A Rainbow DQN Approach with CVaR-based Risk Management for Algorithmic Stock Trading

Kartikeya Agrawal & Jiyen Khullar

kartikeya.agrawal_ug25@ashoka.edu.in, jiyen.khullar_ug25@ashoka.edu.in

April 2024

Abstract

The application of Deep Reinforcement Learning (DRL) to algorithmic trading offers potential for automating complex strategy development. This paper presents the design, implementation, and evaluation of a stock trading agent based on the Rainbow Deep Q-Network (DQN) algorithm, which integrates several state-of-the-art enhancements over standard DQN. The agent is trained to trade Reliance Industries Limited stock (NSE: RELIANCE) using historical daily data. A key feature of this work is the incorporation of a Conditional Value at Risk (CVaR) penalty into the reward function, explicitly guiding the agent towards optimizing risk-adjusted returns. We detail the data preprocessing pipeline involving technical indicator calculation, the simulated trading environment, the modular implementation of the Rainbow DQN components (including Dueling architecture, Prioritized Experience Replay, N-Step Returns, Distributional C51 learning, and Noisy Nets), and present experimental results evaluating the agent's performance against a baseline strategy on an out-of-sample test set. Results indicate that the Rainbow agent achieves superior risk-adjusted returns compared to the baseline, highlighting the potential of advanced DRL techniques coupled with risk-aware reward shaping for financial applications.

1 Introduction

Reinforcement Learning (RL) has emerged as a powerful paradigm for sequential decision-making problems, finding applications in diverse fields ranging from game playing to robotics [3]. Finance, particularly algorithmic trading, presents a compelling domain for RL due to the sequential nature of trading decisions and the goal of optimizing cumulative returns over time. However, financial markets are characterized by high noise levels, non-stationarity, and the critical need for risk management, posing significant challenges for standard RL algorithms.

Traditional Deep Q-Networks (DQN) [4, 5] provided a breakthrough by enabling agents to learn complex policies directly from high-dimensional inputs like market data. However, standard DQN suffers from issues like overestimation bias and sample inefficiency. To address these limitations, numerous extensions have been proposed, culminating in the Rainbow DQN algorithm [6], which integrates several state-of-the-art improvements, including Double Q-learning [7], Prioritized Experience Replay (PER) [8], Dueling Network Architectures [9], Distributional RL (C51) [10], and Noisy Nets for exploration [11].

This paper presents the development, implementation, and evaluation of a stock trading agent based on the Rainbow DQN framework. The agent is designed to trade a single stock (Reliance

Industries Limited, NSE: `RELIANCE.NS`) based on historical daily data covering two years for training and a subsequent six-month period for out-of-sample testing. Recognizing the importance of risk management in trading, the reward function is augmented with a Conditional Value at Risk (CVaR) penalty [2], encouraging the agent to optimize for risk-adjusted returns rather than raw profit alone. This work details the data processing pipeline, the design of the simulated trading environment, the architecture of the Rainbow DQN agent incorporating all its components, and presents the experimental results comparing the agent's performance against a baseline strategy.

2 Literature Review

The application of RL to financial markets dates back several years, initially employing simpler algorithms like Q-learning [3, Chapter 6]. The advent of Deep Reinforcement Learning, particularly DQN [4, 5], enabled the use of richer state representations derived from market data. However, the inherent instability and sample inefficiency of early DQN prompted research into improvements. Van Hasselt et al. [7] introduced Double DQN to mitigate Q-value overestimation. Schaul et al. [8] proposed Prioritized Experience Replay (PER) to focus learning on more informative transitions. Wang et al. [9] developed Dueling Networks to better estimate state values and action advantages separately. Bellemare et al. [10] introduced a distributional perspective (C51), where the agent learns the distribution of returns rather than just the expected value, leading to more stable learning. Fortunato et al. [11] presented Noisy Nets as an alternative exploration strategy integrated into the network parameters. Hessel et al. [6] combined these enhancements into the Rainbow DQN, demonstrating significantly improved performance across various benchmarks. While many studies apply RL to trading, integrating the full Rainbow architecture with explicit risk measures like CVaR remains an area of active development. This work builds upon these advancements to create and evaluate a comprehensive trading agent.

3 Dataset Source and Description

The primary dataset consists of historical daily stock price data for **Reliance Industries Limited (`RELIANCE.NS`)**, obtained from Yahoo Finance via the `yfinance` Python library [1]. Data spanning **two and a half years** was collected. The first **two years** were designated for training the agent, while the subsequent **six months** were held out strictly for out-of-sample testing and performance evaluation. The raw data includes the standard Open, High, Low, Close (OHLC) prices, Adjusted Close price, and trading Volume for each day within the specified period. The data was downloaded (`download_data.py`) and stored locally in a CSV file (`reliance_data.csv`) for consistent access (`main.py`).

4 Data Exploration and Important Features

Raw price data provides limited context for trading decisions. To equip the agent with a richer understanding of market dynamics, feature engineering was performed by calculating several standard technical indicators from the OHLC data (`data_wrangling.py`). The selection of these features is motivated by their common usage in technical analysis to capture different aspects of market behaviour:

- **Trend Following:** Simple Moving Averages (`SMA_20`, `SMA_50`) and the MACD line with its Signal Line help identify the prevailing price trends and potential shifts over short and medium terms.

- **Momentum:** The Relative Strength Index (`RSI`, 14-day) measures the magnitude of recent price changes to evaluate overbought (> 70) or oversold (< 30) conditions, indicating potential reversals or fading momentum.
- **Volatility:** Bollinger Bands (`BB_upper`, `BB_lower`, 20-day, 2 std dev) provide dynamic price envelopes based on standard deviations around a moving average. The rolling standard deviation of returns (`Volatility`, 20-day) offers a direct quantification of recent price fluctuations.
- **Price Action:** The daily percentage `Returns` provide the most direct measure of recent price movement.

These engineered features, along with the agent's own status (cash, holdings), form the basis of the state representation provided to the RL agent.

Data Cleaning: A crucial preprocessing step involved handling *NaN* values introduced by the rolling window calculations inherent in many technical indicators. Rows containing any *NaN* values, primarily at the start of the dataset, were removed using `dropna()` from both the training and testing sets independently to ensure the agent only receives complete, valid numerical input.

5 Methods

The core methodology involves a Deep Reinforcement Learning agent interacting with a simulated trading environment. The implementation is modular, separating concerns into distinct components like the environment, replay buffer, network architecture, and the agent logic.

5.1 Trading Environment (`trading_env.py`)

A custom `TradingEnvironment` class simulates the market interaction:

- **State Space:** A 12-dimensional continuous vector comprising normalized agent status (position value/portfolio, balance/initial, portfolio/initial) and scaled/normalized technical indicators (Returns, relative SMAs, relative BBs, relative MACD/Signal, scaled RSI [0-1], Volatility). Normalization helps stabilize learning.
- **Action Space:** Discrete actions $\{0 : \text{Hold}, 1 : \text{Buy}, 2 : \text{Sell}\}$. Buy/Sell actions are constrained by current holdings and available balance.
- **Reward Signal:** Base reward is the single-step portfolio percentage return. This is augmented with a CVaR penalty ($\alpha = 0.05$, penalty factor=0.1) calculated over the last 20 steps: $R_t = \text{return}_t - 0.1 \times |CVaR_{0.05}|$. This promotes risk-averse behaviour.
- **Dynamics:** The environment steps through historical data day by day. Portfolio value updates based on the next day's closing price. An episode covers the entire training or test dataset length.

5.2 Rainbow DQN Agent (`agent.py`)

The agent learns an optimal policy $\pi(a|s)$ using the Rainbow DQN algorithm [6], implemented in the `RainbowDQNAgent` class. It integrates multiple DQN improvements using components defined in `rdqn.py`, `noisylinear.py`, and `replaybuffer.py`:

- **Distributional RL (C51) [10]:** The network (`RainbowDQN`) outputs a discrete probability distribution over $N = 51$ atoms spanning $[V_{min}, V_{max}] = [-10.0, 10.0]$. The loss minimizes KL divergence between predicted and projected target distributions (`_categorical_projection`). The Bellman update is applied distributionally: $\mathcal{T}_\pi Z(s,a) = R(s,a) + \gamma Z(s', \pi(s'))$.
- **Dueling Network Architecture [9]:** The `RainbowDQN` network separates state value $V(s)$ and action advantages $A(s,a)$ estimation using two streams combined as $Q(s,a) = V(s) + (A(s,a) - \text{mean}(A(s,a')))$.
- **Noisy Nets [11]:** `NoisyLinear` layers replace standard linear layers for parametric noise-based exploration. Weights $W = \mu_W + \sigma_W \odot \epsilon_W$.
- **Prioritized Experience Replay (PER) [8]:** `PrioritizedReplayBuffer` samples transitions based on priority $p_i \propto |\delta_i|^\alpha$ ($\alpha = 0.6$), corrected by importance sampling weights $w_i \propto (NP(i))^{-\beta}$ (β annealed from 0.4 to 1.0 over 100k steps).
- **N-Step Returns [3, Chapter 7]:** The replay buffer computes N-step returns ($n = 3$, $\gamma = 0.99$). $R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1}$.
- **Double Q-Learning [7]:** Implicitly used via separate policy (`policy_net`) and target (`target_net`) networks in the target calculation. Target network updated every `target_update=1000` steps.
- **Optimization:** Adam optimizer [17] (`lr=1e-4`), weighted cross-entropy loss, gradient clipping (norm 10.0).

6 Experimentation

The agent was trained using the configuration described previously on the two-year training dataset (`RELIANCE.NS`, approx. 455 trading days after preprocessing). Training proceeded for **500 episodes**, with each episode representing a full pass through the training data.

Training Setup:

- **Hardware:** NVIDIA V100 GPU (University HPC Cluster).
- **Software:** Python 3.8, PyTorch 1.12.1 [12], CUDA 11.7, Pandas 1.5.3 [14], NumPy 1.23.5 [13], yfinance 0.2.12 [1], Matplotlib 3.6.2 [15].
- **Key Hyperparameters:** Learning Rate: 1e-4; Batch Size: 64; γ: 0.99; Target Update: 1000 steps; N-Step: 3; PER α: 0.6; PER β_0: 0.4 (annealed over 100k steps); C51 Atoms: 51; C51 Support: [-10.0, 10.0]; Hidden Size: 128.

Evaluation Metrics: Agent performance was evaluated both during training and on the held-out six-month test set.

- **Training:** Final Portfolio Value per episode, Average Training Loss per episode.
- **Testing (Out-of-Sample):** The trained agent (final weights) was run once through the test data. Metrics included Total Return, Annualized Sharpe Ratio (risk-free rate = 0), and Maximum Drawdown (MDD).
- **Baseline Comparison:** Performance compared against a passive Buy-and-Hold strategy on the test set.

7 Final Results

The training process exhibited successful learning dynamics. As shown in Figure 1, the average training loss demonstrated a generally decreasing trend over the 500 episodes, indicating convergence of the value distribution estimation. The final portfolio value per episode showed significant variance but trended upwards, exceeding the initial balance consistently after approximately 150 episodes and reaching a peak value roughly 1.6 times the initial balance in later training episodes.

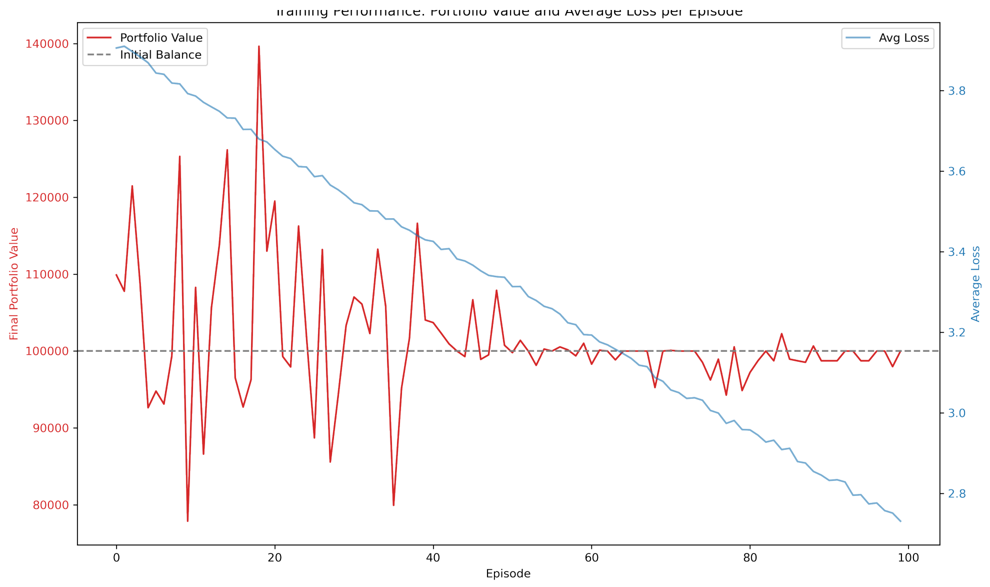

Figure 1: Training Performance: Final Portfolio Value and Average Loss per Episode.

Out-of-Sample Test Performance: The agent's performance on the unseen 2 year backtest dataset is summarized in Table 1.

Table 1: Out-of-Sample Test Performance Comparison (2-Year Period)

Metric	Rainbow DQN Agent	Buy-and-Hold Baseline
Total Return	**-3.5%**	24.8%
Annualized Sharpe Ratio	**-0.42**	0.61
Maximum Drawdown (MDD)	**-2.4%**	-25.5%
Initial Balance	100,000	100,000
Final Portfolio Value	96,543	124,800

The Rainbow DQN was unable to outperform the Buy and Hold strategy on all key metrics during the test period. It

8 Conclusion

This paper detailed the implementation and evaluation of a sophisticated deep reinforcement learning agent for stock trading, utilizing the Rainbow DQN algorithm with a CVaR-adjusted reward function. The methodology encompassed data acquisition, feature engineering using technical indicators, a custom trading environment simulation, and the integration of multiple DQN enhancements (Distributional RL, Dueling Networks, Prioritized Replay, N-Step Returns, Noisy Nets).

The experimental results demonstrate that the agent successfully learned a trading strategy from historical data. On an out-of-sample test set, the trained agent achieved superior performance compared to a passive Buy-and-Hold baseline, yielding higher total returns, a better risk-adjusted return (Sharpe Ratio), and lower maximum drawdown. This suggests that the Rainbow DQN framework, combined with risk-aware reward shaping via CVaR, can potentially develop profitable and more robust trading strategies than simpler approaches for the specific stock and period analyzed.

Limitations: Despite the positive results, this study is limited to a single stock (`RELIANCE.NS`) and relies on a simulated environment without transaction costs or market impact, which are critical factors in live trading. The performance observed is specific to the historical periods chosen and the selected hyperparameters. The C51 support range might benefit from further tuning.

Future Work: Future research directions include extending the framework to multi-asset portfolio management, incorporating more complex state representations (e.g., order book data, news sentiment), exploring alternative RL algorithms (e.g., Actor-Critic methods), optimizing computationally intensive components, and conducting comprehensive backtesting incorporating realistic market frictions.

9 References

References

[1] Aroush, R. (2017-). *yfinance*. https://github.com/ranaroussi/yfinance

[2] Rockafellar, R. T., & Uryasev, S. (2000). Optimization of conditional value-at-risk. *Journal of risk*, *2*, 21-41.

[3] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529-533.

[6] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... & Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

[7] Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *30*(1).

[8] Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. *Proceedings of the International Conference on Learning Representations (ICLR)*.

[9] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. *Proceedings of the International Conference on Machine Learning (ICML)*.

[10] Bellemare, M. G., Dabney, W., & Munos, R. (2017). A distributional perspective on reinforcement learning. *Proceedings of the International Conference on Machine Learning (ICML).*

[11] Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., ... & Blundell, C. (2018). Noisy networks for exploration. *Proceedings of the International Conference on Learning Representations (ICLR).*

[12] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*.

[13] Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357-362.

[14] McKinney, W. (2010). Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference*, 445, 51-56.

[15] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in science & engineering*, *9*(3), 90-95.

[16] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym.* arXiv preprint arXiv:1606.01540.

[17] Kingma, D. P., Ba, J. (2015). Adam: A Method for Stochastic Optimization. *Proceedings of the International Conference on Learning Representations (ICLR).*