Portfolio Optimization using Reinforcement Learning

Kshipra Jadav
M. Tech (ICT) Machine Learning
DA-IICT
Gandhinagar, India
202411010@daiict.ac.in

Shreshtha Modi
M. Tech (ICT) Machine Learning
DA-IICT
Gandhinagar, India
202411015@daiict.ac.in

***Abstract*—This project focuses on developing a Deep Reinforcement Learning (DRL) agent for portfolio optimization using financial time-series data. The objective was to design and train an agent capable of making optimal buy, sell, or hold decisions across multiple financial indices, maximizing portfolio returns over time.**

The agent utilized RNN architectures (LSTM and GRU models) to handle the sequential nature of financial data. A Deep Q-Network (DQN) was employed as the reinforcement learning algorithm, leveraging the temporal modeling capabilities of LSTMs and GRUs. Data from multiple NIFTY indices were preprocessed, normalized using `MinMaxScaler`, and fed into the models. The reward function was carefully crafted to reflect portfolio performance relative to initial value. Training results were analyzed using metrics such as rolling average returns and reward trends over episodes.

The LSTM complex model outperformed the GRU model, achieving higher mean rewards and faster convergence, suggesting superior ability to capture long-term dependencies in the data. The GRU model showed steady improvement but exhibited slightly more variability in performance.

The project demonstrated the potential of combining DRL with temporal models like LSTM/GRU for financial decision-making. While the framework achieved significant progress, future work could explore advanced RL algorithms (e.g., PPO, DDPG), include risk-adjusted rewards, and extend the framework to incorporate transaction costs for more realistic simulations. This approach lays the foundation for scalable and adaptive trading strategies.

I. Introduction

A portfolio refers to a collection of financial assets such as stocks, bonds, commodities, and other securities held by an investor. The primary objective of constructing a portfolio is to achieve specific financial goals, which typically include maximizing returns while minimizing risk. Portfolios can be tailored to meet various investment objectives, time horizons, and risk tolerances. They are often diversified to spread risk across different asset classes, sectors, and geographic regions, thereby reducing the overall risk associated with any single investment. Portfolio optimization is a quantitative approach aimed at constructing the most efficient portfolio given a set of constraints and objectives. The fundamental goal is to allocate assets in such a way that the portfolio's expected return is maximized for a given level of risk, or conversely, the risk is minimized for a given level of expected return. This process is grounded in modern portfolio theory (MPT), pioneered by Harry Markowitz in the 1950s.

Reinforcement Learning (RL) is a subfield of machine learning focused on training agents to make sequential decisions by interacting with an environment. The fundamental principle of RL is to learn optimal behaviors through trial and error, guided by a reward signal that evaluates the quality of the agent's actions.

Portfolio optimization using reinforcement learning (RL) represents a sophisticated and dynamic approach to investment management, leveraging advanced machine learning techniques to adapt to evolving market conditions. Unlike traditional mean-variance optimization, which relies on static historical data and assumptions about future returns, RL-based methods continuously learn from market interactions to optimize portfolio performance. In this framework, an RL agent interacts with the financial market environment, receiving rewards based on the performance of its portfolio decisions. The agent's objective is to maximize cumulative rewards, typically corresponding to risk-adjusted returns.

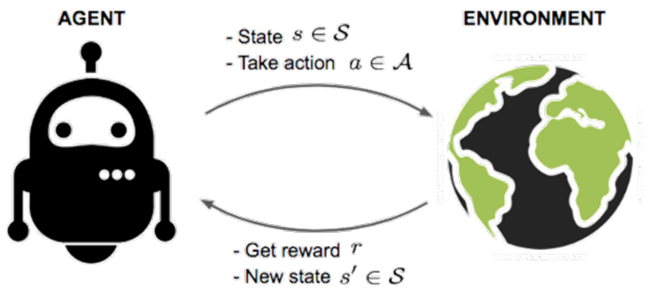
AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

Fig. 1. A Reinforcement Learning Setup

II. Methodology

The code for this project is divided into the following modules:

1) **utils.py**
2) **PortfolioEnvironment.py**
3) **DQNAgent.py**
4) **models.py**

5) **Portfolio Optimization with LSTM_complex.ipynb**
6) **Portfolio Optimization with GRU.ipynb**

A. *utils.py*

This file contains a single function `load_data()`, which uses the `pandas` library to load the dataset. The dataset comprises 5 CSV files sourced from the NSE website, containing yearly data for 5 actively traded ETFs (Exchange Traded Funds) on the National Stock Exchange. Each CSV file contains the following columns:

1) **Date:** The trading date for which the data is recorded.
2) **Series:** The type of security or instrument.
3) **OPEN:** The opening price of the security on the given trading day.
4) **HIGH:** The highest price at which the security traded during the day.
5) **LOW:** The lowest price at which the security traded during the day.
6) **PREV. CLOSE:** The closing price of the security from the previous trading day.
7) **LTP (Last Traded Price):** The most recent price at which the security was traded.
8) **CLOSE:** The closing price of the security on the given trading day.
9) **VWAP (Volume Weighted Average Price):** The average price of the security weighted by the volume of trades.
10) **52W H (52-Week High):** The highest price at which the security has traded in the past 52 weeks.
11) **52W L (52-Week Low):** The lowest price at which the security has traded in the past 52 weeks.
12) **VOLUME:** The total number of shares or units of the security that were traded during the day.
13) **VALUE:** The total monetary value of all trades executed for the security during the day.
14) **No of trades:** The total number of individual trades executed for the security during the day.

The 5 CSV files are loaded into separate DataFrames and combined into a single DataFrame containing only the **CLOSE** column.

B. *PortfolioEnvironment.py*

This module implements a custom class that simulates the trading environment for the reinforcement learning (RL) agent. The environment provides three actions for the agent: **Buy**, **Hold**, and **Sell**. Below is a breakdown of its implementation:

1) Class Constructor:

- **Purpose:** Initializes the environment with historical market data.
- **Parameters:**
  - `data`: A DataFrame containing historical price information for a security.
- **Attributes:**
  - `self.data`: Stores the historical market data.
  - `self.current_step`: Tracks the current time step in the data.
  - `self.max_steps`: Defines the total number of steps in the data.

2) Reset Method:

- **Purpose:** Resets the environment to its initial state for a new training episode.
- **Functionality:**
  1) Sets `self.current_step` to 0.
  2) Returns the initial observation (data for the first day).

3) Step Method:

- **Purpose:** Advances the environment by one time step based on the agent's action.
- **Actions:**
  - **Hold (0):** No action; reward is 0.
  - **Buy (1):** Buys the security; reward is 1% of the closing price.
  - **Sell (2):** Sells the security; reward is -1% of the closing price.
- **Returns:** Current observation, reward, and a boolean indicating if the episode has ended.

C. *DQNAgent.py*

This module implements a custom Deep Q-Learning (DQN) agent. The agent uses Q-values to make decisions in the trading environment. Below are the core methods:

1) Class Constructor:

- **Purpose:** Initializes the DQN agent.
- **Parameters:**
  - `input_shape`: Shape of the input state.
  - `num_actions`: Number of possible actions.
  - `build_model`: Function to build the neural network model.
- **Attributes:**
  1) `model`: Primary neural network for Q-value prediction.
  2) `target_model`: Secondary model for training stability.
  3) `memory`: Stores experiences for replay.
  4) `gamma`: Discount factor.
  5) `epsilon`: Exploration rate.

2) Act Method:

- **Purpose:** Determines the action based on current state.
- **Functionality:**
  - Explores randomly if `epsilon` is high.
  - Exploits the model's predictions otherwise.

3) Replay Method:

- **Purpose:** Trains the model using experience replay.
- **Functionality:**
  1) Samples a minibatch of experiences.
  2) Updates Q-values based on rewards and future predictions.
  3) Trains the model using the updated Q-values.

D. *models.py*

This module defines the neural network architectures used by the DQN agent. Two models are implemented:

- **LSTM Complex Model:** Uses three LSTM layers and dense layers to approximate Q-values.
- **GRU Model:** Similar to the LSTM model but uses GRU layers.

Both models are compiled with mean squared error loss and the Adam optimizer.

E. *Portfolio Optimization with LSTM_complex.ipynb and GRU.ipynb*

These Jupyter notebooks contain the driver code for running the project step-by-step. They execute the program in a sequential manner, making it easier to test and visualize results.

III. Results

In this project, we have trained the Deep Q Learning agent with two separate neural networks as it's backend. We aimed this project to be a head-to-head comparison between an LSTM model and an GRU model. We are well aware of the fact that we cannot quantify and generalize the comparison for all the use cases but we intended to make a comparison for the terms of portfolio optimization.

A. *LSTM Model*

It is evident from the graph that from the 1st episode itself, this model started giving positive rewards. That too, the positive rewards were almost consistent in nature till the last epoch. Althrough the dips that are seen, albeit sparsely, largeley depend on the particular ETF's price history and performance overtime as well. As in, if a ETF is on a downward trend then the bot is definitely going to get a downward trend in it's reward cycle as well. Also, it is worth to note that the 7 day rolling average of reward for this model is much more stable than the GRU model. This indicates that the model's learning stabilized overtime giving more accurate predictions.

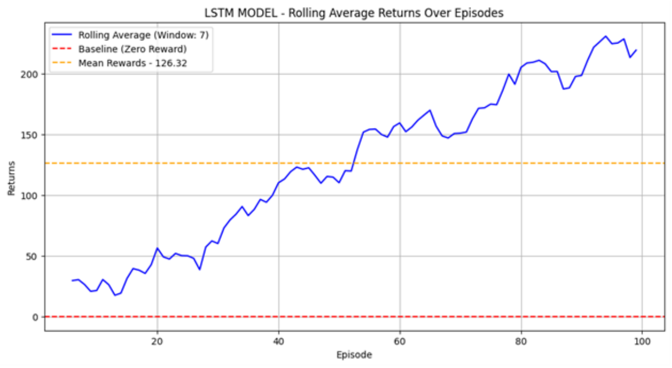

Fig. 2. 7 Day Rolling Average Returns of LSTM_complex Model

B. *GRU Model*

Immediately, we can notice here that the GRU model took a considerably longer time than the LSTM_complex model to generate positive return. We can see that during the first 10 episodes of training, the GRU model gives us negative rewards. This means that we were actually loosing money. This never happened with the LSTM_complex model. We can also see that the 7 day rolling average has a steeper slope than the 7 day rolling average of LSTM_complex model. This means that the model was not getting stabilised while learning and performing trades. It took some erratic decisions and hence, the reward curve got much steeper. Finally, we can clearly observe that the Mean Rewards for the LSTM_complex model are 126.32 whereas the Mean Rewards for the GRU model are 115.64. Hence, the LSTM_complex model has clearly outperformed the GRU model in terms of episodes to rewards ratio.

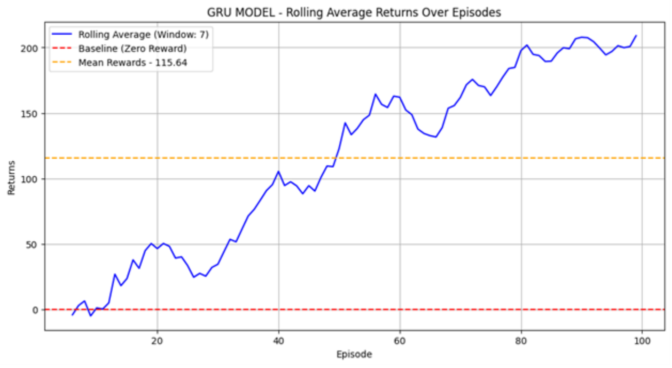

Fig. 3. 7 Day Rolling Average Returns of GRU Model

IV. Conclusion

This project was undertaken to explore and gain hands-on experience with **Reinforcement Learning (RL)** techniques, specifically **Deep Q-Learning (DQN)**, and to analyze the behavior and performance of an artificial trading agent on real-world financial data. By utilizing recurrent neural network architectures like **LSTMs** and **GRUs**, the project aimed to enable the RL agent to make informed and profitable decisions in the portfolio optimization context.

The results demonstrated steady improvement in agent performance over training episodes, as evidenced by the reward plots. Both GRU and LSTM-based models converged to higher rewards, indicating that the agent was successfully learning and adapting its policy. However, the **LSTM complex model** consistently outperformed the GRU model, achieving better consistency, higher peak rewards, and faster learning. This suggests that the LSTM model effectively captured the long-term dependencies and nuanced sequential patterns in the financial time-series data, making it a better choice for more complex datasets. Nevertheless, the GRU model, with its simpler architecture and lower computational overhead, remains a viable option for scenarios where computational efficiency is a higher priority.

The project also highlighted the importance of well-designed environments and reward functions in RL applications. By incorporating transaction costs and portfolio value changes into the reward mechanism, the agent was aligned with realistic trading objectives, making its behavior more practical and interpretable.

A. Future Scope

The project has laid a solid foundation for further exploration in reinforcement learning for portfolio optimization. Future work will focus on:

- **Incorporating Additional Features**: Extending the input features beyond closing prices to include *technical indicators* such as Moving Averages, RSI, and MACD. These enhancements aim to improve the agent's ability to respond to sudden market changes and make more robust decisions.
- **Real-World Deployment**: Testing the agent in live trading scenarios by connecting it to real-time stock market data. Deploying the agent in actual trading environments will allow us to evaluate its real-world performance and fine-tune its behavior.
- **Experimenting with Advanced RL Algorithms**: Exploring algorithms like *PPO (Proximal Policy Optimization)*, *DDPG (Deep Deterministic Policy Gradient)*, or *SAC (Soft Actor-Critic)* for potentially better performance in dynamic trading scenarios.
- **Improving Risk Management**: Integrating metrics such as portfolio volatility, Sharpe ratio, and maximum drawdown into the reward function to account for risk-adjusted returns.
- **Scalability**: Expanding the framework to handle multiple assets and dynamically allocate portfolio weights in a continuous action space.

All the code for this project and relevant files for this project including instructions on how to run this project on your local machine can be found at Portfolio Optimization Using RL.

References

[1] H. M. Markowitz, *Portfolio selection*, Yale University Press, 2017.

[2] H. Konno and H. Yamazaki, "Mean-absolute deviation portfolio optimization model and its applications to Tokyo stock market," *Management Science*, vol. 37, no. 5, pp. 519–531, May 1991.

[3] T.-J. Chang, N. Meade, J. E. Beasley, and Y. M. Sharaiha, "Heuristics for cardinality constrained portfolio optimisation," *Computers & Operations Research*, vol. 27, no. 13, pp. 1271–1302, Nov. 2000.

[4] A. Fernández and S. Gómez, "Portfolio selection using neural networks," *Computers & Operations Research*, vol. 34, no. 4, pp. 1177–1191, Apr. 2007.

[5] L. Yu, S. Wang, and K. K. Lai, "Neural network-based mean–variance–skewness model for portfolio selection," *Computers & Operations Research*, vol. 35, no. 1, pp. 34–46, Jan. 2008.

[6] T.-J. Chang, S.-C. Yang, and K.-J. Chang, "Portfolio optimization problems in different risk measures using genetic algorithm," *Expert Systems with Applications*, vol. 36, no. 7, pp. 10529–10537, Sep. 2009.

[7] G.-F. Deng, W.-T. Lin, and C.-C. Lo, "Markowitz-based portfolio selection with cardinality constraints using improved particle swarm optimization," *Expert Systems with Applications*, vol. 39, no. 4, pp. 4558–4566, Mar. 2012.

[8] H. R. Golmakani and M. Fazel, "Constrained portfolio selection using particle swarm optimization," *Expert Systems with Applications*, vol. 38, no. 7, pp. 8327–8335, Jul. 2011.

[9] K. Anagnostopoulos and G. Mamanis, "A portfolio optimization model with three objectives and discrete variables," *Computers & Operations Research*, vol. 37, no. 7, pp. 1285–1297, 2010.

[10] D. Bertsimas and D. Pachamanova, "Robust multiperiod portfolio management in the presence of transaction costs," *Computers & Operations Research*, vol. 35, no. 1, pp. 3–17, 2008.

[11] D. Li and W.-L. Ng, "Robust portfolio selection: A semi-parametric approach," *Review of Financial Studies*, vol. 13, no. 4, pp. 143–174, 2000.

[12] R. Mansini, W. Ogryczak, and M. G. Speranza, "Linear and mixed integer programming for portfolio optimization," *European Journal of Operational Research*, vol. 234, no. 2, pp. 536–545, Apr. 2014.

[13] J. Branke, S. S. Chaudhry, and M. Deb, "Multi-objective optimization using evolutionary algorithms for portfolio management," *ACM Transactions on Economics and Computation*, vol. 1, no. 1, pp. 24–43, Jun. 2009.

[14] Z. Jiang, D. Xu, and J. Yue, "A deep reinforcement learning framework for the financial portfolio management problem," *arXiv preprint arXiv:1706.10059*, 2017.

[15] J. Zhang, X. Li, and W. Dai, "Cost-sensitive portfolio optimization using deep reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 6, pp. 2344–2355, Jun. 2021.

[16] Y. Yang, T. Lai, and S. Wang, "Enhancing mean-variance portfolio optimization using LSTM-based stock price forecasting," *Expert Systems with Applications*, vol. 130, pp. 153–166, 2019.

[17] D. Maringer, *Portfolio Management with Heuristic Optimization*, Springer, 2003.

[18] F. J. Fabozzi, P. Natarajan, and F. J. Zaccaria, "Robust portfolio optimization," in *Handbook of Portfolio Construction*, Springer, 2007, pp. 595–617.

[19] W. Wu, S. Wang, and L. Tang, "Multi-objective evolutionary algorithms for portfolio optimization: A comprehensive review," *Expert Systems with Applications*, vol. 185, pp. 1–19, 2021.

[20] V. K. Chopra and W. T. Ziemba, "The effect of errors in means, variances, and covariances on optimal portfolio choice," *Journal of Portfolio Management*, vol. 19, no. 2, pp. 6–11, 1993.