Searching for Optimal Autonomous Trading Strategies via Reinforcement Learning

Malcolm Mashig
CS 4710: Artificial Intelligence
University of Virginia
Charlottesville, VA
mjm6jy@virginia.edu

***Abstract*—This report details a final project for CS 4710 Artificial Intelligence at the University of Virginia. The project involves the utilization of reinforcement learning to train an autonomous stock portfolio manager that simulates various strategies in historical market environments, converging to the use of the strategy that obtains the highest average reward.**

Keywords*—*Reinforcement Learning, Automated Trading, Trading Simulations

I. PROBLEM

A consistently successful stock trading strategy is rare and difficult to come by.

Stock prices are often viewed as random walks, a sequence of random changes built on top of each other. For this reason, predicting stock price movement is a seemingly impossible task, and even experts in time series analysis may tell you that the best guess for the next observed stock price is the current price. This is intuitively true, because if future prices were predictable, then everyone would easily make money in the stock market, and that is not the case.

The factors that lead to changes in the prices are numerous and too complex to simplify into a model. For this reason, model-free, empirical experimentation is necessary to highlight the do's and don'ts of stock trading.

While it is difficult to determine a consistently strong trading strategy, it is easier to experiment with a multitude of strategies and determine those that are better than others. This makes stock trading a prime area to apply reinforcement learning.

Manual (human-decided) investment strategies that lack automation are impractical to experiment with. Humans are prone to biased, inconsistent, unexplainable, and slow thinking. In this way, we cannot fairly test how a human would decide trades in historical market environments because they would be biased by knowledge of the future and it would simply take too long.

For these reasons, an automated portfolio manager encoded to act in different but reproducible ways is necessary to conduct this experimentation.

Designing such an automated portfolio manager was the goal of this project, in order to identify the best trading strategies from a pool of many.

II. IMPORTANCE

Besides the potential to make money with a strong trading strategy, this approach to automated trading experimentation is important in a number of ways.

First and foremost, an automated trader than can simulate outcomes in historical market environment lays the groundwork for much more intricate stock trading analysis.

There are many constraints and imperfections when making trades. Different amounts of cash will be available at different times. Prices when a decision to trade is made may differ than the price at which the trade is executed. These are just a few examples of the nuances in stock trading, and the automated trading framework accounts for many of such examples, by simulating through real market situations.

The framework makes it easy to add more nuances – in the form of parameters – and allows one to test what rules for portfolio management will best navigate these nuances.

The framework is also easy to roll out into real-time trading. If competing strategies make it unclear what the optimal strategy truly is, the integration of reinforcement learning within the framework will lead the autonomous portfolio manager to converge to the selection and use of the best strategy.

III. APPROACH

A. *Data*

In order to train an autonomous portfolio manager to perform well in a variety of market environments, a large dataset of stock prices was necessary.

Daily opening and closing prices from 2000 through 2019 for over 5,000 NYSE and NASDAQ stock symbols (over 12 million total observations) were obtained via the Tiingo [2] and AlphaVantage [1] financial market APIs. To ensure the accuracy of the prices, only those that matched in the two data sources were kept.

To mimic multiple potential market environments, the data was separated into over 500 studies, where a study is a consecutive 50-day period with a batch of around 500 stocks available to trade.

Some time periods in history were reserved solely for validation and for testing so that we could later evaluate trading techniques in completely unseen market environments. A ~60-20-20 split was then conducted to separate the studies into training, validation, and testing sets.

Minor cleaning was necessary to prevent problems from arising during training. For example, observations of stock prices occurring shortly after a stock split (when companies alter the number of outstanding shares by some factor, thus changing price by some factor) were removed, as the resulting price change might have been misinterpreted by the feature used to decide entries and exits. Additionally, penny stocks (those trading for less than $1) were removed as they were prone to unusually extreme returns (percent changes) which might lead to extremely strong or poor portfolio performance.

B. *Simulation*

To simulate portfolio management in each study, a structured simulation function was required. For each study, the simulator iterates through each day sequentially, deciding entries and exits and updating wealth, equivalent to the cash available plus the value of holdings, along the way.

Entries and exits were simply decided based on “relative price”, the average return of the stock over an n-day trailing window.

To mimic the delay between deciding and actually executing trades, closing prices were referenced to decide trades, and the next day’s opening prices (highly correlated with closing prices) were referenced to execute trades.

For each day, the simulator evaluated wealth first, decided exits next, then decided entries based on the estimated change in cash from those exits. A portion of cash was set aside at all times to correct for differences between opening and closing prices.

After simulating a study, the percent change in wealth each day was compared to the benchmark return, which will be explained in the next section.

C. *Benchmark*

The benchmark is perhaps the most important factor in the training process, as it directly influences what the portfolio manager would consider optimal.

The benchmark used by the simulator was the maximum of either the market return (percent change in wealth if invested in all available stocks throughout the entire study period) or a desired return. In this way, beating the benchmark would require both outperforming the market and outperforming an expected growth in wealth.

Beating the benchmark would imply that the trading technique used was both useful, in that it outperforms the arbitrary strategy of riding the market, and risk averse, in that it does not lead to a decrease in wealth (if the expected return was 0, for example), even when the general market performs poorly. That said, this benchmark is fairly strict and difficult to beat.

D. *Parameters*

A number of parameters were available to tune the simulation function. The parameters were:

- Trading philosophy: either reversion (buying low, selling high) or momentum (buying high, selling low).
- Trailing window size and cutoffs used to calculate relative price
- Maximum proportion of wealth to invest in stocks, fixed at 0.9
- Maximum proportion of wealth to invest against stocks (short trades)
- Range of wealth to invest for one trade, fixed at 0.5% - 1.5%
- Maximum number of trades per day, fixed at 20 long trades and 10 short trades
- Whether or not to exit after holding a stock for a certain period of time (timed exits)

- Whether or not to exit after a stock achieves a certain return (stop losses / wins)

During training, to limit the number of possible strategies (combinations of the above parameters), only six of the parameters were experimented with. These include trading philosophy (reversion vs momentum), window size (5 vs 18 days), maximum short proportion (0.05 vs 0.2), timed exits (5-day limit vs no limit) (short-trades only), stop losses (-5% return limit vs no limit), and stop wins (5% return limit vs no limit). Together, they constructed 64 (6^2) different combinations / strategies to attempt during training. The other parameters were fixed as specified above.

E. *Hyper-parameters*

A number of hyperparameters were also available to tune the training process and/or assist with convergence. The hyper-parameters were:

- Expected return, used to calculate the benchmark and thus the reward
- Exploration probability (epsilon): probability of choosing a random strategy
- Epsilon decay: the factor by which to decrease the exploration probability after each study simulation
- Whether or not to randomly-sample entries from qualified entries
- Epochs of training
- Collapse rate: the factor by which to decrease the pool of possible strategies after each epoch

During the first training process, expected return was set at 0.05, epsilon was set at 0.4, decay was set at 0.998, random-sampling was used, 4 epochs were run, and the collapse rate was set at 0.5.

F. *Training*

To train the portfolio manager, a reward table including all 64 strategies was made available and the probability of selecting each strategy was initialized to be uniform.

As the simulator iterates through each study in random order, the reward table is updated to include the sum of rewards, each reward being the return observed minus the benchmark.

From this, the reward table calculates the average reward for each strategy – the respective sum of rewards observed divided by the number of times the strategy was attempted – and the average reward is used to update the probability of selecting each strategy in the table.

Min-max normalization was utilized so that the worst strategy had zero chance of being selected (unless a random selection was made) and so that the best strategy had the highest chance of being selected. All strategies were then normalized to add up to one.

For each study, there was a certain (epsilon) chance that a random strategy would be selected (regardless of reward) in order to promote exploration of all strategies and to avoid an impulsive conclusion of what the optimal strategy was. After each simulation, epsilon is decreased slightly to promote exploitation of the optimal strategy, after the others have been adequately explored/attempted.

After each epoch of training, the reward table (pool of potential strategies) is collapsed by a certain factor (one of the hyper-parameters) in order to promote convergence away from the worst-performing strategies.

IV. RESULTS

A. *Validation*

After training, the best strategy finished with an average return 0.30 below the benchmark.

Throughout training, the reward obtained for each simulation increased by 3e5 on average, demonstrating the convergence towards better trading strategies as exploration decreased and exploitation increased.

When tested on the validation studies, the empirically optimal strategy performed slightly worse than in training, being on average 0.362 below the benchmark. As a silver lining, the strategy performed better (not worse, as is feared) in completely unseen time periods that were not included in training.

After adjusting the epsilon and epsilon decay parameters, to 0.2 and 1 respectively, a second training session yielded a different "optimal" strategy that also performed on average 0.30 below the benchmark. It also similarly performed 0.363 below the benchmark on average during validation, and again optimistically better in unseen time periods.

B. *Testing*

Despite the minor difference in training and validation performance, I decided to move forward with the first training session's optimal strategy for the final testing phase.

When attempted on the testing studies, it performed on average 0.341 below the benchmark, and performed nearly the same in unseen vs familiar time periods.

C. *Overview*

While the results may not be satisfactory when compared to the strict benchmark that was set or if hoping to make consistent money, the training framework still demonstrated clear and successful reinforcement learning, which was ultimately the main goal of the project.

I believe this framework can be effectively applied at trading firms who wish to test a small number of competing trading models, in historical market environments or in real-time (current) markets.

There are many limitations to the framework as it is. For instance, trades are decided only with past prices, only two daily price-points are available for each symbol, and many nuances of trading (dividends, stock-splits, taxes, maintenance requirements) are not incorporated.

As the complexities of trading – in the form of more parameters -- are included, the framework will lose value because the pool of potential strategies will explode, and exponentially more training will be required to adequately explore/attempt each of the strategies.

That said, the most important improvement would be to implement some form of gradient descent that can be applied to the continuous parameters – such as trailing window size, entry/exit cutoffs, max holding periods, etc. – into the training process. This would leave only combinations of discrete parameters – trading philosophy, whether to have timed exits, or stop loss/win exits, etc. – which would prevent the pool of strategies from becoming too large.

REFERENCES

[1] https://www.alphavantage.co

[2] https://www.tiingo.com