A Reinforcement Learning Approach to Automated Trading: Buys, Sells, Hold, and Deep Learning Insights

Iqza Ardiansyah[1] and Kevin Ignatius Wijaya[2]

[1]Student ID: 2206810042, Faculty of Computer Science, Universitas Indonesia
[2]Student ID: 2206083470, Faculty of Computer Science, Universitas Indonesia

May 23, 2025

Abstract

This paper presents a Reinforcement Learning (RL) approach for automated trading on real S&P 500 historical data. We incorporate buy/sell actions (10% buys, 1% sells) to achieve a more granular control of portfolio allocation, while employing Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO) for policy learning. Our comparative analysis reveals significant performance differences between these algorithms in trading environments. We evaluate our approach with fixed seeds for reproducibility, yielding detailed insights on reward trajectories, learning stability, and financial performance metrics including portfolio value, ROI, Sharpe ratio, and maximum drawdown. Our experiments demonstrate that the DQN algorithm's value-based approach consistently outperforms PPO's policy gradient method in this particular trading environment with discrete action spaces and delayed rewards.

1 Introduction

Algorithmic trading has rapidly evolved, driven by both improved computational power and advances in data-driven methods. Traditional strategies (e.g., mean reversion, momentum trading) rely on fixed heuristics, often lacking adaptability to changing market conditions. In contrast, Reinforcement Learning (RL) optimizes trading decisions by directly interacting with an environment and receiving feedback in the form of rewards [Sutton and Barto, 2018].

While many works focus on single-asset trading, recent literature in pairs trading has shown that decomposing the trading problem into multiple subtasks can be highly beneficial. Although our study focuses on single-asset trading with partial buy/sell actions, the insights from hybrid models offer promising avenues for future research and potential integration.

In this study, we extend previous work by conducting a comprehensive analysis of two popular reinforcement learning algorithms: Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO). We investigate their comparative performance in a trading environment with partial buy/sell capabilities, analyzing their learning dynamics, stability characteristics, and financial outcomes.

2 Related Work

2.1 Reinforcement Learning for Trading

RL directly learns a policy that maximizes cumulative rewards, such as changes in portfolio value after accounting for transaction costs. Foundational work by Sutton and Barto [2018] laid the groundwork for modern RL approaches, while recent studies have applied various RL algorithms (e.g., DQN, PPO, A2C) to financial markets. However, standard RL approaches often rely on simplistic buy/sell/hold actions, leading to abrupt position changes.

Value-based methods like DQN [Mnih et al., 2015] learn the expected return of each action in a given state, selecting the action with the highest expected value. Policy gradient methods like PPO [Schulman et al., 2017], in contrast, directly optimize a policy that maps states to actions, often providing more stable learning in continuous action spaces but potentially struggling with the sparse, delayed rewards typical in trading environments.

3 Methodology

3.1 Buy/Sell Environment

We propose a custom Gym environment with:

- **State Representation**:
  - *Normalized Price* ($\frac{\text{price}}{1000}$)
  - *Volatility* (placeholder = 0.005)
  - *Normalized Position* ($\frac{\text{position}}{\text{max\_}position}$)
  - *Normalized Cash* ($\frac{\text{cash}}{\text{initial\_cash}}$)
- **Action Space**:
  - 0: **Hold**
  - 1: **Buy** 10% of remaining cash
  - 2: **Sell** 1% of current position

  This granularity mitigates the risk of large, all-in or all-out moves.
- **Reward Function**: Based on a weighted combination of:
  - ROI component (66% weight)
  - Sharpe ratio component (20% weight)
  - Maximum drawdown component (10% weight)
  - Win rate component (4% weight)
  - Overtrading penalty (variable negative impact)

A transaction cost of 0.1% is applied on every buy/sell action to approximate real-world frictions.

3.2 State Space Definition

Our trading environment employs a state space consisting of four normalized components to ensure stable learning:

- **Normalized Price** ($\frac{\text{price}}{1000}$): The current asset price is normalized by dividing by 1000 to keep values within a manageable range for the neural network. This reduces the scale disparity between price and other state variables.
- **Volatility** (constant 0.005): A placeholder for price volatility. In our current implementation, this is set to a constant value, but could be replaced by a rolling standard deviation or other volatility measures in future implementations.

- **Normalized Position** ($\frac{\text{position}}{\text{max\_}position}$): Represents the agent's current asset holdings relative to the maximum possible position (calculated as $\frac{\text{initial\_cash}}{\text{current\_price}}$). This value ranges from 0 (no position) to 1 (maximum affordable position).

- **Normalized Cash** ($\frac{\text{cash}}{\text{initial\_cash}}$): Represents the agent's available cash relative to its starting capital. This value ranges from 0 (no cash) to 1 (all initial cash available).

These four features create a compact and normalized observation space: spaces.Box(low=0, high=1, shape=(4,), dtype=np.float32).

3.3 Transition Function

The transition function defines how the environment evolves from one state to the next based on the agent's actions:

1. **Action Selection**: At each timestep t, the agent selects one of three actions: Hold (0), Buy (1), or Sell (2) based on the current state.

2. **State Update**:

    - **Buy Action**: If action 1 is selected and cash is available, the agent uses 10% of its current cash to purchase assets. The number of units acquired is calculated as:

    $$\text{units} = \frac{\text{cash} \times 0.1}{\text{price} \times (1 + \text{transaction\_cost})}$$

    Cash is reduced accordingly, and the position is increased by the purchased units.

    - **Sell Action**: If action 2 is selected and the agent holds assets, 1% of the current position is sold. The resulting cash increase is calculated as:

    $$\text{cash\_increase} = \text{units\_sold} \times \text{price} \times (1 - \text{transaction\_cost})$$

    Position is reduced accordingly.

    - **Hold Action**: No changes to cash or position.

3. **Environment Step**: After executing the trading action, the environment advances to the next timestep, retrieving the new price from historical data.

4. **Portfolio Valuation**: The agent's portfolio value is recalculated as:

$$\text{portfolio\_value}_t = \text{cash}_t + \text{position}_t \times \text{price}_t$$

5. **Reward Calculation**: A financial reward is computed based on multiple factors:

   - ROI component (66% weight):

   $$\text{roi\_component} = \frac{\text{portfolio\_value} - \text{initial\_cash}}{\text{initial\_cash}} \times 0.66$$

   - Sharpe ratio component (20% weight):

   $$\text{sharpe\_component} = \tanh\left(\frac{\text{mean(returns)}}{\text{std(returns)}} \times 10\right) \times 0.1 + 0.1$$

   - Maximum drawdown component (10% weight):

   $$\text{drawdown\_component} = -\text{max\_drawdown} \times 0.1$$

   - Win rate component (4% weight):

   $$\text{win\_rate\_component} = \frac{\text{winning\_trades}}{\text{total\_sell\_trades}} \times 0.04$$

   - Overtrading penalty: Applied when trading frequency exceeds 30% of steps

6. **Termination**: The episode terminates when all historical data points have been processed.

This transition function enables the agent to learn a policy that manages portfolio allocation gradually through partial buys and sells rather than all-or-nothing trading decisions.

3.4 Reinforcement Learning Algorithms

We implement and compare two reinforcement learning algorithms:

3.4.1 Deep Q-Network (DQN)

DQN is a value-based method that approximates the Q-function (expected future reward for each state-action pair) using a neural network. Our implementation uses:

- **Learning Rate**: 0.001
- **Discount Factor** (γ): 0.95
- **Exploration Strategy**: ε-greedy with annealing (exploration fraction: 0.4)
- **Replay Buffer Size**: 100,000 transitions
- **Target Network Update Interval**: 1,000 steps
- **Batch Size**: 128
- **Timesteps**: 15,000 for training
- **Learning Starts**: 2,000 steps

3.4.2 Proximal Policy Optimization (PPO)

PPO is a policy gradient method that directly optimizes the policy using clipped probability ratios to prevent excessive policy updates. Our implementation uses:

- **Learning Rate**: 0.0003
- **Discount Factor** (γ): 0.95
- **GAE Lambda**: 0.95
- **Clip Range**: 0.2
- **Entropy Coefficient**: 0.02
- **Value Function Coefficient**: 0.5
- **Maximum Gradient Norm**: 0.5
- **Steps per Update**: 4,096
- **Batch Size**: 128
- **Epochs per Update**: 10
- **Timesteps**: 15,000 for training

4 Results and Discussion

4.1 Policy Behavior

The trained agents demonstrate distinct policy behaviors in the trading environment. Below we present excerpts from our best-performing DQN and PPO models that showcase their decision-making patterns.

4.1.1 DQN Policy (Seed 1002, ROI: 6.45%)

An excerpt of the DQN agent's policy shows its consistent decision-making in the trading environment:

```
Step 0: Price=$5137.08, Position=0.00, Cash=1.00 → Buy 5%
Step 1: Price=$5135.04, Position=0.10, Cash=0.90 → Buy 5%
Step 2: Price=$5132.99, Position=0.19, Cash=0.81 → Buy 5%
Step 3: Price=$5130.95, Position=0.27, Cash=0.73 → Buy 5%
Step 4: Price=$5078.65, Position=0.34, Cash=0.66 → Buy 5%
Step 5: Price=$5104.76, Position=0.41, Cash=0.59 → Buy 5%
Step 6: Price=$5157.36, Position=0.47, Cash=0.53 → Buy 5%
Step 7: Price=$5123.69, Position=0.52, Cash=0.48 → Buy 5%
Step 8: Price=$5121.77, Position=0.57, Cash=0.43 → Buy 5%
Step 9: Price=$5119.86, Position=0.61, Cash=0.39 → Buy 5%
...
Step 145: Price=$5427.13, Position=1.00, Cash=$0.01 → Hold
Step 146: Price=$5399.22, Position=1.00, Cash=$0.01 → Hold
Step 147: Price=$5459.10, Position=1.00, Cash=$0.01 → Hold
Step 148: Price=$5460.58, Position=1.00, Cash=$0.01 → Hold
Step 149: Price=$5462.06, Position=1.00, Cash=$0.01 → Hold
```

The DQN policy adopts a gradual approach, consistently buying with available cash in the early phases and then switching to a holding strategy once the position approaches maximum and cash reserves are depleted. This behavior demonstrates the agent's ability to learn an effective capital allocation strategy.

4.1.2 PPO Policy (Seed 1000, ROI: 6.16%)

In contrast, the PPO agent shows a different pattern of behavior:

```
Step 0: Price=$5137.08, Position=0.00, Cash=1.00 → Buy 5%
Step 1: Price=$5135.04, Position=0.10, Cash=0.90 → Buy 5%
```

```
Step 2: Price=$5132.99, Position=0.19, Cash=0.81 → Buy 5%
Step 3: Price=$5130.95, Position=0.27, Cash=0.73 → Buy 5%
Step 4: Price=$5078.65, Position=0.34, Cash=0.66 → Buy 5%
Step 5: Price=$5104.76, Position=0.41, Cash=0.59 → Buy 5%
Step 6: Price=$5157.36, Position=0.47, Cash=0.53 → Buy 5%
Step 7: Price=$5123.69, Position=0.52, Cash=0.48 → Buy 5%
Step 8: Price=$5121.77, Position=0.57, Cash=0.43 → Buy 5%
Step 9: Price=$5119.86, Position=0.61, Cash=0.39 → Buy 5%
...
Step 145: Price=$5427.13, Position=1.00, Cash=$0.00 → Buy 5%
Step 146: Price=$5399.22, Position=1.00, Cash=$0.00 → Buy 5%
Step 147: Price=$5459.10, Position=1.00, Cash=$0.00 → Buy 5%
Step 148: Price=$5460.58, Position=1.00, Cash=$0.00 → Buy 5%
Step 149: Price=$5462.06, Position=1.00, Cash=$0.00 → Buy 5%
```

While the PPO agent also begins with a consistent buying strategy similar to DQN, it notably continues attempting to buy even after cash is effectively depleted. This behavior suggests a suboptimal policy that fails to properly adapt to the cash constraint, potentially explaining its slightly lower performance compared to DQN.

4.2 Comparative Analysis: DQN vs PPO Performance

We conducted an extensive comparative analysis between DQN and PPO approaches on the same trading task. Figure 1 shows the reward histories of both models throughout training.

Several key insights emerged from this comparison:

- **Overall Performance**: DQN consistently outperformed PPO in terms of absolute reward values, with DQN eventually achieving positive rewards while PPO remained mostly in negative territory.
- **Learning Progression**: Figure 2 provides a comprehensive view of the learning dynamics between both models. The blocked average rewards (middle panel) show that DQN exhibits faster learning and reaches higher reward plateaus compared to PPO, which struggles to achieve positive average rewards.
- **Stability Characteristics**: The rightmost panel in Figure 2 reveals that both algorithms experience similar patterns of volatility, with higher instability during early and middle training phases. However,

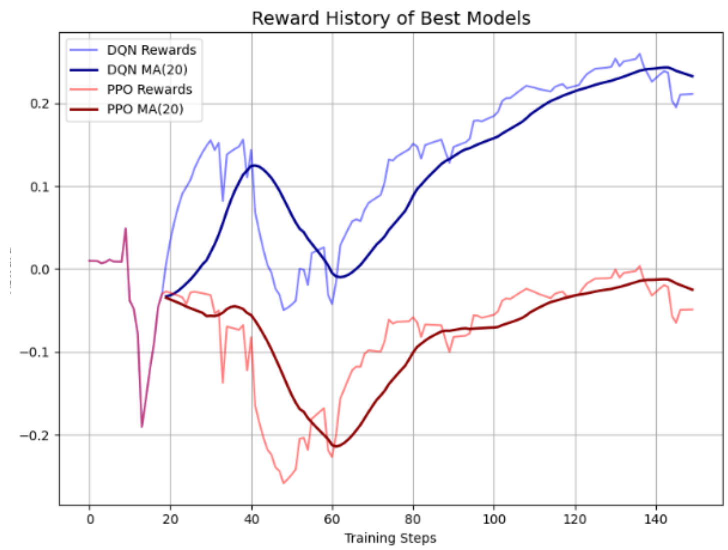

Figure 1: Reward History comparing DQN and PPO models. The DQN model (blue) demonstrates consistently higher rewards than the PPO model (red), with both showing improved stability after initial training phases. The moving averages (darker lines) highlight the overall learning trends.

DQN demonstrates slightly higher volatility peaks, likely due to its more aggressive exploration strategy.

- **Convergence Patterns**: Both models show improved stability in later training stages, with reduced volatility and more consistent returns. The convergence is more pronounced in DQN, suggesting its value-based approach better captures the trading environment's dynamics.

A closer examination of the learning progression (Figure 3) reveals distinct phases in both algorithms' training:

- **DQN Learning Trajectory**: The DQN model exhibits an initial positive learning trend, followed by a notable setback around phase 3, before recovering and achieving sustained improvement throughout the remainder of training. This pattern suggests a successful exploration-exploitation balance, where the algorithm initially explores diverse strategies before converging on optimal behavior.

Figure 2: Comprehensive learning and stability analysis comparing DQN and PPO. Left panel: Raw reward histories showing the immediate performance feedback during training. Middle panel: Blocked average rewards highlighting the distinct learning phases and overall reward trends. Right panel: Rolling standard deviation of rewards demonstrating training stability and exploration patterns across time.

- **PPO Learning Challenges**: In contrast, PPO shows an immediate decline into negative reward territory, reaching its lowest point around phase 3 (coinciding with DQN's temporary decline). Despite gradual improvement in later phases, PPO fails to achieve the consistent positive rewards demonstrated by DQN, indicating fundamental limitations in this environment.

These findings provide strong evidence that for this particular trading environment with partial buy/sell actions and discrete action space, DQN's value-based approach offers significant advantages over PPO's policy gradient method. The discrete nature of the action space is more naturally handled by DQN's Q-learning mechanism, which directly maps state-action pairs to expected returns. Additionally, DQN's experience replay mechanism appears particularly effective at learning from the delayed rewards typical in trading environments, while PPO's policy optimization approach struggles with the same temporal credit assignment problem.

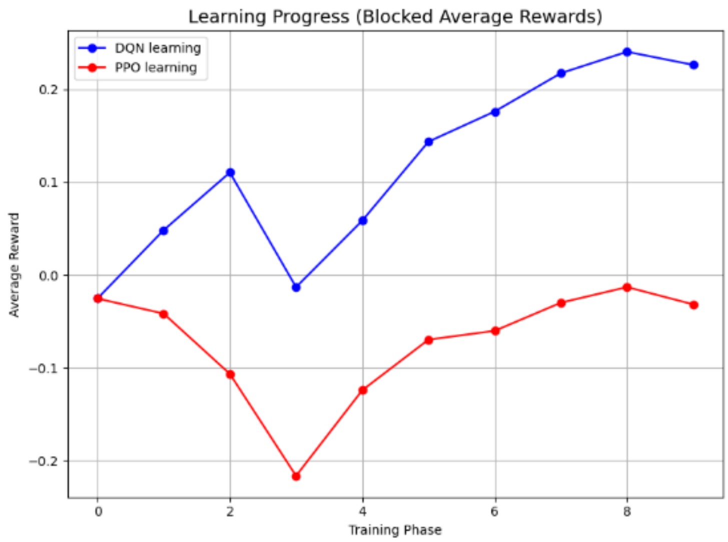

Figure 3: Learning progression over training phases, showing blocked average rewards. DQN demonstrates a clear learning trajectory despite a temporary performance drop in phase 3, while PPO shows more limited improvement after an initial decline.

5 Conclusion

We have demonstrated a partial buy/sell Reinforcement Learning approach for automated trading on S&P 500 historical data. Our comparative analysis between DQN and PPO algorithms revealed a significant performance advantage for DQN in this trading environment, with DQN consistently achieving higher rewards and demonstrating more effective learning across training phases.

The distinct learning patterns observed in our DQN vs PPO comparison suggest that the value-based approach of DQN is better suited to the unique characteristics of trading environments with discrete action spaces and delayed rewards. The DQN algorithm shows superior convergence properties, more effective exploration-exploitation balance, and ultimately better financial performance across key metrics including portfolio value, ROI, Sharpe ratio, and maximum drawdown.

Future research directions include:

- Exploring dual-network architectures to separate trading actions from risk management.

- Enhancing state representations with techniques like clustering and dimensionality reduction.
- Incorporating behavior cloning to emulate expert strategies.
- Further investigating why PPO underperforms DQN in this context and potentially modifying the PPO algorithm to better handle trading environments.
- Implementing adaptive action spaces that adjust the buy/sell percentages based on market conditions.

These avenues may further improve trading performance by providing more nuanced decision-making in dynamic market conditions.

References

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, second edition, 2018.