DQN learning

Reinforcement learning in Computational Finance

BT21CSE077 Robin Bansal
BT21CSE084 Data Kumar
BT21CSE146 Sony Bhagavan

Executive Summary

This report presents a comprehensive analysis of applying **Deep Q-Network** (**DQN**) reinforcement learning to stock market trading. The implemented system uses recurrent neural networks to analyze historical stock data along with technical indicators to make intelligent trading decisions. When tested on Apple (AAPL) stock data from **2020-2022**, the **DQN** strategy achieved a total return of **60.89%**, significantly outperforming a baseline moving average crossover strategy that returned only **8.45%**. This implementation demonstrates the potential of reinforcement learning for developing effective algorithmic trading strategies.

Introduction:

Algorithmic trading has revolutionized financial markets, with machine learning approaches gaining significant traction in recent years. This project explores the application of Deep Reinforcement Learning (DRL) to stock trading, specifically focusing on a Deep Q-Network (DQN) implementation. Unlike traditional approaches that rely on predefined rules, reinforcement learning allows an agent to learn optimal trading strategies directly from market data through trial and error.

LEARNING PHASE: DQN-LEARNING IN ACTION:

The learning phase of a Deep Q-Network (DQN) involves training a neural network to make smart decisions by interacting with an environment. Through repeated experiences, the DQN learns to predict the value of actions and gradually improves its policy.

Initialize Q

Start by initializing the neural network (Q-network) with random weights to estimate Q-values for actions. A target network Q′(s,a;θ−)Q'(s, a; \theta^-)Q′(s,a;θ−) is also initialized to stabilize training.

Action Selection DQN

Select an action using the ε-greedy policy, balancing exploration and exploitation based on the current Q-values.

Perform Action

Execute action **a**(**t**) in environment, receive reward **r**(**t**)**,** give penalty, and observe next state **st+1**. Store transition (**st,at,rt,st+1**) in Replay Buffer for experience replay.

.

Measure Reward

Calculate the target Q-value using the Bellman Equation:

yt=rt+γa′maxQ′(**st+1,a′;θ−**)

Here, γ\gammaγ is the discount factor that weighs future rewards.

Update Q

IStore the experience, compute the target Q-value, and update the Q-network through backpropagation to reduce prediction error.

Methodology

Data Collection and Preprocessing

- **Source:** Historical daily stock data for Apple Inc. (AAPL) from Jan 2020 to Mar 2022 using Yahoo Finance (yfinance).

Indicators Used:

- **Trend:** SMA (20, 50), EMA (20), MACD
- **Momentum:** RSI, Stochastic Oscillator
- **Volatility:** Bollinger Bands, ATR
- **Volume:** OBV, VWAP
- **Feature Engineering:** Includes percent changes, high-low ratios.
- **Normalization:** Feature values are scaled by their mean to enhance training stability.

Technical Indicators

From data, we extract **Close series , High series, Low series, Volume series**

📊 **Trend Indicators**

- SMA_20, SMA_50 = 20/50-day simple moving averages
- EMA_20 = 20-day exponential moving average

📉 **MACD** (**Moving Average Convergence Divergence**)

- MACD line, MACD signal, MACD histogram

⚡ **Momentum Indicators**

- RSI = Relative Strength Index
- Stochastic Oscillator (%K and %D)

🌊 **Volatility Indicators**

- Bollinger Bands (High, Low, Mid)
- ATR = Average True Range

📦 **Volume Indicator**

- OBV = On-Balance Volume

🧮 **VWAP** (**Volume Weighted Average Price**)

- VWAP = (cumulative volume × price) / cumulative volume

🔁 **Price Differentials**

- Close_Pct_Change = % change in closing price
- High_Low_Pct = (High - Low) / Low

Reinforcement Learning Framework:

The implementation follows the standard reinforcement learning paradigm:

State Space: Each state consists of a window of historical data points (default: 128 time steps) with all the calculated features

Action Space: The agent can select from 11 possible actions:
- Sell 100%, 80%, 60%, 40%, or 20% of current holdings
- Hold (no action)
- Buy shares worth 20%, 40%, 60%, 80%, or 100% of available cash

Reward Function: Based on portfolio value changes with penalties for:
- Transaction costs (0.1%)
- Overtrading frequency
- Significant drawdowns

Initilisations and Actions

Value Initialization:

- **Architecture =** 'LSTM'
- **Dense Layer** = 2
- **Dense Size** = 128
- **Dropout Rate** = 0.2
- **Trade Fees** = 0.001%
- **Overtrading P.** = 0.001%
- **Drawdown Rate** = 0.1
- **Starting Cash =** 10000
- **Starting Shares =** 0
- **Window Size =** 128
- **Actions Mapping =** 11

Action mapping (for 11 actions):

0: Sell 100% of shares
1: Sell 80% of shares
2: Sell 60% of shares
3: Sell 40% of shares
4: Sell 20% of shares
5: Hold
6: Buy shares worth 20% of cash
7: Buy shares worth 40% of cash
8: Buy shares worth 60% of cash
9: Buy shares worth 80% of cash
10: Buy shares worth 100% of cash

Reward and Penalty

In Reinforcement Learning, the reward signals how good or bad an action was.
✅ Encourages stable growth
⚠️ Penalizes: Risky behavior, Excessive trades, Sharp drawdowns
🧠 Leads to profitable and safer strategies over time.

Penalty	Purpose	Trigger	Formula
💸 Trade Fee	Simulates transaction cost	Every trade	Reflected in portfolio value
🔁 Overtrading	Discourages frequent switching	trade_count > 10	penalty × reward
📉 Drawdown	Prevents sharp losses	Drop > 5% from peak	penalty × drawdown × abs(reward)

Step	Description	Formula
1	📊 Measure current portfolio	portfolio_value_before = cash + shares × price
2	🎮 Execute action (Buy/Sell/Hold)	Adjust cash and shares
3	📈 Get next state portfolio	portfolio_value_after = cash + shares × next_price
4	💰 Raw reward = portfolio delta	reward = after - before
5	⚠️ Apply penalties	💸 Trade Fee + 🔁 Overtrading + 📉 Drawdown
6	✅ Final reward returned	adjusted_reward = reward - penalties

Core Features:

- **Experience Replay:** Transition storage for batch learning
- **Target Network:** Stabilizes Q-value updates
- **Epsilon-Greedy Policy:** Controls exploration-exploitation with decay

Training Process

- **Data Split:** 80% training, 20% testing.
- **Incremental Training:** Monthly data blocks for progressive learning.
- **Experience Collection:** Environment interaction for state-action-reward tuples.
- **Network Update:** Every 4 steps via replay buffer sampling.
- **Target Sync:** Every 100 steps.
- **Model Saving:** After each month's training cycle

DQN Architecture:

Model Variants: RNN, LSTM, and GRU supported (default: LSTM).
Input: Historical feature window
Recurrent Layer: Sequential data processing
Dense Layers: Two fully connected layers with ReLU & dropout
Output: Q-values for 11 actions

Evaluation Metrics

The trading strategy is evaluated using standard financial performance metrics:

- **Total Return:** Percentage gain or loss over the testing period
- **Sharpe Ratio:** Risk-adjusted return (higher is better)
- **Maximum Drawdown**: Largest percentage drop from peak to trough
- **Win/Loss Ratio:** Ratio of profitable to unprofitable trades
- **Annualized Volatility**: Standard deviation of returns, annualized

Trading Behavior Analysis

The agent demonstrated sophisticated trading behavior with several notable characteristics:

- **Adaptive Positioning:** Adjusted trade size based on market trends.
- **Pattern Detection:** Recognized price movements and signals.
- **Risk Control:** Minimized losses using drawdown penalties.

Limitations

Despite its strong performance, the implementation has several limitations:

- **Market Dependency:** May underperform in changing regimes.
- **Overfitting Risk:** High complexity could cause overfitting.
- **Heavy Computation:** Requires significant training resources
- **Single Stock:** It work only with 1 stock at a time unlike real time

Results and Analysis :

Performance Overview When tested on Apple (AAPL) stock from January 2020 to March 2022, the DQN trading strategy achieved impressive results:

Metric	DQN Strategy	Baseline Strategy
Total Return	60.89%	8.45%
Sharpe Ratio	0.0906	0.2833
Max Drawdown	-75.30%	N/A
Win/Loss Ratio	1.0806	N/A
Annualized Volatility	0.2664	N/A

The DQN strategy significantly outperformed the baseline moving average crossover strategy in terms of total return, though interestingly, the baseline strategy achieved a better Sharpe ratio, indicating better risk-adjusted returns despite lower overall performance.

Conclusion:

This implementation demonstrates the potential of reinforcement learning for algorithmic trading. The DQN-based approach achieved impressive returns on Apple stock, significantly outperforming a traditional moving average crossover strategy. The combination of recurrent neural networks with technical indicators allowed the model to capture complex patterns in stock price movements. While challenges remain, particularly regarding overfitting and real-world implementation, the results suggest that reinforcement learning offers a promising avenue for developing sophisticated trading strategies that can adapt to changing market conditions. The modular design of the implementation allows for easy extension and modification, providing a solid foundation for further research and development in reinforcement learning for financial markets.

Interpretability Achieved:

- The DQN agent exhibited clear decision patterns such as adaptive position sizing and risk-aware actions, making its behavior understandable.
- Its actions aligned with traditional trading logic, providing insights into learned strategies beyond black-box predictions.

Thank you!

Group-14

BT21CSE077 Robin Bansal
BT21CSE084 Data Kumar
BT21CSE146 Sony Bhagavan