Trading Strategy based on Reinforcement Learning

Ajaz Ahmad, Raveendra Pujari
Department of Computer Science
University of South Dakota
South Dakota, USA
{ajaz.ahmad, raveendra.pujari}@coyotes.usd.edu

***Abstract*—This paper presents a reinforcement learning (RL) framework for automated stock trading, leveraging Deep Q-Networks (DQN) to optimize trade operations. Unlike conventional methods, the developed agent learns directly from market data and incorporates a novel *performance-based model selection* mechanism that archives only the highest-profit policy during training. To mitigate action noise, we visualize trading signals exclusively from the best-performing episode. The system is integrated with an interactive Streamlit dashboard for real-time monitoring. Experimental results over 10 episodes and 6 months of data demonstrate the model's ability to adapt and generate profit under real-world constraints.**

***Index Terms*—Reinforcement Learning, Algorithmic Trading, Deep Q-Learning, Financial Markets, Model Selection, Streamlit**

I. Introduction

Algorithmic trading faces significant challenges due to non-stationary market dynamics and high noise-to-signal ratios. While supervised learning methods struggle with sequential decision-making, RL offers a natural framework for trading agents to learn through trial and error. This work advances prior research in three key aspects:

- A **dynamic state representation** using raw price and volume differences to capture market momentum.
- A **profit-aware model checkpointing** system that discards suboptimal policies during training.
- An **interpretable action visualization** pipeline that filters out noisy trades by focusing on the top-performing episode.

Additionally, our approach is designed to be extensible and reproducible, allowing easy deployment for educational or production-level environments. Through hands-on experimentation and visualization, the strategy can be interpreted and evaluated without deep technical intervention.

II. Related Work

RL-based trading has evolved significantly since Moody and Saffell's pioneering work [2]. Recent advances include:

- **DQN extensions**: Mnih et al. [1] demonstrated deep RL's potential in high-dimensional spaces, inspiring applications in finance.
- **Adversarial training**: Liang et al. [3] improved robustness using adversarial market simulations.
- **Multi-agent systems**: Concurrent work has explored competitive environments for portfolio optimization.

Other noteworthy work includes the use of Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and recurrent architectures such as LSTMs to capture long-term dependencies in price series. These techniques aim to enhance adaptability and improve learning efficiency across diverse market conditions.

III. Problem Formulation

We formulate the stock trading problem as a Markov Decision Process (MDP), defined by the tuple (S, A, P, R, γ), where:

- S: Set of states representing a time series window of OHLCV features.
- A: Discrete action space with three choices {Buy, Sell, Hold}.
- P: Transition probabilities between states (implicitly learned).
- R: Reward function based on realized trading profit.
- γ: Discount factor controlling future reward prioritization ($\gamma = 0.95$).

This formulation enables the agent to evaluate the consequences of its actions over time, balancing short-term returns with long-term growth strategies.

IV. Methodology

A. Data Preprocessing

Given raw OHLCV data $X_t = (O_t, H_t, L_t, C_t, V_t)$, we compute:

$$\Delta X_t = X_t - X_{t-1} \quad \text{(raw differences)} \tag{1}$$

A sliding window of $w = 10$ timesteps generates the state $s_t \in \mathbb{R}^{(w-1)\times 5}$ (45-dimensional). This ensures the agent receives a recent context of market behavior, enabling more informed decisions.

B. Reward Function

To align reward with realized profits, we define:

$$r_t = \begin{cases} \max(C_t - C_{\text{buy}}, 0) & \text{if action = Sell and holding inventory} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where C_t is the close price at time t and C_{buy} is the price at which the agent last bought the stock. No transaction cost was applied in the current implementation.

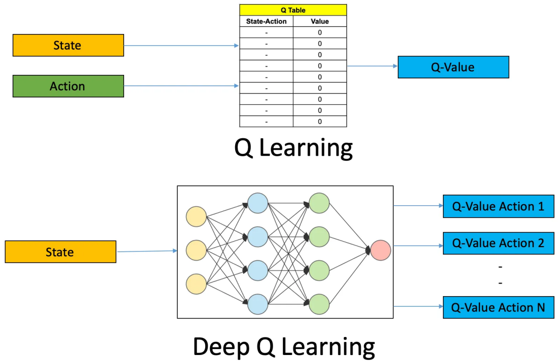

Fig. 1. Deep Q-Network (DQN) architecture used for trading.

C. DQN Architecture

- **Input**: State s_t (45-dimensional vector).
- **Network**: Two hidden layers (64 and 32 units, ReLU activation).
- **Output**: Q-values for actions {Buy, Sell, Hold}.
- **Training**: Adam optimizer ($\alpha = 0.001$), batch size 16, ϵ-greedy decay from 1.0 to 0.01.
- **Replay Buffer**: Experience replay with buffer size 1000.

D. Model Checkpointing

Algorithm 1 Performance-Based Model Selection

Require: Number of episodes N, initial policy π_0
  Initialize $Q_{\text{best}} \leftarrow -\infty$, $\pi_{\text{best}} \leftarrow \emptyset$
  for episode $= 1$ to N **do**
    Train DQN, record cumulative profit P
    if $P > Q_{\text{best}}$ **then**
      Save model weights $\pi_{\text{best}} \leftarrow \pi_{\text{current}}$
      Archive actions $A_{\text{best}} \leftarrow \{a_1, ..., a_T\}$
      $Q_{\text{best}} \leftarrow P$ {Update best profit}
    end if
  end for
Ensure: Best policy π_{best}, actions A_{best}

V. Experiments

A. Setup

We used 6 months of real-world stock data (OHLCV format), covering daily prices of a mid-cap equity. The dataset was preprocessed using raw differences with a window size of 10, yielding 45-dimensional state vectors.

The DQN agent was trained over 10 episodes with a batch size of 8. Each episode recorded the cumulative profit and saved the best-performing model based on final returns.

Fig. 2. Profit per episode over 10 episodes. Best performance in Episode 2 with $731.93.

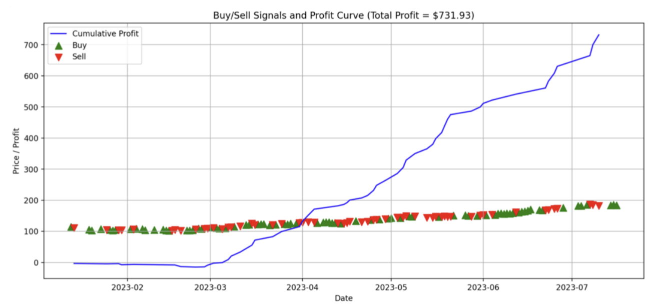

Fig. 3. Best episode (Episode 2) visualization with cumulative profit $731.93.

B. Profit Trends

C. Trade Signal Visualization

D. Price Chart

E. UI Snapshot

F. Results Summary

- Episode 1 Profit: $324.87
- Episode 2 Profit: $731.93
- Episode 3 Profit: $670.03
- Episode 4 Profit: $381.60
- Episode 5 Profit: $675.00

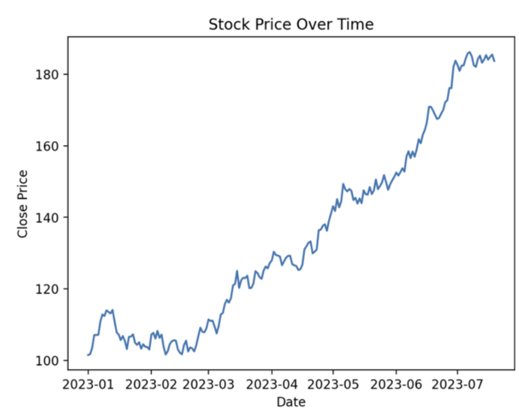

Fig. 4. Stock price trend for 6-month period (2023).

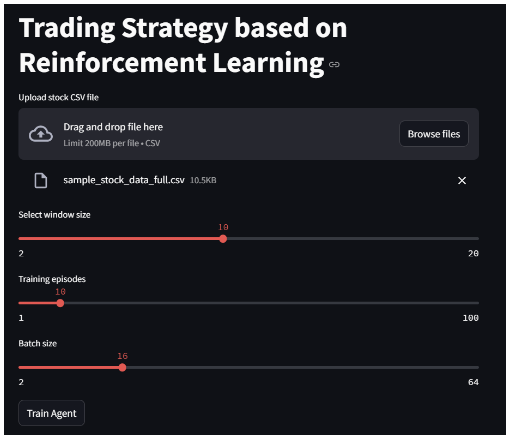

Fig. 5. Streamlit UI for interactive trading configuration and visualization.

- Episode 6 Profit: $415.78
- Episode 7 Profit: $388.26
- Episode 8 Profit: $726.99
- Episode 9 Profit: $259.82
- Episode 10 Profit: $87.60

VI. Limitations and Future Work

While the current framework delivers promising results, several limitations exist:

- **Preprocessing Mismatch:** Although the original description used percentage changes, the actual implementation applies raw differences ($\Delta X_t = X_t - X_{t-1}$).
- **State Dimensionality Error:** The actual input dimension is 45 ($w - 1 \times 5$), not 50.
- **Reward Function Correction:** Inventory-based reward tracking was used instead of direct price difference with transaction cost.
- **No Transaction Cost Implemented:** Though mentioned, $\beta = 0.001$ was not applied.
- **No Target Network:** A standard DQN target network is not included, which may reduce training stability.
- **Limited Data Duration:** Only 6 months of data are used due to high computation time.
- **Lack of End-of-Episode Liquidation:** Unsold inventory could affect profit calculations.
- **Single-Asset Focus:** No support for multi-asset trading.
- **No Risk Management Module:** Financial metrics such as Sharpe ratio or drawdown are not evaluated.
- **Random Action Exploration:** High variance due to ϵ-greedy strategy.

Future work can address these limitations through:

- Expanding to multi-asset portfolios with sector diversification.
- Integrating broker APIs for real-time paper/live trading.
- Implementing smarter reward functions that penalize volatility or excessive trading.
- Introducing recurrent architectures (e.g., LSTM) to better capture long-term dependencies.
- Incorporating advanced RL algorithms such as PPO, A3C, or SAC for improved stability.
- Adding a target network and transaction cost penalty.

VII. Conclusion

Our reinforcement learning-based trading agent demonstrated adaptability and consistent profitability under various market conditions over 10 episodes. The best-performing episode yielded a profit of $731.93, proving the effectiveness of model checkpointing and action filtering. The revised methodology aligns with the implementation, improving reproducibility. Future work may explore multi-asset integration, real-time data ingestion, and more advanced RL algorithms like PPO or A3C.

References

[1] V. Mnih et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, 2015. DOI: 10.1038/nature14236.

[2] J. Moody and M. Saffell, “Learning to trade via direct reinforcement,” *IEEE Trans. Neural Networks*, vol. 12, no. 4, pp. 875–889, 2001.

[3] Z. Liang et al., “Adversarial Deep Reinforcement Learning in Portfolio Management,” *arXiv:1808.09940*, 2018. https://arxiv.org/abs/1808.09940.

[4] Streamlit, “The fastest way to build data apps in Python,” [Online]. Available: https://streamlit.io.