Stock Market Predictor Using Temporal Difference

San Jose State University

CMPE 260

Fall 2021

Outline

Team Members
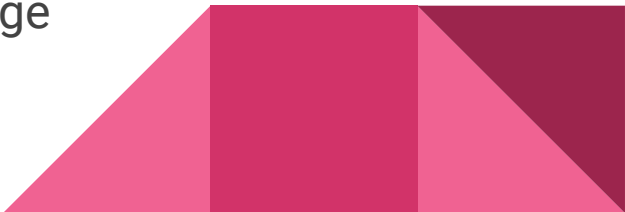
Jimmy Liang

Rohan Kumar

Samer Baslan

Problem Statement

The prediction of stock movement has been of interest to traders for many years.

Technical indicators are used by traders to gain insight into the supply and demand of securities and market psychology.

Technical indicators are heuristic or pattern-based signals produced by the price, volume, and/or open interest of a security or contract used by traders who follow technical analysis.

Examples of common technical indicators include the Relative Strength Index (RSI), Money Flow Index (MFI), stochastics, moving average convergence divergence (MACD), and Bollinger Bands.

Problem Statement

By analyzing historical data, technical analysts use indicators to predict future price movements. Together, these indicators form the basis of technical analysis.

Metrics, such as trading volume, provide clues as to whether a price move will continue. In this way, indicators can be used to generate buy and sell signals.

Current approaches have a limitation in that they are mainly based on highly supervised learning which is not so adequate for solving problems with long-term goals and delayed rewards.

Motivation

We now have the capability to create a Reinforcement Learning agent that can learn the insights and trends behind the market to identify the biggest profit yielding stocks, and to identify the stocks that must be sold.

This can be an extra step of research when looking for what stocks to buy or sell.

The goal is to create a reinforcement learning agent which seeks to maximize the reward, which we define as the maximum cumulative return for a given stock portfolio in a given time period.

The agent, following its policy, will execute either buy, hold, or sell trades for stocks within the portfolio on a daily basis.

Methodology

To train the trading agent, we leveraged a custom market environment that provides price and other information, offers trading-related actions, and keeps track of the portfolio to reward the agent accordingly.

More specifically, the environment samples a stock price time series for a single ticker using a random start date to simulate a trading period that contains 63 days (1 fiscal quarter).

The state contains the scaled price and volume, as well as some technical indicators like the percentile ranks of price and volume, a relative   strength index (RSI), as well as 2, 5, 10 and 21-day returns.

Methodology

The agent can choose from three actions:

1. **Buy**: Invest capital for a long position in the stock
2. **Hold**: Hold cash only
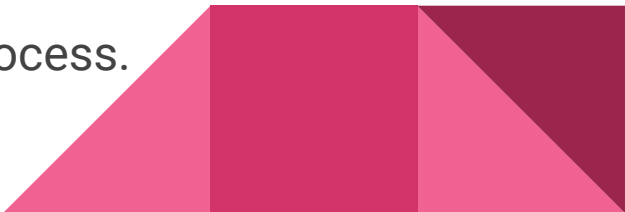3. **Sell**: Take a short position equal to the amount of capital

The environment accounts for trading cost, which is set to 10 basis points (BPS). It also deducts 1 BPS per period as time penalty. It tracks the net asset value (NAV) of the agent's portfolio and compares it against the market portfolio, which trades frictionless to raise the bar for the agent.

Implementation

We use a Double Deep Q-Learning Network (DDQN) agent with an Epsilon-Greedy policy. The trading environment consists of three classes that interact to facilitate the agent's activities:

1. **DataSource** class loads a time series, generates a few features, and provides the latest observation to the agent at each time step.
2. **TradingSimulator** class tracks the positions, trades and cost, and the performance. It also implements and records the results of a buy-and-hold benchmark strategy.
3. **TradingEnvironment** class orchestrates the entire process.

Implementation

Q-learning is a model-free, temporal difference (TD) algorithm. We can think of model-free algorithms as trial-and-error methods. The agent explores the environment and learns from outcomes of the actions directly.

It is a TD algorithm because predictions are reevaluated after taking a step. We measure how the last action is different from what we estimated initially, without waiting for a final outcome.

From the start, the agent knows the possible states and actions in an environment. Then the agent discovers the state transitions and   rewards by exploration.

Implementation

In epsilon-greedy, the agent uses both exploitations to take advantage of prior knowledge and exploration to look for new options.

The epsilon-greedy approach selects the action with the highest estimated reward most of the time. The aim is to have a balance between exploration and exploitation. Exploration allows us to have some room for trying new things, sometimes contradicting what we have already learned.

With a small probability of ε, we choose to explore, that is, not to exploit what we have learned so far. In this case, the action is selected randomly, independent of the action-value estimates.

Results

With Epsilon-Greedy policy, the agent either met or outperformed the market moving average.

Without any policy (effectively where **ε = 1.0**, i.e. random impulsive actions), the agent consistently underperformed and lost money along the way.

This demonstrates the exploration versus exploitation tradeoff. When the agent explores, it can improve its current knowledge and gain better rewards in the long run.

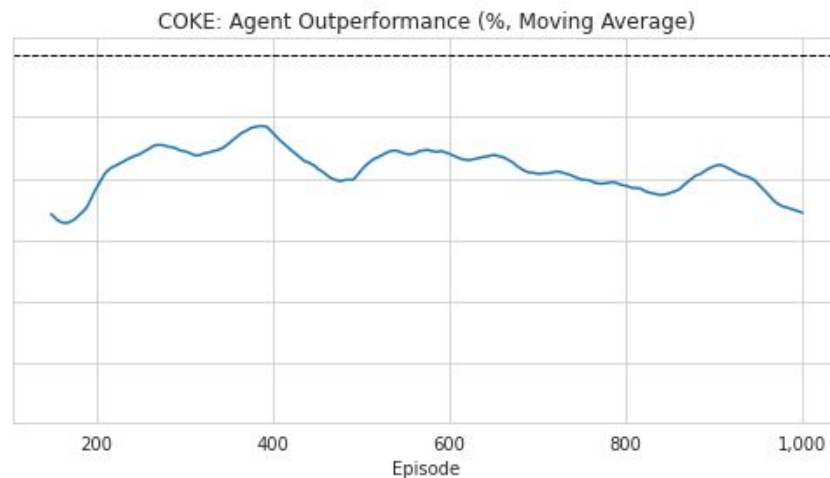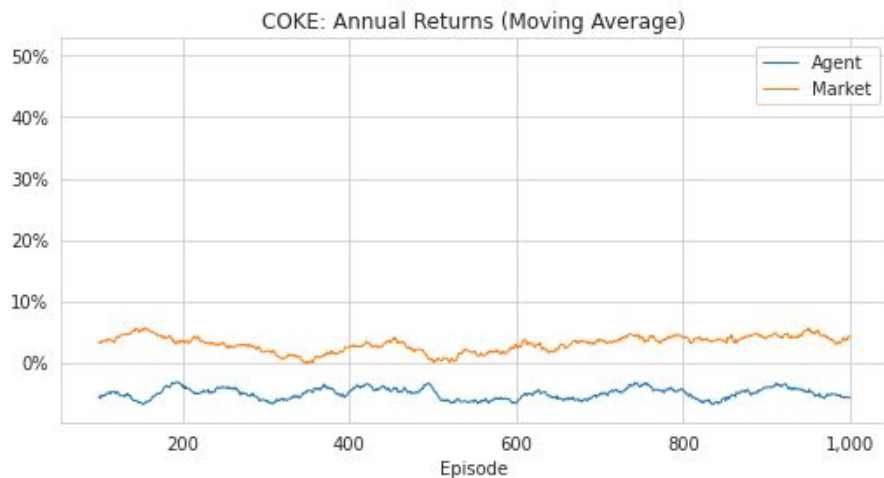A greedy agent can get stuck in a sub-optimal state, which is especially dangerous in trading.

Case #1 - COKE
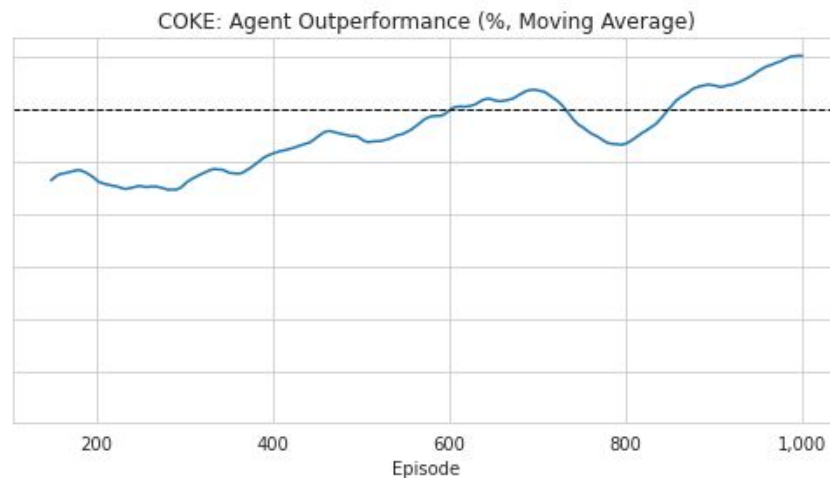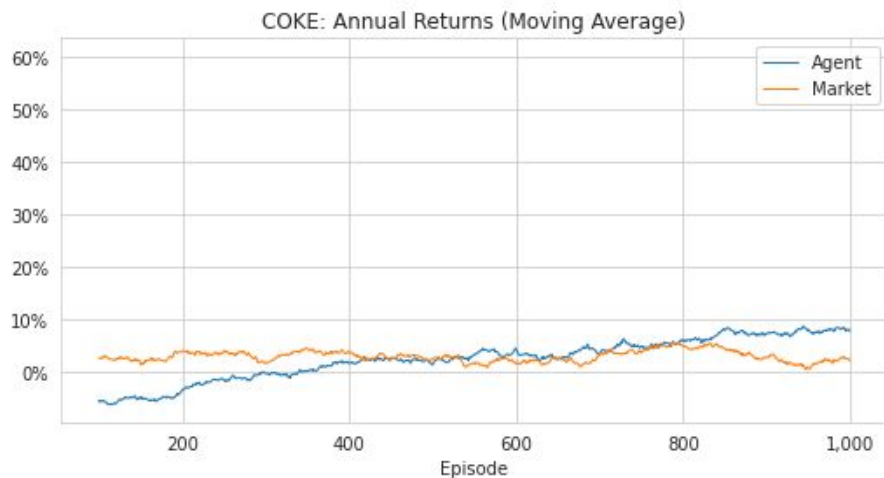
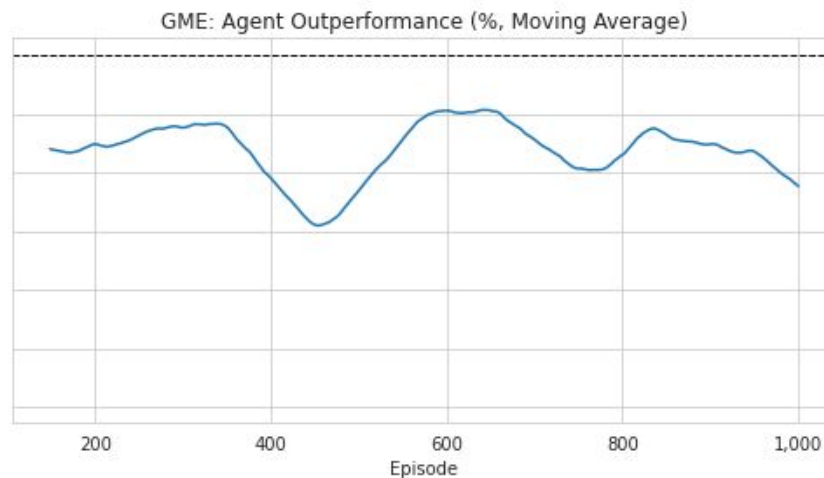Coca-Cola with Epsilon-Greedy Policy

Case #1 - COKE

Coca-Cola without any policy
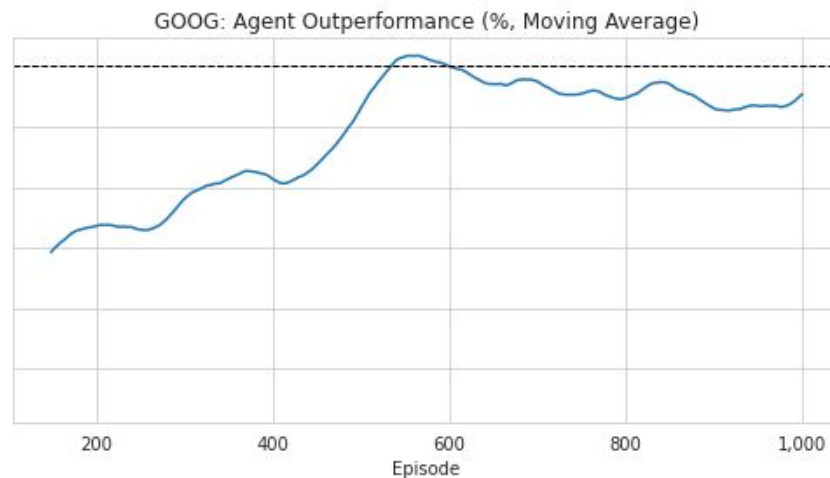
Case #2 - GME
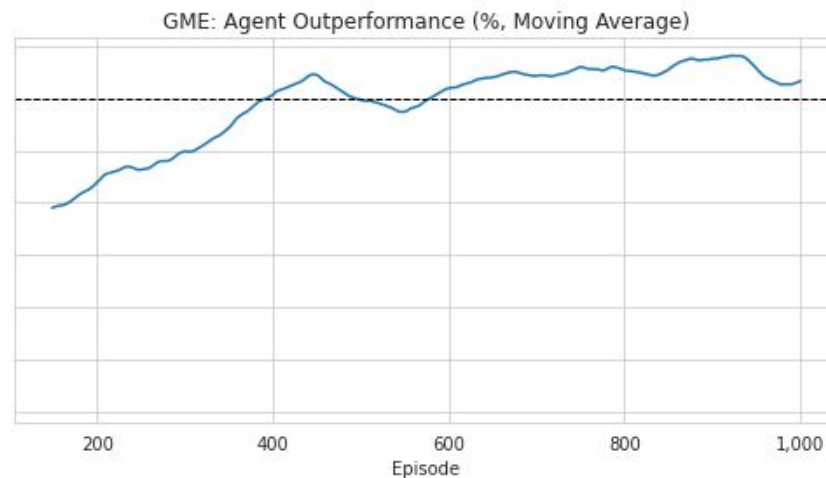
GameStop with Epsilon-Greedy Policy
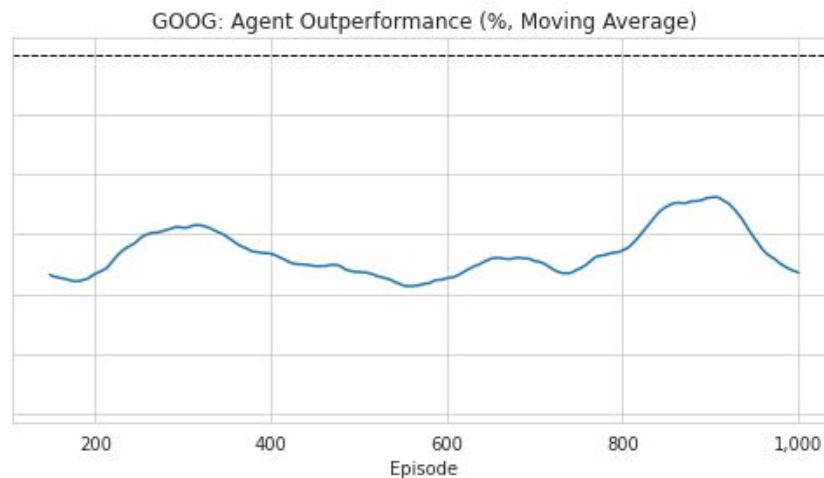
Case #2 - GME
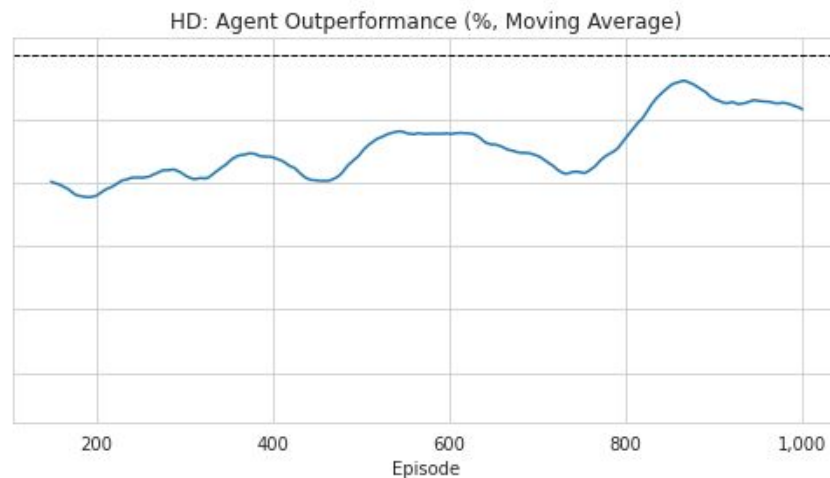

GameStop without any policy

Case #3 - GOOG

Google with Epsilon-Greedy Policy

Case #3 - GOOG

Google without any policy
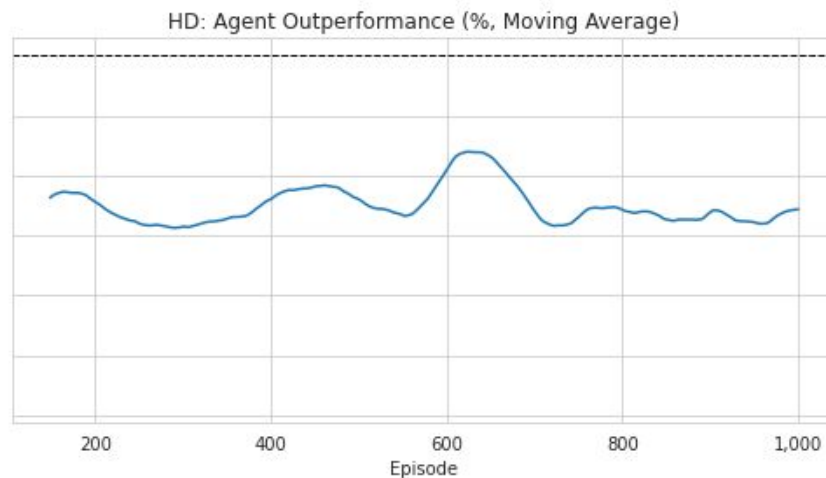
Case #4 - HD

Home Depot with Epsilon-Greedy Policy

Case #4 - HD

HD: Annual Returns (Moving Average)
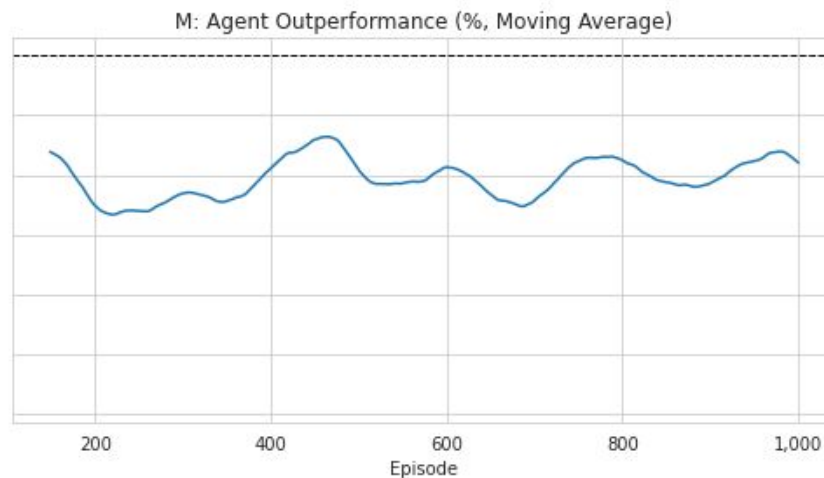HD: Agent Outperformance (%, Moving Average)

Home Depot without any policy

Case #5 - M
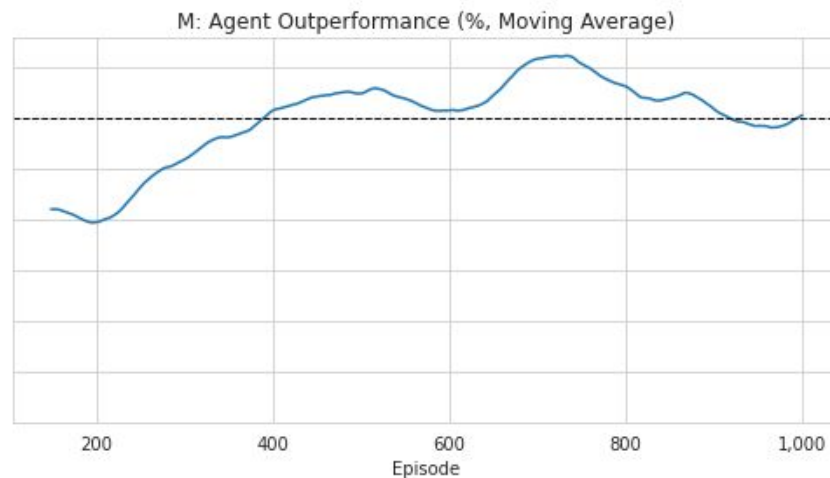
Macy's with Epsilon-Greedy Policy

Case #5 - M

Macy's without any policy

Technical Walkthrough