Hierarchical Reinforced Trader

Gaurav Agarwal, Govind Singh, Vaisakh M. Menon

April 30, 2025

1 The Problem

1.1 Problem Statement

Financial markets are characterized by their complexity, volatility, and the influence of a wide range of interdependent factors, making the development of effective trading and portfolio management strategies a persistent challenge. Traditional quantitative models, such as Modern Portfolio Theory (MPT), have provided valuable frameworks for balancing risk and return. However, these models often assume static conditions and linear relationships, which limit their scalability and effectiveness in dynamically evolving market environments.

In contrast, Deep Reinforcement Learning (DRL) has gained attention for its ability to learn adaptive policies directly from market data, enabling automated decision-making in complex, high-dimensional spaces. Despite its promise, the application of DRL to financial trading is hindered by several key limitations. These include the difficulty of operating in high-dimensional action spaces, the presence of trading inertia—where agents fail to adapt efficiently to new information—and a lack of sufficient diversification, which can increase portfolio risk. Furthermore, issues such as data sparsity, noise, and non-stationarity exacerbate the challenges faced by DRL models.

Addressing these limitations is critical to advancing the application of DRL in real-world financial settings. There is a pressing need for enhanced algorithms that not only leverage the strengths of DRL but also incorporate mechanisms to promote stability, scalability, and robust portfolio diversification in dynamic market conditions. [3]

1.2 RL Problem Formulation

1.2.1 High Level Controller

- **States:**
  s = [fr,ss]
  fr represents **predicted forward returns** from historical price and volume data.
  ss represents **sentiment scores** from text data, such as news or tweets.

- **Actions:**
  a = [-1,0,1]
  -1, 0, 1 represents **selling, holding, and buying** respectively.

- **Rewards:**

$$R_h = \begin{cases} \text{sign}(a_i) \cdot \text{sgn}(\Delta P_i) & \text{if } a_i \neq 0 \\ 0 & \text{if } a_i = 0 \end{cases}$$

1.2.2 Low Level Controller

- **States:**
  s = [pt, ht, bt, ah]
  pt, ht, bt represents **stock prices, holdings and cash balance**. Augment **ah** with decisions from HLC.

- **Actions:**
  For one stock, 0,1...k where $k <= hmax$ (how much of the stock to be traded). Normalized to [-1,1] (continuous space)

- **Rewards:**
  Change in portfolio value from trades executed at **t to t+1**.

2 Methodology

2.1 Data Preprocessing

Data was collected from YahooFinance [1] API which includes Open, High, Low, Close, and Volume (OHLCV) data. The Volume Weighted Average Price (VWAP) and Forward Return is derived daily from OHLCV data.

- **Open**: Stock price at Session Open, used to compute forward returns for reward estimation.
- **Close**: Last traded price of the period, used for detecting trend and decision-making.
- **High/Low**: Highest and lowest price during the period, useful for risk assessment and volatility analysis.
- **Volume**: Number of shares traded, this indicates the market activity and liquidity.
- **VWAP (Volume Weighted Average Price)**: Reflects the true average price based on volume, used for optimal execution strategies.
- **Forward Return**: Future percentage change in the Opening price, serves as a reward signal for the RL agent.

We have also utilized the FinGPT Model [2] for Sentiment Analysis, which (explain how it chooses the sentiment, number of randim news articles and what is the score range.).

2.2 Implementation

The RL Model is a Hierarchical Based model involving a High-Level Controller (HLC), Low-Level Controller (LLC) & a Central Controller.

- **High-Level Controller (HLC):** Selects stocks to buy, sell, or hold based on market trends and sentiment analysis using PPO (Proximal Policy Optimization Algorithms)
- **Low-Level Controller (LLC):** Determines the optimal number of shares to trade using DDPG (Deep Deterministic Policy Gradient), optimizing execution efficiency.
- **Central Controller** Utilizes Phased Alternating Training, a joint training mechanism ensuring both controllers learn complementary strategies to maximize portfolio performance.

2.2.1

State Space: The state at each time step is represented as a vector combining normalized stock prices, normalized holdings, normalized cash, and the HLC (Hold, Long, Close) decisions. Specifically:

Normalized prices: Derived from historical stock data. Normalized holdings: Represented as the fraction of maximum shares held. Normalized cash: Scaled to the range [-1, 1]. HLC decisions: Encoded as [-1, 0, 1] for sell, hold, and buy, respectively. The state is constructed using the $_{g}et_{s}state()methodintheStockExecutionEnvclass, ensuringallco$

Action Space: The action space is continuous, representing the proportion of the portfolio to allocate to each stock. Actions are clipped to the range [-1, 1] to ensure valid allocations.

Reward Function: The reward is designed to reflect the change in portfolio value after executing an action. It incorporates:

The net profit or loss from stock transactions. Penalties for excessive risk or transaction costs. Transition Dynamics: The environment transitions to the next state based on the executed action, updating cash, holdings, and the current step. The StockExecutionEnv class handles these transitions.

Objective: The RL agent's goal is to maximize the cumulative reward over the trading horizon, which corresponds to maximizing portfolio value while adhering to constraints.

The RL formulation leverages the StockExecutionEnv environment, which simulates the stock market dynamics and provides a realistic setting for training the agent.

2.3 Methodology Used

This project employs the Deep Deterministic Policy Gradient (DDPG) algorithm, a model-free RL method suitable for continuous action spaces. Below, we detail the methodology used:

2.3.1 1. Environment Design

The environment is implemented in the StockExecutionEnv class, which extends gym.Env. Key features include:

State Normalization: Prices, holdings, and cash are normalized to ensure numerical stability. HLC Decisions: The environment allows setting predefined HLC decisions for each episode, enabling the agent to adapt its strategy accordingly. Reward Calculation: Rewards are computed based on portfolio value changes, incentivizing profitable actions. The environment is initialized with historical stock price data and parameters such as initial cash and maximum shares.

2.3.2 Agent Architecture

The agent is implemented using the DDPG algorithm, which consists of two neural networks:

Actor Network: The actor network maps states to actions. It has three fully connected layers with ReLU activations, followed by a tanh activation to constrain actions to the range [-1, 1]. The output is scaled by the maximum allowable action.

Critic Network: The critic network evaluates the Q-value of state-action pairs. It concatenates the state and action inputs and processes them through three fully connected layers.

Both networks are optimized using Adam optimizers with learning rates of 1e-4 (actor) and 1e-3 (critic).

2.3.3 Training Process

The training process is implemented in the train$_{l}lc()functionandinvolvesthefollowingsteps:$

2.3.4 Data Preparation:

Historical stock price data is loaded and preprocessed to create the price DataFrame.

Environment Initialization: The StockExecutionEnv is instantiated with the price data, initial cash, and maximum shares.

Agent Initialization: The DDPG agent is initialized with the state and action dimensions derived from the environment.

Training Loop: For each episode:

The environment is reset, and random HLC decisions are generated. The agent interacts with the environment by selecting actions using its policy (select$_{a}ction()method).Transitions(st$
$batchessampledfromthereplaybuffer.ModelSaving: Aftertraining, theactorandcriticnetworksaresavedfor$

2.3.5 DDPG Algorithm Details

The DDPG algorithm combines policy gradients and Q-learning. Key components include:

Replay Buffer: A deque stores transitions for experience replay, breaking temporal correlations in training data.

Target Networks: Target networks for the actor and critic stabilize training by providing slowly updated targets.

Loss Functions:

Critic Loss: Mean squared error between predicted Q-values and target Q-values. Actor Loss: Negative mean Q-value of the actor's actions. Optimization: Gradients are computed and applied to update the actor and critic networks.

2.3.6 Evaluation

The agent's performance is evaluated based on:

Cumulative reward over episodes. Final portfolio value compared to the initial cash. Progress is logged during training, and the trained models are saved for deployment.

This methodology ensures a robust and scalable approach to solving the stock execution problem using RL. The combination of a well-designed environment, a powerful RL algorithm, and careful training ensures the agent learns an effective trading strategy.

3 Github Link

`https://github.com/AlphaHawk91/reinforcement_learning_hrt`

References

[1] Yahoo finance. `https://finance.yahoo.com/`. Accessed: 2025-04-30.

[2] Boyu Zhang, Hongyang Yang, and Xiao-Yang Liu. Instruct-fingpt: Financial sentiment analysis by instruction tuning of general-purpose large language models. *arXiv preprint arXiv:2306.12659*, 2023.

[3] Z. Zhao and R. E. Welsch. Hierarchical reinforced trader (hrt): A bi-level approach for optimizing stock selection and execution. *arXiv preprint arXiv:2410.14927*, 2024.