Project Proposal

Deep Reinforcement Learning for Portfolio Optimisation

IEDA4000F - Deep Learning for Decision Analytics

Submitted by:

CHONG, Tin Tak (20920359)

Department of Industrial Engineering and Decision Analytics

The Hong Kong University of Science and Technology (HKUST)

Supervisor:

MEI, Sanyou

Submission Date: November 13, 2025

1 Background and Motivation

Portfolio optimisation lies at the heart of quantitative finance, with the objective of balancing risk and return under market uncertainty. Traditional approaches such as Markowitz's mean–variance optimisation or Black–Litterman models rely on assumptions of stationarity and Gaussian returns, which are often violated in real markets. Moreover, these static methods are not adaptive to rapidly changing market conditions.

Recent advances in **Deep Reinforcement Learning (DRL)** have enabled the development of adaptive decision-making systems that can learn optimal trading or allocation policies directly from data. DRL agents can continuously adjust portfolio weights in response to new market information, learning to maximise long-term risk-adjusted returns.

This project aims to design and implement a DRL-based portfolio optimisation system that dynamically allocates capital among multiple assets (e.g., equities, ETFs, or cryptocurrencies). The project seeks to compare the DRL strategy against traditional benchmarks and investigate its robustness under different market regimes.

2 Objectives

The main objectives are:

1) Formulate portfolio optimisation as a reinforcement learning problem, defining appropriate state, action, and reward structures.

2) Implement and train deep reinforcement learning agents (e.g., DQN, PPO, or DDPG) for dynamic asset allocation.

3) Evaluate model performance using financial metrics such as annualised return, Sharpe ratio, maximum drawdown, and volatility.

4) Compare the DRL strategy against benchmark portfolios (equal-weighted, mean–variance, and risk parity).

5) Analyse the agent's behaviour under different market conditions and transaction cost settings.

3 Methodology

3.1 Model Design

- **Algorithms:** Start with DQN for discrete actions, and extend to PPO or DDPG for continuous allocation.
- **Network Architecture:** 2–3 hidden layers (ReLU activation), outputting continuous weights for assets.

3.2 Mathematical Formulation

We give a formal MDP description and the principal equations used in the implementation.

Notation Let N be the number of assets and $t = 0, \ldots, T$ index trading steps (days). Denote the vector of asset prices by $\mathbf{p}_t \in R^N$ and the vector of simple returns by

$$\mathbf{r}_t = \frac{\mathbf{p}_t - \mathbf{p}_{t-1}}{\mathbf{p}_{t-1}}.$$

Portfolio weights (fraction of wealth allocated to each asset) at time t are denoted $\mathbf{w}_t \in R^N$.

State The state s_t contains market observations and optionally the agent's previous allocation:

$$s_t = \left[\, \mathbf{p}_{t-K:t},\ \mathbf{x}_t,\ \mathbf{w}_{t-1} \,\right],$$

where $\mathbf{p}_{t-K:t}$ denotes the window of past prices (or returns) of length K, and $\mathbf{x}_t$ denotes engineered features (technical indicators, realised volatility, momentum, macro signals, etc.). Example feature definitions:

$$\begin{aligned} \text{SMA}_t^{(L)} &= \frac{1}{L}\sum_{i=0}^{L-1} p_{t-i}, \\ \text{EMA}_t^{(\alpha)} &= \alpha p_t + (1-\alpha)\,\text{EMA}_{t-1}, \\ \text{Momentum}_t^{(L)} &= \frac{p_t - p_{t-L}}{p_{t-L}}. \end{aligned}$$

Action The action a_t is the portfolio allocation for the next period. We parameterise the policy to output either:

- continuous weights $\mathbf{w}_t$ with the budget constraint enforced by normalization:

$$\mathbf{w}_t = \frac{\exp(\mathbf{z}_t)}{\mathbf{1}^\top \exp(\mathbf{z}_t)} \quad \text{(softmax transform)},$$

  where $\mathbf{z}_t \in R^N$ are the network outputs; or

- unconstrained raw allocations $\tilde{\mathbf{w}}_t$ followed by projection onto feasible set $\mathcal{W} = \{\mathbf{w} : \mathbf{1}^\top \mathbf{w} = 1,\ w_i \geq 0\}$ using

$$\mathbf{w}_t = \text{Proj}_{\mathcal{W}}(\tilde{\mathbf{w}}_t).$$

Portfolio return and value update Given previous weights $\mathbf{w}_{t-1}$ and returns $\mathbf{r}_t$, the portfolio simple return (without costs) is:

$$R_t^{\text{gross}} = \mathbf{w}_{t-1}^\top \mathbf{r}_t.$$

If V_{t-1} is portfolio value at $t-1$, then (ignoring costs) $V_t = V_{t-1}(1 + R_t^{\text{gross}})$.

Transaction costs and turnover Define turnover between periods:

$$\text{Turn}_t = \sum_{i=1}^{N} \left| w_{t,i} - w_{t-1,i} \right|.$$

A simple linear transaction cost model subtracts

$$\text{cost}_t = c \cdot \text{Turn}_t,$$

from returns, where $c > 0$ is a per-unit transaction cost parameter (round-trip cost). Then the net portfolio return is

$$R_t = R_t^{\text{gross}} - \text{cost}_t.$$

Value update with costs:

$$V_t = V_{t-1}\,(1 + R_t).$$

Reward design Several reward choices are common. Example instantaneous reward (risk-penalised return):

$$r_t = R_t - \lambda\,\hat{\sigma}_t^2,$$

where $\hat{\sigma}_t^2$ is an estimate of recent portfolio return variance (e.g. sample variance over last m days) and $\lambda \geq 0$ controls risk aversion. Alternatively, one may use a Sharpe-like reward:

$$r_t = \frac{R_t - r_f}{\hat{\sigma}_t},$$

or a utility-based reward (CRRA/mean–variance):

$$r_t = R_t - \tfrac{1}{2}\,\alpha R_t^2.$$

MDP objective The agent seeks to maximise expected discounted cumulative reward:

$$J(\pi_\theta) = E_{\pi_\theta}\Big[\sum_{t=0}^{T} \gamma^t r_t\Big],$$

where $\pi_\theta(a_t \,|\, s_t)$ is the policy with parameters θ and $\gamma \in (0, 1]$ is the discount factor.

Policy-gradient (actor-critic) formulation The policy gradient theorem gives:

$$\nabla_\theta J(\pi_\theta) = E_{\pi_\theta}\big[\nabla_\theta \log \pi_\theta(a_t \,|\, s_t)\, A^\pi(s_t, a_t)\big],$$

where the advantage $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$.

Proximal Policy Optimisation (PPO) objective Using importance sampling ratio $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, the clipped surrogate loss is:

$$\mathcal{L}^{\text{CLIP}}(\theta) = E\Big[\min\big(r_t(\theta)A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\big)\Big],$$

optionally with value and entropy terms:

$$\mathcal{L}(\theta) = -\mathcal{L}^{\text{CLIP}}(\theta) + c_1\mathcal{L}^{\text{VF}}(\theta) - c_2 E\big[\mathcal{H}(\pi_\theta(\cdot|s_t))\big].$$

Deterministic Policy Gradient (DDPG) For deterministic policy $\mu_\theta(s)$, the deterministic policy gradient is:

$$\nabla_\theta J \approx E_{s\sim\mathcal{D}}\Big[\nabla_\theta \mu_\theta(s)\, \nabla_a Q_\phi(s, a)\big|_{a=\mu_\theta(s)}\Big],$$

where Q_ϕ is the critic network and $\mathcal{D}$ is a replay buffer. Critic updates minimise:

$$\mathcal{L}(\phi) = E_{(s,a,r,s')\sim\mathcal{D}}\Big[\big(Q_\phi(s, a) - y\big)^2\Big], \quad y = r + \gamma Q_{\phi'}(s', \mu_{\theta'}(s')),$$

with slowly-updated target networks ϕ', θ'.

Constraints Budget constraint:

$$\mathbf{1}^\top \mathbf{w}_t = 1.$$

Long-only constraint if required:

$$w_{t,i} \geq 0, \quad \forall i.$$

Leverage or position bounds can be enforced as:

$$\|\mathbf{w}_t\|_1 \leq L, \quad -b_i \leq w_{t,i} \leq u_i.$$

Regularisation and penalties To discourage excessive trading and overfitting, add regularisation terms to the reward or to the objective:

$$r_t^{\text{reg}} = r_t - \eta \sum_i (w_{t,i} - w_{t-1,i})^2,$$

or include L2 weight decay on network parameters in optimisation.

Evaluation metrics (used for comparison) Annualised return:

$$\text{AR} = \Big(\prod_{t=1}^{T} (1 + R_t) \Big)^{\frac{252}{T}} - 1.$$

Annualised volatility:

$$\sigma_{\text{ann}} = \text{std}(R_t)\sqrt{252}.$$

Sharpe ratio (annualised):

$$\text{Sharpe} = \frac{\text{AR} - r_f}{\sigma_{\text{ann}}}.$$

Maximum drawdown, and turnover:

$$\text{Turnover} = \frac{1}{T} \sum_{t=1}^{T} \text{Turn}_t.$$

Practical implementation notes

- To ensure numerical stability and feasibility, normalise inputs (returns, indicators) using rolling z-score scaling.
- Use softmax output for strictly long-only allocations, and a tanh+normalisation pipeline for allowing short positions with bounded leverage.
- Include transaction cost c and slippage in the environment to get realistic learned policies.

3.3 Training and Evaluation

- **Environment:** Custom trading simulator using the OpenAI Gym interface.
- **Training:** Agent interacts with historical data to learn optimal policies via trial and error.
- **Evaluation Metrics:** Annualised return, Sharpe ratio, maximum drawdown, and turnover.

3.4 Benchmarks

- Equal-weight portfolio
- Mean–variance optimisation
- Momentum-based heuristic portfolio

3.5 Tools and Libraries

Python (Pandas, NumPy, Matplotlib), TensorFlow/PyTorch for DRL implementation, `yfinance` or `ccxt` for data acquisition, and `stable-baselines3` for reinforcement learning.

4 Data Description

- **Source:** Yahoo Finance API
- **Assets:** 5–10 diversified assets (e.g., AAPL, NVDA, TSLA, SPY, GLD, BTC-USD)
- **Period:** 2015–2024 (daily or weekly frequency)
- **Preprocessing:** Log returns, normalisation, feature scaling

5 Expected Results

The DRL-based optimiser is expected to learn adaptive allocation patterns that outperform static or traditional benchmarks in terms of risk-adjusted return. The agent should exhibit dynamic reallocation during volatile or trending markets, demonstrating regime awareness. Empirical evaluation will focus on performance improvements such as a higher Sharpe ratio and a lower drawdown.

6 Potential Challenges and Mitigation

Challenge	Mitigation Strategy
Overfitting to training data	Use out-of-sample testing and cross-validation
High model complexity	Begin with simpler models (DQN) before PPO/DDPG
Transaction cost sensitivity	Include cost penalty in the reward function
Non-stationary data	Apply rolling retraining and normalisation

7 Expected Deliverables

- Technical report detailing methodology and experimental results
- Python implementation, including training and backtesting scripts
- Presentation slides or poster summarising results
- Optional interactive notebook for model demonstration

8 References

- Moody, J.E., Wu, L., Liao, Y., & Saffell, M. (1998). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting, 17*, 441-470.

- Jiang, Z., Xu, J., & Liang, J. (2017). *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem*. arXiv:1706.10059.

- Yang S. (2023). *Deep Reinforcement Learning in Portfolio Management*. arXiv:1808.09940.

- Buehler, H., Gonon, L., Teichmann, J., & Wood, B. (2019). Deep Hedging. *Quantitative Finance, 19*(8), 1271–1291.