QTrader - A Deep Deterministic Policy Gradient (DDPG) Stock Trading Agent

Zander Zemliak
Department of Computer Science, Loyola Marymount University
CMSI 5350 - Machine Learning
December 12, 2022

Abstract

Trading stock successfully may yield fantastic results if one is doing the right thing at the right time. Some may even do the right thing at "the right time" and yield awful losses. Was there something the human eye missed in the vast daily data of the vast financial markets? Impossible! Instead of trading with our brains, let's train a deep reinforcement learning agent to make trades for us. Perhaps the agent can converge towards some optimal trading strategy. We can train the agent, backtest it in a particular time period, and compare the backtest performance to a financial indicator's change in the same time period. Inspiration for this project came from the paper *Practical Deep Reinforcement Learning Approach for Stock Trading*[1] by Xiao-Yang Liu, Zhuoran Xiong, Shan Zhong, Hongyang (Bruce) Yang, and Anwar Walid of Columbia University.

Introduction

The stock market can be a place to build wealth by investing intelligently. Warren Buffet suggests purchasing securities such as index funds[2], instead of investing in one stock, invest in all of them! This is a great strategy and has proven to be successful with relatively low risk. It has made many people very wealthy in the past century. There is a catch though, *time*. For this strategy to yield any kind of successful result, you need to be disciplined, consistent with your monthly contributions into your account, and you need to do it for a very long time.

The "buy and hold" strategy is not the only way you can make money in the stock market. An individual can make large quantities of money by taking on a significantly larger amount of risk and using tools such as derivatives to make what a long-term investor would make in 30 years in *a single day*. The only catch to this is you need to foresee something within a particular

[1] Liu, Xiao-Yang, et al. "Practical deep reinforcement learning approach for stock trading." *arXiv preprint arXiv:1811.07522* (2018).
[2] An index fund tracks the performance of a specific market benchmark—or "index," like the popular S&P 500 Index—as closely as possible. The portfolio is weighted with respect to the market caps of certain stocks.

security/market before everyone else does. In other words, you need to predict the future and you need to be right. If your due diligence is wrong, you can lose a lot of money.

There is so much quantitative data being processed every millisecond during trading hours. Perhaps it would be best to develop and train a model that can consume all of this information and determine the best trade to make. There are many firms on Wall Street that are successful at this; TwoSigma, Renaissance Capital, Jane Street, and Citadel just to name a few.

Hypothesis

An approach we can take to this is to break down the problem into a Markov Decision Process (MDP). The environment is the financial market, our agent consumes some state-space from the markets and a reward, and the agent will execute some action based on this information.

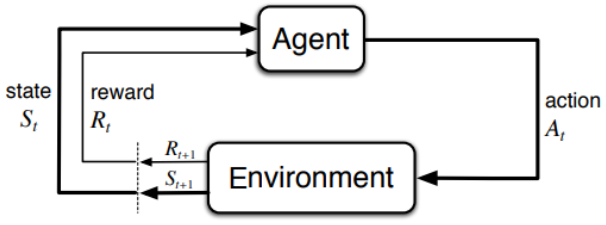
If we have a reinforcement learning agent whose objective is to maximize return, with deep learning, we can have our agent converge towards some general trading policy that will yield successful returns in any kind of market condition. Our agent would not predict the future but instead learn hidden parameters within the state-space that humans cannot comprehend to best guide its action.

Because the state-space of the financial markets is continuous, it is impossible to define a set of possible states. To solve this issue, we will use the Deep Deterministic Policy Gradient (DDPG) algorithm.

Reinforcement Learning Background

Any reinforcement learning agent learns to be successful in their given environment by taking actions which yield rewards. For example, a dog during training learns that when it sits upon command, it receives a treat. Thus forward, when a trained dog is told to sit, it will do so because it has learned that when it does this, a reward is received.

As stated prior, any reinforcement learning problem can be broken down using the Markov Decision Process framework.



An agent takes some action with their environment, the agent then receives two pieces of new information; 1. the new state of the environment and 2. the reward from their previous action. This information guides the agent towards some policy that it can abide by to maximize rewards.

Remember, *agents take actions depending on the current state they perceive*. For example, if I am playing chess and I am one move away from checkmating my opponent, I will take the action that checkmates my opponent. I will not take any other action because maximizing rewards (winning the chess game) is in my best interest.
From a quantitative level, if we are given a state and a set of possible actions, we can define that **value** of a state by using the Bellman equation:

$$V(s) = max_a(R(s,a) + \gamma(V(s'))) \backslash$$

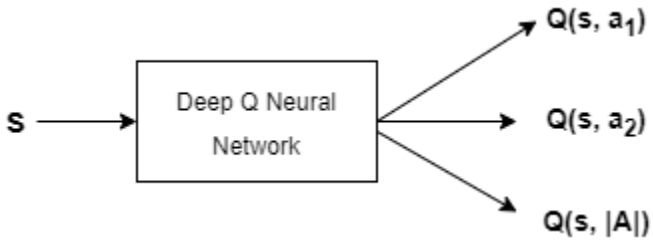
Where:
V(s) = The value of a state 's'
R(s,a) = The reward the agent will receive if it takes action 'a' inside of current state 's'
γ = The discount factor, the amount our agent cares about rewards going forward
V(s') = The value of the next state (s')

The value for our current state, **V(s)**, is equal to the action which maximizes current reward in state ***s*** with action **a** plus the value of the next state **s'**. The core idea of RL is that an agent is guided by some learned policy π. Such that, given some state, the policy will tell the agent what action to take:

$$\pi(s) = a$$

Q-Learning is a simple value-based RL algorithm that attempts to map a finite set of states and actions to some numeric quality (Q) value. This algorithm works great for environments with finite state-action spaces but not for the latter.

Deep RL Background
If we have a large state space, we do not want to store it in memory because this could be computationally expensive. Instead, we will use a neural network to take care of what the value based Q-Learning algorithm does:



Given some state, our network will output |A| possible Q-values and then choose the action which has been assigned with the highest Q-value.

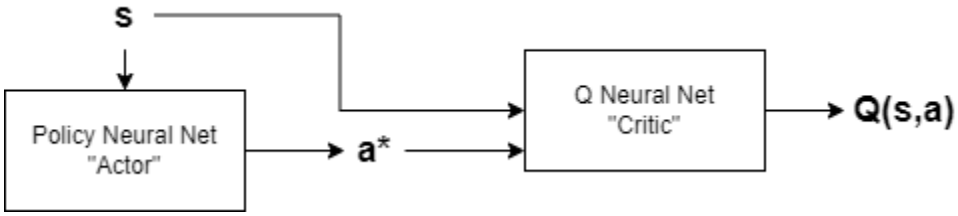
DQN training works by taking an action via epsilon-greedy[3], observing the episode of that action, storing it in memory, and then sampling from the replay memory to update the main prediction NN.
We use two networks for training; a main and target network:

- Main NN: Where our Q-values are derived.
- Target NN: Same parameters as the main NN however, this network is not trained. It is synchronized with the main NN after N steps. This allows us to train the main NN with a single target thus making training stable.

Weights in the neural network are updated in the main network via the Q-Learning Loss function:

$$L(\Theta) = ((r + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta^{target})) - Q(s, a; \theta^{main}))^2$$

Technical Details of DDPG
DDPG is a deep RL algorithm designed to learn in environments with continuous state-space values. We do not need a defined set of state-action pairs, we can teach a neural network to derive the quality of states and what action to take in certain states.



Our agent uses two neural networks for taking actions: an "actor" and a "critic".

Actor NN: Responsible for outputting the best action to take given a state; takes in a state **S** as an input and outputs the optimal action **a***.

Critic NN: Responsible for assigning a quality value to a (state, action) pair. E.g. given a state and an action, return some quantity of how good that action is in that particular state. Takes in a **(state, action)** tuple and returns a Q value; **Q(state, action)**.

During learning, just like with regular deep RL, the actor and critic neural networks both have their own respective target networks for training. The actor network trains to maximize the Q value that is being outputted from the critic network by using gradient ascent and the critic network trains to minimize loss so it uses gradient descent.

[3] Epsilon-greedy is an algorithm where given some probability, if less than P, do a random action. Otherwise conduct the action with the highest associated Q-value.

DDPG Learning Algorithm Pseudocode[4]:

1. Initialize actor θ params (π net), critic ϕ params (Q net), and empty replay buffer B
2. Set target params equal to their respective main networks (just like DQ Learning)
3. Repeat until convergence:
    a. Observe state s and get a* from critic and execute a*
    b. Observe state s', get reward r, and set d = is in terminal state (0=no, 1=yes)
    c. Store episode (s, a, r, s', d) in replay buffer
    d. If d = 1 reset environment parameters otherwise continue
    e. If time to update:
        i. For however many updates:
            1. Get sample batch from B
            2. Compute target via Bellman: $\mathbf{y(r, s', d) = r + \gamma(1-d)Q\phi_{target}(s', \pi\theta_{target})}$
            3. Update critic net via one step of GD using **y(r, s', d)**
            4. Update actor net via one step of GA using: $\mathbf{Q\phi(s, \pi\theta(s))}$
            5. Update target networks with their respective main network

Problem Overview

To summarize, we have covered that for our RL agent to converge towards some policy, we need to define its state, the possible actions the agent can take in a state, and the rewards from taking action 'a' in some state 's'.

We want our agent to make one trade per day and perceive the one day statistics of the stock as our state-space. The rewards for our agent are defined as the return of trades made.

For our stock trading problem, we can break down our problem into the following:
At some time point **t**:

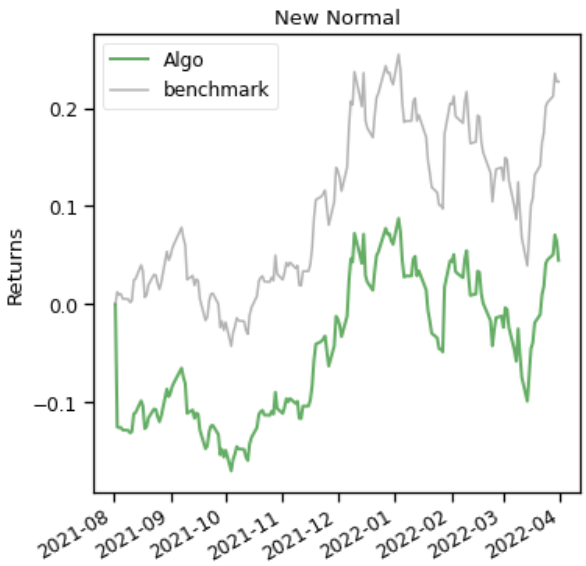
1. State: the current price of our stock and its respective technical indicators
2. Action: Buy, hold, or sell the stock at time **t**
3. Reward: The return of the trade made

The stock we will have our agent trade is Apple (AAPL). For training, we will use about 12 years of AAPL's historical data. For testing, we will use about 8 months of recent AAPL historical data. We will compare the agent's performance to the intraday trading data of Apple within the same time frame that our testing occurs.

[4] For more information on DDPG learning algorithm, read OpenAI's documentation on it: https://spinningup.openai.com/en/latest/algorithms/ddpg.html

Results and Analysis

The testing of our agent was back tested on market data from August 2021 to April 2022:



Return Results:

AAPL (baseline): +22.7%

DDPG Agent: +4.47%

If you had just bought and held shares of Apple stock within this time frame, you would have significantly outperformed the DDPG trading agent. As you can see from the figure, the agent appears to overfit and thus adopts a FOMO trading strategy. It buys the stock when the price is trending upwards and sells it when the price is trending down. It doesn't appear to find patterns within the technical indicators during training and it seems to just be noise to the agent.

What if the agent did *really* well? Would that mean our agent could perform well in any market condition? How would it do during a market crash, a bear market, or a bull market? The type of training done to this model is extremely naive and should never be used with an actual monetary trading account. There are many variables to consider when building a stock trading model, this was just an experiment to see how the model could adopt a potential trading strategy.

Future Improvements

The ultimate goal with a project of this scale is to make the agent as general as possible which means not overfitting to the training data. A really good model whose sole purpose is to maximize return would show a chart trending steadily upwards and have absolutely no correlation with the historical price of the stock it's trading.

Some ways of achieving more generality (potentially):

1. Reducing the batch sample size during training.
2. Playing with the learning rate α.
3. Playing with training with different timesteps for training.
4. Implementing protocols for when the agent reaches some threshold for its cumulative return. E.g. halting trading once +/- 5% is achieved.
5. Figure out which features in the state-space are noise and which ones are significant. More research into technical indicators and how they forecast the potential future movement of a stock.

Acknowledgements

1. Blackburn. "Reinforcement Learning : Markov-Decision Process (Part 1)." *Towards Data Science*, 27 Jul. 2019, towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da.

2. Choudhary, Ankit. "A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python." *Analytics Vidhya*, 18 Apr. 2019, www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.

3. "Deep Deterministic Policy Gradient." *OpenAI*, spinningup.openai.com/en/latest/algorithms/ddpg.html.

4. Dittrich, MA., Fohlmeister, S. A deep q-learning-based optimization of the inventory control in a linear process chain. *Prod. Eng. Res. Devel.* 15, 35–43 (2021). https://doi.org/10.1007/s11740-020-01000-8

5. FinRL Library for learning and backtesting environment: https://github.com/AI4Finance-Foundation/FinRL

6. Karunakaran, Dhanoop. "Deep Q Network(DQN)- Applying Neural Network as a functional approximation in Q-learning." *Towards Data Science*, 18 Apr. 2019, medium.com/intro-to-artificial-intelligence/deep-q-network-dqn-applying-neural-network-as-a-functional-approximation-in-q-learning-6ffe3b0a9062.

7. Liu, Xiao-Yang, et al. "Practical deep reinforcement learning approach for stock trading." *arXiv preprint arXiv:1811.07522* (2018).

8. Shyalika, Chathurangi. "A Beginners Guide to Q-Learning." *Towards Data Science*, 14 Nov. 2019, towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c.