***Dynamic Algorithm Selection for Day Trading: A Deep Reinforcement Learning Approach to Maximize Return***

21AIE311 – REINFORCEMENT LEARNING

**Team 6**

Amruth A1, Ramanan R1, Rhea Paul1, Dr.Amudha J1

1 Dept. of Computer Science and Engineering, Amrita School of Computing, Bengaluru,

Amrita Vishwa Vidyapeetham, India

\*mail: team6@blr.amrita.edu

*Acknowledgement*: We would like to extend our heartfelt appreciation to everyone who helped carry out this project. We would also like to express our sincere gratitude to Dr. Amudha J for their exceptional guidance, expertise, and unwavering support throughout the duration of this project. Their invaluable insights, constructive feedback, and mentorship have been instrumental in shaping our research and enabling us to achieve meaningful outcomes.

*Abstract*: In this paper, the use of deep reinforcement learning in the stock market area is investigated. In particular, a Deep Q Network with Double DQN algorithm is used to train an agent to make trading decisions and select trading strategies using stock price information acquired from the Alpha vantage API. The agent's policy eventually develops the ability to choose activities that produce favorable rewards, leading to successful trading outcomes. The generated earnings, which show the trained model's capacity to make profitable trading decisions, are used to gauge its effectiveness. Examining the effect of increasing the number of training episodes on the profit/loss results, it is shown that there aren't many differences after a certain point. Furthermore, different reward functions that take into account risk and transaction costs, the use of advanced deep learning architectures, and the inclusion of new features to the agent's state representation are all possible future areas for this research.

*Keywords:* *Deep Reinforcement Learning, Stock Market, Trading Strategies, Double Deep Q Network*

# **Introduction**

The stock market is a very complicated, unpredictable environment that experiences regular volatility. For a very long time, traders and investors have tried to anticipate market movements and perfect the timing of their trades in order to increase profits and reduce losses. Deep reinforcement learning (RL), which is based on real-time market behaviour, has become a viable method for forecasting stock market trends as a result of recent developments in machine learning and artificial intelligence.

## Automated Stock Prediction

Finance and stock trading are essential domains that play a critical role in the global economy. Finance and stock trading facilitate capital formation to finance operations and expand operations. They also enable investors to hedge against financial risks and protect investments. Moreover, stock trading provides liquidity to the market, allowing investors to move their investments quickly and efficiently.

Algorithmic trading is a trading strategy that uses pre-defined rules and automated processes to execute trades in financial markets. In recent years, machine learning has been increasingly applied to algorithmic trading to improve its effectiveness. Machine learning algorithms can analyse vast amounts of data and identify patterns and trends that may not be apparent to human traders. These algorithms can be trained on stock market data to learn how the market behaves and develop predictive models that can be used to make trading decisions.

One of the main advantages of using machine learning in algorithmic trading is the ability to adapt to changing market conditions. The machine learning algorithms can continuously learn and adjust to new data, allowing the trading strategies to evolve and remain effective over time.

## Need for the problem

The need for better stock prediction algorithms arises from the fact that the stock market is a highly dynamic and complex system that is affected by numerous factors, such as political events, economic indicators, and company-specific news. This complexity makes it difficult to predict future stock prices accurately, and investors often struggle to make informed trading decisions. These algorithms will therefore enable investors to make informed decisions and minimize their risks while maximizing their profits. Moreover, with the increasing use of automated trading systems and the growth of quantitative trading, there is a greater need for accurate and reliable stock prediction algorithms that can inform these systems' decision-making processes. Overall, better stock prediction algorithms can bring significant benefits to investors, traders, and financial institutions, including reduced risks, increased profitability, and improved market efficiency.

## Objective

Using deep reinforcement learning, we are working to create the best system for forecasting stock market movements. We'll combine tried-and-true RL algorithms with fresh, self-optimized Kalman filter-based techniques to minimise noise and boost performance in order to accomplish this goal. In order to identify which algorithm performs the best utilising Reinforcement Learning techniques, we will compare and assess the performance of these algorithms on actual stock market data.

The objective of this research is to provide a thorough review of cutting-edge methods for utilising deep reinforcement learning to predict stock market trends and to determine the most efficient strategy for optimising technical trading timing for the constantly changing stock market behaviour. We aim to contribute to the expanding body of research in this field and provide valuable insights for investors and traders looking to enhance their trading success by gaining a better grasp of how deep RL can be used to forecast stock market trends.

## Problem Statement

Our problem statement involves finding the best trading strategy so that the stock trader obtains the highest profit for the given market. It involves developing a Deep Reinforcement Learning (DRL) model for dynamic algorithm selection which includes buy&hold, SMA(10,30), SMA(50,200), and KF-SMA, for usage in Financial Market.  Q-Learning model called DDQN model will be used to achieve this.

# **Related Work**

[1] This study presents a theory of deep reinforcement learning for stock trading decisions and stock price prediction. Experimental data demonstrate the model's availability and reliability, and it is compared to the conventional model to demonstrate its superiority. This research demonstrates the viability of deep reinforcement learning in financial markets as well as the legitimacy and benefits of strategic decision-making from the viewpoints of stock market forecasting and intelligent decision-making mechanisms. Use of Deep Reinforcement Learning in stock prediction has gained popularity over the recent years. From this paper, we have understood that using DRL has been shown to outperform common Machine Learning strategies in predicting market trends and making profitable trades. By incorporating feedback and rewards into the learning process, DRL algorithms are able to learn from their mistakes and continuously improve their predictions. As a result, DRL is seen as a promising approach for developing more effective trading algorithms. We plan to extend this idea for optimizing the time and effort taken to choose a trading strategy using DRL technique. [2] Deep learning models have seen success in the stock market prediction field recently and are frequently used. The majority of these deep learning-based models, however, use supervised learning techniques and are unable to handle long-term objectives. The stock market trading model proposed in this research is based on deep reinforcement learning and is suited for forecasting stock price changes and stock transactions. The superiority of the proposed model is assessed by stock price trend predictions and trading on a sample of randomly chosen equities and stock market indexes. Results of the experiments show that the model outperforms baseline techniques on a number of variables. One of the aspects of this study shows the comparison of the model with different baseline models, which shows that DDQN (Double Deep Q -Network) far outperforms the other baseline models. Therefore, we are extending the use of this policy onto our study by using DDQN as our Agent and Policy to achieve the desired results. [3] This paper discusses the difficulty of predicting time series data, particularly in the stock market where small fluctuations in various areas can have a significant impact. The study confirms that results are influenced by the data set and state size, which is predicted by the closing price of several days. Using a modified algorithm of Q-value based on DQN (DDQN) with regularization and LSTM, the experiment shows that the combination of LSTM and DDQN is better than only DQN and full connection layer under certain preconditions. The only indicator used is total profit, and the closing price is used to predict. The study highlights the practical significance of stock prediction and the importance of appropriate algorithms and parameters.

# **RL Problem Formulation**

In this section, we go through the flow of modelling our problem statement as a Markov Decision Process (MDP). We also take into account the considerations and assumptions while modelling this.

## Environment

The environment we have custom designed called MarketEnvironment includes live market dataset for specific company stocks for a whole trading day. The states, internal states and definition for each action is also defined in our environment. The reward function as well as stopping condition for each episode is also defined.

## State

The state in our problem represents the current market conditions or features that the agent observes. The observed states include three elements.

1. Previous price (P): The closing price of the stock at the previous time step.
2. Current price (C): The closing price of the stock at the current time step.
3. Next price (N): The closing price of the stock at the next time step.

With these states, we further calculate 3 internal states Low, Neutral and High which is calculated from the Current Price and the Previous Price.

## Agent

The class DDQNAgent defines the agent. It is a Double Deep Q-Network (DDQN) agent implementation. A neural network is used by the reinforcement learning method DDQN to approximatively calculate the Q-values (action values) for each state-action pair. The agent interacts with its surroundings, gains knowledge from its mistakes, and gradually develops better policies.

## Action

The actions are defined by four market trading strategies that the agent will implement.

1. Buy & Hold: Holds the stock without taking any actions.
2. SMA(10,30): Uses the Simple Moving Average (SMA) crossover strategy with two window sizes (10 and 30). It compares the short-term SMA to the long-term SMA and decides whether to buy or sell the stock.
3. Bollinger Bands: Uses the Bollinger Bands indicator to determine whether the current price is above the upper band or below the lower band. Based on this, it takes appropriate actions.
4. KF-SMA: Implements a Moving Average Crossover Strategy along with Kalman Filter to reduce noise using short and long-term moving averages. It compares the short-term moving average to the long-term moving average and decides whether to buy or sell the stock.

## Reward

Reward will only consist of a positive / negative reinforcement value. Given in the below fig() is the reward function for our formulation.

A diagram of a diagram

Description automatically generated with low confidence

Fig 2. Reward Function

# **LITERATURE SURVEY**

## Summary of the work

This research proposes the application of Reinforcement Learning techniques in stock market trading and Trading strategy selection. The research introduces the concept of using continually changing trading strategies to obtain the best profit for the constantly varying market trends. Our action includes a novel trading strategy called Kalman Filtered SMA which has shown promising results while performing in equity markets while compared to other common trading strategies such as SMA, Bollinger Bands, etc. It also develops upon the concept of using Double Deep Q Networks for stock prediction. As seen in [5], DDQN has shown promising results for stock market prediction where there exists a lot of variation in the environment.

## Gaps and Challenges

1. Extreme Volatility and Complexity

* The stock market's frequent changes and erratic behaviour make precise forecasting challenging.
* A significant amount of data must be processed and analysed in real-time, necessitating advanced algorithms and computer power.

2. Lack of Common Evaluation Metrics

* There is a lack of common evaluation metrics to compare the effectiveness of various RL algorithms.
* It is challenging to identify the best algorithm and to fairly compare its performance to others.

3. Utility in Real-World Scenarios

* The utility of deep RL in forecasting stock market trends and maximizing trading timing in real-world scenarios needs further study.
* The majority of studies in this field have used historical data and vectorized back testing methodologies, making it unclear how effectively these algorithms work in actual market situations.

# **DESIGN**

## Decision 1: Selecting a value function to approximate

**Decision Made**: We chose to approximate the action-value function Q(s,a;θ)

**Justification**: Approximating the action-value function allows the agent to learn the expected return for taking a particular action a*a* in a given state s*s*. This is particularly useful in stock trading where the agent must decide which trading strategy to use in different market conditions.

**Evidence from Code**: In the **DDQNAgent** class, the **act** method predicts Q-values for each action in a given state, which suggests that the action-value function Q(s,a;θ) is being approximated.

def act(self, state):

if np.random.rand() <= self.epsilon:

return np.random.choice(self.action\_size)

state = np.reshape(state, (1, self.state\_size, 1))

q\_values = self.model.predict(state)

return np.argmax(q\_values[0])

## Decision 2: Selecting a neural network architecture

**Decision Made**: The architecture comprises LSTM layers followed by Dense layers. State in value out architecture.

**Justification**: LSTM layers are effective for sequence-based data, such as time-series stock prices. They capture the temporal dependencies which are crucial for making trading decisions. The Dense layers help in the approximation of the action-value function.

**Evidence from Code**: The **build\_model** method in the **DDQNAgent** class reveals the architecture used for approximating the action-value function. The use of LSTM layers is particularly suitable for sequence-based data like stock prices. Aka time series data

def build\_model(self):

model = tf.keras.Sequential()

model.add(tf.keras.layers.LSTM(32, input\_shape=(self.state\_size, 1), return\_sequences=True))

model.add(tf.keras.layers.LSTM(64, return\_sequences=False))

model.add(tf.keras.layers.Dense(32, activation='relu'))

model.add(tf.keras.layers.Dense(self.action\_size, activation='linear'))

model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=self.learning\_rate))

return model

## Decision 3: Selecting what to optimize

**Decision Made**: The optimization goal is to minimize the loss between the predicted Q-values and the target Q-values.

**Justification**: By minimizing this loss, the agent learns to make better approximations of the action-value function, which is essential for making more accurate trading decisions.

**Evidence from Code**: In the **replay** method, the target is computed using the equation Target=r+γmaxQ(s′,a′;θ)

targets[np.arange(len(targets)), actions] = rewards + self.gamma \* np.amax(next\_q\_values, axis=1) \* ~dones

## Decision 4: Selecting the targets for policy evaluation.

**Decision Made**: Temporal-Difference (TD) target is used for policy evaluation, specifically an off-policy TD target.

**Justification**: TD methods are well-suited for problems where the full model of the environment is unknown and only samples are available, which is often the case in stock trading.

**Evidence from Code**: The same **replay** method indicates the use of TD target for policy evaluation. The target value for each state-action pair is updated based on the maximum Q-value for the next state, which is characteristic of off-policy learning.

targets[np.arange(len(targets)), actions] = rewards + self.gamma \* np.amax(next\_q\_values, axis=1) \* ~dones

## Decision 5: Selecting an exploration strategy

**Decision Made**: Epsilon-Greedy Strategy with a 20% chance of random action during training.

**Justification**: The epsilon-greedy strategy ensures sufficient exploration of the state space while allowing the agent to exploit its current knowledge most of the time. This balance is crucial for the agent to discover profitable trading strategies.

**Evidence from Code**: The **act** method in the **DDQNAgent** class employs an epsilon-greedy strategy. The agent chooses a random action with a probability of **epsilon** (set to 0.2) and the action that maximizes the Q-value with a probability of 1−ϵ.

def act(self, state):

if np.random.rand() <= self.epsilon:

return np.random.choice(self.action\_size)

state = np.reshape(state, (1, self.state\_size, 1))

q\_values = self.model.predict(state)

return np.argmax(q\_values[0])

## Decision 6: Selecting a loss function

**Decision Made**: Mean Squared Error (MSE)

**Justification**: MSE is a commonly used loss function for regression problems. It effectively penalizes large errors in prediction, which is desirable for our stock trading application.

**Evidence from Code**: The model is compiled with the MSE loss function. This measures the average of the squares of the differences between the estimated and target Q-values.

Def build\_model(self):

model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=self.learning\_rate))

## Decision 7: Selecting an optimization method

**Decision Made**: Mini-batch Gradient Descent with Adam Optimizer

**Justification**: The Adam optimizer combines the advantages of two other extensions of stochastic gradient descent: AdaGrad and RMSProp. In the fast-paced and volatile stock market, using a robust optimization algorithm like Adam can be beneficial.

**Evidence from Code**: The **replay** method samples a mini-batch from the memory and fits the model on this mini-batch. The Adam optimizer is used for this purpose, as specified in the **build\_model** method.

def replay(self, batch\_size):

minibatch = random.sample(self.memory, batch\_size)

#... (Rest of the code)

self.model.fit(states, targets, epochs=1, verbose=0)

def build\_model(self):

model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=self.learning\_rate))

# **System Architecture**

## High Level Architecture

In this study, a trading agent with a reinforcement learning foundation is applied to the stock market. There are various important parts that make up the architecture. A DDQN Agent serves as the reinforcement learning agent at the system's heart. For each state-action pair, this agent uses a neural network model created with TensorFlow to approximate the Q-values. The stock market environment is interacted with by the agent. The environment retrieves stock price information via the Alpha Vantage API, gives the agent access to the current state, manages the execution of actions, and determines incentives in accordance with the actions taken by the agent. The agent can use a variety of trading strategies in the environment, including Buy & Hold, SMA (Simple Moving Average), Bollinger Bands, and KF-SMA (Kalman Filtered Simple Moving Average Crossover). The agent stores and randomly samples previous events to update the neural network model as it learns and refines its policy over the course of numerous episodes. The training loop carries out the training procedure, going through each episode iteratively and interacting with the environment to gather experience and modify the model parameters for the agent. The study also incorporates hyperparameter tweaking logic to identify the optimum configuration by experimenting with various learning rates, gamma values, and epsilon values. The architecture is typical of RL, with distinct classes for the agent and environment, allowing for easier interaction, learning, and decision-making in the context of the stock market.

A picture containing text, screenshot, font, design

Description automatically generated

Fig 3. High Level System Architecture Flowchart

## DRL Architecture

The research implements a Deep Reinforcement Learning (DRL) architecture for stock market trading, which blends deep learning with reinforcement learning methods. The prior, present, and future stock prices make up the state representation at the beginning of the architecture. This state is fed into a neural network model created with TensorFlow, which comprises Dense layers to approximatively represent Q-values for each state-action pair and LSTM layers to capture temporal dependencies. When choosing actions, the agent balances exploitation and exploration using an epsilon-greedy policy. To eliminate temporal correlations, experiences are kept in a replay memory. The Double Deep Q-Learning algorithm then updates the model's weights by sampling from the replay memory. The agent can interact with the environment, gather data, and update its model while the training loop iterates through episodes. By experimenting with various learning rate, gamma, and epsilon combinations, hyperparameter tweaking is done to determine the best-performing agent configuration. The agent can learn from previous stock price data and the incentives that resulted by combining deep learning techniques with reinforcement learning ideas in the DRL architecture.

A picture containing text, screenshot, font, number

Description automatically generated

Fig 4. Neural Network Architecture

# **References**

[1] Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. 2021. Deep reinforcement learning for automated stock trading: an ensemble strategy. In Proceedings of the First ACM International Conference on AI in Finance (ICAIF '20).

[2] Liu, Xiao-Yang, FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance (November 25, 2020). Available at SSRN: https://ssrn.com/abstract=3737257 or <http://dx.doi.org/10.2139/ssrn.3737257>

[3] Li, Y., Ni, P. & Chang, V. Application of deep reinforcement learning in stock trading strategies and stock forecasting. Computing 102, 1305–1322 (2020). <https://doi.org/10.1007/s00607-019-00773-w>

[4] Gil Cohen “Artificial Intelligence in Trading the Financial Markets” in International Journal of Economics and Business Administration, volume X, issue 1, 2022.

[5] Yong Shi, Wei Li, Luyao Zhu, Kun Guo, Erik Cambria . Stock trading rule discovery with double deep Q-network, Applied Soft Computing (2021)