Emergent Trading Strategies from Deep Reinforcement Learning Models in Stock Market Trading

Mason James Wheeler MWHEEL@UW.EDU
Electrical & Computer Engineering
University of Washington
Seattle, WA 98105, USA

Abstract

In this paper, we explore the application of Deep-Q-Networks (DQN) models to devise optimal trading strategies in the stock market. We present a novel approach that leverages the power of DRL learning to navigate the complex and dynamic landscape of financial markets. Our methodology involves the creation of a custom trading environment, where the agent learns to buy and sell shares based on the observed market data. The agent's actions are guided by an epsilon-greedy strategy, which balances the trade-off between exploration and exploitation. We also introduce a replay memory mechanism that stores and samples transitions, facilitating the learning process. Our results demonstrate the efficacy of DRL models in developing robust trading strategies that adapt to market fluctuations. This research contributes to the growing body of literature on the application of machine learning in finance and opens new avenues for the use of DRL in decision-making tasks in dynamic and uncertain environments.

1 Introduction

The advent of algorithmic trading has significantly transformed the financial markets, now accounting for a substantial portion of daily trading activities. This shift towards automation has opened the door for the application of sophisticated machine learning techniques, such as Deep Reinforcement Learning (DRL), in the realm of trading. In this research, we delve into the application of DRL models, specifically Deep Q-Networks (DQN), to devise optimal trading strategies in the stock market. The stock market's dynamic and intricate nature presents a challenging environment for machine learning models. However, DQN Agents have shown the ability to learn optimal policies in an end-to-end fashion, and therefore we hypothesize may be well-suited to navigate this complexity. Our methodology involves constructing a custom trading environment where a DRL agent, utilizing a DQN, learns to make trading decisions based on observed market data. This data includes various indicators along with price and volume action. The agent's actions are guided by an epsilon-greedy strategy, which strikes a balance between exploration of new actions and exploitation of known information. To facilitate the learning process, we implement a replay memory mechanism. This mechanism stores and samples transitions, enabling the agent to learn from past experiences and improve its future decisions. A unique aspect of our work is the use of locally trained, smaller-sized models. Despite their compact size, we demonstrate that these models, when trained on a large dataset, can generate emergent successful trading strategies. This challenges the prevailing perception that only large, complex models can achieve success in financial markets. Our findings suggest that DQN, a relatively simple yet powerful DRL technique, can be an effective tool for developing intelligent trading strategies.

2 Methodology

In this section, we outline the methodology employed in our research, including the data used, the machine learning model implemented, and the key components of our custom trading environment.

2.1 Data

The data used in this study comprises historical stock market data, including various indicators along with price and volume action. This data is obtained using the get_stock_data function, which fetches intraday data for a given stock symbol and interval. The get_stock_data function further enriches this data by calculating various technical indicators such as Simple Moving Average (SMA), Exponential Moving Average (EMA), Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Bollinger Bands, Average Directional Index (ADX), Commodity Channel Index (CCI), Aroon, On Balance Volume (OBV), and Stochastic Oscillator. These indicators provide a comprehensive view of market conditions at any given time, allowing our model to make informed trading decisions. Ultimately the data was shaped (128, 32) where we gathered 128 minutes of our 32 input-features.

2.2 Deep-Q-Network

The machine learning model used in this study is a Deep Reinforcement Learning (DRL) model, specifically a Deep Q-Network (DQN). The DQN is implemented as a recurrent neural network (RNN) with two layers, followed by a series of dense layers. The RNN layers capture the temporal dependencies in the data, while the dense layers allow for a non-linear transformation of the features. Dropout is applied after each layer to prevent overfitting. The architecture of our DRL model is designed to handle the complexity and variability of stock market data.

2.3 StockEnvironment Class

The StockEnvironment class is a key component of our methodology. It simulates the trading environment, allowing the DRL agent to interact with the market and learn from its actions.

- Action Space and Observation Space: The action space of the StockEnvironment class consists of eleven possible actions, ranging from holding the current position to buying or selling a certain percentage of shares up to 25% of maximum value. The observation space includes the current stock price, volume, and various market indicators.
- Reward Function: The reward function is designed to encourage profitable trading decisions. It calculates the reward based on the change in portfolio value resulting from the agent's actions. In our work we defined it as $\Delta(Portfolio\ Value) * Scalar$
- Step Function: The step function simulates trading actions. It updates the agent's portfolio based on the action taken and returns the new state, reward, and a flag indicating whether the trading period has ended.

2.4 DQN Class

The DQN class implements the DRL model used in this study. It is a subclass of the PyTorch nn.module class and defines the architecture of the model. The forward method of the DQN class describes how the model processes its inputs and generates its outputs. In summary, our methodology combines a custom trading environment, a DQN model, and an epsilon-greedy strategy to create a system capable of learning trading strategies from historical stock market data.

3 Implementation

The implementation of the trading environment and the optimization of the trading strategy involves several classes and functions that interact with each other. We provide a detailed walkthrough of the code, focusing on the meaning of the training process in the context of financial data.

3.1 Deep Q-Network (DQN) Training Process

The core of the DQN algorithm is the Q-function, which estimates the expected return for each action in each state. In the context of financial trading, an action could be buying, selling, or holding a particular stock, and the state could be the current market conditions represented by various financial indicators. The Q-function is represented by a neural network (the Q-network), which is trained to minimize the difference between its predictions and the target Q-values. The target Q-values are calculated based on the Bellman equation, which states that the value of the correct financial choice at any given moment (the Q-value of a state-action pair) is equal to the immediate profit (reward) plus the discounted maximum profit (maximum Q-value) in the following states. This equation captures the trade-off between immediate and future profits

Algorithm 1 Deep Q-learning with Experience Replay in Stock Trading Environment

```
Initialize replay memory $D$ to capacity $N$. This memory will store past experiences of the agent in the form of stock data, action, reward, and next stock data.
Initialize action-value function Q with random weights. This function, parameterized by a neural network (the Q-network), estimates the expected future rewards for each possible action in each possible state.
**for** each episode from 1 to $M$ (where $M$ is the total number of trading periods):
    Initialize the state $s_1$ to the initial stock prices and other relevant financial indicators and preprocess this sequence to obtain $\varphi_1$.
    **for** each time-step t in the trading period:
        With probability $\varepsilon$, select a random action at (e.g., buy, sell, or hold a certain number of shares).
        Otherwise, select the action at that maximizes the estimated Q-value, at = $max_a\ Q^*(\varphi(s_t), a;\ \theta)$
        Execute action at in the Stock Environment class and observe the reward rt (e.g., the change in portfolio value) and the new state $x_{t+1}$(e.g., the new stock prices and other financial indicators).
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess this to obtain $\varphi_{t+1}$
        Store the transition $(\varphi_t, a_t, r_t, \varphi_{t+1})$ in the replay memory $D$.
        Sample a random minibatch of transitions $(\varphi_j, a_j, r_j, \varphi_{j+1})$ from the replay memory $D$.
        If $\varphi_{j+1}$ is a terminal state (i.e., the trading period has ended), set $y_j = r_j$. Otherwise, set $y_j = r_j + \gamma\ max_{a'}\ Q(\varphi(s_t), a;\ \theta)$, where $\gamma$ is the discount factor for future rewards.
        Perform a gradient descent step to minimize the loss $(y_j - Q(\varphi_j, a_j;\ \theta))^2$, which represents the difference between the predicted and actual future rewards.
    **end for**
**end for**
```

3.2 Intuition Behind DQN

The DQN algorithm is based on the idea that the optimal policy (the strategy that maximizes the expected return) can be derived from the optimal Q-function (the function that gives the maximum expected return for each state-action pair). By approximating the Q-function with a neural network and updating the weights based on observed rewards and next states, the agent can learn to predict the expected return for each action and thus choose the best action in each state. The use of experience replays and fixed Q-targets addresses two major challenges in training a Q-network:

- Correlations between consecutive experiences can lead to overfitting and instability. By storing experiences in the replay memory and sampling random batches, the agent can learn from a diverse set of experiences and avoid overfitting to recent experiences.
- The use of the same network to calculate both the predicted and target Q-values can lead to harmful feedback loops and divergence.

By using a separate target network to calculate the target Q-values, the agent can avoid these issues and achieve more stable learning. In the context of financial trading, this means that the agent learns to make trading decisions that not only maximize immediate profit but also consider the potential future profits. The agent learns from past trading experiences, but it also explores new trading strategies. This balance between learning from the past and exploring new possibilities is crucial for successful trading in the ever-changing financial markets.

4 Process and Results

In this section, we present an analysis of our experiments, showcasing the performance of the Deep Q-Network (DQN) agent in comparison to the Buy-and-Hold strategy. This includes an examination of the DQN agent's ability to boost portfolio value and an exploration into the implications of the agent's trading behavior for algorithmic trading.

4.1 Performance Metrics Definition

We evaluated the performance of our DQN agent primarily through the final portfolio value. This measure, encompassing both the cash in hand and the market value of the shares the agent held at the close of the trading period, captures the risk and return associated with the trading strategy. It's important to note that higher returns often come accompanied by higher risk, hence this metric allows us to study the agent's risk-return trade-off. We used the Buy-and-Hold strategy as a benchmark for comparison, a well-regarded approach for long-term investment, which entails purchasing a set number of shares at the onset and retaining them until the conclusion of the trading period.

4.2 Experimental Setup

In each experiment, we allocated $10,000 and 100 shares of a particular stock to the model. Subsequently, we processed the most recent month of intraday data for the given stock. Further, each DQN is initialized with untrained weights and zero memories at the beginning of the evaluation process.

4.3 Evaluation Criteria

Our effectiveness measure of the model’s ability focused on a term known as alpha, we are using this term loosely and not in the strict sense including the risk-free rate. In our experiment we define the alpha as: $\alpha = R_{DQN} - R_{B\&H}$ where R_x is the return of the a given portfolio x over an experimental time period.

4.4 Results and Insights

We observed that the DQN agent's performance varied across different market phases. In upward trending market conditions, the DQN agent was adept at capitalizing on its position to augment returns. Contrarily, during periods of heightened downward volatility, the agent swiftly de-leveraged its position, mitigating potential losses. The graph’s below overlayed on the stock price chart vividly demonstrates this behavior. In the upcoming sections, we will provide detailed graphs and tables to better illustrate these findings.

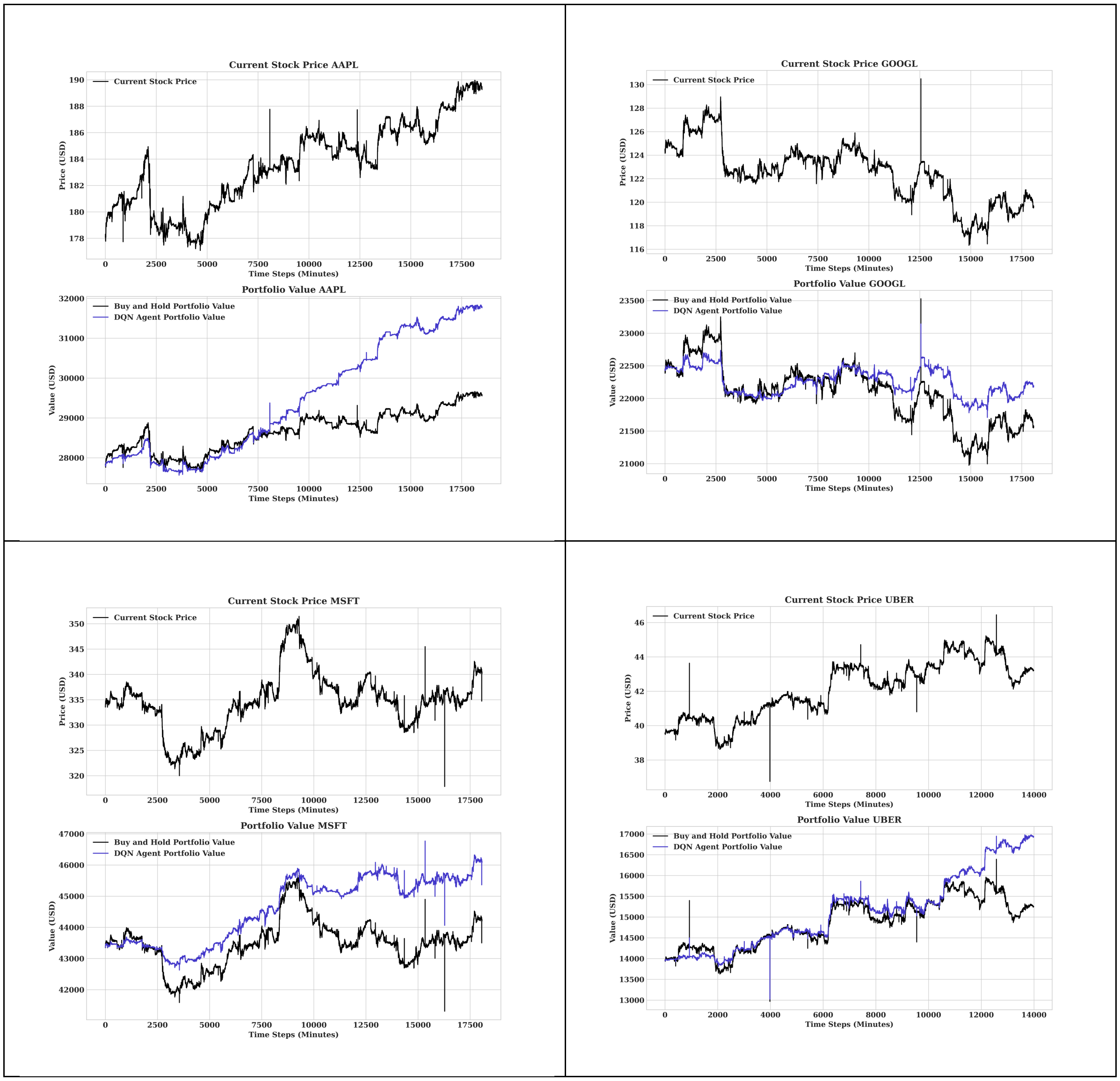

Figure 1. **Comparative performance of the DQN agent and the Buy-and-Hold strategy for multiple stocks over the experimental time period June 1, 2023 – July 1, 2023**

Stock	Alpha (%)	Buy & Hold Return (%)	DQN Return (%)
AAPL	7.48	6.28	14.2
GOOGL	2.87	-3.93	- 1.17
MSFT	4.31	2.00	6.31
UBER	12.02	9.28	21.3
Average	**6.67**	**3.40**	**10.16**

Table 1. **Comparative Returns and Alpha**

4.5 Discussion

The superior performance of the DQN agent can be attributed to its ability to learn complex trading patterns and adapt its strategy to maximize the expected return. The agent demonstrated the capability to make judicious trading decisions, such as selling at highs and buying during lows. This is particularly evident in the graph showing the agent's decisions superimposed on the stock price chart, where the agent's buy and sell actions align well with the lows and highs of the stock price, respectively. Interestingly, the agent also displayed an effective risk management strategy. It refrained from trading during periods of extreme volatility, where the risk of loss is high, and accumulated shares during periods of low volatility, where the risk is lower. This behavior suggests that the agent learned to balance the potential reward of a trade against the associated risk, a key aspect of successful trading. Moreover, the agent's ability to quickly de-leverage during periods of downward volatility is a significant finding. This strategy of reducing exposure to risky assets during market downturns is a well-known risk management technique in finance, and it's encouraging to see that the DQN agent has learned to apply it effectively.

4.6 Comparison with Other Methods

When compared to the Buy-and-Hold strategy, the DQN agent achieved a higher final portfolio value, indicating that it was able to generate a higher return on investment. This result underscores the potential of reinforcement learning methods, such as DQN, to outperform traditional trading strategies under certain market conditions. However, it's important to note that the performance of a trading strategy can be highly dependent on the specific market conditions during the trading period. Therefore, further testing is needed to evaluate the performance of the DQN agent over different time periods and in different market conditions. In future work, it would be interesting to compare the DQN agent's performance with other machine learning-based trading strategies, such as those based on decision trees, support vector machines, or other types of neural networks. This would provide a more comprehensive understanding of the strengths and weaknesses of different machine learning approaches to optimizing portfolio allocation.

5 Concluding Remarks

Our research has demonstrated the potential of Deep Reinforcement Learning (DRL), specifically Deep Q-Networks (DQN), in devising effective trading strategies in the stock market. The DQN agent, trained in a custom trading environment, was able to learn and adapt to complex trading patterns, outperforming the traditional Buy-and-Hold strategy in terms of return on investment. The agent's superior performance can be attributed to its ability to make judicious trading decisions, such as buying during lows and selling at highs. It also demonstrated an effective risk management strategy, refraining from trading during periods of extreme volatility and accumulating shares during periods of low volatility. This behavior suggests that the agent learned to balance the potential reward of a trade against the associated risk, a key aspect of successful trading. Moreover, the agent's ability to quickly de-leverage during periods of downward volatility is a significant finding. This strategy of reducing exposure to risky assets during market downturns is a well-known risk management technique in finance, and it's encouraging to see that the DQN agent has learned to apply it effectively. As shown in Table 1, the DQN agent achieved a higher final portfolio value compared to the Buy-and-Hold strategy, indicating that it was able to generate a higher return on investment with an average excess return of 6.67% in a single month.