 4

Methodology and Algorithm Design

In this chapter we will design a novel Deep Reinforcement Learning(DRL) algorithm to solve the above algorithmic trading problem introduced. We will call this trading strategy, Trading Deep Q-Network algorithm(TDQN). This algorithm is inspired from a successful DQN algorithm presented by Mnih et al.[20] and is modified according to the specific decision problem in hand. We will first discuss what a DQN alogrithm is and why it is being used in our case.

4.1 Deep Q-Network algorithm(DQN)

The DQN (Deep Q-Network) algorithm was developed in 2013 (Figure 4.1). It was able to solve a wide range of Atari games (some to superhuman level) by combining reinforcement learning and deep neural networks at scale. The algorithm was developed by enhancing a classic RL algorithm called **Q-Learning** with **deep neural networks** and a technique called **experience replay**.

4.1.1 Q-Learning

To understand DQN, it's essential first to grasp the concept of Q-learning and its objective, the Q-function. In Q-learning, the Q-function $Q(s, a)$ is designed to approximate the expected cumulative reward that an agent will receive if it starts in a given state s, takes an action a, and then follows a certain policy. Mathematically, the Q-function for a policy π is represented as:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t, a_t, \pi\right] \tag{4.1}$$

The goal in Q-learning is to find the optimal policy π^* that maximizes the expected cumulative reward, leading to the optimal Q-function $Q^*(s, a)$, which obeys the Bellman Optimality Equation:

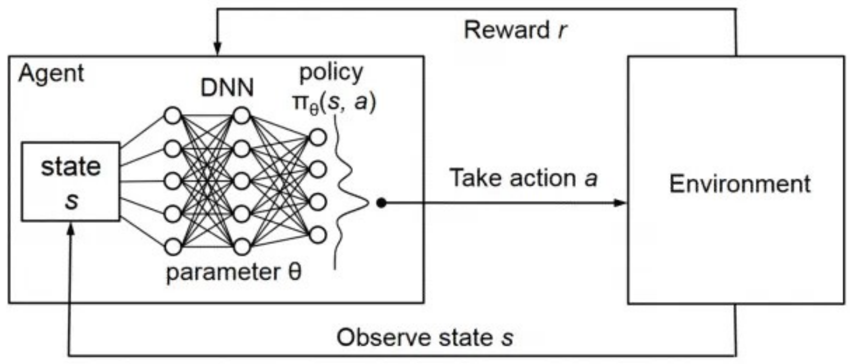


Figure 4.1: DQN Algorithm

$$Q^*(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q^*(s',a')\right] \tag{4.2}$$

Q-Learning Algorithm

In the Q-Learning algorithm, the goal is to learn iteratively the optimal Q-value function using the Bellman Optimality Equation (equation 4.2). To do so, we will store all the Q-values in a table that we will update at each time step using the Q-Learning iteration:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r_t + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

where α is the learning rate (we need to tune this hyperparameter as it controls the convergence). It can be shown that the above iteration will give an optimal Q-function (Q^*)[20].

We know that Q-learning uses a Q-table to store values for each state-action pair. However, in case the state space, action space or both are very large or continuous, maintaining a Table becomes infeasible. DQN solves this problem by approximating Q-function with the help of **deep neural network**.

4.1.2 DQN Architecture & Components

Q-Network

In DQN, we train a deep neural network function approximator with weights θ, to approximate **Q-function** which is denoted by $Q(s,a;\theta)$. The network takes the state s as input and outputs Q-values for each possible action a. The architecture consists of few convolutional layers followed by fully connected layers which makes it suitable for processing high-dimensional inputs.

Experience Replay

DQN uses a technique called **Experience Replay**, which stores its past experiences(or transitions) in a **Replay Memory Buffer** $\mathcal{D}$. At time step t, the transition is a tuple containing the current state of the environment, the chosen action, the reward and the next state of the environment:

$$e_t = \{s_t, a_t, r_t, s_{t+1}\}$$

This technique offers various advantages:

- **Reduces Correlation:** By randomly sampling batches from the buffer, the network avoids learning from sequentially correlated samples, which improves stability and generalization.
- **Efficient Data Utilization:** By reusing past transitions, the algorithm learns more efficiently from limited experiences.

Target Network

This algorithm uses two DNNs to stabilize the learning process.

- The first one the **main neural network**, parameterized by a weight vector θ, and it is used to estimate the Q-values for the current state s and action a: $Q(s,a;\theta)$.
- The second is called **target neural network**, parameterized by a weight vector $\tilde{\theta}$, has the same architecture as the main neural network, but is used to estimate the Q-values for the next state s' and action a'.

The target network $\tilde{\theta}$ is updated less frequently than the main network, often after a fixed number of episodes. This decouples the target generation from the main-network's updates, stabilizing learning by reducing oscillations in Q-values.

4.1.3 Training Procedure

The core of DQN's learning process revolves around updating the Q-values based on the observed experiences and minimizing the error between predicted and target Q-values.

Loss Function

The loss function for a mini-batch sampled from the replay memory buffer is defined as:

$$L(\theta) = \mathbb{E}_{\{s,a,r,s'\}\sim\mathcal{D}}\left[(y - Q(s,a;\theta))^2\right]$$

where

- $L(\theta)$: Loss function for the main neural network parameters θ.
- $\{s,a,r,s'\} \sim \mathcal{D}$: Mini-batch of transitions sampled from the replay memory buffer $\mathcal{D}$.
- $y = r + \gamma \max_{a'} Q(s', a'; \tilde{\theta})$, this equation means that the target y combines the immediate reward r with the maximum estimated future reward from the next state s', computed using the target neural network.

Gradient Update Rule

We will minimize above loss function using **Gradient Descent**. We begin by calculating the gradient of the loss function. This gradient essentially guides us on how much the Q-values need to adjust for the network to improve. Mathematically, this is represented as:

$$\nabla_\theta L(\theta) = \mathbb{E}_{\{s,a,r,s'\}\sim\mathcal{D}}\left[(y - Q(s,a;\theta))\nabla_\theta Q(s,a;\theta)\right]$$

In simpler terms, this expression measures the difference (error) between our target Q-value y and the network's current Q-value prediction $Q(s,a;\theta)$ and uses it to guide the adjustments to parameters θ.
Once we have this gradient, we apply it in a gradient descent update step to adjust θ:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$$

where α is the learning rate, which controls how large each update step should be (as it might overshoot if α is large or might take too long to converge if small).
In each step, the gradient helps us move in the direction that reduces the loss. So with every update, the Q-network should, ideally, get better at estimating the Q-values more accurately.

4.1.4 Exploration & Exploitation

The DQN algorithm is said to be off-policy as it exploits in batch mode previous experiences e_t collected at any point during training. DQN employs a ϵ-Greedy policy to balance exploration and exploitation:

- With probability ϵ, a random action is chosen (exploration).
- With probability $1 - \epsilon$, the action that maximizes the Q-value is selected (exploitation).

The ϵ value is set high at the start and decayed over time, encouraging exploration early in training and favoring exploitation as the agent starts learning.

4.2 Our Novel TDQN Algorithm

In the Algorithmic Trading domain, where understanding the full complexity of the market environment $\mathcal{E}$ is impractical. The Trading Deep Q-Network algorithm(TDQN) is developed to interact with this environment $\mathcal{E}$ using RL techniques, but we will require some modifications in the original DQN algorithm and an artificial trajectory for training in this environment.

4.2.1 Trajectory Generation

As real-time interaction with a live trading environment is infeasible at the time of training, we can only rely on artificial trajectories generated from the limited historical OHLCV data (Open, High, Low, Close, Volume). Below is the procedure:

1. **Definition of Trajectory:** A trajectory τ consists of a sequence of observations $o_t \in \mathbf{O}$, actions $a_t \in \mathbf{A}$, and rewards r_t from the RL agent over a trading period T:

$$\tau = \{(o_0, a_0, r_0), (o_1, a_1, r_1), \ldots, (o_T, a_T, r_T)\}$$

2. **Initial Real Trajectory:** The algorithm begins with a single "real" trajectory derived from historical data, representing the stock market's behavior without the agent's influence (i.e., with no trades executed).

3. **Artificial Trajectories for Exploration:** To simulate various trading scenarios, we generate artificial trajectories by overlaying sequences of trading actions on the historical observations. This way, we can mimic the RL agent's interactions with the environment. These sequences allow the agent to explore different strategies without actually affecting the underlying market data, as the historical price movements remain untouched by these simulated actions.

4. **Actions Exploration:** As the action space $\mathbf{A}$ was reduced to two main actions (Long & Short) in our problem. We can employ a **dual-action trick** for effective exploration:

   - At each time step t, the chosen action a_t will get executed on the primary environment $\mathcal{E}$ and
   - Simultaneously, the opposite action $\tilde{a_t}$ will get executed on a copy of environment $\tilde{\mathcal{E}}$.

   Although, above trick does not completely resolve the exploration-exploitation trade-off, it offers a low-cost way to increase exploration given a limited action space.

4.2.2 Modifications & enhancements

The classical DQN algorithm was adopted to address the specific challenges of algorithmic trading. Below are the following changes and modifications made[18]:

1. Since, the input is financial **time-series data** rather than raw images, the convolutional neural network(CNN) of classical DQN algorithm is replaced with a **feed forward DNN** with Leaky RELU activation functions.

2. The standard DQN suffers from overestimation bias in action-value estimates that leads to poor trading decisions. To address this, we integrates Double DQN[21], where the max operation for Q-value updates is split into action selection and action evaluation. The Double DQN modifies the target calculation as follows:

$$y_t = r_t + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta); \tilde{\theta})$$

   where $\tilde{\theta}$ represents the weights of a target network, which provides stable Q-values over multiple updates.

3. The ADAM optimizer was used to improve the training stability and the convergence speed of the algorithm.

$$\theta \leftarrow \theta - \alpha \text{ADAM}(\nabla_\theta L(\theta))$$

4. We have used **Huber loss** instead of MSE loss function for stability benefits. It is mathematically written as:

$$H(x) = \begin{cases} \frac{1}{2}|x|^2 & \text{if } |x| \leq 1 \\ |x| - \frac{1}{2} & \text{otherwise} \end{cases}$$

5. **Gradient clipping** is applied to manage the exploding gradient problem that can destabilize training. We avoids sudden large updates(which can hinder convergence) by capping gradients to a maximum threshold i.e. gradient = max(gradient, Threshold).

6. To enhance training stability and mitigate overfitting, we incorporates several techniques: **Xavier initialization** to ensure balanced weight distributions across the neural network, **batch normalization** to normalizes inputs, facilitating faster and more robust training. The algorithm also employs dropout, L2 regularization, and early stopping to reduce overfitting. Additionally, data augmentation techniques such as signal shifting, noise addition, and low-pass filtering are applied to generate diverse trading scenarios from limited historical data, thereby improving the agent's robustness to market fluctuations.

Algorithm 1: TDQN Algorithm

```
1. Initialise the experience replay memory $M$ of capacity $C$.
2. Initialise the main DNN weights $\theta$ (using Xavier initialisation).
3. Initialise the target DNN weights $\tilde{\theta} = \theta$.
4. **for** episode = 1 to $N$:
   (a) Acquire the initial observation $o_1$ from the environment $\mathcal{E}$ and preprocess it.
   (b) **for** $t = 1$ to T:
       i. with probability $\epsilon$:
          • Select a random action $a_t$ from $\mathbf{A}$.
       ii. **else**:
          • Select $a_t = \arg\max_{a \in \mathbf{A}} Q(o_t, a; \theta)$.
       iii. Copy the environment $\tilde{\mathcal{E}} = \mathcal{E}$.
       iv. Interact with the environment $\mathcal{E}$ (action $a_t$) and get the new observation $o_{t+1}$ and reward $r_t$.
       v. Perform the same operation on $\tilde{\mathcal{E}}$ with the opposite action $\tilde{a}_t$, getting $\tilde{o}_{t+1}$ and $\tilde{r}_t$.
       vi. Preprocess both new observations $o_{t+1}$ and $\tilde{o}_{t+1}$.
       vii. Store both experiences $e_t = (o_t, a_t, r_t, o_{t+1})$ and $\tilde{e}_t = (o_t, \tilde{a}_t, \tilde{r}_t, \tilde{o}_{t+1})$ in $M$.
       viii. **if** $t \% T' = 0$:
          A. Randomly sample from $M$ a minibatch of $N_e$ experiences $e_i = (o_i, a_i, r_i, o_{i+1})$.
          B. Set
             $$y_i = \begin{cases} r_i & \text{if the state } s_{i+1} \text{ is terminal} \\ r_i + \gamma Q(o_{i+1}, \arg\max_{a \in \mathbf{A}} Q(o_{i+1}, a; \theta); \tilde{\theta}) & \text{otherwise} \end{cases}$$
          C. Compute and clip the gradients based on the Huber loss $H(y_i, Q(o_i, a_i; \theta))$.
          D. Optimise the main DNN parameters $\theta$ based on these clipped gradients.
          E. Update the target DNN parameters $\tilde{\theta} = \theta$ every $\tilde{N}$ steps.
       ix. Anneal the $\epsilon$-Greedy exploration parameter $\epsilon$.
```

Chapter 5

Performance Assessment & Results

In this chapter, we will look into a reliable methodology to objectively assess the performance of algorithmic trading strategies including our novel TDQN algorithm.

5.1 Testbench

In various research, trading strategy performance is assessed on a single asset, such as a specific stock or index, over a given timeframe. This approach introduce bias as the trading data can be selected in such way that the strategy looks profitable, even when it is not the case in general. To mitigate this risk, it is essential to assess strategies across multiple stocks with different characteristics.
This project establishes a testbench of six stocks from two principal regions-the US and India-chosen to represent diverse market dynamics(Table 5.1). A unique trading strategy is trained **independently for each stock** while for the sake of generality, all the algorithm hyperparameters remains unchanged for all the stocks.
To evaluate our method's robustness, the selected training period spans from 2012 to 2019. Although data from 2016 to 2024 might include more recent events, the COVID-19 market crash would **disrupt model training** and introduce **significant volatility**, impairing model stability. Thus, our selected period provides a stable training environment that allow the model to effectively learn underlying patterns. This time period is divided into training and test set as follows:

- Training Set: 01/01/2012 to 31/12/2017
- Testing Set: 01/01/2018 to 31/12/2019

A validation set was also made using the training set for the optimal tuning of hyperparameters for our algorithm. Also, we have fixed the RL agent DNN parameter θ for the entire test set meaning no new points are incorporated for future training.

Sector/ Region	**US**	**India**
Technology	Apple (AAPL)	TCS (TCS)
Financial Services	JPMorgan Chase (JPM)	State Bank of India (SBIN)
Automotive	Tesla (TSLA)	Mahindra (M&M)

Table 5.1: Sector-wise Companies in US and India

5.2 Benchmark Trading Strategies

In order to assess the robustness and weakness of our algorithm, we need some benchmark strategies to compare with. Both passive and active strategies were considered for the comparison of our algorithm. We have used classical trading strategies for the comparison, also for the sake of fairness and correctness, all the reductions made in chapter 3 in the RL formulation were applied to these benchmark strategies as well i.e. these strategies also have same action or input($\mathbf{A}$) and observation or output space($\mathbf{O}$). Below are the following benchmark strategies used:

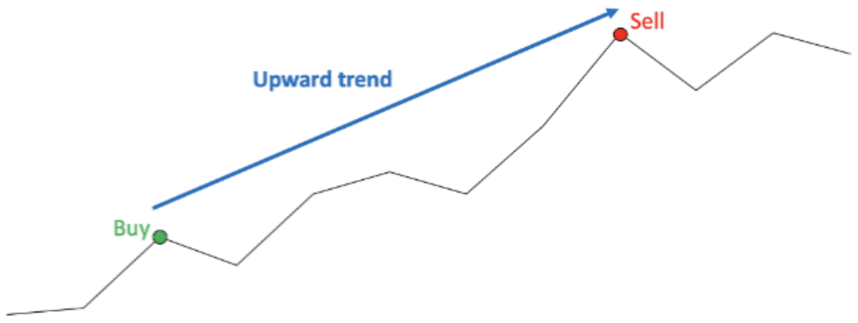


Figure 5.1: Trend following

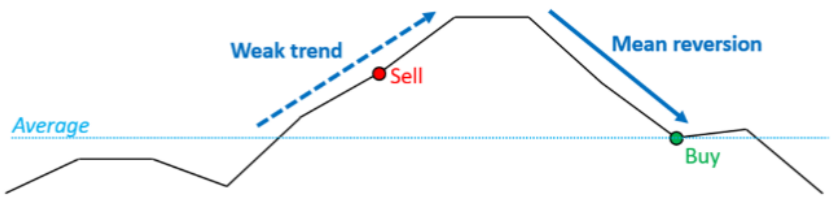


Figure 5.2: Mean reversion

- Buy & Hold (B&H)
- Trend following with moving averages(TF)
- Sell & Hold(S&H)
- Mean reversion with moving averages(MR)

The B&H and S&H strategies are passive strategies as there are no change in trading position over the trading horizon as their name suggests while the rest two are active strategies as multiple changes happen in trading positions over the trading horizon.

A trend following strategy(Figure 5.1) is a strategy that assumes stocks will continue moving in the same direction (up or down) for a period, capitalizing on sustained price movements while mean reversion(Figure 5.2) is based on belief that stock prices will revert to its historical average after some time period, exploiting deviations from this mean for potential profit. Hence we can see that if a trend following strategy is making profit then a mean reversion strategy will be in loss and vice-versa.

5.3 Performance Indicators

We define few performance indicators that statistically represent the success of an algorithmic trading strategy in order to assess its effectiveness. Profitability is the main goal of any trading strategy, thus its effectiveness should ideally match the amount of money it brings in. But only considering profitability sometimes leads to huge loss as it omits the risk associated with trading activity. Generally, a strategy with less profit but stable is better than a strategy with huge profit but without any risk management.
Multiple performance indicators is chosen to assess the trading strategy quantitatively. As we have discussed in Section 3.2.5, the Sharpe ratio is the most important metric which is widely recognized in algorithmic trading for its balanced view of returns relative to risk. Along with Sharpe ratio, we used few more performance indicators to provide more insight. Table 5.2 lists all the performance indicators used to evaluate the trading strategy's effectiveness.

Alongside these quantitative metrics, visualizing the strategy's behavior over time is valuable. We have plotted both stock market prices (p_t) and the portfolio values (v_t) evolutions with the trading actions (a_t) that helps in

Table 5.2: Performance indicators

Performance Indicator	**Description**
Sharpe Ratio	Return of the trading activity compared to its riskiness.
Profit & Loss	Money gained or lost at the end of the trading activity.
Annualised Return	Annualised return generated during the trading activity.
Annualised Volatility	Modelling of the risk associated with the trading activity.
Profitability Ratio	Percentage of winning trades made during the trading activity.
Profit and Loss Ratio	Ratio between the trading activity trades average profit and loss.
Sortino Ratio	Similar to the Sharpe ratio, but penalises only negative risk.
Maximum Drawdown	Largest loss from a peak to a trough during the trading activity.
Maximum Drawdown Duration	Time duration of the trading activity maximum drawdown.

the analysis of our trading strategy's decision-making process. This representation also helps in identifying the strengths, weaknesses, and patterns that enhances our understanding of the strategy's overall performance.

5.4 Results

In this section, we will evaluate our trading strategy using above performance methodology we described. We will perform deep analysis of 2-3 stocks from our testbench which gives good results and bad results, this analysis will help in understanding the strength, weakness and limitations of our algorithm and thus a scope for future to improve.

5.4.1 Good Results

We will give a detailed analysis of our trading strategy on two **Technology Sector stocks** from each region namely, **Tata Consultancy Services Ltd** & **Apple**.

Apple Stock

Metric	**B&H**	**S&H**	**TF**	**MR**	**TDQN**
Sharpe Ratio	1.239	-1.593	1.178	-0.609	1.572
Profit & Loss [$]	79823	-80023	68738	-34630	109638
Annualized Return [%]	28.86	-100.00	25.97	-19.09	34.52
Annualized Volatility [%]	26.62	44.39	24.86	28.33	25.70
Profitability Ratio [%]	100.00	0.00	42.31	56.67	60.00
Profit and Loss Ratio	∞	0.00	3.182	0.492	2.984
Sortino Ratio	1.558	-2.203	1.802	-0.812	2.080
Max Drawdown [%]	38.51	82.48	14.89	51.12	16.81
Max Drawdown Duration [days]	62	250	20	204	25

Table 5.3: Performance Comparison of B&H, S&H, TF, MR, and TDQN strategies for Apple Stock

Table 5.3 gives the performance of each trading strategy on the test set, given that we started with an initial capital of $100,000. As we can see from the table, the TDQN algorithm gives an impressive result from both earning & risk management point of view and clearly surpassing all benchmark active and passive trading strategies. Also, from Fig. 5.4 we can see the evolution of both the stock market price p_t and the portfolio value v_t of the RL agent with the actions a_t produced by the TDQN algorithm. This result indicate that a deep reinforcement learning (DRL) trading strategy can effectively identify and capitalizes on significant market trends but it becomes more cautious during periods of increasing volatility and market behavioral shifts.

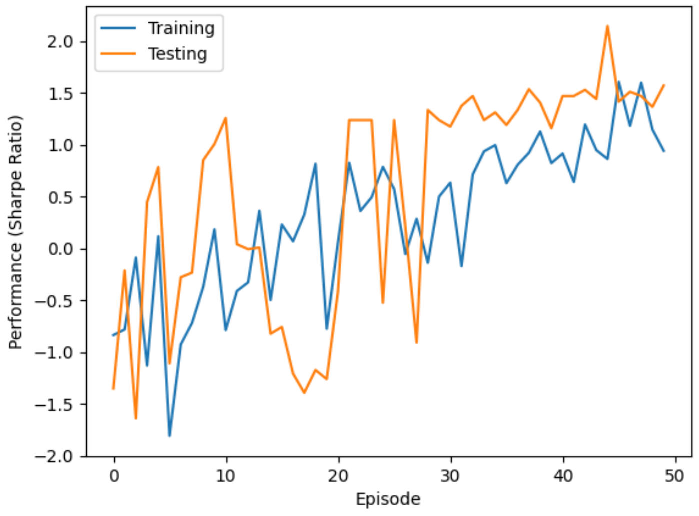


Figure 5.3: Training and Testing Performance of Apple Stock

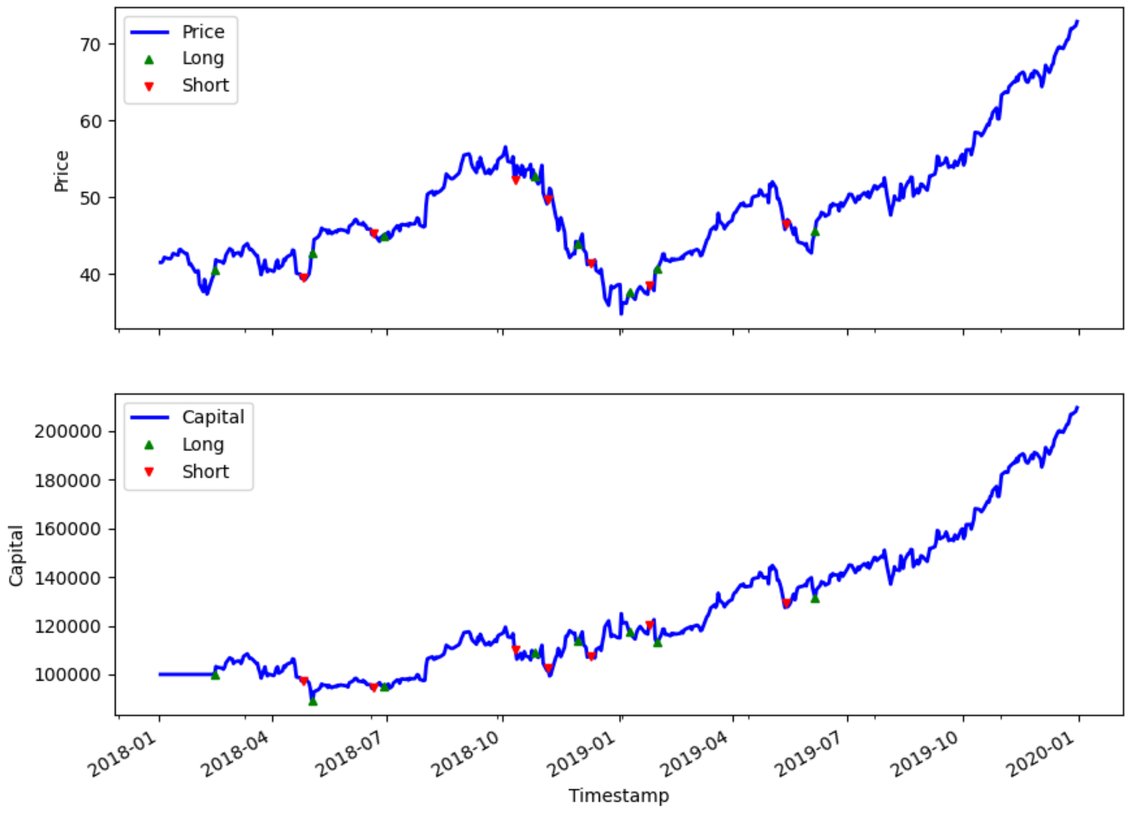


Figure 5.4: Price and portfolio value evolution with agent actions in the test set

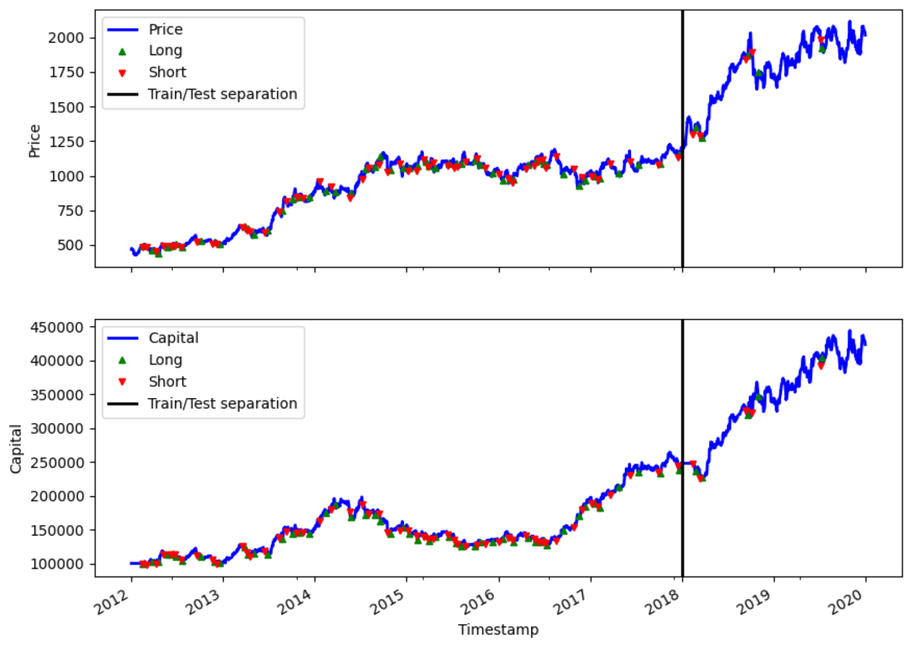


Figure 5.5: Complete trading activity of TCS stock

Tata Consultancy Services Ltd

Similarly, Table 5.4 gives the performance of TDQN on TCS stock for the test set, given that we started with an initial capital of INR 100000.

Performance Indicator	**TDQN**
Profit & Loss (P&L)	70774
Annualized Return [%]	25.97%
Annualized Volatility [%]	22.83%
Sharpe Ratio	1.322
Sortino Ratio	2.026
Maximum Drawdown [%]	12.61%
Maximum Drawdown Duration [days]	30 days
Profitability [%]	70.00%
Ratio Average Profit/Loss	3.096

Table 5.4: Performance Indicators of TDQN Algorithm for TCS

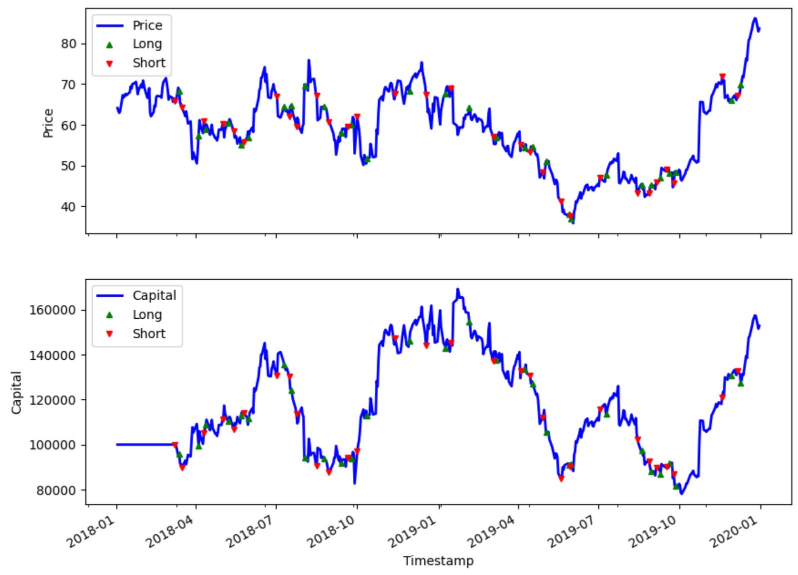
Figure 5.5, shows a complete plot of entire trading activity from training to testing, here a small trick is used to assert a continuity in the capital curve. We find the ratio of the last capital price from the training phase to the initial capital of the testing phase which is 100000, then the entire capital of the testing phase is raised by this ratio. This diagram is used for the comparision between the training and testing phase.

5.4.2 Mitigate Results

Here we will perform a same analysis of our trading strategy as above on a **Automotive Sector** stock **Tesla**, given that we have used $100000 as the initial capital. As you can see from Table 5.5, the sharpe ratio of Tesla

Performance Indicator	**TDQN**
Profit & Loss (P&L)	53466
Annualized Return	25.95%
Annualized Volatility	51.32%
Sharpe Ratio	0.569
Sortino Ratio	0.752
Maximum Drawdown	53.86%
Maximum Drawdown Duration	177 days
Profitability	46.67%
Ratio Average Profit/Loss	1.461

Table 5.5: TDQN Performance Indicators for Tesla



stock is very much less compared to the above stocks we have discussed. The poor results achieved by these benchmark trading strategies suggest that the Tesla stock is very difficult to trade, which can be seen from the table as well. The trading strategy presents a risk level that is likely unacceptable, with an extended maximum drawdown duration that could induce stress for operators. As shown in above figure, which tracks stock prices p_t, RL portfolio values v_t, and TDQN actions a_t, the volatility of Tesla's stock drives frequent trading position changes (where $a_t \neq a_{t-1}$), that amplifies the risks and costs.

5.4.3 Complete result for the test set

Sector	**US Stock**	**SR**	**Indian Stock**	**SR**
Technology	Apple	1.572	TCS	1.322
Finance	JP Morgan	0.722	SBI	0.635
Automotive	Tesla	0.569	Mahindra	0.818

Above results indicate that while our strategy generates profits, it demonstrates a particular strength in managing the technology sector. However, it faces challenges with the finance and automotive sectors due to their volatility and the risk of overfitting within a limited observation space. These factors constrain the strategy's broader applicability, highlighting areas for improvement that we aim to address in future iterations.

Chapter 6

Conclusion & Future work

The Trading Deep Q-Network (TDQN) algorithm showed some good results in algorithmic trading which shows a significant potential in outperforming the benchmark trading strategies on an average across the testbench. By leveraging deep reinforcement learning (DRL), the TDQN algorithm effectively adapts to complex market patterns and yields competitive Sharpe ratios on various stocks, including high-performing assets like Apple and TCS. However, the TDQN algorithm faced some challenges. One significant limitation is its sensitivity towards performance variance; repeated training under the same conditions can produce inconsistent outcomes. This inconsistency emphasizes the need for greater stability and reliability in the model's learning process. Also, the TDQN is prone to overfitting, particularly when it is applied to highly volatile stocks like Tesla, limiting its ability to generalize across diverse market conditions.

Furthermore, this algorithm's largely reactionary nature comes from its constrained observation space **O**. The absence of fundamental market data-such as economic indicators or sentiment analysis-means that TDQN may lag behind major market trends. This limitation underscores the value of broadening its observation parameters to enhance predictive accuracy.

Addressing these limitations presents several avenues for future research and development. One promising direction is the integration of advanced techniques, such as hybrid models that combine sentiment analysis with traditional financial metrics, which could enrich the TDQN's observational capabilities. Implementing transfer learning techniques could also improve the model's adaptability to changing market regimes, enhancing its robustness in dynamic environments.

Bibliography

[1] Yuxi Li. "Deep Reinforcement Learning: An Overview". In: *ArXiv* abs/1701.07274 (2017). URL: `https://api.semanticscholar.org/CorpusID:17540505`.

[2] Ernest P. Chan. *Quantitative Trading: How to Build Your Own Algorithmic Trading Business*. John Wiley Sons, 2009.

[3] Ernest P. Chan. *Algorithmic Trading: Winning Strategies and Their Rationale*. Hoboken, NJ: Wiley, 2013. ISBN: 978-1118460146.

[4] R. Narang. *Inside the Black Box: The Simple Truth About Quantitative Trading*. Wiley, 2009.

[5] A. Arévalo et al. "High-Frequency Trading Strategy Based on Deep Neural Networks". In: *International Conference on Intelligent Computing (ICIC)*. Springer, 2016, pp. 424–436. DOI: `10.1007/978-3-319-42291-6_37`.

[6] Wei Bao, Jun Yue, and Yulei Rao. "A deep learning framework for financial time series using stacked autoencoders and long-short term memory". In: *PLOS ONE* 12.7 (July 2017), pp. 1–24. DOI: `10.1371/journal.pone.0180944`. URL: `https://doi.org/10.1371/journal.pone.0180944`.

[7] F. D. Paiva et al. "Decision-Making for Financial Trading: A Fusion Approach of Machine Learning and Portfolio Selection". In: *Expert Systems with Applications* 115 (2019), pp. 635–655. DOI: `10.1016/j.eswa.2018.08.029`.

[8] J. Moody and M. Saffell. "Learning to Trade via Direct Reinforcement". In: *IEEE Transactions on Neural Networks* 12.4 (2001), pp. 875–889. DOI: `10.1109/72.935097`.

[9] M. A. H. Dempster and V. Leemans. "An Automated FX Trading System Using Adaptive Reinforcement Learning". In: *Expert Systems with Applications* 30 (2006), pp. 543–552. DOI: `10.1016/j.eswa.2005.10.018`.

[10] Y. Deng et al. "Fuzzy Recurrent Deep Neural Networks for Trading Without Technical Indicators". In: *Expert Systems with Applications* (2017).

[11] J. Carapuço, R. F. Neves, and N. Horta. "Reinforcement Learning Applied to Forex Trading". In: *Applied Soft Computing* 73 (2018), pp. 783–794. DOI: `10.1016/j.asoc.2018.09.034`.

[12] G. Jeong and H. Kim. "Improving Financial Trading Decisions Using Deep Q-Learning: Predicting the Number of Shares, Action Strategies, and Transfer Learning". In: *Expert Systems with Applications* 117 (2019), pp. 125–138. DOI: `10.1016/j.eswa.2018.09.009`.

[13] S. Almahdi and S. Yang. "A Constrained Portfolio Trading System Using Particle Swarm Algorithm and Recurrent Reinforcement Learning". In: *Expert Systems with Applications* 130 (2019), pp. 145–156. DOI: `10.1016/j.eswa.2019.04.004`.

[14] K. Lei et al. "Time-Driven Feature-Aware Jointly Deep Reinforcement Learning for Financial Signal Representation and Algorithmic Trading". In: *Expert Systems with Applications* 140 (2020). DOI: `10.1016/j.eswa.2019.112867`.

[15] H. Park, M. K. Sim, and D. G. Choi. "An Intelligent Financial Portfolio Trading Strategy Using Deep Q-Learning". In: *Expert Systems with Applications* 158 (2020), Article 113573. DOI: `10.1016/j.eswa.2020.113573`.

[16] John P. Ioannidis. "Why Most Published Research Findings Are False". In: *PLoS Medicine* (2005).

[17] D. H. Bailey et al. "Pseudo-Mathematics and Financial Charlatanism: The Effects of Backtest Overfitting on Out-of-Sample Performance". In: *Notices of the American Mathematical Society* 61.5 (2014), pp. 458–471.

[18] Thibaut Théate and Damien Ernst. “An application of deep reinforcement learning to algorithmic trading”. In: *Expert Systems with Applications* 173 (2021), p. 114632. ISSN: 0957-4174. DOI: `https://doi.org/10.1016/j.eswa.2021.114632`. URL: `https://www.sciencedirect.com/science/article/pii/S0957417421000737`.

[19] Adam Darmanin. *Deep Q-Learning Applied to Algorithmic Trading*. Accessed: October 30, 2024. 2024. URL: `https://medium.com/@adamdarmanin/deep-q-learning-applied-to-algorithmic-trading-c91068791d68`.

[20] Volodymyr Mnih et al. “Playing Atari with deep reinforcement learning”. In: *CoRR* abs/1312.5602 (2013).

[21] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI Press, 2016, pp. 2094–2100.