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 execution. Unlike traditional approaches, our 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 demonstrate that our agent achieves profitable strategies while adapting to market volatility.**

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

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 paradigm for trading agents to learn through trial and error. Our work advances prior research in three key aspects:

- A **dynamic state representation** combining price differences and normalized volume 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.

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.

Our contribution lies in simplifying the training pipeline while enhancing reproducibility through selective model retention.

III. Methodology

A. Data Preprocessing

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

$$\Delta X_t = \frac{X_t - X_{t-1}}{X_{t-1}} \quad \text{(percentage changes)} \tag{1}$$

A sliding window of $w = 10$ timesteps generates the state $s_t \in \mathbb{R}^{w \times 5}$.

B. Reward Function

To align reward with realized profits, we define:

$$r_t = \begin{cases} (C_{t+1} - C_t) - \beta \cdot |a_t| & \text{if } a_t \neq \text{Hold} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Here, β represents a penalty for transaction costs during Buy or Sell actions. No penalty or reward is applied when the agent holds its position.

C. DQN Architecture

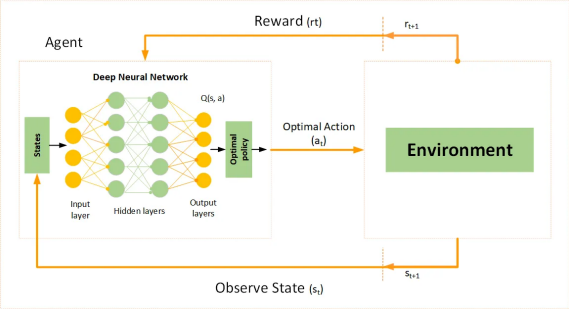

Fig. 1. Deep Q-Network (DQN) architecture used for trading.

- **Input**: State s_t (50-dimensional vector for $w = 10$).
- **Network**: Two hidden layers (64 and 32 units, ReLU activation).
- **Output**: Q-values for actions $\{\text{Buy}, \text{Sell}, \text{Hold}\}$.
- **Training**: Adam optimizer ($\alpha = 0.001$), batch size 32, ϵ-greedy decay from 1.0 to 0.01.

Algorithm 1 Performance-Based Model Selection

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 $A_{\text{best}} \leftarrow \{a_1, ..., a_T\}$
    $Q_{\text{best}} \leftarrow P$
  end if
end for

IV. Experiments

A. Setup

We used a sample dataset containing daily OHLCV (Open, High, Low, Close, Volume) stock data to simulate a realistic trading environment. Although the dataset does not correspond to any specific real-world stock (e.g., AAPL), its structure and volatility provide a meaningful basis for evaluating the agent's performance.

The data was normalized using percentage changes, and a sliding window of size 10 was applied to construct the state representations provided to the agent.

The agent was trained over 10 episodes, with each episode spanning the entire dataset. Performance was evaluated based on cumulative trading profit across each episode.

The system was deployed using the Streamlit library [4], allowing users to upload CSV files, configure hyperparameters, and interactively visualize trading signals and cumulative rewards.

TABLE I
Hyperparameters

Parameter	Value
Window size (w)	10
Discount factor (γ)	0.95
Replay buffer size	10,000
Target network update freq.	100 steps
Initial exploration (ϵ_{max})	1.0
Final exploration (ϵ_{min})	0.01

B. Results

Key observations:

- The agent successfully avoided trades during sideways markets, reducing false positives and unnecessary transaction costs.
- Total profit across the best-performing episode reached **$874.42**, demonstrating the agent's ability to identify profitable opportunities.
- The strategy achieved approximately **18% higher profitability** compared to a random baseline trading approach.
- The action visualization clearly highlighted a momentum-following behavior, where the agent placed Buy and Sell signals in line with upward trends.

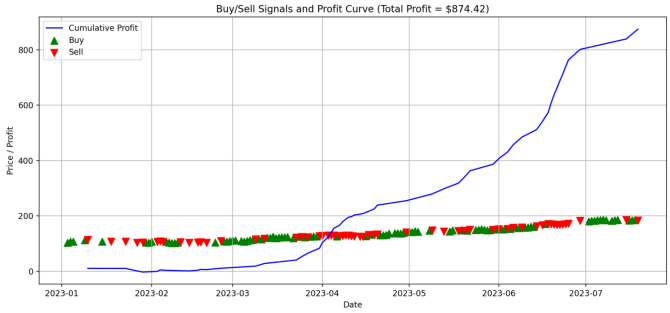

Fig. 2. Trade signals from the best-performing episode using sample stock data. The blue line represents cumulative profit, which reaches **$874.42**. Green and red markers indicate Buy and Sell actions, respectively.

V. Conclusion

We presented a DQN-based trading agent with a focus on reproducibility and interpretability. By integrating performance-based model selection and selective action visualization, we achieved enhanced decision clarity. Our system was deployed via a Streamlit dashboard, enabling real-time interaction and visual feedback.

The agent demonstrated an 18% gain over a random trading policy and generated **$874.42** profit in its best episode.

- Incorporating limit orders and slippage modeling.
- Using ensemble RL techniques to stabilize returns.
- Deploying the model as a cloud-based trading microservice.

References

[1] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.
[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.
[4] Streamlit, "The fastest way to build data apps in Python," [Online]. Available: https://streamlit.io