Algorithmic Trading using Reinforcement Learning

Course: CS3103
Group ID: 25SM03

Team Members

Sakshi Saxena (2301CS45)
Ishika Gupta (2301AI10)
Manshi Prajapati (2302CS08)
Shiwani Shrivastava (2301CS67)

1. Introduction

In dynamic financial markets, traditional rule-based models struggle with volatility driven by unpredictable factors. This project develops a Reinforcement Learning (RL)-based trading agent using Proximal Policy Optimization (PPO) to learn adaptive buy, sell, or hold strategies, maximizing rewards while managing risk. We also implement Deep Q-Network (DQN) for comparison, highlighting PPO's advantages in stability and efficiency. The objectives are: 1) Implement PPO and DQN agents for multi-stock trading and 2) Evaluate performance via profitability, Sharpe ratio.

2. Brief Related Work

RL has transformed trading by handling sequential decisions. Early Q-learning faced instability; Deep Q-Networks (DQN) (Mnih et al., 2015) integrated deep networks for high-dimensional states but limited continuous actions. Proximal Policy Optimization (PPO) (Schulman et al., 2017) improves policy gradients with clipped objectives for stable, sample-efficient learning, excelling in trading by balancing exploration and exploitation. Our system extends this with multi-stock PPO/DQN, transaction costs, and technical indicators for robust evaluation.

3. Methodology

3.1 Dataset

Aspect	Details
Source	Yahoo Finance Dataset (yfinance)
Stocks	MSFT, GOOGL, TSLA, META, NFLX
Period	2015–2024
Fields	Open, High, Close, Low, Volume, Adjusted Close

3.2 Data Preprocessing

The data was cleaned for gaps using forward/backward fill and normalized via StandardScaler. Technical indicators were added including returns, moving averages (5/10 day), EMA10, volatility measures (standard deviation, Bollinger bands), momentum indicators (RSI, MACD), and volume metrics.

3.3 Trading Environment

Component	Specifications
State Space	30-day window of 17 indicators + shares held + cash held
Action Space	0: Hold, 1: Buy 1 share, 2: Sell 1 share
Reward Function	log(portfolio_after / portfolio_before)
Transaction Cost	0.1% per trade
Initial Capital	$10,000 cash

3.4 Algorithms

Algorithm: Deep Q- Network

1. Load tickers, download data, compute features, and fit a StandardScaler for each ticker.
2. Split each ticker's data into train, validation, and test sets.
3. Initialize replay buffer (D), policy network (Q), and target network (Q_target).
4. Prefill replay buffer using random actions.
5. For each training step:
   a. Sample a ticker and reset the TradingEnv.
   b. Select action using epsilon-greedy:
      epsilon = eps_end + (eps_start − eps_end) * exp(− steps / tau)
   c. Execute action and compute reward:
      reward = (new_value − old_value) / (old_value + 1e-6)
   d. Store (state, action, reward, next_state, done) in D.
   e. Sample a minibatch and compute target:
      target = reward + gamma * max(Q_target(next_state))
   f. Update Q by minimizing:
      loss = mean((Q(state, action) − target)^2)
   g. Periodically copy weights:
      Q_target = Q

6. Save the trained model, scalers, and validation history.
7. Evaluate on test data and compute metrics:
   - Total Return = final_value / initial_cash − 1
   - Sharpe Ratio = (mean_daily_return / std_daily_return) * sqrt(252)

Algorithm: Proximal Policy Optimisation

1. Load tickers, download price data, compute features, and fit a StandardScaler.
2. Split each ticker's data into train, validation, and test sets.
3. Initialize PPO components:
   - Actor (policy) network
   - Critic (value) network
   - Adam optimizer
   - Rollout buffer
4. For each training iteration:
   a. Pick a ticker and reset the trading environment.
   b. For each step in the rollout:
   - Choose an action from the policy
   - Run the action and compute reward: (new_value − old_value) / (old_value + 1e-6)
   - Store state, action, reward, value, log-prob, and done in the buffer
5. Compute advantages using GAE and compute returns.
6. PPO update:
   a. Compute probability ratio between new and old policy
   b. Clip the ratio to control large policy updates
   c. Compute policy loss, value loss, and entropy bonus
   d. Update the actor and critic networks
7. Repeat rollouts + PPO updates until training finishes.
8. Save trained policy, value network, scalers, and logs.
9. Evaluate on test data:
   - Total Return = final_value / initial_cash − 1
   - Sharpe Ratio = (mean_daily_return / std_daily_return) × sqrt(252)

4. Key Results

Metric	PPO(Avg)	DQN(Avg)
Total Return	34.84	18.14
Sharpe Ratio	29.53	25.96

PPO achieved 47.9% higher returns than DQN, attributed to stable policy updates.

5. Conclusion

This project successfully implemented PPO and DQN algorithms for adaptive algorithmic trading, with PPO demonstrating superior performance (34.84% returns, Sharpe ratio 29.53) compared to DQN. The study highlights that PPO's clipping mechanism ensures stability in volatile market conditions, technical indicators significantly improve pattern learning and decision-making, and transaction costs make trade minimization crucial for maximizing profits.

References

1) Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
   Link: https://arxiv.org/abs/1707.06347
2) Jiang, Y., Huang, J., Yuan, Y., & Zhang, X. (2024). Risk-sensitive reinforcement learning for portfolio management. Proceedings of the AAAI Conference on Artificial Intelligence, 38(7), 7890-7898.
   Link: https://ojs.aaai.org/index.php/AAAI/article/view/28567
3) *Anonymous.* (2025). *A deep reinforcement learning framework for strategic Indian NIFTY 50 index trading*. **AI**, 6(8), Article 183.
   https://www.mdpi.com/2673-2688/6/8/183
4) Sarkar, S. (2025). *Quantitative trading using deep Q-learning* (Version 2). arXiv:2304.06037.
   https://arxiv.org/abs/2304.06037