Reinforcement learning agent for trading items on the Steam Community Market

Abstract

In general, it is becoming very popular to use machine learning to build trading algorithms because of the increasing amount of data and possible computing power. But the stock market is quite complex in its structure, so it was decided to look for smaller counterparts. Thus it was decided to use the Steam Community Market and to use algorithms to train an agent to trade and achieve profit.

The basic algorithm was a bench of empirical rules, which was optimized with genetic algorithms. The next step was to implement a Deep Q-Learning algorithm to find a fairly simple decision-making policy. The final step was to use the Actor-Critic algorithm to optimize policy at each step instead of the overall trajectory.

As a result, with input data, optimized rule-based algorithms showed that agents could achieve a profit and the best approach was achieved by the algorithm of Deep Q-tables. For the Actor-Critic algorithm, there weren't any significant results with the current data and built environment.

Keywords: neural networks, optimization, portfolio choice, q-learning, actor-critic

Introduction

This article discusses possible approaches to optimize decision-making in an uncertain environment based on the buying and selling of items in the Steam Community Market. The main purpose of this work is to find the optimal algorithm that will automatically make the decision to buy or sell in order to maximize profit.

After analyzing the data and building an empirical base model based on predefined rules, it was decided to use a genetic algorithm to optimize the parameters of the base model. The next step was to use reinforcement machine learning algorithms to find the optimal strategy. As a result, two algorithms were implemented: Deep Q-Learning and the Actor-Critic algorithm.

This work uses data from the Steam Community Market: the price history of items for the day. There are several thousand items in this market; however, only 48 were selected over a 430-day period that met the following criteria:

- The average price over the last three years is greater than 50 units
- At the time of selection, the number of items on the market is between 100 and 1000 units
- Variance of a Price greater than 50

It should also be noted that the market charges a 15% commission for each transaction, which complicates the process of earning a profit when buying and selling.

Only the sales or purchase records where there were more than five items of goods (per day) were selected for the screening of data emissions. The mutual correlation of item prices was also checked, and it was found that in most cases (1013 out of 1104 or 92%) the coefficients lie between -0.7 and 0.7, indicating that there is no explicit linear dependence.

Methodology

Because of the market structure, and based on our own observations, the following basic algorithm for trading profit was constructed:

1. Calculate the maximum price for the last three days of trading on the market.
2. If the current price is less than the maximum multiplied by 0.85 (since 15% of the price is taken by the market commission), then buy the item and consider it the new maximum price – the purchase price. Save the purchase price.
3. If the price for which the item was purchased is lower than the current price multiplied by 0.85, then put the item up for sale.
4. If the purchased item has not been sold within seven days, reduce the price by 5% and deduct three days from the number of days without sale.

If this algorithm is applied to the available data, giving the agent a starting capital of 7,000 monetary units, by the end of time, 15,339 monetary units and 11 difference items will remain in the portfolio, with a current market value of 4,207 monetary units. This result is a very good indicator, as it shows a profit of more than 179% in 16 months.

A genetic algorithm was used to select the optimal parameters of the basic algorithm, that is, to optimize the objective function [1] - profit. At each step, a population of 50 agents was randomly generated, with a random set of parameters (genes). After that, the agents interacted with the environment for 50 time periods (a minimum duration of three days) and received a (profit) reward [1] through the fitness function F: Agents → R, which works from a plurality of agents to a plurality of real numbers.

Table 1. Optimized parameters and their distribution

Parameter	Basic mode	Distribution for generation
Earnings per transactions (monetary units)	0	Module of normal distribution N(0,100).
Earnings per transaction (percentage)	0	Uniform distribution of R ([0,1])
Waiting time to decrease in price (days)	7	Discrete uniform distribution R([2,16])
Price reduction for inactivity (percentage)	5	Uniform distribution of R ([0.6,0.99])
Time between price dropping (days)	3	Discrete uniform distribution R([0,days without sales])
Calculation interval for maximum (days)	3	Discrete uniform distribution of R([2,15])

Source: *specified by Authors*

Based on the best representatives, namely the 11 highest income agents, the parameters (mean and variance) of the normal distribution that will be used to generate the next generation of agents are calculated [1]. Also, a mutation is added to each of the representatives of the new generation, namely the normally distributed random variable N (0,1).

After nine iterations, the final price of the portfolio increased from 6,000 units in the first population to 27,000 in the third iteration, and then began to fluctuate at around 25,000 (Figure 1):

In turn, population parameters varied as follows (Figure 2):

Having obtained such results and using a set of parameters from the last iteration to rerun the basic algorithm on all data, the total portfolio value increased by only 28%, which is a decrease compared to the basic algorithm (179%).

If we use the set of parameters from the second iteration (the best result from the genetic algorithm), then the result is a total portfolio value of 49,630 monetary units (Fig. 3), but it only 454 monetary units, and all the

rest are items that are not a good result, because the agent simply recognized the trend and bought the items in the early stages. The agent did not sell anything because they anticipated high profits in the future.

Combining the results of the parameters of the best and last iterations, as well as the parameters of the base model, the result is a total portfolio value of 20,271 monetary units, where 18,249 are directly monetary units, and 2,022 is the value of the items purchased.

Thus, the genetic algorithm improved the baseline result by 3.7%. This improvement was due to an increase in non-sales waiting days of up to nine days (instead of seven days), an increase in the interval between price reductions of up to five days (instead of three days), a 9.6% drop in price in the absence of sales (instead of a 5% drop), and an increase in the time interval of up to five days (instead of three days).

The next step is to apply reinforcement machine learning algorithms. Such algorithms allow the learning of actions (policies) that maximize the reward function based on the interaction of the agent with the environment. Applying Deep Q-Learning allows us to find a fairly simple decision-making policy in general without being tied to a specific action at a specific point in time "*t*".

To simplify the modeling task, we decided that we would select only one of the items. Accordingly, when comparing results with the basic optimized algorithm we also use only one item.

To validate the algorithm of Deep Q-Learning we use the classical method of splitting data into training and test. Doing so allows us to analyze how the model behaves on data that has not been seen before. The data is split in a ratio of 3:1.

Next, we specify the environment in which the training takes place. For the agent, we define the set of actions discreetly at each time t: 0 - wait, 1 - buy, 2 - sell the whole stock. According to these actions, we determine the reward that the agent receives for them: if the agent chooses to sell, but has an empty portfolio, then the reward received is -1; if the agent chooses to buy (2), then the reward 0 (undefined whether positive or negative action). And if the agent expects, then the reward is calculated by the sign (P) earnings indicator, where P is the possible earnings from the sale of all items.

Thus, at each step, the environment takes action and returns a Boolean variable if its end of data and a history vector, where the first element is equal to the expected profit at time t, and every subsequent element from the end takes the value of the price difference between successive time points ($p_t - p_{t-1}$). The length of the vector was set to 90 steps in the training process. This approach allows us to relay the interaction history from the environment to the decision model.

Next, it is necessary to define a decision model, namely a neural network [4], which acts as a Q-table [2] and returns the expected reward when executing the policy π during the state of the environment s and the received reward a:

$$Q^{\pi}(s,a) = E[R_t] \text{ (Equation 1)}$$

R_t – is the discounted number of future awards for each r_i action

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \text{ (Equation 2)}$$

The model consists of three fully connected linear levels, between which the sigmoid activation function is used, with shapes of first level (91, 100), a second level (100, 100), and a third level (100, 3). As a result, the model returns the probability vector for three actions, and in this approach, the action with the highest probability is selected.

The following is a training process over 80 epochs, with a learning rate of 0.001 and an Adam optimizer [5] with a mean squared error as a function of loss. The gamma parameter is 0.97 (Bellman equation [6]), the epsilon parameter (for additional randomness in training) is 1, with a decrease of 1e-3 after the first 200 iterations until it reaches 0.1.

After completing the training and running the model on the test dataset, the value of the portfolio was 15,273 monetary units, which is a better result when compared to the base model result of 12,532 (an increase of 21%) and the basic optimized model result of 14,588 (an increase of 4%).

It should also be mentioned that this algorithm does not depend on starting capital, and during the training, the algorithm never went lower than zero.

Following the good results of the Deep Q-Learning algorithm, the next step is to apply a more complex algorithm to optimize policy directly at each step rather than the overall trajectory. The Actor-Critic algorithm is well suited for this task.

In the case of the Actor-Critic algorithm [9], we use the same data and the same discrete actions, but define the reward function with the following features [8]:

- If the agent reaches the end of the data with a positive balance, the agent is rewarded with the amount of money earned (g).
- At each step t, the reward is defined as the total number of monetary units earned on the previous time step t-1 (g).
- In order to encourage the agent to trade for a long time rather than spend all of the capital at once, an elapsed time parameter (ts) starts from -300 and increases by one with each time step.
- An inactivity (ip) parameter of five (minus for inactivity, plus for any action) penalizes the agent for inactivity.

As a result, for each action, the rewards function takes the following form:

$$R(a,t) = \pm ip + ts + g \quad (\textit{Equation 3})$$

Next we define the trading environment, which is characterized by a vector of five parameters [8]: the number of items purchased, the number of monetary units, the current price of the item in the market, the current total value of the portfolio, and the average value of the market in the last five days.

The next step is to define the models for actor and critic. For these models, the initial levels are common: a linear fully connected layer (5, 128), a sigmoid activation function, a linear fully connected layer (128, 128), a hyperbolic tangent activation function, two recurrent layers [7] (128, 32), a linear fully connected layer (32, 31), and an activation function: Relu. The model is then divided into two parts:

- Actor – linear fully connected (31,3) – what the action to be taken
- Critic – linear fully connected (31,1) – evaluation of action

Next, there is a training process over 5,000 epochs with a learning rate equal to 0.0003 and an Adam optimizer [5] with an early stopping callback if the model completes the run through the data with a profit of 20% greater than the starting capital and reaches an end point of the data, and a sliding reward greater than 500. Used following parameter gamma = 0, 9 (Bellman equation), the parameter epsilon at each step is random and is determined by the formula:

$$\frac{\varphi}{10^4} - 5 * 10^{-5}, \textit{ where } \varphi \sim R([0, 1]) \quad \textit{(Equation 4)}$$

Research results and discussion

As a result, the algorithm learns that the best solution is inaction, so such a model has no value. Therefore, we need to change the training settings.

To improve the model, the following steps were performed:

1. Removed the linear trend from data.
2. Added sliding statistics: minimum/maximum price for n days and average price of items purchased.
3. Changed the architecture of the network: recurrent levels were added, the network depth was increased with adding linear levels and increasing their parameters.
4. Changed the gamma parameter to increase or decrease the weight of memory.
5. Most of the work was done with the rewards function:
    - Used the reward function as in the previous Deep Q-Learning approach (the algorithm bought everything in the first 50-70 iterations and went bankrupt)
    - Deleted the time elapsed (ts) parameter (the algorithm refused to trade)
    - Changed the reward for purchase to zero (the algorithm was learning, but the result was random and when iterations increased, it was over-fitted and took no action other than waiting for the test data)
    - Defined the reward function as the possible profit – the ratio of the actual price to the average price of the items in the portfolio (the algorithm did not converge)

The associated difficulties with the Actor-Critic algorithm most likely arose from the fact that when calculating the back propagation error, the approximate value of the gradient of expected reward E [Rt] has a very large variance [3], resulting a significant noise and, as a result, the sequence converges to an option with the smallest variant, namely inactivity. This problem can be solved by artificially reducing the variance by subtracting from the reward the value of the basic algorithm at each step [10]. It is also recommended to use the TD-Lambda or GAE training algorithm [10].

Conclusions, proposals, and recommendations

A certain set of algorithms showed their potential could possibly be used in optimizing decision-making in an uncertain market. A rule-based genetic algorithm works, but it only involves optimization of numerical parameters for empirical strategy, which does not guarantee the optimal solution and the highest profit. Deep Q-Learning works best for current data and could be improved with a combination of other approaches (more RNN layers, genetic optimization of model hyperparameters) or adding more parameters to the model, such as response time.

In the future, it is planned that the Deep Q-Learning algorithm will be improved and combined with the results of the genetic algorithm to test them on other datasets. The option of bringing these algorithms to real trading in the market is also being considered, but decisions made based on machine learning are difficult to interpret, so they should be treated with extra caution. Also, during the training of reinforcement algorithms (Q-Learning and Actor-Critic), there was no reaction from the environment because all of the prices are predefined and historical, and the built environment didn't change them. This meant that the actions of the agent couldn't affect the environment, so there could be a different result in a real-world scenario. Otherwise, this historical data could be interpreted as one of the possible scenarios and, with more investigation of the Steam Community Market, the probability of this scenario could be predicted, and a possible income could be calculated.

References:

1. Subbotin S. O., Olijnyk A. O., Olijnyk O. O. Neiteratyvni, evoljucijni ta muljtyaghentni metody syntezu nechitkologhichnykh i nejromerezhnykh modelej: Monografija / Pid zagh. red. S. O. Subbotina. — Zaporizhzhja: ZNTU, 2009. — 375 s.
2. Riedmiller. (2005). Neural fitted Q-iteration: batch-mode Q-learning with neural networks
3. Sutton, McAllester, Singh, Mansour (1999). Policy gradient methods for reinforcement learning with function approximation: actor-critic algorithms with value function approximation
4. Watkins, C.J. & Dayan, P. Machine Learning (1992) 8: 279. https://doi.org/10.1023/A:1022676722315
5. Kingma, D. P. and Ba, J., “Adam: A Method for Stochastic Optimization”, arXiv e-prints, 2014.
6. Bellman, R. (1957a). A Markov Decision Process. Journal of Mathematical Mechanics, 6:679– 684.
7. Li, Hailin & Dagli, C.H. & Enke, David. (2007). Short-term Stock Market Timing Prediction under Reinforcement Learning Schemes. 233 - 240. 10.1109/ADPRL.2007.368193.
8. Kang, Qinma & Zhou, Huizhuo & Kang, Yunfan. (2018). An Asynchronous Advantage Actor-Critic Reinforcement Learning Method for Stock Selection and Portfolio Management. 141-145. 10.1145/3291801.3291831.
9. Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu Asynchronous Methods for Deep Reinforcement Learning: 2016
10. Deep Q Network vs Policy Gradients - An Experiment on VizDoom with Keras - https://flyyufelix.github.io/2017/10/12/dqn-vs-pg.html

Figure 1. Changing the value of the portfolio during iterations of the genetic algorithm

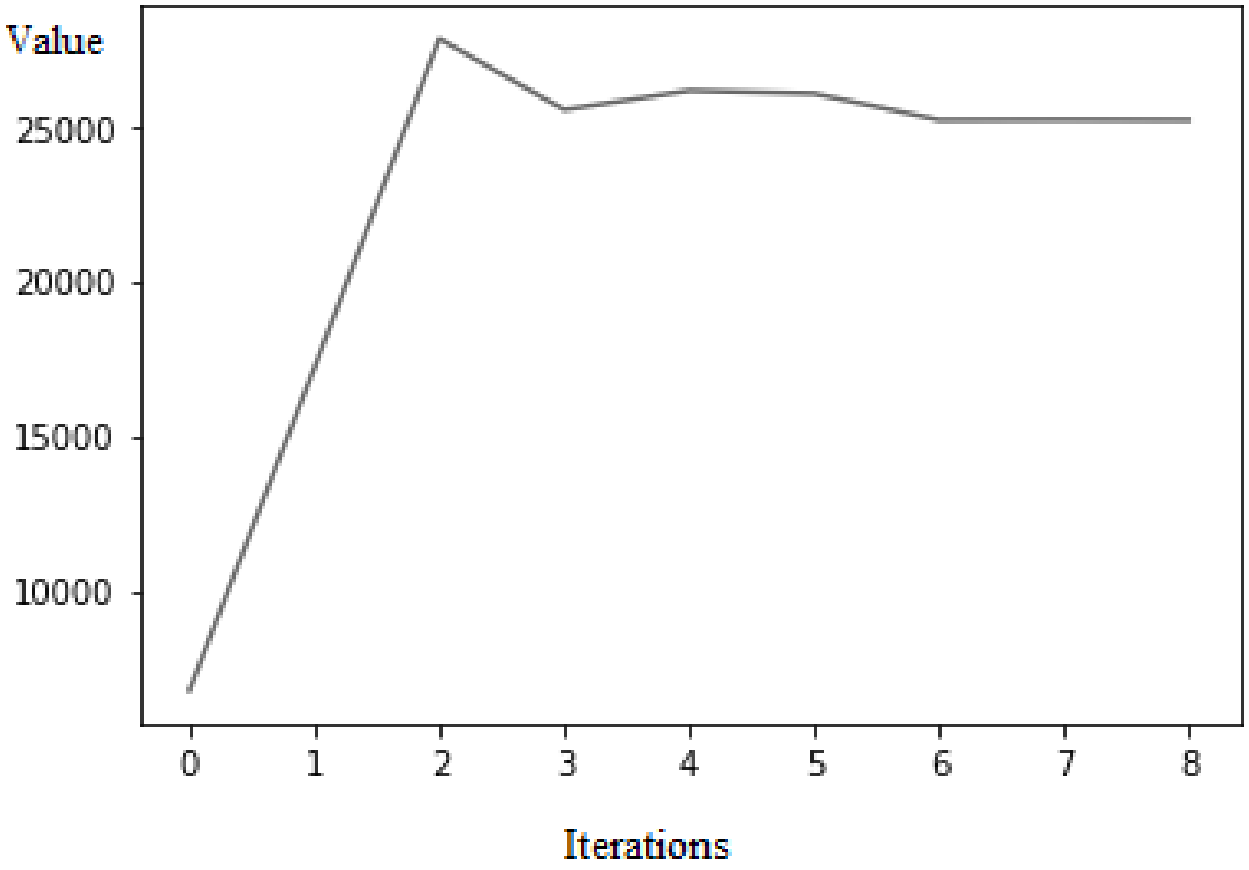

Figure 2: Change of population parameters in iterations of the genetic algorithm

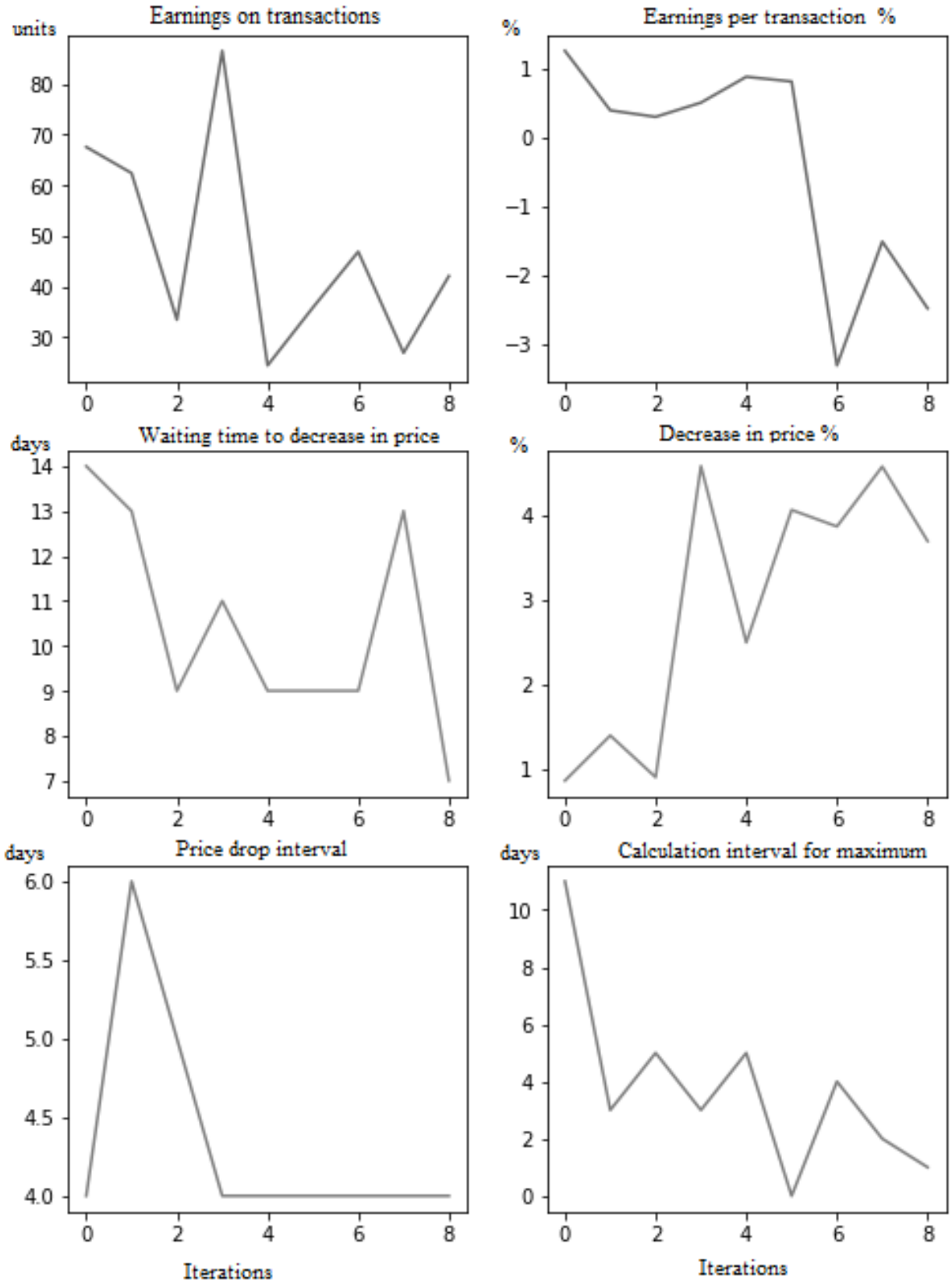

Figure 3. Results of the change in the value of the portfolio with the parameters of the best generation