Proceedings on Distributionally Robust Deep Q-Learning

Giulio Golinelli
golinelli.giulio13@gmail.com

September 2025

1 Improving the Code Base

The experiments in this work build on a fork of the original implementation. The modified version is available at `https://github.com/thisisnotgcsar/RDQN_proceeding`. For details on the improvements made, refer to Appendix A.

2 New gradient-based optimization

The current code base employs the Adam optimizer (Adaptive Moment Estimation) for ascending the lambda value in the convex dual formulation function during training. Adam is a widely used first-order gradient-based optimization algorithm that combines the advantages of both AdaGrad and RMSProp by computing adaptive learning rates for each parameter. It utilizes estimates of first and second moments of the gradients to provide efficient and robust convergence, especially in high-dimensional parameter spaces.

To potentially accelerate convergence and improve training efficiency, we replaced Adam with the Nadam optimizer (Nesterov-accelerated Adaptive Moment Estimation). Nadam extends Adam by incorporating Nesterov momentum, which evaluates the gradient at the projected position in parameter space, rather than the current position. This "look-ahead" mechanism can lead to faster convergence to the supremum of the objective function, particularly in convex optimization landscapes, by reducing oscillations and providing more stable updates.

For a fair comparison, all training runs were conducted on identical hardware and with consistent hyperparameters, including learning rate, batch size, and network architecture. The only variable modified was the choice of optimizer.

Table 1 summarizes our findings regarding training times and evaluation against empirical S&P 500 data for both Adam and Nadam optimizers. Note that, contrary to what was reported in the original paper, I experimentally found a Wealth value of 0.991 when running RDQN as-is using the attached notebook. This discrepancy is likely due to variability across different random seeds, even

though all trainings were performed using the same seed for consistency and time constraints.

The results show that Nadam achieved a far superior trading performance. This was most likely due to Nadam being able to find a better local supremum than the one found by Adam. A very slight reduction in training time is also observed. These characteristics enable a reduction in the number of epochs required to reach convergence to a solution of similar quality. This improvement is attributed to the enhanced momentum dynamics and more aggressive exploration of the parameter space provided by the Nesterov acceleration in Nadam.

Model	ε	δ	Training Time	Wealth	Max Draw.	Vol.	Sharpe	Down dev.	Sortino
S&P 500	-	-	-	9.524	-0.568	0.191	0.409	0.137	0.569
RDQN, Adam	3.0e-3	1.0e-4	3h 31m	0.991	-0.375	0.090	-0.003	0.067	-0.004
RDQN, Nadam	3.0e-3	1.0e-4	3h 28m	2.975	-0.285	0.091	0.414	0.065	0.579

Table 1: RDQN results for network optimized with Adam vs network optimized with Nadam. Results stems from evaluation over empirical S&P 500 data. No other variable has been changed across the two instances.

3 Bolstering RDQN Conservatism with Rough Evolutions

Recent empirical evidence demonstrates that the Robust Deep Q-Network (RDQN) algorithm consistently adopts a more conservative trading strategy compared to the standard Deep Q-Network (DQN). This conservatism is reflected in lower values of risk-related metrics, such as *volatility*, *Sharpe ratio*, *downside deviation*, and *Sortino ratio*, which are typically indicative of exposure to adversarial market conditions. Consequently, when the underlying time series exhibits increased roughness or adversarial dynamics, the RDQN agent tends to generate smoother and more stable performance trajectories.

The S&P 500 index, as a financial time series, is characterized by relatively low volatility and a persistent upward trend, underpinned by robust technological innovation and sustained human capital investment. In such environments, the optimal trading policy is often the straightforward buy-and-hold strategy, which capitalizes on the long-term appreciation of the index.

However, this raises the question of RDQN's efficacy in more challenging market regimes, where the underlying time series is subject to heightened volatility and dynamic fluctuations. In such scenarios, trading becomes substantially more complex, and algorithmic agents may outperform human traders by leveraging advanced diversification and risk management strategies.

To rigorously evaluate RDQN's robustness under increased market turbulence, we constructed a volatility-adjusted S&P 500 dataset. Volatility was artificially amplified by scaling the differences between consecutive prices and superimposing Gaussian noise, parameterized by a noise factor. This approach

simulates a market environment with pronounced distributional shift, thereby testing the agent's ability to generalize and maintain robust performance. Importantly, the neural network was trained using the simulated original S&P 500 as the reference distribution, and the evaluation on the volatility-adjusted data was conducted without re-training. Due to time constraints, re-training the neural network on the outputs of the simulator applied to the adjusted S&P 500 data has not been explored. Samples from the Student's t-distribution were used as the sampling distribution ν, ensuring that the evaluation reflects robustness to out-of-sample volatility and adversarial conditions. The code and scripts used to generate the volatility-adjusted dataset are available in the accompanying repository.

In Table 2, we present a comparative analysis of RDQN's performance on both the original and volatility-adjusted S&P 500 datasets. The results highlight RDQN's capacity to maintain stable and resilient trading behavior, even under more volatile market conditions.

Model	ε	δ	Training Time	Wealth	Vol.	Sharpe	Down dev.	Sortino
S&P 500	-	-	-	9.524	0.191	0.409	0.137	0.569
S&P 500, Vol. Adj.	-	-	-	9.520	0.197	0.397	0.141	0.554
RDQN, Nadam on S&P 500 Vol. adj.	3.0e-3	1.0e-4	3h 28m	1.813	0.094	0.218	0.068	0.302

Table 2: RDQN trading results against volatility-adjusted S&P 500 vs trading results against original S&P 500. No other variable has been changed across the two instances.

Exploring the effects of training the RDQN on the volatility-adjusted dataset, or on entirely new datasets that do not exhibit the same overall positive monotonicity as the S&P 500, represents a promising direction for future research. Such investigations could further elucidate the robustness and adaptability of the RDQN algorithm in diverse and potentially more adversarial market environments.

4 On the sampling distribution ν

The choice of the sampling distribution ν is pivotal in determining the robustness and generalization properties of the agent. This is because ν serves as a prior for the worst-case state-transition distribution, effectively guiding the neural network to align its behavior with ν under the worst adversarial scenarios.

Modeling financial time series with normal distributions often proves inadequate because, while the central limit theorem justifies normality under the assumption of many well-behaved, independent factors influencing asset prices, real-world financial markets are frequently disrupted by extreme events. Traumatic occurrences—such as oil shocks, major corporate bankruptcies, or sudden political upheavals—are not mathematically well-behaved and can lead to significant deviations from normality. Fat tails in market return distributions such as properly set t-students distributions have instead some behavioral origins

(investor excessive optimism or pessimism leading to large market moves) and are therefore studied in behavioral finance.

While the Student's t-distribution is commonly employed due to its heavy tails, which are hypothesized to capture classical financial return behaviors, alternative choices for ν may yield improved performance depending on the deployment context.

For instance, distributions with even heavier tails may more accurately reflect extreme market events and risk profiles observed in financial time series. Conversely, a uniform distribution may be preferable when the agent is intended to operate in environments characterized by high volatility and zero mean growth, such as those exhibiting white-noise dynamics. In such cases, the uniform distribution ensures equal representation of all possible outcomes, potentially enhancing the agent's adaptability to unpredictable market conditions.

Future work could systematically investigate the impact of different choices for ν on the agent's performance across a range of market regimes, with particular attention to environments exhibiting non-standard or adversarial statistical properties.

A Code Base Improvements

The experiments in this work build on a fork of the original implementation, which has been updated for greater reproducibility and stability. The modified version is available at `https://github.com/thisisnotgcsar/RDQN_proceeding`.

A `requirements.txt` file was added to document package dependencies, and the `.gitignore` file was extended with additional entries to improve repository management. A dedicated RDQN class was introduced, making use of the NAdam optimizer. A numerical issue in the logarithm computation was corrected, and a script was created for adjusting volatility in the input data. The main Jupyter notebook was also updated to incorporate these changes and provide a consistent entry point for running experiments.