Stock Trading Strategy Using Deep Reinforcement Learning and Genetic Algorithm

Group 26

Maayann Affriat

Ilan Benhamou

Ruben Hoba

Michael Wolhandler

Abstract

Stock trading involves buying and selling stocks frequently in an attempt to time the market. It is a very challenging task to design a profitable strategy.
Investors who trade stocks do extensive research, often devoting hours a day to following the market. They rely on technical stock analysis, using tools to chart a stock's movements in an attempt to find trading opportunities and trends. We explore an innovative approach based on deep reinforcement learning and genetic algorithm to build trading agents which can manage portfolios.

Introduction

Profitable stock-trading strategy is fundamental to investment companies.
The goal of stock traders is to capitalize on short-term market events to sell stocks for a profit, or buy stocks at a low. Managing a portfolio is challenging, as it requires traders to take all relevant factors into consideration in a dynamic and complex stock market.
In order to become a stock trader, a great knowledge of the markets and of the capital for investment is required. Also, trading requires a lot of research and hence requires an immense amount of time to keep track of the financial markets.
Algorithmic trading, the most widely used form of AI in the financial industry, uses complex and advanced mathematical models to make transaction decisions on behalf of humans. In fact, it allows us to make smarter and faster investing decisions with less risk. Another advantage is that it lies in the ability to analyze the potential impact of trade on the market, adding it to the protection of the emotions (like fear and greed) and a more effective execution of orders a trading bot offers.
In this project, we focus on portfolio consisting of one single stock and explore and compare the performance of two AI algorithm: Deep Q-learning (DQN) and Genetic Algorithm (which will be describe later).
To evaluate our solution, we will use multiple performance indicators: Profit/Loss of the agent at the end of the trading test period, Sharpe Ratio – return of the trading activity compared to its riskiness.
We measure and compare our agent with some basics trading strategies like Buy and Hold, or MACD (moving average convergence divergence).

Methods

1. Deep Q-Learning

In reinforcement learning, an agent is trained in the environment through trial-and-error exploration of some action policy by receiving states and rewards from the environment and taking actions to reach better rewards.
The goal of stock trading is to maximize returns,while avoiding risks. DRL solves this optimization problem by maximizing the expected total reward from future actions over a time period. Stock trading is a continuous process of testing new ideas, getting feedback from the market, and trying to optimize the trading strategies over time. We can model the stock trading process as the Markov decision process which is the very foundation of Reinforcement Learning, and defining the reward function as the change of the portfolio value, Deep Reinforcement Learning maximizes the portfolio value over time. Formally, the problem can be described as follow:

State: We define our state space as $s \in S = \mathbb{R}^t$, a vector of length *t*, which corresponds to the last *t* prices of the stock.

Actions: We define our action space as, $a \in A = \{\text{HOLD, BUY, SELL}\}$.

Reward: The reward *r* at step *t* is the daily returns that the agent gets from his previous action. It is defined as

$$r_t = \frac{c_t - c_{t-1}}{c_t},$$

where c_t is the agent capital at time step t.

The discounted reward R is defined as

$$R = \sum_{t=0}^{\infty} \gamma^t r_t,$$

where $0 \leq \gamma \leq 1$ is the discounted factor.

We define the optimal action-value function $Q^*(s, a)$, as the maximum expected return achievable by following any strategy, after seeing some sequence s and then taking some action a,

$$Q^*(s, a) = \max_{\pi} E[R_t | s_t = s, a_t = a, \pi],$$

where π is a policy mapping states to actions.

The Bellman equation is defined as,

$$Q^*(s, a) = \max_{\pi} E\left[r + \gamma \max_{a'} Q^*(s', a') | s_t = s, a_t = a, \pi\right] ,$$

Then, in the classic Q-Learning, we estimate the action value function by using the Bellman equation as an iterative update,

$$Q^{new}(s_t, a_t) = Q^{old}(s_t, a_t) + \alpha * \left(r_t + \gamma * \max_a Q(s_{t+1}, a\right) - Q^{old}(s_t, a_t),$$

The iterative approach doesn't work for non-discrete observation space. Thus, it is more common to use a function approximator such as neural network.
In the DQN method, a neural network approximator with weights θ is used to approximate the action value function by minimizing a loss function at each iteration *i*,

$$L(\theta i) = E_{(s,a)\sim\rho(\cdot)}[(y - Q(s, a; \theta i))^2],$$

where ρ(s, a) denotes the distribution of the training examples, and *y* is the prediction of $Q(s, a)$ given by the Bellman equation

$$y = r + \gamma \max Q^*(s', a'; \theta i-1)|s, a.$$

This loss function can be minimized by the stochastic gradient descend (SGD) algorithm. The gradient with respect to θ is given by:

$$\nabla\theta_i L(\theta i) = E_{(s,a)\sim\rho(\cdot)}[(r + \gamma \max Q^*(s', a'; \theta i-1) a' - Q(s, a; \theta i))\nabla\theta_i Q(s, a; \theta i)]$$

It is known that deep-learnig can learn hierarchical patterns that are most of the time abstract and invariant against unexpected disturb. It is an important advantage to take in account. In fact, in stock markets, we can meet vast raw and noisy informations and challenge us to identify the market correctly. With the help of the deep q-learning, these patterns can be extracted from the vast and noisy information. They help us to see a bigger picture and to successfully represent the state s of the market.

The neural network used in the DQN agent is a basic fully connected network (FCNN) composed with 4 Dense layers of 64, 128, 64, and 3 units respectively. Each Dense layer is followed by a RELU activation function except the last layer which has no activation function.

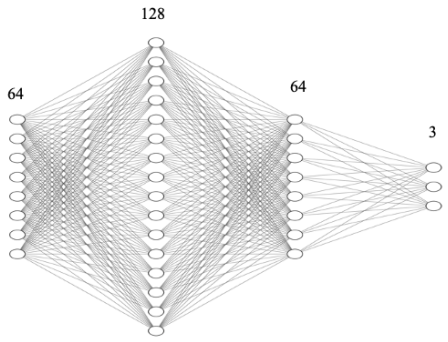

Figure 1: Architecture of the Q-network. The input of the network is a vector of length t, representing the last t prices of the stock. The output of the network is an array of size 3, representing the approximation of the Q-value for the 3 possible actions (HOLD, BUY, SELL).

2. Genetic Algorithm

Genetic algorithms give us the opportunity to solve complex problems by harnessing the power of nature. They are, in fact, brute-force problem-solving methods which use natural evolution to speed up the process. Traders can, using these algorithms to enhance their trading rules, predict security prices, and create new strategies.
They are commonly used as optimizers that adjust parameters to minimize or maximize some feedback measure, which can then be used independently. A genetic algorithm would then input values into our chosen parameters with the goal of maximizing net profit. Over time, small changes are introduced, and those that make a desirable impact are retained for the next generation (using genetic operations as mutation and crossover).

First, we represent each state by a vector $v \in \mathbb{R}^n$. Each state is called an individual and we start with a random set of individuals, called population. At each generation, we select the best individuals of a given generation to produce the next one. The selection is based on some fitness function that rates the individuals. Like in genetics, we have the reproduction between two individuals (states). Crossover is one potential advantage of genetic algorithms and corresponds to the point of merge between two vectors (might be chosen in a random way) and allows to generate a new population. Moreover, mutations can occur and randomly modify a gene (location in the vector) with some small independent probability.

In the trading case, given a trading strategy $T(\theta)$ where $\theta \in \mathbb{R}^n$ is the strategy's parameters. Each individual is represented by his θ vector. Our fitness function is the cumulative return on a time period N.

$$\text{CR} = \frac{c_N - c_0}{c_0},$$

where c_N is the portfolio capital after a time-period N and c_0 is the portfolio capital at the beginning.
We used the genetic algorithm on three basic strategies: MACD, RSI and a mix of both which will refer as MACD+RSI.

Evaluation

To evaluate our solution, we will use multiple performance indicators: Return of the agent at the end of the trading test period and the Sharpe Ratio which is the return of the trading activity compared to its riskiness.

The Sharpe ratio is defined as

$$\frac{r_p - r_f}{\sigma_p},$$

where r_p is the expected portfolio return, r_f is the risk free rate, σ_p is the portfolio standard deviation.
We will also compare our agents with Buy and Hold, MACD and RSI strategies.

1. MACD Strategy

The MACD (moving average convergence divergence) is a trading indicator designed to reveal changes in the strength, momentum and duration of a trend in a stock's price.
It is defined as the difference between a fast exponential moving average (EMA) and a slow EMA of the price series.
The MACD strategy will choose to buy a stock if the MACD line is above the signal line and to sell a stock if the MACD line is under the signal line.

A MACD strategy with parameters s, l, k is given by:

$$MACD_t = EMA_t^s - EMA_t^l,$$

The value of a n period EMA of stock price at time t can be calculated as follow:

$$EMA_t^n = \text{Price(t)} \times \text{k} + EMA_{t-1}^n \times (1 - \text{k}),$$

$$EMA_0^n = \frac{SP_1 + \cdots + SP_n}{n},$$

where k = 2 ÷ (n + 1) and SP_i is the stock price at time i.
Then, the signal line is an k-period EMA of the MACD series.
The basic MACD strategy uses 26, 12, 9 period EMA for the long period, short period and signal period respectively.

2. RSI Strategy

The relative strength index (RSI) is a trading indicator that measures the magnitude of recent price changes to evaluate overbought or oversold conditions in the price of a stock or other asset. The RSI is most typically used on a 14-day timeframe, measured on a scale from 0 to 100, with high and low levels marked at 70 and 30, respectively.
RSI can also be used to identify the general trend.
The RSI is computed with the following formula:

$$RSI = 100 - \left[\frac{100}{1 + \frac{Average\ gain}{Average\ loss}}\right]$$

It will rise as the number and size of positive close increase, and it will fall as the number and size of losses increase. The RSI strategy is then composed of three parameters: the time period n (used to calculates the gain and loss averages), a low threshold and a high threshold.
The strategy will choose to buy a stock if the RSI value is lower than the low threshold and to sell a stock if the RSI value is higher than the high threshold.

3. RSI & MACD Strategy

Finally, the last strategy (RSI&MACD) uses both the MACD and RSI strategy to take an action. If the two basic strategies agree on a position as the position is taken. Otherwise, we hold.

Results

We train and test our agents on two different trading stock: SP&500 and AAPL. For each stock market the data was split between training and test set. The training set starts from 01/01/2015 to 01/01/2018. The test set starts from 01/01/2018 to 01/01/2021. In the case of the Genetic Algorithm, we optimize the basic trading strategy RSI, MACD, and RSI&MACD (with 3, 3 and 6 parameters respectively) on the training set and we test them on unseen data (test set). We will add a + to a strategy when we used Genetic Algorithm to find the best set of parameters like in MACD+, RSI+ and RSI&MACD+. The basic MACD and RSI uses (12, 26, 9) and (14, 30, 70) as parameters respectively. Fig 2 shows the cumulative return and Sharpe ratio of all agents for all stock.

	AAPL		SP&500	
	SHARPE RATIO	CUMULATIVE RETURN	SHARPE RATIO	CUMULATIVE RETURN
BUY&HOLD	1.227	1.974	0.568	0.358
RSI	0.477	0.306	0.424	0.215
RSI+	1.305	2.217	0.785	**0.552**
MACD	1.534	1.424	0.964	0.37
MACD +	2.036	2.119	**1.05**	0.378
RSI&MACD+	**2.122**	**2.513**	0.743	0.509
DQN	1.143	1.08	0.758	0.448

Figure 2: Agents comparison on AAPL and SP&500 stock price on the test set. The higher a fund's Sharpe ratio, the better its returns have been relative to the amount of investment risk taken. The cumulative return was computes on the test set on a period of 3 years.

As we can see from Fig 2., basic trading strategy optimize with genetic algorithm performs better than buy and hold or DQN. Genetic algorithms are often ignored, as another unsupervised learning algorithm that fails to converge effectively. This is partly true, as genetic algorithms do not use partial derivatives and therefore the training algorithm is less direct. However, genetic algorithms allow for the formulation of solutions that would have been impossible, using traditional gradient-based optimization. If the gradient descent based model started at local minimum, it would be permanently stuck, as the gradient of the points on each side would be higher. However, due to the more stochastic nature of genetic algorithms it would eventually find the solution. Additionally, neural networks require labelled data with well-defined loss functions. Genetic algorithms only require data, and a custom loss function (the fitness function) can be cleverly crafted to optimize certain features. This flexibility makes genetic algorithm stand out from the other agents.

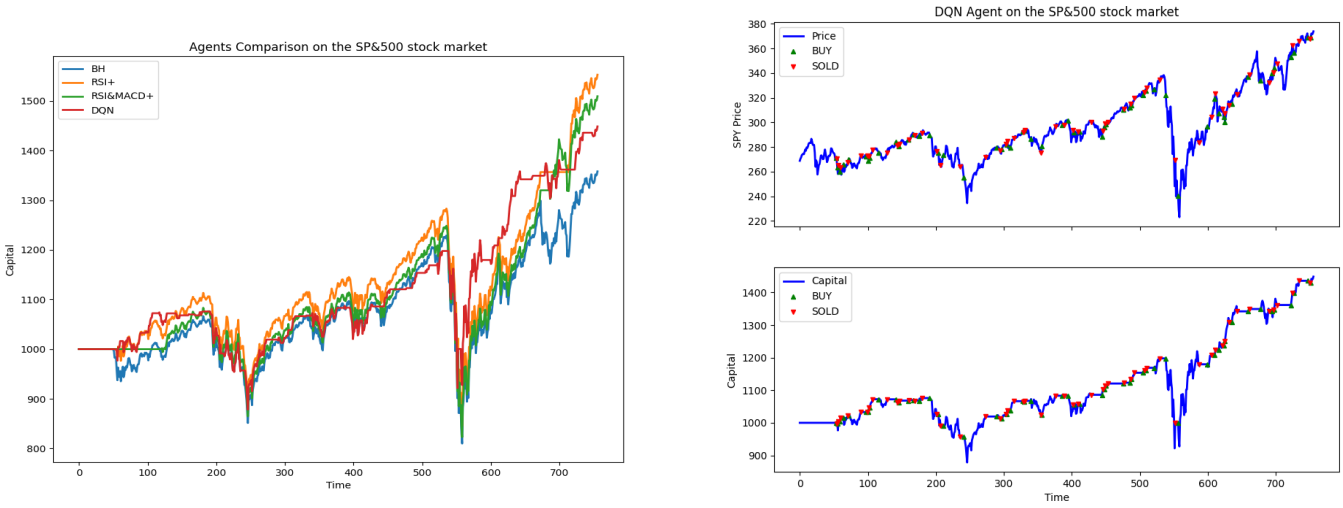

Figure 3: The left graph shows the evolution of the portfolio capital of four different agents on the test set of SP&500 stock market. The starting capital is 1000$ and each time step correspond to one day. On the right, we can see the actions that the DQN agent has taken during the same period.

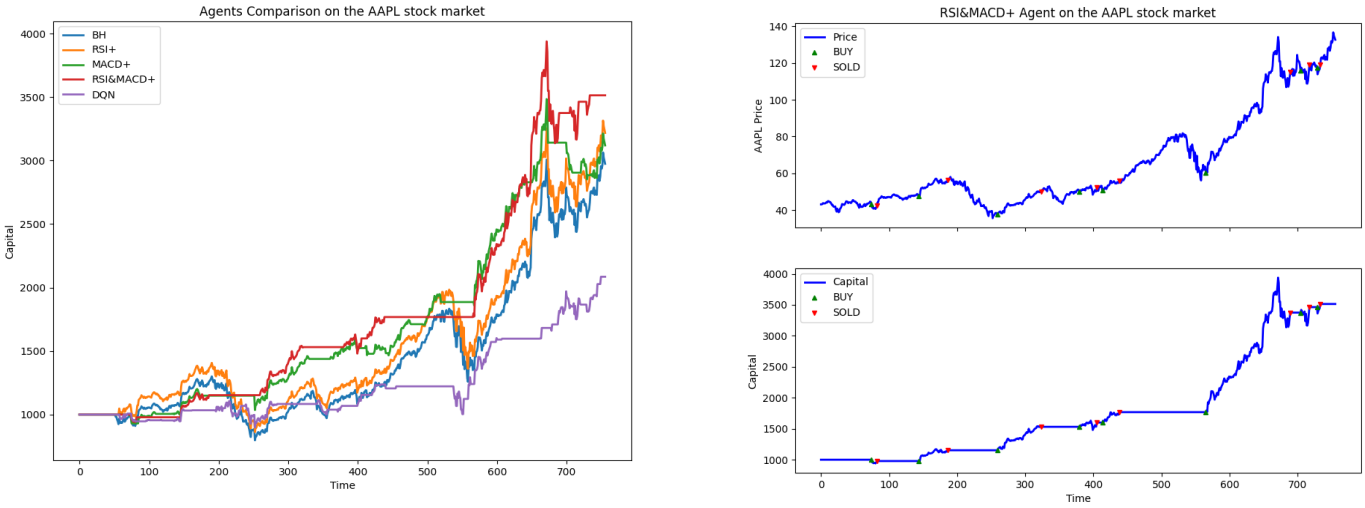

Figure 4: The left graph shows the evolution of the portfolio capital of five different agents on the test set of AAPL stock market. The starting capital is 1000$ and each time step correspond to one day. On the right, we can see the actions that the RSI&MACD+ agent has taken during the same period.

We can see in Fig 3 that the strategies the genetic algorithms optimized (RSI+ and RSI&MACD+) outperform the other strategies, given 1000$ as the starting capital and SP&500 as our stock market. The DQN agent is still better than the Buy & Hold strategy.
On the other hand, with a different stock market like AAPL (see Fig 4), the DQN agent underperform all the other agents (except basic RSI) by an impressive margin. In fact, the apple stock prices are in a bull run. Thus, it is already difficult to overtake the buy and hold strategy. Moreover, we notice that the DQN agent tends to perform a high number of trades. This behavior is unfavorable in the AAPL stock market when it's difficult to sell at a lower price than we buy.

Conclusion

In this project, we explored innovative approaches and compare two different methods, reinforcement learning and Genetic algorithms that are based on biological operators as mutations and crossovers in order to solve the traditional algorithmic trading problem.
We found out that Genetic algorithm is the most stable methods compared to the value-searched-based methods: Q-Learning and Deep Q-Learning. Genetic algorithms as our RSI+, MACD+ and RSI&MACD+ manage to find a policy that outperforms the DQN agent.
The genetic algorithm finds optimized solutions efficiently and has a better learning efficiency. In the current stage, we could see that the genetic algorithm is more likely to have less learning-cost than reinforcement learning. It's simply because in a genetic algorithm, numerous agents learn in simultaneously, whereas reinforcement learning trains only one agent at a time.
The genetic algorithm is simple to apply, however reinforcement learning takes longer to build. Small changes in performance can result in significant differences, and without a good reward system, the agents would perform badly. In fact, adjusting the mutation probability makes it easier to change the genetic algorithm.
The implementation can still be optimized by adjusting the configuration.
For future work, there is a lot of ways to expand our work. First, we could add data augmentation, a technique to increase the amount of data to help reduce overfitting when training our DQN agent..Hence, we suggest there might be some further work to be done through debugging or experimenting with an optimization of our Q-network or with other architecture (convolutional neural network). Another way to expand our works will be to feed our DQN agent with more information about the market. Our current input is only the last t prices, but it is well known that the market depends on factor like news feed, market volume, politics, world events…
Finally, a logical expansion of our project will be to allowed multiple trades. An agent which can perform one trade at a time is limited. Thus, an implementation of an intelligent trades division algorithm and risk management can improve our results significantly.

Appendix A

1. *Code Explanation*

We provide a Jupyter Notebook “Getting Started.ipynb” which explain how to use our code. Enjoy!